Rapport TP4 – Langage Naturel & Bases de Données

El hana Amine - Benslimane Nassim

Avril 2025

Question 1 – Génération de requêtes SQL à partir d'une analyse ICV

L'objectif de cette partie était de transformer une phrase en langage naturel en une requête SQL valide, en s'appuyant sur le modèle **Intention - Concepts - Valeurs (ICV)**.

Détection des concepts

La fonction

```
def process_query(query)
```

extrait les éléments pertinents dans la phrase : noms d'acteurs, réalisateurs, années, genres, ou titres de films. Ces valeurs sont identifiées grâce à un lexique construit dynamiquement à partir de la base de données `base_films_500.csv`. Cela permet d'associer chaque valeur à un concept reconnu.

0.1 Évaluation des Classifieurs

Nous avons évalué nos classifieurs en utilisant différentes méthodes de pondération pour l'extraction des caractéristiques :

- **CountVectorizer Unigrammes** : 99,99 % et 99,94% de précision. (pour select et where resp)
- **CountVectorizer Bigrammes** : 99,99 % 99,94% de précision.
- **TF-IDF Unigrammes** : 100% et 100% de précision.
- **TF-IDF Unigrammes + Bigrammes** : 99,99 % 99,94% de précision.

Légère amélioration constatée avec TF-IDF sur les unigrammes indique que la pondération par TF-IDF renforce modestement la capacité discriminante du modèle. Cependant, les différences de performance restent minimes, négligeables, ce qui suggère que l'ensemble de caractéristiques utilisé est robuste et que le modèle fonctionne presque de manière parfaite quelle que soit la méthode de vectorisation.

Génération de la requête SQL

```
def predict_sql_query(query, select_pipeline, where_pipeline)
```

prend une requête en langage naturel, extrait les concepts/valuers, prédit les clauses `SELECT` et `WHERE` via deux modèles, puis génère une requête SQL complète en remplaçant les concepts abstraits par leurs valeurs réelles.

Exemple

`Requête :` *Donne les films by Steven Spielberg*
`Concepts détectés :` {réalisateur : Steven Speilberg}
`Requête SQL générée :`

```
SELECT titre FROM films WHERE realisateur = 'Steven Spielberg';
```

Question 2 – Interface interactive de consultation d'une base de données

L'objectif de cette section était de proposer un script interactif permettant à un utilisateur de poser une question en langage naturel, et d'obtenir automatiquement la réponse extraite depuis la base de données `base_films_500.csv`.

Structure du système

Le système repose sur les composants développés à la question 1 :
— Extraction des concepts via `process_query`.
— Prédiction des intentions `SELECT` et `WHERE` à l'aide des classifieurs supervisés.
— Génération automatique de la requête SQL avec `predict_sql_query`.
— Traduction et execution de la requête à l'aide de `pandas`

Interface interactive

```
def select_from_csv(csv_file, natural_query)
```

applique une requête en langage naturel à un fichier CSV en la traduisant automatiquement en SQL, puis en exécutant la sélection et les filtres correspondants à l'aide de `pandas`.

`Une cellule notebook est mise à disposition` pour tester la robustesse du système, permettant à l'utilisateur de formuler librement ses requêtes en langage naturel. Le système affiche les résultats directement sous forme de tableau, en s'appuyant sur le moteur `pandas` pour le traitement.

Exemple d'utilisation

`Utilisateur :` *Les titres des films de Leonardo DiCaprio et Brad Pitt*
`Résultat :`

titre
The Silent Dream Game
The Silent Echo Legacy
The Frozen Mission Saga

TABLE 1 – Résultat de la requête en langage naturel

Robustesse

La fonction `select_from_csv` gère également les erreurs d'exécution (requêtes incomplètes ou données manquantes) pour assurer une expérience fluide.

Question 3 – Évaluation quantitative

Nous avons réalisé trois types d'évaluation à partir du corpus d'évaluation fourni, sans adapter nos modèles sur ces données. Ces évaluations visent à mesurer la performance de notre système sur la détection des concepts/valeurs, la classification des intentions, et la production finale des résultats SQL.

Métriques d'évaluation et justification

Pour évaluer notre système de manière quantitative, nous avons utilisé les métriques suivantes :

- **Précision (Precision)** : proportion de concepts ou valeurs prédits qui sont réellement corrects. Cette mesure est importante pour évaluer la fiabilité des prédictions du système.
- **Rappel (Recall)** : proportion de concepts ou valeurs corrects qui ont été effectivement retrouvés par le système. Cela permet de mesurer la couverture de notre système.
- **F1-score** : moyenne harmonique entre la précision et le rappel. Cette métrique donne un indicateur global équilibré de la qualité du système.
- **Exact Match (EM)** : proportion de requêtes pour lesquelles tous les concepts/valeurs ont été correctement détectés. Cela permet d'évaluer la performance globale du système sur une requête entière.
- **Similarité de Jaccard (Jaccard Similarity)** : rapport entre la taille de l'intersection et celle de l'union des ensembles de concepts ou valeurs prédits et ceux attendus. Elle fournit une mesure continue (entre 0 et 1) du degré de recouvrement, où 1 indique une correspondance parfaite et 0 aucune correspondance.

— **Accuracy (Intention SELECT et WHERE)** : pour l'analyse des intentions, nous mesurons la proportion de phrases pour lesquelles le classifieur prédit la bonne catégorie.
— **Taux de réussite sur les résultats SQL** : pour l'évaluation finale, nous comparons la sortie du système avec la valeur attendue. Cela permet de juger la capacité du système à générer une requête SQL correcte produisant le bon résultat.

Ces métriques ont été choisies car elles sont standard dans les tâches de classification et d'extraction d'information, ce qui permet de comparer objectivement les performances du système, aussi bien sur des unités atomiques (concepts) que sur des requêtes complètes.

1. Détection des concepts et valeurs

Nous avons extrait les paires `(concept, valeur)` à partir des requêtes en langage naturel, puis comparé les concepts détectés aux concepts attendus extraits du SQL de référence. Il est à noter que nous avons utilisés les requetes naturelles labellisées "french_query" seulement (pas de paraphrases) pour cette étape.

— **Similarité de Jaccard** : 0.9831
— **Précision** : 0.9813
— **Rappel** : 0.9813
— **F1-score** : 0.9813
— **Exact Match** : 98,31 %
— **FP** : 0
— **FN** : 19

Ces résultats révèlent que le système est très fiable pour identifier les bonnes réponses, puisqu'il ne commet aucune erreur de prédiction injustifiée (0 faux positifs). Toutefois, la présence de 19 faux négatifs suggère qu'il manque encore de couverture sur certains cas, probablement dus à l'opérateur LIKE suivi d'une valeur introuvable dans notre modèle lexical. L'exact match élevé (98,31%) confirme que, lorsque le système prédit une réponse, elle est très souvent correcte. Il serait pertinent d'analyser les cas de faux négatifs pour affiner la compréhension sémantique ou enrichir les données d'entraînement.

2. Classification des intentions (SELECT / WHERE)

On appelant `produce_dataset` pour produire un ensemble de test à partir des données en langage naturel en français, paraphrasés et en anglais ainsi que les requetes sql :

Les intentions prédites par les modèles sont comparées aux intentions extraites du SQL cible, après normalisation.

— **Accuracy SELECT** : 99.66 (TF-IDF)
— **Accuracy WHERE** : 98.90 (TF-IDF)

```
Phrase : Please show me all the information of the films where the title
contains Last, sorted by year in descending order.
SQL : SELECT * FROM films WHERE titre LIKE '%Last' ORDER BY annee DESC;
Label réel : titre LIKE '%<titre>' ORDER BY annee DESC
Prediction : titre LIKE '<titre>%' ORDER BY annee DESC
```

`Phrase : Veuillez me montrer le directeur et le nombre total de films, groupés par réalisateur, triés par ordre décroissant.`
`SQL : SELECT realisateur, COUNT(*) AS total FROM films GROUP BY realisateur ORDER BY total DESC;`
`Label réel :`
`Prediction : titre LIKE '%<titre>' ORDER BY annee DESC`

3. Résultats SQL

Pour un sous-ensemble de requêtes produisant une seule valeur en sortie, nous avons comparé la valeur retournée avec celle attendue.

— **Taux de réussite** : 1692 / 1695 = **99.823%**

Analyse d'erreurs

Les erreurs les plus fréquentes observées dans les évaluations sont les suivantes :

— **Ajout de clauses non nécessaires**, comme `ORDER BY annee DESC`, même lorsque cela n'est pas demandé dans la requête initiale. Cela nuit à l'évaluation exacte de la clause `WHERE`.
— **Confusion dans les intentions SELECT** : dans certains cas, notre système renvoie `SELECT *` alors que seul le titre ou un champ spécifique est demandé (ou inversement).
— **LIKE '%<titre>' vs LIKE '<titre>%'** : L'erreur montre une confusion entre `LIKE '%...'` et `LIKE '...%'` , révélant un problème d'annotation dans le dataset où la requête "contient" est mal alignée avec un label qui signifie "commence par".
— **Absence de la clause WHERE** : le modèle étant entrainé sur un dataset contenant `WHERE` dans chaque requete, ce premier ne sait pas prédire une clause vide.
— **LIKE vs =** : Le modèle utilise `LIKE` même quand une égalité stricte `=` est attendue.
— **Extraction des valeurs partielles** : le système ne parvient pas à extraire les valeurs incomplètes après l'opérateur LIKE, car il ne les associe pas à une valeur du lexique.
— **Ambiguïtés linguistiques** : des formulations vagues comme "films de Natalie Portman" peuvent désigner l'actrice ou supposer qu'elle est réalisatrice, d'où des erreurs dans la génération de la clause WHERE.

Question 4 : Évaluation qualitative

Dans le cadre de cette évaluation, un utilisateur externe (non impliqué dans le développement du système) a été invité à tester notre interface de manière interactive. Il a formulé 20 requêtes en langage naturel, simulant un usage réaliste d'un moteur de recherche de films.

Pour chaque requête, il a attribué un score entre 0 et 3 :

— **0** : si la réponse est incorrecte ou absente,
— **1** : si la réponse est partiellement correcte,
— **3** : si la réponse est entièrement correcte.

Résultats obtenus

Le système a obtenu les scores suivants :

Score	Nombre de requêtes	Pourcentage
0	10	50%
1	3	15%
3	7	35%

Le score moyen est de **1,2 / 3**, ce qui reflète une performance insatisfaisante du système dans un cadre de requêtes totalement libres et complexes, ou encore lorsqu'il s'agit de valeurs non présentes dans la table, tant qu'un système NER n'a pas été implémenté.

Analyse d'erreurs

Les erreurs observées lors de l'évaluation qualitative sont globalement similaires à celles relevées dans l'évaluation quantitative. Cependant, elles mettent davantage en évidence un point spécifique : **l'ambiguïté du langage naturel**.

— Certaines requêtes incluent des formulations vagues comme *"films de Natalie Portman"*, qui peuvent être interprétées comme :
  — Natalie Portman en tant qu'actrice ;
  — ou bien réalisatrice.

On observe que les requêtes trop simples et trop complexes ne sont pas prédites correctement. Le modèle, étant entraîné sur des requêtes contenant des intentions et des concepts inséparables, commet des erreurs en ajoutant une clause `WHERE` additionnelle, puisqu'il est contraint de "choisir" une requête complète transformée présente dans le dataset. Par exemple, la requête "Liste les films où joue Leonardo DiCaprio" n'est pas prédite correctement, tandis que "Liste les films où joue Leonardo DiCaprio et dont le genre est 'action'" l'est.

Les requêtes plus complexes, quant à elles, ne peuvent pas être générées ou comprises en raison du modèle actuel et du dataset actuel.

Il est à noter aussi que l'opérateur `LIKE` ne fonctionnera jamais sans un système capable de reconnaître les valeurs non présentes dans le lexique, ou qui font partie (sous-chaîne) d'une valeur du lexique.

Comparaison avec l'évaluation quantitative

Les résultats quantitatifs et qualitatifs révèlent une divergence frappante : alors que les métriques techniques (Exact Match à 98,31%, F1-score à 0,98) suggèrent un système quasi-parfait, l'évaluation utilisateur montre que 50% des requêtes génèrent des réponses incorrectes. Cette contradiction s'explique par plusieurs facteurs clés :

1. Biais du jeu d'entraînement

Quantitatif : L'évaluation quantitative utilise un corpus prédéfini avec des requêtes similaires à celles du jeu d'entraînement, où les concepts (acteurs, réalisateurs, années) sont explicitement mentionnés et alignés sur le lexique de la base.

Qualitatif : Les requêtes libres introduisent des formulations imprévues ("films de Natalie Portman") ou des valeurs absentes du lexique ("titres contenant 'Shadow'"), exposant les limites de la généralisation du modèle.

2. Gestion de l'ambiguïté sémantique

Le système excelle sur les requêtes déterministes ("Qui a réalisé le film X ?"). En revanche, il échoue face aux ambiguïtés :

Exemple : Les requêtes avec opérateur LIKE ("titres contenant 'Last'") génèrent des erreurs car le modèle ne distingue pas entre '%Last' (fin de chaîne) et 'Last%' (début), une nuance non capturée dans l'entraînement.

3. Robustesse aux requêtes complexes

Requêtes multi-conceptuelles : Le système prédit correctement les requêtes combinant 2 filtres ("films d'action de 2010"), mais ajoute parfois des clauses parasites ((acteur1 = '<actor>' OR acteur2 = '<actor>' OR acteur3 = '<actor>')) issues de biais dans le jeu d'entraînement.

Absence de détection de négation : Des requêtes comme "films non réalisés par Christopher Nolan" ne sont pas gérées, car absentes du corpus d'entraînement.

4. Métriques vs expérience utilisateur

La métrique Exact Match pénalise uniquement les erreurs complètes, masquant des réponses partiellement correctes (ex : 2 concepts sur 3 retournés). À l'inverse, les utilisateurs attribuent un score de 1 même pour une réponse partielle, expliquant le taux élevé de "faux négatifs" perçus.

Améliorations Possibles

Trois axes majeurs d'amélioration pourraient renforcer la robustesse du système tout en restant dans le cadre d'un notebook Jupyter :

1. Gestion des mots-clés implicites

- **Lemmatisation contextuelle** :
  - Utiliser `spaCy` pour normaliser les termes (*"joués"* → *"jouer"*)
  - Gérer les flexions temporelles (*"sortira"* → *"sortir"*)
- **Embeddings sémantiques** :
  - Intégrer `FastText` pour capturer les relations de synonymie
    Exemple : `"comique"` (0.72), `"humoriste"` (0.68), `"comédie"` (0.65)
  - Clusterisation des genres proches (*"sf"/"science-fiction"/"anticipation"*)
- **Expansion lexicale dynamique** :
  - Mapper les variantes via un dictionnaire customisé
    Exemple : `"comique" → ["humour", "comédie", "drôle"]`

— Générer des synonymes avec `WordNet` pour les requêtes hors lexique

Cas d'usage :
Requête : *"Films comiques avec Ryan Reynolds"*
→ Expansion automatique : `WHERE genre IN ('comédie', 'humour')`
→ Détection de l'acteur malgré la variante orthographique (*"Reynolds" vs "Ryan"*)

2. Optimisation des Requêtes Complexes

— Support des intervalles temporels relatifs (*"années 2010", "dernières 5 années"*)
— Gestion des opérateurs booléens imbriqués (*ET négatif, OU exclusif*)

3. Validation Contextuelle

— Vérification de la cohérence acteur/réalisateur contre la base après enrichissement de la table
Exemple : Bloquer `WHERE realisateur = 'Natalie Portman'` si absente de ce rôle
— Alerte sur les combinaisons impossibles (*genre="documentaire" AND acteur="Batman"*)

Conclusion et Perspectives

Ce projet montre qu'une approche **hybride**, mêlant lexique explicite et classifieurs supervisés, permet de générer des requêtes SQL pertinentes dans 98 % des cas courants. Toutefois, des améliorations restent nécessaires avant d'envisager un déploiement en production.

Déploiement Immédiat

— Intégration de `spaCy` pour la lemmatisation (gain estimé : +15% de couverture)
— Mise en place d'un pipeline d'expansion lexicale via `WordNet`
— Ajout de templates de requêtes complexes au jeu d'entraînement

Évolutions à Moyen Terme

— **Modélisation des intentions composites** :
Détection conjointe des filtres (*genre + acteur + annee*) via réseaux de neurones
— **Interface de feedback** :
Permettre aux utilisateurs de corriger les requêtes mal générées directement dans le notebook

Exemple de feuille de route :
Semaine 1-2 : Intégration de la lemmatisation
Semaine 3-4 : Benchmark FastText vs Word2Vec
Semaine 5 : Test A/B sur 200 nouvelles requêtes