Ingénierie des technologies industrielles

2020-2021

Projet de Fin d'Études

"OPTIMISATION DU MOTEUR DE RECHERCHE TEEXMA POUR L'ENTREPRISE BASSETTI"

Maria MORENO PEREZ

Tuteur ICAM : Catherine MAURY

Tuteur UPV : Jose Antonio LOZANO QUILIS

Toulouse, 18 juin 2021

INDEX

1. OBJECTIFS

Le but de ce projet est d'améliorer la pertinence du moteur de recherche du logiciel TEEXMA fourni par la société BASSETTI à ses clients.

Mon objectif est donc de développer une solution permettant au programme d'offrir des résultats plus pertinents et filtrés qui facilitent la recherche de l'utilisateur, en tenant compte non seulement des mots correspondants mais aussi de leur contexte et de leur importance dans les documents traités. Pour ce faire, je vais étudier les différentes solutions possibles qui me permettront d'améliorer et de simplifier le processus de sélection des résultats des moteurs de recherche.

En outre, fournir une solution de mise en œuvre simple et simplifiée, tant pour le programmeur que pour l'utilisateur. En outre, le moteur de recherche doit fonctionner avec n'importe quel format de document, puisque la base de données de l'entreprise en question a numérisé des fichiers de tous types.

2. INTRODUCTION AU PROBLÈME

2.1 Le problème de l'entreprise

BASSETTI possède et collecte une grande quantité d'informations, de documents et de dossiers scientifiques parmi lesquels il est difficile d'obtenir une navigation et une recherche optimales en raison du calibre de la base de données.

Bien que le navigateur TEEXMA dispose de différents critères de recherche, tels que les multicritères, ces filtres ne sont pas suffisants pour trouver le document le plus pertinent car il n'effectue pas de recherche sur cette base. La sélection de données est vaste et non filtrée, et donc longue et fastidieuse pour le client.

En outre, de nombreuses entreprises utilisant ce logiciel contiennent de vieux documents manuscrits et scannés qui ne peuvent être utilisés par l'interface et sont donc gaspillés.

2.2 Motivation

La recherche pertinente dans n'importe quelle base de données est un problème qui, depuis le début de l'ère informatique, progresse et évolue. D'une certaine manière, c'est un domaine qui sera toujours en développement constant. C'est pourquoi j'ai trouvé très intéressant de faire mon mémoire sur ce sujet, car c'est quelque chose qui est à l'ordre du jour et qui ne sera jamais oublié, mais qui va muter au fur et à mesure de nos avancées technologiques. En outre, il est intéressant de se plonger dans la connaissance des différents algorithmes de recherche utilisés par les plus grandes entreprises du monde, telles que Google.

Il convient de noter que les êtres humains sont constamment en contact avec les moteurs de recherche, que ce soit pour rechercher une requête sur Internet, un produit sur le site d'un magasin, une application sur le téléphone mobile, un contact dans l'agenda pour envoyer un message, etc.

En conclusion, c'est un projet très intéressant à aborder d'un point de vue académique et personnel.

3. PRESENTATION DE L'ENTREPRISE BASSETTI ET DU SOFTWARE TEEXMA

3.1 Qu'est-ce que BASSETTI ?

BASSETTI propose des solutions logicielles pour permettre aux entreprises industrielles de structurer, améliorer et valoriser leurs données et leur savoir-faire.
Fondateur de la solution logicielle Technical Expertise Management® (TEEXMA), qui vise à mettre en œuvre des méthodes, outils et processus informatiques. Celles-ci visent à organiser, archiver et diffuser des connaissances à haute valeur ajoutée.

3.2 Les domaines d'application de TEEXMA®.

Cette entreprise dispose de différentes solutions en fonction du domaine d'application de l'industrie cliente, parmi lesquelles TEEXMA pour : LIMS, matériaux, GMAO, conception, qualité, environnement, fabrication et PLM.

Les principales industries avec lesquelles l'entreprise travaille sont l'automobile, la chimie et la pharmacie, l'aéronautique et l'espace, l'énergie et l'ingénierie industrielle en tout genre.

3.3 Fonctionnalités du logiciel

Pour mieux comprendre l'utilité de ce logiciel, nous allons prendre comme exemple le TEEXMA utilisé pour le cas des matériaux. Le client a accès à toutes les données de ses matériels à travers une application web complète, intuitive et sécurisée, accessible dans toute l'entreprise, à son tour il est possible de visualiser ses données à travers des formules dynamiques qui peuvent être modifiées et configurées selon ses besoins. Vous pouvez bénéficier de l'intégration de données matérielles selon vos besoins directement dans l'application (SENVOL, MMPDS, JAHM et de nombreuses autres bases de données contenant des titres commerciaux ou des matériaux normalisés), ainsi que de fonctionnalités éprouvées pour assurer la validation des données disponibles, avec une sécurisation des données grâce au module de gestion des droits et aux modules de workflow, assurant une traçabilité complète de votre activité. Il dispose également de fonctions avancées de calcul et d'analyse graphique pour la définition de vos données de dimensionnement.

Les connecteurs privilégiés peuvent être utilisés pour partager les données sur les matériaux avec les membres du bureau d'études et les utiliser dans les outils de conception et de simulation. Les leçons apprises par les experts sur l'utilisation et le choix des matériaux au cours des différents projets peuvent être utilisées dans les outils de conception et de simulation.

Les enseignements tirés par les experts sur l'utilisation et le choix des matériaux au cours des différents projets peuvent être utilisés pour les futures règles de conception. Il est intéressant de noter que les listes de substances et de réglementations bénéficient d'un soutien et d'un suivi de leur conformité environnementale. Enfin, il contient des fonctions avancées pour rechercher, sélectionner et comparer les données sur les matériaux.

4. MOTEUR DE RECHERCHE

4.1 Qu'est-ce que c'est ?

Les moteurs de recherche sont nés de la nécessité d'organiser, de classer et de gérer les informations sur l'internet en raison du grand nombre de pages web disponibles.

Un moteur de recherche ou moteur de recherche est un système informatique qui recherche des fichiers stockés sur des serveurs web par l'identification d'un mot-clé utilisé par la personne effectuant la recherche et, en conséquence, l'utilisateur obtient une liste de liens qui dirigent vers des sites web où les sujets liés au mot-clé sont mentionnés, bien qu'ils ne soient pas toujours utilisés pour obtenir des liens de sites web, mais ils utilisent les données numériques stockées, qu'elles soient structurées ou non, par exemple : Bases de données, fichiers dans des répertoires, fichiers dans le nuage, entre autres, pour récupérer et former des index à partir de leurs propres critères de recherche.

Ils constituent un type de technologie NoSQL (Not only SQL), ces systèmes sortent du modèle de stockage relationnel et s'affranchissent de la nécessité d'établir toutes les informations dans un modèle relationnel unique pour finir par ajuster les informations à ce que chacune d'entre elles est : un type de données. En d'autres termes, les technologies NoSQL sont optimisées pour résoudre un problème spécifique pour des types de données spécifiques.

4.2 Types de moteurs de recherche

- **Moteurs de recherche hiérarchiques** : Ce type de moteurs de recherche dispose d'interfaces d'interrogation textuelle. Ils vérifient les bases de données des pages web par le biais de leurs araignées et celles-ci compilent les informations sur les contenus compatibles avec la recherche de l'utilisateur. Une fois la requête effectuée, ils classent les résultats par pertinence par rapport à la recherche spécifique et en fonction de l'historique de navigation de l'utilisateur.

- **Répertoires** : Les moteurs de recherche de type annuaire sont des liens vers des pages qui sont regroupées par catégorie. Ils sont très simples, mais nécessitent un soutien humain et une maintenance permanente pour fonctionner. Ces moteurs de recherche ne parcourent pas les sites et ne stockent pas les contenus, ils regroupent uniquement les liens par catégories et sont organisés par date de publication et non par pertinence ou concordance avec une recherche effectuée par l'utilisateur.

- **Méta-moteurs de recherche** : ces interfaces fonctionnent en transmettant les recherches à plusieurs moteurs en même temps. C'est-à-dire qu'ils transmettent la requête à d'autres sites pour analyser les résultats qu'ils présentent, afin d'élargir la marge des mêmes résultats, de présenter leurs propres conclusions et d'ordonner les liens selon l'ordre défini par le système structurel du méta moteur de recherche.

Lorsque l'on parle de moteurs de recherche, la première chose qui vient à l'esprit est le puissant Google, bien qu'il en existe beaucoup d'autres que parfois la majorité des internautes ne connaissent pas, comme Yahoo!, Bing, Ask, MSN Search, GO, entre autres. Mais il est important de souligner qu'il s'agit d'un type de moteurs de recherche appelés Webcrawlers, il y a aussi ceux conçus pour offrir un service comme Solr et Elasticsearch, qui est ce sur quoi nous allons nous concentrer dans ce projet.

4.3 Elasticsearch vs Solr

Lucene (1999) est devenu la base de deux des meilleurs moteurs de recherche open source de nos jours : Solr (2004) et Elasticsearch (2010). Lucene est une bibliothèque de recherche d'informations et est responsable de la construction et de la gestion de ce que l'on appelle un index inversé, une structure de données spécialisée dans la mise en correspondance des documents texte avec les termes de la requête que nous aborderons plus tard.

Les principales caractéristiques de ces moteurs sont :

- **Évolutif** : capable de distribuer le travail (indexation et traitement des requêtes) à plusieurs serveurs dans un cluster.
- **Prêt à être déployé** : les deux solutions sont accompagnées d'exemples pratiques permettant de mettre en place un service avec un minimum d'efforts.
- **Optimisés pour la recherche** : ils sont très rapides, capables d'exécuter des requêtes complexes en quelques dizaines de millisecondes.
- **Grands volumes de données** : ils sont conçus pour traiter des index de milliards de documents.
- **Centré sur le texte** : bien qu'ils supportent les recherches sur les dates et les nombres, leur base et leur principale force est de traiter les textes naturels (emails, pages web, articles, documents PDF et messages de réseaux sociaux tels que les blogs et les tweets) en extrayant la structure implicite de ceux-ci vers l'index des motos pour améliorer la recherche.
- **Résultats triés par pertinence** : en fonction de la requête de l'utilisateur, les documents classés selon la requête sont retournés.

Pour évaluer les solutions possibles et choisir le moteur de recherche le plus adapté à notre projet, nous allons procéder à une comparaison entre les deux. Ils sont les plus utilisés et préférés par les grandes entreprises et suivent la politique open source d'Apache Lucene, de sorte que plusieurs de ses caractéristiques sont très similaires. Toutefois, il existe également de grandes différences en termes de facilité de déploiement, d'évolutivité et d'autres fonctionnalités.

4.3.1 Apache Solr

Il s'agit d'une plateforme de recherche open source construite sur une bibliothèque Java appelée Lucene. Il offre les capacités de recherche d'Apache Lucene d'une manière facile à utiliser. Il offre une indexation distribuée, une réplication, une interrogation équilibrée en fonction de la charge, ainsi qu'un basculement et une reprise automatisés.

S'il est mis en œuvre et géré correctement, il est capable de devenir un moteur de recherche hautement fiable, évolutif et tolérant aux pannes.

De nombreux géants de l'internet, tels que Netflixx, eBay, Instagram et Amazon, utilisent Solr pour sa capacité à indexer et à rechercher sur plusieurs sites.

Parmi ses principales caractéristiques, nous trouvons :

- Recherche en texte intégral.
- Mettez en évidence les termes de recherche.
- Des facettes de recherche qui peuvent inclure l'analytique, le filtrage, la tokenisation et d'autres options.
- Indexation en temps réel.
- Groupement et agrégation.
- Fonctions NoSQL (bases de données non relationnelles) et traitement riche des documents tels que les fichiers Word, PDF, et autres.

4.3.2 Elasticsearch

D'autre part, Elasticsearch est une plateforme de recherche distribuée open source (licence Apache 2), un moteur de recherche RESTful construit au-dessus de la bibliothèque Apache Lucene. Il a été introduit quelques années après Solr.

Il s'agit d'un moteur de recherche plein texte distribué, multi-nœuds, doté d'une interface web HTTP(REST) et de documents JSON sans schéma. Les bibliothèques officielles d'Elasticsearch sont disponibles en Java, Groovy, Ruby, Perl, Python, .NET et JavaScript.

Le moteur de recherche distribué comprend des index qui peuvent être divisés en fragments, et chaque fragment peut avoir plusieurs répliques. Chaque nœud peut avoir un ou plusieurs fragments, et son moteur agit également comme un coordinateur pour déléguer correctement les opérations aux fragments.

Elasticsearch est évolutif et permet une recherche en temps quasi réel. L'une de ses principales caractéristiques est la multi-tenue. Cela consiste à faire fonctionner une seule base de code pour tous les clients, avec la même structure de données, mais des zones de données distinctes pour chacun.

On peut l'expliquer en suivant l'analogie des sols :

Le même bâtiment dispose d'infrastructures communes et d'un plan identique pour chacun des appartements, qui disposent logiquement de leur propre espace

Caractéristiques principales :

- Recherche distribuée.
- Multi-indexation.
- Une chaîne d'analyseurs syntaxiques.
- Recherche analytique.
- Regroupement et agrégation.
- Mettez en évidence les termes de recherche.

4.3.3 Principales différences

Elastiscsearch a été lancé en 2010 sous le nom de Compass et, contrairement à Lucene et Solr, il n'est pas un projet de la fondation Apache et repose sur Github. Cependant, les deux sont utilisés sous la même licence Apache 2.0.

C'est en 2014 qu'Elasticsearch a dépassé Solr en termes de tendance et de popularité.
Il est important de noter que Solr est véritablement open source, et pourtant tout le monde peut aider et contribuer. En revanche, dans Elasticsearch, seuls les employés d'Elastic Stack peuvent accepter telles contributions. C’est pour ça que Solr aura beaucoup plus de fonctionnalités de qualité acceptable, tandis qu'Elasticsearch s'en tiendra à moins de fonctionnalités communautaires, mais à un niveau de qualité bien supérieur.

D'après différentes données d'expérience avec les deux, il n'est pas possible de voir une différence de performance entre l'un et l'autre pour les applications de recherche interne et externe si elles sont conçues, mises en œuvre et utilisées correctement.

En conclusion, Solr est plus orienté vers la recherche de texte en tant que moteur de recherche.

Dans l'entreprise en question, le logiciel qu'ils utilisent fonctionne avec Solr et dans ce travail j'ai également décidé de travailler avec ce moteur de recherche car il supporte plus de formats de données et dans notre cas nous avons une base de données de tous types.
Certains des types de formats avec lesquels Solr travaille sont JSON XML CSV, tandis qu'Elasticsearch ne prend en charge que le format de fichier JSON. Solr a également plus de fonctionnalités et comme nous voulons mettre en œuvre différents filtres pour affiner la recherche, c'est la meilleure solution.

5. FONCTIONNEMENT DES MOTEURS DE RECHERCHE

5.1 Résumé

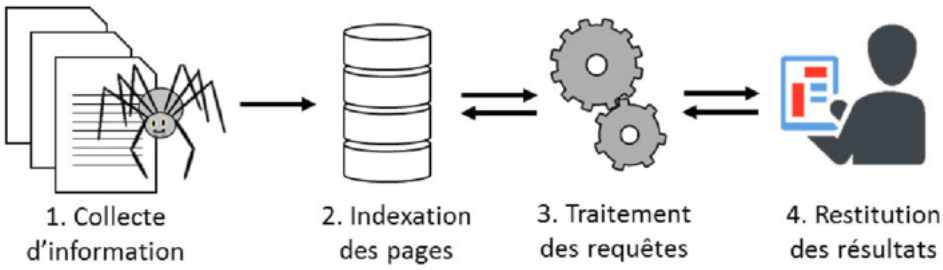
On peut dire de manière très simple qu'un moteur de recherche se compose de quatre parties :

- Une interface permettant à l'utilisateur d'effectuer des recherches.
- Un robot ou une araignée qui recherche des informations sur Internet.
- Un algorithme qui relie les demandes des utilisateurs à la base de données.
- Une base de données dont le contenu a été indexé.

Le cœur de chaque moteur de recherche est l'algorithme (assez complexe dans la plupart des cas) qui dirige le robot ou l'araignée et classe ensuite les informations à afficher après les demandes des utilisateurs.
Il est important qu'en plus de ces fonctions, le moteur de recherche remplisse deux autres fonctions pour que l'algorithme soit valide :

- Collecter des informations en utilisant des techniques de crawling.
- Stocker et indexer l'information

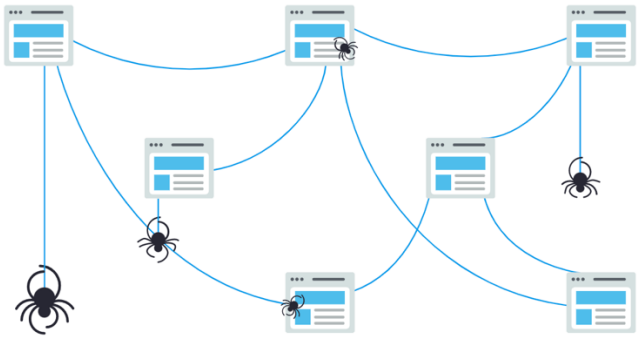
Les moteurs de recherche créent des listes de pages web à l'aide de leurs araignées ou robots grâce à des techniques d'exploration et organisent ensuite les informations trouvées, en créant des index du contenu.



5.2 Processus

5.2.1 Crawling

En général, les moteurs de recherche trouvent les informations à l'aide d'un agent ou d'un robot (web crawler) qui entre dans une page web pour recueillir des données, identifie les liens sur cette page web et enfin suit ces liens comme le ferait un utilisateur pour répéter le processus encore et encore.

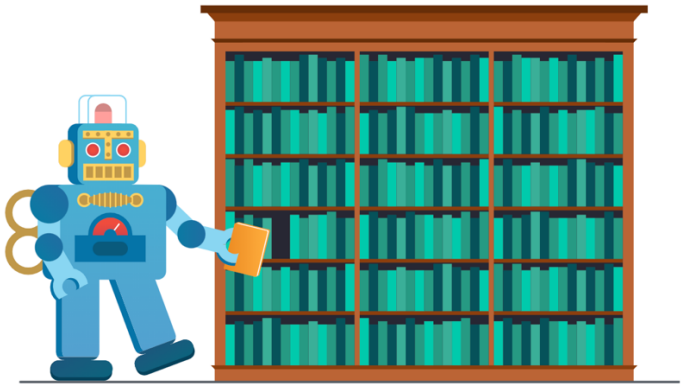


De cette façon, le robot saute d'un lien à l'autre et navigue sur différentes pages à partir desquelles il collecte des données pour alimenter sa base de données.

C'est ainsi qu'un moteur de recherche trouve des informations. C'est la moitié du processus. Ensuite, il est temps d'indexer l'information.

5.2.2 Indexation

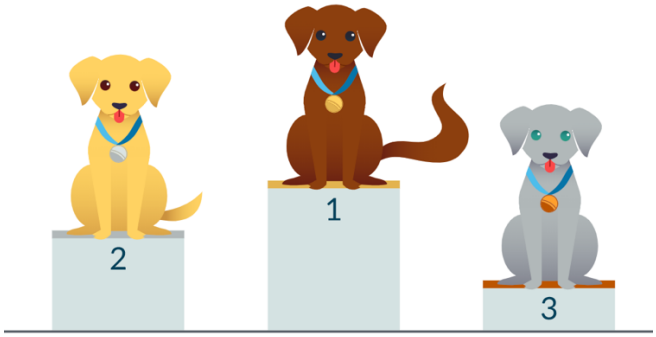
Une fois que le robot ou araignée, également connu sous le nom de crawler web (d'où le nom de link crawling), trouve l'information, il crée un index avec les informations essentielles de la page web qui est stocké dans la base de données et ne sera affiché aux utilisateurs que lorsqu'ils feront leur demande dans l'interface de recherche du moteur de recherche.



L'affichage ou non des informations indexées à la demande d'un utilisateur dépend finalement de l'algorithme du moteur de recherche.

5.2.3 Ranking

Après le crawling et l'indexation c'est nécessaire de profiler les éléments de contenu qui répondront le mieux à la requête d'un utilisateur, ce qui signifie que les résultats sont classés du plus pertinent au moins pertinent.



Pour déterminer la pertinence, les moteurs de recherche utilisent des algorithmes, c'est-à-dire un processus ou une formule permettant de récupérer des informations stockées et de les ordonner de manière pertinente. Ces algorithmes ont subi de nombreuses modifications au fil des ans afin d'améliorer la qualité des résultats de recherche. Google, par exemple, procède chaque jour à des ajustements d'algorithmes. Certaines de ces mises à jour sont des ajustements mineurs de la qualité, tandis que d'autres sont des mises à jour d'algorithmes fondamentaux/étendus déployés pour s'attaquer à un problème spécifique, comme Penguin pour lutter contre le spam de liens.

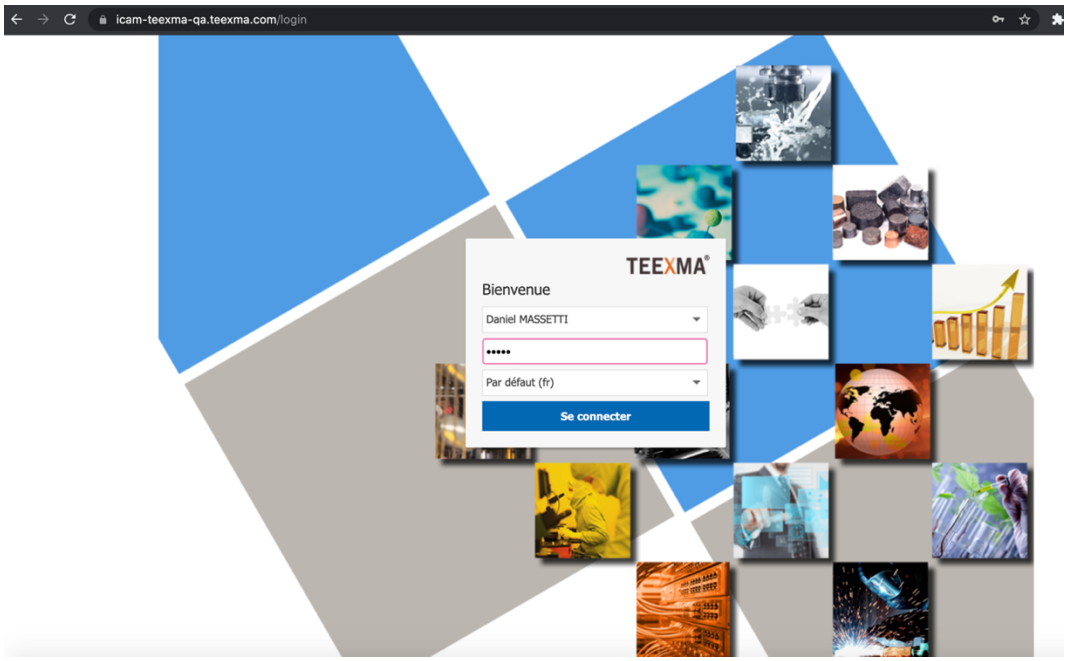
6. FAMILIARISATION AVEC LE LOGICIEL TEEXMA

6.1 Écran de bienvenue et accueil

Afin de nous familiariser avec le moteur de recherche utilisé par BASSETTI, la société m'a fourni un nom d'utilisateur et un mot de passe pour une version de démonstration. Ce portail est personnalisé pour chaque client et le secteur dans lequel ils travaillent.

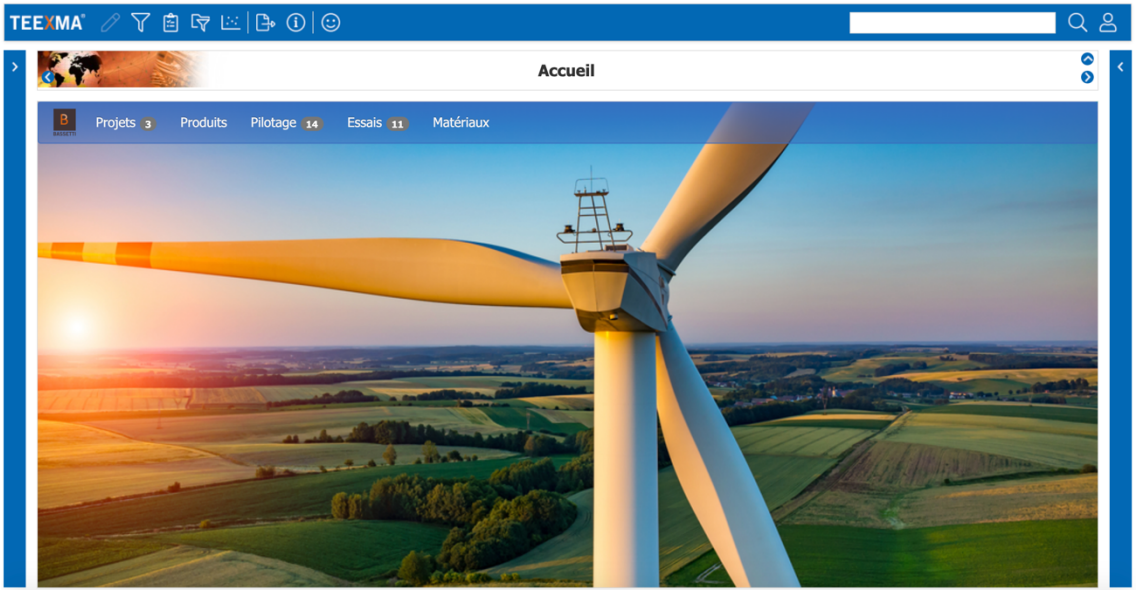
En accédant depuis sur internet à l'adresse suivante : (icam-teexma-qa.teexma.com/teexma?idObject=5) nous trouvons la page suivante ou nous devons introduire les paramètres suivants :

Utilisateur : Daniel MASSETTI
Mot de passe : ddddd

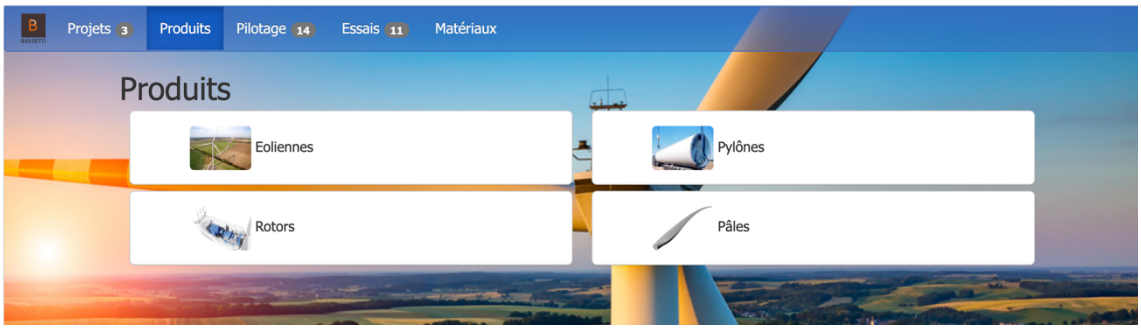


Maintenant il nous amène au portail principal du logiciel dans lequel nous pouvons visualiser différentes options :

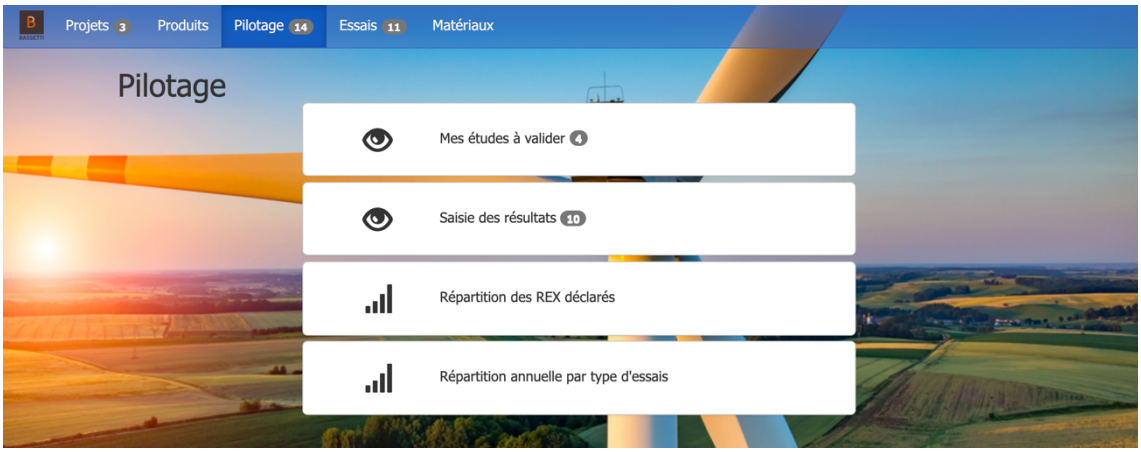
1- Les différents projets dans lesquels nous sommes impliqués.



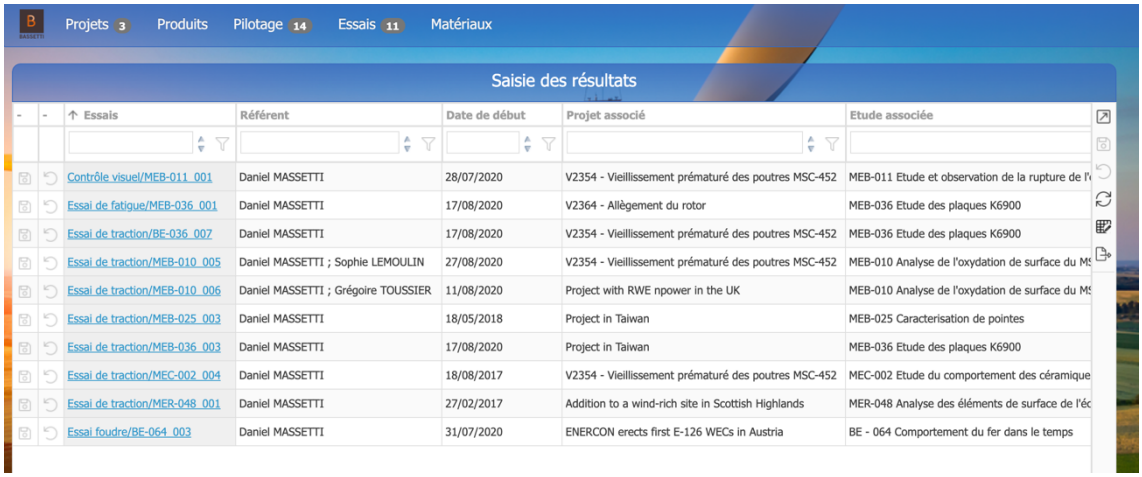
2- Les produits dont dispose chaque client, dans notre cas :



3- Le pilotage, où nous trouvons les études qui doivent encore être validées, les résultats des projets déjà testés et les graphiques que nous pouvons voir dans l'image suivante :



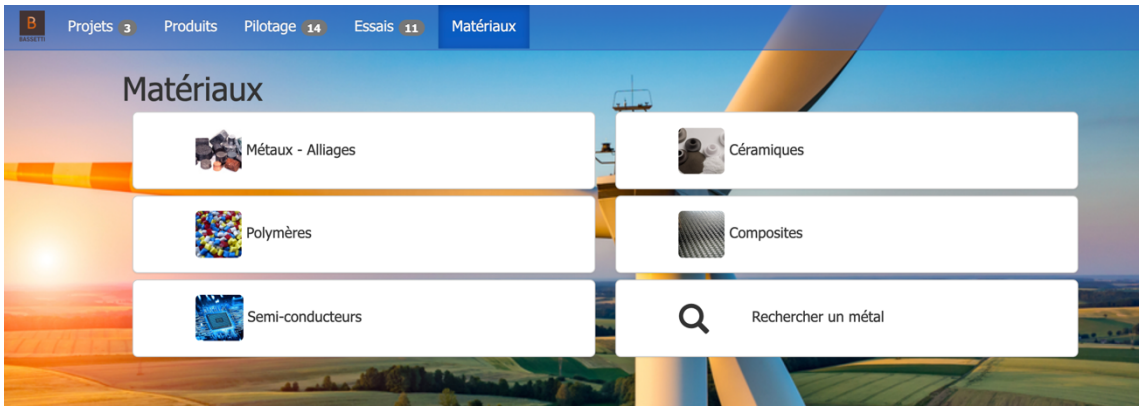
Si nous allons dans les résultats obtenus, nous pouvons voir pour chaque test la personne affectée au projet, les dates et les études associées.



Essais	Référent	Date de début	Projet associé	Etude associée
Contrôle visuel/MEB-011_001	Daniel MASSETTI	28/07/2020	V2354 - Vieillissement prématuré des poutres MSC-452	MEB-011 Etude et observation de la rupture de l'é
Essai de fatigue/MEB-036_001	Daniel MASSETTI	17/08/2020	V2364 - Allègement du rotor	MEB-036 Etude des plaques K6900
Essai de traction/BE-036_007	Daniel MASSETTI	17/08/2020	V2354 - Vieillissement prématuré des poutres MSC-452	MEB-036 Etude des plaques K6900
Essai de traction/MEB-010_005	Daniel MASSETTI ; Sophie LEMOULIN	27/08/2020	V2354 - Vieillissement prématuré des poutres MSC-452	MEB-010 Analyse de l'oxydation de surface du MS
Essai de traction/MEB-010_006	Daniel MASSETTI ; Grégoire TOUSSIER	11/08/2020	Project with RWE npower in the UK	MEB-010 Analyse de l'oxydation de surface du MS
Essai de traction/MEB-025_003	Daniel MASSETTI	18/05/2018	Project in Taiwan	MEB-025 Caracterisation de pointes
Essai de traction/MEB-036_003	Daniel MASSETTI	17/08/2020	Project in Taiwan	MEB-036 Etude des plaques K6900
Essai de traction/MEC-002_004	Daniel MASSETTI	18/08/2017	V2354 - Vieillissement prématuré des poutres MSC-452	MEC-002 Etude du comportement des céramique
Essai de traction/MER-048_001	Daniel MASSETTI	27/02/2017	Addition to a wind-rich site in Scottish Highlands	MER-048 Analyse des éléments de surface de l'éc
Essai foudre/BE-064_003	Daniel MASSETTI	31/07/2020	ENERCON erects first E-126 WECs in Austria	BE - 064 Comportement du fer dans le temps

4- Les tests qui sont en cours d'exécution.

5- Enfin, les matériaux disponibles :

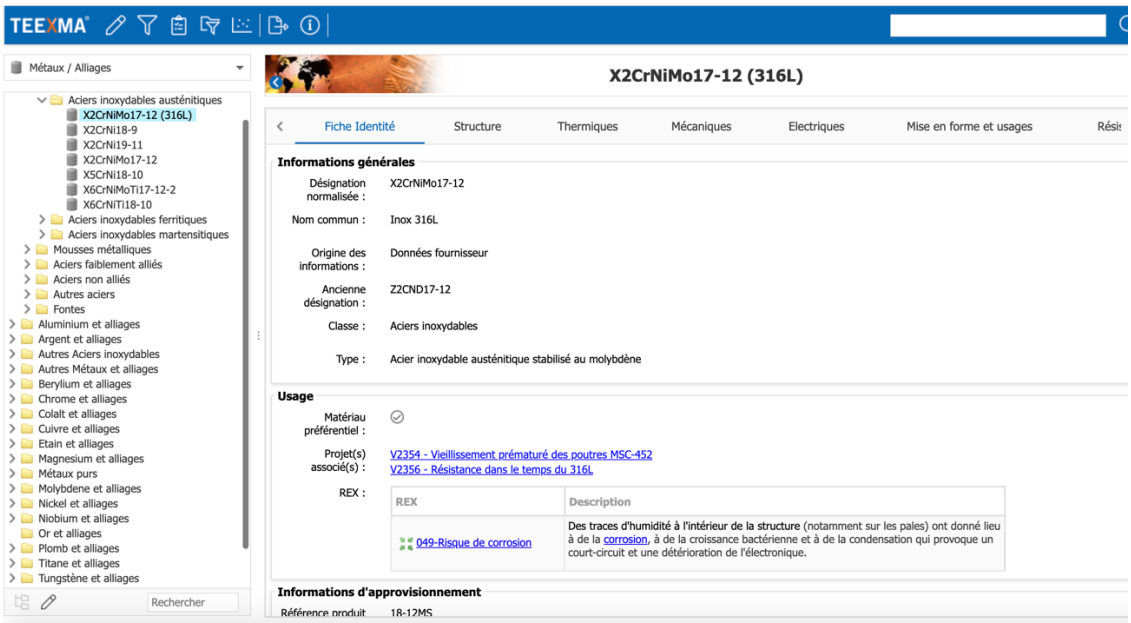


Nous pouvons voir qu'il nous donne l'option de rechercher un matériel spécifique que nous aimerions rechercher, c'est là que mon rôle entre en jeu.

Dans cette version de démonstration, on m'a fourni une base de données d'un parc éolien afin que je puisse tester le fonctionnement du moteur de recherche. Dans ce cas, la base de données n'est pas très grande et il n'y a pas trop de problème pour trouver avec pertinence le résultat souhaité, mais dans le cas de clients réels qui ont une base de données très étendue, la recherche n'est plus précise et il est très difficile de trouver le document souhaité.

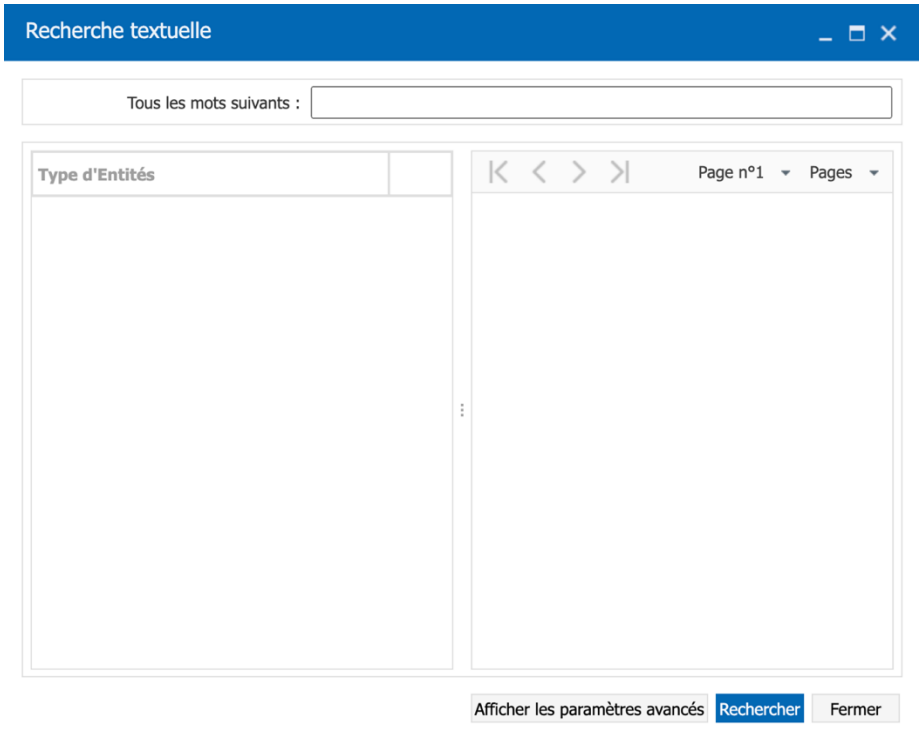
6.2 Types de recherche

En cliquant sur l'option pour effectuer une recherche, on accède au menu avec tous les matériaux disponibles.

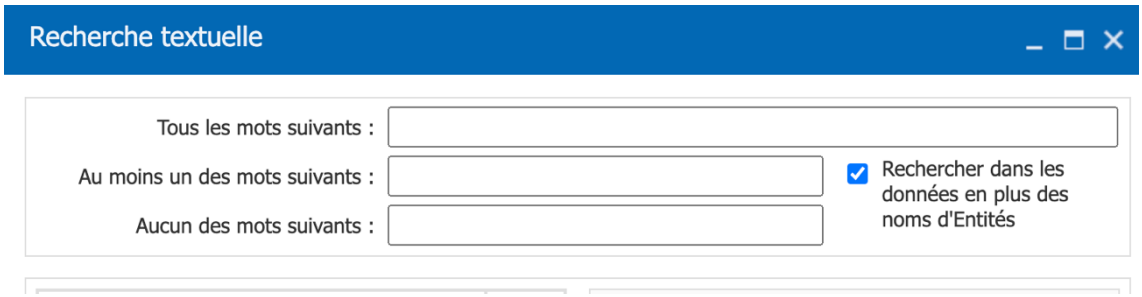


6.2.1 Recherche textuelle

La première option, la plus simple, est la recherche textuelle, par mots ou par phrases.

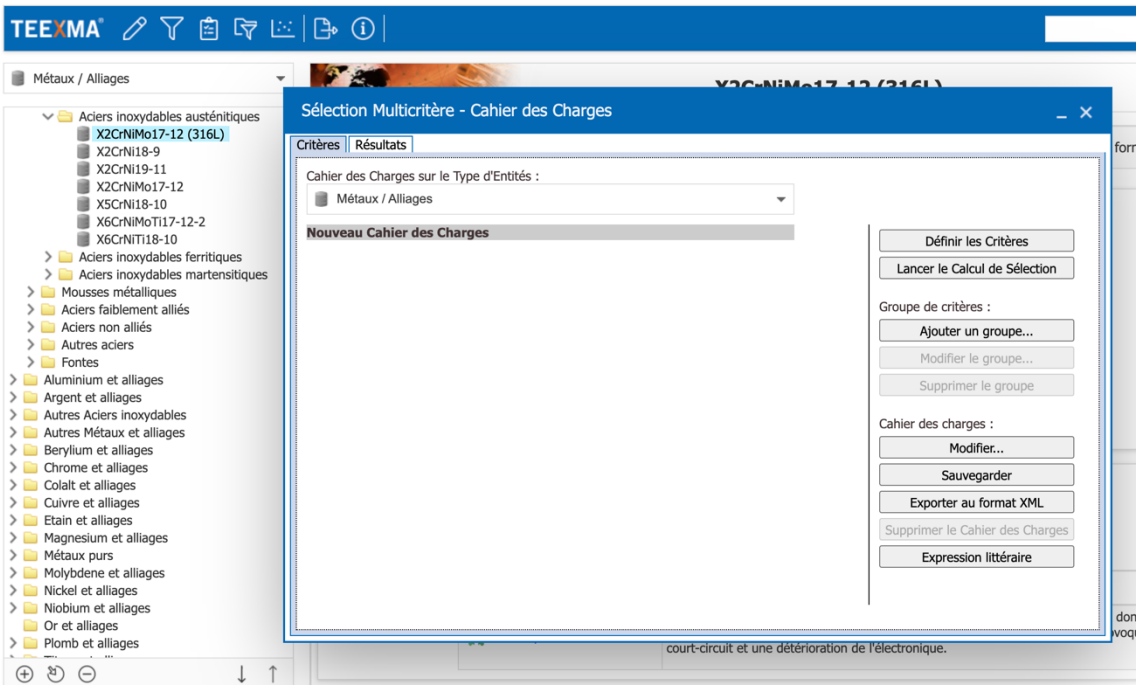


Si nous voulons rendre cette recherche plus précise, nous pouvons également demander au moteur de recherche que les résultats qu'il nous donne contiennent au moins un des mots que nous demandons et il nous donne aussi la possibilité d'entrer les mots que nous sommes sûrs de ne pas vouloir figurer dans les documents.

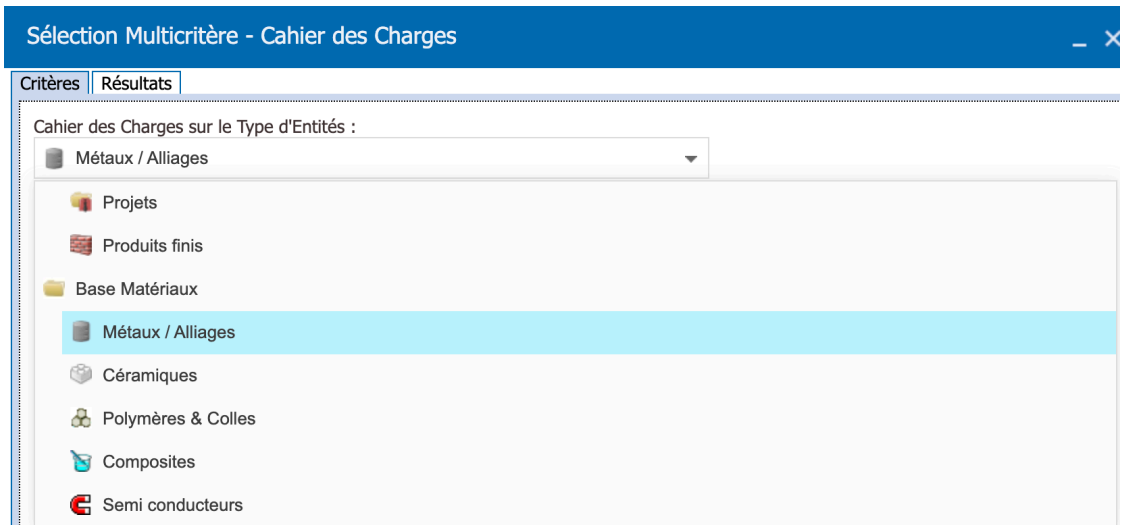


6.2.2 Recherche par sélection multicritère

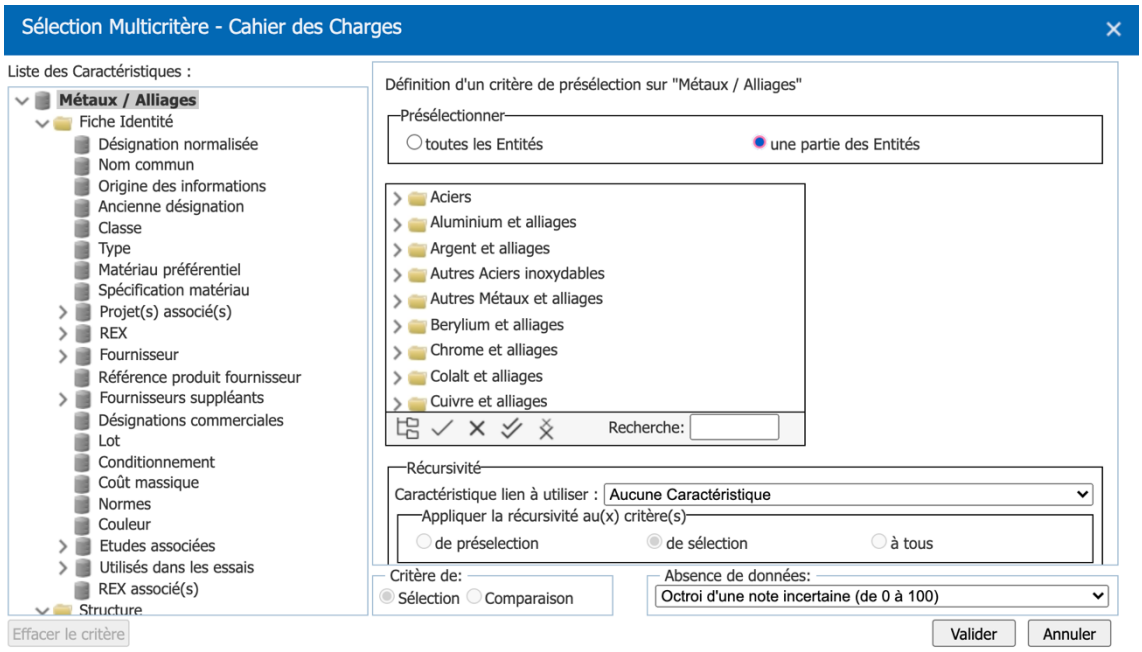
Le logiciel dispose d'une recherche multicritère où vous pouvez définir le type de recherche que vous souhaitez.



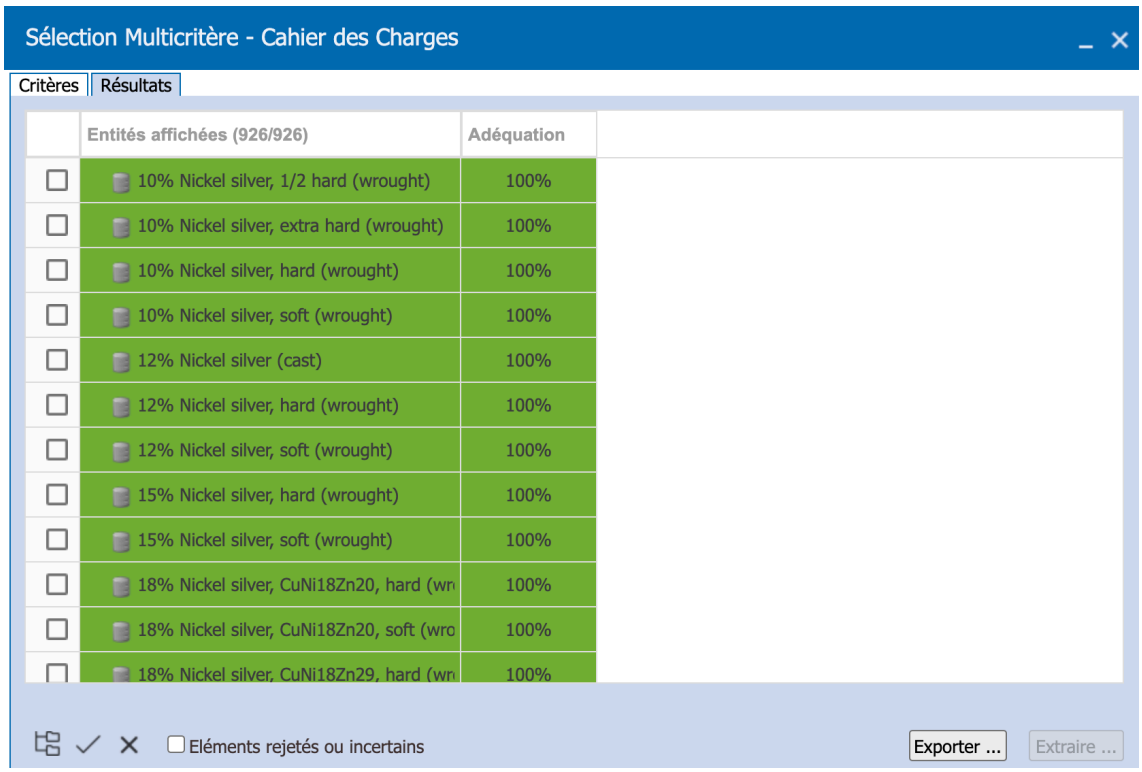
Ici, vous pouvez sélectionner le ou les répertoires dans lesquelles vous voulez que la recherche soit effectuée :



Il donne également la possibilité d'attribuer un critère de présélection au sein du groupe sélectionné, s'il s'agit d'une entité spécifique ou de plusieurs entités :



Enfin, il renvoie comme résultat les différentes entités dans lesquelles notre recherche a été effectuée.

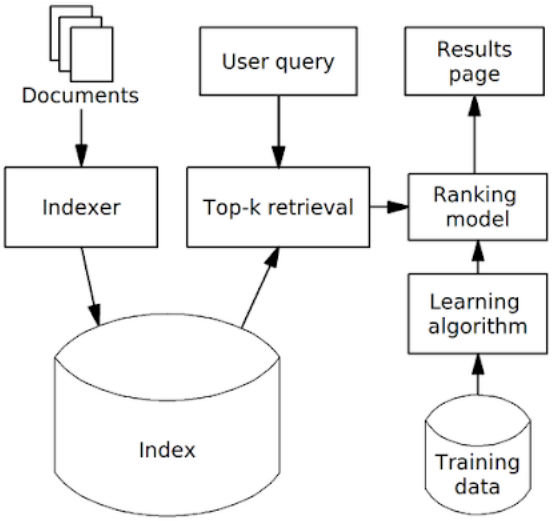


Une fois que nous avons pu tester le fonctionnement de la recherche, nous nous rendons compte qu'il y a certains facteurs qui pourraient être ajoutés pour l'améliorer et dont nous parlerons plus tard.

Dans ce qui suit, je vais présenter différentes solutions applicables.

7. HYPOTHESES D'AMELIORATION

7.1 Top k-retriveal



Avant d'effectuer le classement final, nous effectuons généralement un premier filtrage que nous appelons Top-k Retrieval avec les documents potentiellement pertinents. Nous ne pouvons pas analyser tous les documents car cela prendrait trop de temps à l'algorithme, il est vrai que dans une petite base de données il n'y aurait pas de problème mais nous ne nous trouvons généralement pas dans ce cas.

7.2 Élévation

L'élévation des résultats lors d'une recherche consiste en un ensemble d'actions visant à améliorer l'expérience de recherche d'un utilisateur en fonction de ses recherches précédentes.

Si nous voulons ajuster la pertinence des résultats de recherche dans Solr, dans notre cas, en fonction de l'utilisateur qui l'utilise, sur la base du nombre de fois que cet utilisateur a cliqué sur un résultat avant, nous pouvons appliquer cette fonctionnalité.

Il fournirait une élévation personnalisée pour des identifiants de documents particuliers dans la requête, dans l'ordre que nous souhaitons. C'est-à-dire qu'il montrerait d'abord le nombre n de documents résultants de cette élévation et ensuite ceux qui sortiraient avec une recherche normale.

Pour mettre en œuvre cela, j'ai pensé à configurer un champ dynamique pour stocker les clics de l'utilisateur.

Nous initialiserions à 0 par défaut le nombre de clicks pour chaque document indexé et chaque fois que l'utilisateur ouvrirait un document, cela additionnerait le nombre de clicks qui serait stocké dans une variable avec le format suivant, par exemple :

clickNombre = clickNombre + 1.

Il est intéressant de noter que cette technique est l'une des principales techniques utilisées par Google pour afficher les pages Web qu'il considère comme plus pertinentes et utiles pour l'utilisateur. Dans notre cas, nous voulons appliquer la même chose mais pour des documents dans une base de données.

7.3 Word2Vec

Pour comprendre le fonctionnement de Word2Vec, qui appartient aux algorithmes de Word embedding, il est nécessaire d'en savoir plus sur des différents facteurs du traitement de langue naturel pour le traitement des textes. En gros ça permet une machine de comprendre ce que les mots veulent dire et les comparer mathématiquement.

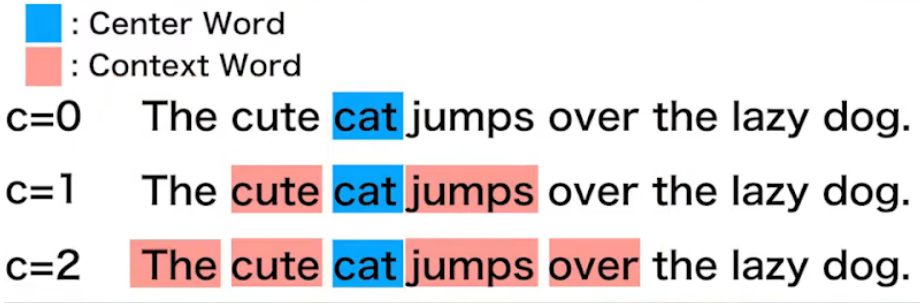
7.3.1 Word Embedding

Le Word embedding permet d'améliorer la recherche d'information. Imagine que nous cherchons dans un moteur de recherche, qui utilise le Word embedding, un restaurant qui remonte le moral. On écrit, par exemple, hamburger dans la barre de recherche et ce moteur va étendre notre recherche en ajoutant d'autre mots comme hamburger, cheeseburger, burger, sandwich etc. Avec ce type de recherche nous allons trouver un contenu plus proche à ce que nous recherchons même si nous ne l'avons pas précisé explicitement. Il est trop pratique parce que ça répond a la problématique du silence dans la recherche d'information, c'est-à-dire quand on fait un recherche mais on a aucun résultat.

7.3.2 Comprendre le sens des mots

Pour comprendre le sens des mots nous utilisons l'hypothèse distributionnelle. Les mots qui se trouvent dans un même contexte linguistique, c'est-à-dire les mots qui se trouvent avant et après du mot que nous observons, ont un sens qui est proche.

Par exemple dans le cas suivant :



Ici, nous avons le mot observé, appelé le mot central, que dans ce cas est chat (cat en bleu) et son contexte en rouge. Pour la taille du contexte on peut prendre 1, 2, 3 mots etc. L'idée c'est que si le mot chien est statistiquement entouré par le même mot que le mot chat alors cela veut dire que le sens du mot chien est proche du sens du mot chat.

7.3.3 Modèle vectoriel

Représenter un texte avec la forme d'un vecteur, c'est-à-dire une liste de valeurs numériques, c'est la manière plus simple de compter les mots. Nous allons prendre, par exemple, 3 phrases :

```
phrases = [
  "le chat mange ses croquettes",
  "le chien aime ses croquettes",
  "le chat ronronne et mange"
]
```

Maintenant, nous allons transformer ces phrases en un vecteur en comptant les mots et nous avons le résultat suivant :

	aime	chat	chien	croquettes	et	le	mange	ronronne	ses
phrase_1	0	1	0	1	0	1	1	0	1
phrase_2	1	0	1	1	0	1	0	0	1
phrase_3	0	1	0	0	1	1	1	1	0

Le problème avec cette manière de créer un vecteur en comptant les mots, c'est que les mots n'ont tous le même poids. Si nous observons la 3ème phrase, le mot « le » a autant de poid que le mot « ronronne », mais intuitivement on sait bien que le mot « ronronne » contient plus d'information sémantique que le déterminant « le ».

La solution à ce problème propose d'utiliser une autre valeur numérique que le nombre de mots, ce que s'appelle le TF-IDF (term frequency inverse document frequency).

7.3.4 TF-IDF

Le terme « frequency » c'est la fréquence d'un mot dans un document spécifique.

Par exemple dans la phrase trois, il y a quatre mots, et le mot « ronronne » apparait une seule fois donc sa fréquence est de 25%. Pour un autre côté, la document frequency c'est la fréquence des documents qui contiennent ce mot. Ici il y a trois documents (trois phrases) dont un seul contient le mot « ronronne » pourtant sa document frequency est donc de 33%.

En gros, le TF-IDF il fait un peu un arbitrage entre ces deux mesures. Un mot qui apparaitrait dans très peu documents mais qui est très présent dans un document spécifique il va avoir un score plus élevé alors qu'on mot qui apparait dans tous les documents il va avoir un score plus bas.

Voici la formule :

TF-IDF = TF (t, d) x IDF (t)

TF : Nombre de fois que le terme t apparait dans un document d.
IDF : Fréquence inverse du document.

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Dans la première partie de la formule (fréquence des termes), une série de modificateurs ou de pondérations sont appliqués aux termes, par exemple :

- Calcul de la fréquence en tenant compte de la longueur du document lui-même.
- Ajustement de la fréquence des termes selon une échelle logarithmique. Au-delà d'une certaine valeur, il est presque inutile de continuer à inclure ce terme.
- Appliquez une normalisation ou une correction, afin que les textes longs ne soient pas davantage récompensés. Par exemple, en divisant la fréquence du terme par la fréquence du terme qui apparaît le plus souvent dans le texte.

Si on utilise cette méthode nous avons les résultats suivants :

	aime	chat	chien	croquettes	et	le	mange	ronronne	ses
phrase_1	0.00	0.47	0.00	0.47	0.00	0.36	0.47	0.00	0.47
phrase_2	0.53	0.00	0.53	0.41	0.00	0.32	0.00	0.00	0.41
phrase_3	0.00	0.41	0.00	0.00	0.53	0.32	0.41	0.53	0.00

Nous pouvons comparer le vecteur qu'on vient d'obtenir avec le vecteur qui comptaient les mots tous simplement. On voie que pour la troisième phrase le mot « ronronne » a maintenant un score plus élevé que le déterminant « le » et ça c'est tout simplement parce qu'il apparait dans moins de documents. Sa document frequency est plus faible et donc il a potentiellement plus de valeur sémantique.

7.3.5 Calculer la distance entre deux textes

Maintenant nous savons comment obtenir un vecteur à partir d'une phrase ou un document en suite on peut calculer la distance entre deux phrases ou documents.

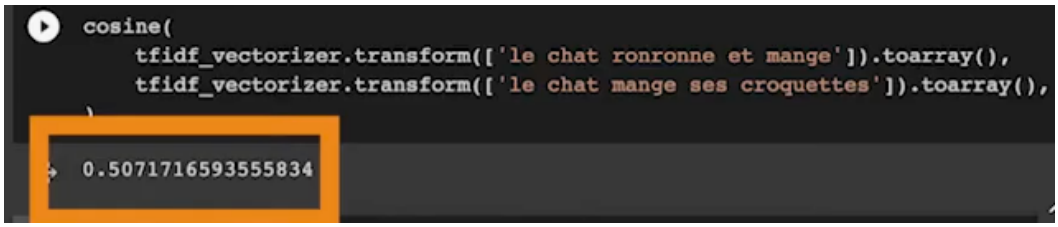
Voyez que dans notre vecteur ici il y a trois phrases qui sont les 3 lignes el il y a 9 mots qui sont les colonnes et bien on peut imaginer que ces 9 mots constituent les neuf dimensions d'un espace vectoriel, du coup chaque phrase, c'est-à-dire chaque vecteur peut- être consideré comme un point dans cet espace à neuf dimensions et ça veut dire du coup, qu'il y a une certaine distance entre ces points.

Cette distance on peut la mesurer, et il s'appelle la distance cosinus :

$$\cos\theta = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}.$$

Je vais donner un exemple :

Si nous utilisons la fonction cosinus de Saïbi, nous allons vérifier à quel point la phrase trois est proche de la phrase un. Au fond, tous les deux contiennent le mot « chat » et le mot « mange » c'est pour ça qu'ils doivent être proche.



En effet ils sont à un distance de cosinus de 50% dans le espace vectoriel de neuf dimensions.

D'autre part, nous voyons que la distance entre la phrase un et la deux c'est de 90% (presque le double) et pourtant ils sont plus éloignés.

7.3.6 Conclusion Word2Vec

Si le sens des mots et liée a leur contexte c'est l'hypothèse distributionnelle, et si le contexte des mots peut être représenté sur la forme de vecteurs et il est possible de calculer la distance entre deux vecteurs, alors nous pouvons mesurer la distance sémantique entre deux mots.

Nous allons prendre une liste de phrases un peu plus grande (6 phrases) :

```
phrases = [
  "le chat mange ses croquettes",
  "le chien dévore ses croquettes",
  "le chat dévore son paté",
  "jean va travailler",
  "le chat mange son repas",
  "jacque aime quand son chien mange"
]
```

Ensuite nous allons prendre chacun des mots de ses phrases et nous allons les isoler par leur contexte :

```
== le ==
['chat', 'chien', 'chat', 'chat']

== chat ==
['le', 'mange', 'le', 'dévore', 'le', 'mange']

== mange ==
['chat', 'ses', 'chat', 'son', 'chien']

== ses ==
['mange', 'croquettes', 'dévore', 'croquettes']

== croquettes ==
['ses', 'ses']

== chien ==
['le', 'dévore', 'son', 'mange']

== dévore ==
['chien', 'ses', 'chat', 'son']

== son ==
['dévore', 'paté', 'mange', 'repas', 'quand', 'chien']

== paté ==
['son']
```

Nous pouvons voir que le mot « le », par exemple, a dans son contexte le mot « chat », « chien », « chat » et « chat ». Et le même pour tous les autres.

Dans ce cas nous avons pris un contexte de taille 1, c'est-à-dire les mots d'avant et ces d'après.

Maintenant nous allons vectorisé ces contextes de la même manière qu'avant :

```
Espace vectoriel de 16 dimensions
['aime', 'chat', 'chien', 'croquettes', 'dévore', 'jacque', 'jean', 'le', 'mange',

== le ==
[0.   0.93 0.37 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.   0.  ]

== chat ==
[0.   0.   0.   0.   0.34 0.   0.   0.74 0.58 0.   0.   0.   0.   0.
 0.   0.  ]

== mange ==
[0.   0.72 0.42 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.42 0.36
 0.   0.  ]

== ses ==
[0.   0.   0.   0.84 0.42 0.   0.   0.   0.35 0.   0.   0.   0.   0.
 0.   0.  ]
```

Nous nous trouvons avec un espace vectoriel que contient 16 dimensions sur les 16 mots différents qui apparaissent dans les 6 phrases. Voyez que pour chaque mot on a un vecteur qui représenté son contexte. Ce vecteur, au fond, il positionne le mot dans l'espace vectoriel de 16 deminsions. Si l'hypothèse distributionnelle est aussi exact ce vecteur du contexte peut être considéré comme des vecteurs de sens.

Nous allons vérifier cette affirmation avec la distance cosinus qui permet mesurer la distance entre deux vecteurs. On va regarder quel et le mot plus proche du mot « chat » :

```
most_similar('chat')

chien    0.795146
```

Le résultat c'est le mot « chien » avec un cosinus de similarité de presque le 80%. Si nous faisons le même pour le mot « chien » nous obtenons la même chose mais a l'inverse, voici les deux :

```
most_similar('chat')

chien    0.795146
dtype: float64

[38] most_similar('chien')

chat    0.795146
dtype: float64
```

Encore nous allons faire le même avec le mot « mange » pour nous assurer qu'il fonctionne :

```
most_similar('mange')

dévore    0.946995
dtype: float64
```

7.4 BM25

La fonction BM25 s'agit de la nouvelle formule utilisée par Lucene pour le traitement des textes. Au lieu du traditionnel "TF-IDF", Lucene est passé à quelque chose appelé BM25 dans le tronc. Cela signifie une nouvelle formule de notation pour Solr et Elasticsearch.

Cette fonction améliore la recherche TF-IDF. Il combine :

- La formule de l'IDF expliquée avant.
- La fréquence des termes du document normalisée par le rapport entre la longueur du document et la longueur moyenne.
- La fréquence des termes de la requête.

BM25 est basé sur le concept de sac de mots par lequel on représente les documents que l'on veut trier en fonction de leur pertinence par rapport à une requête donnée.

Étant donné une requête Q, contenant les mots-clés q1, ..., qn, la valeur de pertinence attribuée par la fonction BM25 pour le document D sera :

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

- f (qi, D) : fréquence d'occurrence dans le document D des termes apparaissant dans la requête Q.
- /D/ est la longueur du document D (en nombre de mots).
- avgdl : longueur moyenne des documents de la collection sur laquelle la recherche est effectuée.

- K1, b : termes qui permettent d'ajuster la fonction aux caractéristiques spécifiques de la collection.

- IDF(qi) : poids IDS (inverse document frequency) des mots-clés de la requête Q.

8. SOLUTIONS DANS SOLR

8.1 Implémenter notre propre moteur de recherche dans Solr

8.1.1 Acquisition et installation de Solr

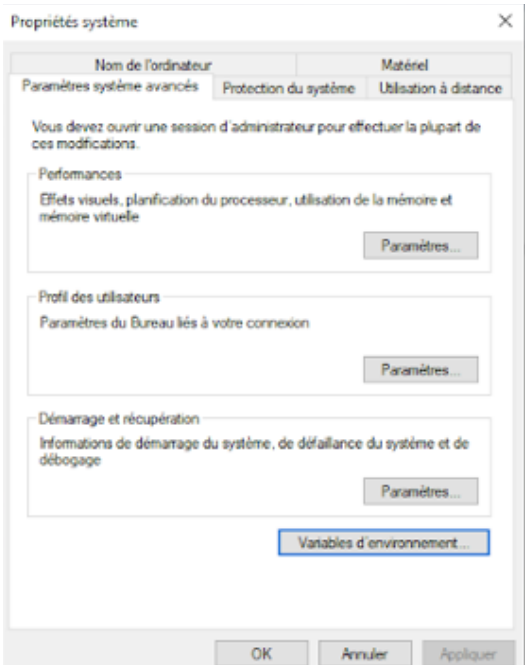
❖ ***Installation de JAVA*** *:*

1 - Au préalable, il faudra installer un JRE (java exécution environnement) afin de pouvoir utiliser Solr. Vous pouvez télécharger sur le site de java :

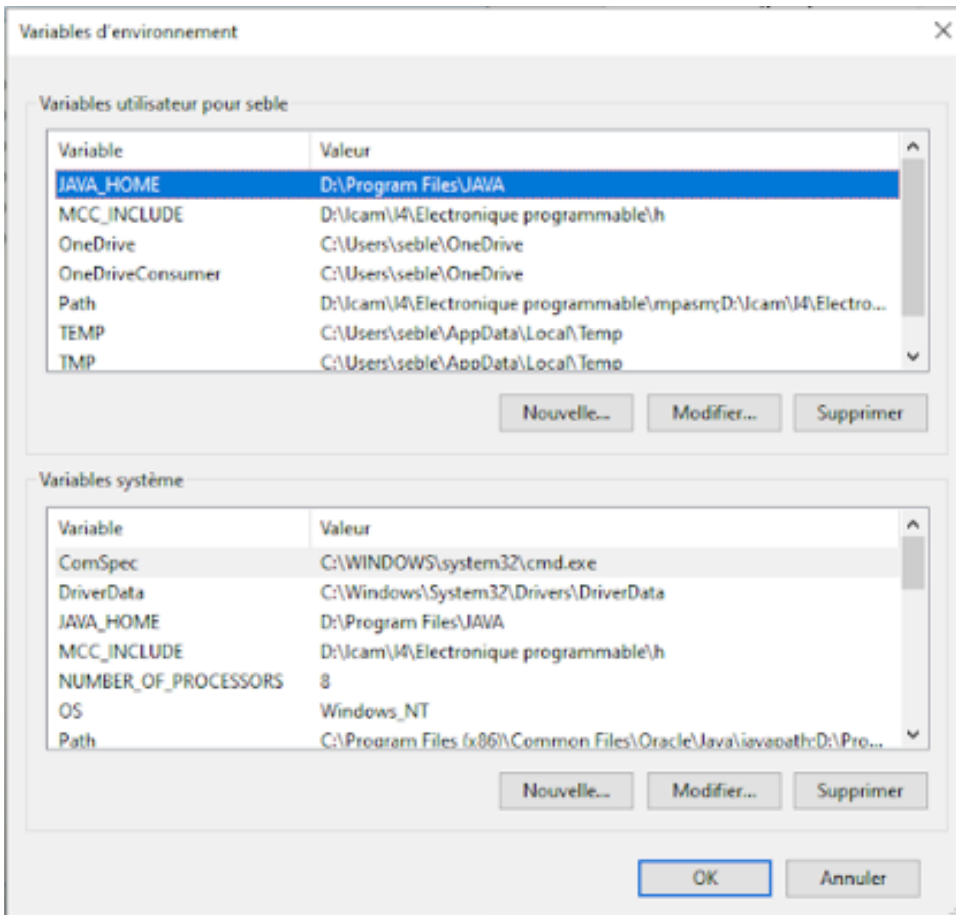
https://www.java.com/fr/download/

Lancer ensuite le fichier téléchargé pour installer Java.

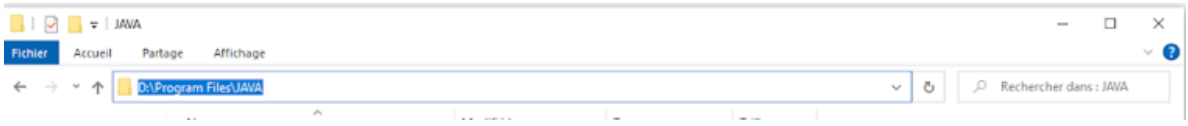
2 – Ensuite, taper dans la barre de recherche Windows *Modifier les variables d'environnement systèmes.* Vous aurez ensuite la fenêtre ci-dessous qui apparaîtra :

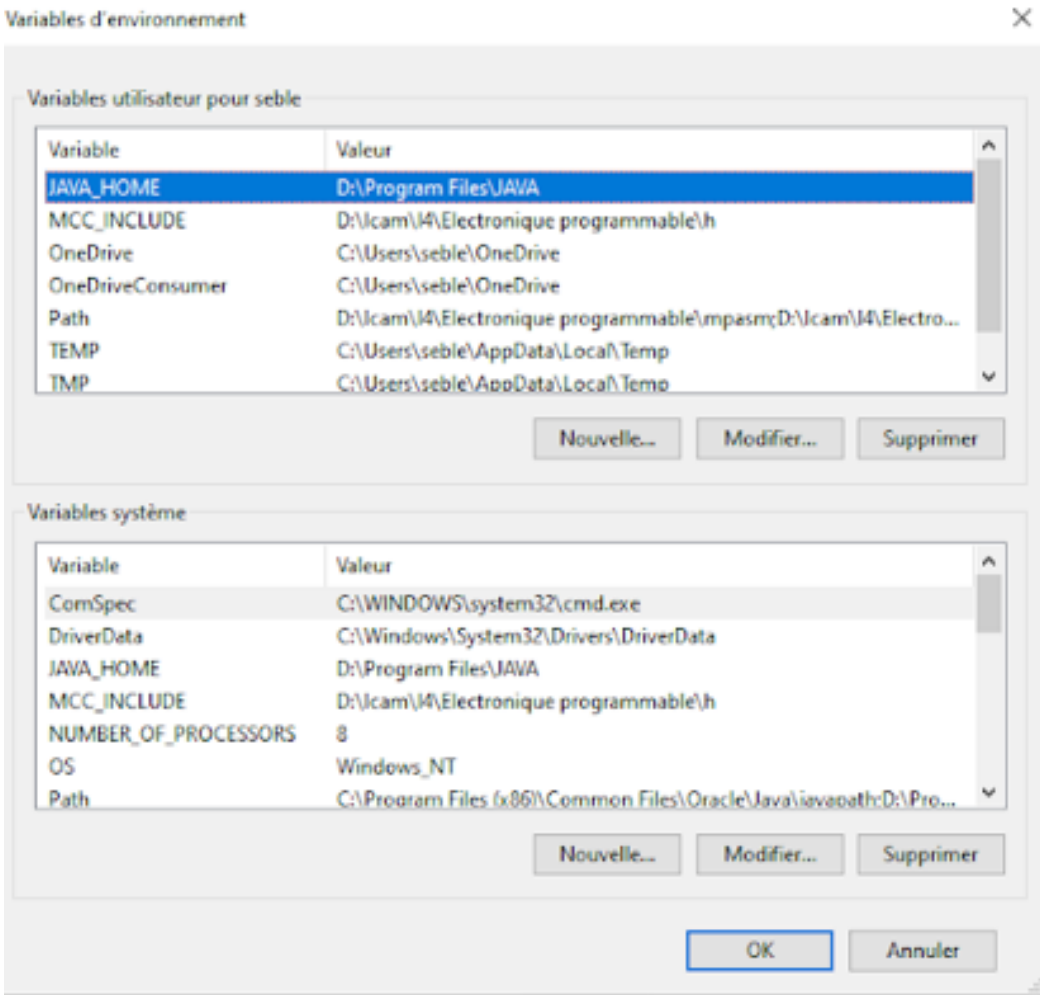


Clique sur Variables d'environnement la fenêtre suivante apparaîtra :

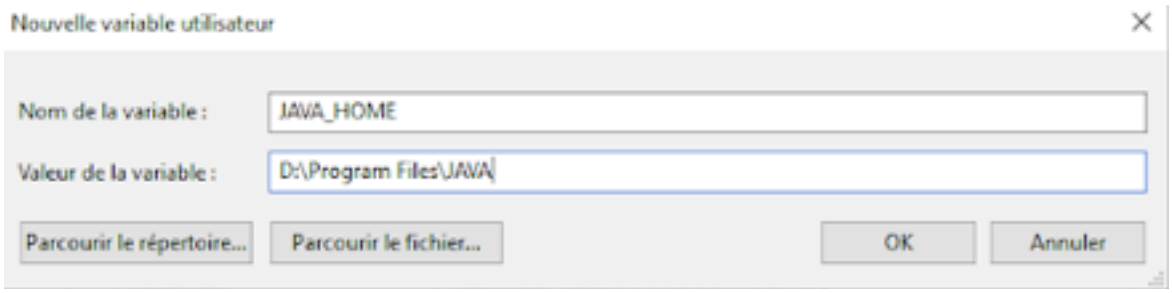


Vous allez ensuite copier l'adresse de l'emplacement où vous avez installé Java.





Pour chaque bouton *Nouvelle…* entrez **JAVA_HOME** dans *Nom de la variable* et l'adresse de l'emplacement d'installation dans *Valeur de la variable.*

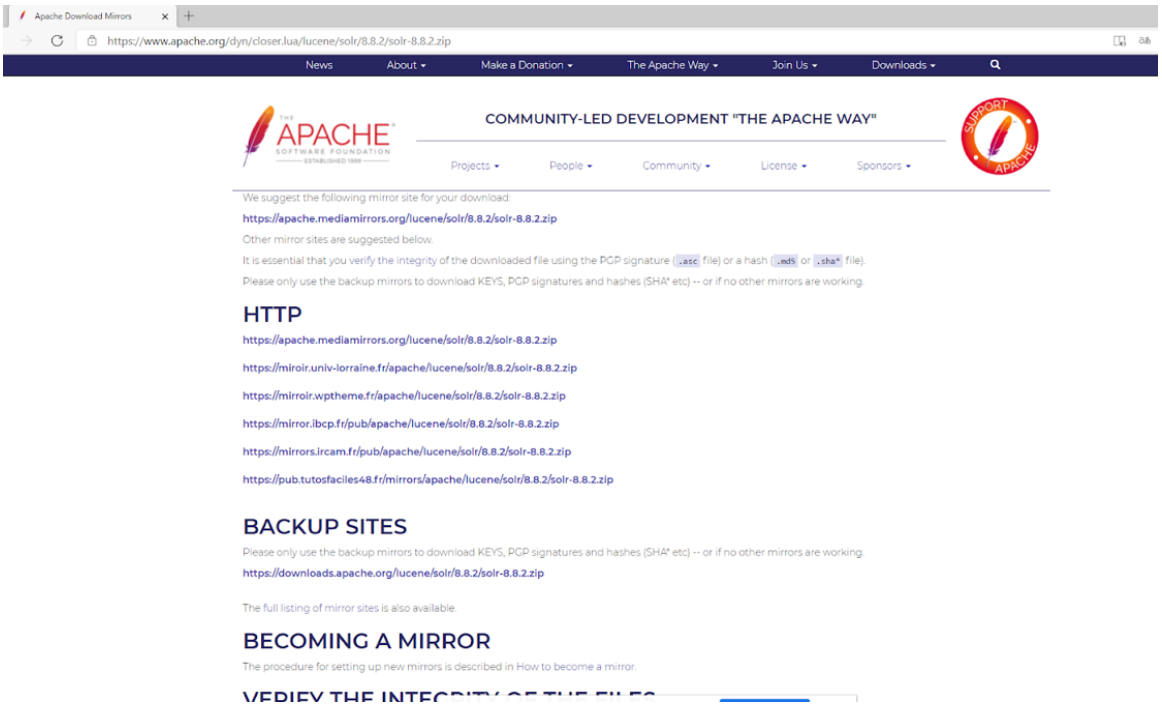


❖ ***Installation de Solr***

1 - Téléchargez la version de Solr 8.8.2 en faisant attention de prendre la bonne extension correspondant à votre système d'exploitation (voir ci-dessous) sur le lien suivant https://solr.apache.org/downloads.html :

- **.tgz** pour les systèmes Linux / Unix / OSX.
- **.zip** pour les systèmes Microsoft Windows.

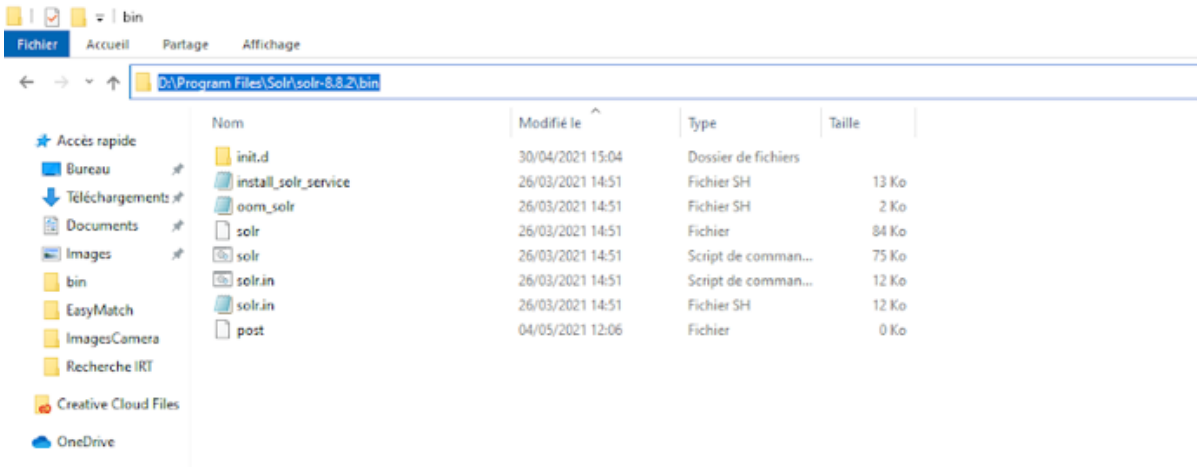
Vous arriverez ensuite sur cette page :



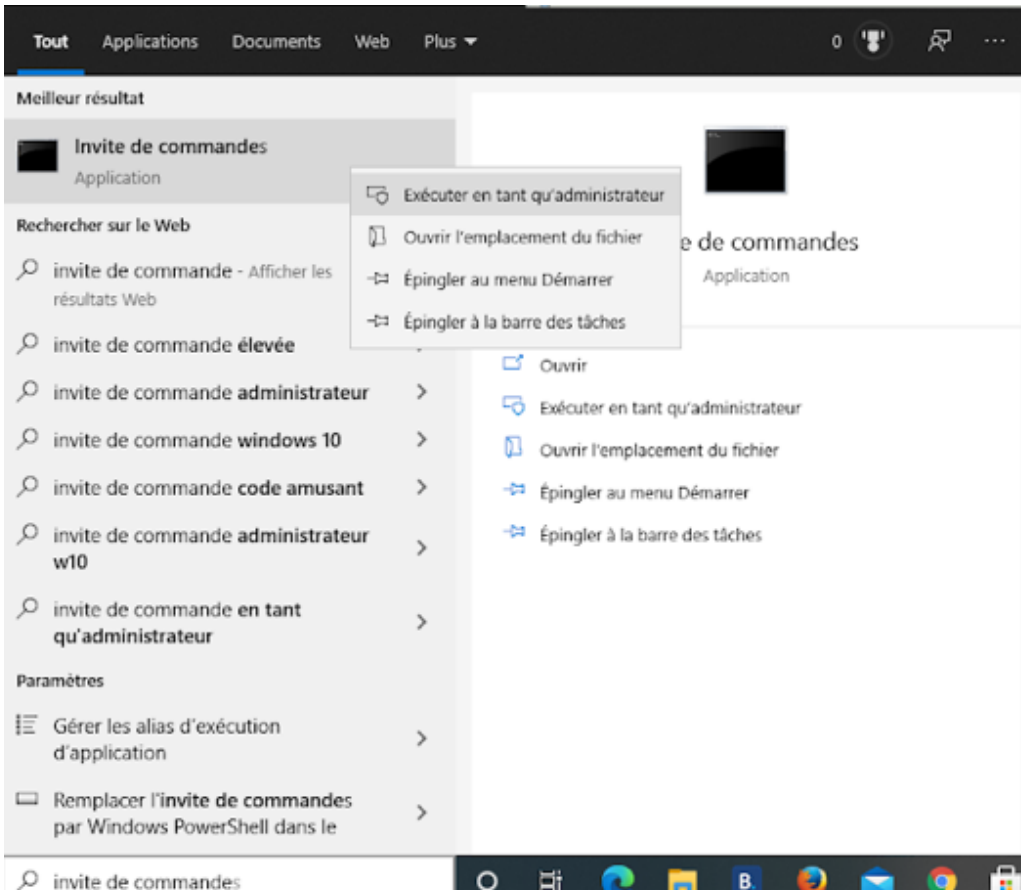
Cliquez alors sur le lien en-dessous de HTTP.

2 – Vous avez maintenant téléchargé un fichier compressé contenant la version de Solr que vous avez téléchargé. Décompressez-le dans le répertoire que vous souhaitez (avec WinRAR ou 7zip par exemple), de préférence dans le *Program Files* ou le dossier d'installation des logiciels.

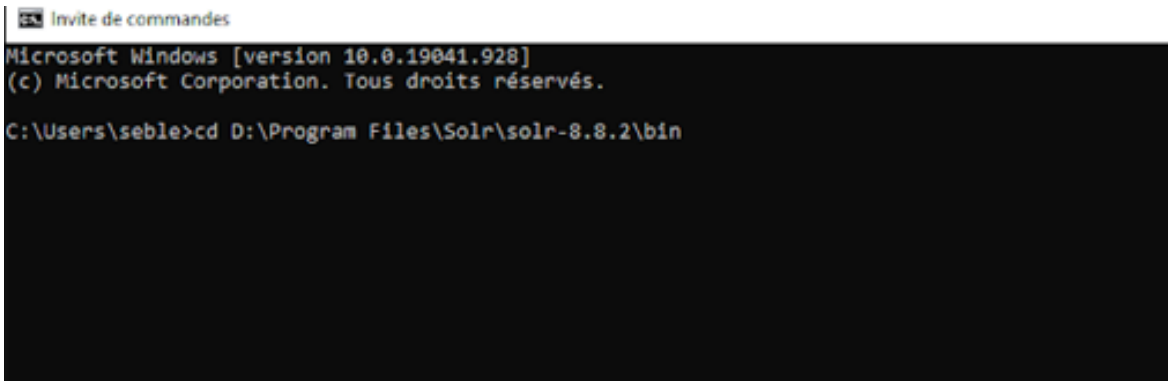
Ensuite allez dans le dossier de Solr puis dans le dossier bin et copier l'adresse d'emplacement de ce fichier.



3 – Ensuite cherchez l'*Invite de commande* dans la barre de recherche Windows et lancez-le en tant qu'administrateur

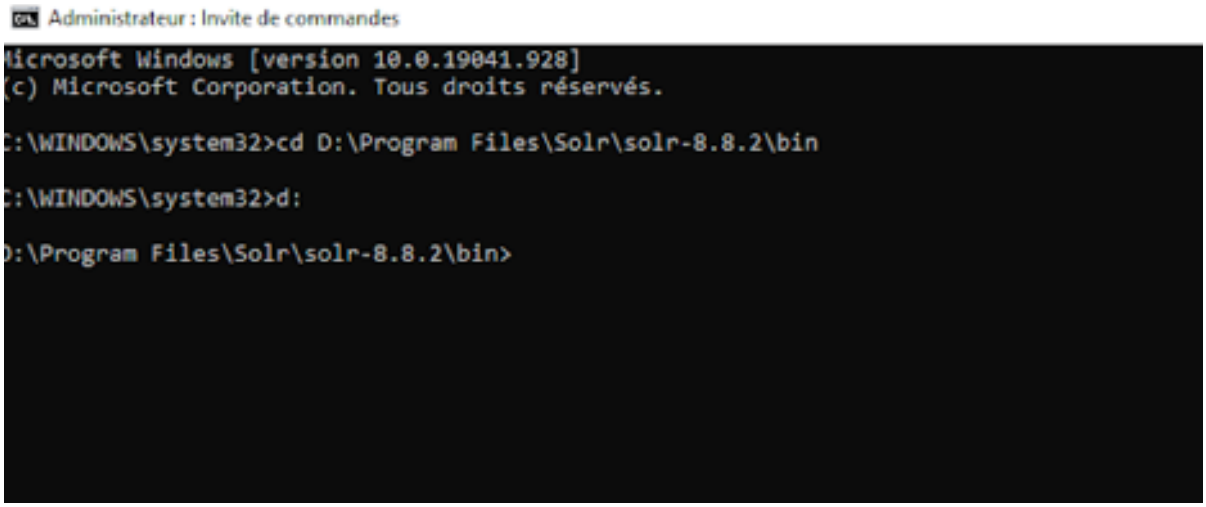


Vous arrivez sur l'invite de commande. Tapez *cd* suivi de l'emplacement du fichier que vous avez copier précédemment.

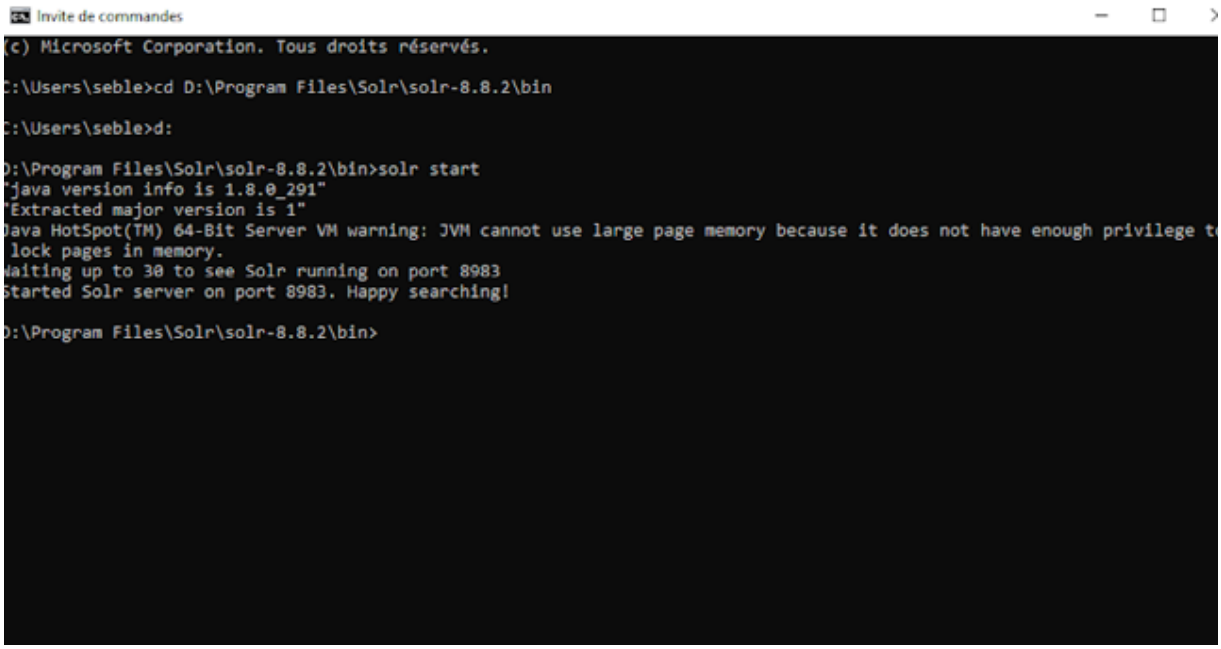


Si vous avez besoin de changer de disque dur pour aller à l'emplacement de Solr (par exemple pour passer de C: à D:), tapez ensuite *d:* pour accéder à l'emplacement du fichier de Solr depuis l'invite de commande.

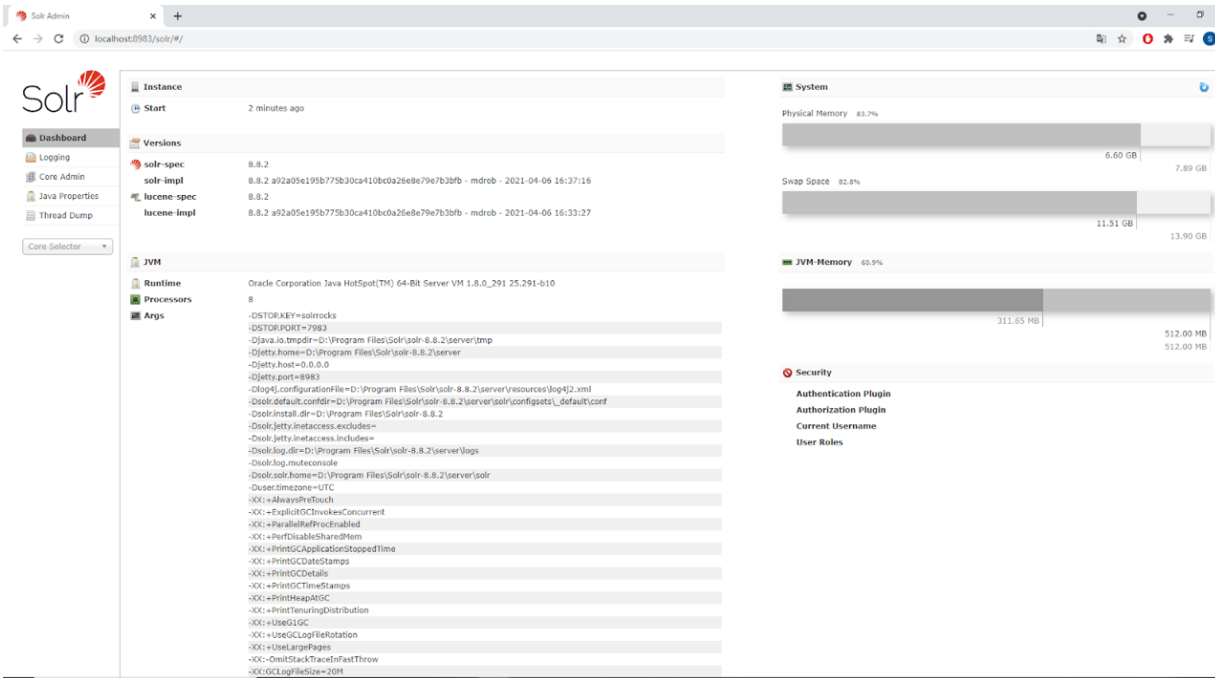
Sinon passez à la suite.



Tapez ensuite *solr start* pour démarrer Solr lorsque vous êtes à l'emplacement du ficher d'installation de Solr (ici D:\Program Files\Solr\solr-8.8.2\bin). Par défaut, Solr démarrera sur le port 8983. Si une fenêtre « windows security alert » apparaît, cliquez sur *Allow Acces*.



4 – Ouvrez maintenant votre moteur de recherche et entrez *localhost :8983*



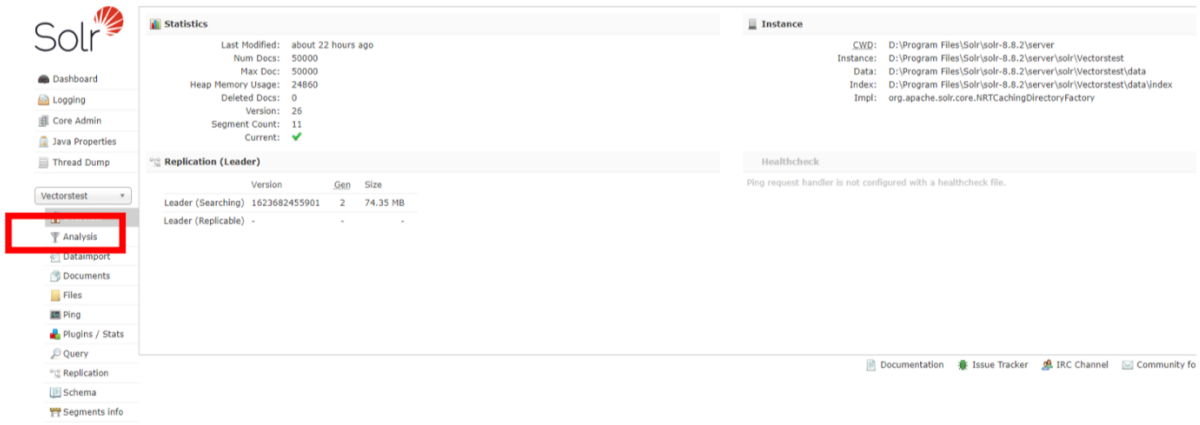
Vous pouvez maintenant utiliser Solr !

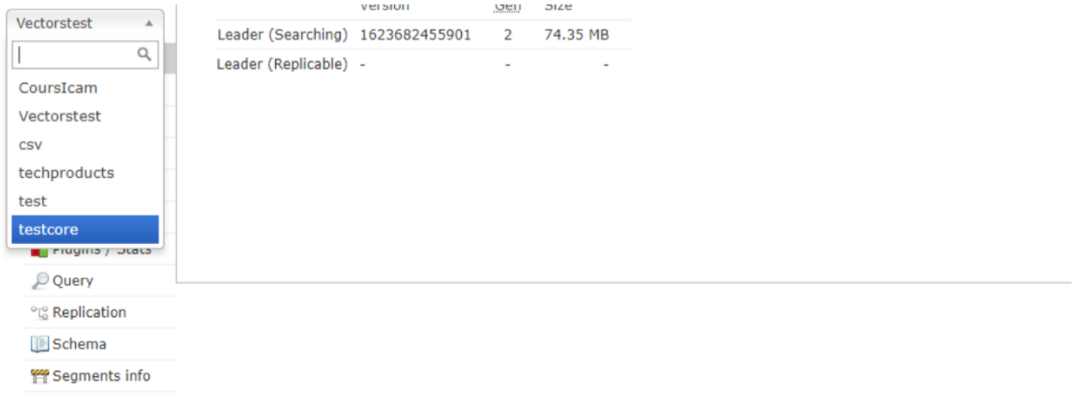
8.1.2 Indexer des données avec un fichier .csv

Pour pouvoir indexer des données, il faut créer un *nœud (core)*.
Pour cela, allez dans l'*invite de commande (command prompt)* et tapez la commande suivante : solr create -c *nom_du_noeud.*
A la place de *nom_du_noeud* mettez le nom que vous souhaitez donner à votre nom (ici testcore) :

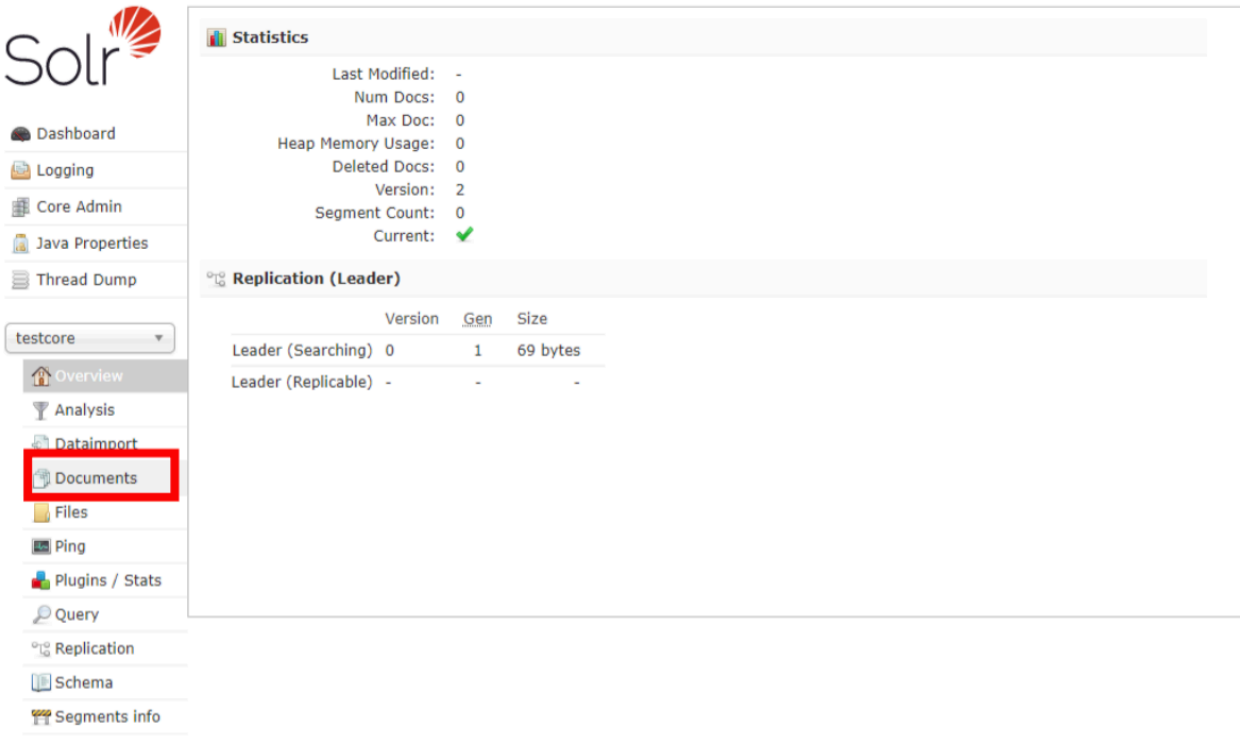
```
D:\Program Files\Solr\solr-8.8.2\bin>solr create -c testcore
"java version info is 1.8.0_291"
"Extracted major version is 1"
WARNING: Using _default configset with data driven schema functionality. NOT RECOMMENDED for production use.
         To turn off: bin\solr config -c testcore -p 8983 -action set-user-property -property update.autoCreateFields -value false

Created new core 'testcore'

D:\Program Files\Solr\solr-8.8.2\bin>
```

Dans Solr, allez dans le menu déroulant et cliquez sur le nom de votre nœud :





Allez ensuite dans l'onglet *Documents* :

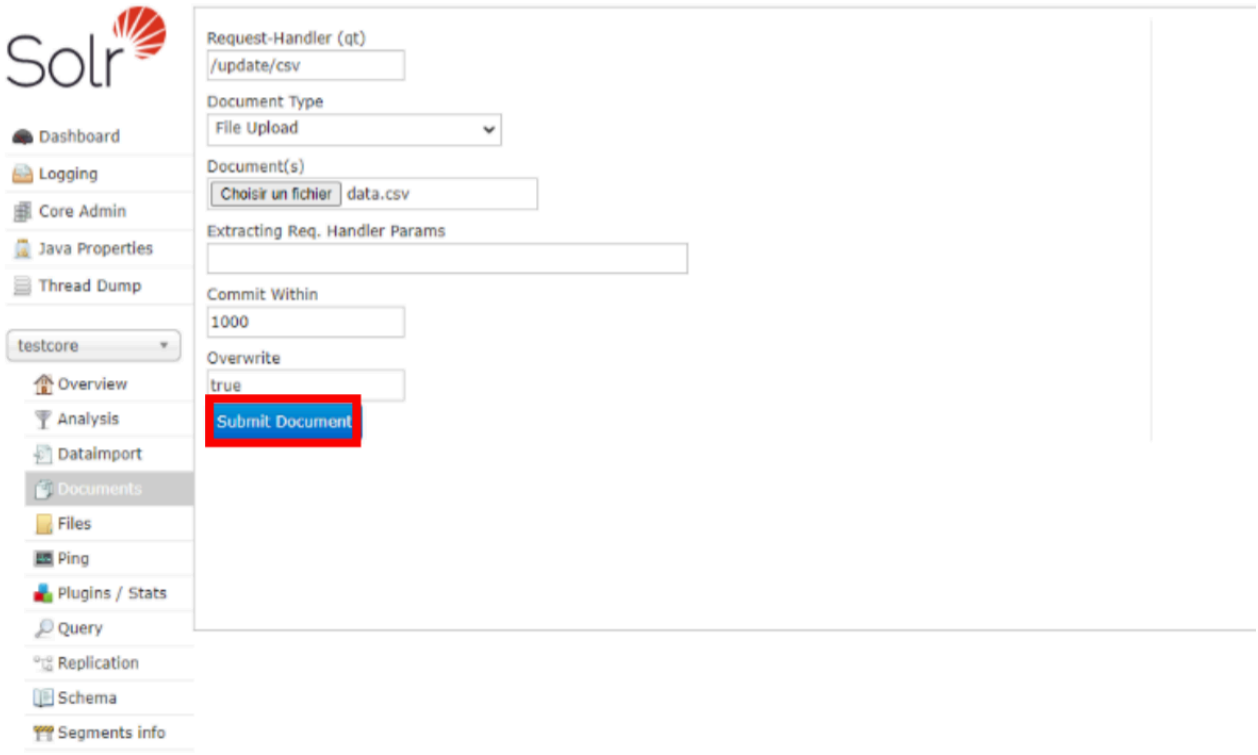


- Dans *Request Handler* tapez /update/csv.

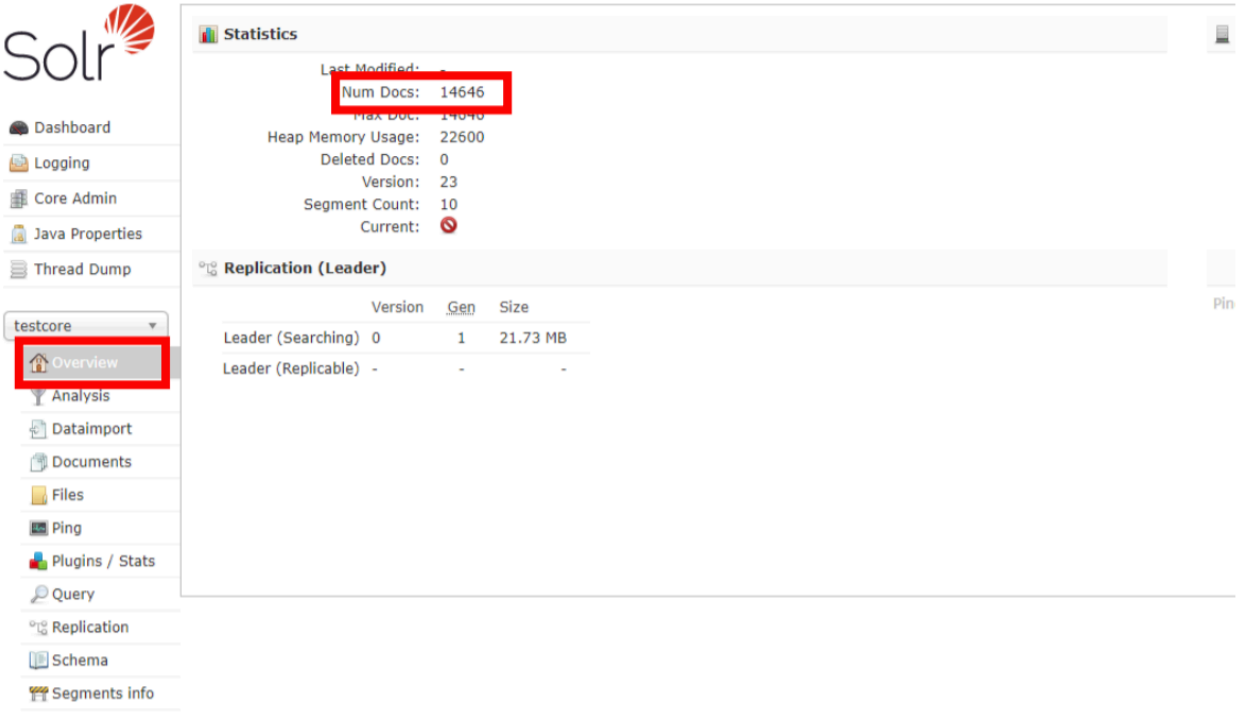
- Dans *Document type* choisissez *File upload.*

- Puis dans *Document(s)* choisissez votre fichier **.csv** que vous voulez indexer (ici data.csv).

Laissez les champs *Extracting Req. Handler Params, Commit Within* et *Overwrite* comme ils sont. Enfin cliquez sur *Submit Document :*

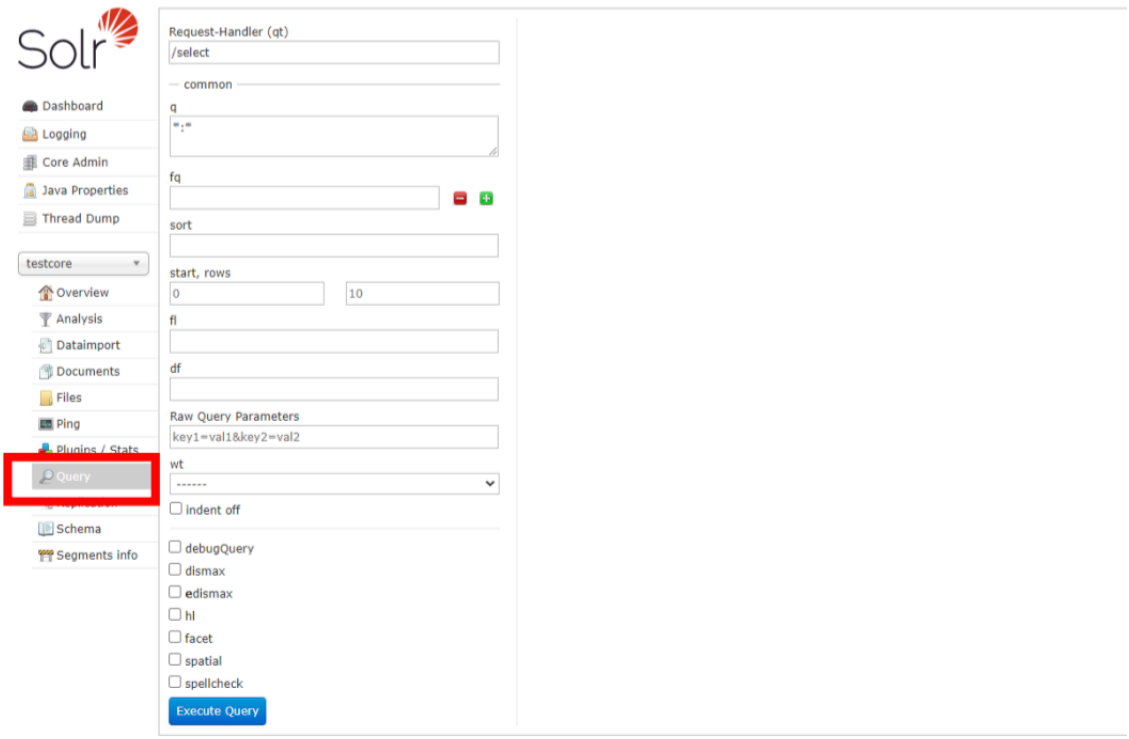


Vous avez maintenant indexé vos données dans Solr. Vous pouvez vérifier cela en allant dans l'onglet *Overview* et regardant *Num Docs* :

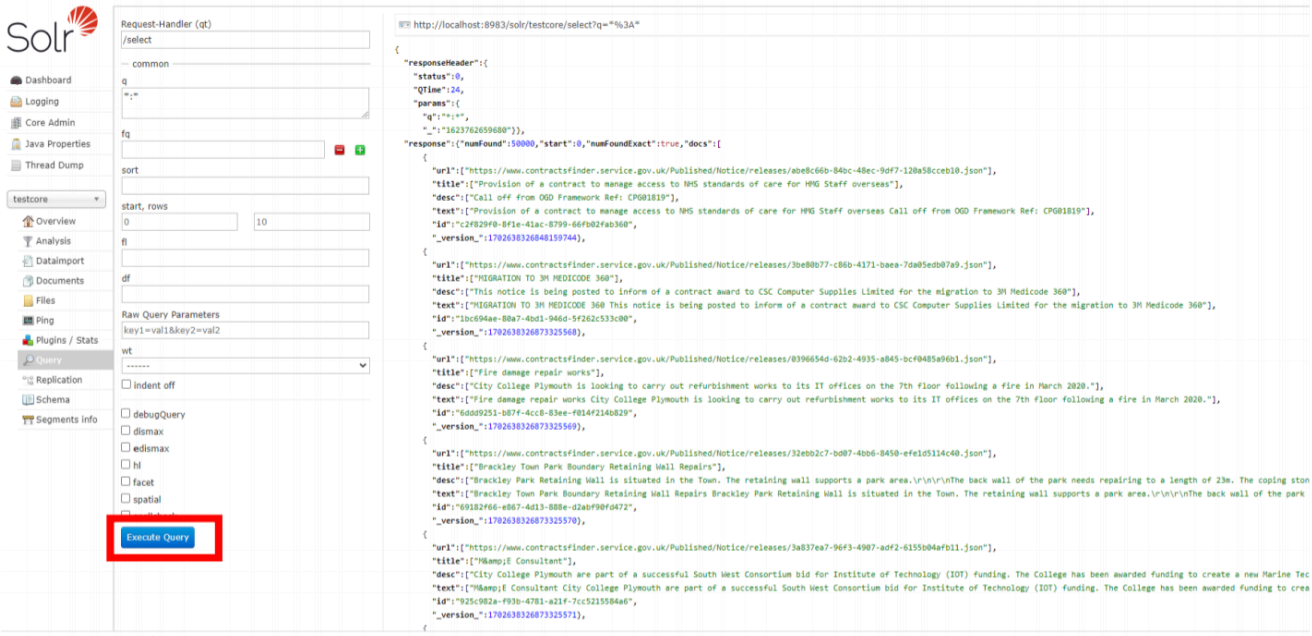


8.1.3 Faire une recherche en Solr

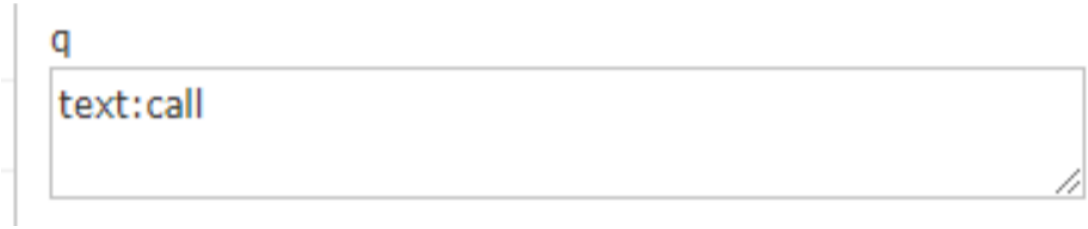
Allez dans l'onglet *Query*.



Laissez *.* dans le champ q et cliquez sur *Executive Query* pour recherche toutes vos données du core et qui seront affiché sur la droite.



Le champs q permet de rechercher des mots dans l'entité que l'on souhaite. Par exemple, si l'on souhaite chercher le mot « call » dans l'entité *text*, il suffit de taper *text :call*.



Il vous sera afficher ensuite tous les documents qui contiennent le mot « call » dans l'entité *text*.

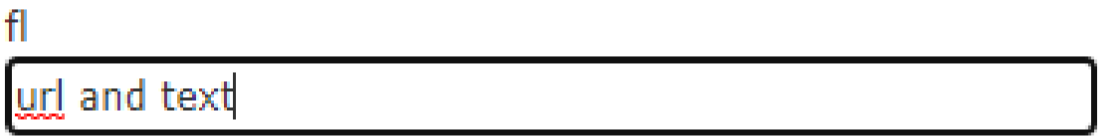

```
"responseHeader":{
  "status":0,
  "QTime":31,
  "params":{
    "q":"text:call",
    "_":"1623762659680"}},
"response":{"numFound":3053,"start":0,"numFoundExact":true,"docs":[
    {
      "url":["https://www.contractsfinder.service.gov.uk/Published/Notice/releases/378f2fd3-8883-417f-a509-7c75360bf145.json"],
      "title":["2020/21 Windows and Doors - Year 3 call off"],
      "desc":["Year 3 call off from Windows and Doors Framework Call Off order for windows and doors."],
      "text":["2020/21 Windows and Doors - Year 3 call off Year 3 call off from Windows and Doors Framework Call Off order for windows and doors."],
      "id":"1fc78f72-b7d2-4c74-be46-ea78d41b0d9f",
      "_version_":1702638325519613963},
    {
      "url":["https://www.contractsfinder.service.gov.uk/Published/Notice/releases/5bb93f00-8630-4388-9406-d37117239dc7.json"],
      "title":["Nurse Call"],
      "desc":["static wireless nurse call system"],
      "text":["Nurse Call static wireless nurse call system"],
      "id":"12008a74-2752-48cf-8185-e9f7e6b1f9e4",
      "_version_":1702638329060655116},
```


Les champs *start*, *row* définisse le nombre de documents qu'il sera affiché pour votre recherche.

Ici Solr affichera les 20 premiers documents qui répondront à votre requête.

Ici Solr affichera du 5ème au 15ème résultat de votre recherche.

Le champs *fl* permet de visualiser seulement certaines entités des documents. Par exemple, si l'on veut seulement visualiser le titre et l'URL des documents, tapez *url and title.*




```
"responseHeader":{
  "status":0,
  "QTime":1,
  "params":{
    "q":"text:call",
    "fl":"url and text",
    "start":"5",
    "rows":"15",
    "_":"1623762659680"}},
"response":{"numFound":3053,"start":5,"numFoundExact":true,"docs":[
    {
      "url":["https://www.contractsfinder.service.gov.uk/Published/Notice/releases/0f837be0-da28-4202-aa7f-83e87343d801.json"],
      "text":["Call Handling Service Bureau Call Handling Service Bureau- Aviation"]},
    {
      "url":["https://www.contractsfinder.service.gov.uk/Published/Notice/releases/cf8f75e9-9a1f-4fb5-a3fd-fc4ffb6e55d6.json"],
      "text":["Call off for testing Call off for testing services"]},
    {
      "url":["https://www.contractsfinder.service.gov.uk/Published/Notice/releases/fa08cb60-852b-4788-82ca-b819f7b3c1b8.json"],
      "text":["OOH Call Handling &amp;amp; Telecare Monitoring Service Futures Housing Group is looking to appoint a provider to manage our Out of Hours Call Handling &amp;amp; Telecare Response Call Mo
    {
      "url":["https://www.contractsfinder.service.gov.uk/Published/Notice/releases/76af9750-bed4-48ad-a67f-049a43456ae0.json"],
      "text":["Out of Hours Call Handling Out of Hours Call Handling"]},
    {
```


8.2 Différentes fonctionnalités extra en Solr

Comme nous l'avons mentionné précédemment, Solr est une plateforme open source. Il est donc possible d'ajouter différentes parties au logiciel interne en fonction des besoins et des exigences de l'entreprise ou du particulier qui l'utilise. C'est pourquoi, pour améliorer l'expérience de recherche dans TEEXMA, nous ajoutons des fonctionnalités telles que celles qui vont être discutées ci-dessous.

8.2.1 Suggester

Suggester dans Solr fournit aux utilisateurs des suggestions automatiques de termes d'interrogation. Cela peut être utilisé comme une puissante fonction de suggestion automatique dans l'application de recherche.
Cette approche utilise l'implémentation du Suggester de Lucene et il est compatible avec toutes les implémentations de recherche disponibles dans Lucene.

Les principales caractéristiques de Suggester sont les suivantes :

- Connectivité de la mise en œuvre de la recherche.
- La possibilité d'insérer des termes dans le dictionnaire, ce qui offre la possibilité de choisir l'implémentation du dictionnaire.
- Support distribué.

Par exemple, si l'utilisateur écrit le mot « énergie » dans la barre de recherche, le suggérant affichera des suggestions telles que :

- Energa gedania Gdansk
- Energies de Dieu
- Secrétaire à l'énergie des États-Unis
- Énergie cinétique

Toutes ces options apparaisent suffisamment rapides pour satisfaire des situations très exigeantes.

8.2.2 SpellCheck

Cette fonction est conçue pour fournir des suggestions de requêtes en ligne basées sur d'autres termes similaires. La base de ces suggestions peut être des termes dans un champ dans Solr, des fichiers texte créés en externe ou des champs dans d'autres index Lucene.

8.2.3 FuzzyQuery

Une autre des fonctions les plus utiles est la recherche floue. La recherche floue est un outil puissant pour trouver des correspondances inexactes dans l'index Solr.

Lorsque nous parlons de correspondance inexacte dans Solr, nous prenons en charge plusieurs types de requêtes, telles que les recherches par caractères génériques, les recherches par plages, les recherches floues et les recherches par proximité. Dans cette étude, nous nous concentrerons sur les deux derniers.

Le mécanisme est basé sur l'édition à distance. Et qu'entendons-nous par-là ?
L'édition de la distance est un moyen de mesurer la similarité de deux chaînes de caractères. Elle est définie comme le nombre minimum d'opérations primitives pour convertir une chaîne de caractères en une autre. Par opération, on entend l'insertion, la suppression, la substitution ou la transposition de deux caractères.

Dans Solr, la distance d'édition ou la similarité est calculée à l'aide d'un algorithme appelé distance de Damerau-Levenshtein. Les opérations primitives comprennent :

Insertion : geek -> greek
Suppression : geek -> gee
Substitution : geek -> geez
Transposition : geek -> geke

Les recherches floues et les recherches de proximité sont basées sur le calcul de la similarité présenté ci-dessus.

9. SOLUTIONS DANS PYTHON

Précédemment, j'ai expliqué et proposé plusieurs hypothèses d'amélioration que je vais maintenant appliquer en python pour voir les résultats.

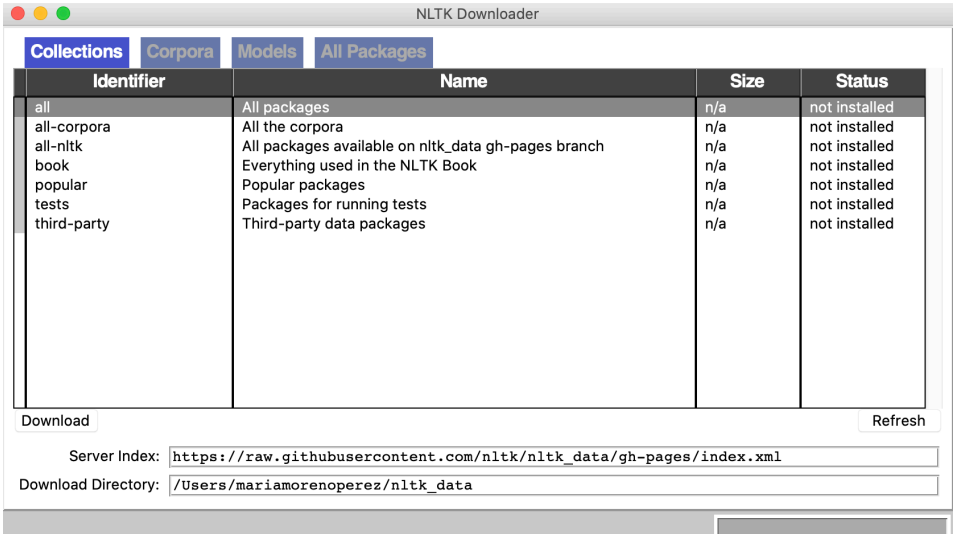
J'ai choisi python comme logiciel de travail parce que c'est l'un des plus utilisés de nos jours, en raison de la facilité d'utilisation de son langage malgré le fait que je ne sois pas un expert en programmation et parce qu'il peut également être mis en œuvre en conjonction avec le moteur de recherche utilisé par BASSETTI, Solr.

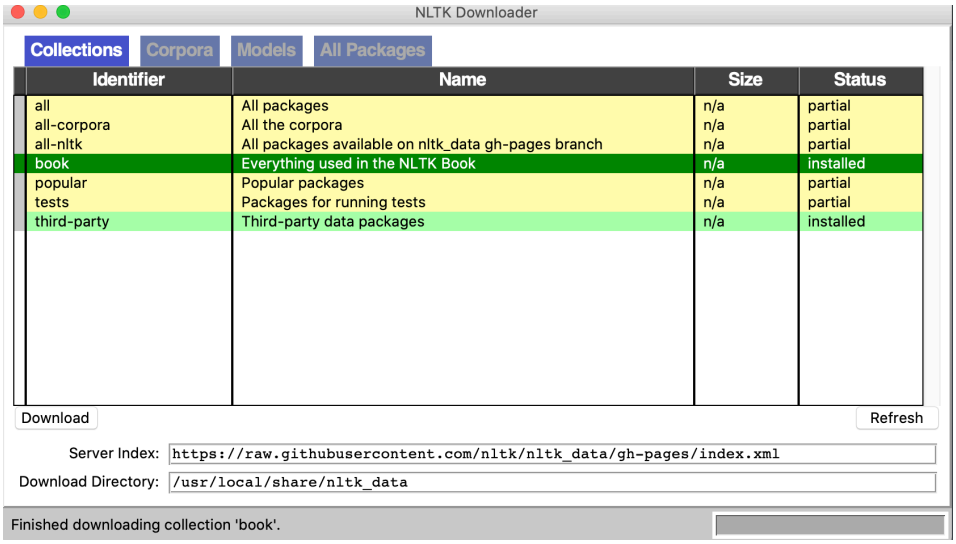
9.1 Code 1 : Application du Word2Vec

Le premier programme recherche la similarité des mots en utilisant Word2Vec. Nous importons les différents modules python que nous allons utiliser pour tokeniser le texte que nous allons utiliser.

Pour cela, nous avons besoin de la plateforme nltk « Natural language toolkit » qui est utilisée pour travailler avec des données de langage humain en Python. Pour l'installation de toutes les fonctions qui vont être nécessaires, nous devons avoir installé pip3 dans notre ordinateur avec la fenêtre de commande avant.

Un fois nous avons nltk et ses différentes fonctionnalités installées avec pip3 :



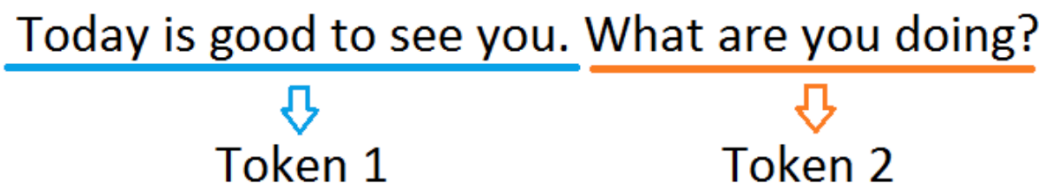


Nous pouvons importer nltk dans le code python.

```
# importing all necessary modules
from nltk.tokenize import sent_tokenize, word_tokenize
import warnings

warnings.filterwarnings(action='ignore')
```

Conceptuellement, tokeniser consiste à diviser des jetons ou des chaînes de caractères en parties plus petites telles que des phrases, des mots ou des symboles.

Par exemple, si nous appliquons la tokenisation à un paragraphe, le résultat sera une liste de tokens qui sera composée de phrases, si nous faisons la même chose mais pour une phrase, nous obtiendrons une liste de tokens composée des mots qui composent cette phrase. Avec la phrase suivante, nous pouvons voir que lorsque nous effectuons la tokenisation, nous obtenons deux jetons, le bleu et l'orange. Ceux-ci peuvent à leur tour être séparés en mots, mais pour notre exemple, nous utiliserons d'abord la séparation par phrases.



En Python, le résultat sera un vecteur avec des jetons pour chaque phrase.

Nous lisons un fichier qui, dans ce cas, est .txt et remplaçons les sauts de ligne par des espaces, puis nous tokenisons le texte en phrases et ensuite en mots.

```
#  Reads 'alice.txt' file
sample = open("/Users/mariamorenoperez/Desktop/alice.txt", "r")
s = sample.read()

# Replaces escape character with space
f = s.replace("\n", " ")

data = []

# iterate through each sentence in the file
for i in sent_tokenize(f):
    temp = []
```

```
    # tokenize the sentence into words
    for j in word_tokenize(i):
        temp.append(j.lower())

    data.append(temp)
```

Pour mettre en œuvre word2vec, nous utilisons la bibliothèque gensim, qui est axée sur la modélisation de sujets, l'indexation et la recherche de similarités à partir de grands corpus.

```
import gensim
from gensim.models import Word2Vec
```

Il existe deux mécanismes pour utiliser word2vec, CBOW et Skip gram. J'ai mis en œuvre les deux pour voir les différents résultats. Il s'agit d'un encastrement qui est la représentation numérique des mots.

Dans le modèle CBOW, les représentations distribuées du contexte sont combinées pour prédire le mot central. Alors que dans le modèle Skip-gram, la représentation distribuée du mot d'entrée est utilisée pour prédire le contexte.

```
# Create CBOW model
model1 = gensim.models.Word2Vec(data, min_count=1,
                                vector_size=100, window=5)
```

```
# Create Skip Gram model
model2 = gensim.models.Word2Vec(data, min_count=1, vector_size=100,
                                window=5, sg=1)
```

Nous sélectionnons deux mots dans l'ensemble des mots et le programme nous montrera avec le cosinus de Word2Vec la similarité entre eux.

```
# Print results
print("Cosine similarity between 'alice' " +
      "and 'wonderland' - CBOW : ",
      model1.wv.similarity('alice', 'wonderland'))

print("Cosine similarity between 'alice' " +
      "and 'machines' - CBOW : ",
      model1.wv.similarity('alice', 'machines'))
```

```
# Print results
print("Cosine similarity between 'alice' " +
      "and 'wonderland' - Skip Gram : ",
      model2.wv.similarity('alice', 'wonderland'))

print("Cosine similarity between 'alice' " +
      "and 'machines' - Skip Gram : ", model2.wv.similarity('alice', 'machines'))
```

Plus le cosinus est élevé, plus la distance entre les vecteurs est grande et donc plus les mots sont similaires :

```
Cosine similarity between 'alice' and 'wonderland' - CBOW :  0.983732
Cosine similarity between 'alice' and 'machines' - CBOW :  0.8752993
```

9.2 Code 2 : Word2Vec 2.0

Nous utiliserons la bibliothèque bs4 (Beautifulsoup) qui facilite l'extraction d'informations des pages web.
Nous chargeons un lien Wikipedia sur l'intelligence artificielle et le lisons. Pour lire des documents directement d'internet nous avons besoin aussi de la fonction urllib.request.

```
import bs4 as bs
import urllib.request
import re
import nltk

# CREATING CORPUS

# Downloading Wikipedia article
scrapped_data = urllib.request.urlopen(
    'https://en.wikipedia.org/wiki/Artificial_intelligence')
article = scrapped_data .read()

# Reading the content and we analize it using BeautifulSoup
parsed_article = bs.BeautifulSoup(article, 'lxml')
```

De la même manière que précédemment, nous assemblons les paragraphes, puis nous tokenisons le texte et convertissons également tous les caractères en minuscules. Nous procédons à la tokénisation en phrases, puis en mots. Nous supprimons les mots d'arrêt (il s'agit de mots couramment utilisés, comme "le", "a", "a", "dans") qu'un moteur de recherche a été programmé pour ignorer, à la fois lors de l'indexation des entrées pour la recherche et lors de leur récupération à la suite d'une requête de recherche.

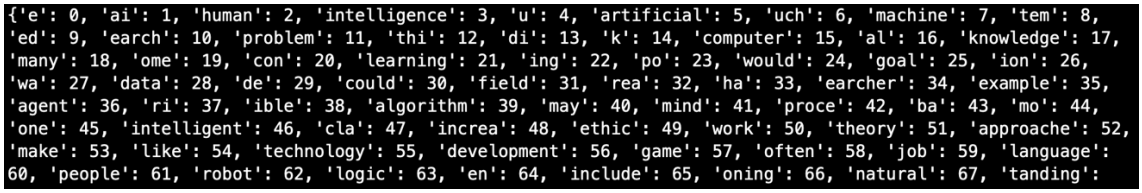
```
# Searching the entire content of the article's paragraph tags
paragraphs = parsed_article.find_all('p')

# Joining paragraphs and store article in the following variable
article_text = ""

for p in paragraphs:
    article_text += p.text


# PREPROCESSING

# Cleaning the text
processed_article = article_text.lower()
processed_article = re.sub('[^a-zA-Z]', ' ', processed_article)
processed_article = re.sub(r's+', ' ', processed_article)

# Preparing the dataset
all_sentences = nltk.sent_tokenize(processed_article)

all_words = [nltk.word_tokenize(sent) for sent in all_sentences]
```

```
# Removing Stop Words
from nltk.corpus import stopwords
for i in range(len(all_words)):
    all_words[i] = [w for w in all_words[i]
                    if w not in stopwords.words('english')]
```

Nous utilisons à nouveau gensim pour implémenter la fonction word2vec. Nous avons inclus plusieurs fonctionnalités dans ce code.

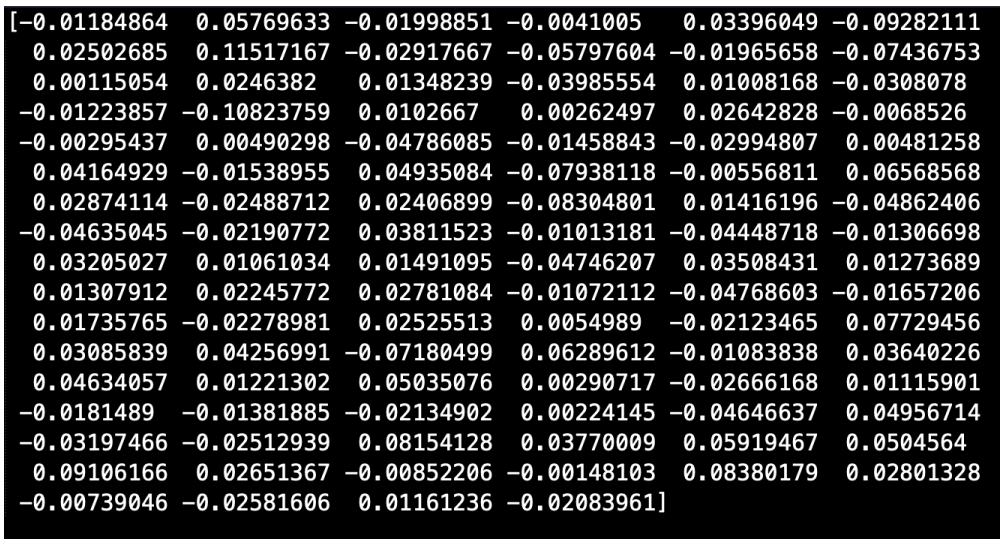
```
# CREATION WORD2VEC MODEL

from gensim.models import Word2Vec

# Specification that only words that appear 2 times in the corpus are included
word2vec = Word2Vec(all_words, min_count=10)

# Seeing the unique words that appear 2 times
vocabulary = word2vec.wv.key_to_index
print(vocabulary)

# FINDING VECTORS FOR ONE WORD

v1 = word2vec.wv['artificial']
print(v1)

# FINDING SIMILAR WORDS

sim_words = word2vec.wv.most_similar('intelligence')
print(sim_words)
```

1- On retrouve les mots qui ont été répétés n fois dans le texte. Nous pouvons varier n en fonction de nos intérêts. Et avec print, le programme nous montre les mots qui ont été répétés ce nombre n de fois (10 fois dans ce cas) :



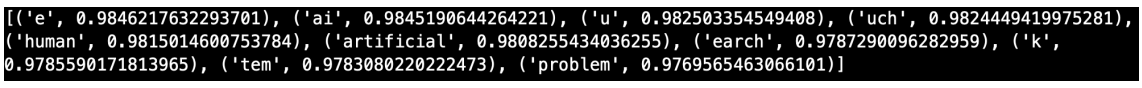
```
{'e': 0, 'ai': 1, 'human': 2, 'intelligence': 3, 'u': 4, 'artificial': 5, 'uch': 6, 'machine': 7, 'tem': 8,
'ed': 9, 'earch': 10, 'problem': 11, 'thi': 12, 'di': 13, 'k': 14, 'computer': 15, 'al': 16, 'knowledge': 17,
'many': 18, 'ome': 19, 'con': 20, 'learning': 21, 'ing': 22, 'po': 23, 'would': 24, 'goal': 25, 'ion': 26,
'wa': 27, 'data': 28, 'de': 29, 'could': 30, 'field': 31, 'rea': 32, 'ha': 33, 'earcher': 34, 'example': 35,
'agent': 36, 'ri': 37, 'ible': 38, 'algorithm': 39, 'may': 40, 'mind': 41, 'proce': 42, 'ba': 43, 'mo': 44,
'one': 45, 'intelligent': 46, 'cla': 47, 'increa': 48, 'ethic': 49, 'work': 50, 'theory': 51, 'approache': 52,
'make': 53, 'like': 54, 'technology': 55, 'development': 56, 'game': 57, 'often': 58, 'job': 59, 'language':
60, 'people': 61, 'robot': 62, 'logic': 63, 'en': 64, 'include': 65, 'oning': 66, 'natural': 67, 'tanding':
```

2- La deuxième fonctionnalité consiste à demander au programme de nous montrer le vecteur qu'il a attribué au mot que nous voulons connaître (‘artificial’ dans notre cas). Son utilité est simplement de comprendre le fonctionnement de word2vec :



```
[-0.01184864  0.05769633 -0.01998851 -0.0041005   0.03396049 -0.09282111
  0.02502685  0.11517167 -0.02917667 -0.05797604 -0.01965658 -0.07436753
  0.00115054  0.0246382   0.01348239 -0.03985554  0.01008168 -0.0308078
 -0.01223857 -0.10823759  0.0102667   0.00262497  0.02642828 -0.0068526
 -0.00295437  0.00490298 -0.04786085 -0.01458843 -0.02994807  0.00481258
  0.04164929 -0.01538955  0.04935084 -0.07938118 -0.00556811  0.06568568
  0.02874114 -0.02488712  0.02406899 -0.08304801  0.01416196 -0.04862406
 -0.04635045 -0.02190772  0.03811523 -0.01013181 -0.04448718 -0.01306698
  0.03205027  0.01061034  0.01491095 -0.04746207  0.03508431  0.01273689
  0.01307912  0.02245772  0.02781084 -0.01072112 -0.04768603 -0.01657206
  0.01735765 -0.02278981  0.02525513  0.0054989  -0.02123465  0.07729456
  0.03085839  0.04256991 -0.07180499  0.06289612 -0.01083838  0.03640226
  0.04634057  0.01221302  0.05035076  0.00290717 -0.02666168  0.01115901
 -0.0181489  -0.01381885 -0.02134902  0.00224145 -0.04646637  0.04956714
 -0.03197466 -0.02512939  0.08154128  0.03770009  0.05919467  0.0504564
  0.09106166  0.02651367 -0.00852206 -0.00148103  0.08380179  0.02801328
 -0.00739046 -0.02581606  0.01161236 -0.02083961]
```

3- Enfin, nous sélectionnons un mot à volonté et le programme nous montre les mots présents dans le texte, en utilisant le contexte, qu'il considère comme similaires au mot recherché (‘intelligence’ dans notre cas) :



```
[('e', 0.9846217632293701), ('ai', 0.9845190644264221), ('u', 0.982503354549408), ('uch', 0.9824449419975281),
('human', 0.9815014600753784), ('artificial', 0.9808255434036255), ('earch', 0.9787290096282959), ('k',
0.9785590171813965), ('tem', 0.9783080220222473), ('problem', 0.9769565463066101)]
```

9.3 Code 3 : Tokenisation des fichiers PDF

L'entreprise BASSETTI m'avait commenté qu'ils avaient des problèmes pour traiter les documents .pdf parce que le logiciel n'était pas capable d'extraire les informations de ce type de ficher. C'est pourquoi j'ai trouvé un moyen avec Python de travailler avec ce genre de fichiers, bien que ce soit plus fastidieux à lire car nous avons besoin de fonctions spécifiques dans python mais c'est facile avec l'utilisation de celles-ci.

Nous devons importer en python la fonction PyPDF2 qui est en charge de traiter des documents et qui a sa propre méthode d'utilisation :

```
import nltk
import PyPDF2
import textract


# CREATING CORPUS

# Reading document
import PyPDF2
pdf_file = open('alice.pdf', 'rb')
read_pdf = PyPDF2.PdfFileReader(pdf_file)

num_pages = read_pdf.numPages
count = 0
text = ""

while count < num_pages:
    pageObj = read_pdf.getPage(count)
    count += 1
    text += pageObj.extractText()

if text != "":
    text = text

else:
    text = textract.process(
        "/Users/mariamorenoperez/Desktop/alice.txt", method='tesseract',

print (text)
```

Ce code va nous donner le texte extrait du pdf et après nous pouvons le tokeniser pour obtenir chaque mot :

```
# PREPROCESSING

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

tokens = word_tokenize(text)
punctuations = ['(', ')', ';', ':', '[', ']', ',']
stop_words = stopwords.words('english')

keywords = [
    word for word in tokens if not word in stop_words and not word in punctuations]

print(keywords)
```

Le résultat est le suivant :

```
['The', 'Project', 'Gutenberg', 'eBook', 'AliceÕs', 'Adventures', 'Wonderland',
'Lewis', 'CarrollThis', 'eBook', 'use', 'anyone', 'anywhere', 'United', 'States',
'andmost', 'parts', 'world', 'cost', 'almost', 'restrictionswhatsoever', '.', 'You',
'may', 'copy', 'give', 'away', 're-use', 'termsof', 'Project', 'Gutenberg', 'License',
'included', 'eBook', 'online', 'atwww.gutenberg.org', '.', 'If', 'located', 'United',
'States', 'youwill', 'check', 'laws', 'country', 'located', 'beforeusing',
'eBook.Title', 'AliceÕs', 'Adventures', 'WonderlandAuthor', 'Lewis', 'CarrollRelease',
'Date', 'January', '1991', 'eBook', '#', '11', 'Most', 'recently', 'updated',
'October', '12', '2020', 'Language', 'EnglishCharacter', 'set', 'encoding',
'UTF-8Produced', 'Arthur', 'DiBianca', 'David', 'Widger', '*', '*', '*', 'START',
'OF', 'THE', 'PROJECT', 'GUTENBERG', 'EBOOK', 'ALICEÕS', 'ADVENTURES', 'IN',
'WONDERLAND', '*', '*', '*', 'Illustration', 'AliceÕs', 'Adventures', 'Wonderlandby',
'Lewis', 'CarrollTHE', 'MILLENNIUM', 'FULCRUM', 'EDITION', '3.0Contents', 'CHAPTER',
'I', '.', 'Down', 'Rabbit-Hole', 'CHAPTER', 'II', '.', 'The', 'Pool', 'Tears',
'CHAPTER', 'III', '.', 'A', 'Caucus-Race', 'Long', 'Tale', 'CHAPTER', 'IV', '.',
'The', 'Rabbit', 'Sends', 'Little', 'Bill', 'CHAPTER', 'V.', 'Advice', 'Caterpillar',
'CHAPTER', 'VI', '.', 'Pig', 'Pepper', 'CHAPTER', 'VII', '.', 'A', 'Mad', 'Tea-Party',
```

9.4 Code 4 : Moteur de recherche BM25

Pour l'implémenter c'est nécessaire de charger la bibliothèque pandas qui est une bibliothèque logicielle pour la manipulation et l'analyse de données pour le langage de programmation Python. En particulier, il fournit des structures de données et des opérations permettant de manipuler des tableaux numériques et des séries chronologiques.
Nous utilisons pandas set_option(), pour modifier le nombre de lignes et de colonnes en fonction de ce que nous recherchons. Dans ce cas, nous utilisons max.colwidth pour initialiser la largeur maximale de la colonne.

```
import pandas as pd
import spacy
import time


pd.set_option('display.max_colwidth', 0)

df = pd.read_csv('/Users/mariamorenoperez/Desktop/data.csv')
```

La première étape consiste à extraire tous les mots de la colonne "texte" de cet ensemble de données pour créer une "liste de listes" composée de chaque document et des mots qu'il contient. C'est-à-dire que nous utilisons à nouveau la tokenisation et la gérons avec l'excellente bibliothèque spaCy.

```
# PREPROCESS AND TOKENIZE

from tqdm import tqdm

nlp = spacy.load("en_core_web_sm")
text_list = df.text.str.lower().values
tok_text = []  # for our tokenized corpus
```

Nous utilisons core pour un pipeline à usage général avec vocabulaire, syntaxe, entités et vecteurs de mots. Web est le type de texte et sm indique la taille du paquet.

Nous convertissons ensuite toutes les valeurs en minuscules et nous tokenisons le texte.

```
# Tokenising using SpaCy:

for doc in tqdm(nlp.pipe(text_list, disable=["tagger", "parser", "ner", "lemmatizer"])):
    tok = [t.text for t in doc if t.is_alpha]
    tok_text.append(tok)
```

Dans ce cas, nous utilisons tqdm qui signifie "progrès" afin que notre boucle affiche instantanément une mesure de la progression. Il suffit d'envelopper notre itération avec cette fonction pour que cela se produise. Comme nous testons initialement ce code avec un fichier .csv contenant 50 000 documents, il est très utile car le processus prendra du temps et il est intéressant de voir comment il progresse.

Lorsque nous appelons nlp sur le texte, spaCy commence par tokeniser le texte pour produire un Doc. Le Doc est ensuite traité en plusieurs étapes différentes.

La chaîne de traitement comprend généralement un étiqueteur, un lemmatiseur, un analyseur syntaxique et un reconnaisseur d'entités. Chaque composant de la chaîne de traitement renvoie le document traité.
Dans ce code, nous utilisons nlp.pipe pour traiter un itérable (potentiellement très grand), donc nous désactivons certains des composants de traitement du sujet, qui seront chargés mais non exécutés.

Pour la recherche, nous avons utilisé l'algorithme BM25 (best match 25), qui utilise le mécanisme TF-IDF mettant en œuvre deux fonctions, la fréquence du terme dans le token et la longueur du texte. Il est facile de l'utiliser en python grâce à la bibliothèque rank-bm25 importé dans le code.

```
# SIMPLE BM25 SEARCH

from rank_bm25 import BM25Okapi
bm25 = BM25Okapi(tok_text)

query = "flood defence"
tokenized_query = query.lower().split(" ")

t0 = time.time()
results = bm25.get_top_n(tokenized_query, df.text.values, n=3)
t1 = time.time()
print(f'Searched 50,000 records in {round(t1-t0,3) } seconds \n')

for i in results:
    print (i)
```

Nous écrivons la phrase ou les mots que nous voulons rechercher (‘flood defence’ dans ce cas), nous tokenisons la requête que nous voulons et le code retournera le texte le plus approprié et le plus pertinent parmi tous les documents du fichier.

J’ai utilisé un fichier avec 50000 documents différents et le résultat de ma recherche c’est :

```
Forge Island Flood Defence and Public Realm Works Award of Flood defence and public
realm works along the canal embankment at Forge Island, Market Street, Rotherham as
part of the Rotherham Renaissance Flood Alleviation Scheme.
Flood defence maintenance works for Lewisham and Southwark College **AWARD**
Following RfQ NCG contracted with T Gunning for Flood defence maintenance works for
Lewisham and Southwark College
Freckleton St Byrom Street River Walls Freckleton St Byrom Street River Walls,
Strengthening of existing river wall parapets to provide flood defence measures
```

J'ai également ajouté la fonction temps pour savoir combien de temps il faut pour le trouver. En exécutant le code, il nous montre les itérations qu'il a effectuées et le temps de chacune d'entre elles.

```
0it [00:00, ?it/s]
1it [00:01,  1.23s/it]
132it [00:01, 137.15it/s]
257it [00:02, 123.24it/s]
378it [00:02, 204.72it/s]
505it [00:02, 307.55it/s]
600it [00:03, 212.55it/s]
743it [00:03, 318.73it/s]
835it [00:04, 198.12it/s]
971it [00:04, 285.35it/s]
1058it [00:05, 176.82it/s]
1206it [00:05, 263.80it/s]
1295it [00:06, 193.39it/s]
1436it [00:06, 278.75it/s]
```

[…]

Finalement :

```
50000it [03:22, 247.07it/s]
Searched 50,000 records in 0.07 seconds
```

10. POINT DE DEPART ET EVOLUTION

Lors de la première réunion avec la société BASSETTI, on m'a fait part du problème que certains de leurs clients, ceux qui disposent d'importantes bases de données, rencontraient pour trouver les documents qu'ils souhaitaient. Ils m'ont présenté leur logiciel, leurs programmeurs, l'organisation de la société et ses objectifs.

Au départ, il n'était pas clair pour eux s'il était possible d'améliorer la recherche dans leur logiciel, ils m'ont donc proposé comme objectif principal de découvrir s'il était possible ou non d'optimiser leur moteur de recherche. En plus de le compléter avec mes propres études et les moyens de le faire, mais l'idée était d'étudier s'il était vraiment possible de le faire et avec le moins d'heures de travail possible, puisqu'ils ont un grand nombre de programmeurs mais avec d'autres fonctions dans l'entreprise.

Petit à petit, j'ai rassemblé des informations sur le fonctionnement des moteurs de recherche, leurs types, les améliorations possibles en open source et j'ai pu constater qu'il existait différentes solutions.

J'ai commencé par m'informer sur le sujet car c'est un domaine dans lequel je n'étais jamais entré et une fois que j'ai compris comment cela fonctionnait, j'ai commencé à chercher des solutions possibles sans me limiter à un seul type de programme. C'est pourquoi j'ai cherché des alternatives dans Elasticsearch, Solr, Python...

Enfin, je me suis concentré sur Solr et Python.

11. PROBLEMES RENCONTRES

Au début, j'avais des doutes quant à ma capacité à fournir une bonne solution au problème, car je n'avais jamais essayé ou ne connaissais rien aux moteurs de recherche. De plus, je n'ai pas de notions très profondes en programmation, ce qui m'a fait passer beaucoup de temps à comprendre Python et son fonctionnement.

L'un des principaux problèmes était de travailler avec Solr, une plateforme dotée de nombreuses fonctionnalités et difficile à utiliser. J'ai eu des difficultés à travailler avec toutes sortes de documents, car au départ il ne me permettait pas de charger des fichiers PDF, et à traiter des bases de données très volumineuses.

12. CONCLUSION

Pour conclure, je peux dire que les résultats ont été très bons. L'amélioration des moteurs de recherche est un domaine qui sera toujours en développement constant, ce qui m'a motivé à continuer. Les résultats finaux ont été très bons et j'ai réussi à me former et à approfondir mes connaissances dans ce sujet intéressant. Les solutions parviennent à donner des résultats tout à fait pertinents et vous pouvez également appliquer différentes fonctionnalités supplémentaires qui amélioreront la recherche de l'utilisateur.

En ce qui concerne BASSETTI, ils sont très satisfaites avec les résultats, et ils vont les appliquer de la meilleure façon possible. Ce fut un plaisir de travailler avec eux à tout moment et je voudrais remercier toute l’équipe de m’avoir accueilli et traité tellement bien.

13. BIBLIOGRAPHIE

(2020, 17 janvier) Que sont les moteurs de recherche ? Elasticsearch vs Solr. 100% Entrepreneur.
https://100emprendedor.com/que-son-los-motores-de-busqueda-elasticsearch-vs-solr/

Archanco, R. (2014, 3 mars). Nous avons parlé à plusieurs reprises dans ce blog des recherches et des moteurs de recherche sur Internet. Aujourd'hui, je voudrais donc vous parler du fonctionnement d'un moteur de recherche. Il ne s'agit pas de Lire la suite. Papiers d'intelligence compétitive.
https://papelesdeinteligencia.com/como-funciona-un-motor-de-busqueda/

M. (2021, 30 junio). *How Search Engines Work: Crawling, Indexing, and Ranking | Beginner's Guide to SEO*. Moz.
https://moz.com/beginners-guide-to-seo/how-search-engines-operate

Fabre, C. (2015). *Sémantique distributionnelle automatique : la proximité distributionnelle comme mode d'accès au sens*.
https://www.cairn.info/revue-ela-2015-4-page-395.htm

Erickson, E. (2021, 16 marzo). *Improving the Search Experience With Solr Suggester*. Lucidworks.
https://lucidworks.com/post/solr-suggester/

Cómo construir un motor de búsqueda. (2020, 21 diciembre). ICHI.PRO.
https://ichi.pro/es/como-construir-un-motor-de-busqueda-175440727662252

David, J. (2020, 25 agosto). Solr: como funciona el servidor de búsqueda de Apache. Hebergement webs. https://www.hebergementwebs.com/configuracion/solr-como-funciona-el-servidor-de-busqueda-de-apache

de la Vega, R. (2021, 7 febrero). ▷ Implementación de Word2Vec con la biblioteca Gensim en Python. Pharos. https://pharos.sh/implementacion-de-word2vec-con-la-biblioteca-gensim-en-python/

Installing NLTK Data — NLTK 3.6.2 documentation. (2021). NLTK 3.6.2 Documentation.
https://www.nltk.org/data.html

{{metadataController.pageTitle}}. (2021). Packt. https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781783553235/1/ch01lvl1sec08/overview-and-installation-of-solr

spaCy 101: Everything you need to know · spaCy Usage Documentation. (2021). SpaCy 101: Everything You Need to Know. https://spacy.io/usage/spacy-101/

tokenize — Tokenizer for Python source — Python 3.9.6 documentation. (2021). Python. https://docs.python.org/3/library/tokenize.html

Welcome to. (2021). Apache Solr. https://solr.apache.org/

Wikipedia contributors. (2021, 16 junio). Word2vec. Wikipedia. https://en.wikipedia.org/wiki/Word2vec