Large Language Models

Code du cours – ADIA94

Stefani EL KALAMOUNI

CM1 – Introduction aux LLMs

01

Rappel de Concepts de Base

Rappel

1. Bases du Machine Learning et de l'IA

1. Qu'est ce qu'un modèle de machine Learning?
2. Quels sont les modèles supervisés et non supervisés?

2. Notions de NLP (Natural Language Processing)

1. Que savez-vous du traitement automatique du langage (NLP) ?
2. Avez-vous déjà utilisé des outils ou bibliothèques de NLP, comme NLTK ou SpaCy ?
3. Quels sont les processus de traitement de texte en NLP ?

Qu’est ce qu’un modèle de machine Learning?

Un modèle de Machine Learning est un programme informatique qui apprend à réaliser des tâches spécifiques en analysant des données, sans qu’on lui ait explicitement indiqué comment le faire. Plutôt que de suivre des instructions fixes, il "apprend" en détectant des motifs dans les données.

Par exemple, pour reconnaître des images de chats, un modèle de Machine Learning étudie des milliers de photos de chats et d’autres objets. Avec le temps, il apprend les caractéristiques d’un chat (comme les oreilles pointues, la fourrure, etc.) et peut identifier des chats dans de nouvelles images.
Le modèle est constitué de paramètres, que l’algorithme ajuste pendant l’apprentissage pour minimiser les erreurs, améliorant ainsi la précision du modèle.

Quels sont les modèles supervisés et non supervisés ?

Modèles supervisés :
Dans l’apprentissage supervisé, le modèle apprend à partir de données qui sont déjà **étiquetées**, c'est-à-dire que chaque exemple d'entraînement est accompagné de la réponse correcte.
Par exemple, pour un modèle qui classifie les emails comme "spam" ou "non-spam", on fournit un ensemble d’emails avec leur étiquette correspondante (spam ou non-spam).
Le modèle apprend donc la relation entre les emails et les étiquettes pour pouvoir prédire de nouvelles données.

Modèles non supervisés :
Dans l’apprentissage non supervisé, le modèle travaille avec des données **non étiquetées**, c'est-à-dire qu’il doit lui-même trouver des motifs ou des groupes (appelés **clusters**) dans les données sans qu’on lui indique de réponse correcte.
Par exemple, un modèle non supervisé pourrait analyser des comportements d'utilisateurs et trouver des groupes d'utilisateurs aux comportements similaires, ce qui aide les entreprises à mieux cibler leurs offres.

Qu’est ce que c’est le traitement automatique du langage (NLP) ?

C’est une branche de l’intelligence artificielle qui permet aux ordinateurs de comprendre, générer et analyser le langage humain. Le NLP regroupe diverses tâches comme la traduction automatique, l'analyse de sentiments, la classification de texte et la génération de texte.

Objectif : Faciliter l’interaction entre les humains et les machines en permettant aux ordinateurs de traiter le langage de manière fluide et naturelle.

Exemples courants de NLP :
Assistants virtuels comme Siri ou Alexa, qui interprètent les commandes vocales.
Traduction automatique (Google Translate).
Analyse de sentiments pour déterminer l’émotion d'un texte (positif, négatif, neutre).

NLP: Bibliothèques NLTK & SpaCy ?

NLTK (Natural Language Toolkit) : NLTK est une bibliothèque en Python qui fournit des outils pour traiter et analyser le texte, utile pour la tokenisation, l'analyse syntaxique, et d’autres tâches de base en NLP.

SpaCy : SpaCy est une autre bibliothèque populaire en Python, qui est optimisée pour les performances et fournit des fonctionnalités avancées comme l'analyse des dépendances, la reconnaissance des entités nommées (par exemple, pour identifier les noms de personnes ou de lieux), et le lemmatisation (réduction des mots à leur forme de base).

Ces bibliothèques simplifient énormément le travail en NLP et sont largement utilisées dans l’industrie pour créer des applications robustes.

Processus de traitement de texte en NLP

1. Nettoyage des données

Suppression de ponctuation : Éliminer les signes de ponctuation non pertinents, qui pourraient ajouter du bruit au modèle.

Conversion en minuscules : Unifier le texte en minuscules pour que "Chat" et "chat" soient traités de la même manière.

Suppression des caractères spéciaux et des chiffres : Enlever les caractères qui n'apportent aucune valeur à l'analyse, selon le contexte.

Obligatoire ? Pas toujours, mais très souvent nécessaire.

Pourquoi ? Le nettoyage du texte est généralement la première étape pour réduire le bruit et simplifier les données.

Quand l'utiliser ? Utilisé presque toujours, notamment pour rendre les textes homogènes et éviter que des différences de casse ou de ponctuation perturbent les résultats du modèle.

Processus de traitement de texte en NLP

2. **Tokenisation**

C’est le processus qui consiste à diviser un texte en unités plus petites appelées tokens. Les tokens peuvent être des mots individuels, des sous-mots, ou même des caractères, selon les besoins.

Les modèles de NLP comprennent le texte en manipulant des tokens, et non pas des phrases entières. La tokenisation transforme ainsi un texte brut en un format que le modèle peut analyser.

Par exemple, dans la phrase "Les chats dorment souvent", la tokenisation pourrait produire les tokens :

["Les", "chats", "dorment", "souvent"].

Cela permet au modèle de traiter chaque mot individuellement pour mieux comprendre le contexte et le sens.

Obligatoire ? Oui, pratiquement toujours.

Pourquoi ? La tokenisation découpe le texte en unités plus petites (mots ou phrases), ce qui permet au modèle d’analyser le texte en éléments exploitables.

Quand l’utiliser ? La tokenisation est une étape de base en NLP et est utilisée pour presque toutes les tâches, car elle prépare le texte pour le modèle.

Processus de traitement de texte en NLP

3. Suppression des mots vides (Stop Words)

Les mots vides (stop words) sont des mots très fréquents mais qui n’apportent souvent pas de signification importante pour certaines analyses (comme "le", "de", "et" en français). Les retirer peut améliorer l'efficacité du modèle, bien que certains modèles récents conservent ces mots pour mieux **comprendre** le contexte

Obligatoire ? Non.

Pourquoi ? Les mots vides sont des mots très fréquents qui apportent peu de sens en termes de contenu (comme "le", "de", "et"). Les supprimer peut rendre l'analyse plus efficace.

Quand l’utiliser ? Utilisée principalement pour des tâches où l’on cherche à extraire des informations sémantiques (ex : analyse de sentiments, classification de texte). Pour les modèles de type Transformer (comme BERT ou GPT), cette étape peut être facultative car le modèle peut apprendre à ignorer les mots peu pertinents.

Processus de traitement de texte en NLP

4. Lemmatisation

Réduire chaque mot à sa forme canonique ou lemme (ex. : "manges", "mangeons", "mangé" deviennent "manger").

Cela conserve le sens des mots et améliore la cohérence.

Obligatoire ? Non, la lemmatisation n'est pas toujours obligatoire, mais elle est souvent recommandée lorsque le but est de **comprendre le sens précis des mots**. Elle est particulièrement utile lorsque l'analyse requiert que le mot soit sous sa forme correcte, comme pour éviter les doublons sémantiques (ex. : "manger" et "mange").

Pourquoi ? La lemmatisation ramène chaque mot à son **lemme** (forme de base) en utilisant des règles linguistiques et en tenant compte de la signification du mot. Contrairement au stemming, la lemmatisation conserve un mot compréhensible en langue naturelle (par exemple, "ran" devient "run" et non "rn").

Elle aide à **réduire les mots inflectés** (pluriels, conjugaisons) tout en maintenant le sens des mots, ce qui est crucial pour les modèles qui cherchent à comprendre la sémantique.

Quand utiliser ? Utilisez la lemmatisation pour des **tâches sémantiques** où la signification exacte des mots est importante, comme

- **Classification de texte** : pour regrouper les variantes d'un même mot.
- **Analyse de sentiment** : pour traiter les mots sous une forme cohérente.
- **Recherche d'information et analyse de texte** : où il est essentiel que les mots soient représentés dans leur forme correcte.

Processus de traitement de texte en NLP

4. Lemmatisation et Stemming

Lemmatisation : Réduire chaque mot à sa forme canonique ou lemme (ex. : "manges", "mangeons", "mangé" deviennent "manger"). Cela conserve le sens des mots et améliore la cohérence.

Stemming : Réduire les mots à leur racine en supprimant les suffixes, sans toujours garantir une forme correcte (ex. : "manges" et "mangé" deviennent "mang"). Le stemming est plus rapide mais moins précis que la lemmatisation.

5. Reconnaissance des entités nommées (Named Entity Recognition - NER)

Cette étape identifie les noms propres, lieux, dates, organisations et autres informations spécifiques dans le texte. Par exemple, dans "OpenAI a été fondée en 2015", NER reconnaîtra "OpenAI" comme organisation et "2015" comme une date.

Processus de traitement de texte en NLP

5. Stemming

1.Stemming : Réduire les mots à leur racine en supprimant les suffixes, sans toujours garantir une forme correcte (ex. : "manges" et "mangé" deviennent "mang"). Le stemming est plus rapide mais moins précis que la lemmatisation.

Obligatoire ? Non, le stemming n'est pas obligatoire et peut parfois être omis si l'on veut conserver la forme exacte des mots. Il est utilisé principalement pour réduire les mots à leur racine sans forcément respecter la signification exacte du mot.

Pourquoi ? Le stemming est une technique plus simple et plus rapide que la lemmatisation, car il coupe les suffixes pour ramener le mot à sa **racine** (exemple : "running" devient "run").

Cette approche est moins précise que la lemmatisation, mais elle peut être suffisante pour certaines applications où une représentation brute de la racine des mots est suffisante.

Quand utiliser ? Utilisez le stemming pour des **tâches lexicales de base** où une précision totale n'est pas essentielle, par exemple :

1. **Recherche d'information de base** : comme pour récupérer des documents similaires en termes de mots racinaires.
2. **Analyse rapide de grandes quantités de données** : lorsque l'efficacité et la vitesse sont plus importantes que la précision.
3. **Réduction de vocabulaire** dans des cas où la compréhension précise du mot n'est pas nécessaire.

Processus de traitement de texte en NLP

6. Reconnaissance des entités nommées (Named Entity Recognition - NER)

Cette étape identifie les noms propres, lieux, dates, organisations et autres informations spécifiques dans le texte. Par exemple, dans "OpenAI a été fondée en 2015", NER reconnaîtra "OpenAI" comme organisation et "2015" comme une date.

Obligatoire ? Non.

Pourquoi ? La NER identifie des entités spécifiques (noms propres, lieux, dates) dans le texte, ce qui est utile pour certaines analyses spécifiques.

Quand l'utiliser ? Utilisée pour des tâches où il est important de savoir "qui" ou "quoi" est mentionné dans le texte (ex : extraction d'informations, analyses dans des contextes professionnels comme le médical, le financier, etc.).

Processus de traitement de texte en NLP

7. Part-of-Speech Tagging (POS Tagging)

Le POS Tagging consiste à attribuer à chaque mot une étiquette grammaticale (verbe, nom, adjectif, etc.). Cette information aide le modèle à comprendre le rôle de chaque mot dans la phrase.

Obligatoire ? Non.

Pourquoi ? L'étiquetage grammatical attribue une catégorie (nom, verbe, adjectif, etc.) à chaque mot, ce qui aide à comprendre la structure du texte.

Quand l'utiliser ? Utilisé pour des tâches d'analyse syntaxique ou pour des modèles qui exploitent la structure grammaticale (par ex. pour la traduction automatique ou l'analyse grammaticale).

Processus de traitement de texte en NLP

8. Chunking

Le chunking est une étape avancée qui regroupe les mots en phrases nominales ou verbales (par exemple, "le chat noir" devient une phrase nominale). Cette étape permet de capturer des groupes de mots cohérents pour une meilleure compréhension du contexte.

Obligatoire ? Non.

Pourquoi ? Le chunking regroupe les mots en blocs cohérents (ex : phrases nominales) et aide à capturer des structures de texte.

Quand l'utiliser ? Utilisé surtout dans des tâches nécessitant une compréhension détaillée de la structure du texte, comme le traitement de phrases complexes, les analyses syntaxiques avancées ou les résumés : (par exemple : Si votre méthode de résumé consiste à sélectionner des phrases ou des passages clés du texte source, le chunking peut aider à identifier des segments particulièrement informatifs. Idem dans des documents très structurés (comme des articles scientifiques ou des documents de loi), le chunking peut aider à isoler des termes ou groupes de mots importants qui devraient être présents dans le résumé.

En revanche, dans les approches modernes avec des LLMs (comme GPT ou BERT), le chunking est généralement **moins pertinent**, car ces modèles utilisent des mécanismes d'attention pour comprendre le contexte global et les relations entre les mots. Ils n'ont donc pas besoin de repères syntaxiques supplémentaires comme ceux offerts par le chunking.

En plus pour les textes simples ou courts où l'information est claire et directe, le chunking peut ne pas ajouter de valeur au résumé.

Processus de traitement de texte en NLP

9**. Encodage des mots en vecteurs (Word Embedding)**

Transformer les mots en vecteurs numériques pour qu'ils soient compréhensibles par le modèle. Des techniques comme Word2Vec, GloVe, et FastText génèrent des représentations de mots basées sur leur contexte.

Obligatoire ? Oui, pour la plupart des modèles modernes.

Pourquoi ? Les modèles de machine learning et deep learning nécessitent une représentation numérique des mots. Les embeddings permettent de transformer les mots en vecteurs en préservant leurs similarités sémantiques.

Quand l’utiliser ? Utilisé pour presque toutes les tâches NLP modernes nécessitant un modèle basé sur le deep learning (comme les modèles de classification ou de génération de texte).

Processus de traitement de texte en NLP

10 a. Construction de séquences et de contextes (sliding windows) Fenêtres Glissantes

Les fenêtres glissantes consistent à découper le texte en segments de longueur fixe, tout en tenant compte des mots voisins pour capturer le **contexte local**.

Par exemple, avec une fenêtre de taille 3 appliquée sur la phrase "Le chat noir dort paisiblement", on obtient des séquences de trois mots :

- ["Le", "chat", "noir"]
- ["chat", "noir", "dort"]
- ["noir", "dort", "paisiblement"]

Pourquoi l'utiliser ? : Chaque segment donne au modèle une idée de la relation immédiate entre les mots voisins, ce qui aide à capturer des associations sémantiques de proximité. C'est utile pour des modèles de type RNN (réseaux de neurones récurrents) ou dans des tâches simples comme la détection de motifs.

Quand l'utiliser ? : Les fenêtres glissantes sont souvent utilisées dans des modèles qui ne capturent pas naturellement le contexte global et ont besoin d'informations locales, comme dans les modèles de Markov ou les RNN basiques.

Processus de traitement de texte en NLP

10 b. Construction de séquences et de contextes (position encoding)

L'encodage de position est une méthode utilisée dans les modèles **Transformers** pour intégrer l'information de position des mots dans une phrase.

Comme les Transformers ne traitent pas les séquences dans un ordre spécifique, ils utilisent l'encodage de position pour indiquer la place de chaque mot dans la phrase. Chaque mot se voit attribuer un vecteur unique selon sa position, ce qui permet au modèle de comprendre la structure de la phrase.

Pourquoi l'utiliser ? : Dans les modèles comme les Transformers (BERT, GPT), qui n'ont pas de mémoire de position comme les RNN, l'encodage de position est essentiel pour capturer la séquence des mots.

Quand l'utiliser ? : Utilisé dans tous les modèles de type Transformer, car ils doivent comprendre l'ordre des mots pour interpréter correctement le contexte global et local.

Technique classique en NLP

Une des techniques classiques de traitement automatique du langage (NLP) c'est transformer du texte en vecteurs numériques, puis mesurer la similarité entre ces vecteurs afin de trouver une réponse appropriée.

TF-IDF Vectorisation

Le **TF-IDF** (*Term Frequency-Inverse Document Frequency*) est une méthode qui permet de représenter des documents textuels sous forme de vecteurs numériques. Voici comment cela fonctionne :

TF (Term Frequency) : Mesure la fréquence d'un mot dans un document. Plus un mot apparaît souvent, plus sa fréquence est élevée.

IDF (Inverse Document Frequency) : Mesure l'importance d'un mot à travers plusieurs documents. Si un mot est très fréquent dans beaucoup de documents, il est moins pertinent pour le document en question. L'IDF réduit donc l'importance de mots trop courants (comme "le", "de" en français).

Imaginons que vous avez plusieurs documents (ou phrases) et que vous voulez identifier les mots qui **représentent le mieux chaque document**. Certains mots comme **« le », « de », « et »** apparaissent fréquemment dans presque tous les documents. Ces mots n'apportent pas beaucoup d'informations sur le contenu spécifique d'un document car ils sont communs à tous. En revanche, des mots moins fréquents comme **« banque »** ou **« chat »** pourraient être plus représentatifs de certains documents spécifiques. L'IDF aide donc à **réduire l'importance des mots très fréquents** et **mettre en avant ceux qui apparaissent dans un nombre limité de documents**.

Technique classique en NLP

Comment l'IDF est-il calculé ?

La formule de l'IDF est la suivante $\mathrm{IDF}(t) = \log\left(\frac{N}{n_t}\right)$

t est le mot que l'on analyse.

N est le nombre total de documents/phrases dans l'ensemble.

nt est le nombre de documents/phrases contenant le mot t.

Exemple simple

Supposons que nous ayons un ensemble de 5 documents et que nous examinons les mots suivants :

Le mot **« le »** apparaît dans tous les documents (5 documents).

Le mot **« banque »** n'apparaît que dans 1 document.

Calcul de l'IDF pour chaque mot : $\mathrm{IDF}(\text{« le »}) = \log\left(\frac{5}{5}\right) = \log(1) = 0$

Cela signifie que **« le »** n'a pas de valeur informative particulière, car il apparaît dans tous les documents. Son importance est donc réduite à 0.

$$\mathrm{IDF}(\text{« banque »}) = \log\left(\frac{5}{1}\right) = \log(5) \approx 1.6$$

Ce score IDF plus élevé pour **« banque »** signifie que ce mot est plus informatif. Il est moins commun, donc il apporte une information unique sur le document où il apparaît.

Technique classique en NLP

Comment ça marche dans le code ? (voir solution TP1 question 9)

La ligne **X = vectorizer.fit_transform(lemmatized_sentences)** transforme chaque phrase (ou document ou token) de votre corpus en un vecteur numérique où chaque position dans le vecteur représente l'importance de chaque mot pour cette phrase.

Le résultat X est une matrice où chaque ligne est une phrase du corpus et chaque colonne représente un mot unique du corpus.

Transformer une requête utilisateur en vecteur

Lorsque l'utilisateur entre une question, celle-ci doit aussi être transformée en vecteur pour être comparée aux vecteurs des phrases du corpus.

user_input_processed = normalize_text(user_input) et lemmatized_input = lemmatize_sentence(user_input_processed) normalisent et lemmatisent la question de l'utilisateur.

user_input_vector = vectorizer.transform([lemmatized_input]) utilise le même vectorizer pour transformer la question en un vecteur TF-IDF. Ce vecteur a la même structure que ceux du corpus et peut être comparé avec eux.

Technique classique en NLP

Cosine Similarity (Similarité Cosinus)

La **similarité cosinus** est une mesure qui permet de déterminer le degré de similarité entre deux vecteurs. Elle est particulièrement adaptée pour comparer des vecteurs de texte, car elle mesure l'angle entre les vecteurs, et non leur magnitude.

Formule : La similarité cosinus entre deux vecteurs A et B est définie par : $$\text{cosine similarity} = \frac{A \cdot B}{||A|| \times ||B||}$$

Ou A.B est le produit scalaire des deux vecteurs, et ||A|| et ||B|| sont les normes (longueurs) des vecteurs.

Valeurs : Elle varie entre -1 et 1.

1 signifie que les vecteurs sont identiques en orientation.

0 signifie qu'ils sont orthogonaux (aucune similarité).

Une valeur proche de -1 indiquerait une orientation opposée (mais en NLP, cela arrive rarement avec des vecteurs TF-IDF).

Dans le code :

similarities = cosine_similarity(user_input_vector, X).flatten() calcule la similarité cosinus entre le vecteur de la question de l'utilisateur (user_input_vector) et chaque phrase du corpus (représentée par les lignes de la matrice X).

Le résultat similarities est un tableau de scores de similarité entre la question de l'utilisateur et chaque phrase du corpus.

Technique classique en NLP

Trouver la réponse appropriée

La ligne max_similarity_index = similarities.argmax() identifie l'indice de la phrase dans le corpus qui a la similarité la plus élevée avec la question de l'utilisateur.

Ensuite, le code vérifie si la similarité dépasse un certain seuil (0.1 dans cet exemple). Ce seuil évite de renvoyer une réponse si la question de l'utilisateur est trop différente de tout ce qui se trouve dans le corpus.

Pourquoi passer du texte au vecteur et utiliser la similarité cosinus ?

Pourquoi le vecteur TF-IDF ? : Transformer le texte en vecteur TF-IDF permet de capturer l'importance relative des mots dans chaque phrase et de comparer les phrases en fonction de leur contenu.

Pourquoi la similarité cosinus ? : Elle est une mesure adaptée pour déterminer la proximité entre deux phrases en termes de direction (ou contenu sémantique), indépendamment de leur longueur ou de leur fréquence de mots.

Technique classique en NLP

Exemple de transformation avec TF-IDF

Imaginons que nous ayons les phrases suivantes :

Phrase 1 : "Le chat dort"

Phrase 2 : "Le chien aboie fort"

Étape 1 : Créer le vocabulaire global

En utilisant un TfidfVectorizer, nous identifions tous les mots uniques (ou *tokens*) présents dans toutes les phrases. Dans cet exemple, notre vocabulaire global est constitué des mots :

"le", "chat", "dort", "chien", "aboie", "fort"

Chaque mot du vocabulaire global correspond à une **position spécifique** dans les vecteurs des phrases. Ainsi, chaque mot unique devient une colonne dans la matrice TF-IDF, et chaque phrase est représentée par une ligne dans cette matrice.

Technique classique en NLP

Étape 2 : Créer les vecteurs TF-IDF

Pour chaque phrase, on calcule le **TF-IDF** de chaque mot du vocabulaire global. Si un mot du vocabulaire n'apparaît pas dans la phrase, sa valeur TF-IDF sera 0 pour cette phrase.

Voici à quoi ressemble la matrice TF-IDF pour nos phrases, avec chaque colonne correspondant à un mot du vocabulaire :

	le	chat	dort	chien	aboie	fort
Phrase 1	TF-IDF(le)	TF-IDF(chat)	TF-IDF(dort)	0	0	0
Phrase 2	TF-IDF(le)	0	0	TF-IDF(chien)	TF-IDF(aboie)	TF-IDF(fort)

Même si **Phrase 1** et **Phrase 2** contiennent des nombres de mots différents, leurs vecteurs TF-IDF sont de même longueur car chaque vecteur a une dimension égale au nombre total de mots uniques dans le vocabulaire global.
Vocabulaire global : ['aboie' 'chat' 'chien' 'dort' 'fort' 'le']
Matrice TF-IDF :

	aboie	chat	chien	dort	fort	le
0	0.000000	0.577350	0.000000	0.577350	0.000000	0.577350
1	0.516397	0.000000	0.516397	0.000000	0.516397	0.447214

Même si chaque phrase a un nombre de mots différent, chaque vecteur est de même longueur (une dimension pour chaque mot unique du vocabulaire global). Les positions avec des valeurs de 0 correspondent aux mots absents dans cette phrase spécifique.