

# Veille technologique Analyse de texte et traitement du langage naturel

par : Hichem, Ewen et Maïna



# 1. Introduction au text mining

Data Mining: L'exploration de données fait référence au processus d'analyse d'un grand ensemble de données afin d'identifier les modèles significatifs

Text Mining: l'exploration de texte consiste à analyser les données textuelles qui sont dans un format non structuré et à les mettre en correspondance dans un format structuré pour en tirer des informations significatives.



3 niveaux dans le NLP :

- Corpus : l'ensemble des documents
- documents : les différents texte
- tokens

Tokenization : séparé une chaîne de texte, trouver les phrases dans les paragraphes ou les mots dans les phrases.

Méthodes pour tokenizer : split, regexn, nltk, spaCy



# collecter les données du web avec Python



## 2) matrice documents termes

Représentation en sacs de mots (bag of words):

Une représentation bag-of-words classique sera donc celle dans laquelle on représente chaque document par un vecteur de la taille du vocabulaire

$|V|$

$|V|$  et on utilisera la matrice composée de l'ensemble de ces  $N$  documents qui forment le corpus comme entrée de nos algorithmes.




# Preprocessing

Etat du pre-processing des token : désaccentuation, passer en minuscules, supp les stopwords (petits mots le, la, les, etc ) simplifier avec Stemmer ou Lemmastiser => nltk, spaCy, PyStymmer



## Recherche mots clés : python

- 1) Marisa trie cf capture
- 2) tâche NLP au niveau de doc : bag of world : 1 si mots cherché présent, 0 sinon, cf + et - sur capture, mise en place d'un poids
- 3) NER et POS par token : structure,
- 4) word embeddings , librairie Gensim, mot de concept similaire (voiture et camion, king et queen,pays et capitale etc...)



### 3) La catégorisation de textes (document classification)

Matrice de confusion :

Elle compare les données réelles pour une variable cible à celles prédites par un modèle. Les prédictions justes et fausses sont révélées et réparties par classe, ce qui permet de les comparer avec des valeurs définies.

accuracy score:

Il indique le pourcentage de bonnes prédictions.

$$\text{Accuracy} = \frac{\text{Vrai positif} + \text{Vrai négatif}}{\text{Total}}$$





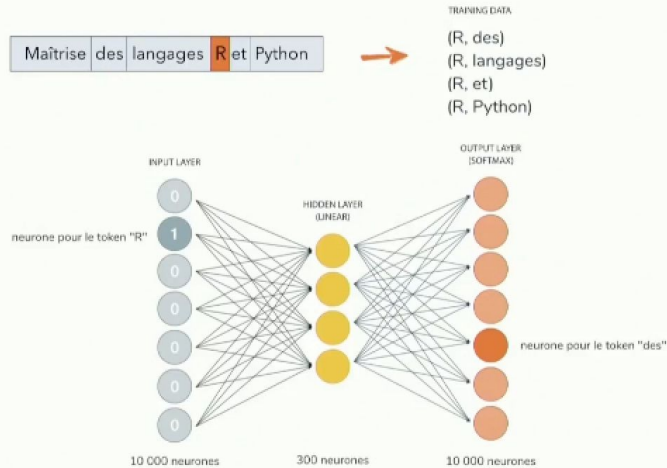
## 4) Word embedding (Word2Vec)

Le word embedding est capable en réduisant la dimension de capturer le contexte, la similarité sémantique et syntaxique (genre, synonymes, ...) d'un mot.

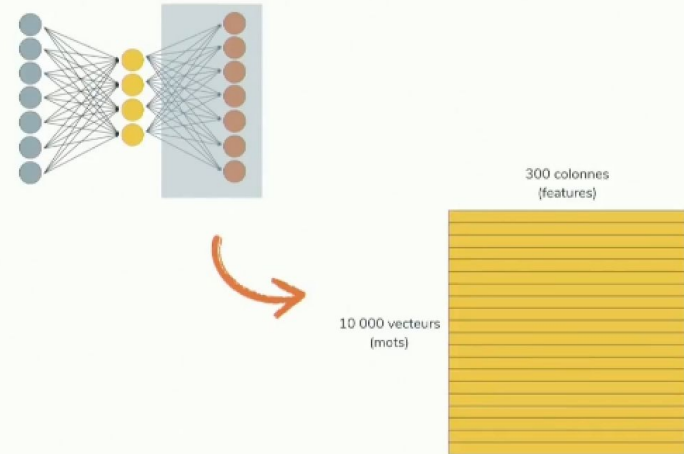
Il existe deux variantes du Word2vec, les deux utilisent un réseau de neurones à 3 couches (1 couche d'entrée, 1 couche cachée, 1 couche de sortie) : Continuous Bag Of Words (CBOW) et Skip-gram.

## 4) Word embedding (Word2Vec)

### Word2vec : le principe de la "fake task"



### Récupération des vecteurs



## 4) Word embedding (Word2Vec)

### Word embeddings avec Gensim



```
import gensim

stopwords = ["le", "la", "les", "l", "''", "de", "du", "des", "et", "en", "ou", "qui", "un", "une"]

sentences = [
    ["maitrise", "des", "langages", "r", "et", "python"],
    ["connaissance", "avancee", "des", "langages", "python", "ou", "r"],
    ["recherche", "un", "developpeur", "python"],
    ["recherche", "developpeur", "java", "qui", "maitrise", "python"],
    ["comptabilite", "bilan", "bancaire", "declarations", "fiscales"],
    ["comptabilite", "analytique", "", "bilan", "et", "compte", "de", "resultat"],
    ["vos", "principales", "missions", "comptabilite", "generale", "(", "compte", "de", "resultat",
    "bilan", "fiscalite"]]

sentences = [[w for w in sent if w not in stopwords] for sent in sentences]

model = gensim.models.Word2Vec(sentences, window=5, size=300, min_count=1, iter=10)

>>> model.wv.most_similar("java")[:3]
[('python', 0.0810871571302414),
 ('r', 0.041719600558200945),
 ('recherche', 0.036696240305900574)]

>>> model.wv.most_similar("comptabilite")[:3]
[('avancee', 0.15022701025009155),
 ('bancaire', 0.1351984143257141),
 ('declarations', 0.08660304546356201)]
```

## 4) Word embedding (Word2Vec)

### Clustering et moteur de suggestion

