Web scrapping et création d'une interface R-Shiny

Analyse globale des offres d'emploi du site Indeed

LE GALÈZE Pierre, MOISSONNIER Rayan, REGAZZETTI Léa
M2 SISE - Université Lumière Lyon 2

7 février 2022

Table des matières

1 Introduction

Dans le cadre de notre cours de Text Mining de deuxième année de Master SISE (Statistique et Informatique pour la Science des Données), nous avons eu comme projet de coder une interface R Shiny permettant une analyse d'offre d'emploi provenant de sites accessibles en ligne. Ainsi, l'objectif de ce projet est d'analyser le corps des annonces, en mettre l'accent sur un type d'emploi ou de compétences. C'est pourquoi nous avons choisi les offres d'emploi militaire en France sur le site Indeed. Voir ce lien : `https://fr.indeed.com/jobs?q=militaire&l=France&vjk=f27176ef24e95146`

Le rendu de notre travail se présente donc sous la forme d'une application R Shiny interactive destinée à l'exploration et l'analyse du corpus. Un des points essentiels de cette application réside dans l'analyse territoriale des offres, avec notamment des représentations cartographiques. Pour réaliser cela, il nous a été demandé d'utiliser les techniques de web scrapping afin d'extraire les offres d'emplois pour alimenter une base de données, qui servira de point d'appui pour l'application R Shiny. Par ailleurs, deux tutoriels vidéos ont également été demandés. Le premier décrivant la procédure d'installation du Système de Gestion de Base de Données (SGBD) que nous avons utilisé, ainsi que l'import des données dans celui-ci. Le deuxième présentant la procédure de lancement de l'application R-Shiny, les différentes fonctionnalités de celle-ci et les analyses que nous pouvons en tirer.

Dans un premier temps, nous allons donc présenter la manière dont nous avons récupéré les données, mais aussi les nettoyages qui ont été faits dessus. Ensuite, nous décrirons le stockage de nos données dans un SGBD libre ainsi que l'interface R-Shiny en lui-même et ses différents onglets. Pour finir, nous analyserons les résultats obtenus.

2 Web scrapping

2.1 Les packages rvest et polite

Afin de récupérer les offres du site, nous avons fait du web scrapping. Pour cela, nous avons utilisé le package rvest, qui permet de parser (autrement dit de parcourir et d'aller chercher) en quelques lignes de code, le contenu d'une page web, pour le rendre exploitable par R. En l'utilisant pour une seule page, tout va bien, mais les problèmes commencent si l'on essaye d'en récupérer plusieurs : rvest va tout simplement ne rien récupérer au bout d'un moment. Ce problème est dû au robot.txt, dont le rôle est d'empêcher les bots d'accéder à certaines ressources présentes sur un site.

De sorte à scrapper de manière plus respectable, nous nous sommes tournés vers le package polite, vivement recommandé dans ce cas-là. Malheureusement, à cause de ce dernier, Indeed nous refuse directement l'accès au site en affichant un message sous R : "no scrapping allowed here" (pas de scrapping autorisé ici).

Après quelques recherches, nous avons finalement trouvé un moyen de contourner le problème de rvest en obligeant notre programme à faire des pauses régulièrement. Même si nous pouvons récupérer une grande partie de ce que nous voulons, cela prend du temps (1-2 minutes par page) et, au bout d'un moment, si l'on essaye de récupérer trop de page, nous nous faisons bloquer. Pour palier à cela, nous faisons tourner le programme sur un nombre de pages où nous sommes sûrs de ne pas nous faire bloquer, puis nous recommençons avec le même nombre de pages suivantes et faisons une jointure de l'ensemble. Plus simplement, au départ on récupère les pages de 1 à 15, puis de 16 à 30, 31 à 45, ...

On aurait aussi pu utilisé l'API proposer par Indeed mais il y avait trop peu d'information dessus sur internet et nous trouvions l'API assez mal documenté.

2.2 Scrapping

Pour rappel, nous avons décidé de récupérer des informations sur les propositions d'emploi militaires en France.

Voir ce lien : `https://fr.indeed.com/jobs?q=militaire&l=France&vjk=f27176ef24e95146`

2.2.1 Récupération avec rvest

Dans un premier temps, nous récupérons le nombre d'emplois de notre recherche, ce qui nous permet de calculer combien de pages il y a au total (Indeed affiche toujours 15 emplois par page).

Ensuite, on fait tourner la boucle principale en fonction du nombre total de pages que l'on veut récupérer. On se retrouve donc dans une boucle qui nous permet de nous balader de page en page, on peut donc y récupérer toutes les informations qui nous intéressent tel que le lieu, le salaire, la date ou encore le nom de la compagnie.

À partir de là, on se rend compte que toutes les informations ne sont pas accessibles sur la page, mais qu'il faut rentrer dans l'offre en elle-même (et donc récupérer le lien correspondant). Une fois cela fait, on peut alors obtenir le contenu, le type et aussi le titre de l'offre.

Pour comprendre de manière plus visuelle, voir ci-dessous :

Algorithm 1 Structure du scrapping du site Indeed

Require: un lien vers le site Indeed avec des critères de recherche
  $n\_emploi \leftarrow$ nombre d'emplois de la recherche
  $n\_page \leftarrow$ ceilling(n_emploi / 15) (arrondit à l'entier le plus grand)
  for i in 1 :n_page **do**
    À chaque itération, on rentre dans une nouvelle page
    $lien \leftarrow$ lien de chaque offre de la page
    $lieu \leftarrow$ tous les lieux de la page
    $date \leftarrow$ toutes les dates de la page
    $company\_name \leftarrow$ tous les noms des compagnies de la page
    $salaire \leftarrow$ tous les salaires de la page
    for i in 1 :length($lien$) **do**
      À chaque itération, on rentre dans un nouveau lien
      $contenu \leftarrow$ contenu de l'offre
      $type \leftarrow$ type de l'offre
      $titre \leftarrow$ titre de l'offre
    end for
  end for
  return plusieurs listes remplies : $lien$, $lieu$, $date$, $company\_name$, $salaire$, $contenu$, $type$, $titre$, $lien$

Pour récapituler, les informations que nous récupérons sur Indeed sont les suivantes :

— les liens des pages (pour pouvoir vérifier à la main)
— les titres des annonces
— les noms des compagnies qui recrutent
— la date de création de l'offre
— le contenu de l'offre
— le type de l'offre (par exemple : CDI)
— le lieu
— le salaire proposé

Chaque information est récupérée dans des listes dont il faut ensuite vérifier la longueur (elles doivent toutes avoir la même).

En faisant cela, on se rend compte que l'on ne récupère pas tous les salaires : certaines personnes ne remplissent pas forcément le salaire et rvest nous récupère juste les cases remplies (il ne prend pas en compte les NA), ce qui nous retourne une liste dont les index ne correspondent pas aux autres. Pour contrer cela, nous avons simplement comparé chaque salaire récupéré aux salaires trouvés dans le contenu et, lorsque qu'il y a correspondance, on associe le salaire à l'index du contenu.

2.2.2 Nettoyage

Maintenant que chaque liste est de même longueur, nous mettons le tout dans un dataframe pour avoir une vision plus globale et pouvoir commencer le nettoyage de donnée.

Le premier nettoyage est celui de la date : on cherche simplement à récupérer tous nombres situés entre "il y a" et "jour" ou "jours". On remarque que, des fois, on récupère "EmployerDernière activité", qui, après une recherche manuelle n'apparaît jamais dans le code des pages. Après plusieurs tests, on se rend compte que, pour aucune raison, rvest fait n'importe quoi :

```
> lin <- "https://fr.indeed.com/jobs?q=militaire&l=France&start=50&vjk=fc88f5e579d796e4"
> date_page <- xml2::read_html(lin) %>%
+   rvest::html_nodes("span")  %>%
+   rvest::html_nodes(xpath = '//*[@class="date"]') %>%
+   rvest::html_text() %>%
+   stringi::stri_trim_both()
> date_page
 [1] "Postedil y a 3 jours"                       "Postedil y a 5 jours"
 [3] "EmployerDernière activité : il y a 13 jours" "Postedil y a 5 jours"
 [5] "EmployerDernière activité : il y a 4 jours"  "Postedil y a 3 jours"
 [7] "Postedil y a 6 jours"                       "EmployerDernière activité : il y a 5 jours"
 [9] "Postedil y a 30+ jours"                     "EmployerDernière activité : il y a 3 jours"
[11] "EmployerDernière activité : il y a 3 jours"  "EmployerDernière activité : il y a 4 jours"
[13] "PostedAujourd'hui"                          "Postedil y a 11 jours"
[15] "Postedil y a 30+ jours"
> lin <- "https://fr.indeed.com/jobs?q=militaire&l=France&start=50&vjk=fc88f5e579d796e4"
> date_page <- xml2::read_html(lin) %>%
+   rvest::html_nodes("span")  %>%
+   rvest::html_nodes(xpath = '//*[@class="visually-hidden"]') %>%
+   rvest::html_text() %>%
+   stringi::stri_trim_both()
> date_page
 [1] ""       "Posted" ""       ""       "Posted" ""       ""       "Posted" ""       ""       "Posted" ""
[14] "Posted" ""       ""       "Posted" ""       ""       "Posted" ""       ""       "Posted" ""       ""
[27] ""       ""       "Posted" ""       ""       "Posted" ""       ""       "Posted" ""       ""       "Poste
[40] ""       "Posted" ""       ""       "Posted" ""
> lin <- "https://fr.indeed.com/jobs?q=militaire&l=France&start=50&vjk=fc88f5e579d796e4"
> date_page <- xml2::read_html(lin) %>%
+   rvest::html_nodes("span")  %>%
+   rvest::html_nodes(xpath = '//*[@class="date"]') %>%
+   rvest::html_text() %>%
+   stringi::stri_trim_both()
> date_page
 [1] "Postedil y a 3 jours"   "Postedil y a 3 jours"   "Postedil y a 5 jours"   "Postedil y a 18 jours"
 [5] "Postedil y a 11 jours"  "Postedil y a 3 jours"   "Postedil y a 6 jours"   "Postedil y a 7 jours"
 [9] "Postedil y a 30+ jours" "Postedil y a 7 jours"   "Postedil y a 27 jours"  "Postedil y a 6 jours"
[13] "PostedAujourd'hui"      "Postedil y a 11 jours"  "Postedil y a 30+ jours"
```

En observant bien le code qui a été lancé, on voit bien que rien n'a été changé entre la première requête et la dernière requête, pourtant, le résultat n'est pas le même. Pour ne pas avoir de soucis, on supprime chaque "EmployerDernière activité".

On transforme par la suite, la variable date en intervalle pour mieux pouvoir l'analyser :

— Dans les 10 derniers jours

— Entre 10 et 20 jours

— Entre 20 et 30 jours

— Plus de 30 jours

Le second concerne le salaire, qui est récupéré sous la forme d'une chaîne de caractère ("Entre ..€ et ..€") dont nous devons récupérer les montants (du premier salaire proposé, mais aussi du deuxième s'il y a). À noter que tous les salaires ne sont pas sur une période mensuelle donc il faut le prendre en compte. Une fois le salaire propre, on calcule la moyenne pour pouvoir mieux comparer.

Ensuite, on s'est intéressé au contenu, dont on a fait un nettoyage classique tel que vu en cours avec, en plus, la gestion des adresses e-mails, des accents (ce que nous n'avions pas dans les textes anglophones) et des espaces supplémentaires (type d'espace différent de ceux habituels).

Pour les types, les choses ont été différentes. Nous ne nous sommes pas appuyés sur ce qui avait été récupéré par rvest, mais directement sur le contenu. Cela n'était pas prévu à la base, mais nous nous sommes rendu compte en récupérant toutes les pages, qu'en fait, il n'y avait aucune structure dans la balise type, et que beaucoup l'indiquait juste dans le contenu. Nous nous sommes donc repliés sur notre solution en cherchant simplement si les mots CDI, CDD ou stage/internship (on récupère parfois des annonces en anglais) apparaissent dans le contenu.

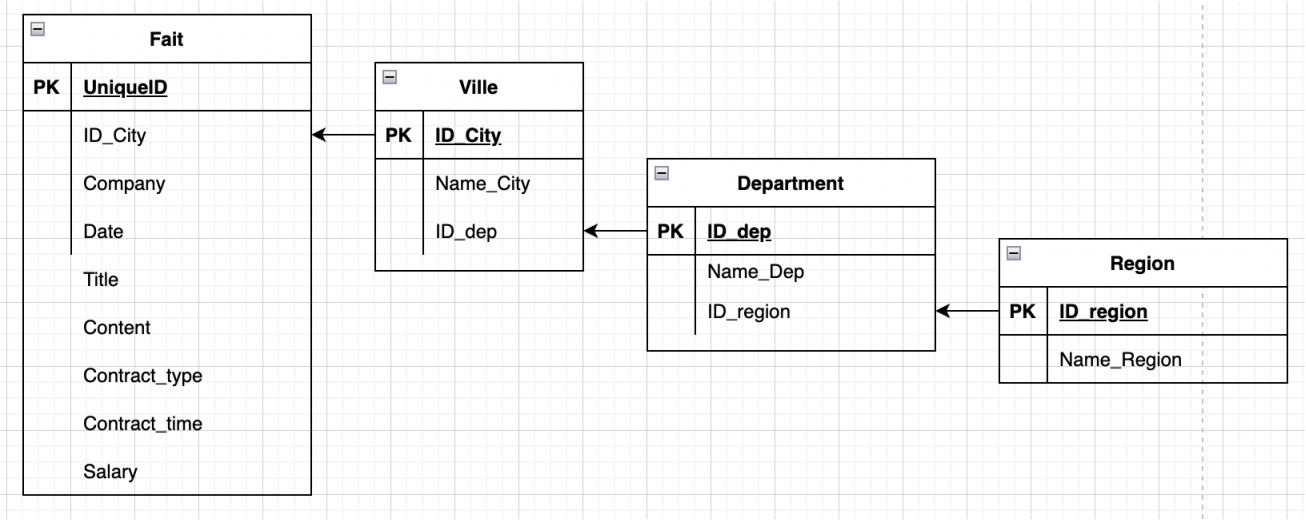
Concernant le lieu, on l'a divisé en deux : la variable département et la variable ville. Le département était assez simple à récupérer, car, la plupart du temps situé entre parenthèse ou bien en début de ligne. Pour la variable ville, il a fallu prendre en compte les potentiels ajouts des quartiers (ex : Paris 20e) ou encore des caractères étranges (ex : •).

Enfin, le titre a été nettoyé de la même manière que le contenu, c'est-à-dire, sans la ponctuation, les nombres, les stops-words, etc.

Une fois le nettoyage de chaque information récupéré terminé, on regroupe tout sous un dataframe et on enlève les potentiels doublons que l'on a pu récupérer par erreur, nous laissant 400 observations pour 8 variables.

3 L'entrepôt de données

Une fois les données pré-nettoyées, il s'agit d'alimenter la base de données. Celle-ci est modélisée de la manière suivante :



La table de fait est composée des différents attributs selon lesquels il est possible d'analyser les différentes mesures. Ainsi, les attributs sont le nom de l'entreprise, la date de parution de l'annonce, le titre, son contenu et le type de contrat. Les mesures sont le salaire et implicitement le nombre d'offres. Puis la dimension localisation est divisée en 3 hiérarchies, partant de la ville, puis du département et enfin la région.

Pour stocker la base de données, nous avons eu recours au SGBD MySQL. La procédure d'installation est décrite dans la vidéo jointe. Ainsi, la première étape consiste à se connecter au SGBD en renseignant l'hôte, le port, le nom d'utilisateur et le mot de passe de connexion. Ensuite, nous vérifions que la base de données que nous allons créer n'existe pas, autrement nous la supprimons, puis nous la créons. Il s'agit alors de se connecter à cette base, et de l'alimenter grâce aux données web scrappées et nettoyées. Il faut d'abord créer la dimension localisation avec ses hiérarchies, en partant de la hiérarchie la plus large (région) pour venir à la hiérarchie la plus fine (ville). Pour cela, nous avons utilisé les données territoriales mises à disposition par le gouvernement français. Il reste alors à créer la table de faits, tout simplement en sélectionnant les données que l'on souhaite garder. Sans oublier de se déconnecter de la base de données une fois cela terminé.

La principale difficulté rencontrée dans cette partie a été la gestion de l'encodage. En effet, les données, le logiciel R et le SGBD MySQL n'utilisant pas le même, nous avons été contraints de faire quelques ajustements avec l'option **encoding**.

4 L'interface R-Shiny

Shiny est un package R permettant de construire des applications interactives depuis R. En effet, elles sont interactives, car nous pouvons ajouter des "inputs" permettant de fournir des valeurs et données à l'application, mais aussi de modifier l'affichage des sorties.

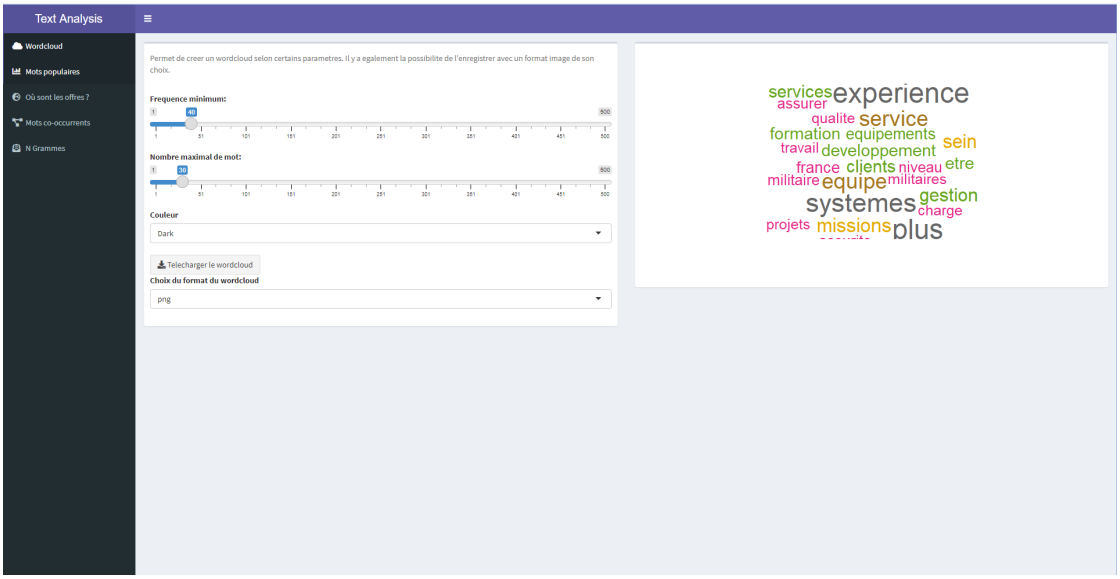
Une application Shiny se structure en deux parties :

- Un côté UI qui regroupe tous les éléments de mise en forme et d'affichage de l'interface elle-même, c'est-à-dire, les inputs et les sorties (graphiques, etc).
- Un côté Server, où sont exécutés les codes R permettant de produire les sorties telles que les graphiques, cartes interactives, etc et de les mettre à jour lorsqu'il y a un changement dans les valeurs d'inputs.

Pour les prochaines sous-parties, nous allons détailler l'utilité, les différents inputs et outputs pour chaque onglet de l'application.

4.1 Onglet Wordcloud

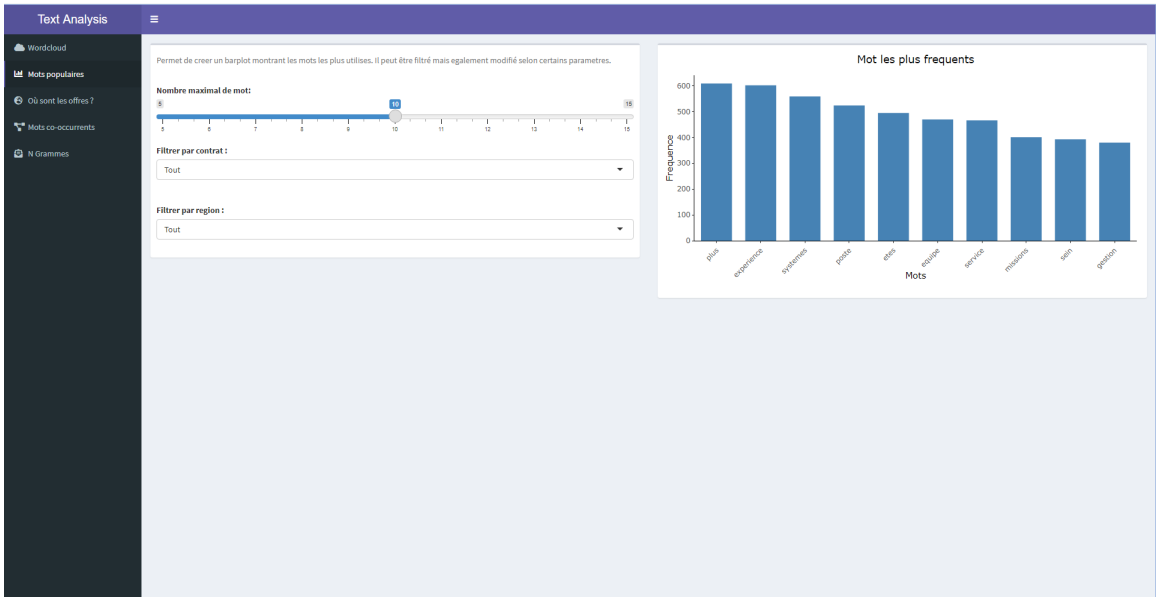
Cet onglet est le premier, celui qui s'affiche au lancement de l'application et permet simplement d'afficher un wordcloud du corpus de mot.



Les inputs permettent de modifier les paramètres de la fonction permettant d'afficher le wordcloud. Ainsi, nous pouvons changer la fréquence minimum pour apparaitre dans le wordcloud, mais également choisir le nombre maximal de mot à afficher. Également, nous pouvons modifier la nuance de couleur utilisée dans le wordcloud. Nous avons également ajouté un bouton pour télécharger le wordcloud obtenu dans ses fichiers en laissant l'utilisateur choisir le format de l'image (Png, jpeg, etc).

4.2 Onglet Mot populaires

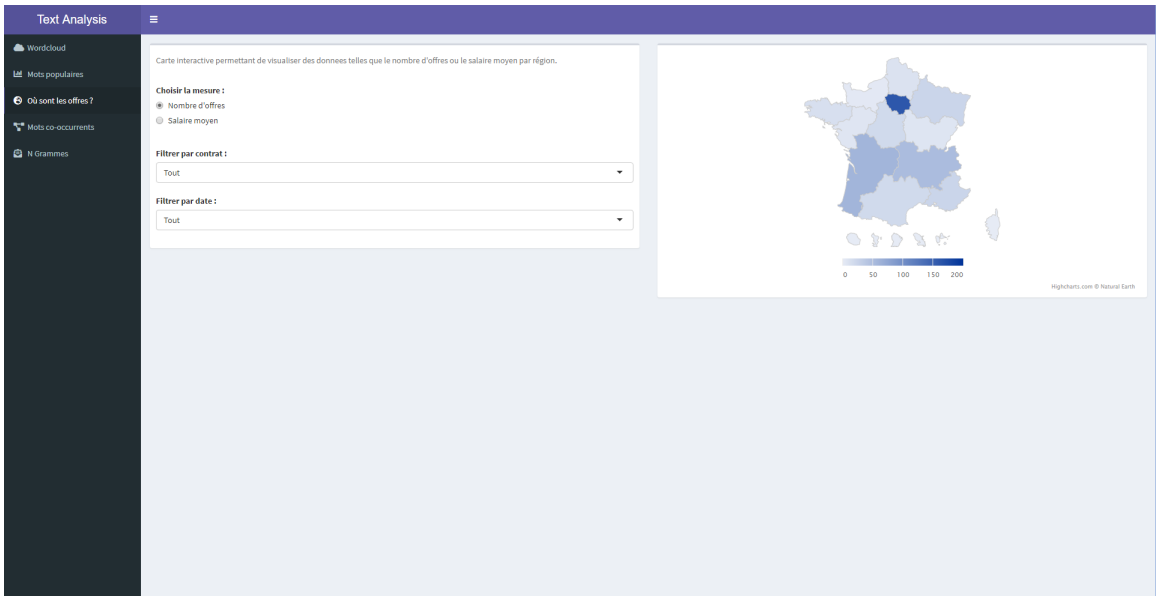
Cet onglet s'affiche lorsqu'on appuie sur son nom dans le menu situé à gauche de l'application. Il permet de créer un barplot montrant les mots les plus utilisés dans le corpus.



Les inputs permettent de filtrer les données qui permettent de créer le corpus. Nous avons choisi d'ajouter un filtre sur le type de contrat et la région. Également, nous pouvons modifier le nombre maximal de mot à afficher sur le graphique.

4.3 Onglet Où sont les offres ?

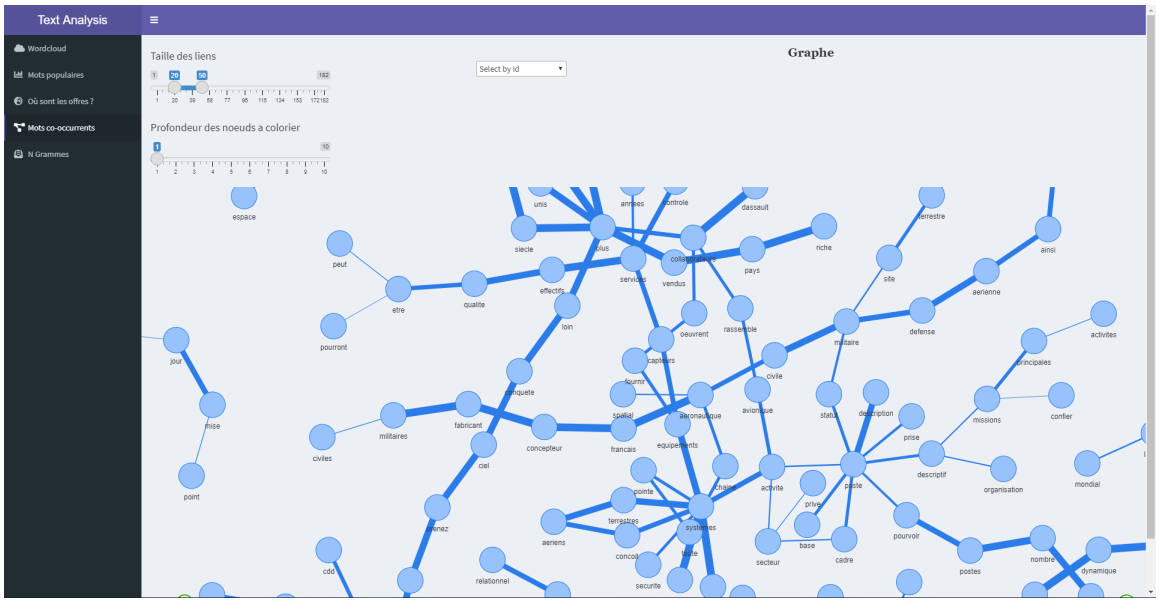
Dans cet onglet, nous pourrons apprécier des données géographiques. En effet, nous avons deux mesures différentes qui peuvent être choisies à l'aide de boutons radio : le nombre d'offres et le salaire moyen en fonction des régions. À noter que, lorsque nous avions des données manquantes pour les salaires, la valeur 0 sera attribuée (sinon la région n'apparaitrait pas, ceci est dû au package highcharter).



Les inputs permettent de filtrer les données. Nous avons choisi d'ajouter un filtre sur le type de contrat et la date (qui est une variable catégorielle). Cette carte est interactive et permet d'afficher les valeurs en passant la souris sur une région.

4.4 Onglet Mots co-occurents

Il s'agit dans cet onglet d'une représentation sous forme de graphe des mots du corpus. Les mots sont ainsi représentés par les nœuds du graphe et les liens entre les nœuds décrivent la régularité avec laquelle les mots se produisent simultanément dans les documents.

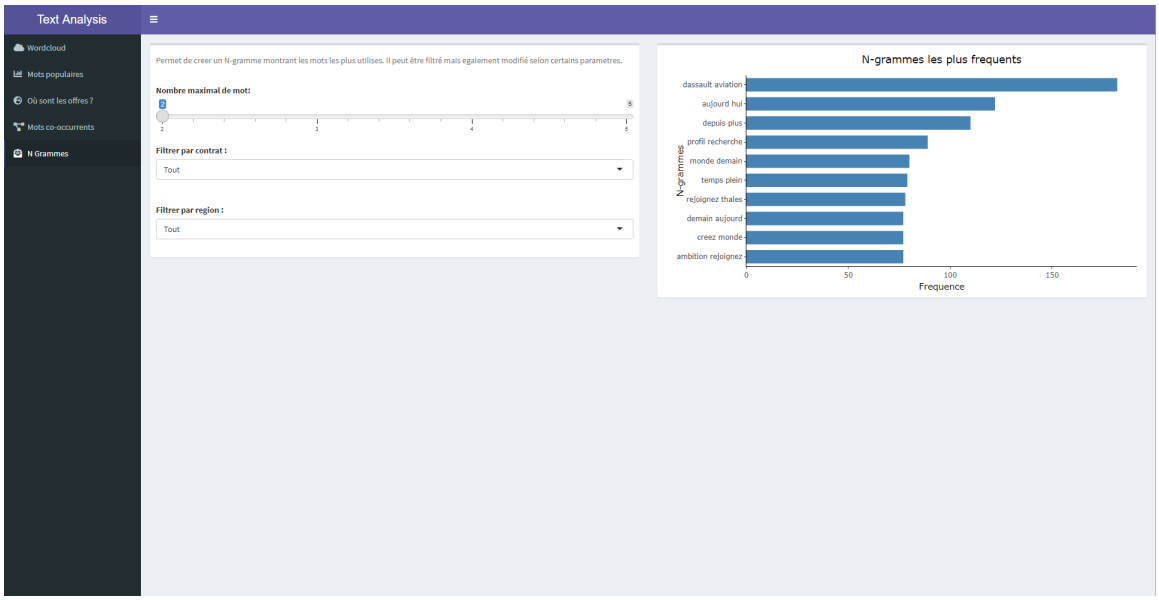


Il est possible de choisir, grâce au curseur *taille des liens*, le nombre minimal et maximal de fois où deux mots co-occurents dans le corpus. Si l'on souhaite visualiser un mot en parti-

culier, on peut le sélectionner dans la liste déroulante. Il est alors mis en surbrillance, ainsi que les mots qui l'entourent, dont on peut choisir le nombre grâce au curseur *profondeur des nœuds à colorier*.

4.5 Onglet N Grammes

Dans ce dernier onglet, nous avons voulu ajouter une fonctionnalité pour afficher les n-grammes les plus récurrents dans le corpus. Nous avons choisi, pour ce faire, de créer un graphique à barre horizontale affichant les 10 N-grammes les plus utilisés.



Nous avons ajouté un input permettant de choisir le nombre de mots dans chaque expression, allant de 2 à 5. Il y a également des filtres permettant de filtrer les données du corpus par contrat et par région.

5 Analyses

Dans cette partie, nous allons nous baser sur les sorties de notre tableau de bord pour pouvoir analyser notre corpus.

5.1 Analyse des mots les plus présents dans le corpus

Le wordcloud du premier onglet nous permet d'avoir une vision globale des mots les plus utilisés dans le corpus. Ceux qui y sont le plus mis en avant sont les mots expérience, système, mission, développement, client, service ou encore gestion. En règle générale, on peut dire que beaucoup d'offres souhaitent des candidats avec une certaine expérience dans l'emploi demandé, qui est en relation avec l'utilisation d'un certain type de système, le développement ou la gestion de quelque chose pour un client/l'entreprise (une base de données peut-être ?).

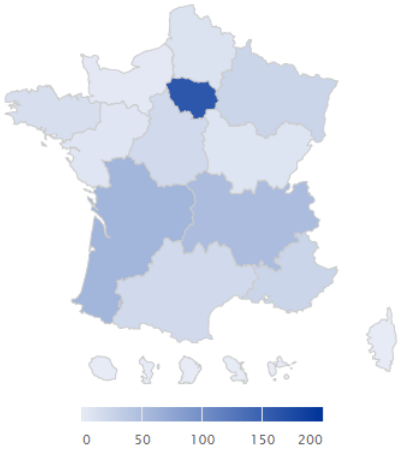
Afin d'obtenir plus de détail en fonction du type de contrat ou de la région, on peut s'intéresser au barplot dans l'onglet "Mots populaires". Ici, on se rend compte que les CDD et les stages en France sont ceux ayant le plus de relation avec le thème rechercher, on y retrouve ainsi le mot militaire ou armée. Les CDI, sont, quant-à-eux, plus en relation avec les systèmes, les clients ou encore le développement. Pour ceux d'Auvergne-Rhône-Alpes, les mots les plus populaires sont développement, système et client (pas plus d'information ici). La région dont les mots changent le plus est Haut-de-France avec production, équipement, client, collaboration et gestion. On peut hypothétiser que la région de la capitale de France se baserait peut-être plus sur la production d'équipement ou bien leur gestion en collaboration avec une équipe pour un client.

5.2 Analyse géographique du nombre d'offres et du salaire moyen selon les régions

Passons maintenant à une analyse géographique de nos données, avec l'onglet "Où sont les offres ?" où nous pouvons analyser une première métrique : le nombre d'offres. Comme nous pouvions nous y attendre, la région ayant le plus d'offre est la région Île-de-France qui possède environ trois fois plus d'offres que la deuxième région, la Nouvelle-Aquitaine. Elle est suivie de près par Auvergne Rhône Alpes. Ce podium possède quasiment deux tiers des annonces françaises de notre base de données.

Il existe aussi quelques régions qu'on peut honorablement mentionner comme PACA ou Grand Est. À noter, qu'il n'y a aucune annonce sur les îles françaises (Corse et outre-mer).

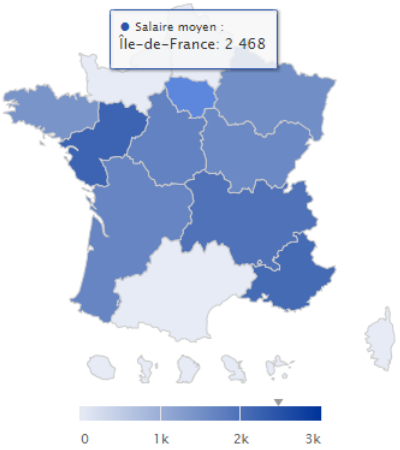
On peut observer ci-dessous, le nombre d'offres par région :



Nous avons aussi appliqué un filtre sur les contrats en regardant, ceux de type stage ou CDI, mais n'avons pas remarqué une grande différence avec ou sans. Cependant, lorsque l'on filtre sur CDD, surtout avec la région Grand Est, on remarque qu'elle possède énormément de contrats de ce type dans nos données, en quasi-égalité avec Île-de-France. Les autres régions possèdent que très peu d'annonce avec des postes en CDD.

Ensuite, nous pouvons analyser les salaires moyens selon les régions, et là encore, sans surprise, nous retrouvons l'Île-de-France avec en moyenne 2468€ brut mensuel, mais avec une différence non-significative avec le Pays de la Loire (2200€) et la région PACA (2083€), qui possèdent des salaires relativement élevés également.

Concernant les salaires voici la carte que nous avons obtenu grâce au tableau de bord :



Il existe beaucoup de données manquantes, ceci est dû au fait que certaines régions ne possèdent pas d'annonces, mais aussi parce qu'il y a beaucoup d'annonce sans salaire annoncé. L'analyse peut donc perdre de son sens quand le salaire moyen d'une région est calculé sur très peu de salaires (voir un seul). Ainsi, les deux régions ayant le salaire moyen le plus bas sont la Bretagne (1425€) et le Grand Est (1522€).

En jouant avec les filtres, on peut remarquer que les salaires en CDI semblent plus élevés que ceux en CDD, pour la majorité des régions. Avec un filtre sur la date de publication des annonces, on se rend compte également que les annonces les plus vieilles sont en général moins bien payés que les nouvelles annonces. Cela peut s'expliquer pour le manque d'intérêt de ces posts par les demandeurs d'emploi (peut-être sous-payés ?).

5.3 Mots co-occurents

Pour analyser le corpus des offres d'emploi d'une autre manière, nous avons représenté les mots co-occurrents sous forme de graphe. Ainsi, deux mots apparaissant consécutivement dans le corpus seront liés. Par ailleurs, un poids est associé à chaque lien, ce qui en fait un graphe pondéré, permettant de visualiser le nombre de fois où les mots ont co-occurré.

Tout d'abord, nous pouvons remarquer, que le nombre maximal de co-occurrence entre deux mots est de 182. Cela correspond à la co-occurrence entre le terme *Dassaut* et le terme *Aviation*, qui n'est autre que le nom d'une entreprise. Avec plus de 100 co-occurrences, on retrouve les termes *aujourd* avec *hui*, et *depuis* avec *plus*.

5.4 N-grammes

Il nous est également apparu intéressant de compléter l'analyse fournie par le graphe, en étandant l'analyse à une sous-séquence de $n > 2$ éléments, formant alors un N-gramme.

Bien évidement, lorsque seuls 2 mots sont demandés, nous retrouvons les mêmes conclusions qu'avec le graphe, à savoir la suite des termes *Dassaut* et *Aviation* qui revient le plus souvent. Ce qui est en revanche intéressant, c'est lorsque nous filtrons par contract, nous n'avons plus les mêmes résultats. En effet, pour les CDD, nous retrouvons en première position *Armée air*, qui fait donc référence à l'Armée de l'Air. Et pour les stages, après *aujourd hui*, on trouve à nouveau *Dassaut Aviation*, ainsi que deux bi-grammes contenant le mot *thales*, qui est une entreprise. On peut donc supposer que Dassaut Aviation propose plutôt des CDI ou des stages, l'Armée de l'Air quant à elle est plutôt tournée vers les CDD, et enfin Thalès recherche des stagiaires. L'analyse par région ne nous a pas permis de tirer de conclusions spécifiques dans la mesure où les principaux bi-grammes diffèrent pour chaque région.

Dans le cas de 3-grammes ou plus, nous avons pu remarquer que la fréquence d'apparition des principaux N-grammes était sensiblement la même : entre 71 et 77 fois pour les 10 premiers 3-grammes, 77 fois pour les 7 premiers 4-grammes ainsi que pour les 6 premiers 5-grammes.

5.5 Latent Dirichlet Allocation (LDA)

Afin d'analyser de manière encore plus pousser notre corpus, nous avons choisi d'utiliser la Latent Dirichlet Allocation (ou l'allocation de Dirichlet latente en français). De manière générale, il considère un document comme une composition de plusieurs topics : par exemple, un document A peut-être composé de 30 % de topic 1 et de 70 % de topic 2, tandis qu'un document B peut être composé de 80 % de topic 1 et de 20% de topic 2.

La visualisation est, quant à elle, très simple avec le package LDAvis et ne requière que quelques lignes de code pour une interface web. À noter, qu'au départ, nous voulions l'intégrer à notre tableau de bord, mais cela nous produisait des erreurs que nous n'avons pas pu résoudre à temps. Malgré tout, nous la trouvons tout de même intéressante et si vous voulez la manipuler vous-même, cliquer sur ce lien : `https://pierrelgz.github.io/LDAvis/#topic=&lambda=0.6&term=`

5.5.1 Présentation de l'algorithme

Le modèle Latent Dirichlet Allocation est un modèle probabiliste génératif qui permet de décrire des collections de documents de texte ou d'autres types de données discrètes.

La LDA fait partie d'une catégorie de modèle appelé "topic model" qui cherchent à extraire des topics (ou thèmes) à partir de textes. Ceci permet d'obtenir des méthodes efficaces pour le traitement et l'organisation des documents de ces archives : organisation automatique des documents par topics, recherche, compréhension et analyse du texte, ou même résumer des textes. Aujourd'hui, ce genre de méthode s'utilise fréquemment, notamment en fouille de données et en traitement automatique des langues.

Ses trois principaux inputs sont : un nombre K de topics que l'on aura choisi au préalable, un dictionnaire de mots (mots présents dans tout) et un corpus de fréquence (la fréquence de chaque mot dans le document).

L'algorithme fonctionne comme suit :

1. Dans un premier temps, chaque topic se voit attribuer à chaque mot de chaque document selon une distribution de Dirichlet sur un ensemble de K topics (soit de manière aléatoire).
2. On va chercher ensuite à améliorer cette affectation en mettant à jour le topic le plus représentatif de chaque mot. Ce nouveau topic est celui qui aurait la plus forte probabilité de générer ce mot dans ce document. On fait donc l'hypothèse que tous les topics sont corrects, sauf pour le mot en question. On a donc : pour chaque mot (w) de chaque document (d), deux probabilités pour chaque topic (t) :
   - p(t | d) : la probabilité que le document d soit assigné au topic t.
   - p(w | t) : la probabilité que le topic t soit assigné au mot w.

   Le nouveau topic est alors celui avec la probabilité p(t | d)* p(w | t) la plus élevée, soit la probabilité que le topic t génère le mot w dans le document d.

3. Le tout est alors répété jusqu'à ce que les assignations ne bougent plus, pour finalement obtenir les thèmes de chaque document, ainsi que les mots associés.

5.5.2 Tuning

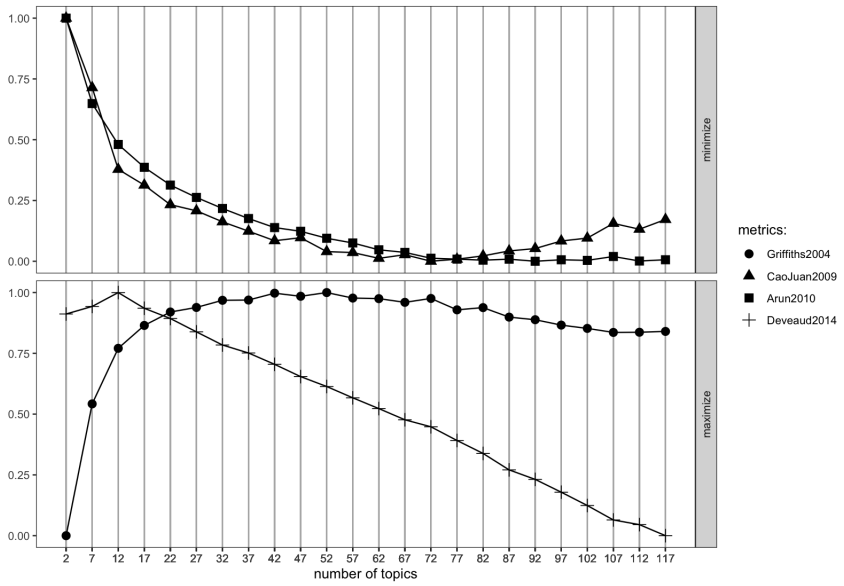
Nombre de topics : Le choix du nombre de topics pour la LDA est un problème ouvert, mais on peut s'appuyer sur des outils comme le package ldatuning pour se rapprocher de la réalité. Ce dernier calcul plusieurs mesures en une seule fois et affiche leurs résultats sous forme d'un graphique pour nous permettre de prendre la meilleure décision.

4 mesures sont calculées avec ldatuning :

— Celles que l'on veut maximiser :
  — Griffiths2004
  — Deveaud2014
— Celles que l'on veut minimiser :
  — CaoJuan2009
  — Arun2010

Chaque mesure a différents avantages/inconvénients et peuvent être plus ou moins applicable aux spécificités de notre jeu de données.

Après avoir fait tourner la fonction pour un nombre de topic entre 2 et 120, on obtient ce graphique :



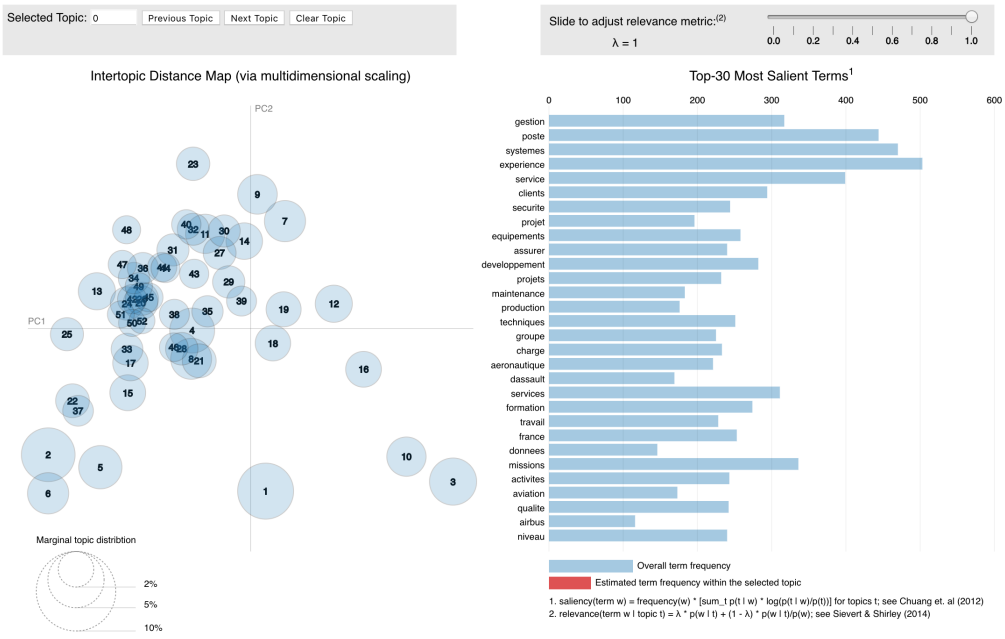
En regardant celui-ci, on remarque directement que la mesure Deveaud2014 ne se maximise pas du tout au fil du temps : elle n'est pas applicable au jeu de donnée. Les résultats des autres mesures nous permettent de conclure que le nombre optimal de topic se situerait entre 52 (point haut de la courbe Griffiths2004) et 77 (point de croisement le plus bas pour les courbes Arun2010 et CaoJuan2009).

Alpha et beta : LDA prend aussi en paramètres alpha et eta. Le paramètre alpha indique à la LDA le nombre de sujets à partir desquels chaque document doit être généré et eta indique le nombre de sujets dans lesquels chaque mot doit se trouver. Pour leur valeur, on fixera un alpha de 50 / K (avec K = nombre de topics) et un eta de 0.1 comme suggéré par Griffiths and Steyvers dans "Finding Scientific Topics" (2004).

5.5.3 Visualisation avec LDAvis

LDAvis : Pour visualiser la LDA implémentée, nous sommes partis sur le package LDAvis qui ne requière que quelques lignes de code pour une interface web. De manière générale, le package extrait des informations d'une LDA entrainer pour créer une visualisation interactive sur le web. Le seul point négatif de ce dernier est que nous n'avons pas réussi à trouver d'exemple de code qui utilise autre chose que la LDA du package tm et qu'il faut donc changer le type de certains inputs (pas de type corpus ici). On a choisi de partir sur 52 topics, de sorte à obtenir une visualisation sans avoir trop de topics se superposant (tout en restant dans l'intervalle choisi préalablement). Pour rappel, vous pouvez retrouver cette visualisation sur ce lien : `https://pierrelgz.github.io/LDAvis/#topic=&lambda=0.6&term=`

Une fois les transformations faites et la LDA implémentée, on obtient une jolie interface html :



Il y a plusieurs points importants à comprendre sur cette interface :

1. Le panneau de gauche permet une visualisation des topics dans un espace à deux dimensions avec la surface des cercles proportionnelle à la quantité de mots appartenant à chaque topic du dictionnaire. Ces cercles sont affichés via un algorithme (Jensen–Shannon divergence) qui permet de convertir un nombre important de dimensions en seulement deux, plus interprétable, en fonction des mots qui les composent. À noter que plus les topics sont proches, plus ils auront de mots en commun.
2. Le panneau de droite présente un diagramme à barres horizontal des 30 termes les plus "salient" (ou saillante en français). "Salient" est une mesure qui permet de mettre en avant les mots les plus utiles pour identifier un topic (voir formule en dessous du graphique), plus elle est élevée, plus le mot est utile. Concernant les couleurs affichées, le bleu représente la fréquence d'un terme donné dans l'ensemble du corpus et le rouge, la fréquence du terme dans un topic spécifique.
3. Le curseur λ permet de classer les termes en fonction de leur pertinence. Par défaut, les termes d'un sujet sont classés par ordre décroissant selon leur probabilité spécifique au sujet ($\lambda = 1$). En déplaçant le curseur, il est possible d'ajuster le classement des termes en fonction de s'ils sont pertinents ou non pour un sujet spécifique. La valeur optimale générale suggérée par les développeurs du package est de $\lambda = 0.6$.

Analyse LDA : En parcourant les différents topics et en étudiant leurs relations les un par rapport aux autres, on peut en déduire plusieurs grands thèmes :

Certains sont un regroupement de plusieurs topics :

— Détails sur l'offre : topics 6, 15, 35, 46, 47 et 51 (entretien / travail / salaire / poste / prime / entraide / valeurs)

— Ingénierie électronique, mécanique et matériaux : topics 12, 29, 31, 42 et 45 (électronique, mécanique, assemblage, fabrication, matériaux, ingénieurs)

— Aérospatial : topics 1, 10, 39 et 48 (avionique / spatial / aéronefs)

— Détails propres aux entreprises du secteur : topics 21, 28 et 39 (Airbus / Ripault / CEA / Novae)

— Sécurité : topics 16, 18 et 19 (système, cybersécurité, Thales, informatique, réseaux, sécurité)

— Marketing et commercial : topics 7 et 38 (offre, client, vente)

— Data science : topics 40 et 41 (data, données, traitement, machine learning, apprentissage artificiel)

— Armée : topics 22 et 37 (marine, officier, grade, air)

— Mots anglais : topics 32, 34

Alors que d'autres thèmes peuvent être déduits à partir d'un seul topic :

— Maintenance : topic 27 (équipement, technicien, réparation)

— Santé : topic 2 (assistant, hôpitaux, pharmacien)

On remarque aussi que plusieurs entreprises ressortent dans le top 30 de LDAvis :

— RTE, responsable du réseau public de transport d'électricité haute tension.

— Dassault, constructeur aéronautique français.

— Sogitec, filiale de Dassault Aviation.

— Sodern, société aérospatiale française.

— CEA, acteur majeur de la recherche, au service de l'État.

— Ripault, rattaché à la Direction des applications militaires du CEA.

— Segula, groupe d'ingénieries dans les secteurs industriels : automobile, aérospatial, énergie, ferroviaire, life sciences, naval et télécoms.

— Novae avec l'aéronautique, le spatial et la défense.

6 Conclusion

Ce projet avait pour but la réalisation d'une application R Shiny permettant d'analyser le corps d'annonces d'offres d'emploi extraites, grâce aux méthodes de web scrapping, de sites accessibles en ligne. Cette procédure permettant alors d'alimenter une base de données, sur laquelle se base l'application R Shiny.

Il s'agissait donc d'explorer le corpus de manière descriptive, mais également de manière plus sophistiquée, avec les méthodes de recherche de thématiques telle que le topic modelling, que nous avons mis en œuvre au moyen de la LDA. Les analyses que nous avons pu réaliser nous ont donc permis de tirer diverses conclusions que ce soit sur les offres d'emploi en termes de contenu des annonces, mais également en termes de métriques sur le nombre d'annonces ou bien encore le salaire. Ainsi, nous avons pu mettre en évidence les mots apparaissant le plus souvent dans le corpus, ainsi que les mots co-occurrents, et ce, par région ou type de contrat. Une analyse géographique des annonces et du salaire a permi de mettre en évidence les régions les plus demandeuses d'emplois, et celles proposant les meilleurs salaires. Enfin, nous avons pu dégager différents thèmes dans le corpus.

Pour terminer, plusieurs pistes d'améliorations seraient envisageables. À commencer par l'intégration de la LDA à l'application R Shiny, qui à l'heure actuelle n'est pas possible, à cause d'erreur d'affichage.
De plus, il pourrait être intéressant de déployer l'application R Shiny grâce au service *Shinyapps.io*, permettant d'héberger des applications R Shiny dans le cloud.
Également, nous pourrions envisager de laisser le choix de la requête sur le thème ou la compétence qu'il souhaite à l'utilisateur.
Enfin, une comparaison, voire un complément des résultats de la LDA avec d'autres modèles serait une piste à envisager.

7 Crédits

— Get topics :

  — https://rstudio-pubs-static.s3.amazonaws.com/264464_213ddb5ec8204c5d9bfbc61f9b8
  html

  — https://cran.r-project.org/web/packages/ldatuning/vignettes/topics.html

— Get About LDAvis :

  — http://bl.ocks.org/AlessandraSozzi/raw/ce1ace56e4aed6f2d614ae2243aab5a5/
  #topic=0&lambda=1&term=

  — https://github.com/cpsievert/LDAvis