Rapport final
écrit par :
Axel NGAUV

Projet - Création d'un moteur de recherche

14 novembre 2022

Master 2 de Sciences du Langage
parcours Langue et Informatique
Indexation sémantique et recherche d'informations

Table des matières

Table des figures

1 Introduction

L'objectif de ce projet est de créer un moteur de recherche en utilisant Solr/Lucene. Solr est un moteur de recherche *open-source* (logiciel libre) permettant l'indexation et le requêtage de documents. Il dispose d'une API (interface de programmation d'application) mais il est également possible de l'utiliser avec une interface en ligne de commande [Apache, 2021]. C'est d'ailleurs l'option privilégiée pour ce projet.
Il me semble important de définir ce qu'est un moteur de recherche, puisque c'est le cœur de ce devoir. Un moteur de recherche peut être défini comme une application offrant la possibilité à un utilisateur de faire une recherche (cette recherche pouvant s'effectuer localement ou en ligne). Il peut ainsi trouver des ressources :

- le moteur de recherche est constitué d'un corpus (il s'agit de métadonnées représentées comme des documents) qui est indexé,
- l'utilisateur peut à l'aide d'une requête (question de l'utilisateur envers le moteur de recherche) composée d'un ou plusieurs termes recevoir une liste de documents correspondants aux termes entrés par l'utilisateur (la requête est comparée à chaque document et une liste de résultats triés s'affiche).

Ce rapport présentera dans un premier temps mon corpus, ses caractéristiques ainsi que les raisons pour lesquelles je l'ai choisi. Il décrira dans un second temps les différentes étapes du projet.

2 La sélection du corpus

Mon corpus est constitué d'une liste de séries et de films d'animation asiatiques (principalement en provenance du Japon, de Chine ou de Corée) datant du 5 janvier 2020. Il provient d'un *dataset*, créé par un utilisateur de *kaggle* répondant au pseudonyme de Marlesson [Marlesson, 2020] en faisant du *crawling*. Le *crawl* ne fait ici aucunement référence à un style de nage, mais plutôt à l'action d'utiliser un robot afin d'explorer toutes les pages d'un site web afin d'en extraire un maximum d'informations. Le site qui a subi le *crawl* est MyAnimeList.net, un site de référence sur l'animation asiatique.

Le *dataset* est composé de 3 fichiers, tous au format CSV :
— *animes.csv* que j'ai choisi pour en faire mon corpus,
— *profiles.csv* qui contient des informations à propos des utilisateurs du site MyAnimeList.net (œuvres visualisées, pseudonyme de l'utilisateur, date de naissance, sexe, œuvres favorites),
— *reviews.csv* qui contient des données concernant les critiques qu'ont fait les utilisateurs (la critique et la note attribuée à l'œuvre).

Comme précisé plus tôt, j'ai choisi de m'intéresser au fichier *anime.csv*. Il contient une liste de 16 214 œuvres différentes. Il existe néanmoins des doublons (mais ayant un id différent, j'ai donc choisi de les garder), ce qui nous amène à un total de 19 311 documents.
Dans cette liste est indiqué le titre, mais aussi un résumé (manquant pour 5% des œuvres, car les pages concernant les œuvres ne possèdent pas toujours de résumé) et les genres auxquels appartient chaque œuvre, la période durant laquelle elle a été diffusée, le nombre d'épisodes, le nombre de membres qui l'ont vu, la popularité, ainsi que le score.

J'ai choisi de travailler sur ce corpus car le domaine de l'animation, notamment celui de l'animation asiatique, m'intéresse grandement. Il est important de se lancer dans un projet qui peut soit nous être utile, soit nous intéresser. Mon entourage et moi-même n'avons aucun besoin particulier qui nécessiterait un moteur de recherche, j'ai donc sélectionné mon corpus en fonction de mon intérêt.

Ci-dessous, un extrait de mon corpus représentant une œuvre et ses caractéristiques :

```
uid,title,synopsis,genre,aired,episodes,members,popularity,ranked,score,img_url,link
28891,Haikyuu!! Second Season,"Following their participation at the Inter-High, the Karasuno High School volleyball team attempts to
refocus their efforts, aiming to conquer the Spring tournament instead.

When they receive an invitation from long-standing rival Nekoma High, Karasuno agrees to take part in a large training camp alongside many
notable volleyball teams in Tokyo and even some national level players. By playing with some of the toughest teams in Japan, they hope not
only to sharpen their skills, but also come up with new attacks that would strengthen them. Moreover, Hinata and Kageyama attempt to
devise a more powerful weapon, one that could possibly break the sturdiest of blocks.

Facing what may be their last chance at victory before the senior players graduate, the members of Karasuno's volleyball team must learn
to settle their differences and train harder than ever if they hope to overcome formidable opponents old and new—including their archrival
Aoba Jousai and its world-class setter Tooru Oikawa.

[Written by MAL Rewrite]","['Comedy', 'Sports', 'Drama', 'School', 'Shounen']","Oct 4, 2015 to Mar 27,
2016",25.0,489888,141,25.0,8.82,https://cdn.myanimelist.net/images/anime/9/76662.jpg,https://myanimelist.net/anime/28891/Haikyuu_Second_Sea
son
```

FIGURE 1 – Extrait de mon corpus indiquant les caractéristiques de *Haikyuu !! Second Season*

3 Description des différentes étapes de création du moteur de recherche

3.1 Création du schéma

Lors de la création de la collection, création automatique du schéma d'indexation de Solr. La création de la collection s'est faite en ligne de commande avec : *solr create -c ProjetAxel* ; ProjetAxel étant le nom de ma collection. Ci-dessous, une capture d'écran de mon fichier *managed-schema* après indexation de mon corpus, permettant de voir les noms des différents champs :

```
  </fieldType>
  <field name="_nest_path_" type="_nest_path_"/>
  <field name="_root_" type="string" docValues="false" indexed="true" stored="false"/>
  <field name="_text_" type="text_general" multiValued="true" indexed="true" stored="false"/>
  <field name="_version_" type="plong" indexed="false" stored="false"/>
  <field name="aired" type="text_general"/>
  <field name="episodes" type="pdoubles"/>
  <field name="genre" type="text_general" multiValued="true"/>
  <field name="id" type="string" multiValued="false" indexed="true" required="true" stored="true"/>
  <field name="img_url" type="text_general"/>
  <field name="link" type="text_general"/>
  <field name="members" type="plongs"/>
  <field name="popularity" type="plongs"/>
  <field name="ranked" type="pdoubles"/>
  <field name="score" type="pdoubles"/>
  <field name="synopsis" type="text_general"/>
  <field name="title" type="string"/>
```

FIGURE 2 – Capture d'écran du fichier *managed-schema*

3.2 Annotation du corpus

Cette étape consiste à faire des annotations en GATE afin de prétraiter le corpus. J'ai cependant rencontré un problème lors de cette étape. En effet, le corpus peut se charger sur Gate, mais aucune annotation ne peut être réalisée sur mon corpus, dû à la lenteur de l'API. J'ai choisi de ne pas aller plus loin pour cette étape qui est je pense dispensable, car le cœur du projet reste la création du moteur de recherche.

3.3 Indexation du corpus

L'indexation des documents s'est faite en ligne de commande avec : *java –Dtype=text/csv –Dcommit=yes –Dc=ProjetAxel –jar post.jar animes.csv* ; car mon corpus est en format CSV.

3.4 Création des facettes

Pour cette étape, il faut créer un minimum de 5 facettes (2 facettes de champ, 2 facettes d'intervalle et 1 facette pivot). J'en ai créé 9.

Les facettes de champ que j'ai choisi de créer sont :
- genre (afin de pouvoir cliquer sur un genre en particulier, ce qui donne la possibilité à l'utilisateur de chercher des œuvres d'un genre particulier sans avoir à le taper dans la barre de recherche, ce qui se révèle particulièrement pratique lorsque l'utilisateur n'a pas les genres en tête),
- title, ce qui permet d'avoir les tokens les plus représentés dans les titres des œuvres (utile notamment lorsque l'utilisateur n'a plus aucune information en tête mais souhaite tout de même effectuer une recherche sur une œuvre).

Les facettes d'intervalle que j'ai choisi de créer sont :
- score (moyenne des notes données par les membres ayant vu l'œuvre),
- episodes (nombre d'épisodes),
- popularity (classement décroissant selon le nombre de membres qui ont vu l'œuvre : le numéro 1 est l'œuvre la plus vue),
- ranked (classement décroissant selon le score de l'œuvre : le numéro 1 a donc le meilleur score),
- members (nombre de membres ayant vu l'œuvre).

Les facettes pivot que j'ai choisi de créer sont :

— genre et score, afin de pouvoir trouver directement les œuvres selon leur genre et leur score (les œuvres ayant plusieurs genres se retrouvent donc logiquement dans chacun des genres dont ils appartiennent),
— genre et popularity, dans le but d'offrir la possibilité à l'utilisateur de trouver facilement les œuvres les plus populaires par genre (ici aussi, les œuvres ayant plusieurs genres se retrouvent dans chacun des genres dont ils appartiennent).

Ci-dessous, des captures d'écran illustrant les modifications effectuées dans le fichier *solrconfig.xml* afin de créer les facettes :

```
<codecFactory class="solr.SchemaCodecFactory"/>
<schemaFactory class="ManagedIndexSchemaFactory">
      <bool name="mutable">true</bool>
      <str name="managedSchemaResourceName">managed-schema</str>
</schemaFactory>

<lib dir="${solr.install.dir:../../../..}/contrib/velocity/lib" regex=".*\.jar" />
<lib dir="${solr.install.dir:../../../..}/dist/" regex="solr-velocity-\d.*\.jar" />

<queryResponseWriter name="velocity" class="solr.VelocityResponseWriter" startup="lazy">
    <str name="template.base.dir">${velocity.template.base.dir:}</str>
</queryResponseWriter>
```

FIGURE 3 – Configuration afin d'utiliser *Velocity*

```
<requestHandler name="/browse" class="solr.SearchHandler">
  <lst name="defaults">
    <str name="echoParams">explicit</str>

    <!-- VelocityResponseWriter settings -->
    <str name="wt">velocity</str>
    <str name="v.template">browse</str>
    <str name="v.layout">layout</str>
    <str name="title">Solritas</str>

    <!-- Query settings -->
    <str name="defType">edismax</str>
    <str name="qf">
      <!-- text^0.5 features^1.0 name^1.2 sku^1.5 id^10.0 manu^1.1 cat^1.4
      title^10.0 description^5.0 keywords^5.0 author^2.0 resourcename^1.0 -->
      title^10.0 synopsis^5.0 genre^5.0 aired^2.0 episodes^2.0 members^2.0 popularity^2.0 ranked^2.0 score^2.0
    </str>
    <str name="mm">100%</str>
    <str name="q.alt">*:*</str>
    <str name="rows">10</str>
    <str name="fl">*,score</str>

    <str name="mlt.qf">
      <!-- text^0.5 features^1.0 name^1.2 sku^1.5 id^10.0 manu^1.1 cat^1.4
      title^10.0 description^5.0 keywords^5.0 author^2.0 resourcename^1.0 -->
      title^10.0 synopsis^5.0 genre^5.0 aired^2.0 episodes^2.0 members^2.0 popularity^2.0 ranked^2.0 score^2.0
    </str>
    <!-- <str name="mlt.fl">text,features,name,sku,id,manu,cat,title,description,keywords,author,resourcename</str> -->
    <str name="mlt.fl">id,title,synopsis,genre, aired, episodes, members, popularity, ranked, score</str>
    <int name="mlt.count">3</int>
```

FIGURE 4 – Configuration de *VelocityResponseWriter* et des *Query settings*

```
<str name="facet.mincount">1</str>
<str name="facet.range">score</str>
<int name="f.score.facet.range.start">0</int>
<int name="f.score.facet.range.end">10</int>
<int name="f.score.facet.range.gap">2</int>
<str name="facet.range">popularity</str>
<int name="f.popularity.facet.range.start">0</int>
<int name="f.popularity.facet.range.end">19311</int>
<int name="f.popularity.facet.range.gap">1000</int>
<str name="facet.range">episodes</str>
<int name="f.episodes.facet.range.start">1</int>
<int name="f.episodes.facet.range.end">2000</int>
<int name="f.episodes.facet.range.gap">25</int>
<str name="facet.range">ranked</str>
<int name="f.ranked.facet.range.start">0</int>
<int name="f.ranked.facet.range.end">19311</int>
<int name="f.ranked.facet.range.gap">100</int>
<str name="facet.range">members</str>
<int name="f.members.facet.range.start">0</int>
<int name="f.members.facet.range.end">5000000</int>
<int name="f.members.facet.range.gap">100000</int>
<!-- <str name="facet.range">manufacturedate_dt</str>
<str name="f.manufacturedate_dt.facet.range.start">NOW/YEAR-10YEARS</str>
<str name="f.manufacturedate_dt.facet.range.end">NOW</str>
<str name="f.manufacturedate_dt.facet.range.gap">+1YEAR</str>
<str name="f.manufacturedate_dt.facet.range.other">before</str>
<str name="f.manufacturedate_dt.facet.range.other">after</str> -->
<str name="facet.field">title</str>
<str name="facet.field">genre</str>
<str name="facet.pivot">genre,score</str>
<str name="facet.pivot">genre,popularity</str>
```

FIGURE 5 – Création des différentes facettes

3.5 Modification de *Velocity*

Cette étape consiste dans un premier temps à copier le dossier *Velocity* afin de pouvoir dans un second temps modifier le fichier *richtext.vm* ainsi que le fichier *main.css* dans le but de personnaliser la mise en forme des résultats ainsi que leur mise en évidence (*highlighting*) par rapport aux termes utilisés dans la requête.
Par ailleurs, le fichier *solrconfig.xml* doit lui aussi être modifié, encore une fois, pour paramétrer la mise en évidence.

```
<!-- Highlighting defaults -->
<str name="hl">on</str>
<str name="hl.fl">title synopsis genre aired episodes members popularity ranked score</str>
<str name="hl.preserveMulti">true</str>
<str name="hl.encoder">html</str>
<str name="hl.simple.pre">&lt;span class="titi"&gt;</str>
<str name="hl.simple.post">&lt;/span&gt;</str>
<str name="f.title.hl.fragsize">0</str>
<str name="f.title.hl.alternateField">title</str>
<str name="f.name.hl.fragsize">0</str>
<str name="f.name.hl.alternateField">name</str>
<str name="f.synopsis.hl.snippets">3</str>
<str name="f.synopsis.hl.fragsize">200</str>
<str name="f.synopsis.hl.alternateField">synopsis</str>
<str name="f.synopsis.hl.maxAlternateFieldLength">1000</str>
<str name="facet.field">title</str>
<str name="facet.field">genre</str>
```

FIGURE 6 – Paramétrage de la mise en évidence dans le fichier *solrconfig.xml*

```
.titi {
    color: green;
    font-weight: bold;
    background-color: orange;
    border: 3px black solid;
    font-size: 200%;
}
```

FIGURE 7 – Modification du fichier *main.css*

```
## synopsis
#if($doc.getFieldValue('synopsis'))
  <div>
    synopsis: #field('synopsis')
  </div>
#end

## genre
#if($doc.getFieldValue('genre'))
  <div>
    genre: #field('genre')
  </div>
#end

## aired
#if($doc.getFieldValue('aired'))
  <div>
    aired: #field('aired')
  </div>
#end

## episodes
#if($doc.getFieldValue('episodes'))
  <div>
    episodes: #field('episodes')
  </div>
#end
```

FIGURE 8 – Modification du fichier *richtext.vm*, partie 1

```
## members
#if($doc.getFieldValue('members'))
  <div>
    members: #field('members')
  </div>
#end

## popularity
#if($doc.getFieldValue('popularity'))
  <div>
    popularity: #field('popularity')
  </div>
#end

## ranked
#if($doc.getFieldValue('ranked'))
  <div>
    ranked: #field('ranked')
  </div>
#end

## score
#if($doc.getFieldValue('score'))
  <div>
    score: #field('score')
  </div>
#end
```

FIGURE 9 – Modification du fichier *richtext.vm*, partie 2

4 Conclusion

Enfin, dernière étape du projet : la rédaction du rapport. Il s'agit du rapport ici présent.

Références

[Apache, 2021] Apache (2021). *Apache Solr Reference Guide*. https://solr.apache.org/guide/8_11/about-this-guide.html. Consulté le 24 octobre 2022.

[Marlesson, 2020] Marlesson (2020). Anime Dataset with Reviews - Myanimelist. https://www.kaggle.com/datasets/marlesson/myanimelist-dataset-animes-profiles-reviews/code?select=animes.csv. Consulté le 20 septembre 2022.