POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Introduction au text mining

Mehdi Miah

20 février 2018

Analyse textuelle (text mining) : vise à extraire de l'information à partir de données textuelles (document, paragraphe, phrase, mots, caractères, …)

← Sujet de cette présentation

Analyse sémantique / traitement du langage naturel (natural language processing – NLP) : va plus loin dans la compréhension du message

Données non structurées !

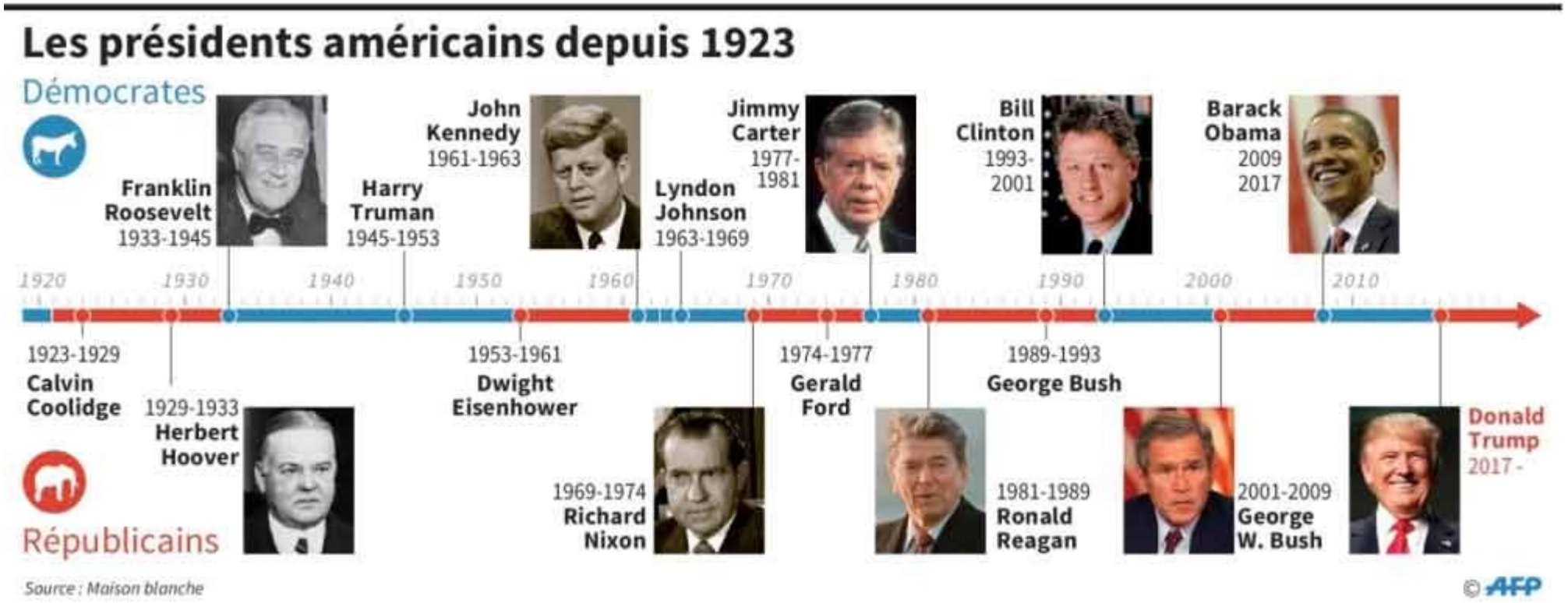
Contexte et objectifs :

- Porter sur l'analyse textuelle
- Structurer les données
- Décrire les unités textuelles (mots, documents, corpus) sous la forme de vecteurs numériques (vector space representation)

Une fois que les textes seront représentés sous forme vectorielle, on pourra employer les techniques classiques de data mining et de machine learning

Plan :

1. Pré-processing des données
2. Méthodes
   a. TF-IDF
   b. LSA
   c. Words embeddings

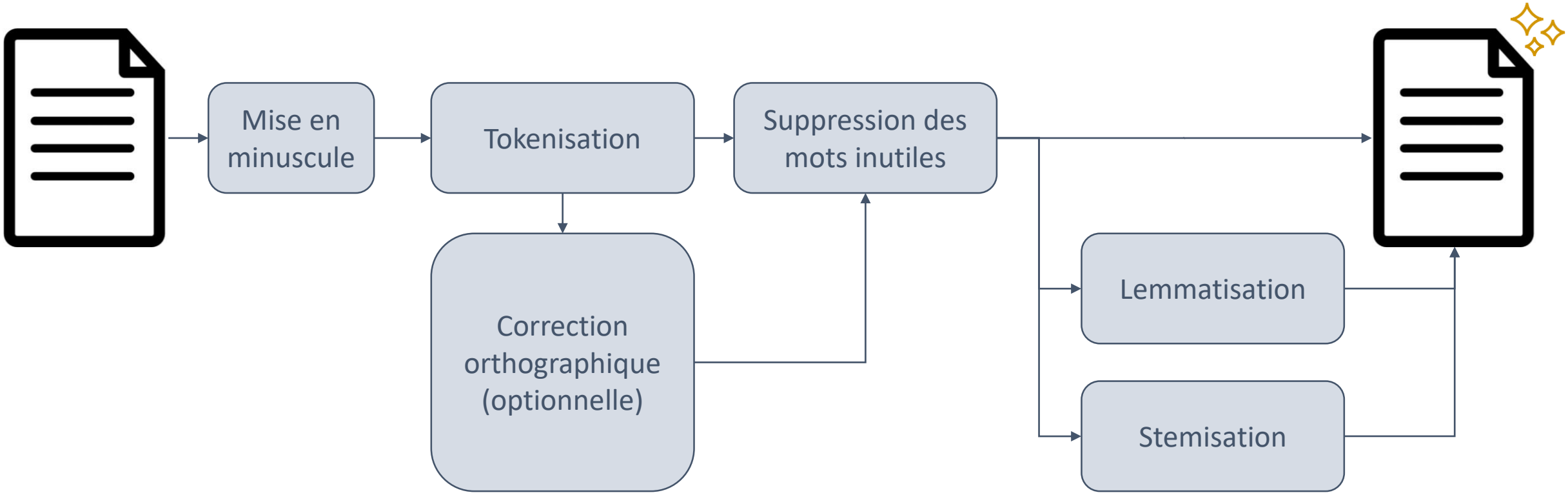


Questions possibles :

- Peut-on connaître le parti politique à partir d'une étude de son discours d'inauguration ?
- Quels sont les présidents ayant le plus de point commun ?
- Trump est-il une valeur aberrante ?
- Peut-on retrouver l'histoire américaine à travers une étude des discours ? Par thème ?

PRE-PROCESSING

Pre-processing



Mise en minuscule (étape 1 - lowerize)

Intérêt :
- Ne tenir compte de la casse des mots

```
# Initial text (step 0)
trump_begin = trump[0:465] #first two paragraphs
print(trump_begin)
```

```
Chief Justice Roberts, President Carter, President Clinton, President Bush, President Obama,
fellow Americans, and people of the world: Thank you.

We, the citizens of America, are now joined in a great national effort to rebuild our country
and restore its promise for all of our people. Together, we will determine the course of Amer
ica and the world for many, many years to come. We will face challenges, we will confront har
dships, but we will get the job done.
```

```
# Step 1 : Lowerisation
trump_lower = trump_begin.lower()
print(trump_lower)
```

```
chief justice roberts, president carter, president clinton, president bush, president obama,
fellow americans, and people of the world: thank you.

we, the citizens of america, are now joined in a great national effort to rebuild our country
and restore its promise for all of our people. together, we will determine the course of amer
ica and the world for many, many years to come. we will face challenges, we will confront har
dships, but we will get the job done.
```

Mise en token (étape 2 - tokenization)

Intérêt :
- n'avoir que les mots employés
- Possibilité de considérer les n-grams

Attention :
- U.N. vs UN
- New York
- 20/02/2018
- One-man-show

```
# Step 2 : Tokenization
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'\w+')
```

```
# Tokenisation on Trump's speechs
tokens_trump = tokenizer.tokenize(trump_lower)
print(tokens_trump)
```

```
['chief', 'justice', 'roberts', 'president', 'carter', 'president', 'clinton', 'president', '
bush', 'president', 'obama', 'fellow', 'americans', 'and', 'people', 'of', 'the', 'world', 't
hank', 'you', 'we', 'the', 'citizens', 'of', 'america', 'are', 'now', 'joined', 'in', 'a', 'g
reat', 'national', 'effort', 'to', 'rebuild', 'our', 'country', 'and', 'restore', 'its', 'pro
mise', 'for', 'all', 'of', 'our', 'people', 'together', 'we', 'will', 'determine', 'the', 'co
urse', 'of', 'america', 'and', 'the', 'world', 'for', 'many', 'many', 'years', 'to', 'come',
'we', 'will', 'face', 'challenges', 'we', 'will', 'confront', 'hardships', 'but', 'we', 'will
', 'get', 'the', 'job', 'done']
```

Suppression des mots inutiles (étape 3 – stopwords removal)

Intérêt :
- n'avoir que les mots « utiles »
- Réduit la dimension du problème

Attention :
- Il faut ajuster le dictionnaire des mots retirés : en anglais, « again » est un stop_word ; en français, « les » n'est pas un stop_word
- Parfois utile : « to be or not to be »

```
# Step 4 : stopwords removal
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english')) #putting it in a set makes computation faster =)
```

```
print(sorted(stop_words)) # words in stop_words
```

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an', 'and', 'any',
'are', 'aren', 'as', 'at', 'be', 'because', 'been', 'before', 'being', 'below', 'between', 'b
oth', 'but', 'by', 'can', 'couldn', 'd', 'did', 'didn', 'do', 'does', 'doesn', 'doing', 'don'
, 'down', 'during', 'each', 'few', 'for', 'from', 'further', 'had', 'hadn', 'has', 'hasn', 'h
ave', 'haven', 'having', 'he', 'her', 'here', 'hers', 'herself', 'him', 'himself', 'his', 'ho
w', 'i', 'if', 'in', 'into', 'is', 'isn', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma', 'me
', 'mightn', 'more', 'most', 'mustn', 'my', 'myself', 'needn', 'no', 'nor', 'not', 'now', 'o'
, 'of', 'off', 'on', 'once', 'only', 'or', 'other', 'our', 'ours', 'ourselves', 'out', 'over'
, 'own', 're', 's', 'same', 'shan', 'she', 'should', 'shouldn', 'so', 'some', 'such', 't', 't
han', 'that', 'the', 'their', 'theirs', 'them', 'themselves', 'then', 'there', 'these', 'they
', 'this', 'those', 'through', 'to', 'too', 'under', 'until', 'up', 've', 'very', 'was', 'was
n', 'we', 'were', 'weren', 'what', 'when', 'where', 'which', 'while', 'who', 'whom', 'why', '
will', 'with', 'won', 'wouldn', 'y', 'you', 'your', 'yours', 'yourself', 'yourselves']
```

```
filtered_trump = [w for w in tokens_trump if not w in stop_words]
print(filtered_trump)
```

```
['chief', 'justice', 'roberts', 'president', 'carter', 'president', 'clinton', 'president', '
bush', 'president', 'obama', 'fellow', 'americans', 'people', 'world', 'thank', 'citizens', '
america', 'joined', 'great', 'national', 'effort', 'rebuild', 'country', 'restore', 'promise'
, 'people', 'together', 'determine', 'course', 'america', 'world', 'many', 'many', 'years', '
come', 'face', 'challenges', 'confront', 'hardships', 'get', 'job', 'done']
```

Lemmatisation (étape 4 – version 1)

Intérêt :

- Réduction à une forme commune
- Réduit la dimension du problème

Attention :

- Nécessite des connaissances en grammaire, part of speech, …

```
# Step 5 : Stemisation or Lemmatisation
from nltk.stem import PorterStemmer, WordNetLemmatizer

stemmer = PorterStemmer()
lemmatiser = WordNetLemmatizer()
```

```
lemm_trump = [lemmatiser.lemmatize(w, pos="v") for w in filtered_trump ]
print(lemm_trump)
```

```
['chief', 'justice', 'roberts', 'president', 'carter', 'president', 'clinton', 'president', '
bush', 'president', 'obama', 'fellow', 'americans', 'people', 'world', 'thank', 'citizens', '
america', 'join', 'great', 'national', 'effort', 'rebuild', 'country', 'restore', 'promise',
'people', 'together', 'determine', 'course', 'america', 'world', 'many', 'many', 'years', 'co
me', 'face', 'challenge', 'confront', 'hardships', 'get', 'job', 'do']
```

Stemisation (étape 4 – version 2)

Intérêt :
- Réduction à la radical
- Réduit la dimension du problème

Attention :
- Le résultat n'est pas toujours lisible

```
# Step 5 : Stemisation or Lemmatisation
from nltk.stem import PorterStemmer, WordNetLemmatizer

stemmer = PorterStemmer()
lemmatiser = WordNetLemmatizer()
```

```
stem_trump = [stemmer.stem(w) for w in filtered_trump ]
print(stem_trump)
```

```
['chief', 'justic', 'robert', 'presid', 'carter', 'presid', 'clinton', 'presid', 'bush', 'pre
sid', 'obama', 'fellow', 'american', 'peopl', 'world', 'thank', 'citizen', 'america', 'join',
'great', 'nation', 'effort', 'rebuild', 'countri', 'restor', 'promis', 'peopl', 'togeth', 'de
termin', 'cours', 'america', 'world', 'mani', 'mani', 'year', 'come', 'face', 'challeng', 'co
nfront', 'hardship', 'get', 'job', 'done']
```

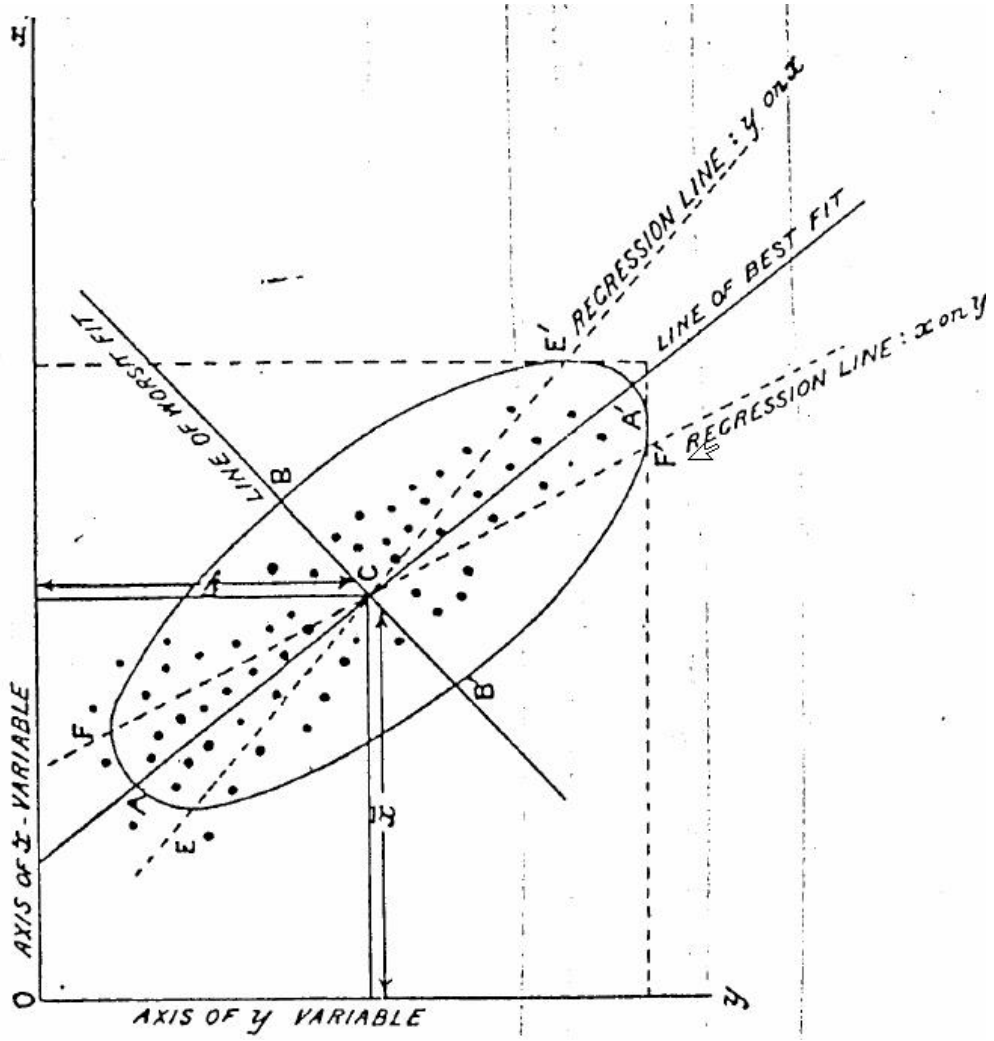
NUAGE DE MOTS : QUI A DIT QUOI ?

METHODE DE VISUALISATION

Analyse en composantes principales

Consiste à transformer des variables liées entre elles (dites « corrélées » en statistique) en nouvelles **variables décorrélées** les unes des autres. Ces nouvelles variables sont nommées « composantes principales », ou axes principaux. Elle permet au praticien de **réduire le nombre de variables et de rendre l'information moins redondante.**

Lorsqu'on veut compresser un ensemble de N variables aléatoires, les n premiers axes de l'analyse en composantes principales sont un meilleur choix, du point de vue de l'inertie ou de la variance.

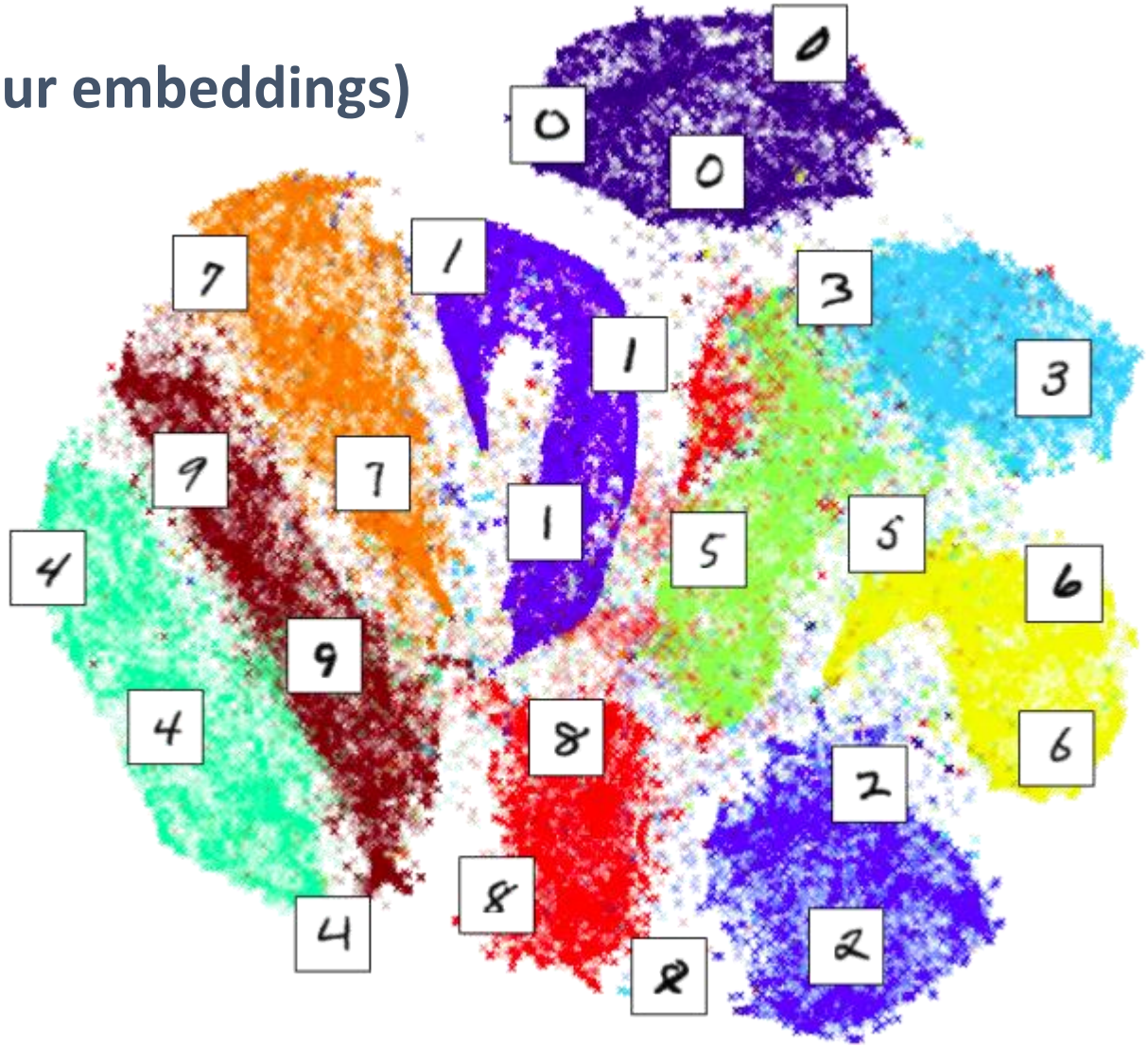


Extrait de l'article de Pearson de 1901 : la recherche de la « droite du meilleur ajustement ».

t-SNE (t-distribution stochastic neighbour embeddings)

Technique de **réduction de dimension** pour la visualisation de données développée par Geoffrey Hinton et Laurens van der Maaten. Il s'agit d'une **méthode non-linéaire** permettant de représenter un ensemble de points d'un espace à grande dimension dans un espace de deux ou trois dimensions, les données peuvent ensuite être visualisées avec un nuage de points.

L'algorithme t-SNE se base sur **une interprétation probabiliste des proximités**.



MNIST images visualised in two dimesnions using t-SNE. Colours indicate the digit of each image.

METHODE DE VECTORISATION

Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus. Le poids **augmente proportionnellement au nombre d'occurrences** du mot dans le document. Il varie également **en fonction de la fréquence du mot dans le corpus**.
Chaque document est représenté par un vecteur dans l'espace de tous les mots utilisés par le corpus

TF : term frequency
IDF : inverse document frequency

$$\mathrm{TF}(w;d) = \frac{\text{nombre d'occurences du mot } w \text{ dans le document } d}{\text{nombre de tokens dans le document } d}$$

$$\mathrm{IDF}(w;c) = \log\left(\frac{\text{nombre total de documents dans le corpus } c}{\text{nombre de documents où le mot } w \text{ apparait}}\right)$$

$$\text{TF-IDF}(w;c,d) = \mathrm{TF}(w;d) \times \mathrm{IDF}(w;c)$$

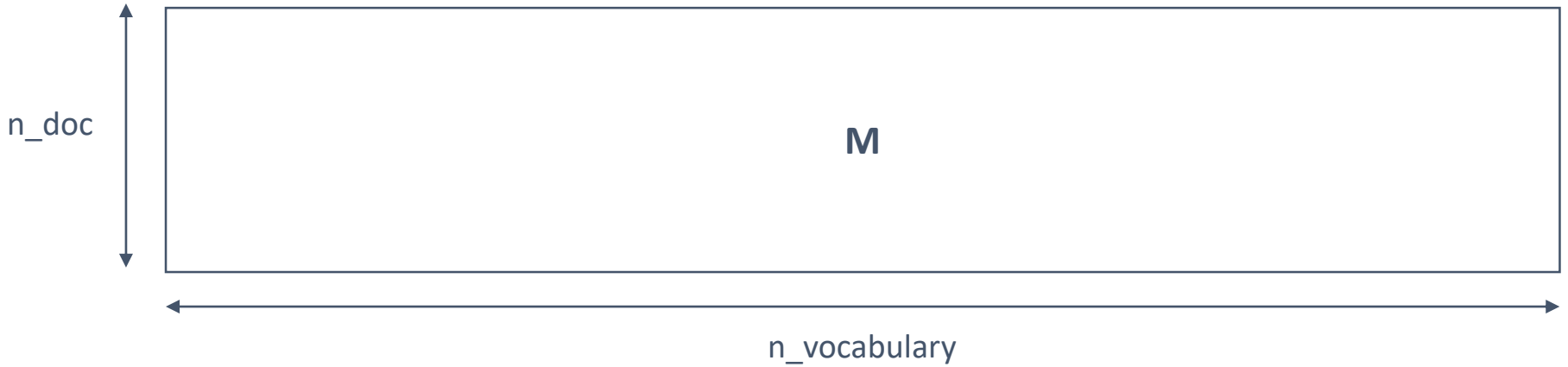
Après avoir calculé les poids de chaque mot dans un document, on obtient un vecteur pour chaque document, il est parfois utile de normaliser (norme L1 ou L2) le vecteur.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tf = TfidfVectorizer(analyzer='word', norm = 'l2')
```

```
tfidf_matrix = tf.fit_transform(speech_output)
```

On obtient une matrice M de dimension :

- Nombre de lignes = nombre de documents
- Nombre de colonnes = nombre de mots uniques dans le corpus



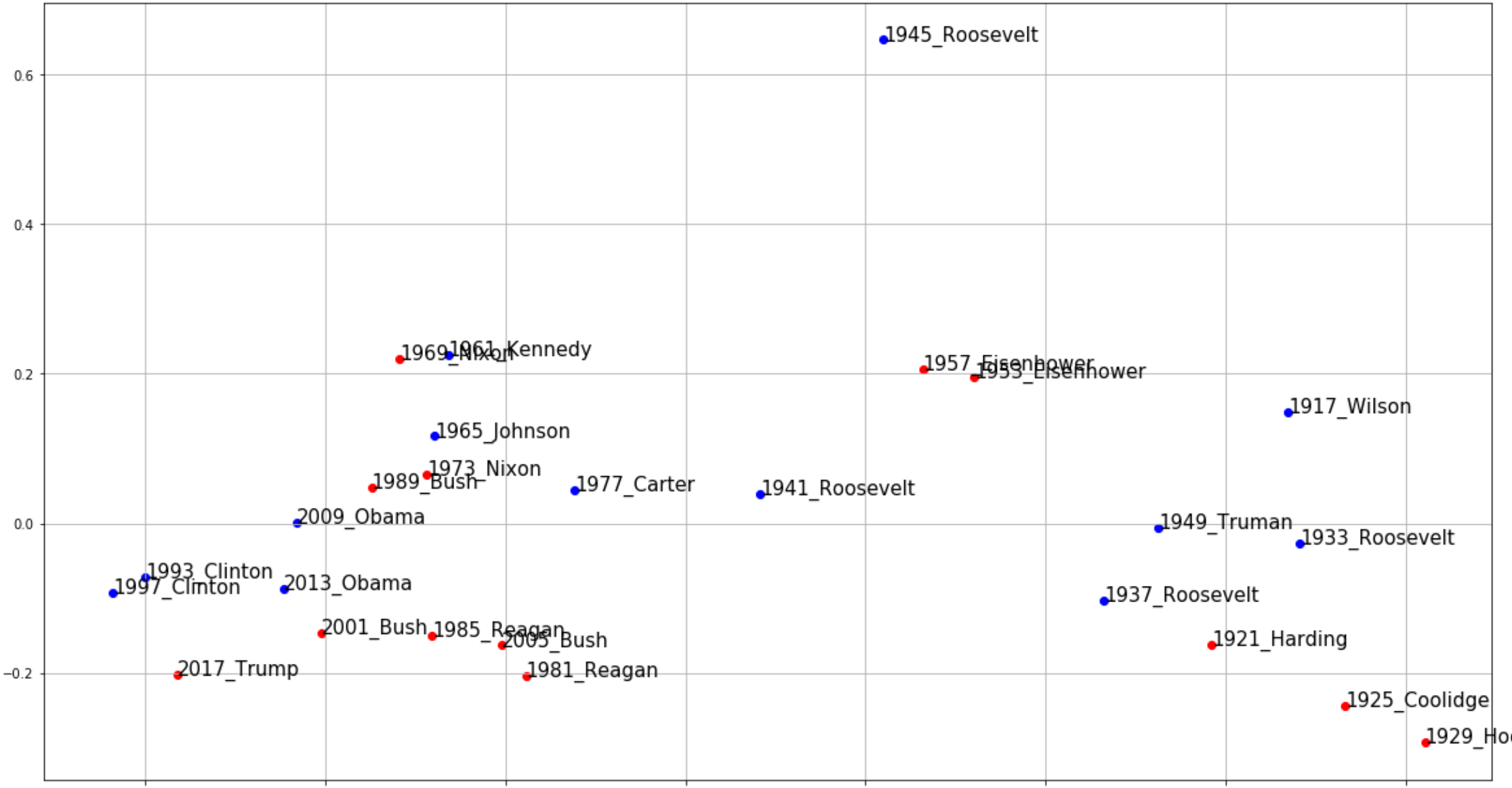
Matrice M :

- Normalisée ligne par ligne
- Sparce : beaucoup d'éléments nuls
- De grande dimension, en général, n_vocabulary >> n_doc

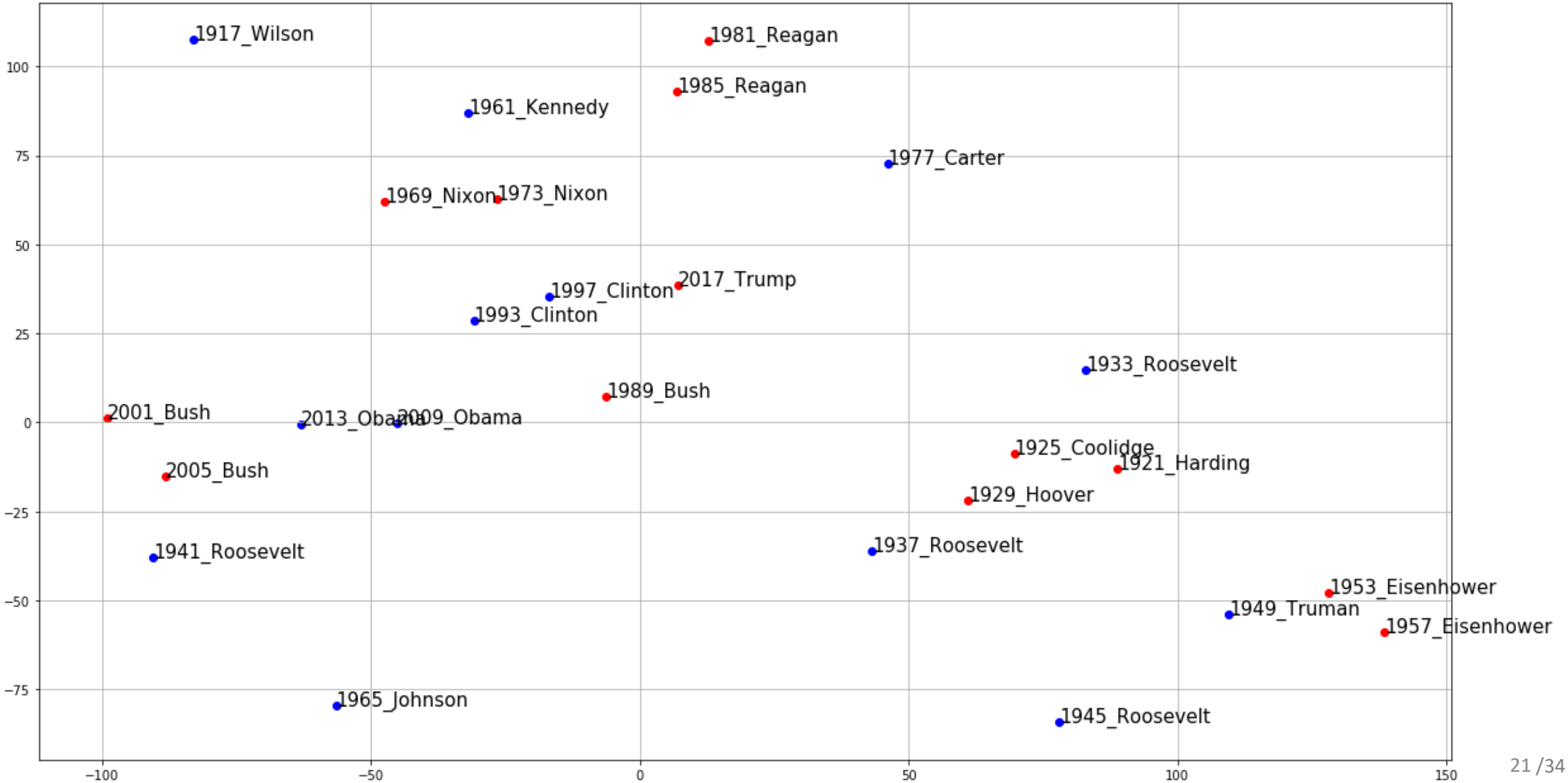
- Modélisation facile à comprendre
- Beaucoup de variantes possibles

- Beaucoup de variantes possibles
- Matrice sparse de grande dimension : visualisation difficile
- Ne tient pas compte de l'ordre des mots (le chat mange la souris VS la souris mange le chat)
- Ne tient pas compte de la polysémie (souris d'ordinateur, souris de laboratoire) : voir algorithme de Lesk pour résoudre l'ambiguïté
- Ne tient pas compte du sens des mots (le félin dévore la souris VS le chat mange le rongeur)

TF-IDF (VISUALISATION PAR ACP)



TF-IDF (VISUALISATION PAR t-SNE)

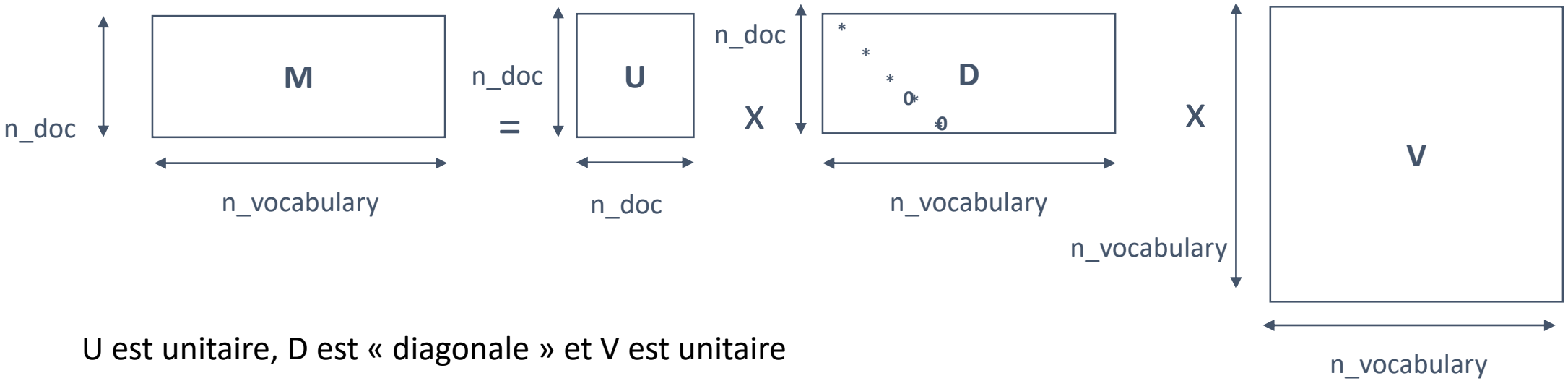


LSA : analyse sémantique latente (Latent semantic analysis)

Vectorisation via TF-IDF

Décomposition en valeurs singulières

Conserver qu'une partie de l'inertie



U est unitaire, D est « diagonale » et V est unitaire

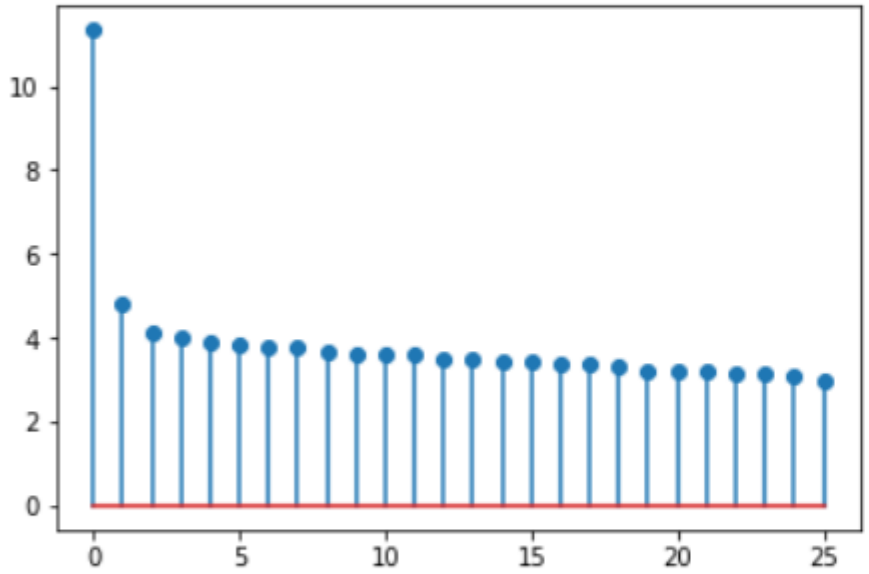
- Permet la création de topics car capable de capturer des similarités sémantiques et syntaxiques
- Paramétrisé par la proportion d'inertie conservé (part des valeurs singulières conservée)
- Réduction de dimension
- Donne en général de meilleurs résultats que TF-IDF

- Nécessite TF-IDF
- Nécessite de tout recalculer si un mot est rajouté
- Cout quadratique pour effectuer la décomposition en valeurs singulières

```
u,sigma,vt = np.linalg.svd(tfidf_matrix.todense())
```

```
plt.stem(range(len(sigma)), sigma/sum(sigma)*100)
```

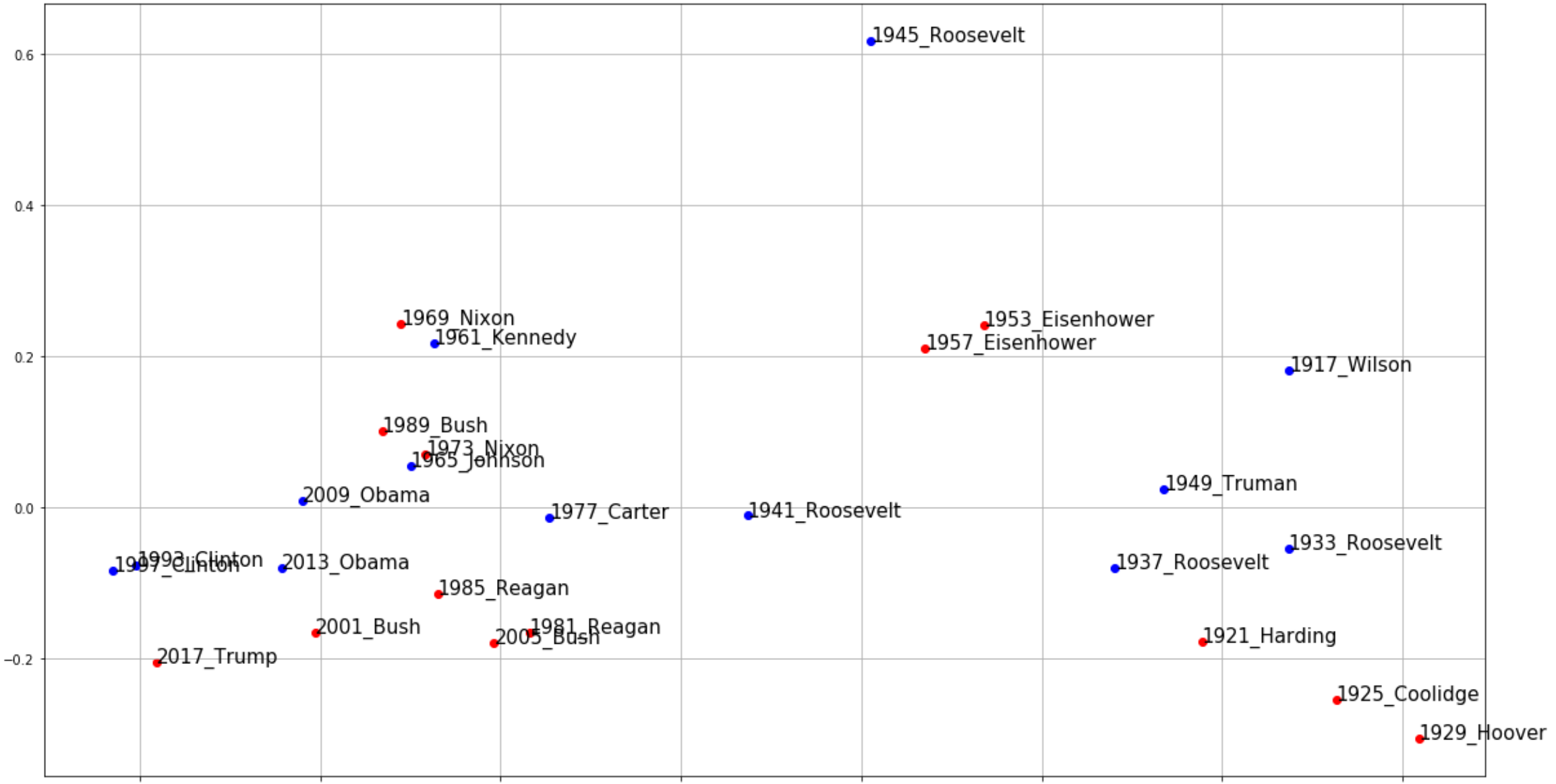
```
<Container object of 3 artists>
```



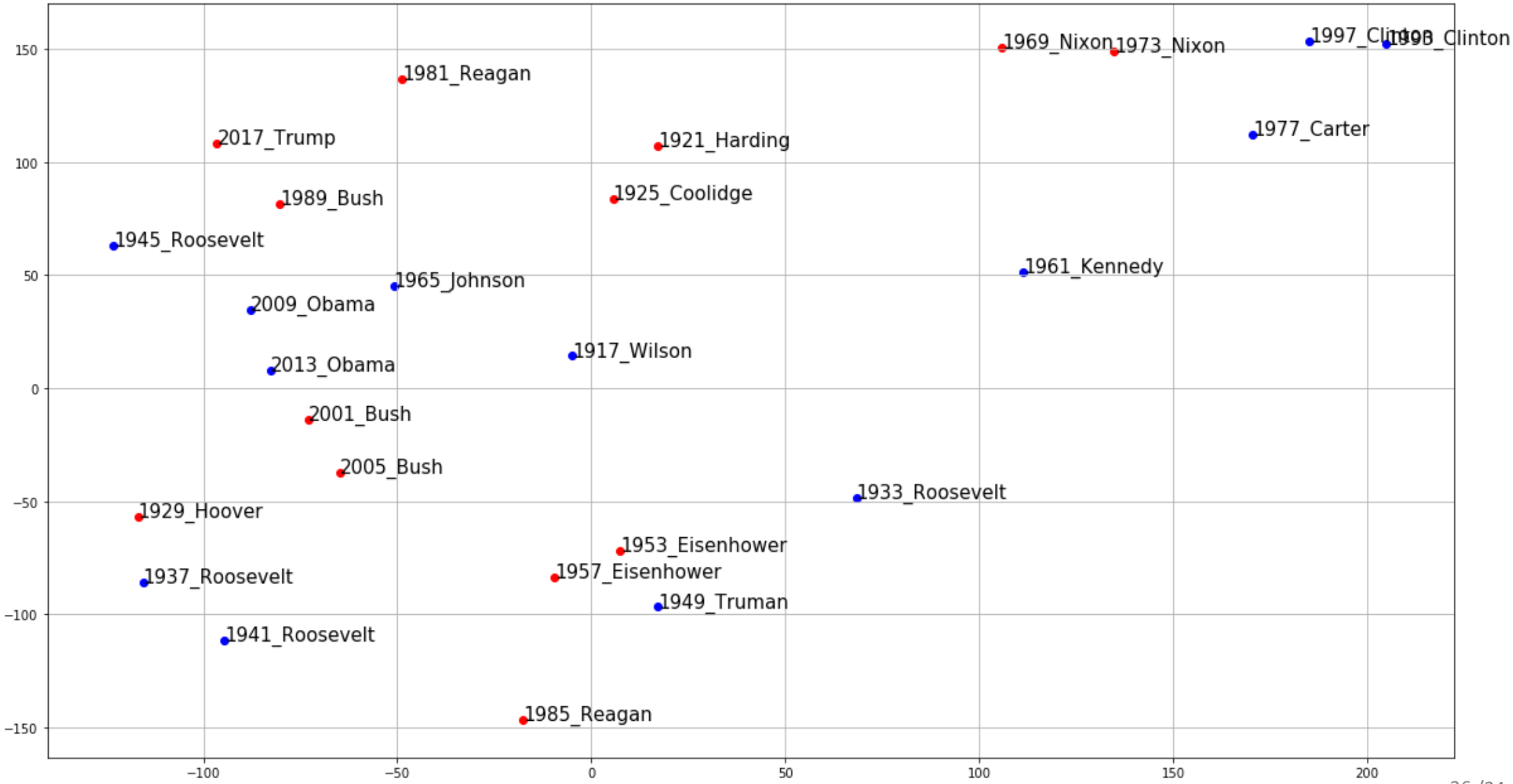

```
# We only keep the first elements of sigma
nb_rank = 12

# One reconstruct the matrix
reconst_matrix = np.dot(u[:,:nb_rank],np.dot(np.diag(sigma[:nb_rank]),vt[:nb_rank,:]))
```

LSA (VISUALISATION PAR ACP)



LSA (VISUALISATION PAR t-SNE)

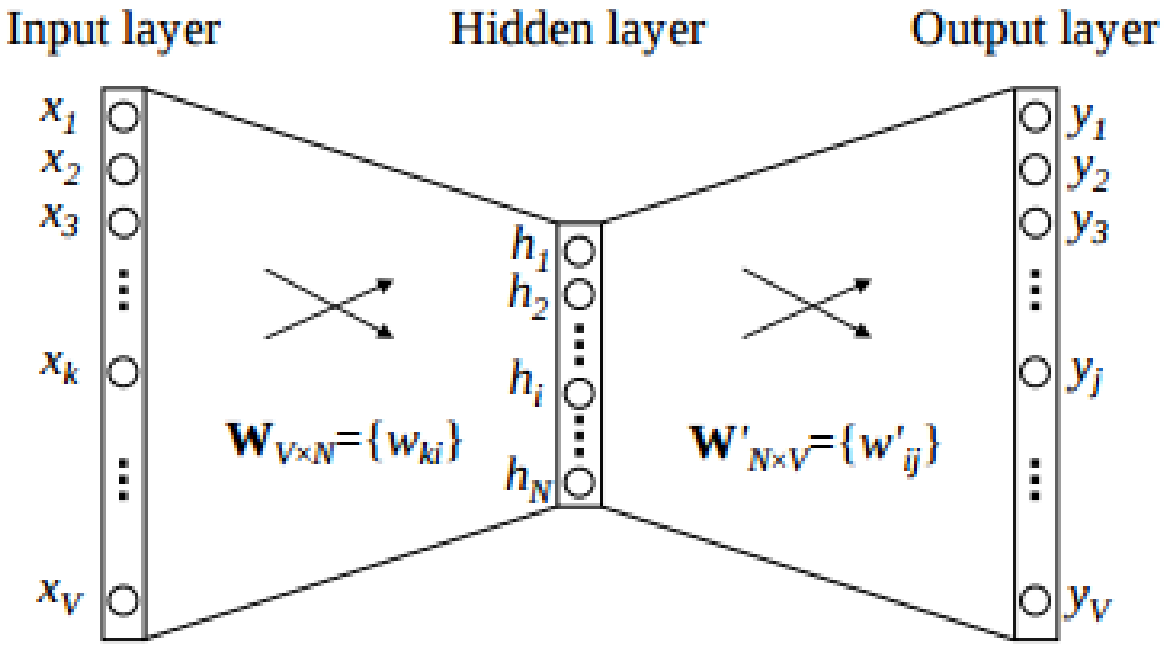


Word2Vec : méthode créée en 2013 par Tomas Mikolov (Google) basée sur un réseau de neurones à une couche cachée

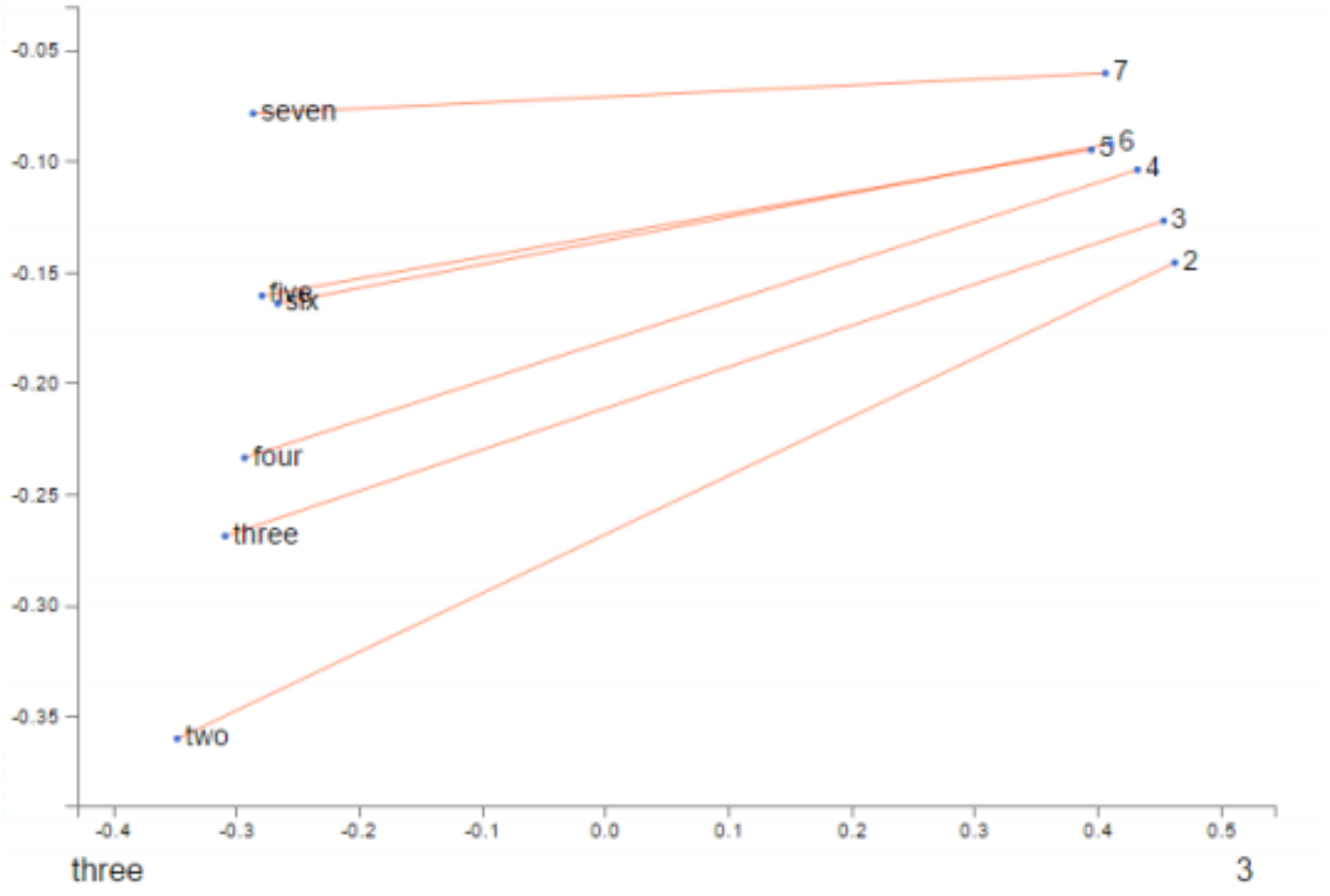
A partir d'une base d'apprentissage construite sur un document (ou ensemble de documents), le modèle prédit le mot $\mathbf{y_i}$ se trouvant dans l'entourage du mot $\mathbf{x_i}$

Une fois le modèle appris, on ne conserve que les poids de la matrice **W**

Ainsi, chaque mot est caractérisé par un vecteur de dimension N, généralement 100.

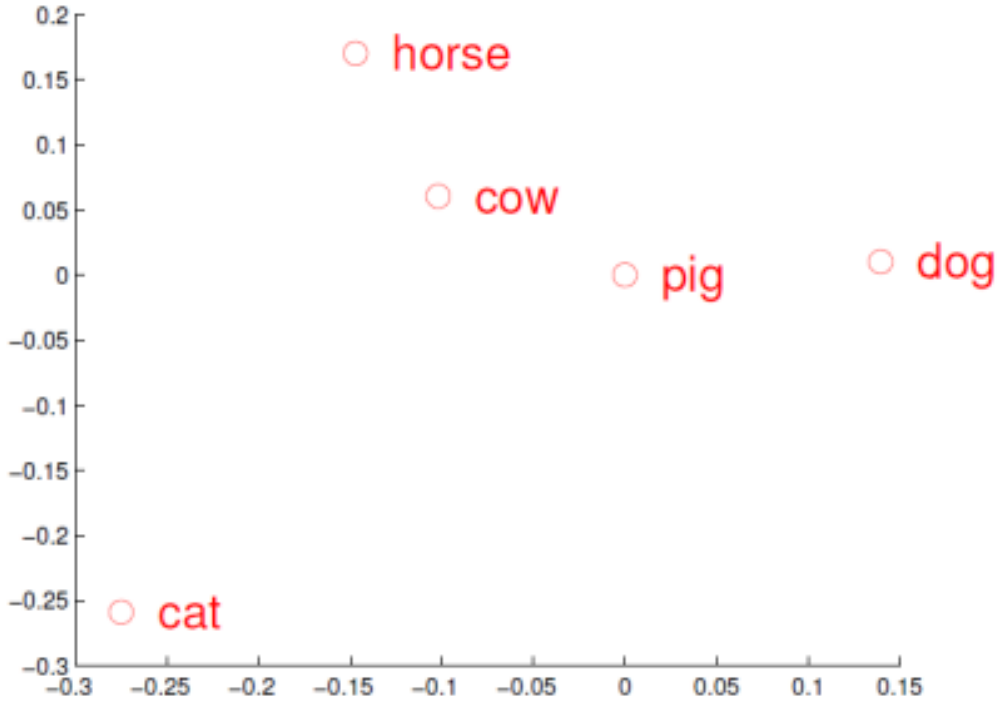


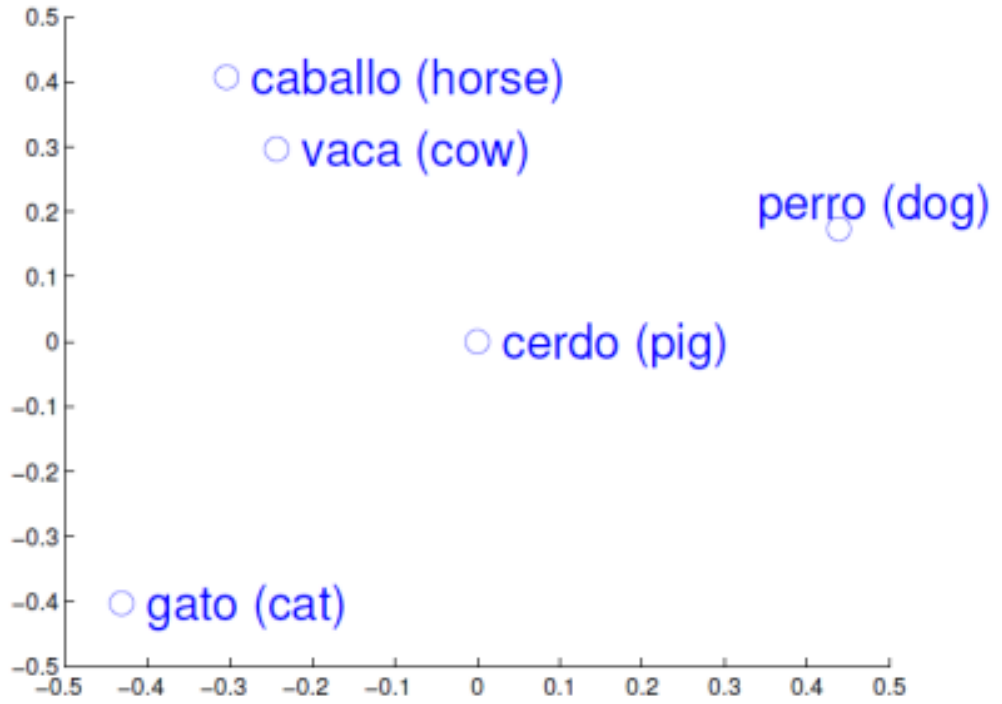
Modèle du Word2Vec



L'ordre des nombres est conservée !

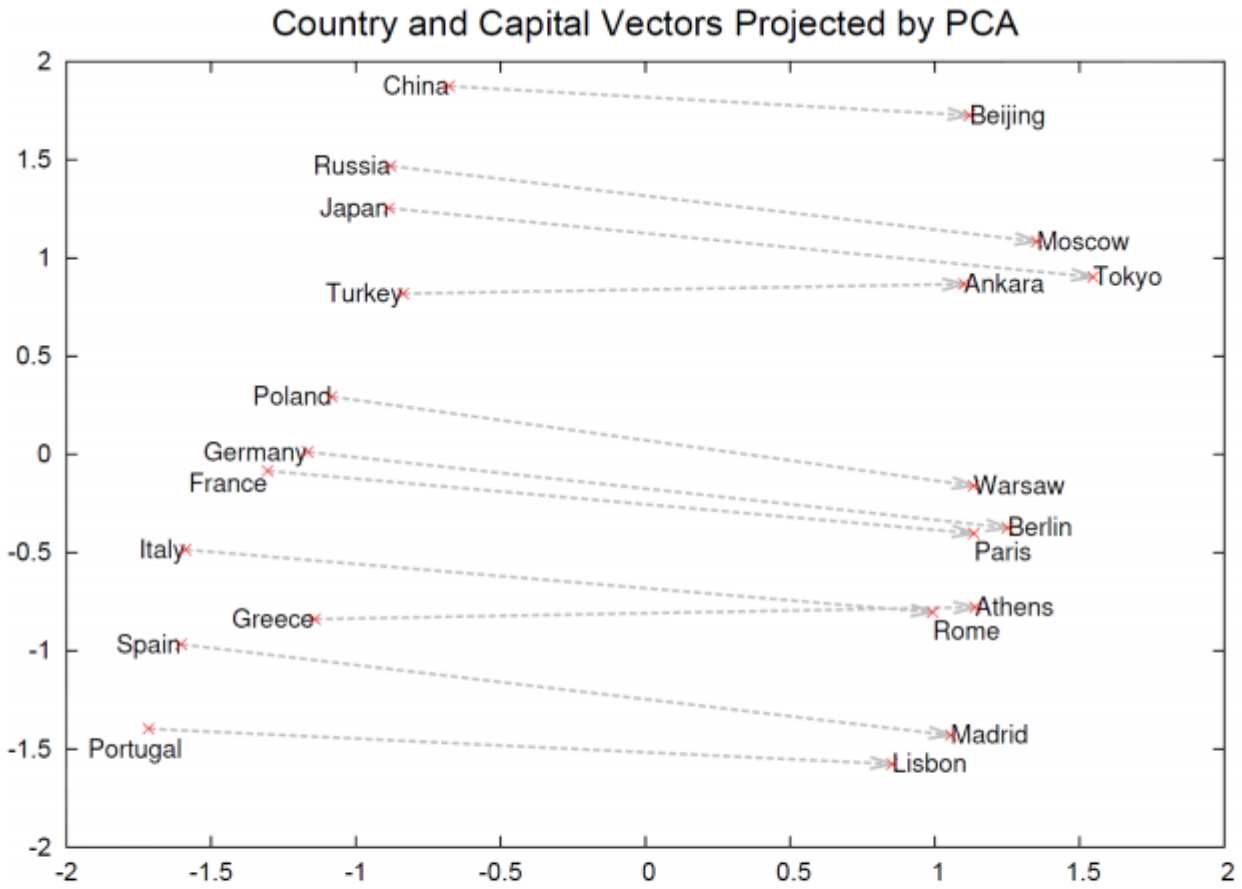
https://lamyiowce.github.io/word2viz/





About 90% reported accuracy (Mikolov et al. 2013c)

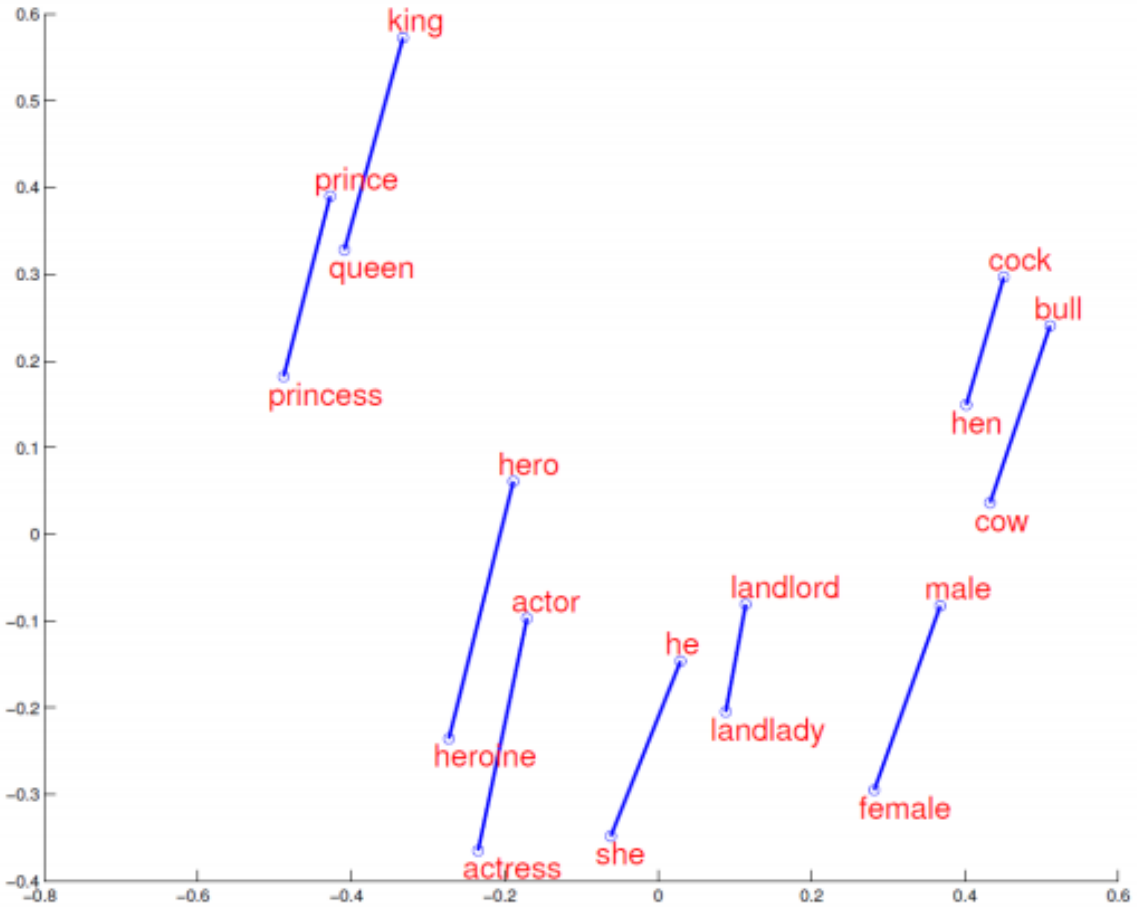
Mikolov, T., Le, Q. V., & Sutskever, I. (2013). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168.*



regularities between words are encoded in the difference vectors
e.g., there is a constant **country-capital** difference vector

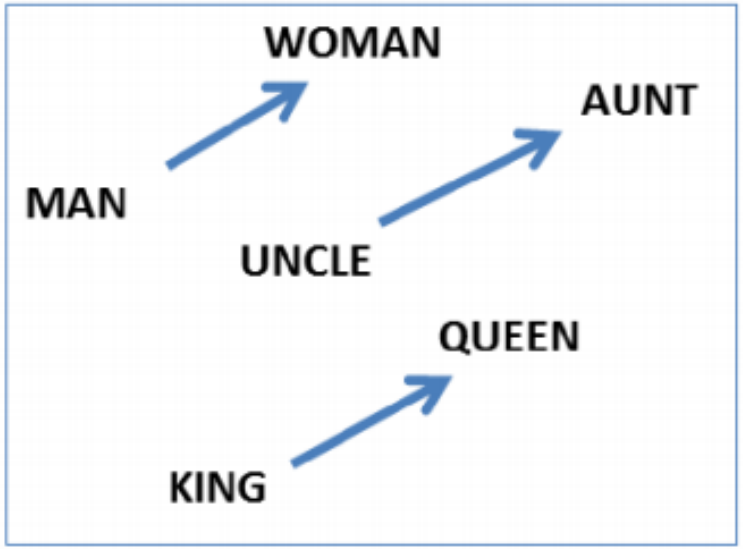
30

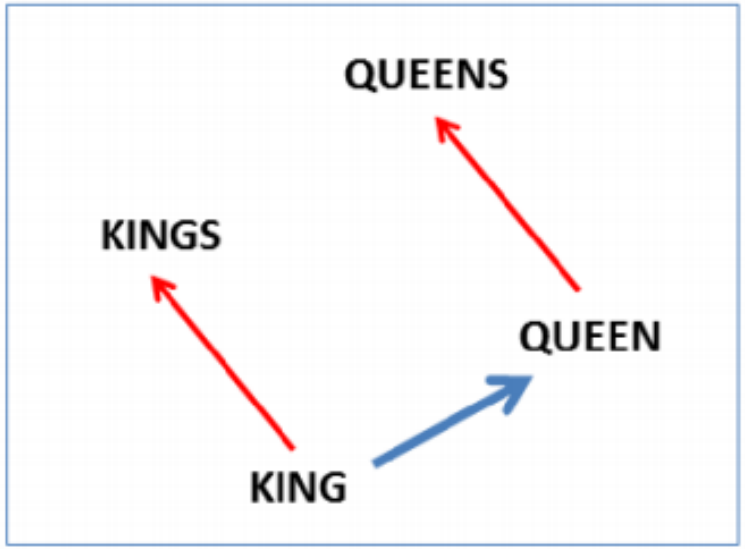
Mikolov et al. (2013b)
Distributed representations of words and phrases and their compositionality



constant **female-male** difference vector

http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd





constant **male-female** difference vector

constant **singular-plural** difference vector

- Vector operations are supported and make intuitive sense:

$$w_{king} - w_{man} + w_{woman} \cong w_{queen}$$

$$w_{einstein} - w_{scientist} + w_{painter} \cong w_{picasso}$$

$$w_{paris} - w_{france} + w_{italy} \cong w_{rome}$$

$$w_{his} - w_{he} + w_{she} \cong w_{her}$$

$$w_{windows} - w_{microsoft} + w_{google} \cong w_{android}$$

$$w_{cu} - w_{copper} + w_{gold} \cong w_{au}$$

- Online demo (scroll down to end of tutorial)

http://rare-technologies.com/word2vec-tutorial/

- Les mots ayant une signification proche sont représentés par des vecteurs similaires
- Les mots qui ont des vecteurs similaires présentent des analogies : synonymes, antonymes, noms, couleurs, lieux, …
- Arithmétique vectorielle permettant d'obtenir des analogies
- Les coordonnées sont pré-calculées (accessibles librement)

- Comment représenter des documents ? Somme des vecteurs des mots ? Voir l'algorithme de « word mover's »

Texte : mot, document, corpus
Nécessité de formater les données non structurées

Pour vectoriser des documents :

1. **Famille des méthodes TF-IDF**
2. **LSA : TF-IDF suivi d'une réduction via SVD**

Pour vectoriser des mots :

1. **Word embeddings : word2vec (visualisation t-SNE pour les embeddings)**

Levy, O., Goldberg, Y., & Dagan, I. (2015) :

- **LSA performe mieux lorsqu'on recherche de la similarité**
- **Word2Vec performe mieux lorsqu'on cherche des analogies**
- **Tuner les hyperparamètres est important**