CentraleSupélec

université PARIS-SACLAY

Recherche d'information sur le Web

ST4-EI
Partenaire : HeadMind Partners

Réalisé par le groupe No : 5
Théo Schneider
Jules Berard
Zenta Utagawa
Salma Maazoum
Joshua Noullier-Jacques

2 juin 2023

Table des matières

Chapitre 1

Introduction et exploration des données

1.1 Introduction

Dans le cadre de l'enseignement d'intégration, un moteur de recherche a été développé à partir des données extraites du formulaire de questions et réponses de "Stack Exchange" [1]. Dans ce rapport, nous détaillerons le processus de conception et de construction du moteur de recherche. Nous commencerons par l'étape de visualisation des données et la familiarisation avec leur contenu, puis nous présenterons les différentes méthodes de recherche mises en œuvre, testées pour évaluer leur performance, et enfin nous fournirons le moteur de recherche le plus performant.

1.2 Requête SQL sur la base de données :

Dans le but de prendre en main la base de données qui servira de fondement au moteur de recherche, plusieurs requêtes SQL ont été exécutées. Nous présentons ici les questions en langage naturel, les réquêtes SQL associées ainsi que les résultats obtenus.

1 - Combien de lignes contiennent les tables Posts, Tags et Comments ?
Requête : SELECT Count (*) FROM TableName, TableName prend le nom de la table
Résultats : 76613 lignes pour la table Posts, 693 lignes pour la table Tags et 79140 lignes pour la table Comments.

2 - Combien de commentaires ont été posté depuis le début de l'année ?
Requête : SELECT Count(*) FROM Comments WHERE CreationDate > '2023-01-01 00 :00 :00'
Résultats : 2165 commentaires.

3 - Combien y a-t-il d'utilisateurs ?
Requête : SELECT COUNT(*) FROM Users
Résultats : 127 712 utilisateurs

4 - Combien y a-t-il de nouveaux utilisateurs chaque année depuis 2020 ?
Requête : SELECT YEAR(CreationDate), COUNT(*) from Users Group BY YEAR(CreationDate) HAVING YEAR(CreationDate) >= 2020
Résultats : 2020 - 19856, 2021 - 16623, 2022 - 13823, 2023 - 4656.

5 - Quel est le contenu du plus petit poste ?
Requête : SELECT Body,LEN(Body) FROM Posts WHERE LEN(Body) = (SELECT MIN(LEN(Body)) FROM Posts)
Résultats : Le plus petit poste est vide.

6 - Quel est le poste le plus voté ?
Requête : SELECT TOP 1 Posts.body, Posts.id, COUNT(*) FROM Posts JOIN Votes on Posts.id = Votes.PostId Group by Posts.id, Posts.body ORDER BY Count(*) DESC
Résultats : Le poste ayant l'identifiant 24051 a le plus grand nombre de votes (462 votes). Voici un extrait du poste : Micro- and macro-averages (for whatever metric) will compute slightly different things, and thus their interpretation differs. A macro-average will compute the metric independently for each class and then take the...

7 - Quels sont les 10 tags les plus fréquents en 2022 ? (du plus fréquent au moins fréquent)

1. https ://archive.org/details/stackexchange

Requête : SELECT TOP 10 Tags.TagName, Tags.count FROM Tags ORDER BY Tags Count Desc
Résultats : machine-learning - 11146, python - 6601, deep-learning - 4791 ...

1.3 Exploration de la base de données :

Après l'importation de la base de données, les différentes tables ont été extraites du fichier Zip. Quant au choix du type de fichier où les données sont stockées, le format XML a été choisis. En effet, XML est lisible par l'utilisateur, ce qui permet de percourir et explorer les données. Il est auto-descriptive avec une structure hiérarchique qui permet aux développeurs de représenter des relations complexes entre les éléments de données, ceci est le cas de notre base de données, puisque chaque table a une clé étrangère qui fait référence à une case d'une autre table. D'autres formats auraient pu être envisagés, notamment CSV pour ces données tabulaires, Protocol Buffers pour un stockage et une transmission efficaces. Voici un exemple après l'extraction XML pour le fichier Tags, ce dernier est désormais représenté par un dataframe et prêt pour être utilisé une fois que les colonnes inutiles sont éliminées.

	Id	TagName	Count	ExcerptPostId	WikiPostId
0	1	definitions	36	105.0	104.0
1	2	machine-learning	11059	4909.0	4908.0
2	3	bigdata	462	66.0	65.0
3	5	data-mining	1172	80.0	79.0
4	6	databases	95	8960.0	8959.0
...	...	...	...	...	...
674	1215	train	1	NaN	NaN
675	1216	analysis	1	NaN	NaN
676	1217	lemmatization	1	NaN	NaN
677	1218	text-to-columns	1	NaN	NaN
678	1219	t	1	NaN	NaN

FIGURE 1.1 – Une vue d'ensemble de la table Tags

1.4 Visualisation des données :

La recherche par requêtes SQL étant limitée, la visualisation des données sous la forme de graphe devient essentielle. Elle permet une meilleure compréhension des distributions et des schémas cachés dans les données.

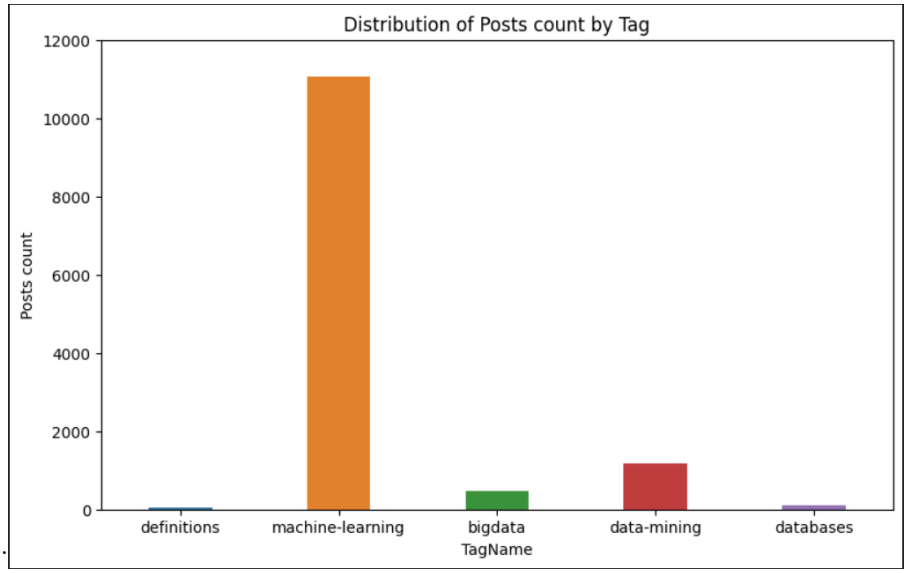

FIGURE 1.2 – Une vue d'ensemble du nombre de postes ayant les étiquettes en abscisse

Les étiquettes peuvent jouer un rôle essentiel dans la conception d'un moteur de recherche. Leur utilisation envisageable consiste à catégoriser les articles et à restreindre la recherche à la catégorie ou aux catégories qui décrivent le mieux la requête.

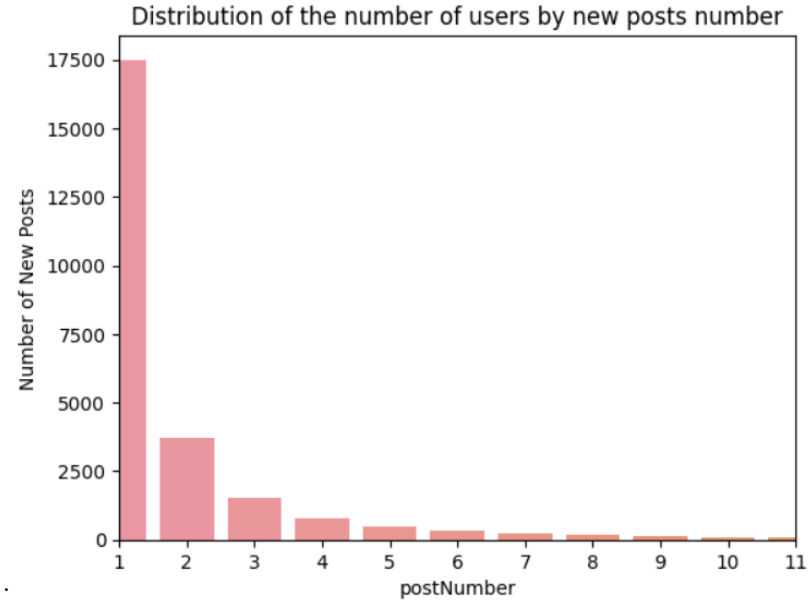

FIGURE 1.3 – Répartition du nombre d'utilisateurs par nombre de nouveaux postes

La tendance est de publier un seul article. Un grand nombre d'utilisateurs ne publient qu'une fois. Cette donnée peut être exploitée pour favoriser les articles publiés par des utilisateurs actifs.

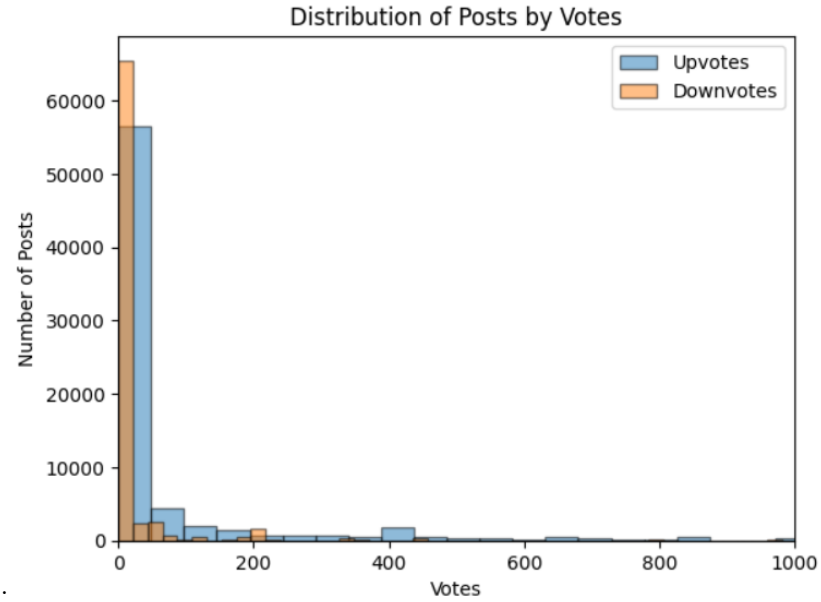

FIGURE 1.4 – Distrubition des nombres de posts par "Upvotes" et "Downvotes" du profil du propriétaire

La plupart des propriétaires des postes n'ont pas de votes sur leur profil ce qui peut biaiser toute pondération basée sur ce critère. Néanmoins une favorisation pour les postes ayant des votes positifs et une pénalité pour ceux ayant des votes négatifs demeurent envisageable.
L'idée à ce stade est de baser le moteur de recherche sur le contenu du poste et de juger sa pertinence par rapport à la requête en utilisant différents types d'algorithmes, notamment une approche naïve que nous tenterons d'améliorer, une recherche par Tf-Idf, une recherche probabiliste, ainsi qu'une fusion de ces méthodes et une amélioration par l'utilisation de métadonnées, de la recherche sémantique et du clustering. Nous détaillerons les résultats de chaque approche dans les chapitres 3 et 4.

Chapitre 2

Traitement des données

2.1 Création de l'index inversé

Afin de créer l'index inversé des postes, un filtrage des données utiles est nécessaire. Ainsi, nous traitons le contenu de chaque poste pour extraire les mots utilisés, supprimer les balises HTML, éliminer les mots vides, enlever la ponctuation et lemmatiser chaque mot. Dans ce processus, nous utilisons la fonction "nltk.tokenize.word_tokenize" de la bibliothèque nltk. De plus, pour construire l'index inversé, nous avons choisi de le stocker dans un dictionnaire, ce qui facilite l'accès aux données utiles lors de la recherche. Les statistiques tf (fréquence des termes) et df (fréquence des documents) sont également conservées dans ce dictionnaire pour être utilisées lors de la pondération dans les méthodes de recherche.

Dans le but d'optimiser le temps d'exécution du programme lors de la recherche des documents les plus pertinents pour une requête donnée, nous avons stocké ce dictionnaire dans un fichier pickle. Ainsi, notre traitement se concentrera sur cette donnée.

Voici un aperçu de la structure du dictionnaire :

Les clés sont les mots lemmatisés. La valeur correspondante à chaque clé est un dictionnaire contenant son df (document frequency) et la liste des documents contenant le terme. De plus, l'idf (inverse document frequency) est normalisé par rapport à la taille du poste, étant donné une diversité de la longueur des postes constatée lors de la familiarisation avec la base de donnée.

```
{'I': {'df': 49841,
  'inv_ind': [(5, 0.06481481481481481),
   (7, 0.034482758620689655),
   (14, 0.03896103896103896),
   (16, 0.0125),
   (21, 0.0019342359767891683),
   (22, 0.011627906976744186),
   (24, 0.013245033112582781),
   (29, 0.014492753623188406),
```

La visualisation de la loi de Zipf a permis de comprendre la distribution des termes rares dans ce corpus. la fréquence d'un mot est inversement proportionnelle à son rang dans le classement des termes par fréquence ce qui est mis en évidende dans la figure ci-dessous. En d'autres termes, les mots les plus fréquents dans un corpus de texte apparaissent beaucoup plus souvent que les mots moins fréquents.

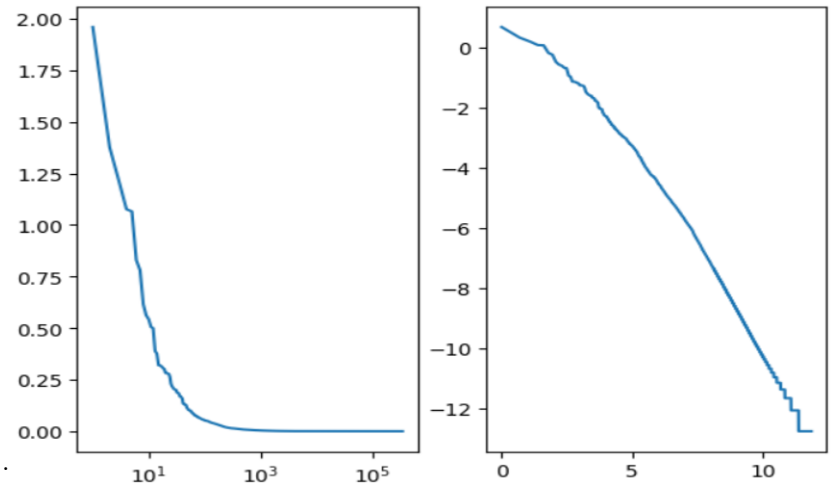

FIGURE 2.1 – La loi de Zipf, en abscisse le rang du terme et en ordonné la fréquence d'apparution

Chapitre 3

Moteurs de recherche et comparaison

3.1 Approche naïve

L'approche naïve se base sur l'approche *sac de mots*. Dans cette approche, un document est représenté par l'ensemble des mots qui le compose (éventuellement pré-traité). Pour une requête donnée, on parcourt ensuite l'ensemble des documents à la recherche de mots en commun.

Le principal inconvénient de cette méthode est le besoin de parcourir l'ensemble des mots de chaque document. C'est pourquoi on peut grandement en améliorer l'efficacité en utilisant l'index inversé pour ne répertorier que les documents qui contiennent un des mots de la requête.

3.2 Recherche booléenne

Le modèle booléen permet de traiter des requêtes utilisant des opérateurs booléens tels que "AND", "OR" et "NOT". Pour l'implémenter, la bibliothèque ttable a été utilisée pour trouver la forme normale conjonctive de la requête, puis effectuer la recherche sur le dictionnaire contenant l'index inversé. Cette méthode a un rappel élevé mais une précision faible. Une amélioration possible consisterait à coupler cette méthode avec la recherche sémantique, par exemple, ou à ajouter une pondération permettant de classer les résultats en fonction d'un score.

3.3 Modèle probabiliste MIB

Le modèle de recherche probabiliste repose sur l'idée que la pertinence d'un document par rapport à une requête peut être quantifiée en utilisant des mesures de probabilité. Dans notre cas, la notion de pertinence d'un document vis-à-vis d'une requête n'est pas accessible, de même que l'apprentissage permettant d'estimer ce paramètre ne peut être effectué. Ainsi, le calcul du score de pertinence ne prend en compte que la fréquence documentaire (df) de chaque terme présent dans la requête et dans le document en même temps. Cependant, le modèle MIB présente des limitations, car il ne tient pas compte de la fréquence des termes dans les documents et les requêtes, ce qui peut réduire la précision dans notre cas. C'est pourquoi l'idée d'ajouter l'idf (inverse document frequency) dans la pondération a été introduite. Cela a permis une meilleure différenciation des documents, car le cardinal de l'ensemble des scores possibles a considérablement augmenté.

Requête : "Machine Learning".

Premier poste renvoyé : "What is regularization in machine learning ?","
What is regularization in machine learning ?
Why do we need this in machine learning ?
"...

Deuxième poset renvoyé : <p>You can use Graph based machine learning : <a href="https ://github.com/stellargraph/stellargra...

3.4 Modèle probabiliste Okapi BM25

Le modèle OBM25 se base également sur un modèle probabiliste, et vient combler les lacunes du modèle MIB. En effet, OBM25 va attribuer un score à chaque document en fonction d'une requête à l'aide de la fréquence de chacun des termes dans le document mais également dans le corpus. Les informations portant sur les lois de probabilités sont modélisées par 3 paramètres qui doivent être estimés par rapport au jeu de données sur lequelnous travaillons, afin de garantir un résultat optimal. Cependant, nous avons pris les estimateurs optimaux pour le cas moyen, qui sont un bon compromis entre la pertinence du résultat et le temps d'implémentation. Ce modèle assure également une normalisation du score d'un document par sa longueur, ce qui lui permet de

mieux gérer un corpus inhomogène. Le modèle fonctionnant bien dès son implémentation, nous n'avons pas jugé utile de lui implémenter des raffinements.

Requête : "training cnn"

Premier poste renvoyé : "How is the Fully Connected Layer of a cnn network trained ?", " When training a cnn network, only the filters are updated, but how are the Fully Connected Layer weights updated ? "

3.5 Evaluation des moteurs de Recherche

Afin d'évaluer et de comparer les différents modèles que nous avons développés, nous avons choisi de nous baser sur le fichier excel fourni qui contient des jugements de pertinence sur un sous-ensemble de posts pour plusieurs requêtes. Nous avons reconstruit tous les index nécessaires sur ce sous-ensemble de posts, puis à partir de ces données, nous avons pu obtenir le NDCG pour chaque méthode.

Bien entendu, cette méthode d'évaluation a des limites. Par exemple, le modèle sémantique et l'approche de clustering sont peu efficaces sur un corpus réduit car il n'y a pas assez de documents pour exploiter la proximité sémantique entre les mots. De plus, Le suos-ensemble de documents étudié pour l'évaluation n'est pas a priori représentatif de l'ensemble de la collection de posts. Cependant, il s'agit d'une première approche pour distinguer les différents modèles développés.

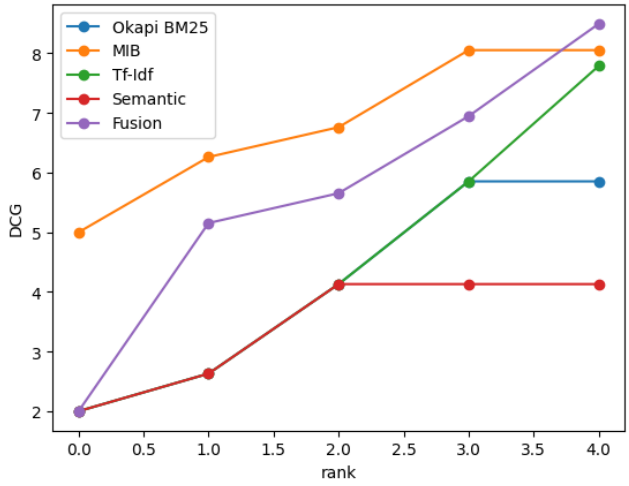

FIGURE 3.1 – Comparaison des différents modèles de recherche grâce au DCG

3.6 Tf-Idf avec scikit-learn

Le modèle Tf-Idf est un modèle de recherche vectoriel. Une première étape de *fitting* construit un espace vectoriel engendré par le vocabulaire du corpus. Ensuite une étape de transformation permet de construire la représentation vectorielle de chaque document dans cet espace, en attribuant une pondération *Tf-Idf* pour chaque terme dans chaque document. On réunit ensuite tous ces vecteurs lignes de documents dans une matrice, qui est la représentation du corpus dans l'espace engendré par son vocabulaire. On applique la même transformation à la requête, et on réalise le produit matriciel entre la matrice du corpus et le vecteur de la requête. On obtient un vecteur unicolonne, qui à chaque document attribue un score lié à la requête. Pour implémenter ce modèle, nous avons utilisé la librairie scikit-learn, qui propose les fonctions de *fitting* et de transformation.

Requête : "introduction of data science".

Premier poste renvoyé : "*no title*", " If you're looking for an introduction to mathematics for data science, ..."

Deuxième poset renvoyé : "*no title*", " There is free ebook "Introduction to Data Science" based on ..."

3.7 Similarité Sémantique

L'analyse sémantique des documents s'éloigne de lapproche sac de mots utilisée jusque là en explorant le contexte dans lequel les mots apparaissent. Pour cela, on utilise le module python *sentence_transformers* qui contient des algorithmes pré-entraînés pour de l'analyse sémantique.

Parmi les différents modèles disponibles, nous avons choisi d'utiliser le modèle *multi-qa-mpnet-base-dot-v1*, car il s'agit d'un modèle entraîné à la reconnaissance de questions et de requêtes entraîné sur une grande base de données. Dans le cas où les recherches avec ce modèle durent trop longtemps, nous avons aussi envisagé de le remplacer par le modèle *multi-qa-MiniLM-L6-cos-v1*, qui est plus rapide, mais légèrement moins performant.

Pour définir la mesure de distance à utiliser entre une requête et un post, nous avons choisi la mesure cosinus, c'est-à-dire le produit scalaire que la fonction *dot_ scores* de la librairie utilisées permet de calculer facilement. Il suffit ensuite d'ordonner les posts en fontion de ce score pour obtenir les posts les plus pertinents.

Requête : "how to track data".

Premier poste renvoyé : "*no title*", " It sounds like you need a second data structure like an index. An index tracks important metadata information about the data. ..."

Chapitre 4

Améliorations

4.1 Text Clustering

Dans le but d'améliorer les performances du moteur de recherche final, nous avons tenté de tester son fonctionnement en utilisant la méthode de clustering. À l'aide de la LDA (Latent Dirichlet Allocation), il est possible d'identifier un thème pour chaque article, ainsi que pour la requête. De cette manière, nous pouvons restreindre notre recherche aux articles directement liés à la requête. En ce qui concerne le choix de l'hyperparamètre du nombre de thèmes, nous avons utilisé la méthode des "elbows". Cependant, même avec 200 thèmes, le score continue à diminuer et nous n'atteignons pas le surapprentissage, d'où notre choix pour cet hyperparamètre.

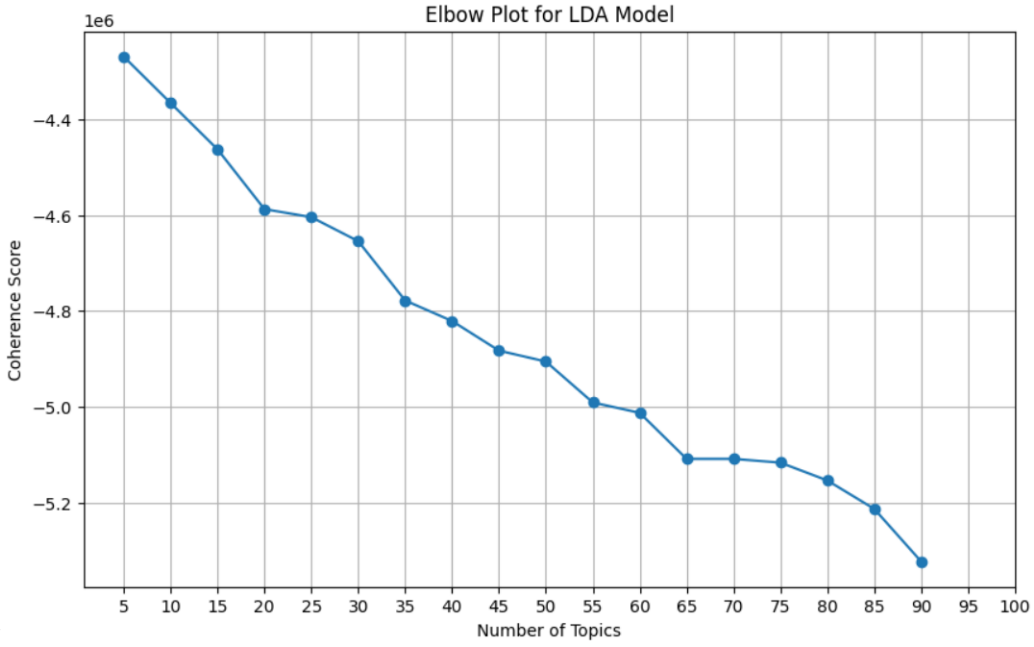

FIGURE 4.1 – Variation du score en fonction du nombre de sujets dans la modélisation thématique ("elblows" méthode)

Voici le résultat d'une requête, le moteur de recherche utilise le clustering couplé avec le BIM : **Requête :** "mesure performance for multiclassification model".

Premier poste renvoyé : "How to choose a machine learning sampling method ?"," I have a multi-classification problem with a training dataset of 3 groups (50 samples :150 samples : 100 samples). ..."

Deuxième poset renvoyé : "Calculating confidence interval for model accuracy in a multi-class classification problem"," In the book Applied Predictive Modeling by Max Kuhn and Kjell Johnson, there is an exercise concerning the calculation of a confidence interval for model accuracy. It reads as follows. ..."

4.2 Fusion des méthodes

Une fois toutes les méthodes susnommées implémentées, nous avons entrepris de les réunir en une unique fonction de recherche, pour augmenter le rappel et la précision moyenne. Seulement, toutes les méthodes n'ont pas le même degré de fiabilité et il convient de pondérer leurs résultats avant d'en faire la synthèse. La plus grande partie du travail était l'optimisation des poids attribués aux différentes méthodes. Après réflexion et analyse, nous avons choisi d'utiliser des poids équirépartis dans un premier temps, qui offraient de bons résultats. Le nouveau score de chaque document est la somme des scores pondérés. Il convient de normaliser les scores de chaques méthodes afin d'avoir un résultat entre 0 et 1. Le modèle booléen ne pouvant pas béneficier de cette pondération est utilisé pour pénaliser les documents ne comprenant pas les termes de la recherche.

4.3 Prise en compte du score (Des métadonnées)

Le corpus utilisé comprend un nombre important de métadonnées pouvant être utilisées dans la recherche de documents. Le score d'un document par exemple, est la synthèse des votes qu'il a pû recevoir des utilisateurs. On peut alors supposer que plus ce score est haut, plus la qualité de l'information contenue est importante. C'est cette métadonnée que nous avons choisi d'exploiter en priorité. Après le traitement effectué par la fonction de recherche, on obtient une liste de document classés par ordre de pertinence, avec leur score attribué par la fonction de recherche. On classe ensuite ces documents par score attribué par les utilisateurs et on leur attribue un bonus en fonction de leur rang et du poids attribué à la méthode. On renvoit alors la liste des documents avec le nouveau classement.

D'autres métadonnées nous ont semblées pertinentes, comme *AnswerId* qui indique pour un post l'existence d'un commentaire accepté comme une réponse satisfaisante à la question posée, ou encore les *flags* qui indiquent si un post est fermé, a trouvé une réponse,... Nous pourrions également utiliser les *tags* ou les titres qui permettent de repérer facilement le thème dont traite un post. Enfin, il serait possible de relier les posts et commentaires à leurs auteurs/contributeurs, puis entre eux avec les liens hyper-texte, afin de réaliser un parcours du jeu de données et d'attribuer des scores d'autorité et de *hub* pour déterminer plus précisément la pertinence des pages.

4.4 Recherche de mots par contexte

Utilisation de World2Vec : Ici on a choisi Skip gram plutôt que CBOW car Skip gram est plus performant pour prédire des mots rares (ce qu'il peut y avoir ici avec du vocabulaire spécialisé).

Utilisation d'un réseau de neurones qui, pour un terme donné, détermine un terme lié à ce dernier. Ce lien se base sur le contexte, qui s'établit à partir d'une base de données textuelles : ici c'est le texte d'un post sur dix de stack exchange. Le but ici est de trouver des mots liés au mot demandé, car celui qui recherche des informations l'a peut être en tête. L'intérêt est alors de choisir les mots avec une probabilité élevé d'être lié au terme recherché.

Ensuite, on récupère les documents pertinents avec le terme donné à l'aide de SearchEngine. On fait de même avec le terme lié au contexte. On peut alors faire une interpolation linéaire des rangs de chaques documents issus des deux listes, et on obtient alors une liste finale de documents pertinents.

Chapitre 5

Conclusion

Finalement, notre proposition de moteur de recherche se base sur l'incorporation de différents modèles existants dans le but de compenser les faiblesses de chaque modèle. Ainsi, le modèle booléen qui ne renvoie pas de classement est utilisé après pour valoriser les documents comportant les termes de la requête par rapport aux autres. La recherche sémantique permet aussi d'élargir les résultats obtenus en explorant les documents contenant des mots sémantiquement similaires à ceux de la requête. Le résultat ainsi obtenu est ensuite à nouveau modifié pour tenir compte des métadonnées disponibles pour chaque post, en particulier le score d'un post, c'est-à-dire la valeur des votes qu'il a reçu.

Il reste encore de nombreuses améliorations que nous n'avons pas eu le temps d'explorer ou d'implémenter. Par manque de temps, la pondération dans la fonction de recherche finale est équirépartie entre les différents modèles. Le développement d'un algorithme pour trouver une meilleure pondération a été initié, mais n'a pas abouti à temps. Afin d'améliorer la précision de notre moteur de recherche, il conviendrait de traiter un sous-ensemble de test pllus large, en ajoutant des posts et des requêtes de test. Enfin, l'ajout d'un modèle de recherche de mots grâce au contexte, comme celui que nous avons développé avec un réseau de neurones, permettrait de se rapprocher d'une recherche intelligente.