Corpus, ressources et linguistique outillée

Traitement et Modélisation des Représentations Textuelles

Dr Wiem Takrouni
Sorbonne Université
Master Langue et Informatique
UFR Sociologie et Informatique pour les Sciences Humaines

April 4, 2025

Overview

Introduction aux Représentations Textuelles

Pourquoi représenter un texte sous forme numérique ?
Problème : Les modèles d'IA ne comprennent pas le texte brut.
Objectif : Transformer le texte en un format exploitable par les algorithmes.

Méthodes principales :

- Basé sur les fréquences : **Sac de mots (BoW)**, **TF-IDF**
- Basé sur les plongements : **Word Embeddings (Word2Vec, FastText, BERT)**

Introduction aux Représentations Textuelles

Limitations des représentations classiques

1. Sac de mots (BoW)

- Représente un texte par la fréquence des mots.
- Formule :

$$V = (n_1, n_2, ..., n_m)$$

2. TF-IDF (Term Frequency - Inverse Document Frequency)

- Pondération des mots rares dans un corpus.
- Formule :

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log\left(\frac{N}{\text{DF}(t)}\right)$$

Limitations :

- Ne prend pas en compte **l'ordre des mots**.
- Ne capture pas la **sémantique** (*"chat"* *"félin"*).

Avantages des Plongements de Mots

Pourquoi utiliser des embeddings ?

- Capturent la **relation sémantique** entre les mots.
- Produisent des **vecteurs continus** au lieu de matrices creuses.
- Intègrent le **contexte** (ex : *BERT est contextuel*).

Propriété fondamentale :

- Les mots similaires sont **proches dans l'espace vectoriel**.
- On peut effectuer des opérations **arithmétiques sémantiques** :

$$\text{Roi} - \text{Homme} + \text{Femme} \approx \text{Reine}$$

Introduction aux Représentations Textuelles

Exemple de représentation d'un mot dans différents modèles
Comparaison des différentes représentations **Exemple : Représentation du mot "chat"**

Modèle	**Représentation du mot "chat"**
BoW	[0, 1, 0, 0, 0, 1, 0, 0]
TF-IDF	0.07 (pondération selon corpus)
Word2Vec	[0.12, -0.34, 0.56, ..., 0.89] (300D)
BERT	[0.23, 0.01, -0.12, ..., -0.45] (contextualisé)

Différences :

- BoW et TF-IDF donnent une **matrice creuse de grande taille**.
- Word2Vec et BERT produisent des **vecteurs denses**.
- **BERT est contextuel** : le mot "chat" varie selon la phrase.

Définition et Objectifs des Plongements de Mots

Définition : Les plongements de mots (*word embeddings*) sont des représentations **vectorielles continues** qui capturent la sémantique des mots.

Objectifs :

- Transformer le texte en un format exploitable par l'IA.
- Capturer les relations **syntaxiques et sémantiques** entre les mots.
- Réduire la dimensionnalité par rapport à BoW ou TF-IDF.

Word2Vec – Skip-gram vs CBOW

Principe : Word2Vec apprend des représentations en prédisant le contexte des mots.

Deux architectures :

- **CBOW (Continuous Bag of Words)** : Prédit un mot à partir de son contexte.

$$P(w_t | w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$$

- **Skip-gram** : Prédit le contexte à partir d'un mot.

$$P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} | w_t)$$

FastText – Gestion des morphologies et langues rares

Limite de Word2Vec :

- Il ne capture pas la **structure morphologique** des mots.
- Il ne gère pas bien les mots rares ou inconnus.

Solution : FastText

- Décompose les mots en **n-grammes**.
- Exemple :

$$\text{"chat"} \rightarrow [\text{"ch"}, \text{"cha"}, \text{"hat"}, \text{"at"}]$$

- Améliore la gestion des **néologismes et mots rares**.

BERT : Un Modèle Contextuel

Problème des modèles classiques :

- Word2Vec et FastText génèrent des **représentations fixes**.
- Ils ne tiennent pas compte du **contexte**.

BERT (Bidirectional Encoder Representations from Transformers)

- Apprend une représentation **contextuelle** des mots.
- Prend en compte le contexte à gauche et à droite (**bidirrectionnel**).
- Exemple :
  - **"La banque du fleuve"** $\rightarrow$ Banque = Rivière
  - **"La banque financière"** $\rightarrow$ Banque = Institution

Différences entre Word2Vec, FastText et BERT
Comparaison des Modèles de Plongement

Modèle	**Représentation**	**Gère la Morphologie**	**Contextuel**
Word2Vec	Fixe	Non	Non
FastText	Fixe	Oui	Non
BERT	Variable	Oui	Oui

Principaux constats :

- Word2Vec est simple mais ne gère pas la morphologie.
- FastText est utile pour les langues morphologiquement riches.
- BERT est le plus puissant, mais coûteux en calcul.

Exemple : Différences entre Word2Vec, FastText et BERT
Phrase 1 : *"Il a déposé son argent à la banque."*
Phrase 2 : *"Le bateau longe la banque du fleuve."*

Résultat des modèles :

- **Word2Vec** : Même vecteur pour "banque" dans les deux phrases.
- **FastText** : Pareil que Word2Vec, mais meilleure gestion des mots rares.
- **BERT** : Différents vecteurs pour "banque" selon la phrase.

Expérimentation : Générer des Plongements de Mots

Implémentation de Word2Vec avec Gensim
Objectif : Apprendre des représentations vectorielles à partir d'un corpus.

Étapes principales :

1. Préparer un corpus de texte.
2. Tokenizer et nettoyer les phrases.
3. Entraîner Word2Vec avec la librairie `Gensim`.
4. Sauvegarder et explorer les vecteurs appris.

Code d'implémentation : [language=Python] from gensim.models import Word2Vec
sentences = [["chat", "aime", "poisson"], ["chien", "aime", "os"], ["oiseau", "vole", "ciel"]]
model = Word2Vec(sentences, $vector_size = 100, window = 5, min_count = 1, workers = 4)model.save("word2vec.model")$

Résultat : Un modèle Word2Vec générant des plongements de mots.

Expérimentation : Générer des Plongements de Mots

Exploration des voisins d'un mot avec Word2Vec
Objectif : Trouver les mots les plus proches d'un mot donné.

Code d'exploration : [language=Python] model = Word2Vec.load("word2vec.model")
$similar_words = model.wv.most_similar("chat", topn = 5) print(similar_words)$

Résultat attendu :

- Affichage des 5 mots les plus similaires à "chat".
- Exemple : `[("chien", 0.85), ("félin", 0.80), ("animal", 0.78)]`.

Visualisation avec Matplotlib : [language=Python] from sklearn.decomposition import PCA import matplotlib.pyplot as plt
words = ["chat", "chien", "poisson", "oiseau"] vectors = [model.wv[word] for word in words]
$pca = PCA(n_components = 2) reduced_vectors = pca.fit_transform(vectors)$
$plt.scatter(reduced_vectors[:, 0], reduced_vectors[:, 1]) for i, word in enumerate(words)$:

Expérimentation : Générer des Plongements de Mots

Exercice : Tester les Plongements sur un Corpus de Critiques de Films
Objectif : Entraîner Word2Vec sur un vrai corpus et analyser les résultats.

Consigne :

- Télécharger un corpus de critiques de films.
- Nettoyer et tokeniser les textes.
- Entraîner un modèle Word2Vec.
- Explorer les voisins sémantiques de certains mots-clés comme "film" ou "acteur".

Exemple de corpus (IMDB) : [language=Python] from datasets import
$\text{load}_d ataset dataset = load_d ataset("imdb", split = "train") sentences =$
$[review.split() for review in dataset["text"]]$
model = Word2Vec(sentences,
$\text{vector}_s ize = 100, window = 5, min_c ount = 2, workers = 4)$
Analyse des résultats :

- Quels sont les mots les plus proches de "film" ?

Expérimentation : Générer des Plongements de Mots

Analyse et Comparaison des Plongements

Évaluation de la qualité des plongements :

- **Cohérence sémantique** : Les mots proches doivent être liés.
- **Visualisation** : PCA ou t-SNE pour voir les groupes de mots.
- **Tâche de classification** : Tester les vecteurs sur une tâche NLP.

Comparaison avec BERT :

- Word2Vec génère des représentations **statiques**.
- BERT produit des vecteurs **contextuels**.
- Exercice : Tester les plongements de BERT avec la librairie `transformers`.

Clustering et Classification de Textes

Introduction au Clustering de Textes
Objectif : Regrouper des textes similaires sans supervision.

Applications :

- Regroupement de documents (articles, avis clients, tweets).
- Détection de thèmes dans un corpus.
- Filtrage d'informations et recommandation de contenus.

Problèmes rencontrés :

- Choix des représentations (TF-IDF, Word2Vec, BERT).
- Définition de la distance entre textes.
- Nombre optimal de clusters (k dans K-Means).

Algorithmes de Clustering

1. K-Means : Partitionne les données en k groupes en minimisant :

$$J = \sum_{i=1}^{k} \sum_{x_j \in C_i} ||x_j - \mu_i||^2$$

où μ_i est le centre du cluster.

2. DBSCAN : Regroupe les points denses et détecte les outliers.

- Paramètres : ϵ (rayon) et *minPts* (min. de voisins).
- Avantage : Pas besoin de définir k.

Comparaison :

Algorithme	Avantages	Inconvénients
K-Means	Rapide, simple	Doit fixer k
DBSCAN	Gère outliers	Sensible aux paramètres

Distances et Mesures de Similarité pour les Textes

1. Distance Cosinus :

$$\text{sim}(A, B) = \frac{A \cdot B}{||A||||B||}$$

Mesure l'angle entre deux vecteurs.

2. Distance Euclidienne :

$$d(A, B) = \sqrt{\sum_i (A_i - B_i)^2}$$

Adaptée aux plongements de mots.

3. Distance Jaccard :

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Utilisée pour des représentations en sac de mots.

Exemple : Clustering avec Word2Vec

1. Entraîner Word2Vec sur un corpus : [language=Python] from gensim.models import Word2Vec sentences = [["chat", "chien", "animal"], ["voiture", "moto", "transport"]] model = Word2Vec(sentences, $vector_size = 50, window = 5$)

2. Appliquer K-Means sur les vecteurs de mots : [language=Python] from sklearn.cluster import KMeans import numpy as np
words = ["chat", "chien", "voiture", "moto"] vectors = np.array([model.wv[word] for word in words])
kmeans = KMeans($n_clusters = 2$)$labels = kmeans.fit_predict(vectors)$

Clustering et Classification de Textes

Introduction à la Classification de Textes
Objectif : Prédire la catégorie d'un texte donné.

Applications :

- Classification des emails (spam / non spam).
- Analyse de sentiment (positif / négatif).
- Détection de sujets (sport, politique, science).

Types de modèles :

- Modèles classiques : SVM, Random Forest.
- Réseaux de neurones : LSTM, BERT.

Modèles Classiques : SVM et Random Forest

Support Vector Machine (SVM) :

$$f(x) = w \cdot x + b$$

- Bon sur petits datasets.
- Sépare linéairement les classes.

Random Forest :

- Ensemble de plusieurs arbres de décision.
- Moins sensible au bruit.

Fine-Tuning de BERT pour la Classification
Objectif : Adapter BERT à une tâche de classification.

Code d'entraînement sur des tweets : [language=Python] from transformers import BertTokenizer, BertForSequenceClassification import torch
model =
BertForSequenceClassification.from$_p$*retrained*("*bert* − *base* − *uncased*")*tokenizer* =
BertTokenizer.*from*$_p$*retrained*("*bert* − *base* − *uncased*")
inputs = tokenizer("Ce film est génial !",
return$_t$*ensors* = "*pt*")*outputs* = *model*(∗ ∗ *inputs*)

Exemples Concrets et Comparaisons

Comparaison des méthodes :

Modèle	Précision	Vitesse
SVM	85%	Rapide
BERT	92%	Lent

Exercice : Classifier des tweets avec un modèle pré-entraîné.

Exploration des Représentations Vectorielles en TP

Chargement des Embeddings
Objectif : Explorer les représentations vectorielles des mots.

Types d'embeddings :

- Word2Vec (sémantique statique)
- FastText (gestion des sous-mots)
- BERT (représentation contextuelle)

Exemple de chargement d'un modèle pré-entraîné : [language=Python] from gensim.models import KeyedVectors model = KeyedVectors.load$_{w}ord2vec_{f}ormat$("$GoogleNews - vectors - negative300.bin$", $binary = True$)

Exploration des Représentations Vectorielles en TP

Exploration des Vecteurs des Mots

Exemple : Comparer les mots proches dans Word2Vec [language=Python]

print(model.most$_s$*imilar*("*banque*"))

Résultat attendu :

- banque -¿ [("finance", 0.85), ("argent", 0.83), ("prêt", 0.79)]
- maison -¿ [("appartement", 0.82), ("bâtiment", 0.76)]

Pourquoi ces résultats ? Analyse sémantique et contexte.

Exploration des Représentations Vectorielles en TP

Réduction de Dimension : PCA / t-SNE
Problème : Les embeddings sont de grande dimension (300D avec Word2Vec).

Solution :

- PCA (Analyse en Composantes Principales) : préserve la variance.
- t-SNE : meilleur pour visualiser les relations locales.

Code : Réduction PCA + t-SNE [language=Python] from sklearn.decomposition import PCA from sklearn.manifold import TSNE
pca = PCA(n$_c$*omponents* = 50).*fit*$_t$*ransform*(*vectors*)*tsne* = *TSNE*(n_c*omponents* = 2).*fit*$_t$*ransform*(*pca*)

Exploration des Représentations Vectorielles en TP

Visualisation des Embeddings
Code : Affichage des mots en 2D [language=Python] import matplotlib.pyplot as plt plt.scatter(tsne[:,0], tsne[:,1]) for i, word in enumerate(words): plt.annotate(word, (tsne[i,0], tsne[i,1])) plt.show()

Problème : Peut-on séparer les groupes de mots ?

Exploration des Représentations Vectorielles en TP

Analyse des Clusters de Mots

Exemple : Groupement de mots similaires

Finance : "banque", "argent","économie"

Transport : "voiture", "train", "avion"

Pourquoi certains mots sont mal classés ?

- Influence du corpus d'entraînement.
- Modèle utilisé (statique vs contextuel).

Application Avancée : Fine-tuning de BERT pour une Tâche Spécifique

Pourquoi Fine-Tuner un Modèle Pré-entraîné ?
Problème : BERT est entraîné sur un large corpus généraliste.

Solution : Adapter BERT à une tâche spécifique :

- Classification d'émotions.
- Analyse de sentiments.
- Détection de fake news.

Application Avancée : Fine-tuning de BERT pour une Tâche Spécifique

Pipeline de Fine-Tuning avec Transformers
Outils : Bibliothèque `Transformers` de Hugging Face.

Code : Charger un modèle pré-entraîné [language=Python] from transformers import BertTokenizer, BertForSequenceClassification model = BertForSequenceClassification.from$_p$*retrained*("*bert* − *base* − *uncased*")*tokenizer* = *BertTokenizer*.*from*$_p$*retrained*("*bert* − *base* − *uncased*")

Application Avancée : Fine-tuning de BERT pour une Tâche Spécifique

Exemple : Classification de Sentiments
Objectif : Classifier des avis clients en "positif" ou "négatif".

Exemple de classification : [language=Python] text = "Ce film est excellent !" inputs = tokenizer(text, return$_t ensors = "pt")outputs = model(**inputs)$

Exercice : Tester sur plusieurs avis clients !

Application Avancée : Fine-tuning de BERT pour une Tâche Spécifique

Bonus : Mini-Projet Étudiant

Idées de projets avec fine-tuning de BERT :

- Détection d'émotions dans les tweets.
- Classification de fake news.
- Recommandation d'articles par sujet.

Objectif : Appliquer le fine-tuning sur un dataset personnalisé.