Une approche YeSQL pour la recherche d'information juridique*

Eric SanJuan[1,*,†], Adrian Chifu[2,†]

[1]Laboratoire Informatique d'Avignon
339 chemin des Meinajariés, BP 1228, 84 911 Avignon Cedex 9
[2]Laboratoire d'Informatique et des Systèmes(LIS) UMR 7020
Aix Marseille Université, 52 Av. Escadrille Normandie Niemen, 13397 Marseille Cedex 20

Abstract

On présente une application simple de recherche de passages dans les décisions de justice françaises en accès libre. Propulsée par PostGreSQL, les décisions sont segmentées en courts passages. On utilise ms-marco-minilm pour associer un plongement à chaque passage et pour convertir à la volée les requêtes utilisateurs en vecteurs. Plongements, contenu des décisions au format json sont insérées dans un schema relationnel normalisé. PostGreSQL permet d'étendre le SQL à la recherche textuelle avec les index généralisés (gin), l'extraction d'éléments json avec un type spécifique et le produit scalaire avec la bibliothèque pg_vector. Nous montrons l'efficacité de cette approche SQL pour la recherche de passages vis à vis du volume des textes et de leur longueur. Nous explorons ensuite les multiples stratégies d'agrégation de passages qui sont rendus possibles par cet environnement relationnel augmenté pour étendre la recherche de passages à celle de décisions entières.

Keywords

RI juridique, RI dense, Données ouvertes juridiques, Bases de données relationnelles étendues, Fonctions d'agrégation, Indexation textuelle, Indexation vectorielle

1. Introduction

Dans le domaine de la recherche d'information juridique [1], l'accès rapide et précis aux passages pertinents des décisions de justice représente un enjeu majeur [2]. Dans le prolongement des approches YeSQL[3], nous explorons l'intérêt et la faisabilité d'étendre l'approche relationnelle à la recherche de passages dans l'ensemble des décisions de justice françaises en accès libre. librement accessibles[1]. Notre expérimentation repose sur le système PostgreSQL avec son extension pgvector[2] et le modèle de langue ms-marco-MiniLM[3][4]. Pour cette première preuve de concept, nous utilisons le modèle de langue sans adaptation supplémentaire. Nous l'utilisons

CORIA 2024
*Position paper
*Corresponding author.
eric.sanjuan@univ-avignon.fr (E. SanJuan); adrian.chifu@lis-lab.fr (A. Chifu)
https://termwatch.es (E. SanJuan); https://adrianchifu.com/about-me/ (A. Chifu)
0000-0002-4057-6691 (E. SanJuan); 0000-0003-4680-5528 (A. Chifu)

[1]https://www.courdecassation.fr/acces-rapide-judilibre
[2]https://github.com/pgvector/pgvector
[3]https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-12-v2

tel quel pour transformer les requêtes des utilisateurs et les contenus des décisions en vecteurs sémantiquement riches [5].

Notre idée est que l'approche relationnelle offre une structure flexible pour représenter le contenu de longue décisions de justice par une large collection de vecteurs associés à de multiples extraits tel que considérés dans [6, 7]. Le sur-coût dû à la mise en oeuvre d'un système de gestion de bases de données relationnelles (SGBDR) complet comme base de données semble abordable vis à vis du volume de décisions de justice. La bibliothèque pgvector permet les opérations vectorielles, comme le produit scalaire, pour une recherche d'extraits[8] et est compatible avec les opérations d'ordonnancement et d'agrégation. Une expérimentation complémentaire sur un plus large corpus, la tâche 1 de CLEF Simple Text [9], a aussi été menée pour vérifier sa capacité à gérer des volumes de documents de l'ordre des 10 millions.

2. Architecture

Les décisions de justices sont trop longues et trop hétérogènes pour être représentées par des vecteurs sémantiques uniques Chalkidis et al. [10]. Plutôt qu'une représentation centrée sur les documents où la représentation vectorielle est jointe comme un attribut supplémentaire, nous proposons ainsi une approche relationnelle.

Nous considérons principalement trois relations:

DOCUMENTS avec 3 champs:

identifiant *text*

date *year-month-day*

juridiction *text*: identifiant court normalisée de la juridiction

contenu *json*: décision de justice comprenant un sous-champ *texte* constitué d'une liste de paragraphes. Ce contenu json peut être enrichi d'annotations sur les entités nommées et les personnes morales.

EXTRAITS avec 5 champs dont deux références externes à DOCUMENTS:

identifiant *sequence*

documents_fk *text* ⟶ DOCUMENTS(identifiant)

paragraphe *int* ⟶ DOCUMENTS(contenu->texte)

texte *text*: texte extrait de la décision de justice mis en forme pour application de modèles de plongements de phrases.

catégorie *enum*: type d'extrait parmi entête, faits, attendus, décision etc.

VECTEURS avec 3 champs dont une référence externes à EXTRAITS:

extrait_fk *int* ⟶ EXTRAITS(identifiant)

modèle *text*: identifiant du modèle utilisé pour générer le, plongement

vecteur vector[150]; plongement généré par l'extrait référencé

avec 3 champs dont une référence externes à EXTRAITS: extrait référence au modèle et contenu numérique.

Cette architecture implique les dépendances fonctionnelles suivantes:

DOCUMENTS ⟵ EXTRAITS ⟷ VECTEURS

Les relations EXTRAITS et VECTEURS pourraient donc être jointes, mais cette possibilité de gérer indépendamment contenu texte et représentation vectorielle a un double avantage:

- l'ajout d'une représentation dense des textes n'altère pas la gestion des documents ni l'indexation des extraits.
- de multiples dimensions de vecteurs peuvent être considérées, nous avons procédé ici avec des dimensions inférieures à 350, il est possible d'ajouter des représentations de cardinalité supérieure dans des relations disjointes.

Nous utilisons PostgreSQL sous licence BSD comme système de gestion de base de données relationnelle.

1. le type json permet de gérer le contenu de l'intégralité de la décisions comme des valeurs uniques sur lesquelles on peut appliquer des opérateurs de parcours d'arborescence pour extraire des sous éléments ;
2. les index généralisés (gin) permettent d'indexer le contenu textuel pour la recherche textuelle classique et peuvent être utilisés avec des dictionnaires spécialisés ;
3. nous utilisons un simple index ivvflat sur les vecteurs ce qui correspond à un k-means et à une réduction quadratique du temps de calcul des plus proches voisins.

Nous avons expérimenté le modèle ms-marco-minilm[4][11, 12] pour générer les plongements des extraits. Ce modèle est suffisamment frugal pour pouvoir être facilement affiné ou même re-appris sur des corpus spécialisés, ce que nous considérons de faire par la suite. Il offre aussi l'avantage de pouvoir être utilisé en ligne pour le plongement des requêtes utilisateurs rédigées en langage naturel, en temps réel sur CPU, donc facilement intégrable à tout système de recherche d'information. Dans le cas de PostGreSQL il peut être intégré sous forme de service externe avec pgcurl[5]. L'utilisation d'un service externe facilité aussi la sécurisation des données gérées par le SGBDR. Ainsi le traitement des exceptions lors de la génération des vecteurs est entièrement externalisé et n'affecte pas l'intégrité du SGBDR.

Cette architecture ne se limite pas à un unique modèle de plongement de phrases, les phrases pouvant être les passages des documents ou les requêtes utilisateur. Le relation VECTEURS peut intégrer non seulement de multiples vecteurs pour un même passage mais aussi des vecteurs issus de n différents modèles. Le service de plongement des requêtes des utilisateurs peut lui produire en parallèle sur n CPUs, l'ensemble des plongements correspondants. La recherche des extraits les plus proches se fait alors par agrégation sur la clef (extrait, modèle) de la relation VECTEURS. Toute fonction affine (combinaison linéaire des similarités par modèle) peut être considérée pour le calcul de l'ordonnancement final des extraits. Ces extraits peuvent à leur

[4]https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-12-v2
[5]https://github.com/pandrewhk/pgcurl

tour être agrégés pour ordonner les documents les plus pertinents. Ces multiples fonctions d'agrégation ne nécessitent pas d'être prédéfinies et peuvent être adaptées sans nécessiter un processus de re-indexation.

3. Complexité

s'avère

Les deux principales variables de notre architecture sont:

- $|D|$ nombre de documents au format XML/json,
- $|E|$ nombre d'extraits au format texte,
- l longueur maximale des passages pouvant être traités par le modèle de langue,
- m fréquence maximale en nombre de passages des tokens indexes,
- r nombre d'extraits recherchés.

La longueur des requêtes et des extrait est limité à la capacité du modèle de langue (110 tokens dans le cas des mini_lm) et la génération des représentations vectorielles se fait en temps constant. Avec les index utilisés la recherche dense des passages se fait en temps et espace $\sqrt{|E|}$ tandis que la recherche textuelle se fait en temps $\ln_2(|E|) \times l$ et espace $|E|$. La complexité de la recherche dense peut être réduite avec l'utilisation de classifications hiérarchiques plutôt que par centroïde, mais cela au prix d'une moindre souplesse[13].

Plutôt de complexifier la recherche dense, le cadre relationnel permet de combiner efficacement recherche textuelle booléenne et dense. Nous utilisons pour cela l'implantation dans PostGreSQL du calcul de la "couverture de densité" (cover density ranking) proposée dans [14]. Le calcul de cette couverture est similaire à une recherche booléenne modifiée pour tenir compte de la proximité des lemmes de la requête trouvés dans les documents. Cette fonction est particulièrement adaptée à la recherche d'information experte.

On obtient alors la complexité globale en temps:

$$O(\ln_2(|E|) \times \sqrt{m} \times r \times l)$$

la complexité en espace disque est elle de:

$$O(|D| \times m + |E|)$$

4. Agrégation

La combinaison d'index texte et denses sur les extraits de décision de justice permet une recherche granulaire avec un large rappel. La segmentation permet de traiter efficacement chaque élément du texte, optimisant la recherche par passage plutôt que par document entier. La jointure entre les relations DOCUMENTS et EXTRAITS permet de reconstruire efficacement le contexte des extraits trouvés. On peut étendre la recherche des extraits à celles des documents en définissant une fonction d'agrégation sur les vecteurs, l'indexation dense dans pgvector étant compatible avec les algorithmes d'ordonnancement et de regroupement définis dans la norme SQL3. La Figure 1 en procure un exemple.

Ceci introduit une souplesse dans le choix de l'agrégation et évite de devoir calculer ces agrégations par document au préalable. Ce modèle relationnel évite ainsi de limiter le nombre de vecteurs associés à un document sans explosion de l'espace disque nécessaire et permet d'adapter la méthode d'agrégation en fonction de la requête. On retrouve ici les mêmes problématiques que celles étudiées dans le contexte du la RI ciblée (Focus Information Retrieval) [15] et la RI de contextualisation [16] mais de manière inversée. Avant de disposer de la RI dense, les meilleurs systèmes procédaient en deux temps: recherche des meilleurs documents suivi d'une extraction des passages les plus pertinents.

5. Expérimentations

Nous avons procédé à une double expérimentation:

JudiLibre open data juridique de la cours de cassation[6]relatif à la mise à la disposition du public des décisions des juridictions judiciaires et administratives [7] pour vérifier sa faisabilité dans le domaine juridique. La mise en forme au format json de ces décisions nous a été procuré par la société Juri'Predis[8].

Aminer réseau de citations[9][17] dans sa version 12[18] qui combine enregistrements de la base bibliographique DBLP[10] avant 2019, Citations issue de l'ACM[11] et contenu des résumés. C'est ce même corpus qui est utilisé par la tâche 1 de l'initiative SimpleText du forum CLEF[9].

L'expérimentation sur JudiLibre est accessible sur le site de l'IUT d'Avignon[12] Le relation DOCUMENTS comprends 113.426 documents tandis que les relation EXTRAITS et VECTEURS en comprennent 19.179.774 enregistrements soit en moyenne plus de 160 vecteurs par documents. Ce nombre élevé de vecteurs par document est la conséquence de la longueur des décision de justice. On peut réduire le nombre de ces vecteurs en utilisant des représentations qui étendent l'amplitude de la fenêtre de texte Yang et al. [19] mais sa mise en oeuvre reste délicate. Le démonstrateur en ligne permet de tester cette approche pour la recherche de passages (système par défaut) et agrégée (système 10). Nous avons comparé les résultats obtenus avec un index standard ElasticSearch 8[13] et MeiliSearch[14]. Les trois dispositifs ont été installés simultanément sur le même serveur dédié (LINUX UBUNTU 20.04, disque SSD 4T, RAM 160 Go, 32 CPUs). MeiliSearch utilise un moteur de stockage LMDB[15] et s'avère être le plus rapide en temps de réponse sur les deux corpus testés. Programmé en RUST, sa rapidité permet d'obtenir les résultats instantanément au moment de l'écriture de la requête, ce qui permet une recherche très

[6]https://www.courdecassation.fr/en/acces-rapide-judilibre
[7]https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000043426865
[8]https://juripredis.com/
[9]https://www.aminer.cn/citation
[10]https://dblp.org/
[11]https://www.acm.org/
[12]Une démonstrateur pour cette approche est disponible ici: https://jpp.iut-avignon.fr/ avec la clef "etupretiut" et un accès à l'API peut être procuré sur demande.
[13]https://www.elastic.co/
[14]https://www.meilisearch.com/
[15]http://www.lmdb.tech/doc/

interactive. MeiliSearch permet aussi de gérer de multiples vecteurs pour un même document mais cela augmente considérablement l'espace disque nécessaire et ralentit considérablement le processus d'indexation. ElasticSearch est le plus efficace pour des requêtes sur la base de BM25 performantes pour la recherche de décisions jruidiques qui sont des documents longs. Les temps de recherches sur notre architecture PostGreSQL sont constants et similaires à l'utilisation de BM25 sur des requêtes courtes. Cette approche en deux temps qui alterne recherche vectorielle sur la relation VECTEURS puis agrégation de ces derniers par jointure avec la relation DOCUMENTS pour extraire des décisions augmente le rappel. A noter que la recherche par passage est suffisamment précise pour étendre celle par mots clefs. En termes d'écriture / ajout / modification de données et indexation, le système relationnel s'avère être le plus rapide pour ce volume de données. Moins d'une heure pour l'importation et indexation du corpus JudiLibre au format JSON après calcul des plongement des extraits. La modification d'un document y compris des extraits liés est elle instantanée.

Pour la deuxième expérimentation les trois relations contiennent le même nombre d'enregistrements, nous avons généré un vecteur unique par titre et résumé de référence bibliographique, soit 4.214.188. Deux APIs ont été mises à disposition des participants à l'atelier comme systèmes de base. Une version propulsée par ElastiSearch et BM25, une deuxième version selon la méthode neuronale proposée ici. Cette version neuronal améliore considérablement le rappel de la première sans perdre en précision sur les 100 premiers documents extraits.

6. Conclusion

Cette expérimentation montre l'intérêt des modèles relationnels pour combiner efficacement recherche textuelle et dense dans la recherche d'information experte par domaine, ici juridique. Il apparaît possible de facilement combiner technologies de base de données et de l'apprentissage automatique pour un accès aux documents précis assuré par la recherche textuelle, avec un rappel étendu en procédant pas similarité. Par la suite nous souhaitons étudier l'impact d'une adaptation (fine tuning) du modèle de langue utilisé sur les données juridiques, en plus d'un enrichissement des documents avec des annotations supplémentaires, de l'utilisation de lexiques métiers et de graphes de références aux articles de loi. Ceci en limitant autant que possible le recours à des processeurs graphiques.

Un autre intérêt de l'approche relationnelle est sa compatibilité par nature avec les contraintes du RGPD: minimisation et traçabilité des données personnelles. Les erreurs dans la pseudonymisation de documents publiés entraînent leur retrait immédiat de l'open data de la cours de cassation et leur remplacement par de nouvelles versions. Ce type d'incident se gère facilement dans le contexte relationnel. Le problème se pause aussi au niveau de la gestions des requêtes utilisateur. L'architecture proposée permet une génération des plongements côté client, ce qui peut être un moyen de protéger le contenu textuel exact de la requête initiale. Enfin, l'ensemble de cette architecture peut être facilement embarquée puisque l'ensemble de l'open data juridique ne nécessite d'infrastructure particulière pour masses de données et l'utilisation des modèles de plongement proposés ne nécessite pas non plus de processeur graphique.

```
CREATE OR REPLACE FUNCTION lib_knn(v_query vector(384), nb numeric)
   RETURNS TABLE( id_dec character varying(100),
                  chunk int,ip float, passage text,line int)
   LANGUAGE plpgsql
   AS $$
   BEGIN
    RETURN QUERY
    SELECT J.id_dec, J.chunk, (J.embedding <#> v_query) AS ip,
          J.passage, J.line
    FROM judilibre_v AS J ORDER BY ip LIMIT nb;
   END;
   $$;
```

Figure 1: Exemple de requête SQL3 vectorielle montrant la compatibilité entre pgvector et l'opérateur d'ordonnancement.

References

[1] T. M. v. Engers, S. J. K. JURIX (Eds.), Legal knowledge and information systems: JURIX 2006 ; the nineteenth annual conference ; [proceedings of the Nineteenth JURIX Conference on Legal Knowledge and Information Systems (JURIX 2006), december 7th-9th, Université Pantheon Assas - Paris II, France], number 152 in Frontiers in artificial intelligence and applications, IOS Press, Amsterdam, Berlin, 2006.

[2] Z. Wang, Legal Element-oriented Modeling with Multi-view Contrastive Learning for Legal Case Retrieval, in: 2022 International Joint Conference on Neural Networks (IJCNN), 2022, pp. 01–10. URL: https://ieeexplore.ieee.org/abstract/document/9892487. doi:10.1109/IJCNN55064.2022.9892487, iSSN: 2161-4407.

[3] Y. Foufoulas, A. Simitsis, Y. Ioannidis, YeSQL: rich user-defined functions without the overhead, Proceedings of the VLDB Endowment 15 (2022) 3730–3733. URL: https://dl.acm.org/doi/10.14778/3554821.3554886. doi:10.14778/3554821.3554886.

[4] N. Craswell, B. Mitra, E. Yilmaz, D. Campos, E. M. Voorhees, Overview of the TREC 2019 deep learning track, 2020. URL: http://arxiv.org/abs/2003.07820. doi:10.48550/arXiv.2003.07820, arXiv:2003.07820 [cs].

[5] N. Reimers, I. Gurevych, Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, 2019. URL: http://arxiv.org/abs/1908.10084. doi:10.48550/arXiv.1908.10084, arXiv:1908.10084 [cs].

[6] E. Schweighofer, Improving Legal Case Summarization Using Document-Specific Catchphrases, volume 346, IOS Press, 2022, p. 76.

[7] H. Xu, J. Savelka, K. D. Ashley, Toward summarizing case decisions via extracting argument issues, reasons, and conclusions, in: Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law, ACM, São Paulo Brazil, 2021, pp. 250–254. URL: https://dl.acm.org/doi/10.1145/3462757.3466098. doi:10.1145/3462757.3466098.

[8] S. Althammer, A. Askari, S. Verberne, A. Hanbury, DoSSIER@COLIEE 2021: Leveraging dense retrieval and summarization-based re-ranking for case law retrieval, 2021. URL: http://arxiv.org/abs/2108.03937. doi:10.48550/arXiv.2108.03937, arXiv:2108.03937 [cs].

[9] L. Ermakova, E. SanJuan, S. Huet, O. Augereau, H. Azarbonyad, J. Kamps, CLEF 2023 SimpleText Track: What Happens if General Users Search Scientific Texts?, in: J. Kamps, L. Goeuriot, F. Crestani, M. Maistro, H. Joho, B. Davis, C. Gurrin, U. Kruschwitz, A. Caputo (Eds.), Advances in Information Retrieval, volume 13982, Springer Nature Switzerland, Cham, 2023, pp. 536–545. URL: https://link.springer.com/10.1007/978-3-031-28241-6_62. doi:10.1007/978-3-031-28241-6_62, series Title: Lecture Notes in Computer Science.

[10] I. Chalkidis, M. Fergadiotis, P. Malakasiotis, N. Aletras, I. Androutsopoulos, LEGAL-BERT: The Muppets straight out of Law School, 2020. URL: http://arxiv.org/abs/2010.02559, arXiv:2010.02559 [cs].

[11] R. Nogueira, K. Cho, Passage Re-ranking with BERT, 2020. URL: http://arxiv.org/abs/1901.04085. doi:10.48550/arXiv.1901.04085, arXiv:1901.04085 [cs].

[12] X. Ma, R. Pradeep, R. Nogueira, J. Lin, Document Expansion Baselines and Learned Sparse Lexical Representations for MS MARCO V1 and V2, in: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, Madrid Spain, 2022, pp. 3187–3197. URL: https://dl.acm.org/doi/10.1145/3477495.3531749. doi:10.1145/3477495.3531749.

[13] H. Li, Q. Ai, J. Zhan, J. Mao, Y. Liu, Z. Liu, Z. Cao, Constructing Tree-based Index for Efficient and Effective Dense Retrieval, in: Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, Taipei Taiwan, 2023, pp. 131–140. URL: https://dl.acm.org/doi/10.1145/3539618.3591651. doi:10.1145/3539618.3591651.

[14] C. L. A. Clarke, G. V. Cormack, E. A. Tudhope, Relevance ranking for one to three term queries, Information Processing & Management 36 (2000) 291–311. URL: https://www.sciencedirect.com/science/article/pii/S0306457399000175. doi:10.1016/S0306-4573(99)00017-5.

[15] D. Alexander, P. Arvola, T. Beckers, P. Bellot, T. Chappell, C. M. DeVries, A. Doucet, N. Fuhr, S. Geva, J. Kamps, others, Report on INEX 2010, in: ACM SIGIR Forum, volume 45, ACM, 2011, pp. 2–17.

[16] P. Bellot, T. Bogers, S. Geva, M. Hall, H. Huurdeman, J. Kamps, G. Kazai, M. Koolen, V. Moriceau, J. Mothe, others, Overview of INEX 2014, in: Information Access Evaluation. Multilinguality, Multimodality, and Interaction, Springer, 2014, pp. 212–228. 00000.

[17] J. Tang, A. C. Fong, B. Wang, J. Zhang, A Unified Probabilistic Framework for Name Disambiguation in Digital Library, IEEE Transactions on Knowledge and Data Engineering 24 (2012) 975–987. URL: http://ieeexplore.ieee.org/document/5680902/. doi:10.1109/TKDE.2011.13.

[18] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. P. Hsu, K. Wang, An Overview of Microsoft Academic Service (MAS) and Applications, in: Proceedings of the 24th International Conference on World Wide Web, ACM, Florence Italy, 2015, pp. 243–246. URL: https://dl.acm.org/doi/10.1145/2740908.2742839. doi:10.1145/2740908.2742839.

[19] L. Yang, M. Zhang, C. Li, M. Bendersky, M. Najork, Beyond 512 Tokens: Siamese Multi-depth Transformer-based Hierarchical Encoder for Long-Form Document Matching, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, ACM, Virtual Event Ireland, 2020, pp. 1725–1734. URL: https://dl.acm.org/doi/10.1145/3340531.3411908. doi:10.1145/3340531.3411908.