

# Big Data : Text Mining (Version CPE)

Alexandre Saidi

LIRIS-ECL  
Mars 2021

ECL - Dép. MI  
LIRIS UMR 5205 CNRS

Big Data - Mars 2021

# Version CPE

- Cette version est prévue pour CPE :
  - Un cours de 2h
  - Pas de TP

# Objectifs TM : Text Mining

**Introduction** : contexte, motivation, définitions, applications

**Fondements** : Bagages théoriques dans

*"Informatique Linguistique"* ( *"Computational Linguistics"* )

**Technologie** :

Transformations suivie des méthodes de Machine Learning

**Applications** (exemples) :

- Résumé automatique,
- Extraction d'information (de corpus textuel)
- Systèmes Q/A (question-answering), ChatBots, ...
- *Opinion mining, Sentiment Analysis* (reviews, opinions, votes)
- *Systèmes de Recommandation*
- WebMining
- ...

# Contexte général

## Constatation : par les techniques de Machine Learning

*Google / Apple / Facebook / Amazon / Ebay / ...*

utilisent des algorithmes pour :

- épier nos habitudes, usages,
- achats,
- questions posées et les mots choisis
- nos réactions aux réponses données
- etc.

## Pourquoi ?

- Par amour
- Par philanthropie
- Pour nous espionner (pas faux mais pas de complotisme please!)
- Par intérêt (commerciaux souvent), etc..

# Contexte général (suite)

## Le règne du Data quelques exemples :

- **Google** détient 67% du marché de la recherche d'info. sur internet
  - 18% pour microsoft *Bing*, *Yahoo* 11%,
  - le reste pour *Ask*, *AOL*, etc.
- **Facebook** :
  - Pré-sélectionne (pour nous) les articles choisis par son algorithme)  
(sauf si vos préférences demandent de tout afficher dans l'ordre chronologique).
- **NSA** Data Collection, Interpretation, and Encryption
- **CRUSH** d'IBM (*Criminal Reduction Utilizing Statistical History*)
  - aurait permis à la police de *Memphis* de réduire les crimes de 30% depuis 2006 (chiffres publiés).
- **IoT / IeE** : déjà là ? la 5G aussi (en 2021) !
- Et le "texte" dans tout cela ?

# Contexte général (suite)

**Le saviez-vous ?** qu'en 1 sec , il y (eu) :

- 200000 SMS sur *Whatsapp*
- 400 heures de séquence *Skype*
- 40000 requêtes *Google*
- 6000 "likes" sur *Facebook*
- Et les biologistes ont crée et stocké leur données sur des supports de stockage à la même vitesse (plus de 29000 Go/s) que celle de la fabrication de ces mêmes supports !

☞ **Origines** : d'où viennent ces données ?

- **de nous mêmes** (mails, SMS, MMS, Facebook, requêtes Google, nos achats , nos déplacements et GPS, etc...),
- la "communauté" dans laquelle on évolue, ...

☞ On estime n'exploiter que 5 – 10% de toute donnée disponible

# Contexte général (suite)

- **Progrès techniques** favorisées par :
  - moyens et capacités de stockage  
facteur  $10^6$  : du Mo au To,
  - vitesse de transfert et de communication  
facteur de 1000 : de paire cuivrée à la fibre optique,
  - vitesse du traitement  
facteur de 5000 : du *Motorola 6800* (1MHz)  
à *Intel I9 multi-cores* ( $\sim 5\text{GHz}$ )

## Volume de données Textuelles :

- On estime à 80% la part du texte dans les BDs.  
→ Mais sous exploité !
- Pléthore de ressources "écrites" :  
livres, journaux, articles, chats, les réseaux sociaux, blogs,...

# Domaines et outils

- On **distingue** (domaine *Pattern Recognition , KD*) :
  - Information Retrieval (IR, cf. Google)
  - Knowledge Extraction (eg. IE/KE)
    - ✖ Machine Learning & Data Mining
  - Text Mining (TM)

**Text Mining** (ou TDM = KDT) :

- Analyse intelligente de texte
- **Objectif** (principal) :  
Extraction de connaissances (et d'information non triviale)  
depuis un corpus de texte libre et non-structuré.

# Qu'est-ce qu'on cherche ?

- Dans les mails, MSN, Blogs, on cherche
    - Des entités (personnes, compagnies, organisations, ...)
    - Des evts : inventions, offres, annonces, attaques, ...
    - Opinion, Sentiment, la description de xx (cf. d'une image)...
    - Détection de **Spam** dans les mails
  - Dans les articles scientifiques, livres :
    - On cherche des sujets / tendances / *Authorship*
      - ☞ En bio-Info/bio-math, le vocabulaire spécifique est un pb. !
  - Dans les journaux / rapports : on peut avoir besoin d'un résumé
  - En Génie Logiciel (les docs) :
    - contrôle des capacités / perf. à partir de la spécification,
    - des conflits entre le code source et la documentation,
- ... ↗

# Qu'est-ce qu'on cherche ? (suite)

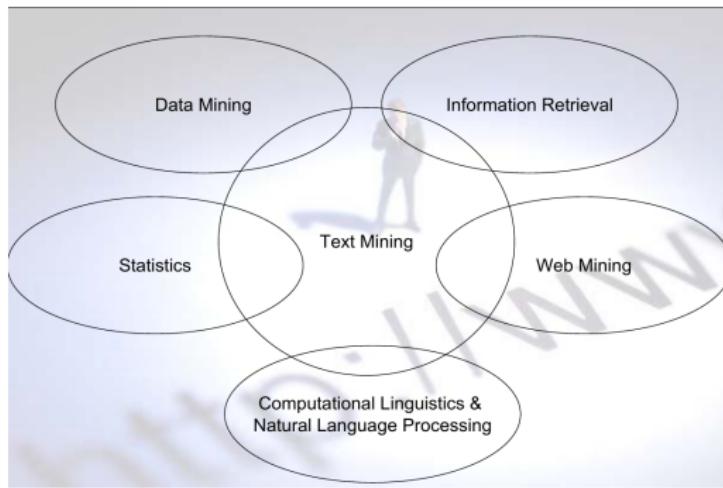
- Sur le WEB

- Chercher de nouveaux produits dans les pages Web des compagnies (pour faire beaucoup de \$\$\$)
- Chercher du contenu illicite !
- Trend / sentiment / Opinion Mining, *Novelty Detection*
- Création d'ontologie, Entity Tracking, IR, ...
  - Difficulté venant de l'hétérogénéité (multi-modal) + hyper liens + ... + + infos cachées ("deep/dark web")

## Que fait-on de ces textes ?

- Classification : ordre / offre
  - Clustering de grosses corpora (*vivisimo / IBM*)
  - Topic maps (*leximancer.com*)
- ☞ Le plus gros système existant : ECHELON (UK/USA)
  - Fouilles industrielles / politique-stratégique / ...

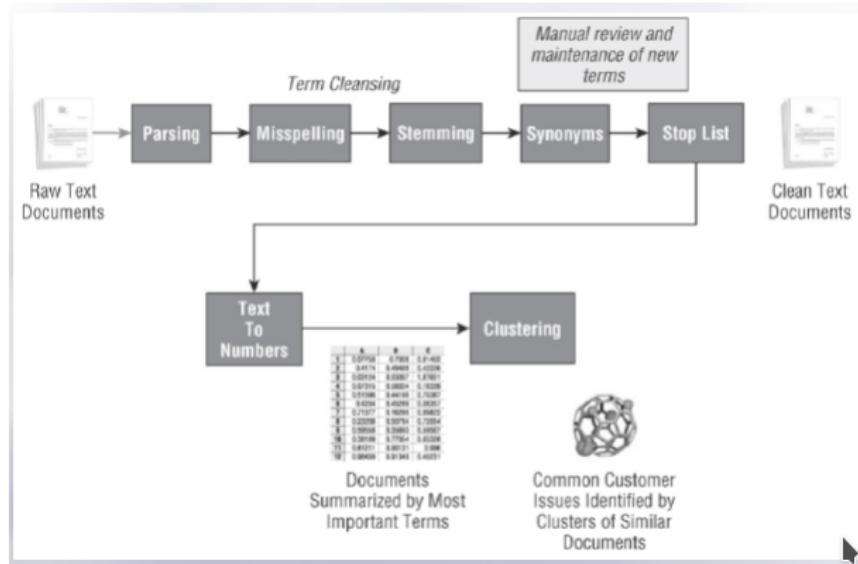
# Domaines impliqués



- TM traite du texte libre en langue naturelle.
- Prend ses sources dans "Computational Linguistics" (CL) et NLP dont des applis pratiques et concrètes sont appelées *Language Technology (LT)*.

# Traitements

Un scénario de Text Mining : **du texte aux nombres**

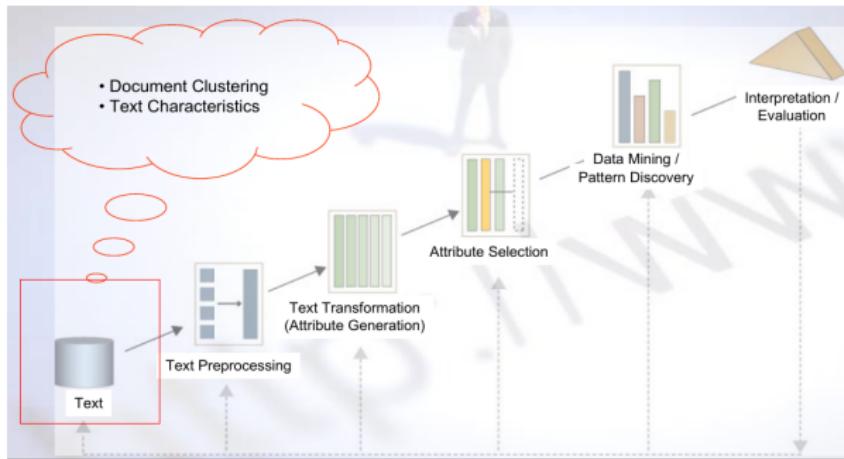


Un scénario "poussé" où on applique le remplacement des termes par des synonymes

- Le texte est (toujours) projeté dans un espace numérique ( $\mathbb{R}$ ) !

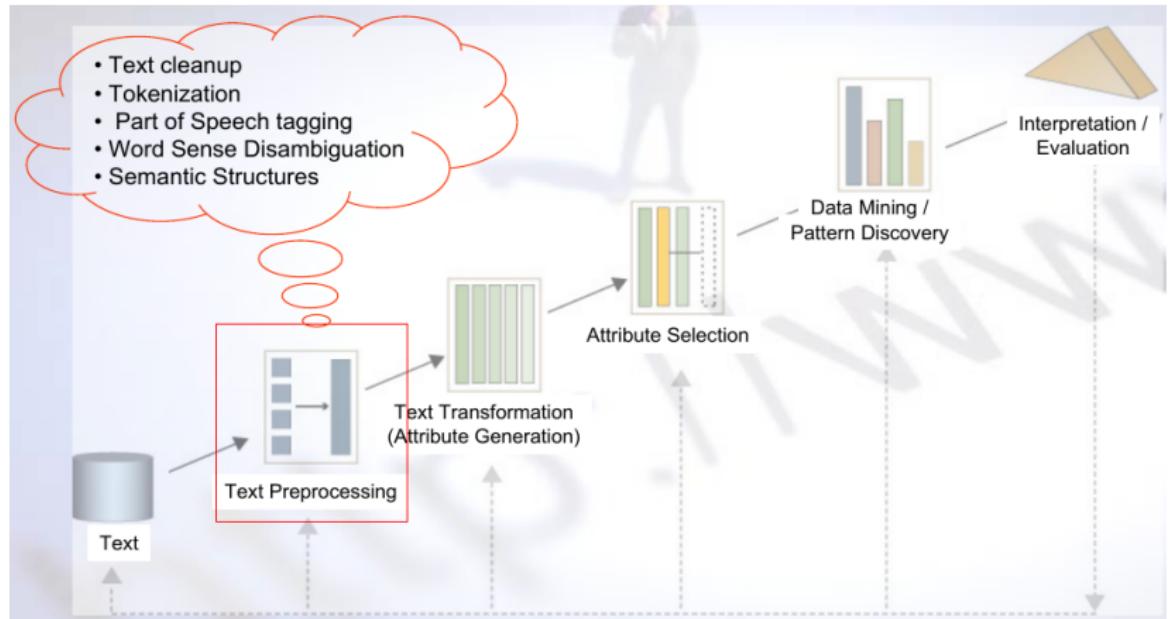
# Traitements (suite)

## Divers Objectifs des traitements



- Faire un cycle Text Mining ... contenant une partie **Analyse**
- **Analyse** : Clustering ou Classification / etc.
  - pour choisir les documents pertinents (relevant) p/r à une requête.
  - pour extraire des informations (IR) / résumer / corriger / contrôler / ....

# Pré processing



# Pré processing (suite)

- **Nettoyage et préparation**

- OCR
- Suppression de pubs / barre de navigation (des pages WEB), etc.
- Normaliser / convertir en format texte les .doc / .pdf / ...
- Traiter tables, formules, légendes, figures, ...
- Ponctuations
- Traiter des mots composés (e.g. *arc en ciel* → *arc\_en\_ciel*)

- **Tokenisation** : isoler des mots

- **Suppression des mots usuels :**

Sauf cas spécifique, les mots tels que "le", "la", "tu", "vous", ... ("the", "a", "an", ...) n'apportent a priori pas d'information intéressante.

- **Stemming** ("racinisation", "désuffixation")

- Identifier le radical / la racine ("stem" = souche) des mots
- Ex : flying/flew deviennent "fly" ➔ Réduire la dimension

# Pré processing (suite)

**Stemming** : deux algorithmes utilisés : *Porter* et *KSTEM*

- KSTEM utilise un dico interne ; moins agressif que Porter.
- **Exemple :**

- Le texte original (dont on supprimera les ponctuations) :

*".... marketing strategies carried out by U.S. companies for their agricultural chemicals, report predictions for market share of such chemicals, or report market statistics for agrochemicals."*

- Résultats **Porter** (quelques différences en rouge)

*"... **market strategi carri** out by u.s. compani for their agricultur chemicals, report predict for market share of such chemicals, or report market statist for agrochemicals"*

- Résultats **KSTEM** (remarquer les "mots")

*"... **marketing strategy carried** out by U.S. company for their agricultural chemicals, report prediction for market share of such chemicals, or report market statistic for agrochemicals"*

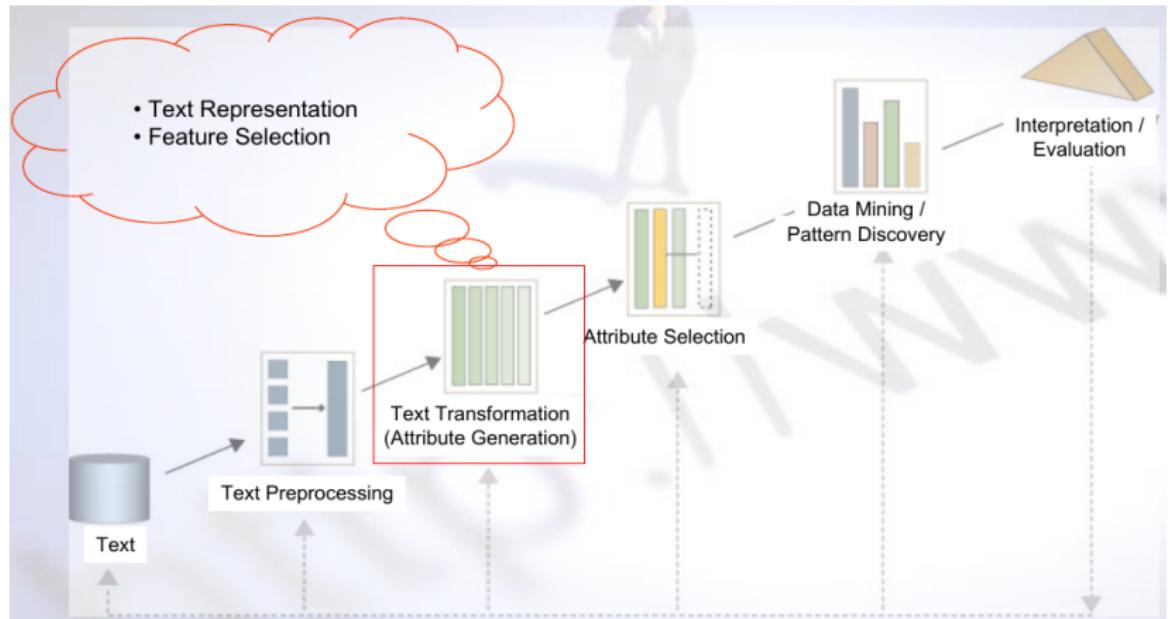
# Pré processing (suite)

Alternative au "Stemming" (et plus proche de KSTEM) : **Lemmatisation**

- **Lemmatisation** : trouve une racine (*Lemme*)  $\in$  dico
- Exemples comparatifs :
  - *Stemming* : "leafs" → "leaf", "leaves" → "leav"
  - *Lemmatisation* : "leafs" → "leaf", "leaves" → "leaf"
  - *Stemming* : "cars" → "car", "automobile" → "automobil"
  - *Lemmatisation* : "cars" → "car", "automobile" → "car"
- ☞ En *Lemmatisation* : "good", "better", "best" → "good"
- Il existe une dizaine d'algorithmes dans ce domaine.

Pour TM, on préfère plutôt la "Lemmatisation" (Attention au cout)

# Génération des attributs



# Techniques d'extraction d'attributs

## 1) *Concept Chunking*

Identification de concepts comme (noms de) "maladies", "lieu" (d'une conf, d'une catastrophe, ...), une "date" (d'un article), etc...

## 2) Annotation *part-of-speech* (POS)

étiquetage des mots dans un texte par leur label de POS (leur classe lex/synt.)

→ Ex. : Noun, Preposition, Conjonction, Pronoun, Verb, Adverb, Adjectif, .....

## 3) *Named Entity Recognition (NER)* :

Identifier des informations textuelles :

les "personnes", "lieux", "organisations", "compagnies", "objets", etc...

## 4) *Analyse syntaxique* : de plus en plus rare.

### • Ces opérations nécessitent la *désambiguation* du sens :

Trouver le sens d'un mot utilisé dans une phrase (e.g. en trouvant son POS-TAG)

- "Les poules du couvent couvent"
- "L'ami de mon oncle qui habite Paris" (*ambiguité de sens*)
- "The king saw the rabbit with his glasses"
- "Paris Hilton was at Hilton in Paris"

# Techniques d'extraction d'attributs (suite)

- Exemple d'Analyse et arbre syntaxique

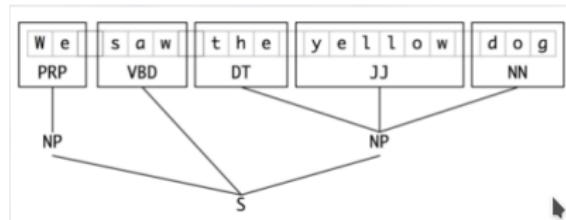
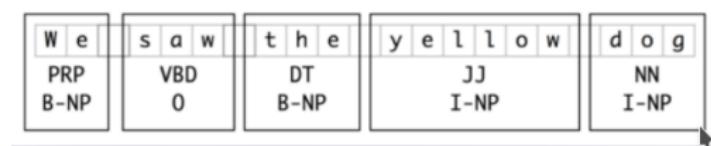


FIGURE 2.1 – Ex. Arbre Syntaxique : Essayer <http://nlp.stanford.edu:8080/parser/index.jsp>

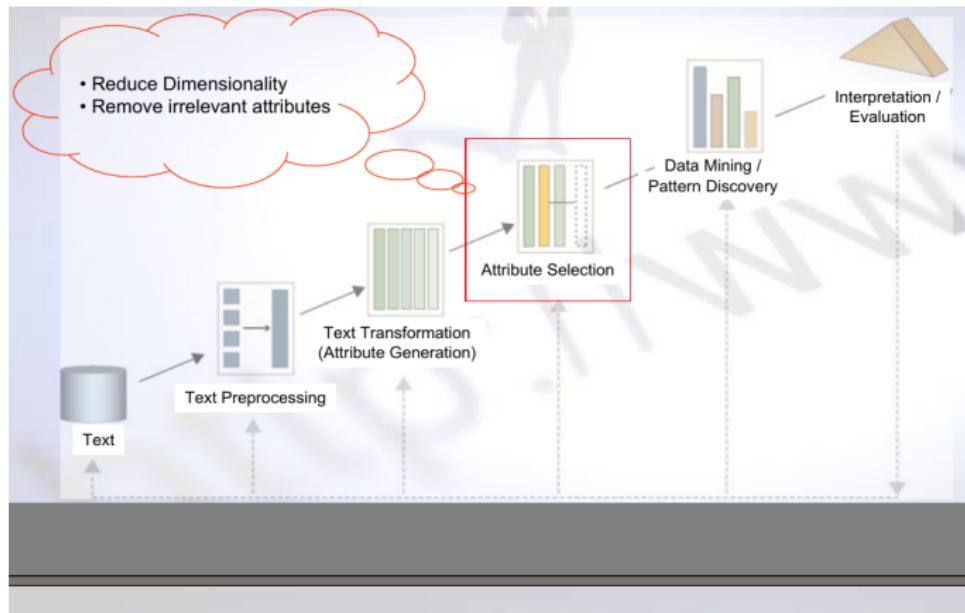
- Exemple de structure de *Chunk*



**I** (inside), **O** (outside), or **B** (begin)

# Sélection d'attributs

**Feature Selection** : sélection d'attributs (après leur extraction)



# Sélection d'attributs (suite)

## Objectifs du "Feature Selection" :

choisir un sous-ensemble **compact** d'attributs avec un pouvoir de **discrimination** maximal.

→ C-à-d. une pertinence (relevance) maximale aux classes et une redondance minimale.

## Quelques outils Python :

- Avec le package **Scikit-learn**, on a les méthodes *Chi-squared*, *Seuil de Variance*, *Score de Ficher*, *Information mutuelle* (MI), etc.
- Il y a aussi *SelectKBest* où l'on précise combien d'attribut choisir.
- Pour la **validation** (& benchmarking) du choix d'attributs, on utilise :  
*Bayes Naive* (très populaire), *KNN*, *Arbre de Décision / Régression*, *SVM*, *K-Means*, *Clustering hiérarchique*, etc.
- Quelques BDs populaires utilisables :  
Amazon review, Movie review, 20Newsgroup, Reuters-21578, BDs. Tweeter, ...

# Sémantique

Peut-on comprendre un langage (fig. thanks toroelof@kth.se)



## Sémantique (suite)

On s'y intéresse selon nos objectifs du Text Mining

- **Sémantique** : on peut être intéressé par :

- Le sens d'un mot : *Lexical semantics*
- Le sens d'une phrase : *[Compositional] semantics*
- Le sens d'un un document ( $\pm$  long) : *Discourse semantics*

## Sémantique : ontologie et WordNet(s)

Un réseau sémantique qui encode les mots d'une langue via

- Synsets : le sens de chaque mot (eg. banque)
- Relations : synonymie, hypernymie / hyponymie (animal / chien), homonymie, hyponymie (Cyan est hyponyme de Bleu), antonymie, holonymie / meronymie(roue / voiture )

- ➔ Le Wordnet anglais encode 155 327 mots avec 175 979 Synsets et 207 016 paires mot-sens (v3.1)
- ➔ Un Wordnet français libre (WOLF) : en dév. (v. beta 2012).



# Sémantique (suite)

- Le mot "tea"

## WordNet Search - 3.1

- [WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options:

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss) "an example sentence"

### Noun

- [S: \(n\) tea](#) (a beverage made by steeping tea leaves in water) "iced tea is a cooling drink"
- [S: \(n\) tea, afternoon tea, teatime](#) (a light midafternoon meal of tea and sandwiches or cakes) "an Englishman would interrupt a war to have his afternoon tea"
  - [direct hypernym / inherited hypernym / sister term](#)
  - [S: \(n\) meal, repast](#) (the food served and eaten at one time)
    - [direct hyponym / full hyponym](#)
      - [S: \(n\) mess](#) (a meal eaten in a mess hall by service personnel)
      - [S: \(n\) square meal](#) (a substantial and nourishing meal) "he seldom got three square meals a day"
    - [S: \(n\) potluck](#) (whatever happens to be available especially when offered to an unexpected guest or when brought by guests and shared by all) "having arrived unannounced we had to take potluck"; "a potluck supper"
    - [S: \(n\) reflection](#) (a light meal or repast)
    - [S: \(n\) breakfast](#) (the first meal of the day (usually in the morning))
    - [S: \(n\) brunch](#) (combination breakfast and lunch; usually served in late morning)
    - [S: \(n\) lunch, luncheon, tiffin, dejeuner](#) (a midday meal)

- [S: \(n\) tea, afternoon tea, teatime](#) (a light midafternoon meal of tea and sandwiches or cakes) "an Englishman would interrupt a war to have his afternoon tea"
- [S: \(n\) dinner](#) (the main meal of the day served in the evening or at midday) "dinner will be at 8"; "on Sundays they had a large dinner when they returned from church"
- [S: \(n\) supper](#) (a light evening meal; served in early evening if dinner is at midday or served late in the evening at bedtime)
- [S: \(n\) buffet](#) (a meal set out on a buffet at which guests help themselves)
- [S: \(n\) picnic](#) (any informal meal eaten outside or on an excursion)
- [S: \(n\) bite, collation, snack](#) (a light informal meal)
- [S: \(n\) nosh-up](#) (a large satisfying meal)
- [S: \(n\) ploughman's lunch](#) (a meal consisting of a sandwich of bread and cheese and a salad)
- [S: \(n\) banquet, feast, spread](#) (a meal that is well prepared and greatly enjoyed) "a banquet for the graduating seniors"; "the Thanksgiving feast"; "they put out quite a spread"
  - [part meronym](#)
  - [direct hypernym / inherited hypernym / sister term](#)
- [domain region](#)
- [S: \(n\) tea, Camellia sinensis](#) (a tropical evergreen shrub or small tree extensively cultivated in e.g. China and Japan and India; source of tea leaves) "tea has fragrant white flowers"
- [S: \(n\) tea](#) (a reception or party at which tea is served) "we met at the Dean's tea for newcomers"
- [S: \(n\) tea, tea leaf](#) (dried leaves of the tea shrub; used to make tea) "the store shelves held many different kinds of tea"; "they threw the tea into Boston harbor"

## Sémantique (suite)

### Pourquoi la sémantique (le sens) est importante ?

- Si on a le sens (correct) de chaque mot, on peut (par exemple) passer à une représentation en logique des prédictats.
  - Le raisonnement logique devient possible.
  - Ex. : en parlant des "compagnies", de la phrase

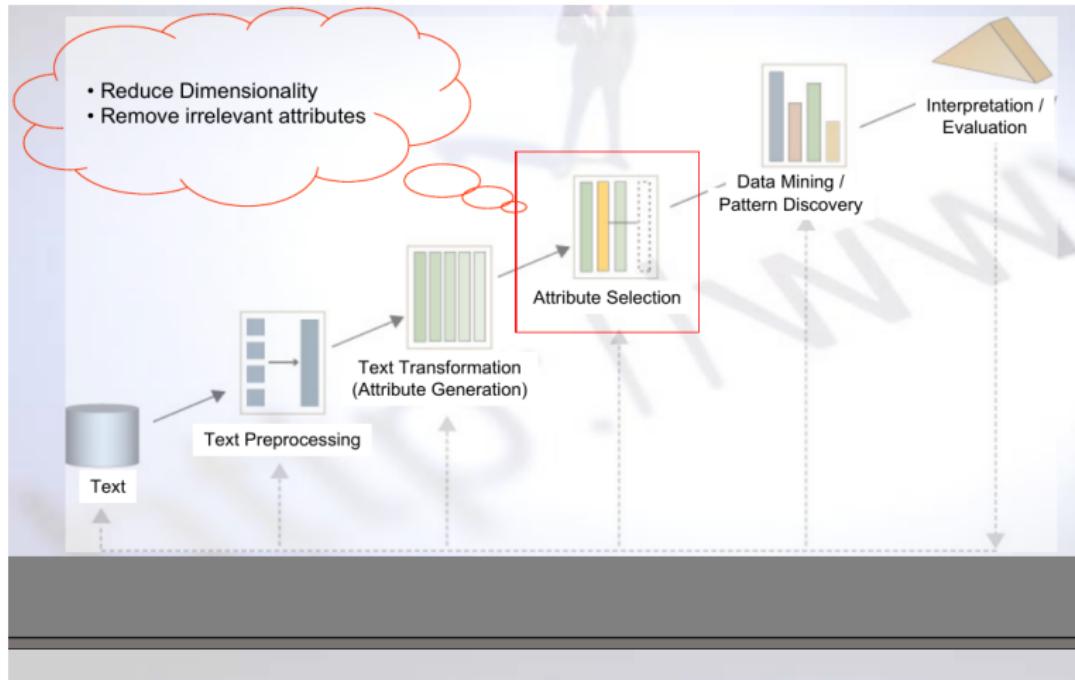
$X$  bought  $Y$       ou       $Y$  was acquired by  $X$ ,

on peut passer à :

$company(X) \wedge company(Y) \wedge buy\_act(X, Y)$

- On peut y intégrer les informations sémantiques venant e.g. de Wordnet.
- On peut mieux trouver des documents pertinents,
- On peut traduire , ...

# Réduction de Dimension



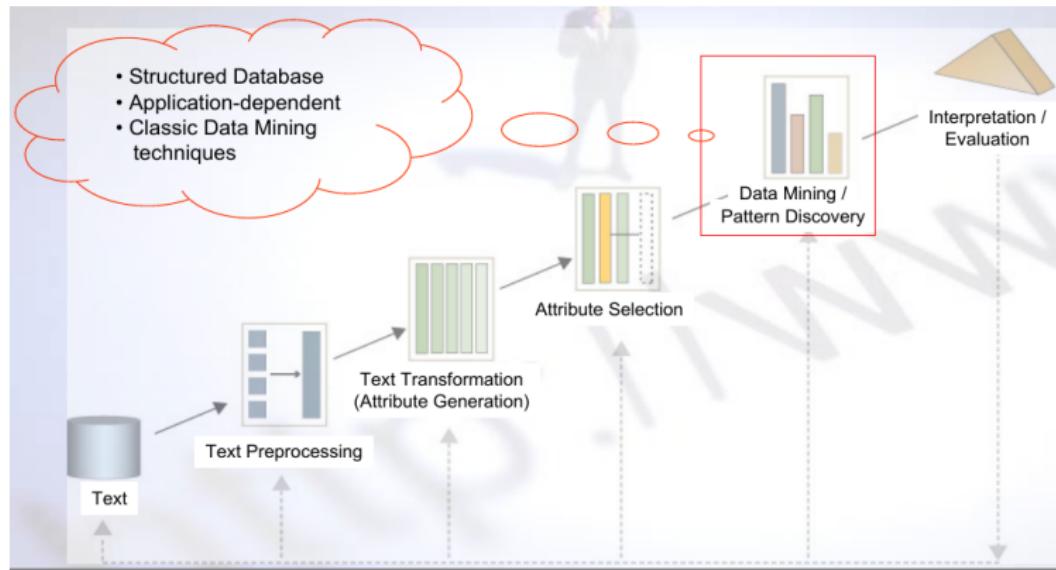
# Réduction de Dimension (suite)

## Réduction de dimension :

- Une grande dimension (beaucoup de termes) pose problème
- Question de ressources (temps/espace) de calcul limitées,
  - La "faisabilité" dépend de la taille du problème
- Certains attributs n'apportent pas beaucoup d'information
  - P. ex. un nom (lexème) dans un article d'actualité n'aidera pas à une classification dans "politique" ou "sport".
- Outils : ACP / SVD / Factorisation de matrices / ...

# Phase de Mining

## Data Mining sur les "termes" retenus



# Phase de Mining (suite)

- Soit : une série d'étapes ont transformé notre corpus en structure numériques.
  - La suite coïncide avec les techniques DM "traditionnelles" (voir aussi NN).
  - Ces techniques sont utilisées sur les résultats (*structures*, p. ex. matrices) issues des étapes précédentes.
- ☞ **Rappel** : tout "apprentissage" est une tache de **recherche d'analogies** dans une **structure** (à découvrir) dans les données.
- On découvre ces structures par des techniques DM telles que :
    - Supervisées : Classification
    - Non Supervisées : Clustering
    - [Semi supervisées]
  - On peut aussi utiliser des techniques ( $\pm$  simples) de IR (à la Google)

# Quelques méthodes de DM

## Clustering de Documents

- Grande Volume de données textuelles  
Milliards de documents à traiter efficacement.
- On ne sait pas d'avance quels sont les documents intéressants (pour notre recherche)
- Clustering : regrouper les documents *similaires* en
  - Par ex. spam ou pas, la classe des (journaux) d'info, de sports, finance, ...
  - C'est un Apprentissage Non-supervisé.
- Quelques méthodes (simples) de *clustering* les plus utilisées
  - K-means (voir ci-après)
  - Bayésiennes (un exemple de MNB en Addendum)
  - Clustering Hiérarchique agglomératif (top-down).
  - ...

# Classification de documents avec Bayes

## Un exemple trivial :

soit un (tout petit) corpus avec les documents :

ID	Ensemble	les mots du document	classe = "Chine" ?
1	Apprentissage	Chinois Pékin Chinois	oui
2	Apprentissage	Chinois Chinois Shanghai	oui
3	Apprentissage	Chinois Macao	oui
4	Apprentissage	Tokyo Japon Chinois	no
5	Test	Chinois Chinois Chinois Tokyo Japon	?

- On voudrait connaître la classe du document (5).

# Classification de documents avec Bayes (suite)

- On aura après les calculs Bayésien

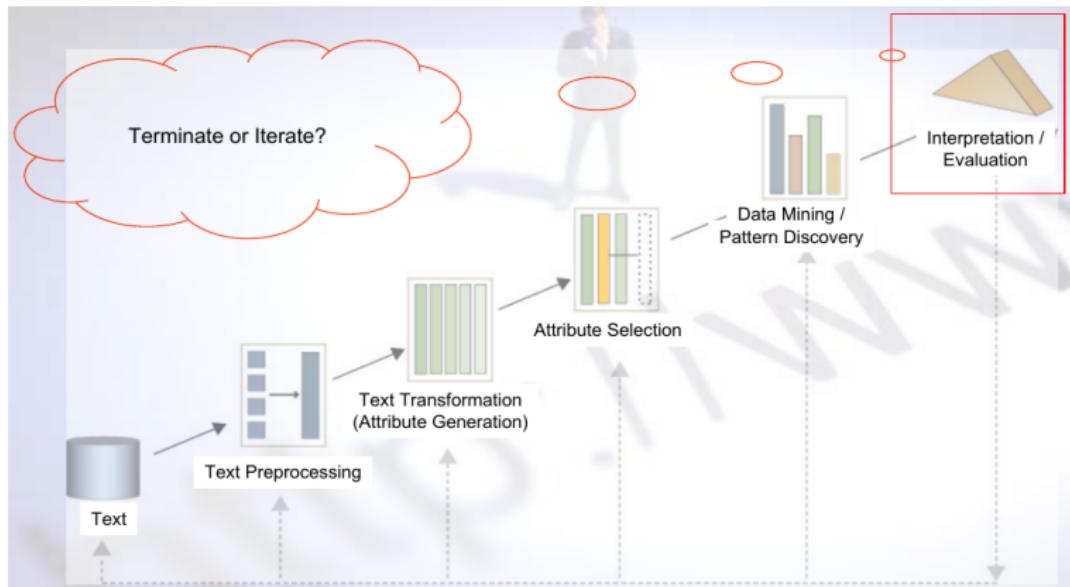
$$\hat{Pr}("Chine" | d_5) = \frac{0,0003}{0,0003 + 0,0001} = 75\%$$

Et       $\hat{Pr}(\overline{"Chine"} | d_5) = \frac{0,0001}{0,0003 + 0,0001} = 25\%$

- **Conclusion** : le document de test ( $d_5$ ) appartient à la classe  $H = "Chine"$ .
  - Ici, les 3 occ de l'indicateur positif ("Chinois") dans  $d_5$  prennent le dessus sur les 2 occ des indicateurs négatifs ("Japon" et "Tokyo").
  - ☞ L'hypothèse de l'indépendance des mots dans la phrase !
  - Une réponse : utilisation de Bi / Diagrammes, Trigrammes, ...

# Evaluation

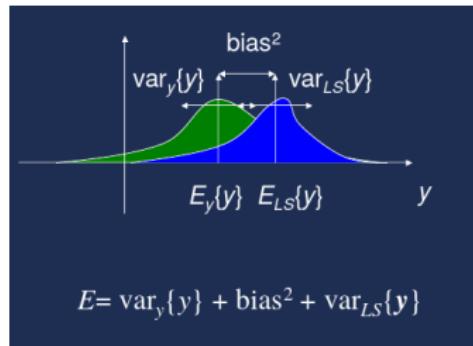
☞ Tout apprentissage artificiel doit être validé.



# Evaluation (suite)

- L'expression de l'erreur d'un apprentissage moindre carré :

$$E = \varepsilon + \text{biais}^2 + \text{var}_{LS}(y)$$



**Notons** : la variance  $\varepsilon = \text{var}_y(y)$  est le **bruit** (noise, Vars. cachées, ...),  
 $\text{biais}^2$  est l'erreur entre l'estimation et les observations (la B.D.)  
 $\text{var}_{LS}(y)$  est la variance de l'estimation-même (app. faits sur  $\neq$  BDs.)

# Evaluation (suite)

## Quelques mesures de base (dans IR) :

- $R = \text{Recall} = \frac{\#\text{extracted relevant}(TP)}{\#\text{total relevant}(TP+FN)} = \text{sensitivity}$

☞ Ne peut être mesurée qu'en phase de validation/test (et non par un end-user car  $FN$  n'est plus dispo)

- $P = \text{Precision} = \frac{\#\text{extracted relevant}(TP)}{\#\text{total extracted}(TP+FP)}$

- $S = \text{Specificity} = \frac{\#\text{extracted Non relevant}(TN)}{\#\text{total negative}(TN+FP)}$

- Et en utilisant  $R$  et  $P$ , on confronte  $R$  vs.  $P$  (observer quand  $R \nearrow$  vs. quand  $P \searrow$ )

$$\mathbf{F-mesure} = \frac{(\beta+1)^2 \cdot P \cdot R}{\beta^2 \cdot R + P} \quad (\beta = 1 \text{ si même poids})$$

- Également  $\text{MAP} = \text{Mean of Average Precision}$

- La valeur de  $MAP$  = la moyenne des maxima de  $P$  (**Precision**) pour différentes valeurs de  $R$  (**Recall**).

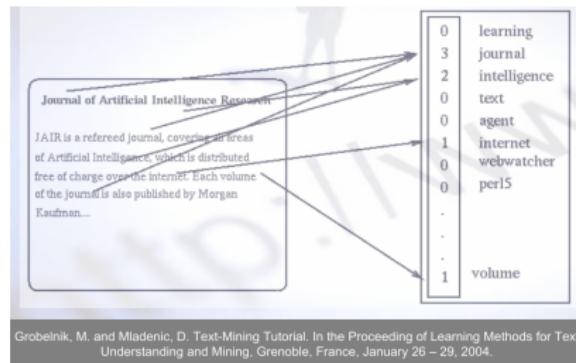
☞ En période de COVID,

- **Sensibilité** (Recall) : capacité à trouver les vrais malades
- **Spécificité** : capacité à bien trouver les non-malades.

# Les Techniques IR

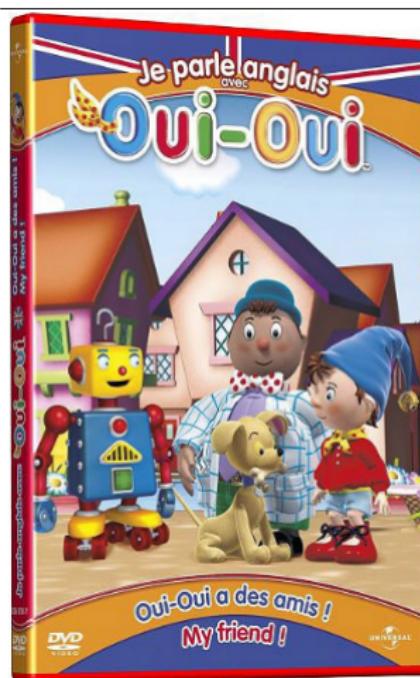
- Deux approches utilisées :
  - Sac de termes / mots ("bag of words" = BOW) : plus classique
  - Vecteur de caractéristiques (Vector space)

## Sac de mots :



- Perte du contexte (avec uni-gramme) → bi-gramme, tri / quadri ...
- Un mot peut être représenté comme une variable aléatoire avec une *importance*.

Passons (un peu) à l'anglais !



# Traditional IR

## Some Problems to face in Text Mining

- **keyword** Mismatch :  
matching based on **keywords** is not sufficient
- **Vocabulary** Mismatch : same concept may be described by different people with different vocabulary
- **Polysemy** : more than one distinct meaning
  - decreases *precision* metric, see below.
- **Synonymy** : many ways to refer to the same object
  - decreases *recall* metric.
- **Performances** evaluation in IR : *Recall* and *Precision*

# Vector space

We can use **Salton's Vector Space Model** to incorporate local and global information in similarity analysis.

- o 1) Binary representation :  $1 \text{ if } \text{term}_i \in \text{doc}_j; 0 \text{ otherwise.}$
- o 2) Count occurrences of a term in a doc :  $\#\text{term}$  in a doc.
- o 3) Compute frequencies (**Tf**) of terms :  $\frac{\#\text{term}}{N}$  in a doc.

		Document-Term Matrix					
		W					
		w <sub>1</sub>	...	w <sub>j</sub>	...	w <sub>J</sub>	
d <sub>1</sub>							
...				...			
d <sub>i</sub>		...		n(d <sub>i</sub> , w <sub>j</sub> )		...	
...				...			
d <sub>J</sub>							

FIGURE 6.1 – co-occurrence matrix

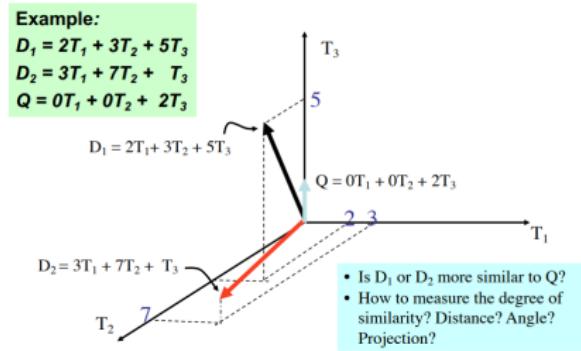
# Vector space (suite)

- 4) Term Weight :  $tfidf(w, d) = tf(w, d) \cdot \left( \frac{|D|}{df(w)} \right)$ 
  - $tf(w, d)$  : term **frequency** (weight of  $w$ 's occurrence in  $d$  = **local** information)
  - $\left( \frac{|D|}{df(w)} \right)$  called  $IDF(w, d)$  : is the **inverse document frequency**
    - a **global** information
  - $df(w)$  : document frequency (#docs containing term  $w$ )
    - $df(w) = |d' \in D, w \in d'|$  is a **global** information
  - $|D|$  : #docs in the database.
- 5) Other Term Weight :  $tfidf(w, d) = 1 + \log(tf(w, d)) \cdot \log \left( \frac{|D|}{df(w)} \right)$
- 6) Other vector space representation :
  - see e.g. **Word2vect**

# Vector space (suite)

## Vector space : an example of graphical representation

- A trivial example for a corpus of
  - 2 documents ( $D_1, D_2$ ),
  - 3 terms ( $T_i$ ) and
  - a query ( $Q$ ).



# Vector space (suite)

- A trivial example of Tf & TfIdf (Thanks to Dr. Grossman)

TERM VECTOR MODEL BASED ON $w_i = tf_i * IDF_i$											
Query, Q: "gold silver truck"											
D <sub>1</sub> : "Shipment of gold damaged in a fire"											
D <sub>2</sub> : "Delivery of silver arrived in a silver truck"											
D <sub>3</sub> : "Shipment of gold arrived in a truck"											
$D = 3; IDF = \log(D/df_i)$											
		Counts, $tf_i$						Weights, $w_i = tf_i * IDF_i$			
Terms	0	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	df <sub>i</sub>	D/df <sub>i</sub>	IDF <sub>i</sub>	0	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>
a	0	1	1	1	3	3/3 = 1	0	0	0	0	0
arrived	0	0	1	1	2	3/2 = 1.5	0.1761	0	0	0.1761	0.1761
damaged	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
delivery	0	0	1	0	1	3/1 = 3	0.4771	0	0	0.4771	0
fire	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
gold	1	1	0	1	2	3/2 = 1.5	0.1761	0.1761	0.1761	0	0.1761
in	0	1	1	1	3	3/3 = 1	0	0	0	0	0
of	0	1	1	1	3	3/3 = 1	0	0	0	0	0
silver	1	0	2	0	1	3/1 = 3	0.4771	0.4771	0	0.9542	0
shipment	0	1	0	1	2	3/2 = 1.5	0.1761	0	0.1761	0	0.1761
truck	1	0	1	1	2	3/2 = 1.5	0.1761	0.1761	0	0.1761	0.1761

# Vector space (suite)

**Do similarity analysis** (between a query and a document)

→ Cosine similarity distance :

$$Sim(Q, D_i) = \text{Cosine } \Theta_{D_i} = \frac{Q \cdot D_i}{|Q| |D_i|} = \frac{\sum_j w_{Q,j} w_{i,j}}{\sqrt{\sum_j w_{Q,j}^2} \sqrt{\sum_j w_{i,j}^2}}$$

- For the above example, one obtains the ranking (vs. query Q) :

**Rank 1 : Doc 2 = 0.8246** ← Cosine(Q, D2) is max.

**Rank 2 : Doc 3 = 0.3271**

**Rank 3 : Doc 1 = 0.0801**

→ Q is more similar to Doc 2 (based on TfIdf)

- *TfIdf* in a vector space method is simple but weak.
- Towards LSA ...

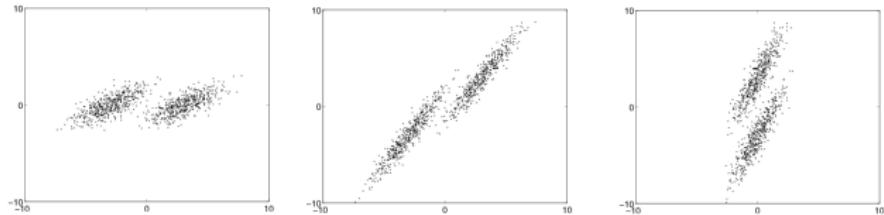
# LSA / LSI and its computational tools

- Try to overcome problems of IR by doing LSA Based on *co-occurrence* matrix.
  - LSA assumes that there exists a **LATENT structure** in word usage, obscured by the variability in a word choice.
- 
- Tools
    - **PCA** : get the principal components by reducing data points dimension (while preserving information)
      - Choose projections that minimize the squared error (distance) of the projection vs. the original data
      - I.e., Get eigenvectors corresponding to the largest eigenvalues of the covariance matrix : **preserve the highest variance of the Data.**
    - **SVD** : see the example...
    - Others

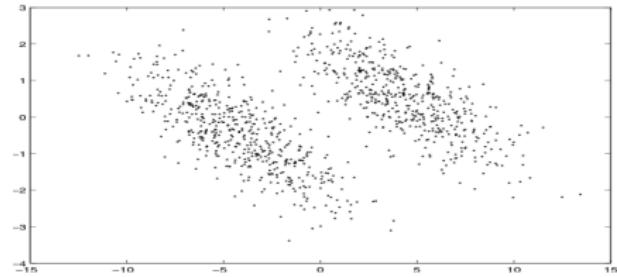
# LSA / LSI and its computational tools (suite)

## An example of dimensionality reduction :

3 sub spaces X-Y, X-Z, Y-Z of a normal distribution 3D space



And the sub space spanned by the first 2 principal components by SVD :  
(similar to PCA)



# LSA / LSI and its computational tools (suite)

- **General idea of LSA**

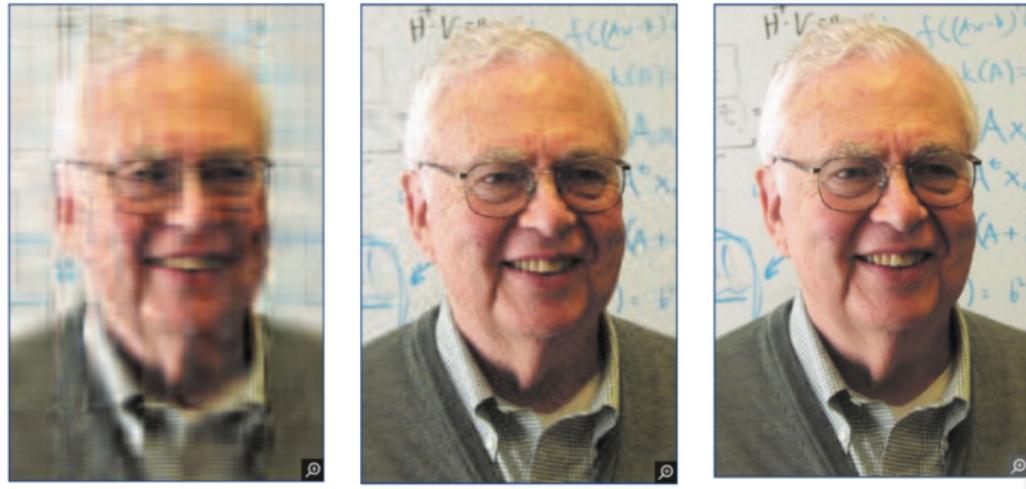
- Map documents (and terms) to a lower-dimensional representation.
- Design a mapping such that the lower-dimensional space reflects semantic associations (*latent semantic space*).
- Compute document similarity based on the inner product in the latent semantic space

- **Goals (and Hyp.)**

- By a similarity measure, "**Similar terms**" map to "**Similar locations** in the lower dimensional space
- Get noise-reduction by dimension reduction

# Dim reduction : SVD

- An Image example (thanks to Matworks.com) :



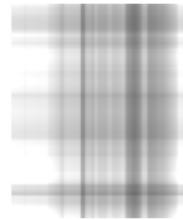
**FIGURE 6.2 –** Rank 12, 50, and 120 approximations to a 598 color photo of *Gene Golub* (a CS. prof at stanford)

# Dim reduction : SVD (suite)

- Another Image : SVD with Tux (See  
[https://en.wikibooks.org/wiki/Data\\_Mining\\_Algorithms\\_In\\_R/Dimensionality\\_Reduction/Singular\\_Value\\_Decomposition](https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Dimensionality_Reduction/Singular_Value_Decomposition))



Original



With one SV



with 35 SV

- ☞ In  $A = USV^T$  ( $A$  is the co-occ term-doc matrix) :
  - The matrix  $U_k$  represent the projection of Terms in the k-space
  - The matrix  $V_k$  represent the projection of Docs in the k-space
- See further farther (the example)

# LSI : a simple example

From LSA (general) to LSI :

Latent Semantic Indexing (LSI) in the context of IR

(LSI Patented in 1989 : <http://lsi.argreenhouse.com/lsi>)

**Purpose :**

- Do dimensionality reduction
- Use LSI to cluster terms.
- Can also find terms that can expand or reformulate the query.

Back to the example : the textual corpus (we had)  $\{D_1, D_2, D_3\}$  :

$D_1$  : "Shipment of gold damaged in a fire"

$D_2$  : "Delivery of silver arrived in a silver truck"

$D_3$  : "Shipment of gold arrived in a truck"

The query : **gold silver truck**.

# LSI : a simple example (suite)

**Step 1 :** Score term weights (#occ.) and construct the term-document matrix A and the query matrix :

(Here #occ. but *Term Weight* :  $w_i = tf_i \times \log\left(\frac{D}{df_i}\right)$  can be used)

$$\begin{array}{c}
 \text{Terms} \\
 \downarrow \\
 \begin{matrix}
 \text{a} & \downarrow & \text{d1} & \downarrow & \text{d2} & \downarrow & \text{d3} & \downarrow & \text{q} \\
 \text{arrived} & \downarrow & 1 & \downarrow & 1 & \downarrow & 1 & \downarrow & 0 \\
 \text{damaged} & \downarrow & 0 & \downarrow & 1 & \downarrow & 1 & \downarrow & 0 \\
 \text{delivery} & \downarrow & 1 & \downarrow & 0 & \downarrow & 0 & \downarrow & 0 \\
 \text{fire} & \downarrow & 0 & \downarrow & 1 & \downarrow & 0 & \downarrow & 0 \\
 \text{gold} & \downarrow & 1 & \downarrow & 0 & \downarrow & 1 & \downarrow & 1 \\
 \text{in} & \downarrow & 1 & \downarrow & 1 & \downarrow & 1 & \downarrow & 0 \\
 \text{of} & \downarrow & 1 & \downarrow & 1 & \downarrow & 1 & \downarrow & 0 \\
 \text{shipment} & \downarrow & 1 & \downarrow & 0 & \downarrow & 1 & \downarrow & 0 \\
 \text{silver} & \downarrow & 0 & \downarrow & 2 & \downarrow & 0 & \downarrow & 1 \\
 \text{truck} & \downarrow & 0 & \downarrow & 1 & \downarrow & 1 & \downarrow & 1
 \end{matrix}
 \end{array}$$

$$\mathbf{A} = \begin{bmatrix} & \text{d1} & \text{d2} & \text{d3} \\ \text{a} & 1 & 1 & 1 \\ \text{arrived} & 0 & 1 & 1 \\ \text{damaged} & 1 & 0 & 0 \\ \text{delivery} & 0 & 1 & 0 \\ \text{fire} & 1 & 0 & 0 \\ \text{gold} & 1 & 0 & 1 \\ \text{in} & 1 & 1 & 1 \\ \text{of} & 1 & 1 & 1 \\ \text{shipment} & 1 & 0 & 1 \\ \text{silver} & 0 & 2 & 0 \\ \text{truck} & 0 & 1 & 1 \end{bmatrix} \quad \mathbf{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

FIGURE 6.3 – Original data term-document matrix A and the query q (we use all words & #occ for simplicity)

# LSI : a simple example (suite)

**Step 2** : Decompose matrix A such that  $A = USV^T$

- Rows of  $V$  hold **eigenvector values**.

These are coordinates of individual term vectors in Docs.

- $V$  (and  $V_k$ ) is the document-TOPIC similarity matrice.
- We may compare documents (Cosine similarity) line by line.

- $U$  is Term-vector coordinates :

- $U$  (and  $U_k$ ) is a term-TOPIC similarity matrice
- we may compare 2 terms by Cosine (is not Synonymy)
- See e.g. lines 6 and 9 : gold and shipment similarity.

$$\begin{array}{c}
 \mathbf{U} \\
 \left[ \begin{array}{ccc}
 -0.4201 & -0.075 & -0.048 \\
 -0.2995 & 0.2001 & 0.4078 \\
 -0.1206 & -0.275 & -0.4538 \\
 -0.1576 & 0.3046 & -0.2006 \\
 -0.1208 & -0.275 & -0.4538 \\
 -0.2626 & -0.378 & 0.1547 \\
 -0.4201 & -0.075 & -0.046 \\
 -0.4201 & -0.075 & -0.046 \\
 -0.2626 & -0.379 & 0.1547 \\
 -0.3151 & 0.6093 & -0.4013 \\
 -0.2995 & 0.2001 & 0.4078
 \end{array} \right] \quad \mathbf{S} \quad \left[ \begin{array}{ccc}
 4.0989 & 0 & 0 \\
 0 & 2.3616 & 0 \\
 0 & 0 & 1.2737
 \end{array} \right] \quad \mathbf{v}^T \\
 \left[ \begin{array}{ccc}
 -0.4945 & -0.6458 & -0.5817 \\
 -0.6492 & 0.7194 & -0.2469 \\
 -0.578 & -0.2556 & 0.775
 \end{array} \right]
 \end{array}$$

# LSI : a simple example (suite)

**Step 3 :** Reduce to a rank  $k = 2$  approximation by keeping the first 2 columns of  $U$  and  $V$  and the first 2 columns and rows of  $S$ .

$$\mathbf{U} \approx \mathbf{U}_k = \begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \quad k = 2$$

$$\mathbf{S} \approx \mathbf{S}_k = \begin{bmatrix} 4.0989 & 0.0000 \\ 0.0000 & 2.3616 \end{bmatrix}$$

$$\mathbf{V} \approx \mathbf{V}_k = \begin{bmatrix} -0.4945 & 0.6492 \\ -0.6458 & -0.7194 \\ -0.5817 & 0.2469 \end{bmatrix} \quad \mathbf{V}^T \approx \mathbf{V}_k^T = \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ 0.6492 & -0.7194 & 0.2469 \end{bmatrix}$$

FIGURE 6.4 – Dim reduction ( $K = 2$  instead of 3)

# LSI : a simple example (suite)

**Step 4 :** Find the new term vector coordinates in this reduced 2-dim. space.

Rows of  $V$  : eigenvector values (coordinates of individual term vectors).

$U_k$  (the reduced Term-vector coordinates) is recalled at right ↴

► E.g. take a look at the similarity of lines **gold** and **shipment**

$$\begin{matrix} \mathbf{S} & & \mathbf{v^T} \\ \begin{bmatrix} 4.0989 & 0 & 0 \\ 0 & 2.3616 & 0 \\ 0 & 0 & 1.2737 \end{bmatrix} & \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ -0.6492 & 0.7194 & -0.2469 \\ -0.578 & -0.2556 & 0.775 \end{bmatrix} \end{matrix}$$

Projection : ( $\text{red } d'_i = d_i \cdot U_k \cdot S_k^{-1}$ )

- $d'_1(-0.4945, -0.6492)$
- $d'_2(-0.6458, 0.7194)$
- $d'_3(-0.5817, -0.2469)$

Terms	Term Vector Coordinates	
a	-0.4201	0.0748
arrived	-0.2995	-0.2001
damaged	-0.1206	0.2749
delivery	-0.1576	-0.3046
fire	-0.1206	0.2749
gold	-0.2626	0.3794
in	-0.4201	0.0748
of	-0.4201	0.0748
shipment	-0.2626	0.3794
silver	-0.3151	-0.6093
truck	-0.2995	-0.2001

# LSI : a simple example (suite)

In LSI, the query is treated as another document ( $d = d^T U S^{-1}$ ).

**Step 5 :** Find the new **query** vector coordinates in the reduced k-dimensional sub space (here,  $k = 2$ ) using  $q_k = q^T U_k S_k^{-1}$

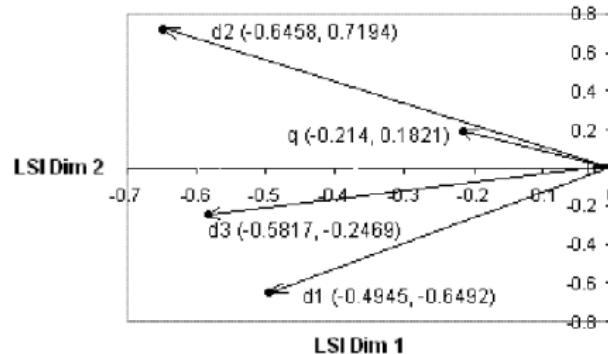
- These are new coordinates of the query in 2-dim space.
- Note the difference between the new representation of **q** (at right) and the original vector of step 1 (at left).

$$q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \begin{bmatrix} 1 \\ \frac{1}{4.0989} & 0.0000 \\ 0.0000 & \frac{1}{2.3616} \end{bmatrix} = \begin{bmatrix} -0.2140 & -0.1821 \end{bmatrix}$$

☞ We can also use  $q_k = U_k^T \cdot q$  (and  $d'_i = U_k^T \cdot d_i$ ).

# LSI : a simple example (suite)

## Step 6 : Doc Cosine similarity :



The textual corpus  $\{d_1, d_2, d_3\}$  :

- $d_1$  : "Shipment of gold damaged in a fire"
- $d_2$  : "Delivery of silver arrived in a silver truck"
- $d_3$  : "Shipment of gold arrived in a truck"

The query : "gold silver truck"

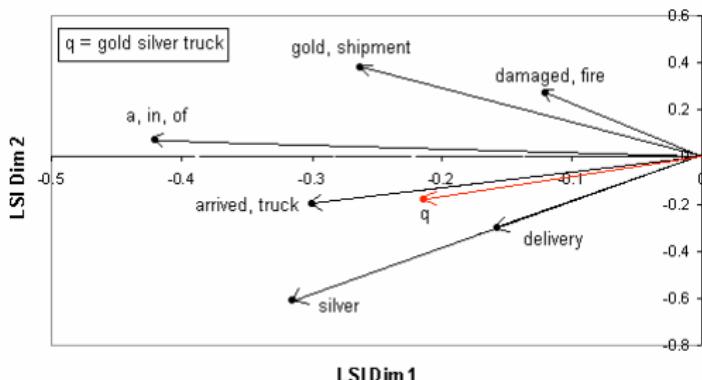
$$q = \begin{bmatrix} -0.2140 & 0.1821 \end{bmatrix}$$

rather near to  $d_2$

# Term clustering & Query Expansion

**Step I.** Group terms into clusters (giving Vector coordinates).

- Clustering by comparing angles' Cosine between any two pair of vectors of  $U_k$
- We can use (e.g.) **K-means**.



- Remember (Doc similarity) : we had  $d_2 > d_3 > d_1$ .

→ Cosine similarity :  $\text{Cosine } \Theta_{d_i} = \frac{q \cdot d_i}{|q||d_i|}$

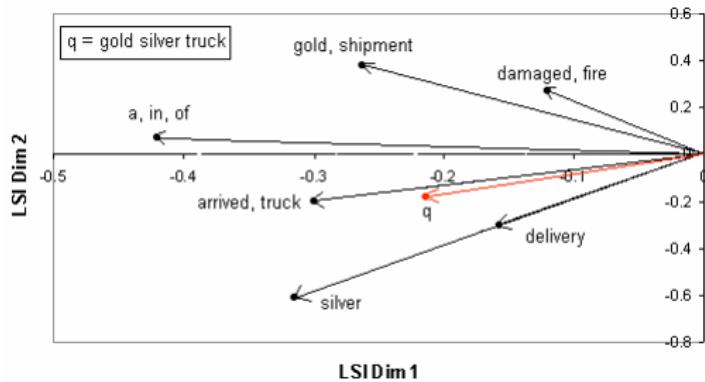
Doing K-means in the reduced space :

→ use  $\text{sim}(q, d) = \text{sim}(q^T U_k S_k^{-1}, d^T U_k S_k^{-1})$

# Term clustering & Query Expansion (suite)

**Step I (cont.)** : The following term clusters are obtained :

1. *a, in, of*
2. *gold, shipment*
3. *damaged, fire*
4. *arrived, truck*
5. *silver*
6. *delivery*



N.B. : Some vectors are not shown since these are completely superimposed / overlapped (those with > 1 term).

N.B. If unit vectors are used (normalization) and small deviation ignored, clusters 3 & 4 and clusters 4 & 5 can be merged.

# Term clustering & Query Expansion (suite)

**Step II.** Find terms that could be used to expand or reformulate the query.

Related to the query "gold silver truck" :

- Clusters 1, 2 and 3 are far away from the query.
- Clusters are 4, 5, and 6 are closer.
- We could use 4, 5, 6 to expand or reformulate the query.

Some examples of the expanded queries that one can test :

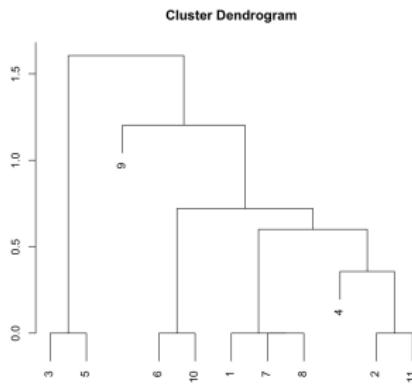
- "gold silver truck *arrived*"     "*delivery* gold silver truck"
- "gold silver truck *delivery*"
- "gold silver truck *delivery arrived*"   etc ...

Documents containing these terms should be more relevant to the initial query.

**Ask me something, I'll tell you what you mean !**

# Hierarchical Clustering

- Example : a Hierarchical agglomerative clustering (*Ward* method) based on the Euclidian distance-matrix of *tfidf* weights gives
  - (order numbers = ranks at right) :



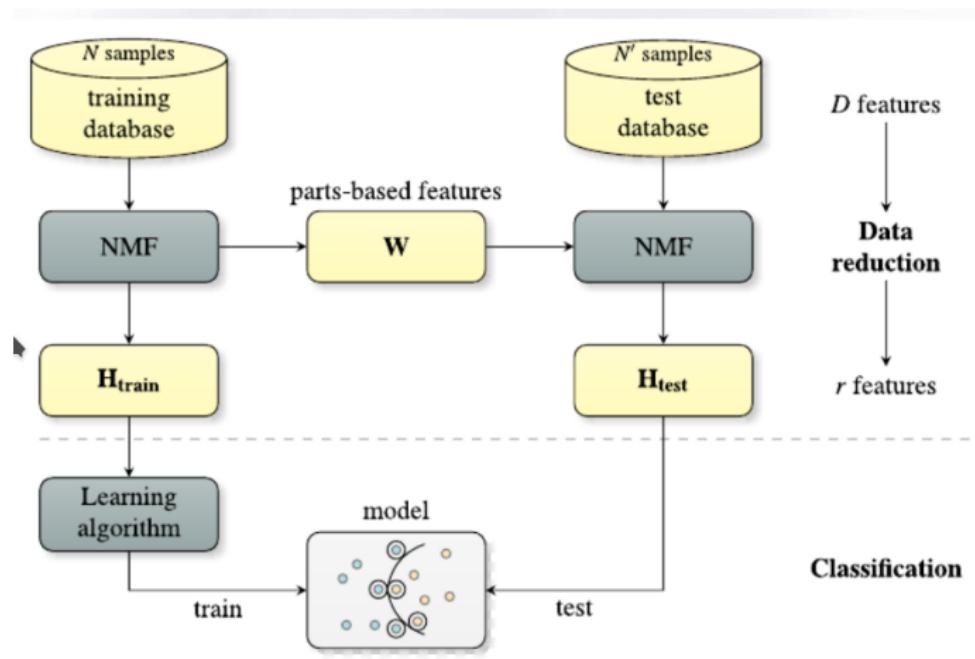
Terms	Term Vector Coordinates	
a	-0.4201	0.0748
arrived	-0.2995	-0.2001
damaged	-0.1206	0.2749
delivery	-0.1576	-0.3046
fire	-0.1206	0.2749
gold	-0.2626	0.3794
in	-0.4201	0.0748
of	-0.4201	0.0748
shipment	-0.2626	0.3794
silver	-0.3151	-0.6093
truck	-0.2995	-0.2001

N.B. : we had very few (11) data.

→ Different weights and distances stay close.

# Compl. : Non-negative matrix Factorization

Another tool for dimensionality reduction



# Back to french



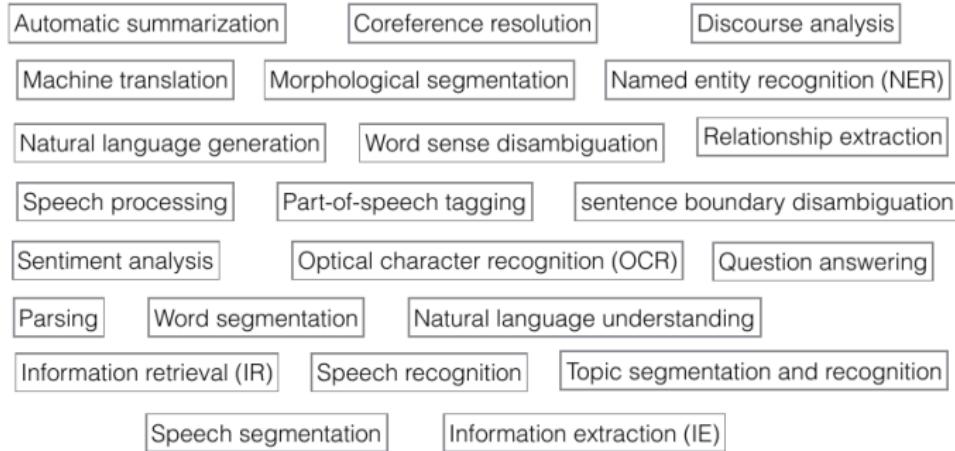
# Word Embedding

**So far !** : si on a "*I am eating an apple*" et "*I am using Apple*"

- Comment tenir compte du **contexte** pour comprendre ...
- Le concept de *vector-space* (e.g. LSA/PLSA/...) a fait l'objet de recherches récentes importantes ;
  - La pléthore de *texte* disponible sur le WEB y est pour qq ch !
- L'objectif est toujours d'améliorer les résultats (de IR), principalement par la prise en charge du **contexte** pour capturer le sens des mots / ph.
- **Word Embedding** est un paradigme qui tente de répondre à ce besoin (avec utilisation des NN/DNN).
- ☞ C'est une technique de **feature learning**
- Parmi les méthodes "word embedding" (où l'on utilise un NN) :
  - Word2vect : CBOW (continuous BOW) / Skip-Gram
  - Glove
  - fastText, Etc.

# Word Embedding (suite)

- Deep learning (DNN) a été appliqué avec succès aux domaines suivants (thanks toroelof@kth.se) :



# Représentation des mots

- Rappel : l'approche *classique* du TLN utilise les mots et les *modèles* (grammaires, modèles linguistiques) pour le traitement du sens.
- Le contexte / le sens peut être représenté par une variable **latente**.  
→ Ces variables sont souvent obtenues (observées) via des calculs algébriques.

**Exemple** : on peut créer une matrice des co-occurrences contenant le nombre d'occurrences des (couples de) mots dans le corpus.

- Les valeurs de la matrice sont pondérées par MI (où  $MI(x, y) = \log_2 \frac{P(x,y)}{P(x).P(y)}$ )

- **Variable latente (shine)** :

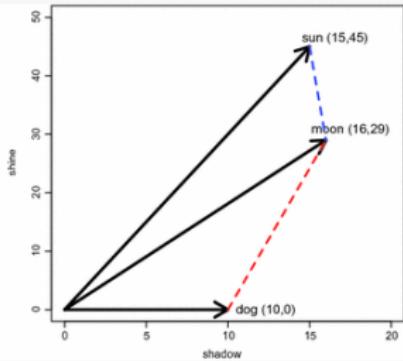
Un exemple partiel :

La distance entre

”dog”, ”sun” et ”moon”

représente une variable latente pouvant être interprétée par ”éclat” (*shine*) → (v. axe Y).

	shadow	shine
moon	16	29
sun	15	45
dog	10	0

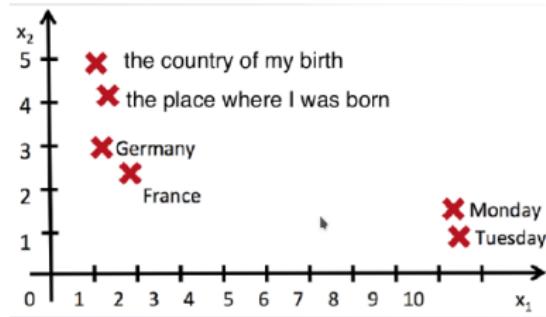
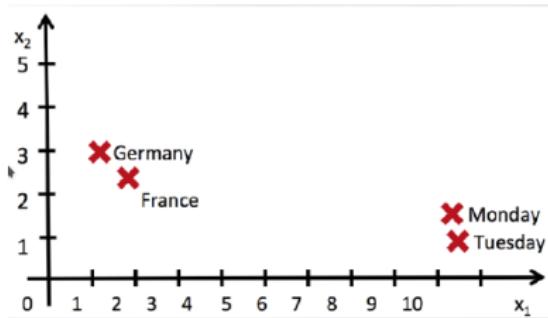


# Représentation des mots (suite)

Contextes proches  $\Rightarrow$  Sémantiques proches

- On devrait retrouver (un monde idéal) :

(Thanks to Bengio. July, 2012, UCLA)



Comment ? :

mot proches / voisins proches / contextes proches /  
 → sémantiques proches (dans un même espace *latent*) ?

- ☞ Plus récent : emploi massif des **Réseaux de Neurones** ....  
 → Les différentes couches des RNs peuvent décrire différents espaces latents.

# Représentation des mots (suite)

## Question de sens et de contexte : vers le Deep learning !

- L'approche vectorielle LSA / co-occ donne des résultats (cf. ex. préc.).
- Approche alternative : préparation du corpus pour Word Embedding :
- Soit la phrase (document) "*I love Candy store*" !
  - L'approche historique utilise les 4 mots de cette phrase.
  - La projection dans un vector-space associe un vecteur de nombres à chaque mot retenu (après pre-processing).
  - Approche plus récente :  
pour tenir compte du contexte, Google a utilisé une représentation dite "**One Hot**" (vect / matrice creuse ↔ rapidité) :  
*Candy* : [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]  
*Store* : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ...]
- Le vecteur *One hot* contient un "1" à l'indice du mot dans le vocabulaire.

# Représentation des mots (suite)

- Un petit exemple de One-Hot avec les 9 mots :

Vocabulary:  
 Man, woman, boy,  
 girl, prince,  
 princess, queen,  
 king, monarch



	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets  
a  $1 \times 9$  vector  
representation

FIGURE 7.1 – Un exemple trivial de codage One-Hot

# Représentation des mots (suite)

## Etape suivante : utilisation des NNs

- On construit un *vector-space* à l'aide des NNS.
- **Continuous Word Embedding :**
  - Combinaison de vector-space avec la prédiction probabiliste
  - La représentation des mots devient + dense (vect. de taille fixe).
  - On présent le vecteur One-Hot de "Candy" et on demande au NN d'apprendre **qu'il a été au voisinage de "Love" / "Store" ?....**

Le NN associe à "Candy" : [0.268, 0.792, -0.177, -0.107, 0.10, -0.542, 0.349, 0.271]

→ Ce vecteur reflète les voisinage de "Candy" dans le corpus.

# Représentation des mots (suite)

☞ Dans *Word2vect* :

- Les mots ont une représentation One Hot
- Ces vecteurs sont en entrée un NN (avec 1 seule couche cachée)
- C'est en quelque sorte son indice dans un dico (look-up table)
- La relation entre un mot et ses mots voisins sera un vecteur de réels extraite de la matrice de pondérations du NN entraîné (voir plus loin).

● Une fois les traitements terminés,

☞ On pourrait par exemple avoir :

*Kingdom + Lady = Queen!*

ou *Kingdom + Man = King*

# Représentation des mots (suite)

- Un (bon) outil : les réseaux de neurones ( $\pm$  Deep).

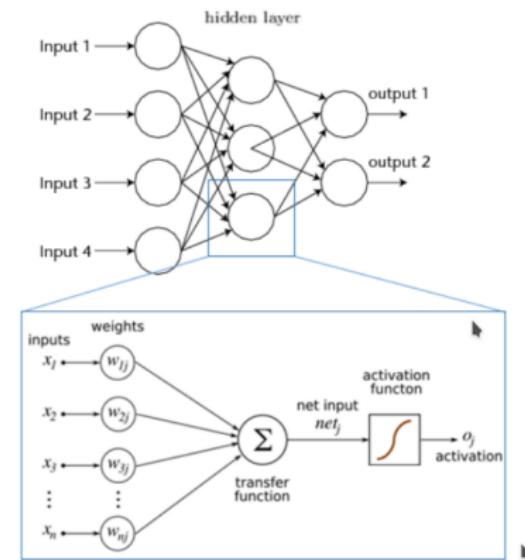


FIGURE 7.2 – Un réseau de Neurones (NN) avec une couche cachée (Deep si nb. H > 1)

# Représentation des mots (suite)

- A l'aide des NN, on obtiendrait des coordonnées adéquates (Fig. Thanks to Bengio. July, 2012, UCLA) (v. aussi +loin) :

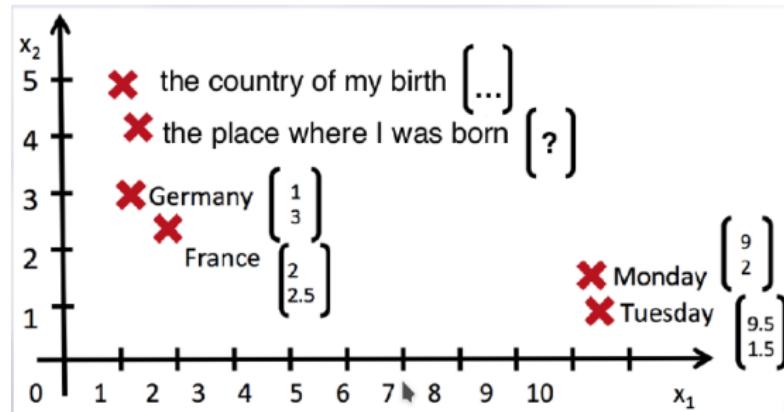


FIGURE 7.3 – Les mots représentées par "Continuous vector-Space" : voir les coordonnées des mots  $\in \mathbb{R}$

# Intro. Word2vect

- Idée : découvrir des régularités et relations linguistiques (cf. WordNet)

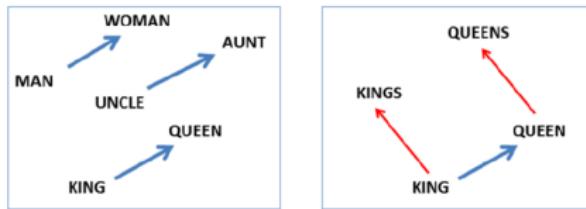


FIGURE 8.1 – Les mots représentées par "Continuous vector-Space" : voir les coordonnées  $\in \mathbb{R}$

- Voyons les deux approches principales citées :
  - Continuous BOW
  - Skip-Gram (Word2vect), Glove = ? Word2vect

# Intro. Word2vect (suite)

```
In [38]: model.most_similar('simple')

Out[38]: [('straightforward', 0.746016800403595),
          ('Simple', 0.7108174562454224),
          ('uncomplicated', 0.6297484636306763),
          ('simplest', 0.6171398162841797),
          ('easy', 0.5990298986434937),
          ('fairly_straightforward', 0.5893306732177734),
          ('deceptively_simple', 0.5743065476417542),
          ('simpler', 0.5537199378013611),
          ('simplistic', 0.551654040813446),
          ('disarmingly_simple', 0.5365327000617981)]
```

```
In [5]: model.most_similar('man')

Out[5]: [('woman', 0.7664012312889099),
          ('boy', 0.6824870705604553),
          ('teenager', 0.6586930155754089),
          ('teenage_girl', 0.6147903203964233),
          ('girl', 0.5921714305877686),
          ('suspected_purse_snatcher', 0.571636438369751),
          ('robber', 0.5585119724273682),
          ('Robbery_suspect', 0.5584409832954407),
          ('teen_ager', 0.5549197196960449),
          ('men', 0.5489762425422668)]
```

FIGURE 8.2 – Les mots similaire apparaissent dans le même sous-espace

## Intro. Word2vect (suite)

```
In [25]: model.similar_by_vector(model['Obama'] - model['USA'] + model['France'])

Out[25]: [('Sarkozy', 0.704483687877655),
          ('Obama', 0.7015002369880676),
          ('President_Nicolas_Sarkozy', 0.642586350440979),
```

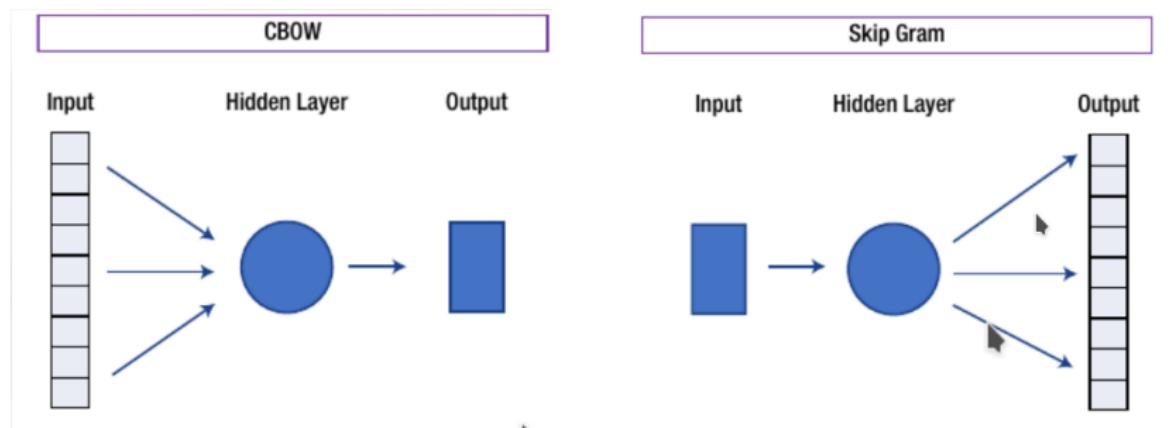
```
In [15]: model.similar_by_vector(model['Dublin'] - model['Ireland'] + model['France'])

Out[15]: [('Paris', 0.740702748298645),
          ('France', 0.6797398924827576),
          ('Issy_les_Moulineaux', 0.6246635317802429),
```

FIGURE 8.3 – Exemple d'opérations algébriques sur les mots dans l'espace vectoriel de mots

# Intro. Word2vect (suite)

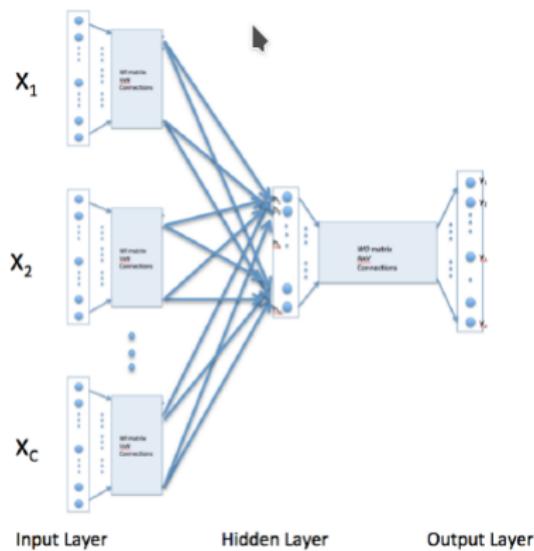
- Deux types de NN pour deux méthodes principales utilisées :  
*Word2vect : CBOW et Skip-Gram*



Symboliquement, CBOW prend en entrée plusieurs mots et prédit le suivant / voisin

Tandis que Skip-gram prend un mot en entrée et prédit les suivants / voisins

# Word2vect : CBOW

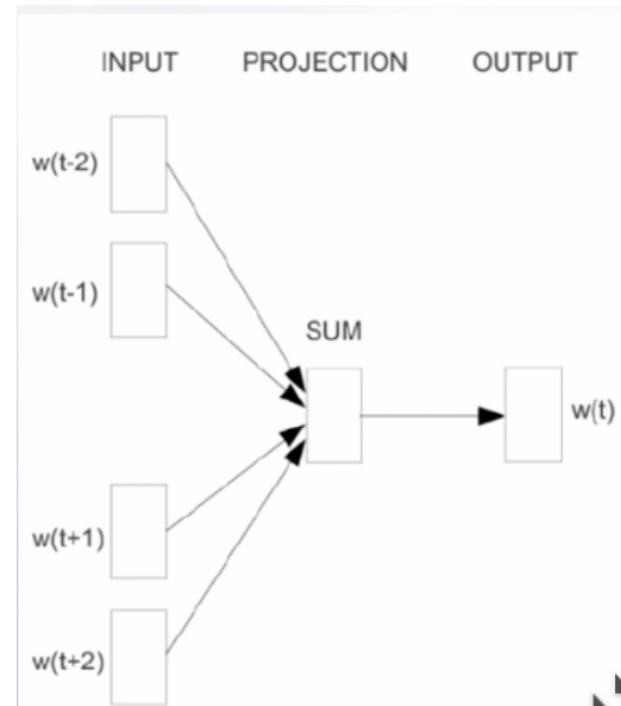


**FIGURE 8.4** — CBOW utilise une seule matrice en entrée qui reçoit de multiples vecteurs Hot-One représentant différents mots d'un contexte (voisinage) pour en prédire un autre.

# Word2vect : CBOW (suite)

## CBOW : caractéristiques du NN

- o L'unique couche cachée est apprise / partagée pour tous les mots
  - o Dans CBOW de base :  
**on perd l'ordre des mots** (du corpus)
  - o La couche de projection peut faire SUM/Average/... ( $\simeq$  fonc. d'activation)
  - o Donnera un vecteur qui désignera le mot à prédire
  - o Mieux : dans une version améliorée (fig. ci-contre), on travaille sur un mot courant en utilisant les mots passés et à venir.
- Voir Exemple ..../..



# Word2vect : CBOW (suite)

Exemple (proche de Doc2Vect, v. +loin) :

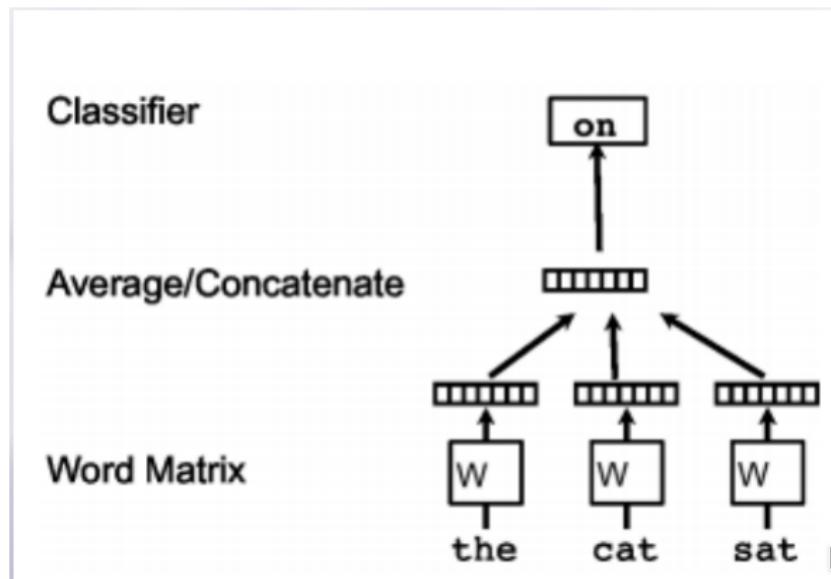


FIGURE 8.5 – CBOW : les mots "the", "cat" et "sat" utilisés pour prédire "on"

# Word2vect : Skip-Gram

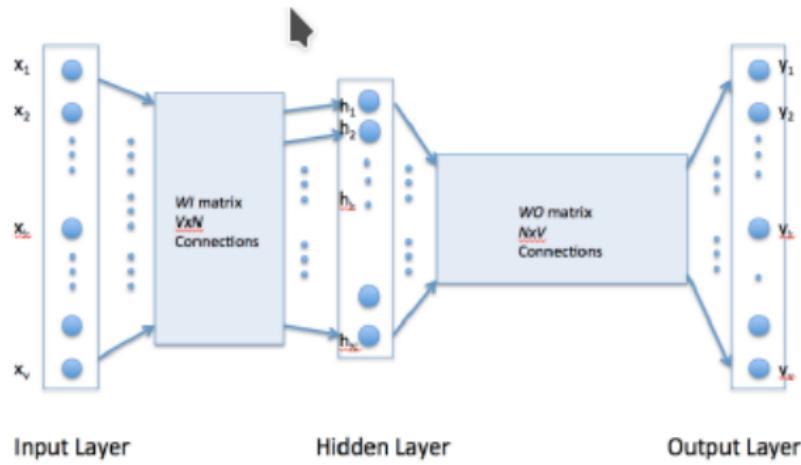
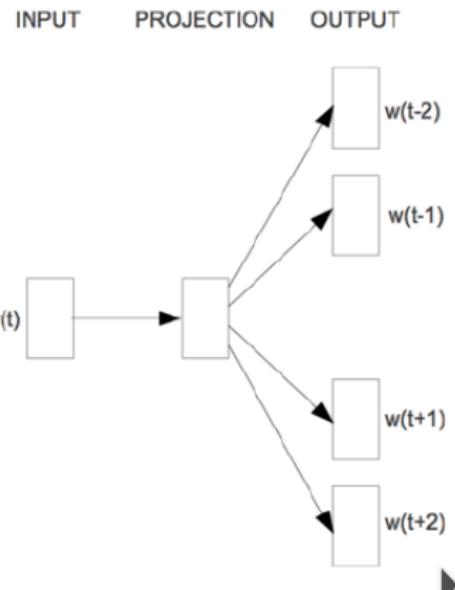


FIGURE 8.6 — Skip-Gram en ses deux matrices de pondération ( $WI$  et  $WO$ ) : on utilise en général  $WI$  pour extraire le vecteur d'un mot

# Word2vect : Skip-Gram (suite)

## Skip-Gram : caractéristiques

- Similaire à CBOW
- Un mot seul  $w_t$  est donné en entrée d'un NN log-linéaire (vs. CBOW)
- Mais au lieu de prédire le prochain mot à l'aide du contexte, on maximise la probabilité de la prédiction d'un mot parmi des mots qui sont en-relation-avec  $w_t$ , rencontrés dans le corpus.
- Ces autres mots considérés auront été (dans le corpus) au voisinage de  $w_t$  dans une "fenêtre" : plus elle est large, mieux est la prédiction
- Si fenêtre (trop) large, une pondération plus faible est appliquée aux mots plus "distants" p/r  $w_t$ .  
(Détails plus loin)



- ☞ **Une idée** : si on mélangeait les mots et les images ?  
→ Sur la base des associations que le NN apprend.

# Word2vect : Skip-Gram (suite)

- Une Illustration de **word2vect**

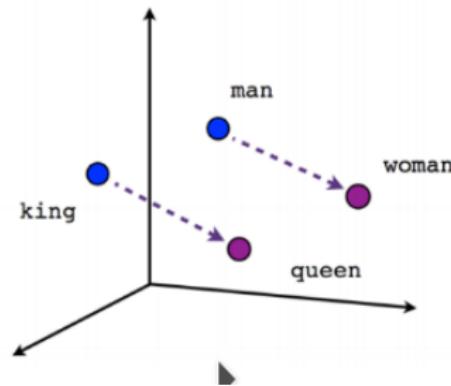


FIGURE 8.7 – Un exemple des mots associés dans l'espace vectoriel créé par *word2vect*

# Skip-Gram : exemple

**Préparation des skip-grams :** pour la phrase "The quick brown fox jumps over the lazy dog." et une fenêtre de taille 3, on aura les paires :

Source Text	Training Samples
The <span style="border: 1px solid black; padding: 2px;">quick</span> brown fox jumps over the lazy dog. →	(the, quick) (the, brown)
The <span style="border: 1px solid black; padding: 2px;">quick</span> brown fox jumps over the lazy dog. →	(quick, the) (quick, brown) (quick, fox)
The quick <span style="border: 1px solid black; padding: 2px;">brown</span> fox jumps over the lazy dog. →	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps <span style="border: 1px solid black; padding: 2px;">over</span> the lazy dog. →	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

☞ Notons les paires de mots.

# Word2vect : synthèse

Rappel (CBOW et Skip-Gram suivent essentiellement le même processus) :

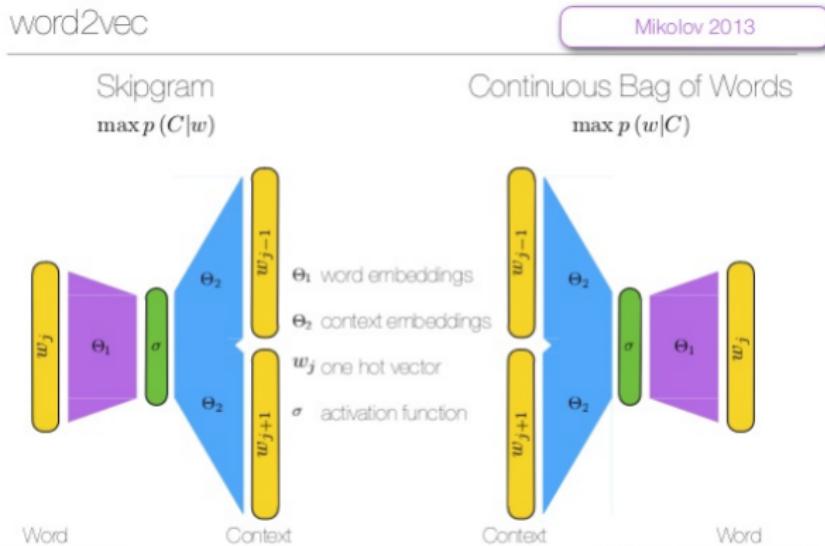


FIGURE 8.8 – Skip-Gram vs. CBOW en un coup d'oeil (merci à Bruno Gonçalves )

# Word2vect : synthèse (suite)

- **CBOW** : on apprend à prédire un mot étant donné un contexte = trouver un mot (le plus probable) dans ce contexte.
  - On évite donc de prédire les mots peu fréquents (dans un contexte donné) car ils ont reçu une probabilité basse (dans le *vector space*).
- **Skip-Gram** : on apprend à prédire un contexte étant donné un mot.
  - Deux mots (soit l'un fréquent et l'autre non) sont traités de la même manière : les deux sont traités comme "un mot dans un contexte".
  - On apprend même les mots rares (peu fréquents), pour peu qu'ils se soient retrouvés au voisinage (dans le même contexte).
- On peut dire que CBOW fait face à de multiples distributions (de mots) en faisant la moyenne des contextes (de mots).
- Skip-Gram ne le fait pas!
  - ☞ Si peu de données, CBOW "se débrouille" mieux mais **plus** il y a des données, **mieux** seront les résultats de skip-gram (qui peut extraire davantage d'information du corpus)

# Word2vect : synthèse (suite)

En python :

```
phrases = [['I', 'love', 'nlp'], ['I', 'will', 'learn', 'nlp', 'in', '2', 'months'],
           ['nlp', 'is', 'future'], ['nlp', 'saves', 'time', 'and', 'solves', 'lot', 'of', 'industry', 'problems'],
           ['nlp', 'uses', 'machine', 'learning']]

import gensim          # !pip install gensim    si pas encore fait
from gensim.models import Word2Vec
from sklearn.decomposition import PCA
from matplotlib import pyplot

skipgram = Word2Vec(phrases, size =50, window = 3, min_count=1, sg = 1)
print(skipgram) # Plein de resultats

# Par exemple, le vecteur pour le mot 'nlp'
print(skipgram['nlp'])

# Un vecteur de 50 reels pour le mot 'nlp' :
[0.00552227 - 0.00723104 0.00857073 0.00368054 - 0.00071274
 0.00837146 0.00179965 - 0.0049786 - 0.00448666 - 0.00182289 0.00857488
 ...
 0.00512787 - 0.00909613 0.00683905]
```

# Word2vect : synthèse (suite)

Visualisations :

```
# save model
skipgram.save('skipgram.bin')

# load model
skipgram = Word2Vec.load('skipgram.bin')

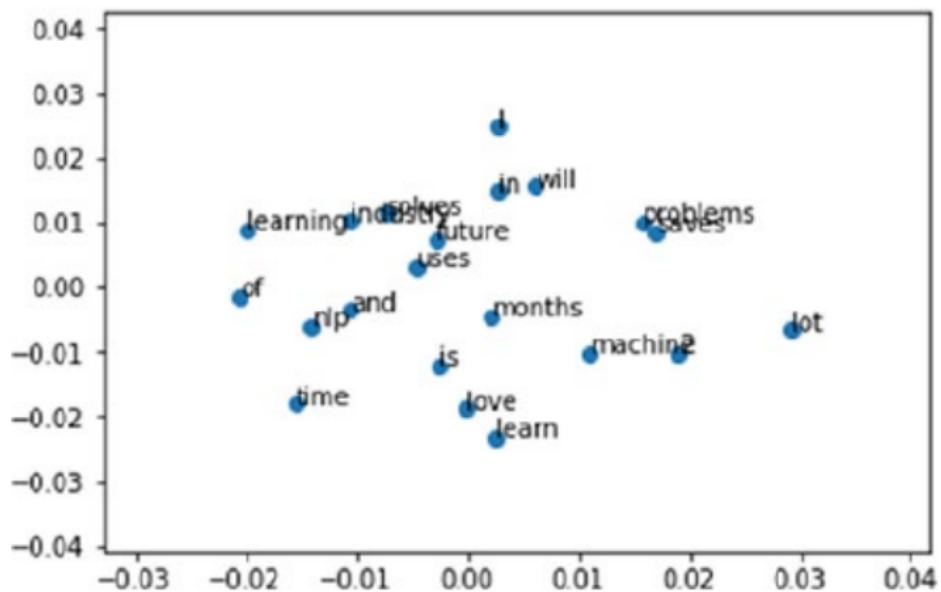
X = skipgram[skipgram.wv.vocab]

# Projection en 2D (au lieu de 50 !)
pca = PCA(n_components=2)
result = pca.fit_transform(X)

# creation du plot (espace de projection)
pyplot.scatter(result[:, 0], result[:, 1])
words = list(skipgram.wv.vocab)
for i, word in itemize(words):
    pyplot.annotate(word, xy=(result[i, 0], result[i, 1]))
pyplot.show()
```

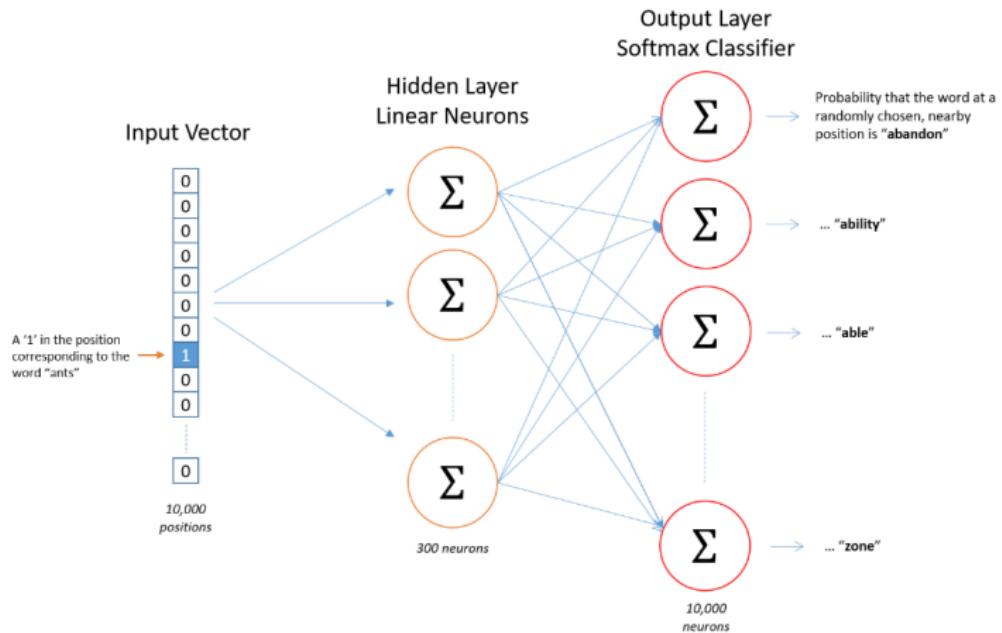
- Le plot (2D) : .../..

# Word2vect : synthèse (suite)



- Se méfier des voisinages en 2D : la bonne dim=50 !

# skip-gram : le fonctionnement du NN



→ Les 300 neurones de Hidden : voir ci-après.

# skip-gram : le fonctionnement du NN (suite)

## Détails du fonctionnement du NN

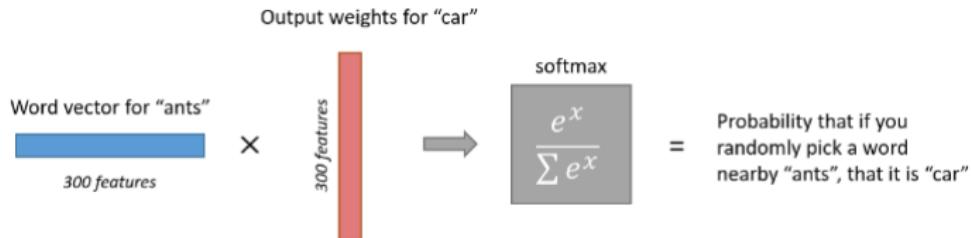
- Les neurones de la couche cachée n'ont pas de (vraie) fonction d'activation.  
→ Une somme suffit (= "rien" / zéros sur One-hot !)
- La couche de sortie utilise la fonction **Softmax** (voir Addendum).
- Pour l'entraînement : on prend une paire de mots (2 One-hot).
- A la fin de l'apprentissage, dans cette matrice des pondérations, ce sont les lignes correspondant aux mots en Input qui nous intéresseront.
  - On note la **réduction de dimension** : de 10000 à 300.
  - La valeur 300 est celle que Goggle a utilisé pour son modèle entraîné sur "Google news dataset"  
(voir <https://code.google.com/archive/p/word2vec/>) .
- Pour un cas donné, on peaufine ce paramètre.

# skip-gram : le fonctionnement du NN (suite)

## La couche output :

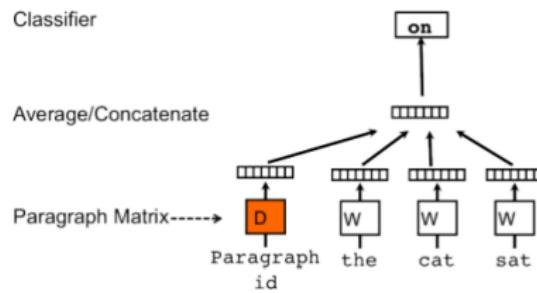
- De l'exemple ci-dessus, le vecteur  $1 \times 10000$  pour "ants" arrive à la couche output.  
→ Softmax produira ensuite une probabilité (de somme 1 !)
- Exemple d'utilisation : "car" vs. "ants" :

Le "vecteur de mot" récupéré sur le Hidden (de la matrice  $W1$  des pondérations) est multiplié par les pondérations  $W2$  (entre Hidden et Output) puis la somme est passé à Softmax pour produire un vecteur de 10000 probabilités (une par mot du vocabulaire).



# Doc2vect

- Dit également *Paragraph2vect* (ou *Par2vect* ).
- **Une Idée** sur la base de Word2vect (approche *PV-DM*) :  
on ajoute un autre vector (Paragraph ID) :



doc2vect comme extension de CBOW

(PV-DM = par2vect with distributed memory, Mikilov-2014)

- ☞ Le modèle fonctionne comme une "mémoire" : se rappelle ce qui manque dans le contexte actuel = comme un "Topic" du paragraphe.
- Le vecteur du document D est également appris pendant l'apprentissage word2vect.

# Doc2vect (suite)

Une version **alternative** de doc2vect : BOW distribué.

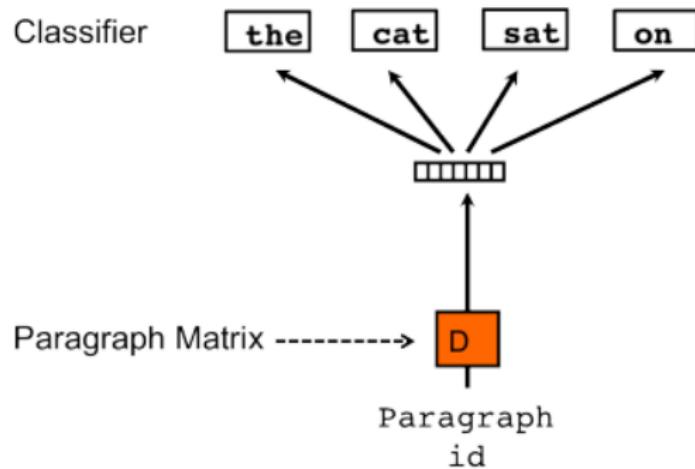


FIGURE 9.1 — PVDBOW (Mikilov-2014) : ici  $D$  représente une distribution via son vecteur

# Doc2vect (suite)

Un exemple d'utilisation de *Paragraph2vect* :

- $PV("Lady\ Gaga")$  et  $PV("LadyGaga") - WV("American") + WV("Japanese")$

(a) Wikipedia nearest neighbours to “Lady Gaga” using Paragraph Vectors. All articles are relevant.

Article	Cosine Similarity
Christina Aguilera	0.674
Beyonce	0.645
Madonna (entertainer)	0.643
Artpop	0.640
Britney Spears	0.640
Cyndi Lauper	0.632
Rihanna	0.631
Pink (singer)	0.628
Born This Way	0.627
The Monster Ball Tour	0.620

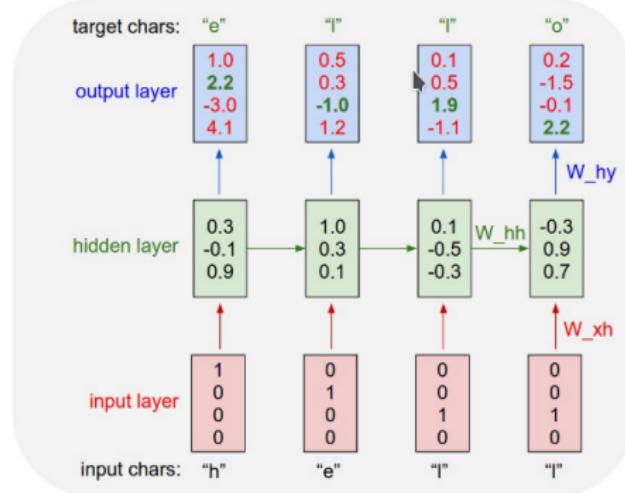
(b) Wikipedia nearest neighbours to  $pV("Lady\ Gaga") - wV("American") + wV("Japanese")$  using Paragraph Vectors. Note that Ayumi Hamasaki is one of the most famous singers, and one of the best selling artists in Japan. She also has an album called “Poker Face” in 1998.

Article	Cosine Similarity
Ayumi Hamasaki	0.539
Shoko Nakagawa	0.531
Izumi Sakai	0.512
Urbangarde	0.505
Ringo Sheena	0.503
Toshiaki Kasuga	0.492
Chihiro Onitsuka	0.487
Namie Amuro	0.485
Yakuza (video game)	0.485
Nozomi Sasaki (model)	0.485

FIGURE 9.2 – PV = paragraphe2vect, WV = word2vect : Dai & al 2014

# NN récurrents : un autre outil TM

Un exemple (à base de caractères) :



**FIGURE 10.1** – Un exemple de NN récurrent à une entrée à 4-dim. Il apprend à deviner le caractère suivant dans un mot. La couche cachée possède 3 neurones (voir aussi <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>).

# NN récurrents : un autre outil TM (suite)

## Remarques :

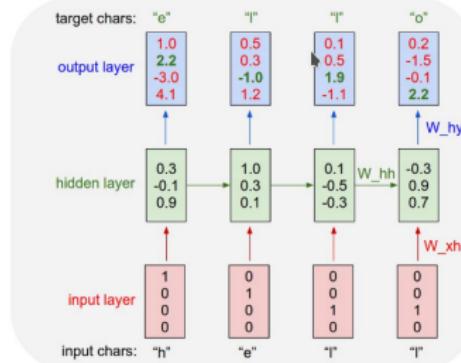
- La colonne droite du diagramme (rappelé) représente l'état du RNN après une itération sur "hell".

Il y a encore loin de la coupe aux lèvres !

- Les **LSTM** sont une amélioration des NN récurrents,  
→ fonctionnent mieux que RNN dans certains cas.

- *FastText* (utilisé par *FaceBook*) apparaît comme une combinaison de RNN et de Word2vect :

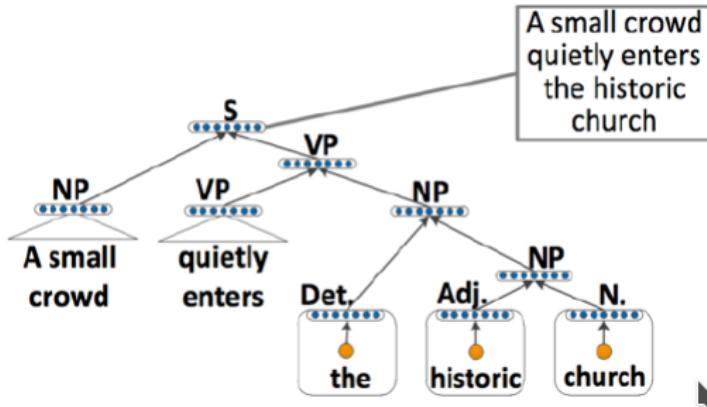
**Par exemple**, pour le mot "where", on envisage les combinaisons (skip-gram de taille 3) "wh", "whe", "her", "ere", "re" (cf. cas particulier du début et fin pour cette taille.)



# NN récursifs

- Une **généralisation** des NN Récursifs.
- Sur la base du principe de la récursivité de la compositionnalité des langues/langages :

"the same operator (same parameters) is applied repeatedly on different components" :



**FIGURE 11.1** – Un réseau de neurones **récursive** (cf. grammaire **récursive** des expressions)

# NN récursifs (suite)

- NN récursifs : un outil pour apprendre le sens et la structure des phrases.

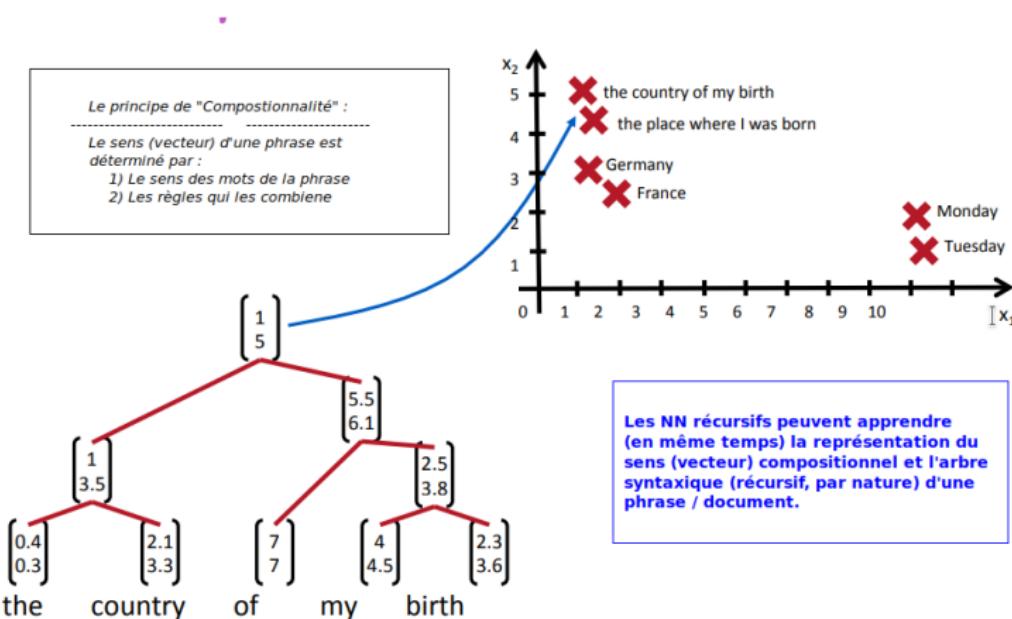


FIGURE 11.2 – Thanks to Bengio. July, 2012, UCLA

## NN récursifs (suite)

- La "composition" des noeuds à l'aide d'un NN :

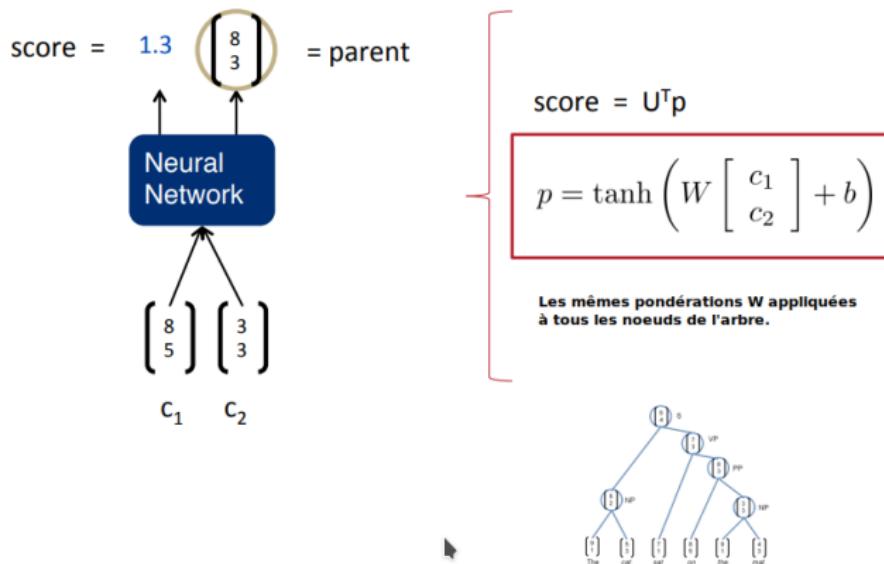
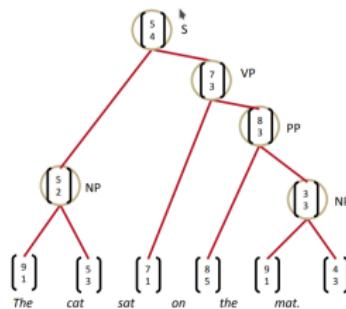
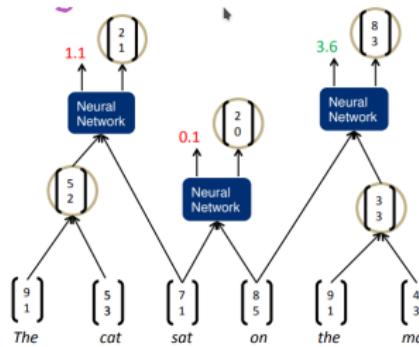
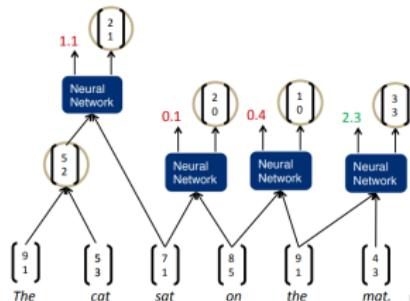
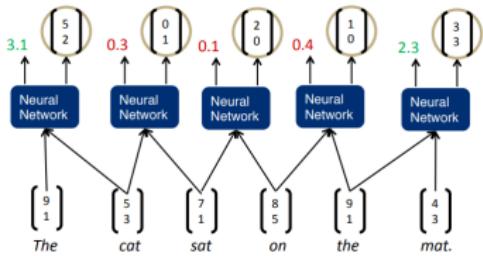


FIGURE 11.3 – Le NN apprend à combiner des noeuds

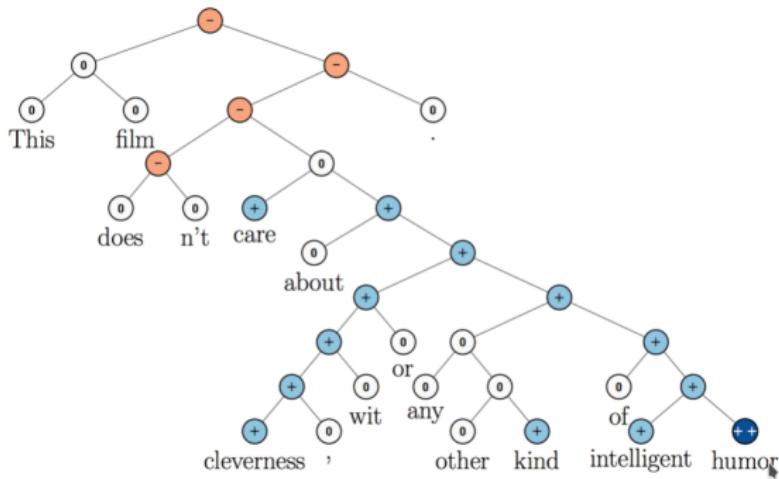
## NN récursifs (suite)

- L'analyse d'une phrase en action :



# NN récursifs (suite)

- Recursif Embedding (RNN) appliqué à l'analyse du sentiment :  
"0" : neutre, "**-**" : négatif, "**+**" : positif. (leur composition suit des règles apprises)



**FIGURE 11.4 –** sémantique Compositionnelle appliquée aux sentiments.  
Socher & al. 2013 (voir [nlp.stanford.edu](http://nlp.stanford.edu))

# Autres

- Comment fait-on en analyse de sentiments / opinions / polarité ?
  - **Lexicon** : utilisé par ex. pour **la polarité des opinions** .
  - **Transformer** : nouvelle approche de la traduction de texte.
  - **PLSA, LDA** (dans les approches plus classiques) : calculs probabilistes des distributions.
- ☞ Liens & similitudes avec le domaine d'images.

# Quelques références (à MAJ!)

- Weiss, Indurkhya, Zhang, Damerau, Text Mining : Predictive Methods for Analyzing Unstructured Information, Springer, 2005.
- Sophia Ananiadou and John McNaught (Eds.), Text Mining for Biology and Biomedicine, Artech House, 2006. Inderjeet Mani, Automatic Summarization, John Benjamins B.V., 2001
- Cunningham, Information Extraction, Automatic, Encyclopedia of Language and Linguistics, 2005.  
<http://gate.ac.uk/sale/ell2/ie/>
- Zhong & Liu (Eds.), Intelligent Technologies for Information Analysis, Springer, 2004
- Peter Morville, Ambient Findability, O'Reilly, 2006
- Et beaucoup d'autres