**GENERALITE SUR LES TECHNIQUES DE WORD EMBEDDING**

1. **DEFINITION**

Pour traiter les données textuelles en NLP (Natural Language Processing), il faut une représentation mathématique simple à même d’être comprise par la machine. Le **word embedding** (ou plongement lexical), est un ensemble de techniques permettant de transformer un mot ou ensemble de mot sous forme de vecteur. Plus clairement, le Word Embedding consiste à représenter les mots sous forme de vecteurs numériques dans un espace continu, où les mots capture leurs similarités et leurs relations. Des mots de sens proche sont alors proches dans cet espace. Par exemple, « **chat »** et « c**hien** » sont souvent représentés par des vecteurs proches car ils ont des contextes similaires. Ces vecteurs capturent la sémantique des mots (genre, synonymie, analogies comme « roi-homme + femme = reine ») et sont utilisés dans des tâches comme la classification de texte.

1. **DIFFERENTES METHODES DE PLONGEMENT DE TEXTE**

Ici nous présentons les différentes méthodes de Word embedding largement utilisé en LLM (Large Language Models) pour améliorer les capacités de ces derniers à générer et à comprendre le langage humain. Nous nous concentrerons dans cette partie aux méthodes les plus utilisées dans des problématiques de classification de texte.

**II-1 Méthode anciennes**

**Pour ce qui est des méthodes anciennes de traitement de texte, nous avons :**

* **One-hot encoding (codage binaire) :** Chaque mot du vocabulaire est représenté par un vecteur de dimension égale à la taille du vocabulaire, contenant un 1 à la position correspondant au mot et 0 ailleurs. Par exemple, si le vocabulaire est {chien, chat, lion}, le mot « chat » serait codé [0, 1, 0]. Cette représentation est très simple et ne nécessite pas d’apprentissage. **Avantages** : facile à comprendre et à calculer, fonctionne pour tout mot connu. **Inconvénients** : vecteurs *très creux* et de haute dimension (un indice par mot)
* **TF-IDF** (Term Frequency - Inverse Document Frequency) : Chaque document est représenté par un vecteur où chaque composante est le produit *TF* du mot dans le document fois *IDF* du mot dans le corpus. Le **TF** (fréquence du terme) est le nombre d’occurrences du mot dans le document (souvent normalisé), et **l’IDF** = log (nombre total de documents / nombre de documents contenant le mot). Ainsi, un mot fréquent dans un document aura un TF élevé, mais si ce mot est très courant dans le corpus (exemple : « le », « et »), son IDF est faible, ce qui diminue son poids. **Avantages** : met en évidence les mots distinctifs et discount les mots très fréquents. Simple à calculer et efficace en pratique pour la catégorisation de documents. **Inconvénients** : reste un sac de mots (bag of words). En effet, pas de sémantiqueni de prise en compte de l’ordre des mots.

**II-1 Méthode modernes**

**Pour ce qui est des méthodes modernes, nous avons :**

* **Word2Vec** est une méthode populaire (2013) utilisant un réseau de neurones peu profond pour créer des embeddings. Il existe deux variantes principales : le **CBOW** (Continuous Bag of Words) et le **skip-gram**. Dans CBOW, le modèle voit un ensemble de mots de contexte et prédit le mot cible au centre. Dans skip-gram, au contraire, on donne le mot cible au réseau et il prédit les mots du contexte. Par exemple, avec la phrase « je mange une pomme », CBOW utiliserait « je » et « une » pour prédire « mange », alors que skip-gram prendrait « mange » pour prédire « je » et « une ». En pratique, CBOW est plus rapide à entraîner, mais skip-gram donne souvent de meilleurs résultats sur les mots rares. Ces modèles (CBOW/skip-gram) produisent finalement un vecteur (*embedding*) pour chaque mot (la sortie de la couche cachée du réseau). Ces vecteurs reflètent des similarités sémantiques (par exemple : « roi » est proche de « reine » que de « chien »). Pour la classification, on utilise ces embeddings comme caractéristiques : on entraîne Word2Vec sur un corpus (comme les descriptions des attaques cybers), puis, pour chaque nouveau descriptif d’attaque cyber, on calcule le vecteur du document en moyennant les vecteurs de ses mots. Ce vecteur moyen sert alors d’entrée à un classificateur pour juger du niveau de sévérité de l’attaque.
* **FastText (Facebook)** : Extension de Word2Vec qui intègre des *n*-grammes de caractères pour représenter chaque mot. Par exemple, le mot **« chatting »** sera décomposé en sous-mots comme « *chat*», « *hat*», « *att*», etc. Chaque mot est donc représenté comme la somme de vecteurs de ses ngrammes et du mot entier. **Avantages** : comme Word2Vec, c’est un modèle statique pré-entraîné (skip-gram ou CBOW), mais il **capture la morphologie** du mot. Ainsi FastText peut générer une embedding pour un mot rare ou jamais vu (*out-of-vocabulary*) en assemblant ses n-grammes. Cela aide pour les mots rares, dérivés ou mal orthographiés. Il est relativement rapide à entraîner et fonctionne bien sur de larges corpus. **Inconvénients** : vecteurs statiques (un mot a le même vecteur quel que soit le contexte). Ne saisit pas le sens polyseme dans la phrase. En classification, il fournit des embeddings de mots de qualité supérieure à Word2Vec dans des langues morphologiquement riches.
* **Doc2Vec (Paragraph Vector)** : c’est une extension de Word2Vec pour encoder des documents en vecteurs fixes (Le & Mikolov, 2014). L’idée est d’associer à chaque document un *ID* supplémentaire appris, qui agit comme un vecteur global du document. Par exemple, deux critiques de film « Le **film** était **excellent** » et « Le **film** était **mauvais** » auront des vecteurs documents distincts reflétant le ton général. **Avantages** : capture le contexte *global* du document, pas seulement la moyenne de mots. Les vecteurs résultants permettent souvent de mieux distinguer les catégories de documents. Des études montrent que Doc2Vec offre une bonne précision en classification de documents (catégorisation de news, sentiment, etc.). Il est particulièrement utile quand on veut comparer ou classifier des textes entiers. **Inconvénients** : modèle statique (un seul vecteur par document) et relativement complexe à entraîner. Les mots rares sont traités comme avec Word2Vec (pas de traitement spécial).
* **GloVe** (« Global Vectors » 2014) est une autre technique d’embedding qui combine des statistiques globales et locales. GloVe se base sur la **cooccurrence** des mots dans un grand corpus. On calcule une grande matrice où chaque case indique combien de fois deux mots apparaissent ensemble dans une fenêtre de texte, puis on effectue une factorisation pour en extraire des vecteurs. Concrètement, GloVe essaie de faire en sorte que le produit scalaire des vecteurs de deux mots soit proche de la probabilité de leur cooccurrence. Par exemple, si « chat » et « chien » apparaissent très souvent ensemble, leurs vecteurs seront ajustés pour être proches. Comme Word2Vec, GloVe produit un unique vecteur statique par mot (indépendamment du contexte spécifique), et on l’utilise de la même façon pour la classification.
* **BERT** (2018, Google) est un modèle contextuel basé sur l’architecture *transformer*. Son nom signifie « Bidirectional Encoder Representations from Transformers ». Contrairement à Word2Vec/GloVe (où chaque mot a un vecteur fixe), BERT prend en compte le contexte entier : un même mot aura des vecteurs différents selon la phrase. Il utilise un apprentissage auto-supervisé : pendant la préformation, certains mots du texte sont masqués et le modèle apprend à les prédire (« Masked LM »), et il apprend aussi à comprendre la relation entre deux phrases (« Next Sentence Prediction »). Grâce à cela, BERT génère des représentations *contextuelles* de chaque token, capturant des informations fines sur la syntaxe et la sémantique du passage. Par exemple, BERT peut différencier le sens de « banque » dans « banque d’un fleuve » vs « banque de prêt », car il a vu le mot dans de nombreux contextes lors de son pré-entraînement. BERT a révolutionné le NLP en 2018, établissant de nouveaux records sur de nombreuses tâches. Pour la classification de texte, on « affine » simplement BERT sur la tâche cible. Concrètement, on enlève la dernière couche de prédiction de mots et on ajoute une couche de sortie adaptée à la classification (par ex. deux neurones pour « sévère/attritionnel »), puis on poursuit l’entraînement sur un jeu annoté de critiques. Cette approche *fine-tuning* permet d’obtenir en général de très bonnes performances, souvent bien supérieures à un modèle classique, même avec peu de données annotées.
* **ELMo (Embeddings from Language Models)** : Vecteurs contextuels de mots développés par Peters et al. (2018). ELMo utilise un modèle de langage bi-directionnel (biLSTM sur caractères) pré-entraîné sur un large corpus. Chaque occurrence d’un mot est alors représentée par un vecteur dépendant de **toute la** **phrase**. Par exemple, le mot **« *banque* »** aura un vecteur différent dans « *Je vais à la banque » (sens* *financier) et «* Le **rivage** de la banque » (sens de rives) parce que le contexte change. **Avantages** : extrêmement expressif. Les embeddings ELMo sont *profondément contextuels,* ils capturent la syntaxe, la sémantique et la polysémie. En pratique, ajouter ELMo à un modèle de classification améliore nettement les performances. **Inconvénients** : modèle lourd et coûteux à entraîner. Les vecteurs doivent être calculés pour chaque phrase (pas de dictionnaire fixe de mots), ce qui demande plus de temps et mémoire. En classification, ELMo nécessite souvent un fine-tuning ou un couplage à un réseau séparé.

**II-3 TABLEAU COMPARATIF RECAPITULATIF**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Méthode | Type de  modèle | Contexte  pris en  compte | Mots  rares | Facilité  entraînement | Ressources | Efficacité  classification |
| One-hot | Fréquence  (codage  binaire) | Non | Faible | Facile | Léger | Faible |
| TF-IDF | Fréquence  pondérée  (bag-ofwords) | Non | Faible | Facile | Léger | Moyenne |
| Word2Vec | Statique  (réseau de  neurones) | Non | Faible | Moyenne | Moyen | Moyenne |
| Doc2Vec | Statique  (neuronal,  document) | Non | Faible | Moyenne | Moyen | Moyenne |
| GloVe | Statique  (réseau de  neurones) | Non | Faible | Moyenne | Moyen | Moyenne |
| FastText | Statique  (neuronal +  sous-mots) | Non | **Bonne** | Moyenne | Moyen | Moyenne |
| BERT | Contextuel  (transformer) | Oui | **Bonne** | Difficile | Élevé | **Haute** |
| ELMo | Contextuel  (biLSTM) | Oui | **Bonne** | Difficile | Élevé | **Haute** |

Ce tableau compare les méthodes selon leur nature (statique, contextuelle, fréquence), leur capacité à gérer le contexte et les mots rares, ainsi que leur complexité d’entraînement et leur pertinence pour la classification de texte. Les approches basées sur la fréquence (**One-hot, TF IDF**) sont simples mais peu expressives, tandis que les modèles contextuels modernes (**ELMo, BERT**) requièrent plus de ressources mais offrent des performances nettement supérieures en classification. Les modèles intermédiaires (**FastText, Word2Vec, GloVe, Doc2Vec**) représentent des compromis entre richesse sémantique et coût de calcul, et peuvent être entrainé sur les jeux de données de taille intermédiaire.

1. **MODELE LSTM ET CLASSIFICATION DE TEXTE**

Pour classifier les textes, les algorithmes suscités peuvent être utiliser. Cependant, il existe une famille de modèle dit modèle LSTM (Long short term memory) qui couplé aux algorithmes de word embedding et après fine tuning, peuvent nous fournir des résultats beaucoup meilleurs.

En effet, un **LSTM** est un type particulier de réseau de neurones récurrents (RNN) conçu pour traiter des données séquentielles (texte, son, séries temporelles). Contrairement aux RNN classiques, qui ont une mémoire courte et oublient vite le contexte lointain, les LSTM peuvent retenir des informations importantes sur le long terme. Ils contiennent une cellule mémoireet trois « portes » (entrée, oubli, sortie) qui filtrent l’information à chaque étape. En pratique, les portes (σ pour sigmoïde, tanh pour tangente hyperbolique) décident quelles informations on garde ou on jette dans la mémoire interne. On peut schématiser cela ainsi :



*Figure : Schéma simplifié d’une cellule LSTM. Les portes ( σ , tanh ) modulent le flux d’information pour conserver (ou oublier) le contenu de la mémoire sur plusieurs mots ou étapes de la séquence*

Cette architecture innovante permet aux LSTM de **garder le fil** d’une séquence : par exemple, en lisant une phrase mot à mot, un LSTM peut « se souvenir » que le début de la phrase parlait d’un voyage, et prédire la fin de la phrase en conséquence. Une analogie imagée compare le LSTM à *« un lecteur* *ultra concentré qui surligne les phrases clés, prend des notes et fait le lien entre passé et futur »*, contrairement à un RNN classique qui « oublie rapidement ».

**LSTM pour la classification de texte**

Pour classer du texte (par exemple détecter le sentiment d’un avis ou la catégorie d’un article), on peut utiliser un réseau à base de LSTM. La chaîne typique est la suivante :

**(Texte brut → Tokens → Vecteurs) = word embedding** → *LSTM* → *sortie de classification*.

Concrètement :

* **Tokenisation** : on découpe le texte en mots ou sous-mots (tokens) et on les convertit en entiers.
* **Couche d’embedding** : chaque mot est transformé en un vecteur dense (embedding) de dimension fixée (par exemple 100 ou 300 dimensions). Cette couche peut être entraînée avec le modèle ou initialisée par des embeddings pré-entraînés (Word2Vec, GloVe, etc.).
* **Couche LSTM** : le LSTM parcourt la séquence de vecteurs (dans l’ordre des mots). Il produit un vecteur caché à chaque étape, qui intègre l’information du mot courant et du contexte passé (mémoire interne). On peut utiliser un **LSTM bidirectionnel** (BiLSTM) qui lit la séquence dans les deux sens pour capter mieux le contexte global (voir **ELMo)**
* **Couches de sortie** : on prend souvent l’état caché final de la (Bi)LSTM, puis des couches denses (fully connected) pour prédire la classe. Par exemple, en analyse de sentiment binaire (positif/négatif), on termine par une couche Dense avec activation sigmoïde.

Toutefois, il existe d’autre approches de classification consistant à représenter simplement chaque description par un vecteur TF-IDF puis à utiliser un classifieur traditionnel. Bien que simpliste, ces modèles peuvent dans certains cas donner des résultats intéressants. Par exemple, un pipeline scikit-learn ***CountVectorizer → TfidfTransformer → SVM.***

Nous pouvons également penser aux modèles Transformers (BERT, DistilBERT, TinyBERT, ALBERT) qui peuvent être finement ajustés pour des tâches de classification. Bien que performant dans la plupart des cas, ces modèles nécessitent de grand volume de données et sont très gourmand en calcul et mémoire, et souvent pré-entrainé sur les bases qui ne reflètent pas particulièrement le sujet sur lequel on travail.

Nous ne terminerons pas sans évoquer les variantes du modèle BERT (**SecBERT, SecureBERT)** pré-entraînés spécifiquement sur du texte de cyber-sécurité. Ces derniers possèdent un vocabulaire et une connaissance du jargon du domaine, ce qui améliore souvent la qualité du prétraitement des textes. Cependant, leur taille est prohibitive pour un entraînement local (des dizaines à centaines de M paramètres). On peut toutefois les employer sans fine-tuning (seulement en extraction de features).