Projet Text Mining

Analyse régionale des offres d'emploi

https://github.com/Romain8816/M2_Text_Mining

Fatim-Zahra El Gaouzi, Elisa Frintz, Romain Dudoit
M2 SISE - Université Lumière Lyon 2

7 février 2022

Table des matières

1 Introduction

Dans le cadre de notre cours de **text mining**, nous avons dû créer une application Rshiny dont le but est d'analyser des offres d'emploi venu du web. L'objectif de notre projet est donc de récupérer des offres d'emploi "au format brut", de les stocker dans une base de données et enfin de les analyser pour en tirer des informations pertinentes. Le langage R étant un des plus utilisés dans le domaine de la science des données, il nous est donc important de le maîtriser.
Afin de présenter au mieux notre travail, nous allons tout d'abord présenter le contexte et les objectifs de notre projet. Ensuite, nous expliquerons la conception de notre base de données ainsi que son alimentation à l'aide de l'API Pôle emploi. Enfin, détaillerons l'architecture de notre application shiny tout en présentant les analyses effectuées. Et pour finir, nous conclurons en expliquant quelques axes d'évolution possibles pour la suite de notre application.

2 Contexte et objectifs du projet

Dans le cadre de ce projet, nous avons voulu analyser différents métiers autour de la data qui sont **très souvent confondus dans l'imaginaire collectif**. En effet, le métier de Data scientist a longtemps été le métier de référence dans le domaine de la data. Pourtant, il existe de plus en plus métiers différents qui permettent de mieux spécifier les rôles et besoins de chacun. Parmi ces nombreux noms de métiers, nous avons voulu nous concentrer sur trois d'entre eux : **Data Scientist**, **Data Analyst** et **Data Engineer**.

Pour cela, nous avons utilisé l'**API de pôle emploi** qui regroupe un certain nombre d'offres d'emploi en France. Afin de ne pas perdre les données des offres d'emploi qui pourrait être retirées, nous avons construit une base de données afin de les stocker dans un serveur **MySQL**. Nous pouvons également mettre à jour cette base de données et en ajoutant les nouvelles offres disponibles.

Pour présenter nos résultats, nous avons construit une interface à l'aide de **Rshiny**. Celle-ci se compose de trois onglets ainsi que d'un bouton pour **mettre à jour la base de données**. Elle se base directement sur la base de données qui doit avoir été installée et construite en amont (à l'aide d'un **script.R** fournit). Nous avons ainsi pu analyser les différentes offres d'emploi afin d'en tirer des informations pertinentes quant à leurs **similarités** et leurs **différences**.

3 Librairies utilisées

Les principales librairies que nous avons utilisées sont :

- RMySQL pour la base de données
- shiny, shinydashboard, shinyalert pour l'application RShiny
- tm, tidytext, wordcloud, stringr, dplyr pour le text mining
- FactoMineR, factoextra pour les AFC
- tidyverse, plotly, httr, jsonlite, utf8, etc...

4 Conception de la base de données

4.1 Récupération des données à partir de l'API

Nous avons récupéré les données des offres d'emploi à l'aide de l'API Pôle Emploi. Pour ce faire, il suffit de créer un compte sur le site dédié aux développeurs (`https://pole-emploi.io/login`) afin de créer et configurer une application nous permettant de nous délivrer un **ID CLIENT** et **Secret CLIENT**. Nous utilisons ces deux éléments pour générer un **token** et puis un **Access token**. À l'aide de ce dernier, nous pouvons accéder au catalogue des Api, notamment, les données de l'ensemble des offres d'emploi disponibles sur le site de Pôle emploi. Cette étape est présentée dans notre code par la fonction "get_token".

À l'aide des URL d'interrogation des API proposées par Pôle Emploi, nous avons effectué une **recherche d'offres à partir de mots clés** :
https ://api.emploi-store.fr/partenaire/offresdemploi/v2/offres/search ?mots Clés=

Nous nous sommes intéressés donc qu'aux offres concernant les trois métiers (data scientist, data engineer et data analyst). Les données sont récupérées au format JSON et stocké par la suite dans un dataframe.

Chaque appel à l'API génère une plage de résultat limitée à 150 offres (lignes). Afin de récupérer toutes les offres, nous avons implémenté notre programme en sorte qu'il récupère l'ensemble des offres remontées et existantes sur le site Pôle emploi (jusqu'à l'obtention d'un **code HTTP 204**). Pour ce faire, nous nous sommes basés sur les règles suivantes :
Lorsque la requête s'exécute sans erreur, en cas de succès il y a 3 codes retour possible :

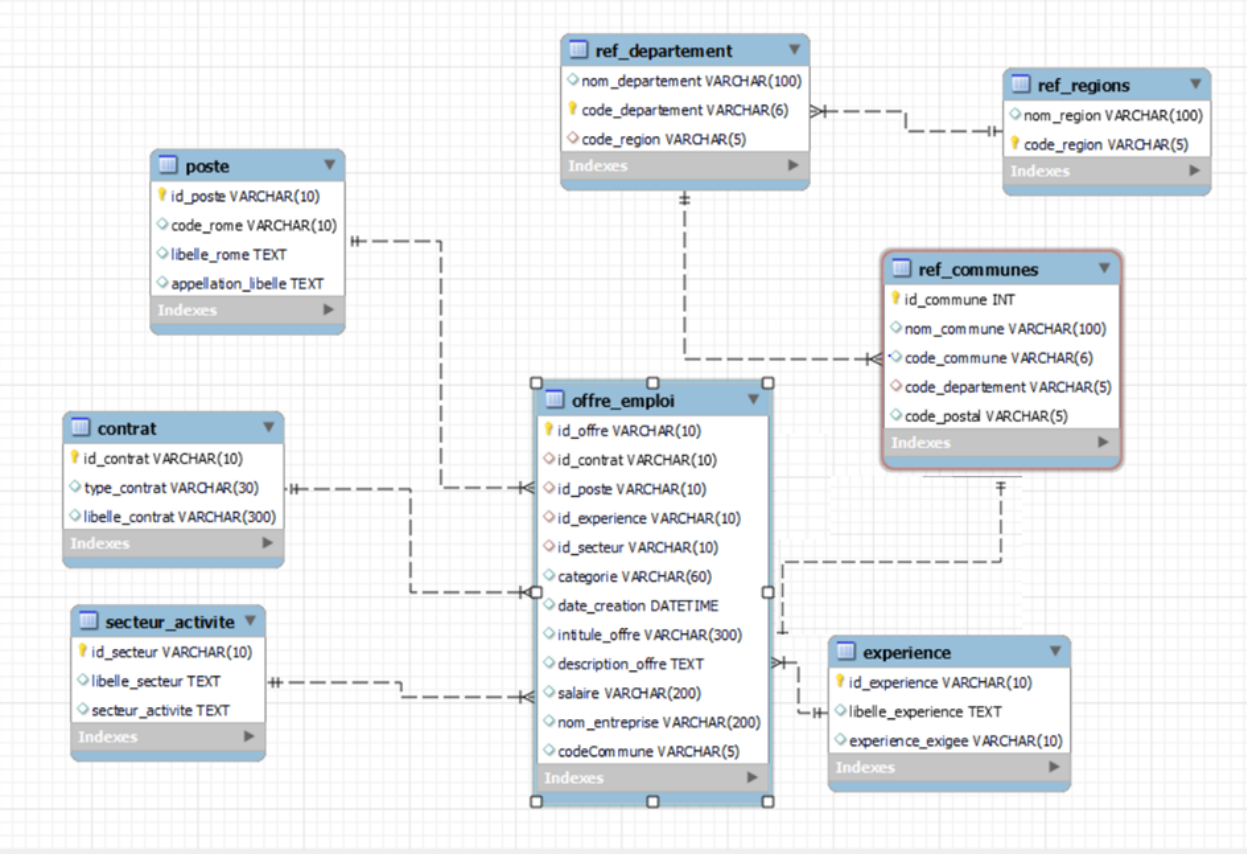
- Si le nombre d'offres issu de la recherche est inférieur au nombre maximal d'offres que le service peut remonter en une requête, un **code HTTP 200 OK** est renvoyé,
- Si le nombre d'offres issu de la recherche est supérieur au nombre maximal d'offres que le service peut remonter en une requête ou supérieur à 150, un **code 206 PARTIAL** est renvoyé,
- Si on a récupéré toutes les offres ou aucune n'existe, on obtient un **code HTTP de 204**.

Nous avons également fait appel aux données des référentiels de l'API Pôle Emploi concernant les régions, les départements et les communes en France afin d'alimenter notre base de données :
https ://api.emploi-store.fr/partenaire/offresdemploi/v2/referentiel/codeReferentiel

4.2 Modélisation de la base de données

Nous avons mis en place notre **modèle relationnel en flocons de neige**. Il est constitué d'une table de fait "Offre_emploi" et cinq dimensions dont une hiérarchisée (modèle ci-dessous).
Nous n'avons gardé que les données jugées pertinentes et nécessaires dans la suite de notre travail. Ces données sont liées principalement au poste, au contrat, à l'expérience, au secteur d'activité et la localisation. La hiérarchie géolocalisation(ref_communes) nous permet de réaliser des analyses par pallier (**drill down**) sur la dimension hiérarchisée si on le souhaite.

Le modèle se présente comme suit :



4.3 Conception de la base de données

Afin de créer notre base de données, nous avons utilisé **MySql** pour sa simplicité et ses performances. Pour faciliter la tâche à l'utilisateur de notre application, nous avons créé une fonction « reset_base_donnes » qui permet de créer la base de données, les tables dans la bases et les relations entre celles-ci. Ses paramètres sont : (user='root', password='root', dbname='textmining', host='127.0.0.1', port=3306).

En utilisant les **API de référentiels** citées précédemment (cf 4.1 Récupération des données à partir de l'AP), nous alimentons les tables (communes, départements et régions). Cette étape qui n'est réalisé qu'une seule fois, prend un peu de temps (environ 3min). Cela est dû au nombre important de communes en France.

Au référentiel des communes, il y a plus de code_département que le référentiel des départements. On a dû supprimer les codes départements en plus, notamment, ceux d'outre-mer. Les fonctions qui gèrent ces trois dimensions sont « mise_a_jour_commune/departement/region ».

Pour les tables des autres dimensions et la table des faits « offre_emploi », l'insertion des données se fait à l'aide de l'ensemble des fonctions « insert_into_nomdetable ». Une vérification est faite au préalable à l'aide des fonctions « get_nomtable » et «id_exists » pour ne pas avoir des redondances dans les tables de dimensions. « insert_data_int_bdd » fait appel à l'ensemble de ces fonctions pour alimenter la base en parcourant ligne par ligne le dataframe déjà stocké en local.

D'autres fonctions supports ont été développées pour gérer les liens avec la base, parmi lesquelles nous retrouvons :

- La fonction na_to_null qui sert à remplacer na et null par le texte « NULL » dans les requêtes
- La fonction connect pour se connecter à la base
- La fonction fermer_connexion pour se déconnecter de la base
- La fonction execute_requete pour exécuter les requêtes en utf-8
- Etc.

Étant donné qu'on a choisi de mettre en place une vraie base de données (sous forme d'entrepôt), nous avons également implémenté une **fonction de mise à jour de la base**. L'idée est de récupérer du site Pole Emploi via l'API, les nouvelles offres avec une date de création dépassant la date de création de la dernière offre de la base.

5 Architecture de l'application Rshiny

Shiny permet de publier sous la forme d'un site web une **application interactive** utilisant du code R. Le site peut fonctionner localement, sur le poste de travail d'un utilisateur qui le lance à partir de RStudio, ou en ligne, sur un serveur dédié.

Une Shiny App se structure en deux parties :

- **La partie serveur** contient l'ensemble du code R qui doit être exécuté par l'appli pour fournir les sorties (graphiques, tables, traitements, etc.) et à les mettre à jour en cas de changement dans les valeurs d'inputs.
- **La partie UI (= User Interface)** contient les instructions de construction/mise en forme de l'interface utilisateur :
  • Les **inputs** sont les composants (widgets) de l'interface graphique qui permettent aux utilisateurs de fournir des valeurs aux paramètres d'entrée.
  • Les **outputs** sont les composants de l'interface graphique qui permettent d'afficher des éléments résultants d'un traitement dans R (graphiques, tables, textes...).

Deux structures de base sont possibles pour les applications shiny : Tout réunir dans un même script (**app.R**), ou bien, séparer la partie UI et la partie server dans deux fichiers (**ui.R** et **server.R**). Dans notre cas, nous avons choisi de différencier les deux parties afin de faciliter la lecture du code. De plus, nous faisons également appel à un fichier **mise_a_jour.R** qui contient toutes les fonctions nécessaire à la mise à jour de la base de données. Cela nous permet d'aérer le code au maximum.

6 Présentation des analyses effectuées

Afin de présenter au mieux notre travail, nous avons réaliser une application shiny avec trois onglets différents. De plus un bouton "mise à jour de la base de données" nous permet de récupérer les dernières offres d'emploi.

6.1 Statistiques descriptives

Dans cette partie, nous avons voulu permettre à l'utilisateur d'obtenir un maximum d'information sur les caractéristiques des offres présentes dans la base de données. Il peut également choisir d'observer uniquement certaines catégories parmi lesquelles Data scientist, Data analyst et Data Engineer. Nous avons ainsi pu analyser ces offres sous différentes dimensions modélisées dans la base de données (expérience, type de contrat, secteur d'activité) sans oublié une cartographie de la répartition des offres en France.
Ces analyses nous ont permis de remarquer que beaucoup des personnes qui publient les offres d'emploi confondent les trois métiers. Nous allons donc essayer de distinguer.

6.2 Analyse des offres

Dans cette partie, nous avons pu mettre en place des techniques de text mining afin de comprendre quels sont les mots les plus importants qui ressortent des différentes descriptions

d'offres. A l'aide de nos connaissances sur ces métiers, nous remarquons que les mots qui ressortent plus sur certains métiers que d'autres sont assez pertinents.
De plus, c'est également ici que nous avons pu approfondir l'analyse des métiers par région à l'aide d'une analyse factorielle des correspondances. Nous avons ainsi pu remarquer que certaines régions sont plus propice à certains métier que d'autres.

6.3 Analyse des compétences

Dans cette partie, nous avons voulu mettre le focus sur certaines compétences clés des métiers analysés (Data scientist, Data analyst et Data Engineer). En effet, nous savons que ces trois métiers sont très proches mais qu'ils restent quand même différents. Nous pouvons ainsi observer la fréquence d'apparition de ces termes pour chaque métier.

Pour aller plus loin et comprendre quelles sont ces différences et similarités, nous avons mis en place une analyse factorielle des correspondances entre les différents métiers et les compétences clés sélectionnées par l'utilisateur. Grâce à cela, nous pouvons observer de manière concrète les liens entre les différents métiers, les compétences qui les réunissent mais également celles qui les diffèrent.

7 Conclusion

Pour conclure, cette application RShiny correspond plutôt bien à ce que nous voulions illustrer de notre projet. En effet, nous avons pu comprendre les différences et les similarités de certaines offres d'emploi très souvent confondues. Ce projet nous a également permis de mettre en relation les différentes compétences acquises dans divers cours de la formation.

En effet, nous sommes parti de la récupération de données sur une API pour tenter de les structurer au sein d'un entrepôt de données. Nous avons ensuite dû utiliser des techniques de textmining afin de présenter une analyse la plus claire possible sous la forme une application interactive RShiny.

8 Difficultés et perspectives d'évolution

Les principales difficultés que nous avons rencontrées dans ce projet ont été liées principalement à l'encodage de l'application et la mise en place de la base de données. Nous avons pu mettre en place un certain nombre de fonctionnalités nécessaires au bon fonctionnement de notre application. Toutefois, des pistes d'améliorations sont envisageables :

- Ajout de plusieurs bouton d'interaction avec la base de données,
- Ajout d'un bouton ok pour la mise à jour de l'application sur la page de l'analyse des compétences,
- Optimisation des temps de calculs qui peuvent parfois être un peu long,
- Déploiement de l'application sur le web à l'aide de shinyapps.io,

— Harmonisation et meilleur utilisation des couleurs pour faciliter la lecture des information, etc...

En plus des connaissances techniques acquises, ce travail nous a permis aussi de collaborer en équipe qui est un aspect très important dans notre formation et future vie professionnelle. De plus nous avons également utiliser GitHub pour partager notre code.

9 Sources

— https://pole-emploi.io/data/api

— https://inter_cati_omics.pages.mia.inra.fr/hackathon_shiny_2020/1_slides/slides.pdf

— https://plotly.com/r/bar-charts/

— https://www.youtube.com/watch?v=FWYZrf2p7MQ&list=PLi0b6yHwHZcEW0q0dSf6xXhE852fbfwQG