

Big Data : Text Mining (Version CPE)

Alexandre Saidi

LIRIS-ECL
Février 2020

ECL - Dép. MI
LIRIS UMR 5205 CNRS

Big Data : Février 2020

Version CPE

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

Objectifs TM

Introduction : contexte, motivation, définitions, applications

Fondements : Bagages théoriques dans

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

Technologie : pour créer un système de "Text Mining"

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
- Chatbots
- ...

Contexte général

Constatation : *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 :

- **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 ? Dès que la 5G sera là !

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 récente en :**

- 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) :
 - contréŽle 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")

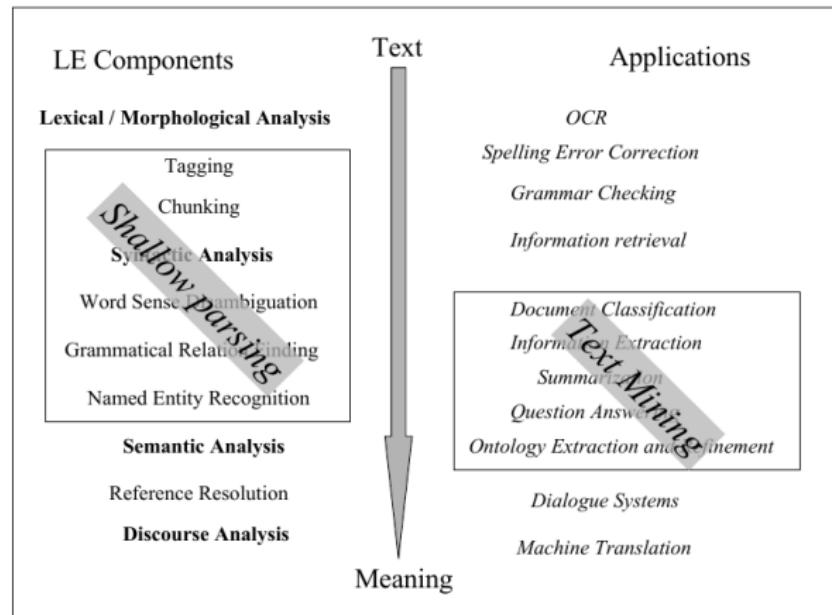
Pour faire :

- Classification : ordre / offre
- Clustering de grosses corpora (*vivisimo / IBM*)
- Topic maps (*leximancer.com*)

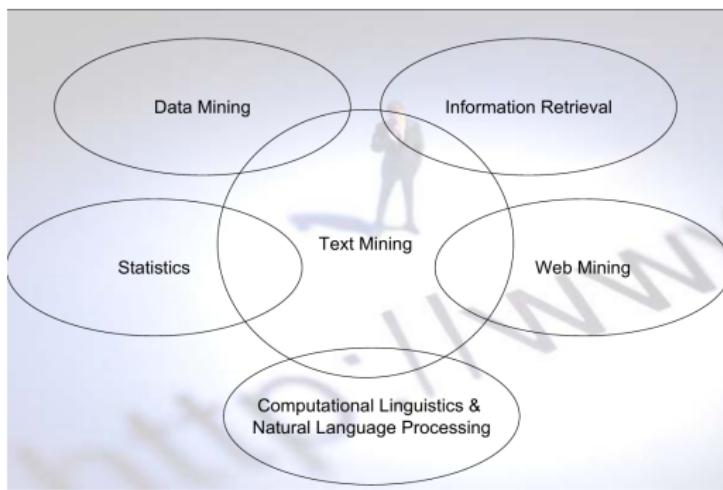
☞ Le plus gros système existant : ECHELON (UK/USA)
→ Fouilles industrielles / politique-stratégique / ...

Approaches

Linguistic Component Extraction (LE) vs. Text Mining (les encadrés internes)



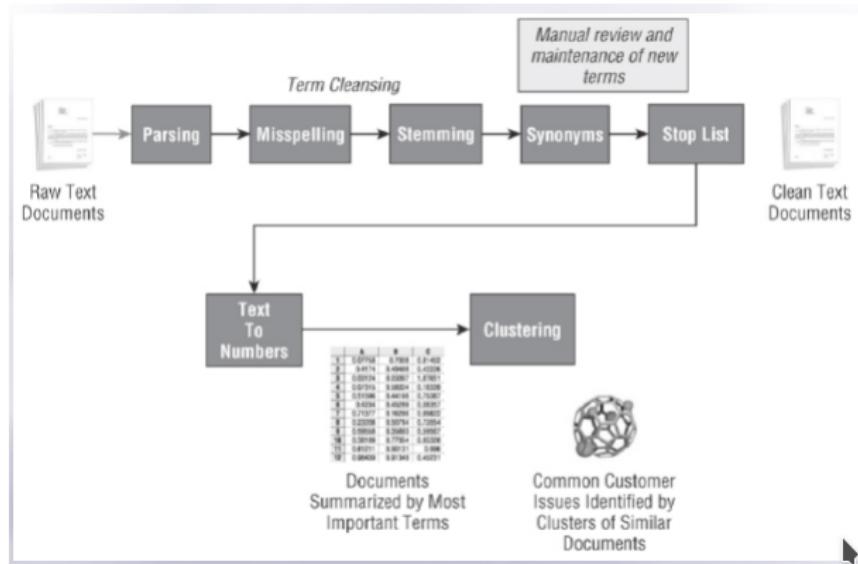
Approaches (suite)



- 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 :

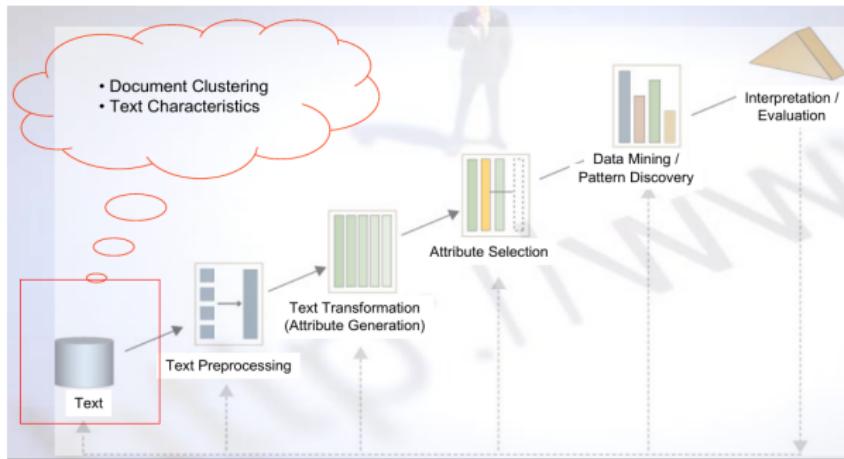


Un scénario "poussé" où on applique le remplacement des termes par des synonymes

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

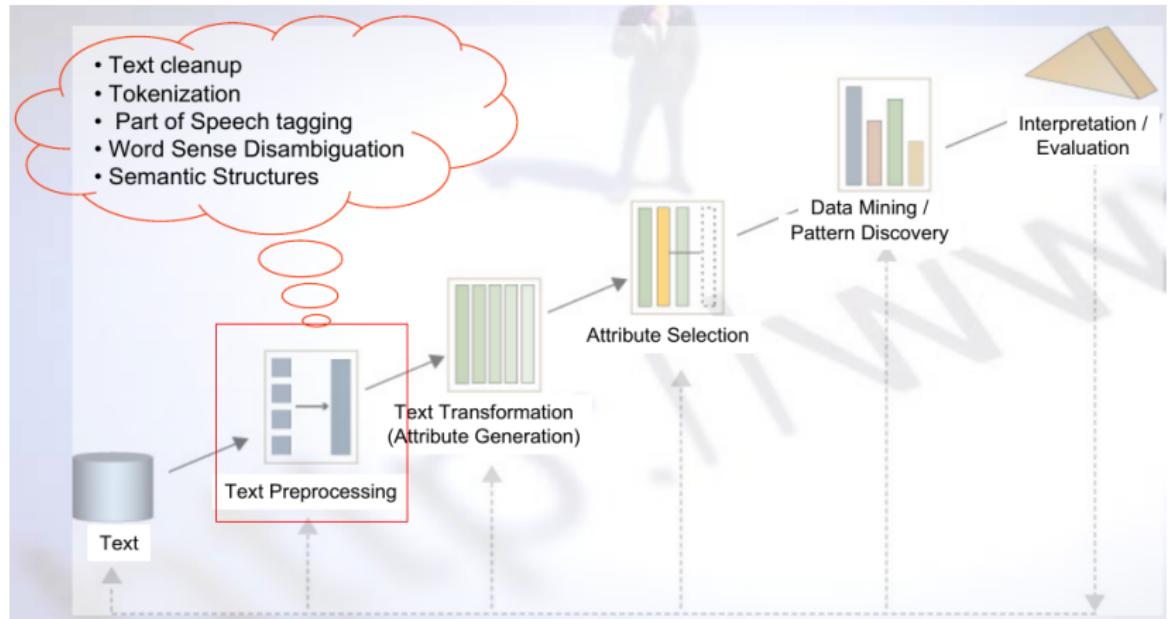
Traitements (suite)

Divers Objectifs des traitements



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

Pré processing



Pré processing (suite)

- **Nettoyage**

- OCR si nécessaire (attention aux erreurs)
- Enlever pubs, barre de navigation (des pages WEB), etc.
- Normaliser, convertir .doc / .pdf / données binaires / ...
- 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**

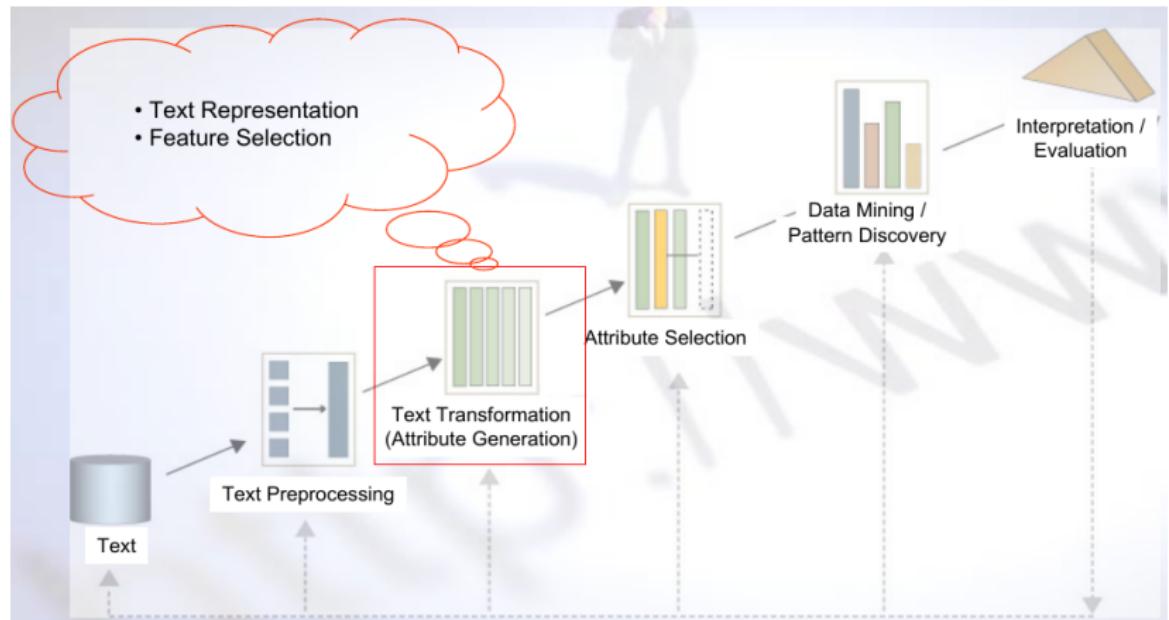
- **Lemmatisation** : trouve une racine (*Lemme*) \in dico
 - Nécessite un POS tagging (part of speech analysis)
 - POS-Tagging : extraction de la catégorie lexicale
 - La lemmatisation peut nécessiter une analyse **Morphologique**
- Exemples comparatifs :
 - *Stemming* : "leafs" → "leaf", "leaves" → "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.

Pré processing (suite)

- La lemmatisation a besoin de **POS-tagging**
 - peut nécessiter une analyse morphologique
 - Lui permet de (p. ex.)
 - distinguer "eur" de "chanteur" (une personne)
de "eur" de "écailleur" = un objet pour écailleur
 - distinguer "arm" de "army" (pas le cas en Stemming)
 - distinguer "couve" de "couvent" (verbe / substantif)
 - Il est possible de faire la Lemmatisation à base de ML :
 - Les "termes" retenus sont des caractéristiques (*feature extraction*)
 - Le "feature extraction" peut être apprise ! (*feature learning*)

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

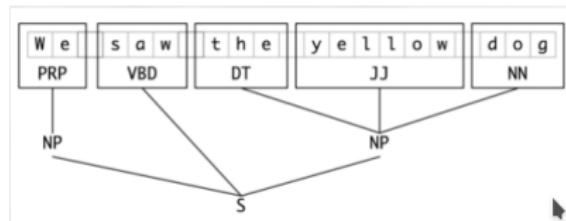
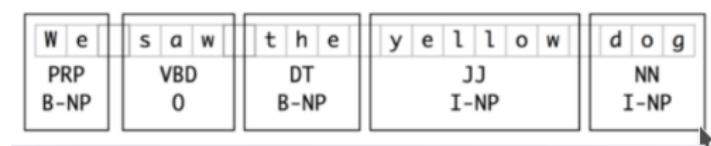


FIGURE 2.1 – Ex. Arbre Syntaxique : Essayer <http://nlp.stanford.edu:8080/parser/index.jsp>

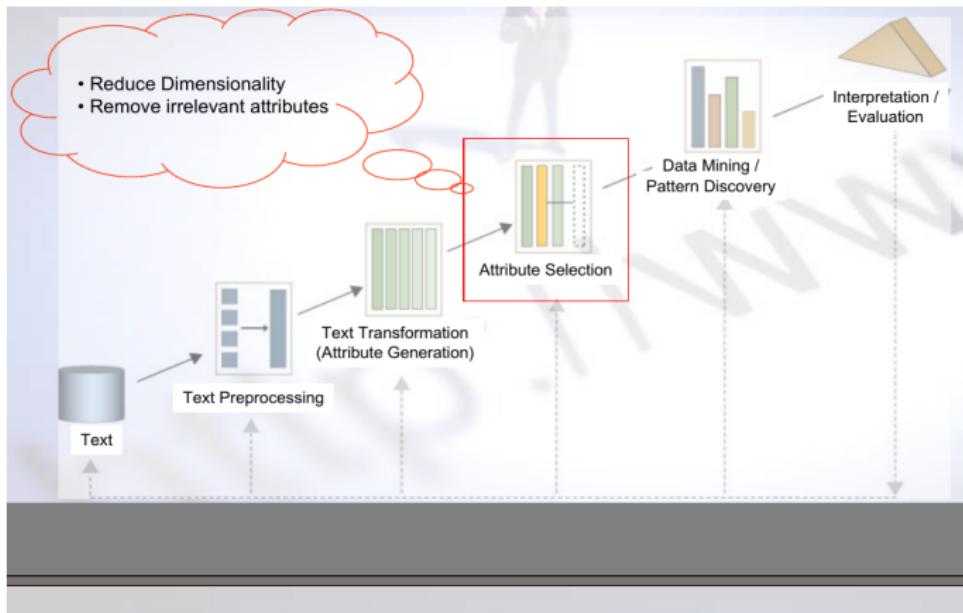
- Exemple de structure de *Chunk*



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, ...

Complément sur Feature Selection

- Quatre grandes approches du "feature selection" en Text Mining :
 - **Filtre** : par analyse statistique ; > 30 méthodes disponibles
 - Ex. : *Chi-squared, entropie, Indexe Gini / Fisher / Bayes, ...*, etc.
 - **Embedded (Feature Learning)** : intégré (*embedded*, embarqué) dans l'étape d'apprentissage
 - Le classifieur nous aide à trouver les meilleurs (V. fig. svte.).
 - **Wrapper** : choisit différents sous-ensembles d'attributs puis les évalue à l'aide d'un classifieur (cf. **RF**) ;
 - Sélection des sous-ensembles par méta-heuristiques à base d'algorithmes Bio-inspirés :
Algorithmes Génétiques, PSO (Particle swarm optimization), algorithmes *Firefly* (lucioles) / Colonies de fourmis / d'abeilles, etc.
 - ➔ V. P. svte. .../..

Complément sur Feature Selection (suite)

- **Hybrid** : combinaison de méthodes *Filtre* et *Wrapper*.
 - Celles qui insistent plutôt sur la minimisation de la redondance
 - Et celles qui maximisent la pertinence de l'information

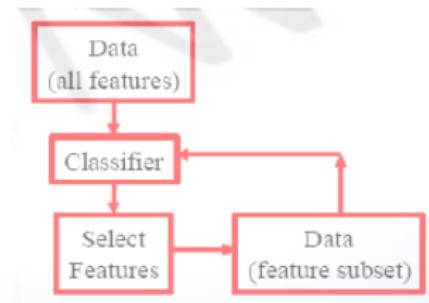
La fusion de ces deux groupes donne les méthodes hybrides.
- On note l'intérêt des méthodes *Embedded* qui **apprennent** à choisir les meilleurs attributs (features).
 - C'est le cas de la plupart des méthodes travaillant avec un vector-space (un corpus devient une matrice de réels) :
LSA, Word2vect (Skip-Gram, CBOW), Glove, fastText, ...
 - Voir plus loin.
- Pour les méthodes Bio-inspirées :
 - Voir les méthodes "Inspyred" (<http://aarongarrett.github.io/inspyred/>) ou "metaheuristic_algorithms_python"
(https://github.com/tadatoshi/metaheuristic_algorithms_python)

Complément sur Feature Selection (suite)

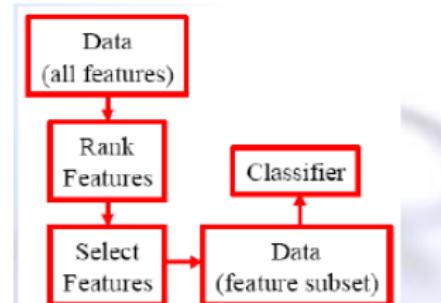
Learning to select Features : embedded vs. classique

- Sélection d'attributs intégrée à l'étape d'apprentissage.
 - Le classifier doit pouvoir évaluer les features (on lui apprend !)
 - Le classifier devient une partie de l'étape "feature selection"
 - Processus itératif et ad-hoc
 - **Par Ex.** une régression LASSO ajouterait la pénalité $\lambda||W||_1 = \lambda \sum^k w_i$ au cout de la classification où W est le vecteur des pondérations associé aux k features à trouver.
Le choix des features minimisera le cout total (cout+pénalité) du process ML.

N.B. : p



Méthode Embedded (intégrée au ML)



Méthode non-Embedded (e.g. Filtre)

Sémantique

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



Sémantique (suite)

Selon nos objectifs du Text Mining, on peut (à l'étape suivante) passer à la sémantique (ou pas).

☞ Attention aux difficultés / céZut.

- **Sémantique** : on peut être intéressé par :
 - Le sens d'un mot : *Lexical semantics*
 - Le sens d'une phrase : *[Compositional] semantics*
 - Le sens d'un document (\pm long) : *Discourse semantics*
- TLN s'intéresse à découvrir une *distribution de probabilités* sur un espace de mots et de phrase dans une langue naturelle.

Dans une phrase, la probabilité d'un mot est (à défaut !) simplement sa fréquence : $P(w_i) = \frac{\text{nb occ de } w_i}{\text{total des mots}}$

Sémantique (suite)

A propos de la distributions de probabilités :

- Par exemple, dans un document (de notre corpus !)

"She was tired of reading asking herself what was the use of a book without picture!"

- On a la fréquence de chaque mot $P(w_i) = \frac{\text{nb occ de } w_i}{\text{total des mots}}$
- Mais dans cette phrase, on peut s'intéresser au contexte :

$$P(w_{i+1}|w_i) = \frac{P(w_{i+1}, w_i)}{P(w_i)} = \frac{P(w_i|w_{i+1}) \cdot P(w_{i+1})}{P(w_i)}$$

- Et de calculer (méthode Bayes Naive)

$$P(\text{"of"} | \text{"tired"}) = 1$$

$$P(\text{"of"} | \text{"use"}) = 1$$

Sémantique (suite)

Sémantique : un outil intéressant (ontologie) :

- **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 frané§ais libre (WOLF) : en dév. (v. beta 2012).

- Ex : voyons si "thé" (tea) est qq chose qu'on peut boire : · · · ↗

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

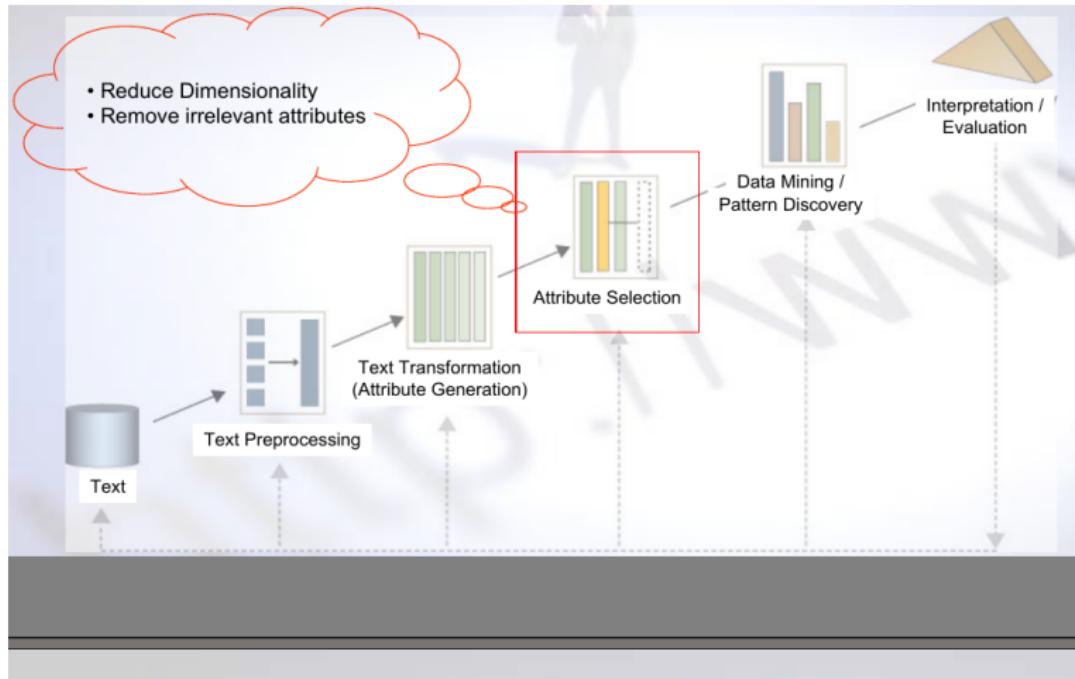
$X \text{ bought } Y$ ou $Y \text{ was acquired by } X,$

on peut passer à :

$\text{company}(X) \wedge \text{company}(Y) \wedge \text{buy_act}(X, Y)$

- 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



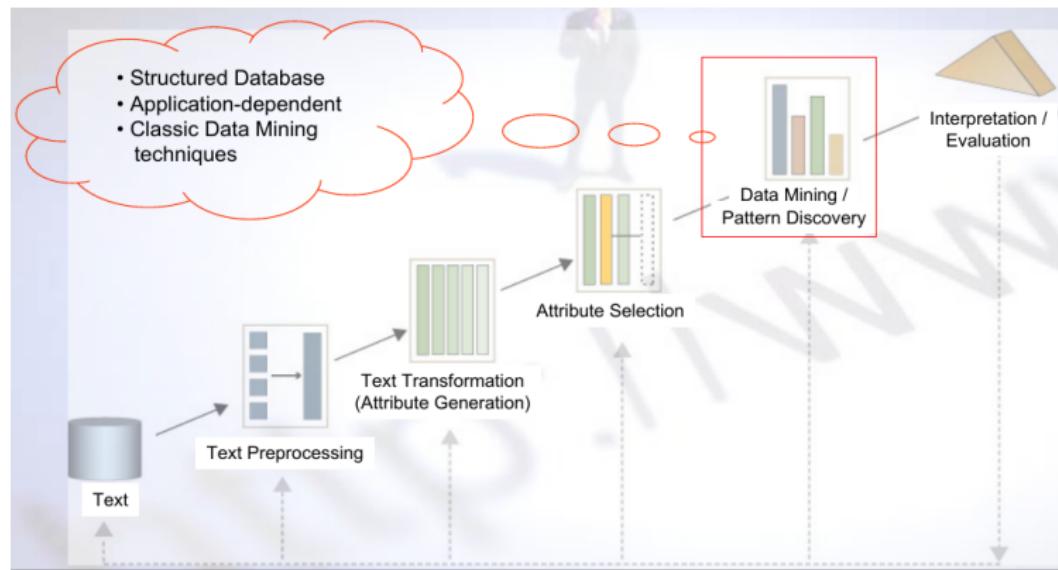
Réduction de Dimension (suite)

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

DM : Text Mining sur les termes retenus (BOW)



Phase de Mining (suite)

- Soit : une série d'étapes ont transformé notre corpus en structure numériques.
 - La suite coincide 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 ce structure 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

- Les docs sont caractérisés par les mots qui les constituent.
- Chaque doc représente une instance d'une classe (*Topic*) de documents.
 - Par ex, la classe des (journaux) d'info, de sports, de spam, ...
- On pourrait utiliser une méthode basique et triviale : traiter la présence/absence d'un mot, puis de décider sur ces simples fréquences des mots ;
- **Plus efficace** (et rapide) : la méthodes Bayésienne Naïve
 - On tiendra compte des fréquences des mots.
 - Pour ce faire, on applique une forme modifiée de Bayes naïve :
multi-nominal Naïve Bayes (MNB).
 - Dans **MNB**, les docs sont considérés comme des *sacs-de-mots* :
Un doc : un ensemble pouvant contenir plusieurs fois le même mot.
- Le but : donner la classe le plus probable d'un (nouveau) document

Classification de documents avec Bayes (suite)

- MNB s'appuie sur une distribution *multinomiale* pour la classification.
- Pour cette distribution, la probabilité qu'un doc E (composé de n_E mots clefs, voir ci-dessous) soit d'une classe H est :

$$Pr[H|E] \propto Pr(H) \prod_{j=1}^{n_E} Pr[w_j|H]$$

- w_j : les mots (clefs du dico) des documents de la catégorie H
- $Pr[w_j|H]$ est la proba que w_j figure dans les docs de la catégorie H
= combien w_j contribue à ce que H soit (la vraie) classe de E .
- $Pr(H)$: probabilité *a priori* qu'un doc soit de la catégorie H .
- La meilleure classe H pour E est celle la plus **vraisemblable** = celle avec une probabilité *a posteriori maximum* ($MAP = \text{maximum a posteriori}$).

Classification de documents avec Bayes (suite)

- $w_1, \dots w_j \dots w_{n_E}$ sont les mots clefs dans E
 n_E = nombre de ces mots dans E
- P.ex, $w_1, \dots w_j \dots w_{n_E}$ pour un doc avec une seule phrase :

"ECL et EML sont ensemble dans un Bateau"

sera $\langle ECL, EML, ensemble, Bateau \rangle$ avec $n_E=4$ (une fois tokenisé)

- Notons par \hat{Pr} une estimation de Pr (à partir d'une base d'apprentissage).
- Utiliser \log pour ne pas perdre de la précision dans les multiplications.
- On choisira le maximum de $\log(\hat{Pr}[H|E]) \propto \log(\hat{Pr}(H)) + \sum_{j=1}^{n_E} \log(\hat{Pr}[w_j|H])$
 - Cette somme indique "combien" le doc. E peut être de la classe H .
 - $\log(\hat{Pr}[w_j|H])$ donne la valeur (la contribution) de w_j pour désigner la classe H
 - $\log(\hat{Pr}(H))$ indique la fréquence relative de la classe H :
 - plus la classe est fréquente, plus elle a la chance d'être choisie.

Classification de documents avec Bayes (suite)

Estimation de $\hat{Pr}(H)$ et $\hat{Pr}[w|H]$:

- On utilisera MLE (maximum de vraisemblance) est ici simplement la fréquence relative dans la base d'apprentissage :

$$\bullet \hat{Pr}(H) = \frac{N_H}{N} = \frac{\text{nombre de documents dans la classe } H}{\text{le nombre total des documents du corpus}}$$

- L'estimation pour $\hat{Pr}[w_j|H]$ est la fréquence relative du mot w_j dans les documents de la classe H .

$$\rightarrow \hat{Pr}[w|H] = \frac{T_{Hw}}{\sum_{w' \in V} T_{Hw'}}.$$

- T_{Hw} est le total de toutes les occurrences du mot w dans la base d'apprentissage (dans le vocabulaire des *mots clefs* V).

Classification de documents avec Bayes (suite)

- Rappel : on est indép de la position des mots dans les docs (pas d'ordre)
 - On ne calcule donc pas différentes estimations pour différentes positions
 - Si un mot apparaît 2 fois, alors $\hat{Pr}[w|H]$ sera identique pour les 2 occ.

→ P. ex. les documents $\{Ecully\ Dardilly\ Ecully\}$ et $\{Ecully\ Ecully\ Dardilly\}$ sont considérés identiques et les mots répétés ont le même poids.
- Pour le problème du *veto de zéro* : *Lissage de Laplace*

$$\hat{Pr}[w|H] = \frac{T_{Hw} + 1}{\sum_{w' \in V} (T_{Hw'} + 1)} = \frac{T_{Hw} + 1}{(\sum_{w' \in V} T_{Hw'}) + B}$$

où B est la constante du lissage de Laplace = ici la taille du vocabulaire.

- L'ajout de 1 signifie une *a priori* uniforme (comme si chaque mot apparaissait une seule fois dans chaque classe).
- Il est évidemment possible de généraliser ce lissage (cf. Bayes).

Classification de documents avec Bayes (suite)

Un exemple : soit les documents

ID	Ensemble	les mots du document	classe $H = "Chine"$?
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 a également, pour $H = "Chine"$, $\hat{Pr}(H) = 0,75$ et $\hat{Pr}(\bar{H}) = 0,25$
- Calcul des probabilités conditionnelles :

$$\hat{Pr}("Chinois" | H) = \frac{5 + 1}{8 + 6} = \frac{3}{7}$$

$$\hat{Pr}("Tokyo" | H) = \hat{Pr}("Japon" | H) = \frac{0 + 1}{8 + 6} = \frac{1}{14}$$

$$\hat{Pr}("Chinois" | \bar{H}) = \frac{1 + 1}{3 + 6} = \frac{2}{9}$$

$$\hat{Pr}("Tokyo" | \bar{H}) = \hat{Pr}("Japon" | \bar{H}) = \frac{1 + 1}{3 + 6} = \frac{2}{9}$$

- On utilise les dénominateurs $(8 + 6)$ et $(3 + 6)$ car la longueur des textes de la classe $H = "Chine" = 8$ et celle de $\bar{H} = 3$ et
- La constante B du lissage de Laplace = 6 (= nb. termes dans le vocab.)

Classification de documents avec Bayes (suite)

- On aura $\hat{Pr}(\text{"Chine"} | d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0,0003$

Et $\hat{Pr}(\overline{\text{"Chine"}} | d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0,0001$

- NB : pour passer de \approx à $=$, on normalise en divisant ces valeurs par leur somme :

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

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

- **Conclusion** : le document de test (d_5) appartient à la classe $H = \text{"Chine"}$.

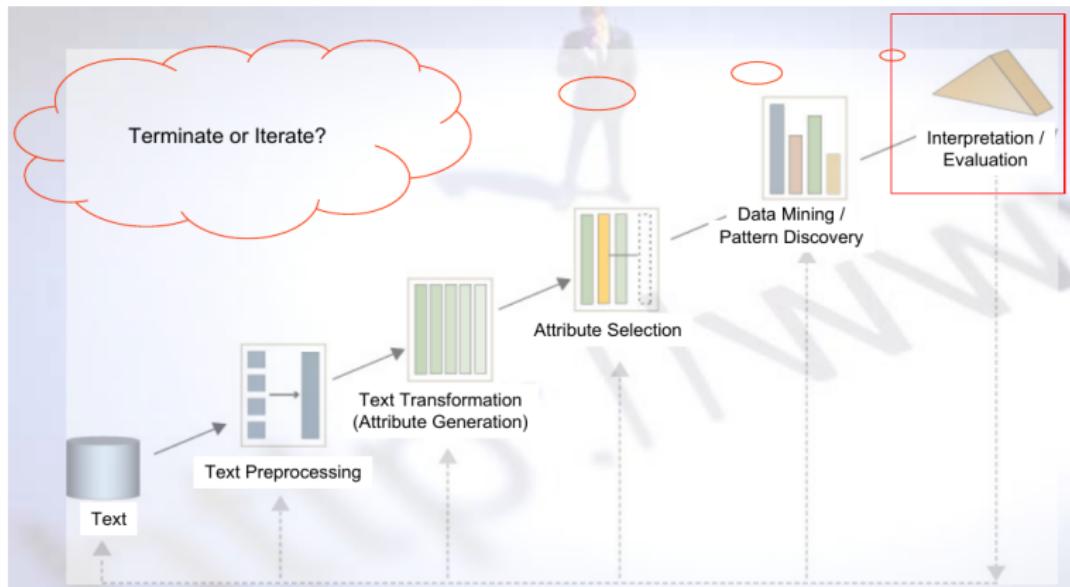
→ Ici, les 3 occ de l'indicateur positif ("Chinois") dans d_5 prennent le dessus sur les 2 occ des indicateurs négatifs ("Japon" et "Tokyo").

☞ 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)

Quelques rappels :

- On demande à un bon modèle de ne pas être juste bon sur un (seul ?) ensemble d'apprentissage mais sur tous les ensembles concernés (qu'on appellera Ls : learning sets).

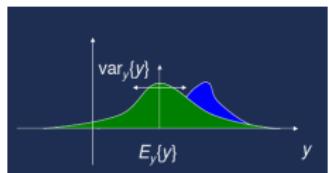
- L'expression de l'erreur : $E = \mathbb{E}_y(y - \hat{y})^2 = \mathbb{E}_{Ls}\{\mathbb{E}_y\{(y - \hat{y})^2\}\}$ que l'on développe :

$$E = \mathbb{E}_y\{(y - \mathbb{E}_y\{y\})^2\} + \mathbb{E}_{Ls}\{(\mathbb{E}_y\{y\} - \hat{y})^2\}$$

Le premier terme est l'erreur résiduelle

$$\text{var}_y\{y\} = \varepsilon$$

qui est le minimum atteignable de (toute) l'erreur E

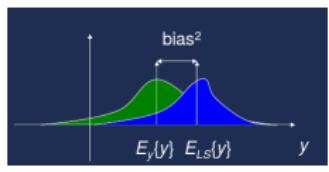


- Si on développe le 2e terme de E ci-dessus

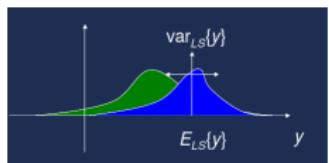
$(\mathbb{E}_{Ls}\{(\mathbb{E}_y\{y\} - \hat{y})^2\})$, on obtient :

$$(\mathbb{E}_y\{y\} - \mathbb{E}_{Ls}\{\hat{y}\})^2 + \mathbb{E}_{Ls}\{(\hat{y} - \mathbb{E}_{Ls}\{\hat{y}\})^2\}$$

dont la 1e composante est le **bias²** = l'erreur entre le modèle obtenu (ML) et le modèle moyen (connu).

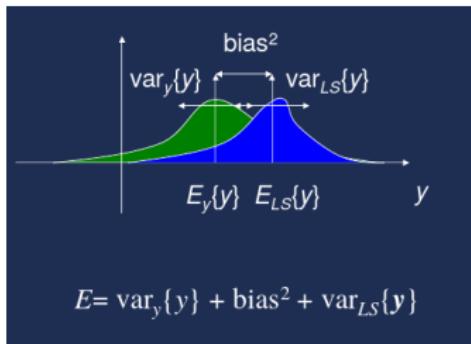


- Enfin le terme $\mathbb{E}_{Ls}\{(\hat{y} - \mathbb{E}_{Ls}\{\hat{y}\})^2\} = \text{var}_{Ls}\{\hat{y}\}$
 $= \text{var}_{Ls}\{\hat{y}\} = \text{var}_{Ls}\{y\}$ (sachant $\mathbb{E}_y\{y\} = \mathbb{E}_y\{\hat{y}\}$)
 $=$ une estimation de la variance
 $=$ une mesure de l'Overfitting éventuel.



- Par conséquent : $E = \varepsilon + \text{bias}^2 + \text{var}_{Ls}(y)$

Evaluation (suite)



Notons : la variance $\varepsilon = \text{var}_y(y)$ est le **bruit** (noise, Vars. cachées, ...),
 bias^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) :

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

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

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

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

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

— également **MAP** = *Mean of Average Precision*

- Si AP (*Average Precision*) est la moyenne des valeurs de P (**Precision**) aux pourcentages où on a trouvé un document pertinent,
alors $MAP =$ la moyenne de ces APs .

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

Evaluation (suite)

Comparaison des mesures :

- Soit R_1 et R_2 deux mesures d'évaluation.
 - R_1 peut être la sortie d'un système IR et R_2 la sortie (attendue) par un user.
 - Elles peuvent être 2 mesures statistiques d'évaluation, etc.
 - Elles peuvent être p. ex. **Recall** et **Precision** (ci-dessous) !

- On évalue l'accord (*agreement*) entre ces deux "R"s :

- Mesures souvent utilisées : (dont deux des celles dites "mesures RR" :)

— $Jaccard = P(similarity) = \frac{|R_1 \wedge R_2|}{|R_1 \vee R_2|}$ les barres : card/fréq

→ Sur des vecteurs, on peut exprimer cette mesure de corrélation par

$$Jaccard = \frac{\sum_i (R_1 \wedge R_2)}{\sum_i R_1 + \sum_i R_2 - \sum_i (R_1 \wedge R_2)}$$
 sans hyp. d'indép entre les 2

— $\text{cosine} = P(similarity) = \frac{\sum_i (R_1 \cdot R_2)}{\sqrt{\sum_i R_1^2} * \sqrt{\sum_i R_2^2}}$ $\begin{matrix} \leftarrow \text{dot prod. si vecteurs} \\ \leftarrow \text{Produit des normes} \end{matrix}$

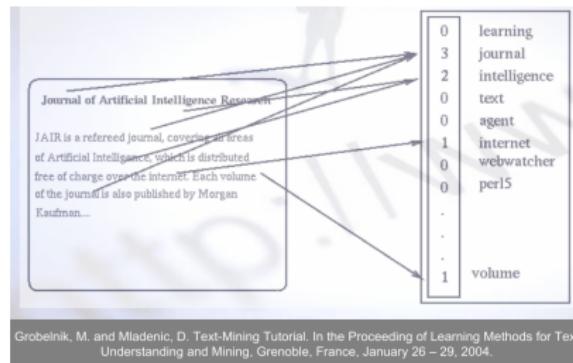
— beaucoup d'autres mesures et méthodes de comparaison...

- ☞ Le réZle de l'expert (si disponible).

Introduction aux Techniques IR

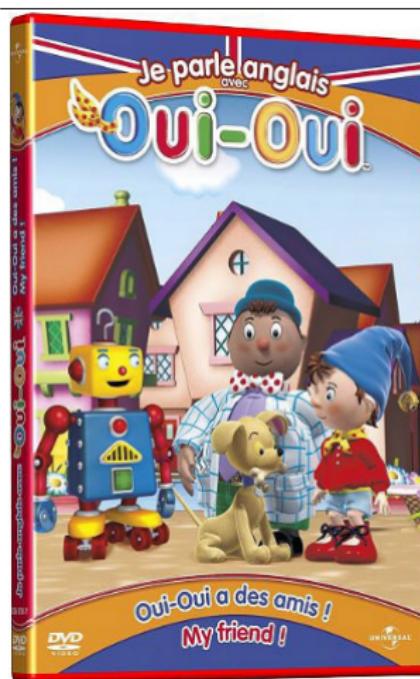
- Deux approches utilisées :
 - Sac de termes / mots ("bag of words" = BOW) : plus classique
 - Vecteur de caractéristiques (Vector space)

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

Remember : 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 ₁	...	w _j	...	w _J	
d ₁							
...				...			
d _i		...		n(d _i , w _j)		...	
...				...			
d _J							

FIGURE 6.1 – co-occurrence matrix

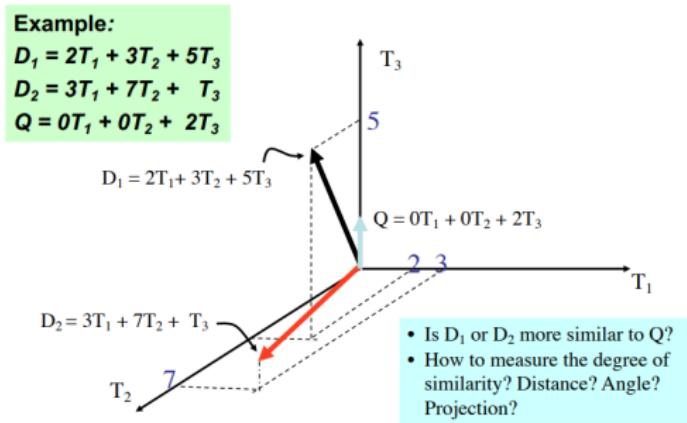
Vector space (suite)

- 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)

- 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 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"											
D = 3; IDF = log(D/df _i)											
		Counts, tf _i						Weights, w _i = tf _i *IDF _i			
Terms	0	D ₁	D ₂	D ₃	df _i	D/df _i	IDF _i	0	D ₁	D ₂	D ₃
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 :

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

- 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 ...

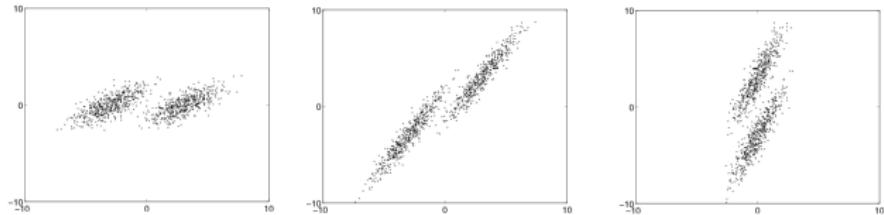
LSA / LSI and its computational tools

- 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

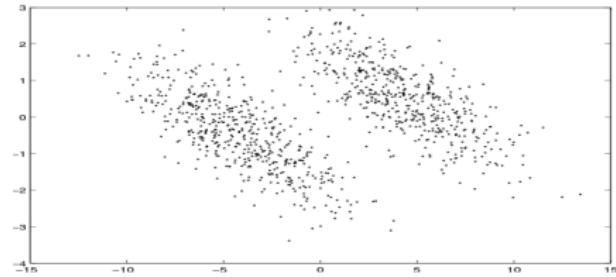
LSA / LSI and its computational tools (suite)

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

Complément : from PCA to SVD

- PCA tries to fit an n-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a **principal component**.
 - ☞ If some axis of the ellipse is small, then the **variance** along that axis is also small, and by omitting that axis and its corresponding principal component from our dataset, we lose only a small amount of info.

How PCA works :

- For $x_1, \dots, x_n \in \mathbb{R}^p$ a set of *centered* (with $\mu_{w_i} = 0$) points, find a k-dimensional subspace with the least loss of information.
- PCA finds a $n \times k$ (rather than $n \times p$) **linear projection matrix** V_k so that the sum of squared dist. from the data pts. to their proj. is minimized :

$$\text{Loss cost} : \mathfrak{L}(V_k) = \sum_{i=1}^n \|x_i - V_k V_k^T x_i\|^2 \quad x_i \text{ is a doc. for us}$$

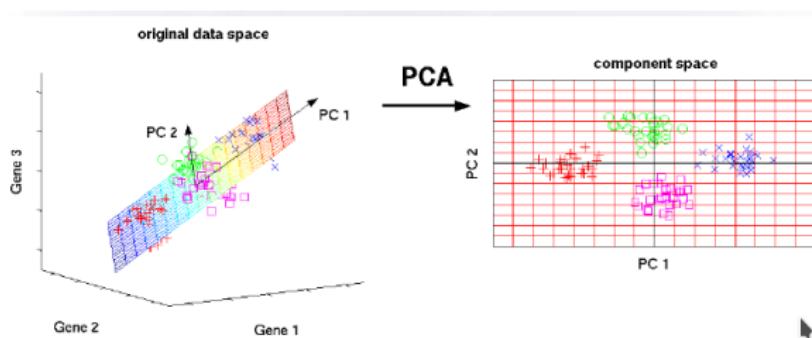
$V_k^T x_i$: **projection of x_i** in the k-subspace spanned by the column vectors of V_k

$V_k V_k^T x_i$: the (new) repr. in the original p-space of the projected vector $V_k^T x_i$

- ☞ PCA supports only sum-square distance, not *Cosine*, etc. (convergence issue)

Complément : from PCA to SVD (suite)

Example : PCA and Bioinformatics



- A 3-dimensional gene expression data mainly located within a 2-dimensional subspace.
- The **three original variables** (genes) are reduced to a lower number of **two new variables** termed **principal components** (PCs).
- Using PCA, we can identify the two-dimensional plane that optimally describes the highest variance of the data.
- This two-dimensional subspace can then be rotated and presented as a two-dimensional component space (right).
- This will allow us to draw qualitative conclusions about the **separability** of experimental conditions (marked by different colors).

Complément : from PCA to SVD (suite)

In PCA (Cont.) :

- We have to minimize the **cost** $\mathfrak{L}(V_k)$:

$$\mathfrak{L}(V_k) = \sum_{i=1}^n \|x_i - V_k V_k^T x_i\|^2 = \sum_{i=1}^n \|x_i\|^2 - \sum_{i=1}^n \|V_k V_k^T x_i\|^2 \quad \text{where}$$

$\sum_{i=1}^n \|V_k V_k^T x_i\|^2$: the empirical variance of the projections.

and $V_k^T V_k = Id \Leftrightarrow V_k$ is orthogonal

☞ The projection matrix V_k that minimizes $\mathfrak{L}(V_k)$ is the one that maximizes the variance in the projected space.

- The solution to V_k can be computed by SVD.
- We can also use SVD directly.

Addendum : PCA details, PLSI

why $\mathfrak{L}(V_k) = \sum_{i=1}^n \|x_i - V_k V_k^T x_i\|^2 = \sum_{i=1}^n \|x_i\|^2 - \sum_{i=1}^n \|V_k V_k^T x_i\|^2 ?$

$$\begin{aligned}
 & \Rightarrow \|x_i - V_k V_k^T x_i\|^2 = \langle x_i - V_k V_k^T x_i, x_i - V_k V_k^T x_i \rangle \\
 & = \|x_i\|^2 - 2 \langle x_i, V_k V_k^T x_i \rangle + \|V_k V_k^T x_i\|^2 \\
 & = \|x_i\|^2 - 2 \langle V_k^T x_i, V_k^T x_i \rangle + \|V_k V_k^T x_i\|^2 && \text{since } \langle A, B \rangle = \langle CA, CB \rangle \text{ if } C^T C = Id \\
 & = \|x_i\|^2 - 2\|V_k^T x_i\|^2 + \langle V_k V_k^T x_i, V_k V_k^T x_i \rangle && \text{And } V_k^T V_k = Id \\
 & = \|x_i\|^2 - 2\|V_k^T x_i\|^2 + \langle V_k^T V_k V_k^T x_i, V_k^T x_i \rangle && \text{same as above} \\
 & = \|x_i\|^2 - 2\|V_k^T x_i\|^2 + \langle V_k^T x_i, V_k^T x_i \rangle && \Leftrightarrow V_k^T V_k = Id \\
 & = \|x_i\|^2 - 2\|V_k^T x_i\|^2 + \|V_k^T x_i\|^2 && \Leftrightarrow \text{But } V_k V_k^T \neq Id \text{ otherwise there is no loose !} \\
 & = \|x_i\|^2 - \|V_k^T x_i\|^2 = \|x_i\|^2 - \|V_k V_k^T x_i\|^2 && \text{We had } \|V_k V_k^T x_i\|^2 = \|V_k^T x_i\|^2 \text{ (from lines 3 to 7)}
 \end{aligned}$$

- This shows that $\|V_k V_k^T x_i\| = \|V_k^T x_i\|$
 - But we better use $\|V_k V_k^T x_i\|$ to handle the cost function.
- Note that for any matrix X , $XX^T = \|X\|^2$ is symmetric

Addendum : PCA details, PLSI (suite)

More details about PCA :

- To find the axes of the ellipse in **PCA**, first **center** the data around the origin (i.e. subtract the mean of each variable from the dataset).

Then compute the covariance matrix of the data, and calculate the eigenvalues and corresponding eigenvectors of this covariance matrix.

Then, orthogonalize the set of eigenvectors, and normalize each to become unit vectors.

Once this is done, each of the mutually orthogonal, unit eigenvectors can be interpreted as an axis of the ellipsoid fitted to the data.

The **proportion of the variance** that each eigenvector represents can be calculated by dividing the eigenvalue corresponding to that eigenvector by the sum of all eigenvalues.

- ☞ Note that PCA is sensitive to the scaling of the data : this is a **limitation** of PCA
 - there is no consensus as to how to best scale the data to obtain optimal results.
- **Mathematically**, PCA is defined as an orthogonal linear transformation of the data to a new coordinate system such that the **greatest variance** by some projection of the data comes to lie on the first coordinate (called the *first principal component* : PC1), the second greatest variance on the second coordinate, and so on.
 - The principal components are orthogonal because they are the eigenvectors of the covariance matrix, which is symmetric.
- PCA is computed by using the **correlation** or the **covariance** matrix method

Addendum : PCA details, PLSI (suite)

Towards PLS (and NIPALS) :

A PCA improvement : the NIPALS method (Non-linear iterative partial least squares)

- For very-high-dimensional datasets (e.g., genomics, metabolomics) it is usually only necessary to compute the first few PCs.

The non-linear iterative partial least squares (NIPALS) algorithm calculates the first components from the data set.

Then this component is **subtracted** from the data set leaving a **residual** matrix.

This can be then used to calculate subsequent PCs.

This results in a real reduction in computational time since calculation of the covariance matrix is avoided.

- However, for **large data matrices**, or matrices that have a high degree of column collinearity, NIPALS suffers from loss of orthogonality due to machine precision limitations accumulated in each iteration step.
- A Gram-Schmidt (GS) re-orthogonalization algorithm is applied to both the scores and the loadings at each iteration step to eliminate this loss of orthogonality.
- For an online/sequential (streaming) estimation, a different method must be applied.

Complement : SVD

The aim of SVD is to find a diagonal matrix, to reduce dimensionality of the original co-occurrence matrix (np) to (nk) and use it instead of the original matrix (the data set).

- SVD : given the matrix A with $x_1, \dots, x_n \in \mathbb{R}^p$

Let the SVD of matrix A be given by $A = USV^T$ with

$$U = [u_1, u_2, \dots, u_p],$$

$$S = \text{Diag}[s_1, s_2, \dots, s_p],$$

$$V = [v_1, v_2, \dots, v_p]$$

and $\text{rank}(A) = r$.

Matrix A_k defined by $A_k = \sum_{i=1}^k u_i s_i v_i^T$ is the closest rank- k matrix to A in term of Euclidian (Frobenious) norm.

- For k given, the matrix V_k (in k -subspace and composed of the first k columns of V) constitutes the **rank k solution to $\mathcal{L}(V_k)$** .

Complement : SVD (suite)

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

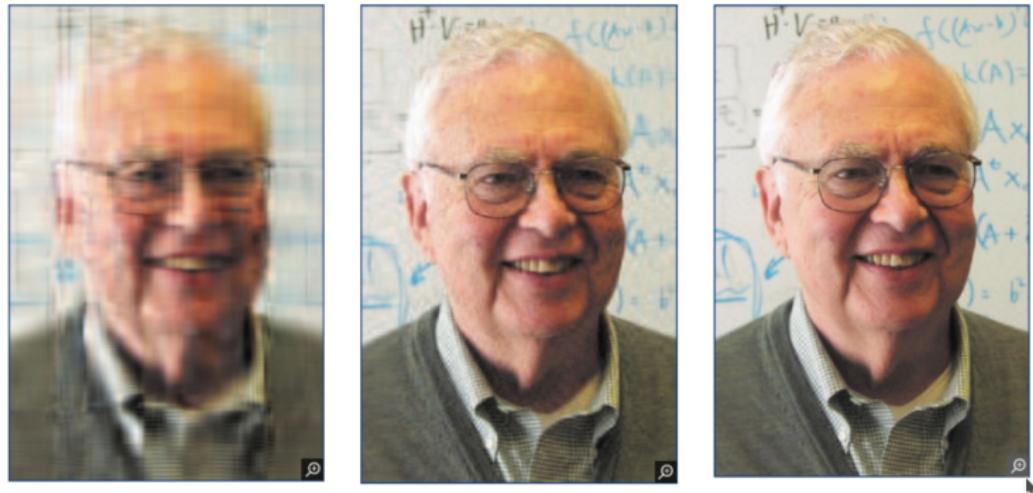
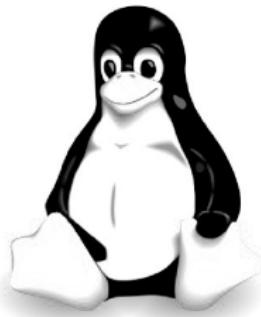


FIGURE 6.2 – Rank 12, 50, and 120 approximations to a 598 color photo of *Gene Golub* (a CS. prof at stanford)

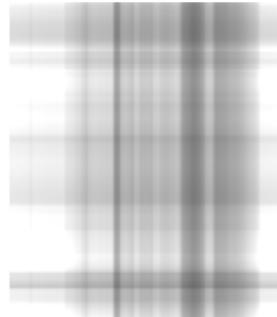
Complement : SVD (suite)

- Another Image : SVD with Tux (See

https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Dimensionality_Reduction/Singular_Value_Decomposition)



Original



With one SV



with 35 SV

Complement : SVD (suite)

- Main contributions of SVD ($A = USV^T$) :
 - Reducing dimension of A : the k -largest singular values (diagonal of S) approximates the original matrix A .
 - Discarding small singular values \equiv discarding semi-dependent, noisy, trivial variation in the data set
 $(\equiv$ discard non-essential axes of the original feature space $)$
 - Emphasizes the salient underlying structure of the data set
 - Data points with similar features will be mapped **near** to each other in the k -dimensional partial singular vector space.

☞ 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 (SVD application)

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 (SVD application) (suite)

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

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

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

$$\mathbf{A} = \left[\begin{array}{cccc} & \text{d1} & \text{d2} & \text{d3} \\ \text{a} & 1 & 1 & 1 \\ \text{arrived} & 0 & 1 & 1 \\ \text{damaged} & 1 & 0 & 0 \\ \text{delivery} & 0 & 1 & 0 \\ \text{fire} & 1 & 0 & 0 \\ \text{gold} & 1 & 0 & 1 \\ \text{in} & 1 & 1 & 1 \\ \text{of} & 1 & 1 & 1 \\ \text{shipment} & 1 & 0 & 1 \\ \text{silver} & 0 & 2 & 0 \\ \text{truck} & 0 & 1 & 1 \end{array} \right] \quad \mathbf{q} = \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{array} \right]$$

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

LSI : a simple example (SVD application) (suite)

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

$$\begin{array}{c}
 \mathbf{U} \\
 \left[\begin{array}{ccc}
 -0.4201 & -0.075 & -0.046 \\
 -0.2995 & 0.2001 & 0.4078 \\
 -0.1206 & -0.275 & -0.4538 \\
 -0.1576 & 0.3046 & -0.2006 \\
 -0.1206 & -0.275 & -0.4538 \\
 -0.2626 & -0.379 & 0.1547 \\
 -0.4201 & -0.075 & -0.046 \\
 -0.4201 & -0.075 & -0.046 \\
 -0.2626 & -0.379 & 0.1547 \\
 -0.3151 & 0.8093 & -0.4013 \\
 -0.2995 & 0.2001 & 0.4078
 \end{array} \right] \\
 \mathbf{S} \\
 \left[\begin{array}{ccc}
 4.0989 & 0 & 0 \\
 0 & 2.3616 & 0 \\
 0 & 0 & 1.2737
 \end{array} \right] \\
 \mathbf{V}^T \\
 \left[\begin{array}{ccc}
 -0.4945 & -0.6458 & -0.5817 \\
 -0.6492 & 0.7194 & -0.2469 \\
 -0.578 & -0.2556 & 0.775
 \end{array} \right]
 \end{array}$$

LSI : a simple example (SVD application) (suite)

Remember : In $A = USV^T$

- 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.

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

LSI : a simple example (SVD application) (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 (SVD application) (suite)

Step 4 : Find the new term vector coordinates in this reduced 2-dimensional 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**

$$\begin{matrix} \mathbf{S} & & \mathbf{V^T} \\ \left[\begin{matrix} 4.0989 & 0 & 0 \\ 0 & 2.3616 & 0 \\ 0 & 0 & 1.2737 \end{matrix} \right] & & \left[\begin{matrix} -0.4945 & -0.6458 & -0.5817 \\ -0.6492 & 0.7194 & -0.2469 \\ -0.578 & -0.2556 & 0.775 \end{matrix} \right] \end{matrix}$$

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

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

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

LSI : a simple example (SVD application) (suite)

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

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

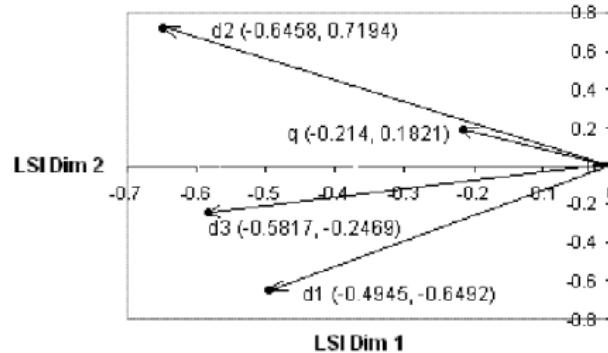
- These are new coordinates of the query in 2-dim space.
- Note the difference between the new representation of **q** (at right) and the original vector of step 1 (at left).

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

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

LSI : a simple example (SVD application) (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"

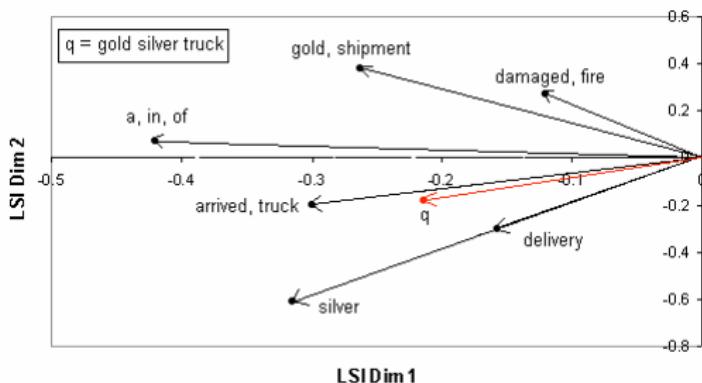
$$q = \begin{bmatrix} -0.2140 & 0.1821 \end{bmatrix}$$

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

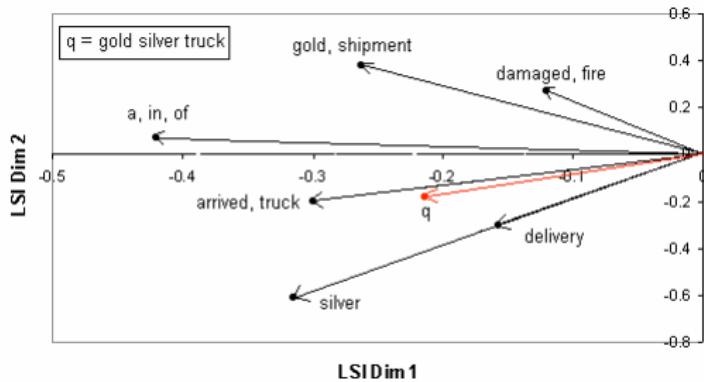
Doing K-means in the reduced space :

→ 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 :

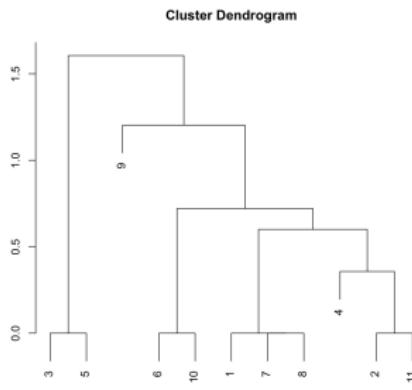
- "gold silver truck *arrived*" "*delivery* gold silver truck"
- "gold silver truck *delivery*"
- "gold silver truck *delivery arrived*" etc ...

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

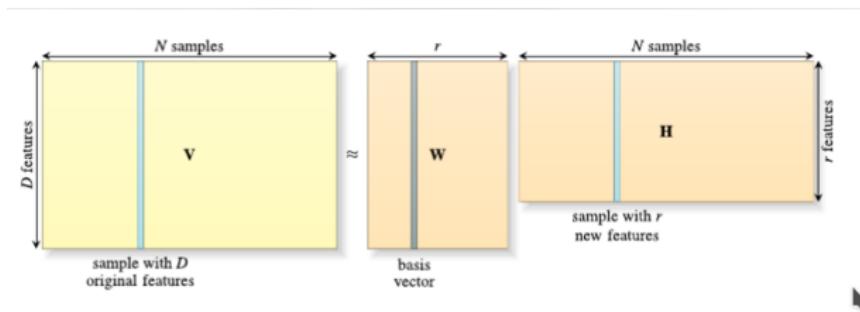
N.B. : we had very few (11) data.

→ Different weights and distances stay close.

Compl. : Non-negative matrix Factorization

Another tool for dimensionality reduction

- Decomposes a matrix (containing only non-negative coefficients) into the product of two other matrices (also composed of non-negative coefficients) :
 - a parts-based matrix (W) and
 - a matrix (H) containing the fractional combinations of the parts that approximate the original data.

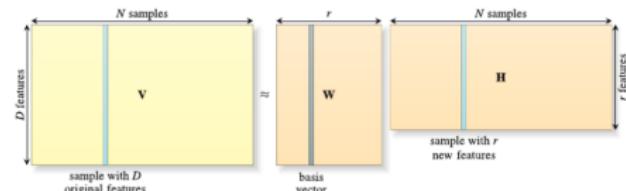


Compl. : Non-negative matrix Factorization (suite)

- Each column of the original matrix V contains a sample with D features.

- NMF is a method for extracting a new set of r **features** from the original data.

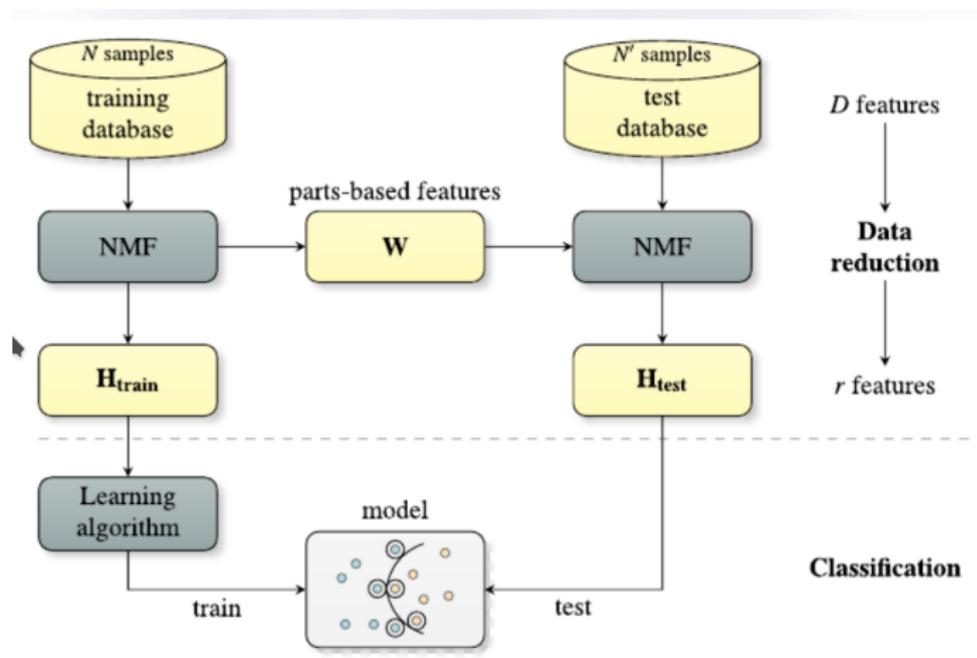
- The parts-based matrix, W will contain the *basis vectors* (one per column), which **define a new set of features** as an additive function of the original ones.



- H will contain the fractional combination of basis vectors that is used to create an **approximation of the original samples** in V .
- Each column of H will contain a sample (mapped to a new r -dimensional space), which results from superposing the (fractional) contribution of each individual *basis vector* that approximates the original sample data
- The original data is reconstructed through additive combinations of the parts-based factorized matrix representation.

Compl. : Non-negative matrix Factorization (suite)

Using NMF :



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).

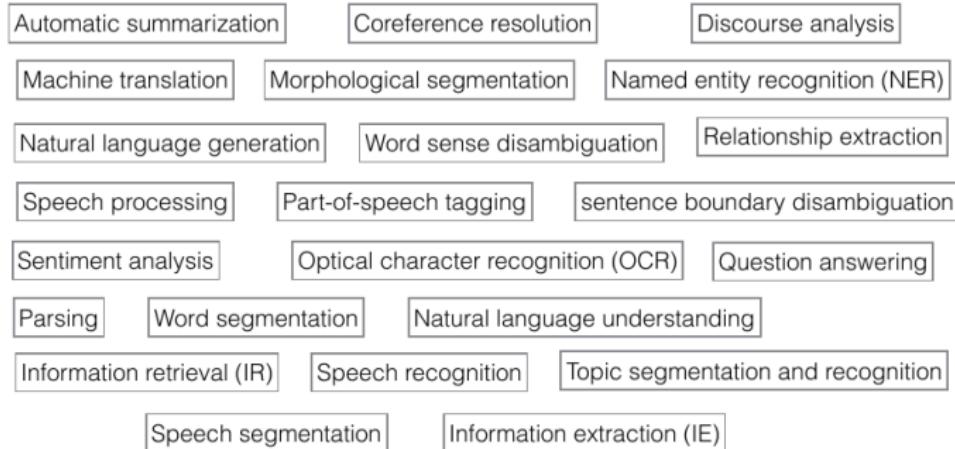
☞ C'est une technique de **feature learning**

- Parmi les méthodes "word embedding" (où l'on utilise un NN) :

- Word2vect : CBOW (continuous BOW) / Skip-Gram
- Glove
- fastText, Etc.

Word Embedding (suite)

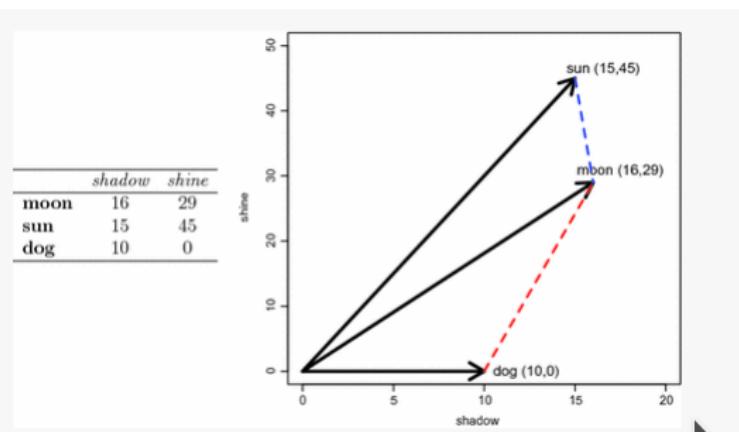
- Deep learning (DNN) a été appliqué avec succès aux domaines suivants (thanks toroelof@kth.se) :



Représentation des mots

- Rappel : l'approche *classique* du TLN utilise les mots et les *modèles* (grammaires, modèles linguistiques) pour le traitement du sens.
- Pour tenir compte du contexte, le sens peut être capturé par une variable latente.
P. Ex. : 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 $MI(x, y) = \log_2 \frac{P(x, y)}{P(x).P(y)}$)

- Un exemple (partiel) :
- la distance entre "dog", "sun" et "moon" représente une variable latente pouvant être interprétée par "éclat" (*shine*) (axe Y, latt.).

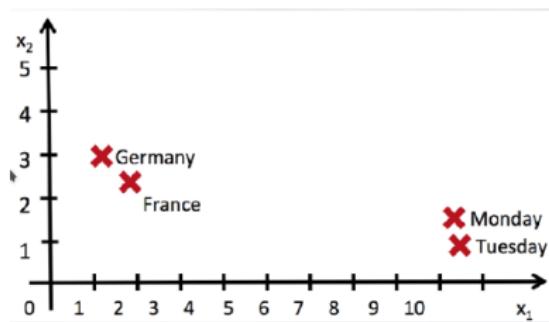


Représentation des mots (suite)

Contextes proches \Rightarrow Sémantiques proches

- On devrait retrouver (un monde idéal) :

(Thanks to Bengio. July, 2012, UCLA)



Comment ? :

mots proches / voisins proches / contextes proches /
 ➔ sémantiques proches (dans un même espace *latent*) ?

Représentation des mots (suite)

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

- 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).
 - L'approche vectorielle LSA / co-occ donne des résultats (cf. ex. préc.).
 - Plus récemment :
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)

☞ On pourrait par exemple avoir :

Kingdom + Lady = Queen !

ou *Kingdom + Man = King*

☞ Mais sur l'exemple "*I love Candy store*", avec

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, ...]

écrire :

Candy AND Store = 0 !

→ pose problème (voir plus loin) !

Représentation des mots (suite)

Utilisation des NN :

- On construit un *vector-space* avec des vecteurs plus denses.
- **Continuous Word Embedding :**
 - Combinaison de vector-space avec la prédiction probabiliste
 - La représentation des mots devient + dense (vect. de taille fixe).
 - P. Ex. le **codage One Hot** de "Candy" est utilisée en entrée du NN et en sortie du NN, il lui sera associé un vecteur tenant compte du voisinage de "Candy".
P. ex. : [0.268, 0.792, -0.177, -0.107, 0.10, -0.542, 0.349, 0.271]
- ☞ Les deux formes de vecteurs seront utiles à *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 de la 2e forme extraite de la matrice de pondérations du NN entraîné (voir plus loin).

Représentation des mots (suite)

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

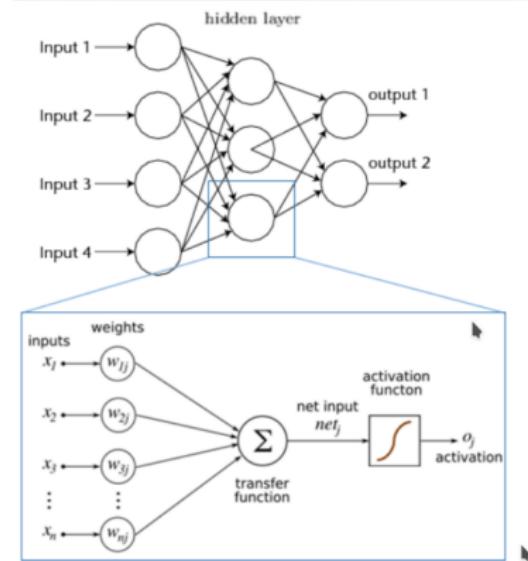
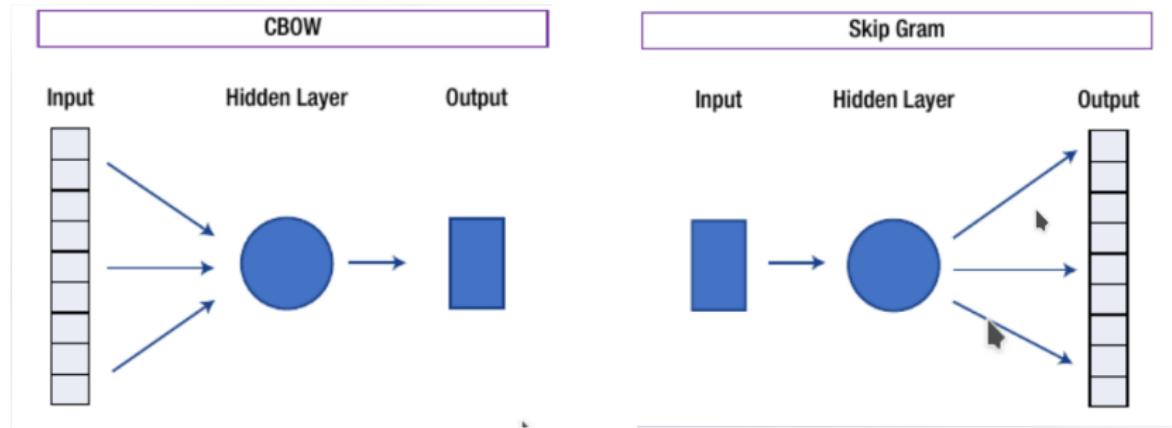


FIGURE 7.1 – Un réseau de Neurones (NN) avec une couche cachée (Deep si nb. H > 1)

Représentation des mots (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

Représentation des mots (suite)

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

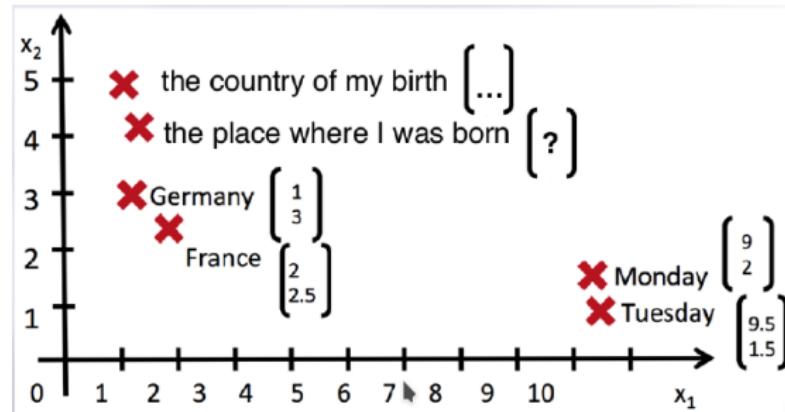


FIGURE 7.2 – Les mots représentées par "Continuous vector-Space" : voir les coordonnées des mots $\in \mathbb{R}$

Intro. Word2vect

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

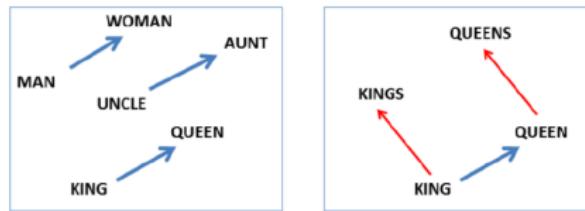


FIGURE 8.1 – Les mots représentées par "Continuous vector-Space" : voir les coordonnées $\in \mathbb{R}$

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

Word2vect : CBOW

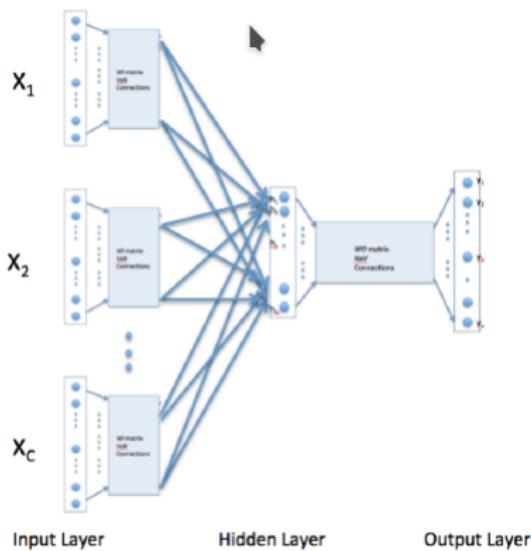
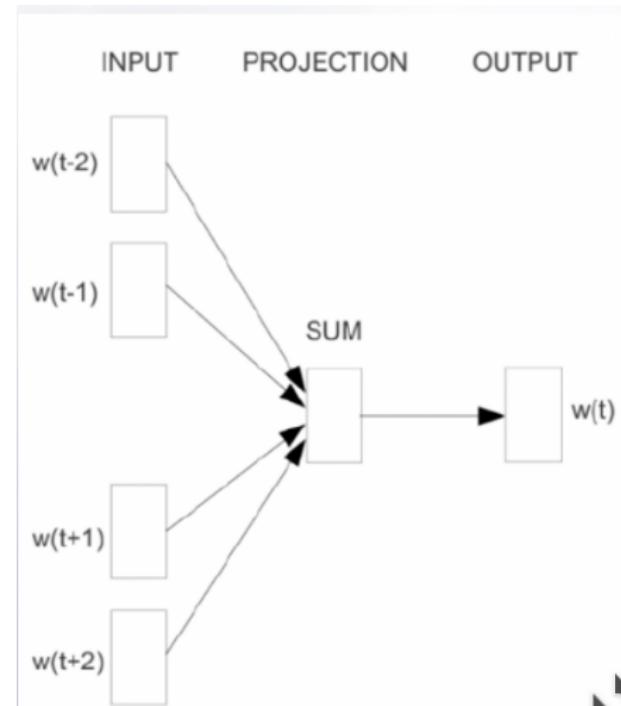


FIGURE 8.2 — CBOW utilise une seule matrice en entrée qui reçoit de multiples vecteurs Hot-One représentant différents mots d'un contexte (voisinage) pour en prédire un autre.

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) :

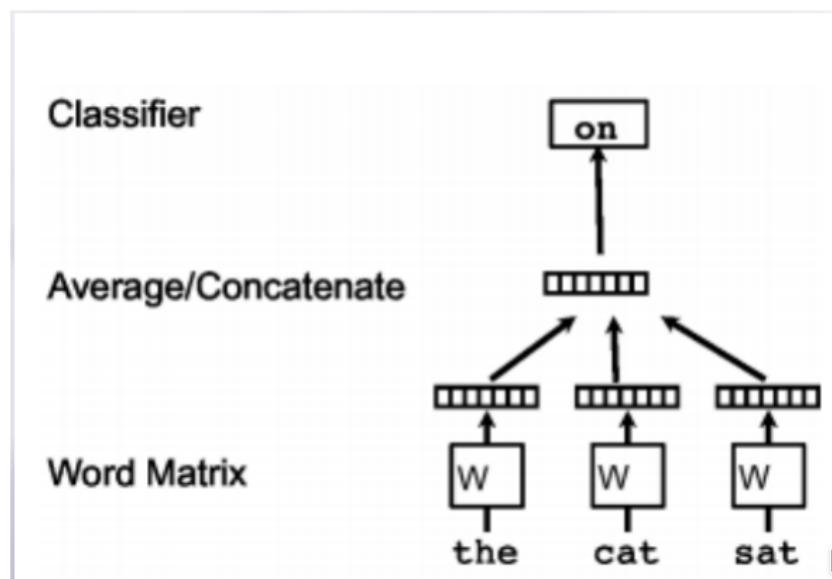


FIGURE 8.3 – CBOW : les mots "the", "cat" et "sat" utilisés pour prédire "on"

Word2vect : Skip-Gram

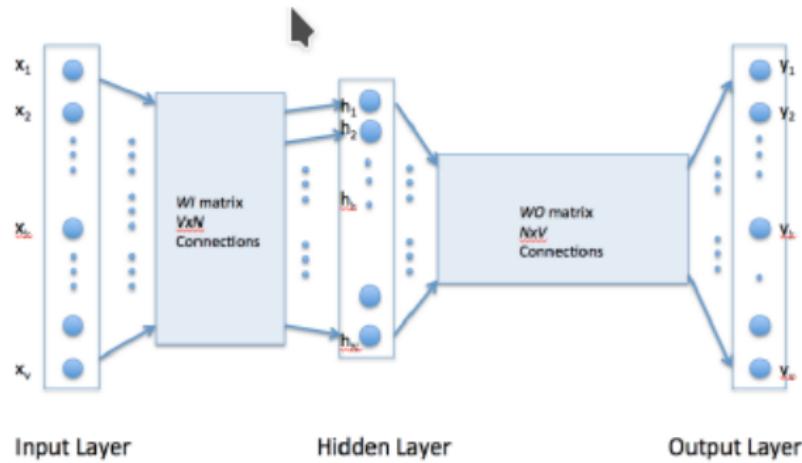
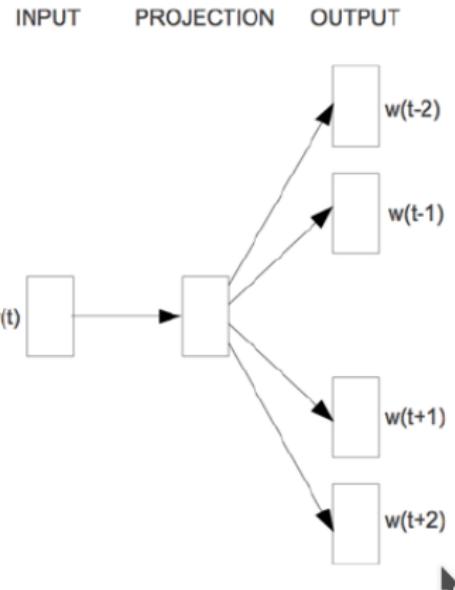


FIGURE 8.4 — Skip-Gram en ses deux matrices de pondération (WI et WO) : on utilise en général WI pour extraire le vecteur d'un mot

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**

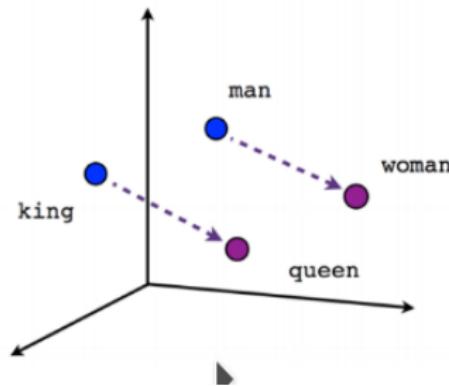


FIGURE 8.5 – Un exemple des mots associés dans l'espace vectoriel créé par *word2vect*

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 quick brown fox jumps over the lazy dog. →	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog. →	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog. →	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog. →	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

☞ Notons les paires de mots.

Word2vect : synthèse

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

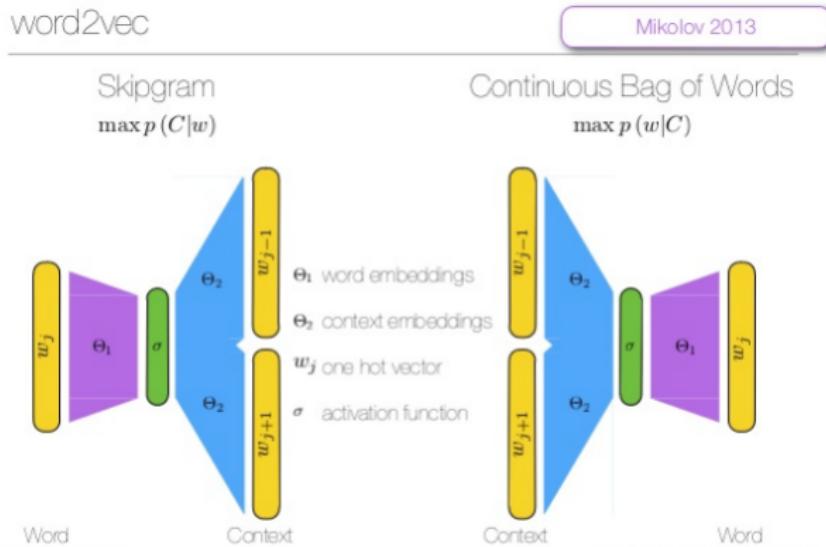


FIGURE 8.6 – Skip-Gram vs. CBOW en un coup d'oeil (merci à Bruno Gonçalves)

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')

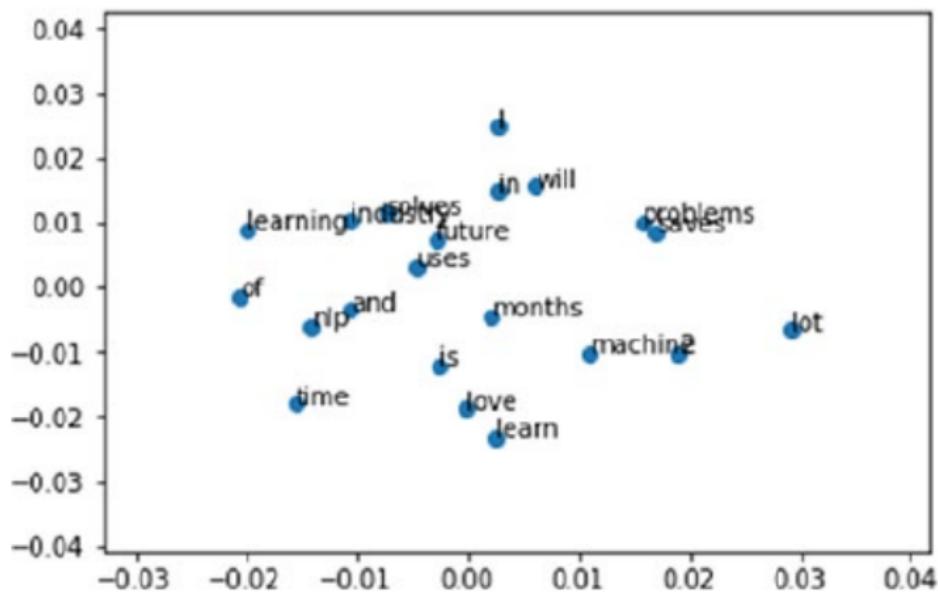
X = skipgram[skipgram.wv.vocab]

# 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

Le fonctionnement du NN (point de vue probabiliste) :

étant donné un mot "centrale" w_c (input word pour le NN) d'une phrase, on envisage aléatoirement un mot w_v au "voisinage" de w_c et le NN nous dira la probabilité pour N'IMPORTE quel mot du vocabulaire d'être le w_v que nous avons choisi.

→ "voisinage" : les mots dans une fenêtre d'une certaine taille (dans la pratique, 5 mots voisins à gauche et 5 à droite) autour de w_c .

- Pour cela, le NN est entraîné par des paires de vecteurs de mots.

skip-gram : le fonctionnement du NN (suite)

- En fait, le NN va apprendre des statistiques : de combien et comment (contexte) ces paires ont figuré ensemble.

Par exemple : si le NN est entraîné pour un input = "Bordeaux", les probas en sortie seront plus élevées pour les mots tels que "Gironde" et "Médoc" par rapport aux mots sans rapport tels que "Choucroute", "Sasquatch" ou "wampimuk" (qui ne se seraient pas retrouvés avec "Bordeaux", "Médoc" ou "Gironde" dans des phrases.)

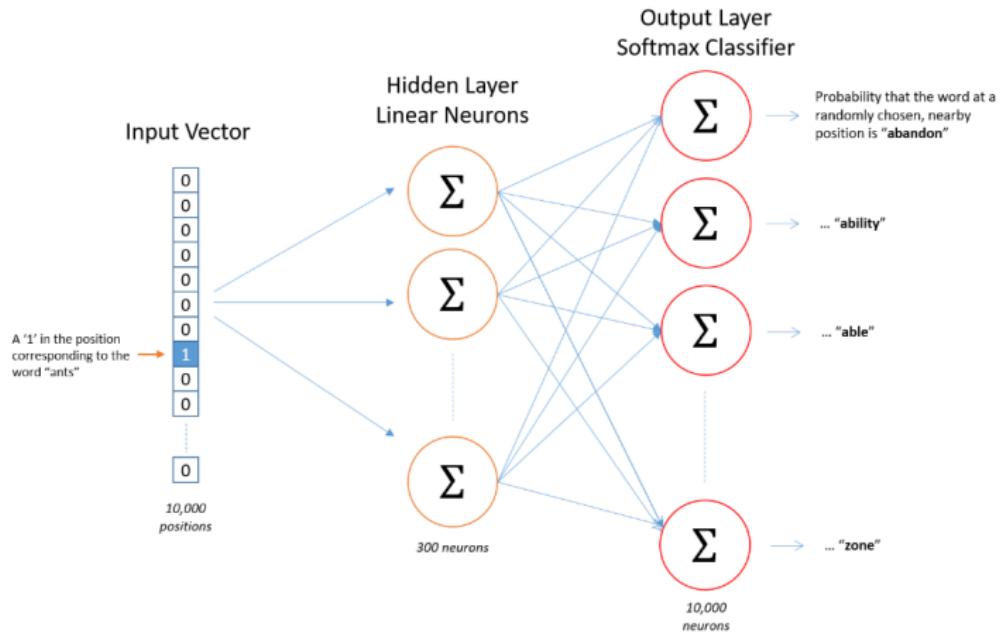
- Dans notre exemple, le NN aura croisé souvent la paire ("Bordeaux", "Gironde") que ("Bordeaux", "wampimuk").
La probabilité pour "Gironde" sera plus élevée que pour "wampimuk" de se trouver au voisinage de "Bordeaux".
- Quelles sont les entrées de ce NN ?/..

skip-gram : le fonctionnement du NN (suite)

Codage One-Hot des mots présentés au NN :

- Supposons avoir un vocabulaire de taille 10000 (mots uniques, sans *stemming* pour simplifier) dont P. ex. le mot "ants".
- Les mots numérisés par **One-hot coding** : un vecteur par mot
 - Ce vecteur aura 10000 composants (un par mot dans le vocabulaire) avec un "1" dans la case correspondant à l'indexe de "ants" (mots triés dans le vocabulaire) et 0 ailleurs (9999 zéros).
- La sortie du NN est un seul vecteur de 10000 positions contenant, pour chaque mot w du vocabulaire, la proba pour que un mot aléatoire du vocabulaire proche de "ants" soit w .

skip-gram : le fonctionnement du NN (suite)



→ 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).
 - On présente le premier en input, le 2e en output (le mot attendu).
 - Le NN affiche des probas (cf. SoftMax) et non une sortie one-hot,
 - Dans le cas idéal, il faudrait avoir une proba "1" dans la même case du vecteur one-hot du mot attendu.
- ☞ N.B. : On remarque ici pourquoi on a une représentation one-hot
 - Si on utilise un codage (p. ex.) *grey*, comment rattacher la probabilité au mot attendu ?

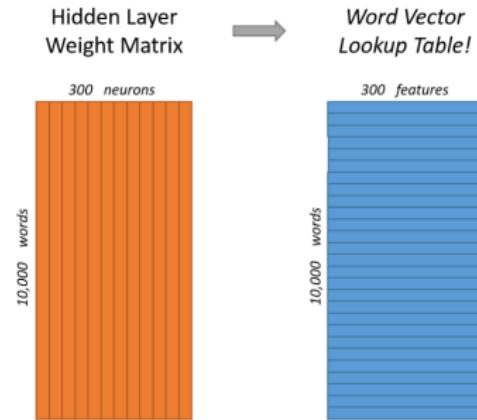
skip-gram : le fonctionnement du NN (suite)

L'importance de la couche cachée :

- Supposons décider d'avoir une couche cachée de 300 neurones.
(compression : de 10000, on passe à 300).
- On aura donc une matrice de pondérations de 10000×300 entre Input et Hidden.
 - Rappel : chaque mot est représenté par un vecteur de 10000 booléens.
 - 10000 : une ligne de la matrice par mot du vocabulaire.
- 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)

- Si on regarde cette matrice de pondération



- La matrice des pondérations (entre Input et Hidden) se comporte comme *indice dans une table* et chacune de ses lignes est justement le "vecteur appris" pour le présenté en Input.
- Le but final de cet apprentissage est seulement la matrice de gauche.
Le output est délaissée à la fin.

skip-gram : le fonctionnement du NN (suite)

L'intérêt du codage one-hot

- A priori, il y a beaucoup de zéros et on pourrait faire plus efficace.
- Si on multiplie un vecteur one-hot de taille 10000 par une matrice de 10000×300 , on choisit effectivement la ligne de la matrice correspondant au "1" dans le vecteur :

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \ 12 \ 19]$$

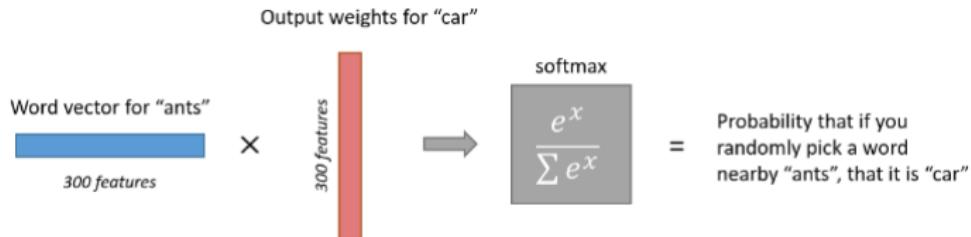
- Appelons cette matrice de pondérations $W1$
- ☞ A propos de l'entrée "biais" (dans les NN) ...

skip-gram : le fonctionnement du NN (suite)

La couche output :

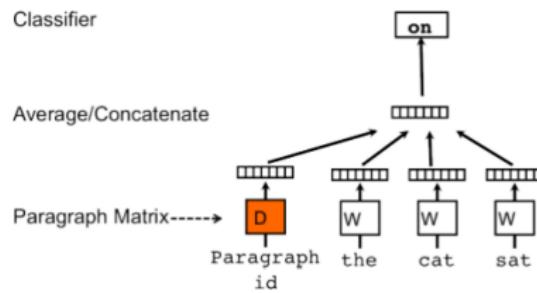
- De l'exemple ci-dessus, le vecteur 1×10000 pour "ants" arrive à la couche output.
→ Softmax produira ensuite une probabilité (de somme 1!)
- Exemple d'utilisation : "car" vs. "ants" :

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.

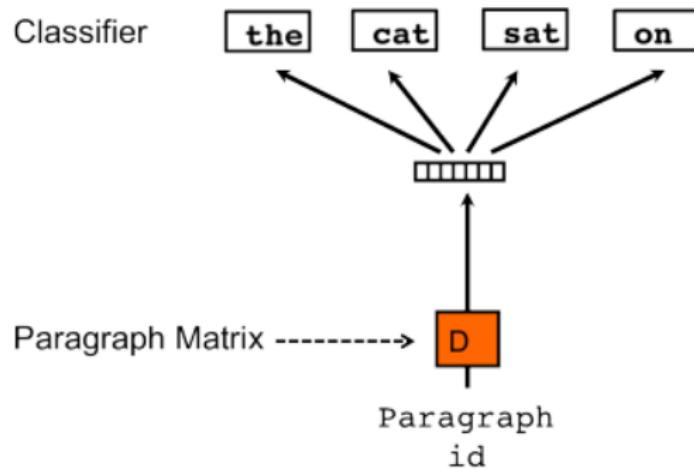


FIGURE 9.1 — PVDBOW (Mikilov-2014) : ici D représente une distribution via son vecteur

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

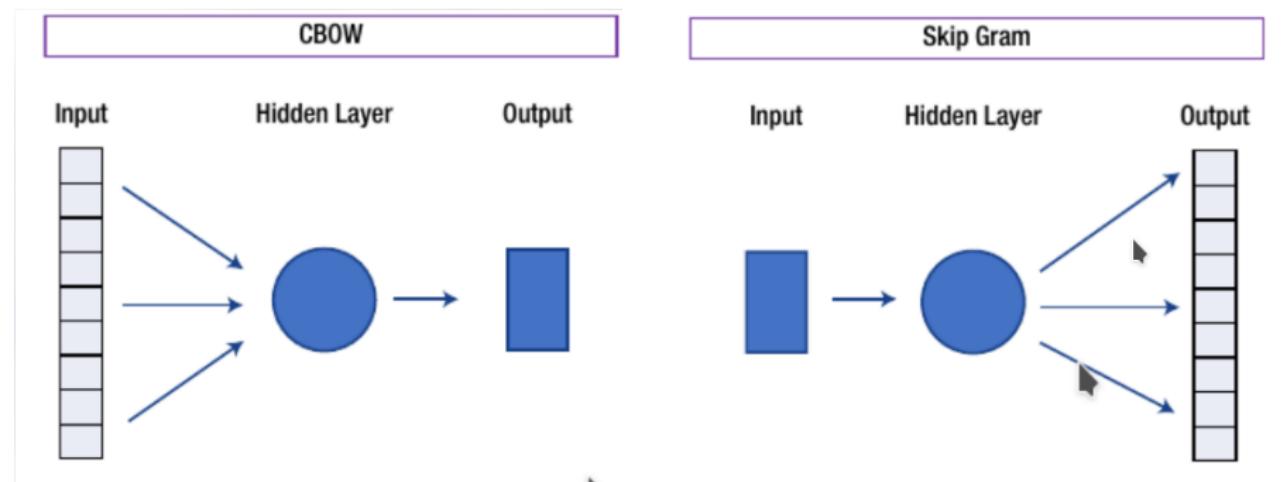
FIGURE 9.2 – PV = paragraphe2vect, WV = word2vect : Dai & al 2014

Complément Word2vect : détails

- Word2vect : plateforme de Deep-learning de Google pour la mise en place (apprentissage) du *word embedding*.
- Deep ? : un NN assez simple avec une seule couche cachée
- Peut prédire les mots proches (voisins) possibles : **contexte**.
- On entraîne ce NN non pas pour sa (couche de) sortie **mais juste pour extraire les pondérations de sa couche cachée !**
 - Ces pondérations sont justement les vecteurs de mot ("word vector", le sens *latent*) que l'on cherche à apprendre !
- C'est une technique connue et utilisée dans l'apprentissage non-supervisé en **feature learning** (☞ Idées des **CNN**).
 - Le NN utilisé permet de **compresser** un vecteur d'entrée dans la couche cachée et de le **décompresser** vers la couche de sortie.

Complément Word2vect : détails (suite)

- Rappel : deux types représentations dans Word2vect :
CBOW et Skip-Gram



Symboliquement, CBOW prend en entrée plusieurs mots + un contexte et prédit le suivant

Tandis que Skip-gram prend un mot en entrée et prédit les suivants (le contexte)

NN récurrents : un autre outil TM

Un exemple (à base de caractères) :

- Soit un vocabulaire de 4 lettres {h,e,l,o} ;
→ on veut entraîner un RNN sur la séquence "hello".
- L'ensemble d'apprentissage est en fait composé de 4 exemples d'apprentissage (4 lettres)
- Regardons le mot "hello" : **à la fin du processus**
 - 1. La proba de "e" doit être élevée dans le contexte de "h",
 - 2. idem pour "l" dans le contexte (voisinage) de "he",
 - 3. également "l" devrait apparaître dans le contexte de "hel",
 - 4. Enfin, "o" le devrait étant donnée le contexte "hell".
- On encode chaque caractère par One-Hot (dit aussi encodage *1-of-k*) avant de les envoyer UNE PAR UNE dans un NN.
- On observe ensuite la séquence de sortie de vecteurs de dim. 4 (une par caractère) que l'on interprète comme la confiance (proba) que le NN attache à chaque caractère **suivant** :

NN récurrents : un autre outil TM (suite)

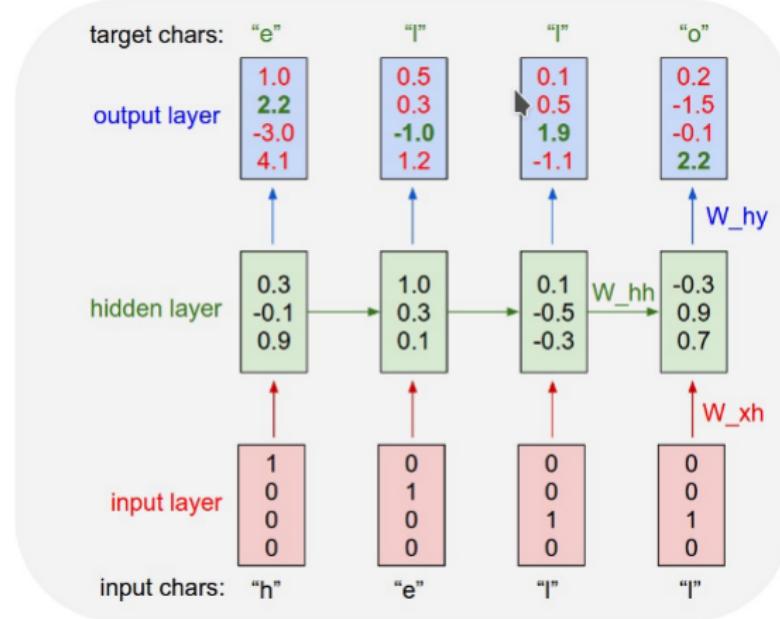


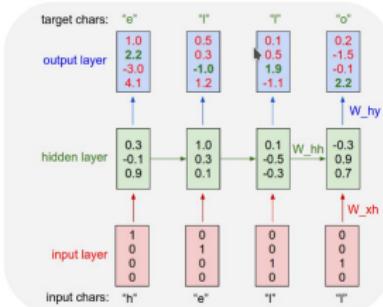
FIGURE 11.1 – Un exemple de **NN récurrent** à une entrée à 4-dim. Il apprend à deviner le caractère suivant dans un mot. La couche cachée possède 3 neurones (voir aussi <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>).

NN récurrents : un autre outil TM (suite)

- Ce diagramme représente 4 états successifs d'un RNN, déplié après 4 étapes du passage de "hell" :

- On a présenté "h" et on a obtenu en sortie
[1.0, 2.2, -3.0, 4.1]
- Pour l'instant, ce n'est pas "bon" :
il faudra que la valeur en vert (associée à la proba de "e") soit maximisée.
- A l'étape suivante ("e" est présentée), on obtient [0.5, 0.3, -1.0, 1.2] :
→ la valeur du rang de "l" (3e dans one-hot) doit s'améliorer, ...

- Dans un RNN, chaque itération s'appuie sur les résultats de l'itération précédente + son propre entrée.
- Ce processus se répète avec beaucoup d'autres mots et on réitère
- Avec un texte suffisamment long, on peut entraîner un réseau récurrent capable de générer (automatiquement) du texte (poèmes).
- L'utilisation des caractères (cf. *fastText* de **FaceBook**) nécessite un apprentissage plus long mais le résultat est plus fin (qu'avec des mots).



NN récurrents : détails

- Détails internes d'un NN récurrent :

input

$$x(t) = w(t) + s(t-1)$$

hidden layer(s) $s_j(t) = f \left(\sum_i x_i(t) u_{ji} \right)$

output layer $y_k(t) = g \left(\sum_j s_j(t) v_{kj} \right)$

+ sigmoid activation function

+ softmax function:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

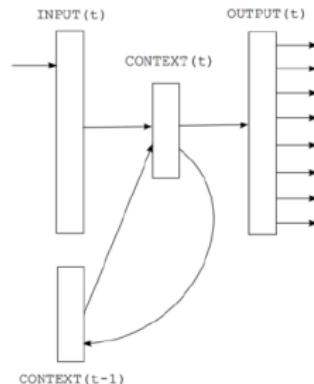


FIGURE 11.2 – éléments d'un réseau de neurones récurrent : $f = \text{Sigmoide}$, $g = \text{softmax}$

NN récurrents : détails (suite)

- Dans un RNN, l'état (de la couche) cachée actuel $h(t)$ est une fonction f de l'état (de la couche) cachée précédente $h(t - 1)$ ET de l'entrée $x(t)$.
- Le réseau utilise $h(t)$ comme un **résumé** de la séquence d'entrée passée jusqu'à t .

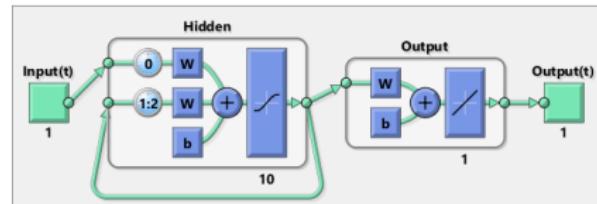
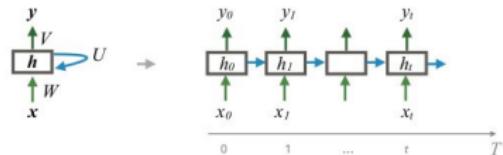


FIGURE 11.3 – Un RNN avec des couches cachées récurrentes dont l'activation dépend du temps / cycle précédent

NN récurrents : détails (suite)

Remarques :

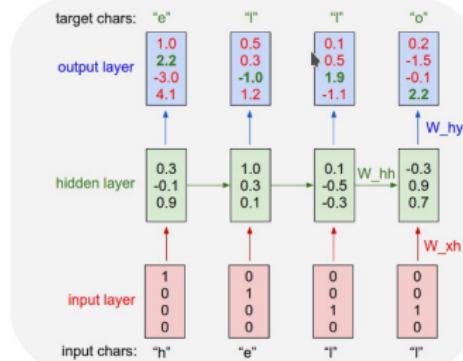
- La colonne droite du diagramme (rappelé) représente l'état du RNN après une itération sur "hell".

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" :

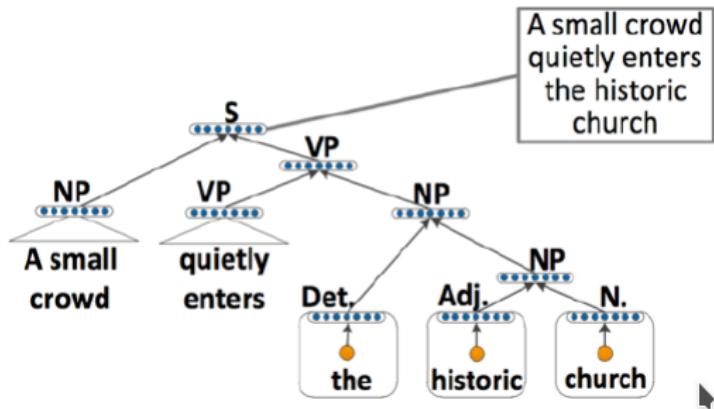


FIGURE 12.1 – Un réseau de neurones **récursive** (cf. grammaire **récursive** des expressions)

NN récursifs (suite)

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

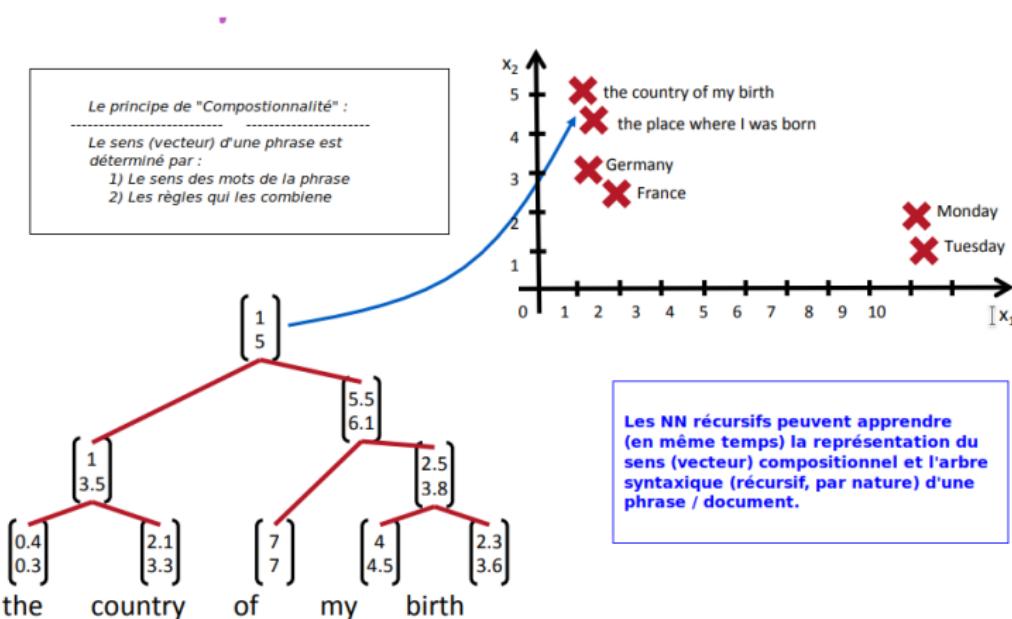


FIGURE 12.2 – Thanks to Bengio. July, 2012, UCLA

NN récursifs (suite)

- D'où vient la structure de la phrase : on l'apprend !

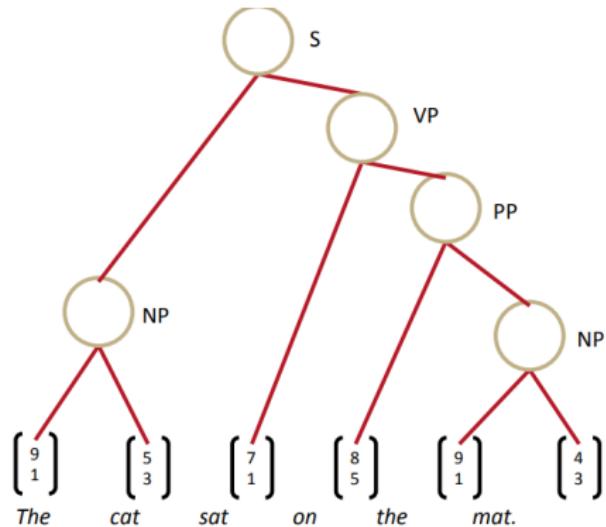


FIGURE 12.3 – La structure syntaxique de la phrase et les vecteurs (sens) de taille 2 calculés par un NN récursif

NN récursifs (suite)

- Si on regroupe les deux :

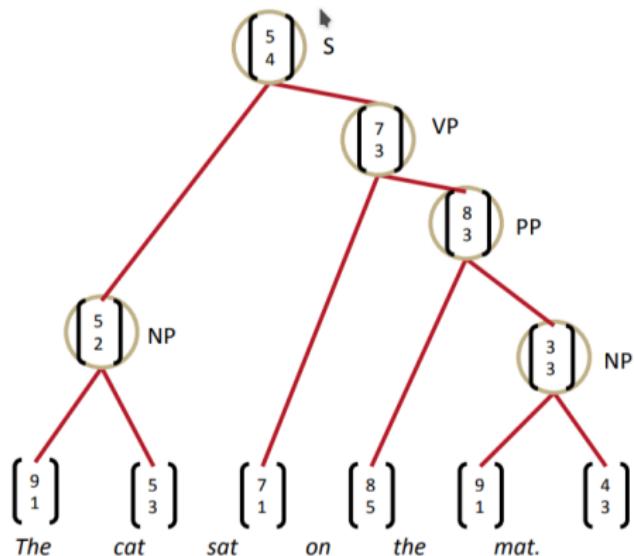


FIGURE 12.4 – La structure syntaxique et les vecteurs (sens)

NN récursifs (suite)

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

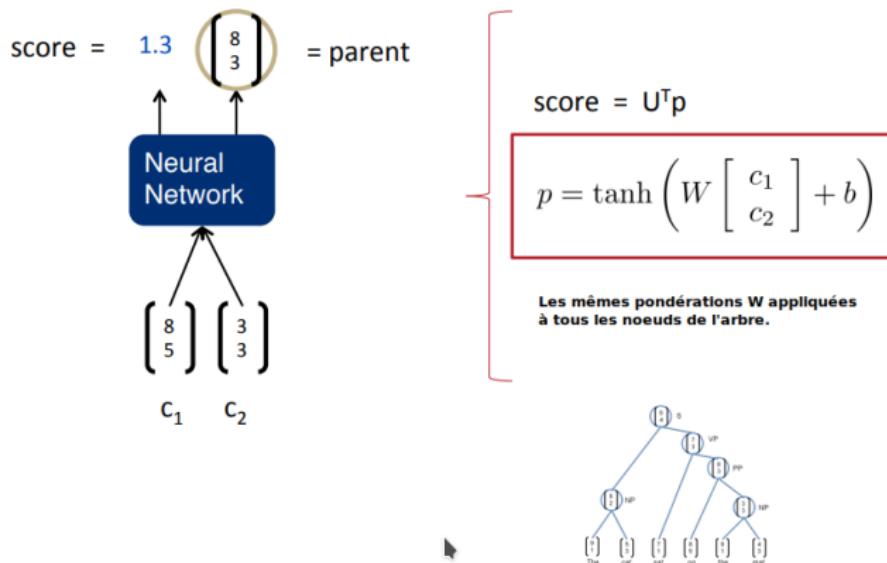


FIGURE 12.5 – Le NN apprend à combiner des noeuds

NN récursifs (suite)

- Comment "remonter" la structure ? :
 - Entrées : deux noeuds candidats (avec leur vecteur)
 - Sortie : le score de plausibilité du nouveau noeud.

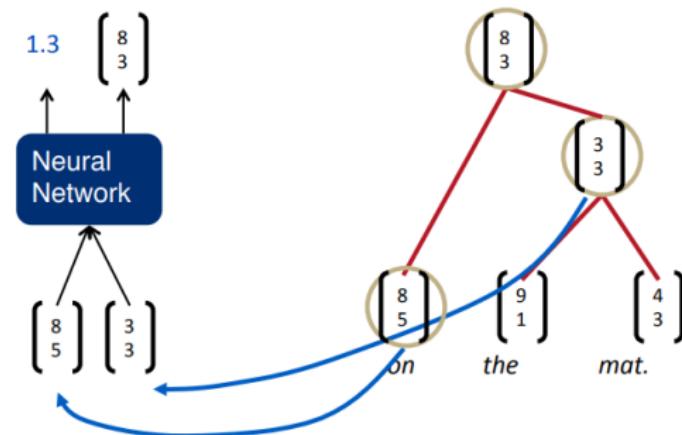
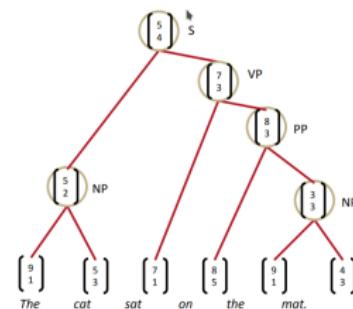
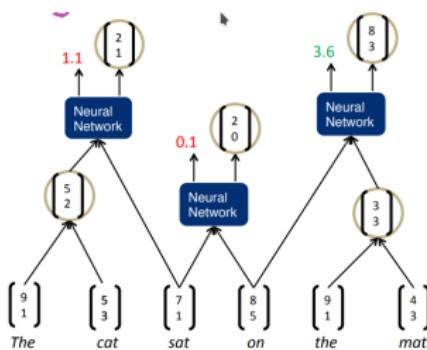
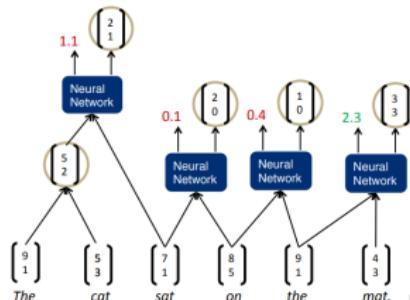
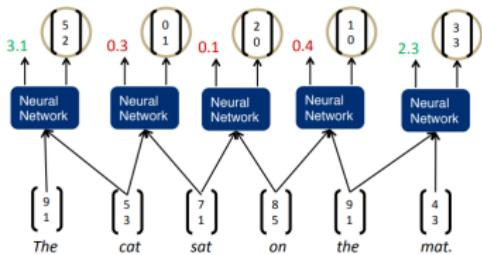


FIGURE 12.6 – Le nouveau noeud le plus plausible est retenu

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)

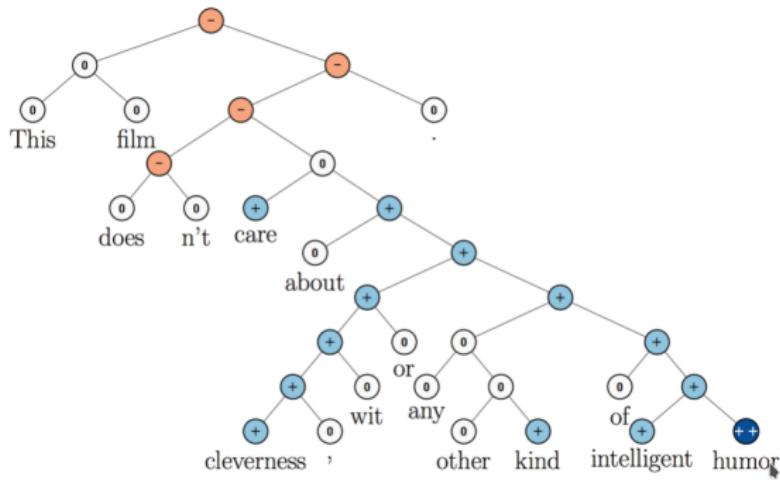


FIGURE 12.7 — sémantique Compositionnelle appliquée aux sentiments.
Socher & al. 2013 (voir nlp.stanford.edu)

Lexicon

- Utilisé par ex. pour la polarité des opinions .
- Une trace d'exécution (code disponible, me demander) :

```
~/Big-Data/Big-Data-18-19/Lexicon-sur-Tweets-exemple$ python doAnalysis.py
Positive: 2648 (37.8%)
Negative: 889 (12.7%)
Neutral: 3460 (49.4%)
```

TOP NEGATIVE TWEETS

```
id=2677868053, score=-4.00, clean=new twitter throw down ~ some rant and scream "coup d'tat!" ~ others mumble "meh" lqw
haiku
newtwitter
id=2697604166, score=-4.00, clean=how can i get rid of the newtwitter banner? it is only slightly less annoying than
newtwitter.
id=2707537275, score=-4.00, clean=i try and try and try — bloody nothing! newtwitter
id=2706989783, score=-4.00, clean=newtwitter needs 2 up its game soooo bloody slow and annoying at times
id=2706682313, score=-4.00, clean=why does the new twitter hate me?!? newtwitter fail fail rude doublefail
id=2717601341, score=-4.00, clean=im mad as hell i got the newtwitter. this shxt ugly as hell .
id=2717149384, score=-4.00, clean=the newtwitter waste my time..ugh too slow and complicated —.—
id=2717093643, score=-4.00, clean=tons of complaints about newtwitter & lost backgrounds. you either have a small monitor
or it's
set to a low resolution, try over 1024x768
id=2618801378, score=-3.00, clean=new twitter has these little black bubbles popping up everywhere saying follow this person
that
i hate newtwitter it has a few plus but su
id=2626151362, score=-3.00, clean=cant we get rid of the try new twitter panel at the top?! & while we're at it..get rid of
newtwitter completely!
```

TOP POSITIVE TWEETS

```
id=2651359182, score=5.00, clean=just watching glee on this newtwitter, love glee not so sure about the new twitter
```

Lexicon (suite)

id=2657803366, score=5.00, clean=do you like my new twitter back ground and does it fit in newtwitter i don't want to "tryitnow"
 again just incase i can't come out again
 id=2659289358, score=5.00, clean=love love love the new twitter setup. it's so pretty and so awesome. well done twitter people.
 newtwitter
 id=2697712951, score=5.00, clean=newtwitter please bring back the ability to x out trends. on the positive side, i do like the pop out which shows pics/vids
 id=2697140446, score=5.00, clean=dear newtwitter banner at the top of the web page, i tried you, didn't like you so i'm moving on.
 now please go away.
 id=2706900778, score=5.00, clean=@dan303 i agree regarding the newtwitter hate, it is pretty cool...people just don't like change
 of any kind :p
 id=2717317688, score=5.00, clean=@theknottybride oooh! i love the newtwitter! i especially love that it helps predict who u r mentioning so u can be sure to spell it right
 id=2717292836, score=5.00, clean=dear newtwitter please add back the who you both follow fxn... that was awesome and i need it
 back !!!! :(it would only make it cooler ;)
 id=2679203040, score=6.00, clean=newtwitter you can change back to old twitter right top ur name, push leave preview will get you
 back to old twitter i like new twitter
 id=2717618096, score=6.00, clean=i like the newtwitter — its more modern and runs great ... the right way twitter, please go on
 like this ;)

TOP NEUTRAL TWEETS

id=2602860537, score=0.00, clean=10 things missing in the new twitter interface <http://bit.ly/blzxi3> newtwitter
 id=2602678435, score=0.00, clean=seriously, still no new twitter layout? how many weeks is this suppose to take?! newtwitter
 id=2602600751, score=0.00, clean=confieso q el newtwitter no me mata, c/vez que veo "psst the new twitter **is** here" (o como sea que dice) lo ignoro esperando a q desaparezca
 id=2602585892, score=0.00, clean=i have the new twitter. who else? :) newtwitter
 id=2602486242, score=0.00, clean=when is the new twitter going to rollout to me? newtwitter

Lexicon (suite)

```
id=2602304053, score=0.00, clean=7 really cool things about the newtwitter http://ow.ly/1qyuwg
id=2602293760, score=0.00, clean=how to make backgrounds for newtwitter http://bit.ly/9h1c9b via @askdebra @gautamghosh
@shilpiiz
@socialsidermedia
id=2602291642, score=0.00, clean=7 really cool things about newtwitter | @socialmedia2day http://bit.ly/dccug6
id=2605718918, score=0.00, clean=@twitter and finally the new twitter has come to me! wow!!! http://goo.gl/trgc newtwitter
id=2605716033, score=0.00, clean=@theonlyanil you just block to use newtwitter reason behind some celeb complain @ev dont
give him
new twitter lolz
```

Quelques références (needs a 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