INALCO M2 Master ingénierie multilingue 2025
Ingénierie des connaissances : Contrôle n°1 : Correction

1) Quelles sont les limites des plongements lexicaux (embeddings) pour stocker les connaissances humaines ?

Les plongements lexicaux (embeddings) sont des outils dans divers applications de NLP tels que la recherche d'information, mais ils présentent un certain nombre de limitations :
* Manque d'adaptation à des domaines de spécialité (juridique, médical) conditionné par le manque de données spécialisées dans le corpus d'entraînement. Les termes techniques seront trop proches dans l'espace vectoriel latent pour que cela fonctionne bien, flou.
* problème de la mise à jour des connaissances (création de nouveaux termes, de nouvelles connexions)
* problème du multilinguisme. Même les modèles multilingues ne traitent correctement qu'un petit nombre de langues. Comment stocker dès lors les connaissances spécifiques liées à une aire géographique et culturelle particulière ?
* Pas de possibilité donc d'avoir un modèle unique pour tout encore, il faut des modèles spécialisés pour que certaines applications fonctionnent correctement, donc pas d'encodage universel. + coût de mise en place
* problème du décodage des vecteurs. Il faut penser à des solutions pour un décodage déterministe, non trivial à partir d'embeddings de phrase.

2) Voici le schéma d'OLAF, un framework permettant d'automatiser la création d'ontologies.

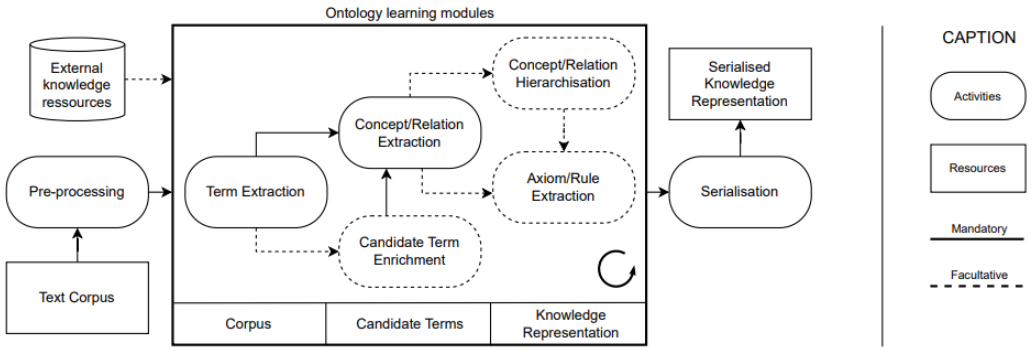


Fig. 2. Ontology Learning Process

Expliquez à quoi correspondent les différentes étapes de traitement, à quoi elles servent et comment on peut les réaliser techniquement.

Dans l'ordre d'exécution :
* Term extraction : extraire des textes des expressions/n grammes qui sont des candidats potentiels pour être des concepts de l'ontologie. Pour ce faire on utilise les techniques traditionnelles de keyword extraction (algorithmes statistiques, ou modèle spécialisé comme keybert).
* candidate term enrichment (facultatif) : renforcer le pool de termes candidats à partir de ressources linguistiques ou de bases de données externes, en cherchant par exemple des synonymes
* Concept / Relation extraction : sélection parmi les termes candidats des concepts de l'ontologie, et aussi des relations entre ces termes.
Pour trier les concepts on peut regarder leurs apparitions relatives, tenter des clustering… c'est une étape difficile.

Pour les relations on peut repérer des patrons syntaxiques qui relient les concepts retenus, ou utiliser des modèles.
* concept/ relation hierarchisation : cela consiste à structurer l'ontologie en lui donnant une structure. Cela peut se faire par des méthodes de clustering (hiérarchique), ou en appelant des LLMs
* extraction de règles / d'axiomes dans l'ontologie. Déterminé par des systèmes de règles, des LLMS… ces dernières étapes sont les plus complexes et dépendent de l'usage applicatif de l'ontologie à produire.

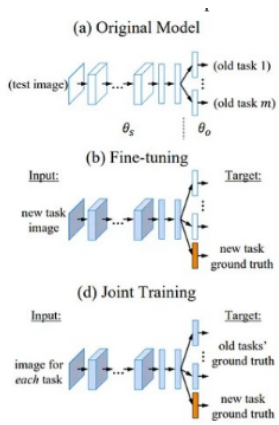
3) A votre avis, utiliser des LLMs pourrait-il être une bonne solution pour automatiser la création d'ontologies ? Argumentez votre réponse.

La faculté fine de compréhension du langage des LLM peut être un atout conséquent pour automatiser la création d'ontologies :
* permet d'automatiser à partir d'instructions spécifiques des tâches non triviales telles que la détermination de relations taxonomiques, la hiérarchisation de concepts, la détection de règles…
* Est bien plus rapide que de laisser ces tâches à des annotateurs.
* permet une certaine souplesse via le prompt engineering.
Mais des limitations se font vite sentir.
* nécessité de vérifier l'alignement des résultats produits avec les attentes des experts humains.
* très coûteux si on les applique pour les tâches comme l'extraction de termes.
* propagation de l'erreur si on les utilise à toutes les étapes du processus. Non déterministes par essence.
Leur utilité n'est visible que si bien encadrée. Il sera intéressant d'explorer des méthodes hybrides ne faisant appel au LLM que de façon limitée ou de « garder l'humain dans la boucle » (human in the loop) pour valider la qualité des résultats.

4) Qu'appelle-t-on Open IE en NLP ? Donnez une définition de cette tâche. Comment peut-on la réaliser techniquement ? Est-ce une tâche facile à évaluer ? Donnez des exemples et proposez un protocole.

Open IE : Open Information Extraction. Consiste à trouver des relations sous forme de triplet (El1, Relation, E2) de façon libre, sans qu'aucune restriction ne soit placée sur le type de relation (vs closed IE ou on trouve la relation parmi un ensemble de k relations possibles).
Cette tâche peut être réalisée au moyen de moteurs de règles, de modèles spécialisés ou en faisant appel à des LLMs. Elle est très dure à évaluer car la réussite de la tâche dépend de ce que l'on considère comme une relation valide. L'expression au centre doit-elle être un verbe ? (liens de possession), peut-elle comprendre plusieurs mots ? Si oui comment les découper ? Comment traiter les verbes intransitifs ?
Pour l'évaluer il faut donc disposer d'un guide clair définissant ce qu'on doit extraire comme relation dans un texte ainsi que d'exemples annotés. Ainsi on pourra comparer les résultats du système à évaluer avec ces références avec des métriques dérivées de la précision et du rappel (pour tenir en compte des différences possibles entre les termes relevés dans la relation).

5) En quoi consiste le joint training ? Quelle est son utilité ? Expliquez comment faire en pratique.



Le joint training consiste à entraîner ou fine-tuner le même modèle de fondation /pré-entraîné, le plus souvent un modèle de type BERT, simultanément sur deux tâches différentes. L'idée est que certaines des informations capturées dans l'entraînement pour une tâche sont directement utiles à l'effectuation de la deuxième tâche. Par exemple, si on veut créer un système de NER et un système de POS tagging, les informations sur les Part of Speech sont utiles. Les entités nommées sont en général des noms. On peut donc s'attendre à ce qu'un système entraîné sur les deux tâches soit meilleur.
Pour ce faire on crée dans l'architecture du réseau un embranchement avec la plupart des couches commune. La partie commune va stocker les paramètres utiles aux deux tâches et la partie spécialisée pour chaque tâche la partie spécifique. Pendant l'entraînement, on alterne des exemples sur chacune des tâches, et il y a backpropagation sur la partie commune, mais seulement pour la partie spécifique de chaque tâche. Cela permet de s'assurer que l'entraînement sur une tâche ne dégrade pas les performances sur l'autre.

6) Quelles sont les limitations des bases de données SQL ?
Qu'est ce qu'une base noSQL ? Citez en les types les plus connus et si possible des noms d'implémentations connues.

Les bases SQL ont des limitations liés à leur formalisme :
* Elles se présentent comme des tableaux, donc impossibilité de conserver les liens entre des éléments si ce n'est par le processus de jointure
* nécessite d'avoir une idée précise des données qu'on veut stocker, il faut « forcer la réalité dans la structure de la table »
* d'un point de vue pratique, impossibilité de laisser une case vide dans le tableau
* impossibilité d'ajouter une nouvelle colonne au tableau sans recréer une table. Le schéma d'une table ne peut pas être modifié après création.
* certaines opérations (jointures) sont très coûteuses.

Une base no SQL (not only SQL) est une base de données reposant sur des critères différents pour dépasser ces limitations.
On a :
* des bases clé valeur donc les vectorielles (Pinecone, chromaDB)
* les bases à base de documents (MongoDB)
* les bases à base de graphes (Neo4J)

7) Décrivez les étapes principales de la Data preparation et leur utilité.
Qu'est ce qu'un connecteur ? À quoi cela sert-il ?

La data préparation désigne l'ensemble des opération de nettoyage et de transformation des données avant qu'elles ne soient intégrées dans une base de données.
Les étapes principales sont :
* l'acquisition des données
* L'exploration (comprendre leur distribution)
* Leur nettoyage
* leur transformation / enrichissement
* Enfin l'ingestion dans la base

Un connecteur est un système qui permet de relier une source d'information à une base de données. Il permet l'ingestion facile de données en continu dans un système informatique. Le système (base de données, application) est informé dès que de nouvelles informations sont disponibles dans la source, et périodiquement ces nouvelles informations sont transmises, avec une fréquence dépendant du besoin de l'application.
Par exemple pour une application de newsletter, tous les jours les informations de différents sites de journaux sont transmises pour pouvoir être intégrées.

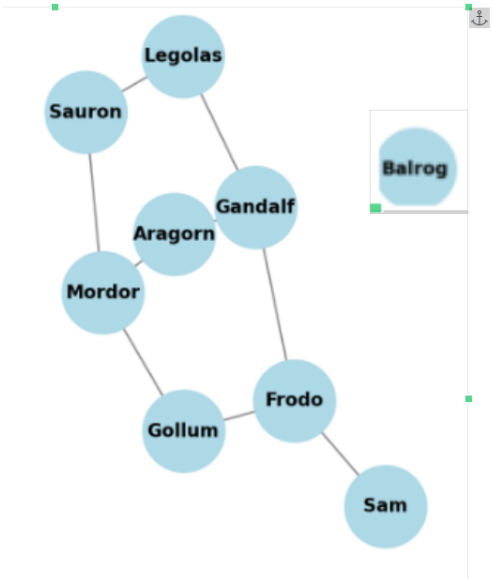
8) Citez quatre manières de décrire un graphe informatiquement. Donnez ces quatre représentations pour le graphe suivant.

Liste des relations :

```
edges3 = [
    ("Frodo", "Gandalf"), ("Frodo", "Gollum"), ("Gandalf", "Aragorn"),
    ("Gandalf", "Legolas"), ("Aragorn", "Mordor"), ("Legolas", "Sauron"),
    ("Sauron", "Mordor"), ("Gollum", "Mordor"), ("Sam", "Frodo")
]
```

Liste d'adjacence :

```
Frodo: Gandalf, Gollum, Sam
Gandalf: Frodo, Aragorn, Legolas
Aragorn: Gandalf, Mordor
Legolas: Gandalf, Sauron
Sauron: Legolas, Mordor
Mordor: Aragorn, Sauron, Gollum
Gollum: Frodo, Mordor
Sam: Frodo
Balrog:
```



matrice d’adjacence

	F	G	A	L	S	M	G	S	B
F	0	1	0	0	0	0	1	1	0
G	1	0	1	1	0	0	0	0	0
A	0	1	0	0	0	1	0	0	0
L	0	1	0	0	1	0	0	0	0
S	0	0	0	1	0	1	0	0	0
M	0	0	1	0	1	0	1	0	0
G	1	0	0	0	0	1	0	0	0
S	1	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0

matrice d’incidence

	e1	e2	e3	e4	e5	e6	e7	e8	e9
Frodo	1	1	0	0	0	0	0	0	1
Gandalf	1	0	1	1	0	0	0	0	0
Aragorn	0	0	1	0	1	0	0	0	0
Legolas	0	0	0	1	0	1	0	0	0
Sauron	0	0	0	0	0	1	1	0	0
Mordor	0	0	0	0	1	0	1	1	0
Gollum	0	1	0	0	0	0	0	1	0
Sam	0	0	0	0	0	0	0	0	1
Balrog	0	0	0	0	0	0	0	0	0

9) La recherche en profondeur (DFS)

La recherche en profondeur est un algorithme utilisé pour parcourir ou rechercher une structure de données, telle qu'un graphe ou un arbre. L'idée fondamentale de DFS est d'explorer une branche du graphe ou de l'arbre aussi loin que possible avant de revenir en arrière pour explorer d'autres branches.
Implémentez cet algorithme en python pour un graphe présenté comme ceci :
graph = { 'A': ['B', 'C'], 'B': ['A', 'D', 'E'], 'C': ['A', 'F'], 'D': ['B'], 'E': ['B', 'F'], 'F': ['C', 'E'] }
Fait en cours : BFS

```python
def bfs(graph, start):
    # Initialiser le dictionnaire des statuts
    status = {node: "unseen" for node in graph}
    queue = [start]  # Utiliser une liste au lieu d'une deque
    status[start] = "visited"
    paths = {start: [start]}  # Stocker les chemins vers chaque nœud

    while queue:
        current = queue.pop(0)  # Retirer le premier élément (comportement FIFO)
        print(f"Traitement de {current}")  # Traiter le nœud

        for neighbor in graph[current]:
            if status[neighbor] == "unseen":
                queue.append(neighbor)
                status[neighbor] = "visited"
                paths[neighbor] = paths[current] + [neighbor]  # Construire le chemin

        status[current] = "completed"  # Marquer le nœud comme terminé

    return status, paths
```

```python
def dfs(graph, start):
    # Initialiser le dictionnaire des statuts
    status = {node: "unseen" for node in graph}
    stack = [start]  # Utiliser une pile au lieu d'une file
    status[start] = "visited"
    paths = {start: [start]}  # Stocker les chemins vers chaque nœud

    while stack:
        current = stack.pop()  # Retirer le dernier élément (comportement LIFO)
        print(f"Traitement de {current}")  # Traiter le nœud

        for neighbor in graph[current]:
            if status[neighbor] == "unseen":
                stack.append(neighbor)
                status[neighbor] = "visited"
                paths[neighbor] = paths[current] + [neighbor]  # Construire le chemin

        status[current] = "completed"  # Marquer le nœud comme terminé

    return status, paths
```

La seule différence est la structure de donnée qui décide de quel nœud on explore ensuite.
Dans BFS : FIFO on explore tous les nœuds dans l'ordre où on les met dans la liste de visite
Dans DFS : LIFO on explore le dernier ajouté à la liste.

10) Les Elfes de la forêt de Lothlórien souhaitent établir un réseau de routes magiques reliant leurs cités cachées à travers la Terre du Milieu. Cependant, pour préserver l'harmonie avec la nature, ils veulent construire ces routes de manière optimale, minimisant la longueur totale des chemins tout en assurant que toutes les cités restent connectées.

a) En « langage mathématique », que doit on construire à partir du graphe pour répondre à ce problème ?
b) Pour résoudre ce problème, on pourrait utiliser l'algorithme de Prim.
L'algorithme de Prim fonctionne en partant d'un sommet et en ajoutant progressivement les arêtes les moins coûteuses connectant un sommet déjà dans l'arbre à un sommet non encore inclus.
Écrivez l'algorithme de Prim. Il doit prendre en entrée une liste d'arrêtes représentant la distance entre les villes (triplets from, to, longueur de la route) et le sommet choisit comme pivot, et renvoyer en sortie la liste des routes à construire de façon à connecter toutes les villes en minimisant la longueur des routes construites.

```python
routes = [
    ("Fondcombe", "Lórien", 7),
    ("Fondcombe", "Gondolin", 5),
    ("Lórien", "Valinor", 8),
    ("Lórien", "Doriath", 7),
    ("Gondolin", "Doriath", 9),
    ("Valinor", "Doriath", 5),
]
```

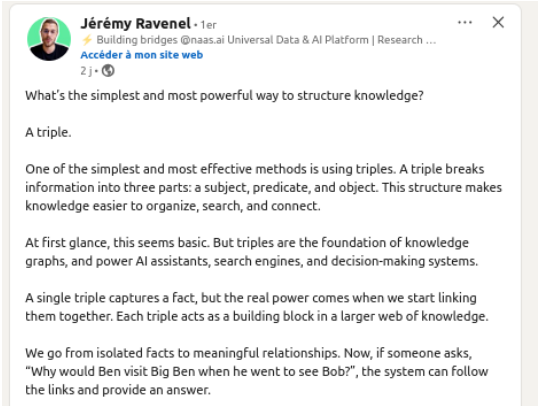
a) Résoudre le problème consiste à trouver un Arbre Couvrant Minimal.
En anglais … MST (pour Minimal Spanning Tree) c'est à dire un arbre qui recouvre tous les nœuds du graphe mais dont la somme totale des poids est minimale.

b)

```
def prim(edges, start):
        nodes.add(v)

    visited = set([start])  # Sommets déjà inclus dans l'arbre
    mst = []  # Arêtes de l'arbre couvrant minimal

    while len(visited) < len(nodes):
        # Trouver l'arête de poids minimal reliant un sommet visité à un non-visité
        min_edge = None
        min_weight = float('inf')

        for u, v, weight in edges:
            if (u in visited and v not in visited) or (v in visited and u not in visited):
                if weight < min_weight:
                    min_edge = (u, v, weight)
                    min_weight = weight

        if min_edge is None:
            break  # Le graphe n'est pas connexe

        mst.append(min_edge)
        visited.update([min_edge[0], min_edge[1]])

    return mst
```

L’idée est d’initialiser avec une liste des nœuds visités. Au départ seul un nœud est dedans.
Puis on déduit de cette liste des nœuds visités la liste des arrêtes reliant un de ces nœuds à un nœud qu’on a pas encore atteint. Puis on trie pour sélectionner celle qui a un poids minimal et on l’ajoute à l’arbre couvrant. Le nouveau nœud atteint est ajouté à la liste des nœuds visités et on continue jusqu’à la fin.

BONUS : Que pouvez-vous répondre à cet influenceur Linkedin à partir de vos connaissances ?



Comme le dit Jérémy, le triplet (EntitéA, Relation, EntitéB) est une structure de donnée très simples pour stocker l'information et elle est à la base des graphes de connaissances et des ontologies. Cela a permis le développement de nombreuses applications permettant de gérer des connaissances dans des domaines de spécialité.
Comme il le dit les triplets n'ont pas de valeur seuls, mais quand on les regroupe. On peut ainsi voir les connexions entre les éléments d'une base (recherche de plus courts chemins) ce qui débloque des cas d'usage dans la recherche d'information, la lutte contre la fraude…

Néanmoins sa vision est idéaliste et caricaturale sur plusieurs aspects ;
* il ne traite pas la question du dédoublonnage.
(Ben, visit, Big Ben) (Ben, see, Bob). Que faire quand on a plusieurs Ben et plusieurs Bob ? Ça va être le capharnaüm… traiter ce problème est nécessaire pour que ce soit utilisable.
* Le triplet est une information trop réduite, manque de granularité
* Le triplet est incapable de stocker certaines relations, notamment celles qui impliquent plus d'actants (verbes intransitifs).
* difficulté à créer des hiérarchies dans les relations. Certaines sont plus fondamentales que d'autres. Avec une base de triplets, tout est aplani.
* On perd l'information que plusieurs triplets cooccur dans le même contexte.
Il faut donc réfléchir aux moyens de dépasser ces limitations pour étendre leur potentiel applicatif.