

**Baptiste
O'Jeanson**

**Rapport de stage de
spécialité**

Étudiant: Baptiste O'JEANSON

Maître de stage: Christian
FRISCH

Entreprise: Data Publica

Année: 2014-2015

Dates: du 01/06/2015 au
28/08/2015

Lieu: Paris, France

**Classification de signaux
entreprises avec une
approche machine learning**

**INFORMATION RETRIEVAL, TEXT MINING AND NATURAL
LANGUAGE PROCESSING**

Remerciements

Premièrement, j'aimerais remercier François Bancilhon¹, directeur général de Data Publica, et Christian Frisch², directeur technique de Data Publica, de m'avoir donné l'opportunité de découvrir le monde du travail en start-up. Je voudrais aussi les remercier de m'avoir fait confiance et de m'avoir confié une mission très intéressante.

Ensuite, je voudrais remercier Samuel Charron, Clément Chastagnol et Guillaume Lebourgeois qui m'ont suivi durant mon stage et qui ont enrichi mes connaissances en informatique.

Je souhaiterais également remercier l'ensemble de l'équipe de développeurs et l'ensemble de l'équipe de marketing de m'avoir accueilli si chaleureusement.

Enfin, merci à mon tuteur de stage, Nicolas Malandain, pour sa disponibilité.

1. François Bancilhon, présenté ici 1.2

2. Christian Frisch, présenté ici 1.1

Table des matières

Remerciements	3
1. Présentation de l'entreprise	5
1.1. L'entreprise	5
1.1.1. L'activité de Data Publica	5
1.1.2. L'équipe de Data Publica	6
1.2. C-Radar	7
1.2.1. Présentation commerciale de C-Radar	7
1.2.2. Présentation technique de C-Radar	7
2. Présentation du sujet	10
2.1. La fonctionnalité de C-Radar	10
2.2. Ma mission chez Data Publica	11
2.2.1. Inscription dans le domaine du <i>Big Data</i>	11
2.2.2. Synthèse du travail à réaliser	11
2.3. Présentation des signaux	12
3. Travail effectué	14
3.1. Démarche de travail	14
3.1.1. Mes acquis à l'INSA	14
3.1.2. Déroulement du stage	15
3.2. Travaux réalisés en Java, le <i>Text Mining</i> et le <i>Natural Language Processing</i> avec la bibliothèque de Stanford	15
3.2.1. Présentation de mon environnement de travail	15
3.2.2. La bibliothèque : Stanford Natural Language Processing	16
3.2.3. Première mise en pratique de la bibliothèque de Stanford	19
3.2.4. Amélioration de ma première application et prétraitement	22
3.2.5. Les travaux de prétraitement :	24
4. Conclusion	25
5. Résumé	26
A. Annexes	28
A.1. Titre de l'annexe	28

1. Présentation de l'entreprise

L'évolution des technologies et leurs usages ont fait exploser la quantité de données générées. Selon IBM, 2.5 milliard de gigabytes (GB) de données a été générée tout les jours de l'année 2012. De plus, cette quantité de données, double tout les deux ans. Cependant, seules 0,05% de ces données sont analysées.

L'exploration ou la fouille de données (« data mining ») consiste à en extraire des informations utiles, et ceci peut s'avérer très fructueux. La question principale qui se pose est de savoir comment utiliser intelligemment cette immense masse de données pour en tirer une plus-value ? C'est le rôle des entreprises spécialisées dans l'exploitation de ces données.

1.1. L'entreprise

La société Data Publica a été fondée en juillet 2011 par François Bancilhon et Christian Frisch, respectivement l'actuel directeur général et l'actuel directeur technique.

1.1.1. L'activité de Data Publica

Data Publica est un des précurseurs de l'open data en France. Cette société , qui a bénéficiée d'investissements technologiques faits en 2010 dans le cadre d'un projet de R&D, a été financée initialement par un groupe de business angels et le fonds d'amorçage **IT Translation**. Data Publica est un start-up spécialisée dans les données entreprises, l'open data, le big data et la dataviz. C'est une société relativement jeune, axée R&D. Son leitmotiv, alimenté par une équipe très dynamique et compétente, est la recherche constante du dépassement technique.

Historiquement, Data Publica ne faisait que de l'open-data. C'est-à-dire que la société se servait de données accessibles à tous (provenant d'institutions gouvernementales notamment) pour créer des jeux de données sur mesure pour des entreprises. Ainsi, la société s'est spécialisé dans l'identification des sources de données, leur extraction et leur transformation en données structurées.

Depuis quelques années, Data Publica se spécialise dans les données sur les entreprises française en dépit de son activité open-data qu'elle a progressivement mis de côté. Les services qu'elle propose ne sont plus tout-à-fait les mêmes. En effet, Data Publica réutilise les données open-data concernant les entreprises française dans son produit phare. Ce produit est lui même conçu pour les entreprises du B2B. Le produit est décrit en partie 1.2.

Data Publica participe également à de nombreux projets de recherche français et européens tels que XDATA, Diachron ou Poqemon, en partenariat avec l'INRIA.

1.1.2. L'équipe de Data Publica

Data Publica emploie 14 personnes réparties en 2 équipes : une équipe commerciale (4 personnes) et une équipe technique (10 développeurs). Les deux équipes travaillent chacune dans son open-space. Pendant mon stage, j'ai été immergé au sein de l'équipe technique.

L'équipe technique : Elle est composée de 10 développeurs (ordonnés par ancienneté visible en figure 1.1) :

- Christian Frisch, directeur de l'équipe
- Thomas Dudouet, **Java / Back end** développeur
- Guillaume Lebourgeois, chef de produit C-Radar
- Samuel Charron, **Data scientist Python et mon maître de stage**
- Loïc Petit, **Java JBM**
- Clément Chastagnol, **data scientist Python et mon maître de stage**
- Clément Déon, **Front end** développeur
- Fabien Bréant, **Back end** développeur
- Jacques Belissent, **?**
- Vincent Ysmal, **Java / Back end** développeur



Figure 1.1. – L'équipe technique de Data Publica

L'équipe commerciale : Elle est composée de 4 commerciaux (ordonnés par ancienneté visible en figure 1.2) :

- François Bancelhon, directeur général
- Benjamin Gans, Responsable Communication et Marketing
- Emmanuel Jouanne, Business Development Manager

- Philippe Spenato, Ingénieur d'affaire
- Justine Pourrat, Responsable Communication et marketing



(a) François B. (b) Philippe S. (c) Justine P.

Figure 1.2. – L'équipe commerciale de Data Publica

1.2. C-Radar

1.2.1. Présentation commerciale de C-Radar

Son produit est un moteur de recherche B2B (Business to Business). Celui-ci a pour objectif de permettre aux services ventes et marketing des entreprises B2B de vendre plus et mieux. Ce moteur de recherche, appelé C-Radar, est un produit de vente prédictive construit sur une base de référence des entreprises françaises. Il regroupe beaucoup d'informations sur les entreprises françaises, dont notamment leurs informations administratives, financières et toutes celles qui découlent de leur communication sur les réseaux sociaux et le web.

C-Radar est un concentré de technologies du big data. En effet, il utilise diverses technologies comme le crawling, le scraping ou encore le machine learning. Ceci afin d'offrir à l'utilisateur diverses fonctionnalités : moteur de recherche d'entreprises, fiche d'activité d'entreprises avec contacts commerciaux, détection de nouveaux prospects, scoring de prospects existants, segmentation automatique d'entreprises, identification de marché, etc.

1.2.2. Présentation technique de C-Radar

Les technologies utilisées par C-Radar

Pour répondre aux problématiques auxquelles Data Publica se confronte, la société a acquis 4 expertises majeures :

- Le web crawling / web scraping : la récupération des données ;
- Le data mining / text mining : l'analyse, l'extraction et l'enrichissement des données ;
- Le machine learning : l'apprentissage automatique à partir de données structurées ;
- La dataviz : la visualisation des données.

Le crawling : Le crawling est l'action réalisée par un programme informatique, appelé le web crawler, qui va de site en site afin d'en extraire automatiquement toute l'information qui est présente sur les différentes pages. Cette technique est utilisée notamment pour l'extraction de données non structurées : la structure du site n'est pas connue à l'avance, l'extraction des

données se fait directement sur le contenu (c'est à dire le code HTML) de la page crawlée. Ce processus est « brutal ».

Le scraping : Le scraping est l'action réalisée par un programme informatique pour extraire des unités d'information structurées d'un site web. Contrairement au crawling, il est question d'extraire des données précises, et pas la totalité des données disponibles sur le site. Le site « scrapé » et sa structure doivent donc être connus et analysés à l'avance afin d'adapter le scraper au site. Ce processus est « intelligent ».

Le data mining / text mining : Une fois des sites web crawlés et scrapés, ou que des flux (RSS ou réseaux sociaux) aient été captés, on peut commencer à analyser le contenu récupéré à la recherche d'informations sous forme de patterns particuliers, par exemple. On analyse afin de normaliser des problèmes d'encodages, de structures de date, de numéros de téléphone, etc. Globalement, cette phase consiste à regarder les données « dans le fonds des yeux »¹ afin de voir leurs fonds mais aussi leurs formes.

Le machine learning : Quand les données sont correctement formatées et normalisées, on peut construire des applications capable d'apprendre automatiquement de ces données, de les classer. Ainsi quand on aura une nouvelle donnée l'application sera capable de prédire sa classe d'après ses caractéristiques.

L'idée derrière le machine learning est de pouvoir extraire automatiquement des informations d'une nouvelle donnée et ainsi prédire une classe de donnée.

La dataviz : C'est la dernière étape. Elle présente les données de manière visuelle et interprétable. Ainsi, on peut comprendre plus facilement et rapidement les informations extraites des données.

L'architecture technique de C-Radar

L'architecture de C-Radar peut être divisée en plusieurs parties (visible en figure 1.3) :

- Différentes bases de données pour différents stockages (une base de données Mongo, une base de données Cassandra et une base de données PostgreSQL) ;
- Un moteur de recherche sémantique (Elastic Search) ;
- Un gestionnaire de queue (RabbitMQ) ;
- Différents plugins Python s'interfaçant avec le gestionnaire de queue RabbitMQ ;
- Un gestionnaire de flux entre les différents composants précédents, le Workflow ;
- Une application Java s'interfaçant avec Elastic Search et les bases de données Mongo et PostgreSQL.

Le JBM ou Java Base Manager : Le JBM est le projet Java qui rassemble le Workflow et l'application app.c-radar.com. Ce projet a été conçu et construit au dessus de Spring. Le framework Spring est une plate-forme Java qui fournit une architecture complète permettant de développer des applications Java. Spring gère l'architecture de manière à ce que le développeur n'est à s'occuper que de l'application. Il prend en charge énormément de chose dont notamment le modèle MVC, la sécurité de l'application, l'interface avec les gestionnaire de queue, etc. (Pour plus d'information, voir <http://spring.io/>).

1. [steph.canu](http://steph.canu.fr).

Le Workflow : Le Workflow est le gestionnaire de flux permettant de lancer les différents processus de récupération et d'analyses des données. Il gère tout ce qui est « computing ». C'est lui qui lance RabbitMQ qui lui même lance l'exécution des plugins Python gérant le processus de crawling de sites web, par exemple. Une fois les sites webs crawlés, le Workflow va les stocker dans Cassandra. Il gère tout ce qui est exécution des plugins Python (crawling et scraping du web, capture et catégorisation des signaux, etc), stockage des données produites à l'issue de ces exécutions et indexation dans Elastic Search.

Pour résumer, il prépare les données que l'application va se charger de présenter à l'utilisateur.

L'application app.c-radar.com : L'application, « présente » les données aux utilisateurs. Elle fait le reporting des données produites par le Workflow. Elle permet de rechercher des entreprises, de voir leur répartitions géographiques, de créer des listes d'entreprises, etc. C'est l'application visible et utilisée par l'utilisateur.

Les bases de données : Les bases de données stockent différents types de données. La base Mongo stocke les données liées aux entreprises et les signaux, par exemple. (MongoDB est une base de données NoSQL orientée documents. Les documents y sont stockés au format JSON.)

Les plugins Python : Enfin, les plugins Python sont des applications Python connectées à d'autres composants, afin d'exécuter une tâche sur des données. Ces données sont reçues depuis d'autres composants et le plugin leur retourne les résultats de sa tâche.

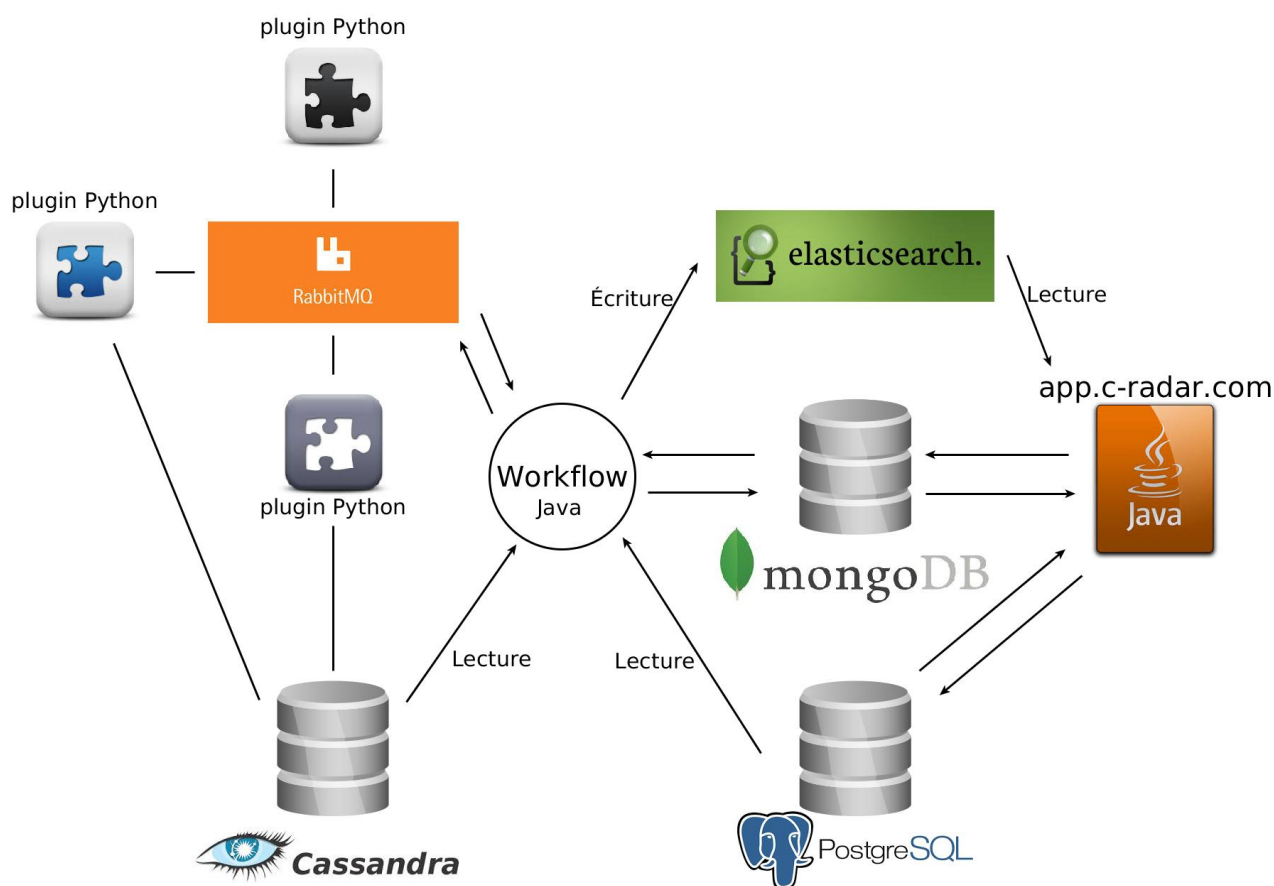


Figure 1.3. – L'architecture générale de C-Radar

2. Présentation du sujet

À la suite d'une candidature spontanée et de premiers contacts avec Thomas Dudouet et Clément Chastagnol, je me suis entretenu avec Christian Frisch afin de savoir quelle pourrait être ma contribution chez Data Publica. Ayant suivi une formation à l'INSA plutôt orienté big data / data mining, Christian Frisch m'a alors proposé de travailler sur l'une des fonctionnalités du produit C-Radar.

2.1. La fonctionnalité de C-Radar

L'objectif de mon stage serait le suivant : améliorer la fonctionnalité de C-Radar permettant d'être au courant de l'intégralité de la communication des entreprises françaises et belges simplement en s'abonnant à un newsletter.

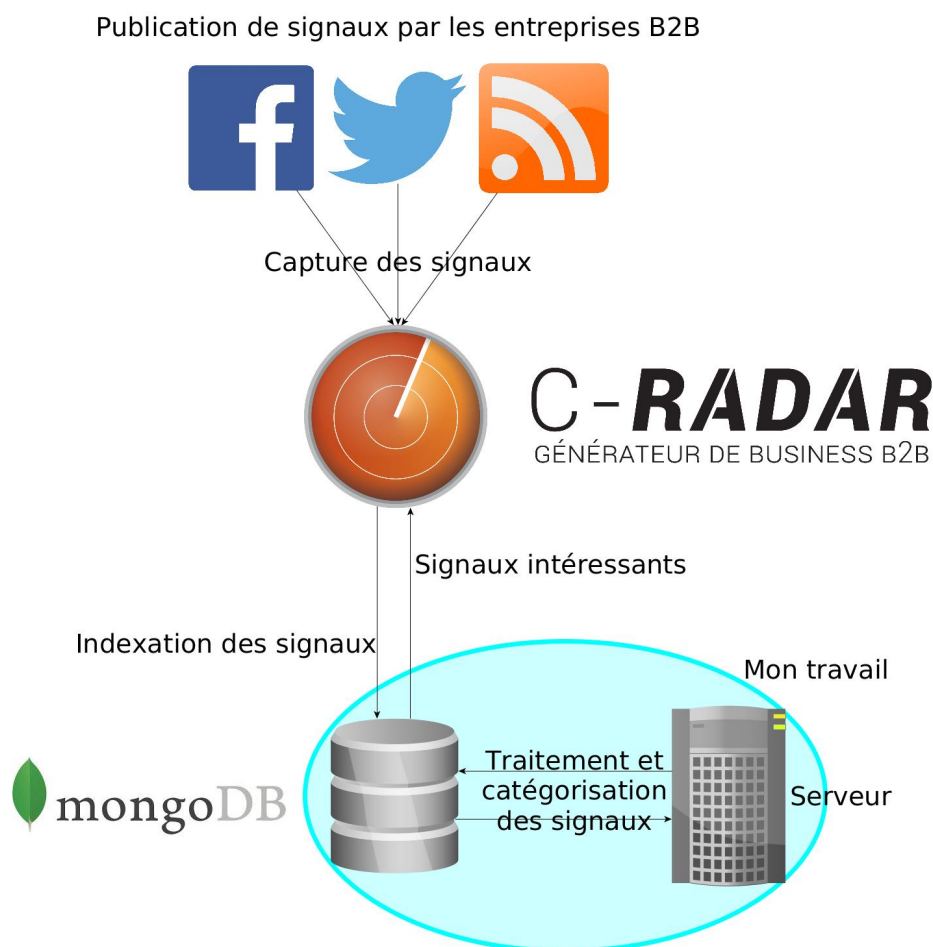


Figure 2.1. – Le fonctionnement générale de cette fonctionnalité.

De l'offre d'emploi, à la participation à des salons, en passant par les nominations de personnel ainsi que les présentations des derniers produits ou encore des éventuelles levées de fonds

ou investissements, la communication des entreprises n'est plus équivoque mais belle et bien ordonnée grâce à cette fonctionnalité. On peut désormais savoir quelles sont les derniers postes à pourvoir chez Orange, par exemple, ou encore les dernière nomination qui ont eu lieux à la Banque Postale.

Le fonctionnement générale de cette fonctionnalité est le suivant (visible en figure 2.1) :

1. C-Radar capte tout les signaux émis par les entreprises sur les réseaux sociaux (Facebook et Twitter), dans les médias ou via des flux RSS. Ces signaux sont ensuite stocké dans une base de données Mongo.
2. Une fois ces signaux capturés et stocké, il faut les traiter afin d'identifier l'intérêt potentiel de leur contenu.

C'est sur ce second point que j'interviens.

2.2. Ma mission chez Data Publica

Ma mission est de construire une chaîne de traitement automatique, un plugin Python, récupérant la liste des signaux émis par les entreprises en entrée depuis un point d'API, leur appliquer les traitements nécessaires afin de fournir, en sortie, la liste des signaux intéressants ainsi que leur catégories respectives.

Il s'agit donc de construire une application, un plugin Python, capable de classifier un signal dans une catégorie par la seule connaissance de son contenu (éventuellement un titre). L'application traitera donc des documents textuels.

Le travail se divise en deux étapes :

1. Appliquer une série de prétraitements sur le contenu des signaux afin de sélectionner les features jugés porteurs d'information. En effet, comme les données manipulées sont textuelles, il faut filtrer certaines chaînes de caractères considérées comme du bruit.
2. Construire un classifieur à partir des features sélectionnés parmi les données que l'on juge porteur d'information.

2.2.1. Inscription dans le domaine du *Big Data*

Les tâches qui découlent de ma mission, et notamment tout les prétraitements, sont directement liées à la discipline de la fouille de textes (*Text Mining*) (et de l'*Information Retrieval*) pour trouver des informations dans le contenu des signaux. Elles sont également liées à la discipline du traitement automatique du langage naturel (*Natural Language Processing*). Enfin, la construction du classifieur automatique implique des compétences en *Machine Learning*.

2.2.2. Synthèse du travail à réaliser

Si l'on devait résumer le travail à effectuer (visible en figure 2.1), on pourrait le synthétiser comme ceci : Construire une application (plugin Python), qui prend en entrée un signal émis par une entreprise et répond aux questions suivantes :

1. Ce signal a-t-il de l'intérêt ?
2. Si oui, quel est le sujet du signal ? Parle-t-il d'une offre d'emploi, d'une nomination, d'une levée de fond, etc ?

La figure 2.2 montre ce que l'application (le plugin Python) doit être capable de faire.

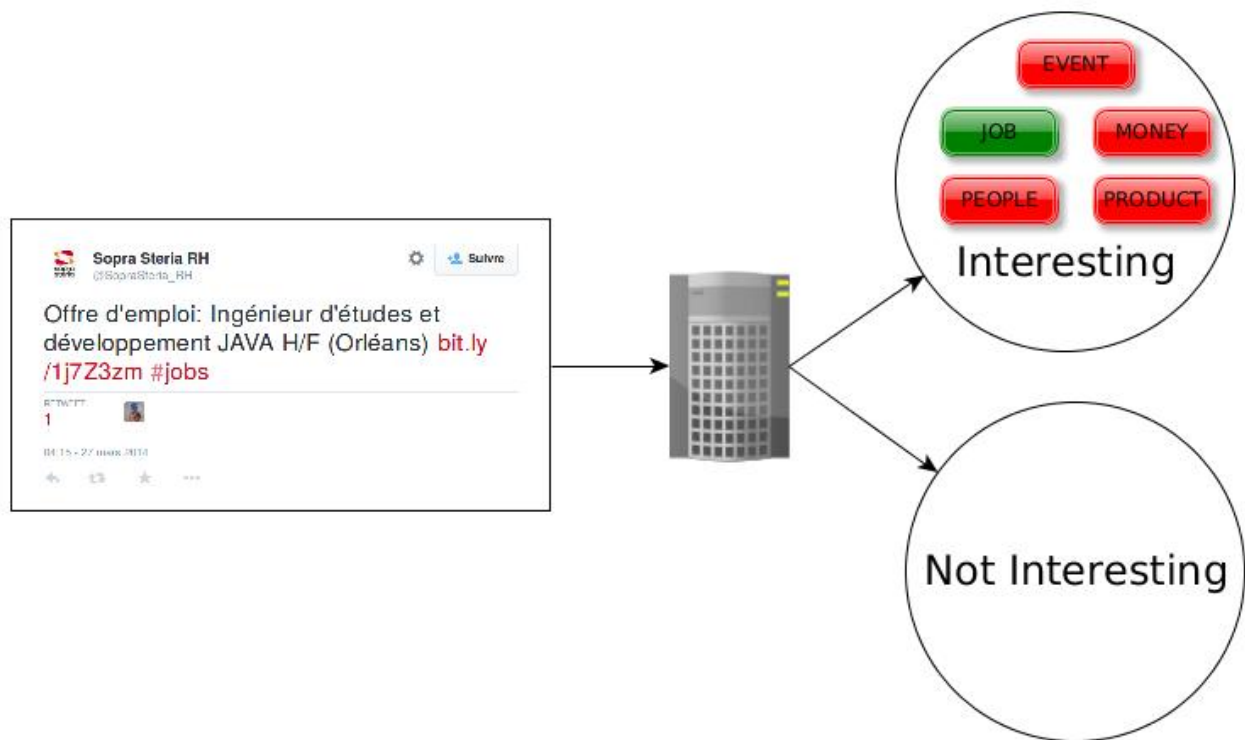


Figure 2.2. – La tâche du plugin Python.

2.3. Présentation des signaux

Les signaux sont des posts Facebook, des tweets ou bien des flux RSS publiés par des entreprises.

Hypothèses de départ : On considérera qu'un signal est intéressant si son contenu a pour sujet :

- Une offre d'emploi (ou un stage) à pourvoir au sein de l'entreprise qui l'a postée. Par la suite, on associera le tag **JOB** à cette catégorie.
- Un évènement auquel l'entreprise participe (un salon par exemple). Par la suite, on associera le tag **EVENT** à cette catégorie.
- Un produit que l'entreprise vient de présenter. Par la suite, on associera le tag **PRODUCT** à cette catégorie.
- Une nomination d'un employé dans l'entreprise ou d'une entreprise vers une autre entreprise. Par la suite, on associera le tag **PEOPLE** à cette catégorie.
- Une levée de fonds, un investissement, ou encore une déclaration de résultats ou de chiffre d'affaire.. Par la suite, on associera le tag **MONEY** à cette catégorie.

Exemple de signal type pour chaque catégorie :

- **JOB** : « Offre d'emploi : Ingénieur d'études et développement JAVA H/F (Orléans) <http://t.co/LOvN8rLH1e> #jobs »
- **EVENT** : « Fraispertuis sera présent dès demain jusqu'à dimanche inclus au salon « Tourrissimo » de Strasbourg Parc des Expositions, Hall 21 Stand B40, venez rendre visite au capt'ain Fraispl ! »
- **PRODUCT** : « Découvrez quelques unes de nos réalisations de Pergola Biotempérée pour notre clientèle de Toulouse et sa région. Plus d'informations sur notre site www.pergola-

biotemperee.com »

- **PEOPLE** : « Matthieu Frairot a été nommé Directeur Associé au sein de l'agence FullSIX France, l'agence marketin <http://t.co/Z2ENeeWZmF> »
- **MONEY** : « Keyrus : Publication des résultats annuels 2013. <http://t.co/k4TPJ11fW4> »

État de l'ensemble des signaux : Les travaux a réalisé implique de l'apprentissage supervisé. Ainsi, une base de signaux a été validée manuellement. Au 8 juin 2015, seul 1426 signaux étaient validés manuellement avec potentiellement un tag (JOB, EVENT, PRODUCT, MONEY ou PEOPLE), dans le cas où le signal est intéressant.

Le nombre de signaux validés par classes était le suivant :

- la classe **JOB** : 123 ;
- la classe **EVENT** : 100 ;
- la classe **MONEY** : 14 ;
- la classe **PRODUCT** : 6 ;
- la classe **PEOPLE** : 0 ;
- le reste (soit 1183) est dit « inintéressant ».

3. Travail effectué

Des travaux initiaux avaient été réalisés en Python par Samuel Charron. Il avait réalisé un plugin récupérant les signaux, construisant un classifieur naïf bayésien multinomiale avec et permettant de classifier de nouveaux signaux. cependant les performances n'étaient pas suffisantes. N'étant pas formé au Python, j'ai préféré commencer mes travaux en utilisant le Java avec l'accord de Samuel. Je savais en m'orientant vers le Java, qu'une fois que l'application obtiendrait de bonnes performance, j'aurais à implémenter son fonctionnement général en Python sous forme de plugin.

3.1. Démarche de travail

Ce projet s'inscrit parfaitement dans le type de projet R&D. De ce fait, l'avancement est très difficile à planifier dans le temps. Surtout lorsque l'on ne connaît pas les différentes notions sous-jacentes au projet et qu'il y a une bonne part d'auto-formation avant de pouvoir développer une application.

3.1.1. Mes acquis à l'INSA

Les connaissances générales que j'avais en *Data Science*, avant le début du stage, concernaient le *Data Mining* en contexte **numérique** et étaient les suivantes :

- Concepts en analyse et normalisation de données : Analyse en Composantes Principales (ACP), centrage et réduction de données numériques ;
- Concepts d'apprentissage non-supervisé : méthodes de regroupement des données (Clustering : Classification Hiérarchique Ascendante, Algorithme des K-Means, Modèles de mélanges et Algorithmes EM) ;
- Base de l'optimisation : méthodes du gradient et de Newton, introduction aux outils mathématiques pour l'optimisation sous contraintes convexe ;
- Concepts d'apprentissage supervisé : méthodes pour la discrimination de données (Décision Bayésienne, Régression logistique, SVM linéaire) et notions de validation croisée.

Ce projet ne permet pas de mettre mes connaissances en apprentissage non-supervisé en avant. Cependant, mes notions d'apprentissage supervisé telles que : la démarche à suivre pour construire un classifieur, les concepts liées à la validation des performances (validation croisée) et la notion de sur-apprentissage ; ont été fort utiles.

D'une manière générale, mes connaissances en *Text Mining* n'étaient pas suffisamment étoffées pour pouvoir dire tel classifieur est plus performant qu'un autre dans tel contexte (binaire ou multi-classe). En effet, ma formation (à l'INSA) est axée manipulation et traitement de données en contexte **numérique**. De ce fait, la manipulation et le traitement de données textuelles m'étaient inconnus.

Une formation en *Text Mining* m'a donc été indispensable avant de pouvoir commencer à travailler.

3.1.2. Déroulement du stage

Ainsi, durant les deux premiers mois, j'ai exploré le domaine du *Text Mining* et du *Natural Language Processing* au travers de la bibliothèque de Stanford implémentée en Java (*Stanford Natural Language Processing*). Conjointement, j'ai étudié les cours associés, et construit une première application Spring répondant aux contraintes évoquées en partie 2.2 (sauf le critère du langage). Le travail en ressortant est décrit en partie 3.2.

Ensuite, lors du dernier mois, j'ai implémenté le comportement général de cette application sous la forme d'un plugin Python, visible en partie ???. Certains composants n'existaient pas en Python, je les ai donc ré-implémentés.

3.2. Travaux réalisés en Java, le *Text Mining* et le *Natural Language Processing* avec la bibliothèque de Stanford

3.2.1. Présentation de mon environnement de travail

Dans C-Radar tous les traitements de type « computing » (calcul) sont réalisés en Python (cf partie 1.2.2). De ce fait, créer une application permettant de classer les signaux, devrait être fait en Python sous la forme d'un plugin (comme le requiert ma mission). Ayant fait le choix de commencer mes recherches en Java, il a fallu m'initialiser un environnement de travail un peu différent de l'environnement de travail Python.

Spring et MongoDB : Ainsi, Loïc Petit m'a créé une application Spring de base permettant de me connecter en local à une base de données Mongo. L'application me fournit un environnement de travail dans lequel construire le classer à partir des signaux validés. Les signaux eux-mêmes récupérés depuis une base de données Mongo. Cette base de données contient mon ensemble de signaux permettant de construire et tester mon classer. Les 1426 signaux validés y sont stockés ainsi que 350.000 autres signaux non validés (voir le dernier paragraphe de la partie 2.3).

Voici comment les signaux sont stockés dans Mongo :

```
{
  "id" : "TWITTER:agencenetdesign:329129810423083009",
  "content" : "Salon eCom Genève : l'équipe ND est en place au stand E1 \ :) #ecomSITB",
  "publicationDate" : ISODate(2013-04-30T07:07:16Z),
  "sourceId" : "TWITTER:agencenetdesign",
  "source" : {
    "type" : "TWITTER",
    "resourceId" : "agencenetdesign"
  },
  "externalSignalId" : 329129810423083009,
  "validated" : false,
  "validatedTags" : [ ],
  "tags" : [
    "EVENT"
  ],
  "url" : "http://twitter.com/agencenetdesign/status/329129810423083009"
}
```


Le signal comporte :

- un identifieur unique *id* ;
- un contenu *content* ;
- une date de publication *publicationDate* ;
- l'identifieur de la source *sourceld* ;
- la source *source* composé :
 - du type de réseau dont provient le signal *type* ;
 - de l'identifieur du publieur dans ce réseau *resourceld* ;
- un identifieur externe *externalSignalId* ;
- un booléen spécifiant si le signal a été manuellement validé ou non *validated* ;
- la liste des tags s'il a été validé *validatedTags*, la liste des tags potentiels (trouvés par le classifieur) *tags* ;
- l'url là où a été publié le signal *url*.

Formation Spring et Mongo : Je me suis rapidement formé à Spring et Mongo pour pouvoir interfacier ces deux composants ensemble. Pour cela, j'ai suivi les tutoriels de Spring disponibles sur <https://spring.io/guides/gs/accessing-data-mongodb/>.

Grâce aux tutoriels, j'ai appris à créer mes premiers points d'API permettant de faire des requêtes dans Mongo. Ces requêtes sont relativement basiques : récupérer tout les signaux sous forme de liste, récupérer uniquement les signaux validés (également sous forme de liste), savoir combien de signaux sont stockés dans ma base, etc. J'ai appris à faire ce type de requête dans Mongo grâce à la documentation sur <https://docs.mongodb.org/manual/>. Les concepts de base de Mongo ne sont pas très compliqués à comprendre quand on a des notions de base de données.

Dès que j'ai été capable de faire ce type de requêtes auprès de ma base de données, il fallait à présent me concentrer sur le traitement des signaux, et donc la construction du classifieur. C'est à ce moment que je me suis intéressé à la bibliothèque de Stanford.

3.2.2. La bibliothèque : Stanford Natural Language Processing

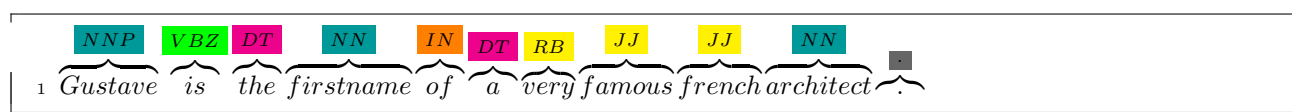
Samuel Charron avait connaissance de l'existence de cette bibliothèque. De ce fait, il m'a conseillé de me former en *Text Mining* et en *Natural Language Processing* au travers de celle-ci puisqu'elle est implémentée en Java. Je me suis donc plongé dedans afin de découvrir les diverses fonctionnalités qu'elle propose. Celle-ci propose un ensemble d'outils pour le traitement automatique du langage naturel de la langue anglaise, chinoise et espagnole. Voici ces différents outils.

Stanford POS Tagger (Part-Of-Speech) :

Le *POS Tagger* permet de savoir la fonction grammaticale de chaque mot d'une phrase. Celui-ci fonctionne aussi pour le français.

Exemple : Entrée « Gustave is the firstname of a very famous french architect. »

Sortie :



NNP signifie qu'il s'agit d'un nom propre singulier (*noun, proper, singular*), **VBZ** signifie qu'il s'agit d'un verbe à la 3ème personne du singulier au présent (*verb, present tense*,

3rd person singular), **DT** signifie qu'il s'agit d'un déterminant (*determiner*), **NN** signifie qu'il s'agit d'un nom commun singulier (*noun, common, singular*), **IN** signifie qu'il s'agit d'une préposition ou d'une conjonction de subordination (*preposition or conjunction, subordinating*), **RB** signifie qu'il s'agit d'un adverbe (*adverb*) et **JJ** signifie qu'il s'agit d'un adjectif (*adjective*).

Stanford Parser :

Le **Parser** permet de connaître la structure grammaticale d'une phrase, à savoir quel(s) groupe(s) de mots forme(nt) le sujet, quel(s) groupe(s) de mots forme(nt) le verbe et quel(s) groupe(s) de mots forme(nt) le complément. Cet outils est une sur-couche du **POS Tagger**. En effet, il réutilise, entre autres, son résultat pour en déduire la structure grammaticale d'une phrase.

Exemple : Entrée « My internship was a rewarding experience. »

Sortie :

```
1 (ROOT
2   (S
3     (NP (PRP$ My) (NN internship))
4     (VP (VBD was)
5       (NP (DT a) (JJ rewarding) (NN experience))
6     (. .)))
```

Cet outils a pour vocation d'identifier les groupes de mots, ainsi que leurs dépendances à d'autres groupes de mots.

Stanford Named Entity Recognizer :

Le **Named Entity Recognizer** permet d'identifier les groupes de mots qui sont des noms de personne, d'entreprises, de gènes, etc.

Exemple : Entrée « François Bancilhon is the CEO of Data Publica, a Startup located in Paris. »

Sortie :

```
1 François Bancilhon is the CEO of Data Publica, a Startup located in Paris.
```

Les mots surlignés en magenta comme **François Bancilhon** sont potentiellement des noms de personnes (**PERSON**). Ceux surlignés en orange comme **Data Publica** sont potentiellement des noms d'organisations (**ORGANIZATION**). Enfin, les mots surlignés en violet comme **Paris** sont potentiellement des noms de lieux (**LOCATION**).

Stanford Classifier :

Le **Classifier** permet de construire un classifieur automatique pour la catégorisation de texte. Un classifieur est un outils d'apprentissage automatique qui prend des données et les classifie dans k classes. Un classifieur probabiliste peut aussi donner la distribution des probabilités de chaque classes pour une donnée. L'implémentation Java de Stanford est un classifieur **maximum entropy**. Les modèles **maximum entropy** sont aussi connu sous le noms de **softmax classifieurs** et sont équivalent aux modèles de régression logistique multi-classe (avec des paramètres différents, dans le sens où ce modèle est avantageux en cas de données sparse). Cet outils sera détaillé d'avantage par la suite.

Stanford CoreNLP :

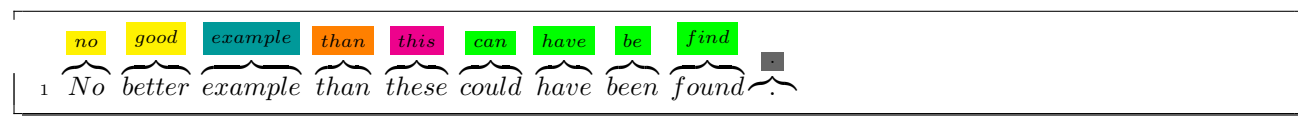
Le *CoreNLP* permet de construire une suite de traitements automatiques (les traitements précédents) sur de l'anglais, du chinois, de l'espagnol. Un exemple de chaîne de traitement est visible en figure 3.1. La particularité de cet outils par rapport aux précédents, est qu'il permet de réutiliser les traitements précédents les uns à la suite des autres (sauf le *Stanford Classifier*). De plus, il permet aussi d'appliquer des traitements supplémentaires à ceux énoncés jusque là. En effet, celui-ci réutilise certain des traitements précédents pour faire de la recherche morphologique telle que la lemmatisation ou du stemming. Le *POS Tagger* et le *Tokenizer* sont notamment réutilisés.

Lemmatisation :

La lemmatisation est le traitement consistant à trouver les lemmes de chaque mots d'une phrase. Ce traitement est détaillé en partie ???. Pour cela, ce traitement nécessite que la phrase soit préalablement découpée en token (les mots de la phrase) et que chacun d'entre eux soit tagué par le *POS Tagger*. Ainsi, en couplant la fonction grammaticale d'un mot avec un dictionnaire, il est possible de retrouver le lemme du mot.

Exemple : Entrée « No better example than these could have been found. »

Sortie :

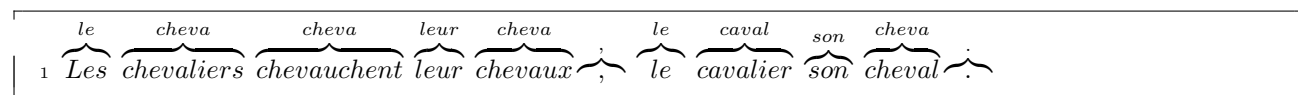


Stemming :

Le stemming est un traitement assez similaire à la lemmatisation. En effet, celui-ci consiste à trouver les stems de chaque mots d'une phrase. Ce traitement est détaillé en partie ??.

Exemple : Entrée « Les chevaliers chevauchent leur chevaux, le cavalier son cheval. »

Sortie :



Premier bilan sur la bibliothèque :

Ces traitements n'ont pas tous un intérêt pour moi dans la construction de mon classifieur. En effet, dans un premier temps, mon objectif est seulement de réutiliser la partie *Stanford Classifier* de la bibliothèque pour créer un classifieur binaire permettant de classifier les signaux relatifs aux offres d'emploi et de stage (soit le tag *JOB*).

Cependant, il est certain que des traitement comme la lemmatisation ou le stemming me seront utiles par la suite, lorsqu'il faudra normaliser mes données.

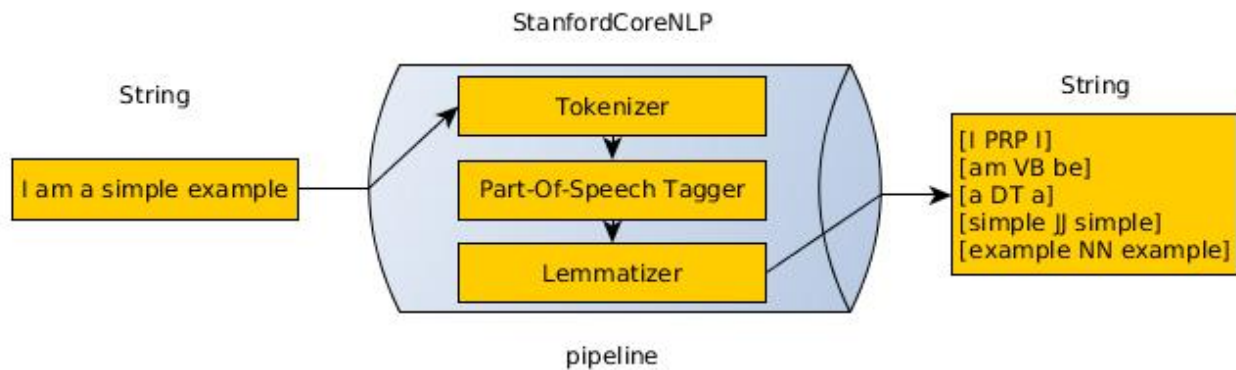


Figure 3.1. – Trois traitements ont été assignés au pipeline (entité de la classe StanfordCoreNLP) : Tokenization, POS-Tagging et Lemmatization. L’objet en sortie est une chaîne de caractère présentant le résultat de chaque traitement en colonne.

3.2.3. Première mise en pratique de la bibliothèque de Stanford

Ma première application Spring :

J’ai construit une application réalisant les actions suivantes (visible en figure 3.2 et expliquée ci-après) :

- Récupérer les signaux stockés dans Mongo sous forme de liste ;
- Créer un ensemble de données à partir des signaux validés manuellement ;
- Diviser aléatoirement cet ensemble de données en deux ensembles (un pour entraîner le classifieur et un pour le tester) tout en gardant la proportion de chaque classe dans les deux ensembles ;
- Entraîner un classifieur binaire naïf bayésien ;
- Fixer les hyper-paramètres du classifieur par validation croisée pendant la phase d’apprentissage ;
- Évaluer la qualité du classifieur construit (l’erreur de généralisation) en calculant sa précision et son rappel sur un ensemble de données de test (qui n’ont pas « été vu » jusqu’à ici par le classifieur).

Le modèle de classifieur :

La première question, à laquelle j’ai dû répondre pour pouvoir créer un premier classifieur binaire sur la classe **JOB**, est la suivante : Quel type de classifieur construire ? Un SVM, une régression logistique, un modèle naïf bayésien, etc.

Dans la littérature de la classification de texte, comme la détection de spam dans les emails ou l’analyse des sentiments (savoir si un texte est critique ou élogieux), il est plutôt commun de construire des classifieurs naïf bayésien avec comme caractéristiques la fréquence des mots. Ainsi, j’ai choisi de construire un tel classifieur pour catégoriser mes signaux. (C’est également ce qu’avait fait Samuel Charron en Python.)

Construction de l’ensemble de données :

Ensuite, s’est posée la question de comment utiliser les données labellisées pour construire un ensemble de données pour l’apprentissage et la validation.

Sur ce point, j’ai choisi de construire mon ensemble de données selon le modèle du sac de

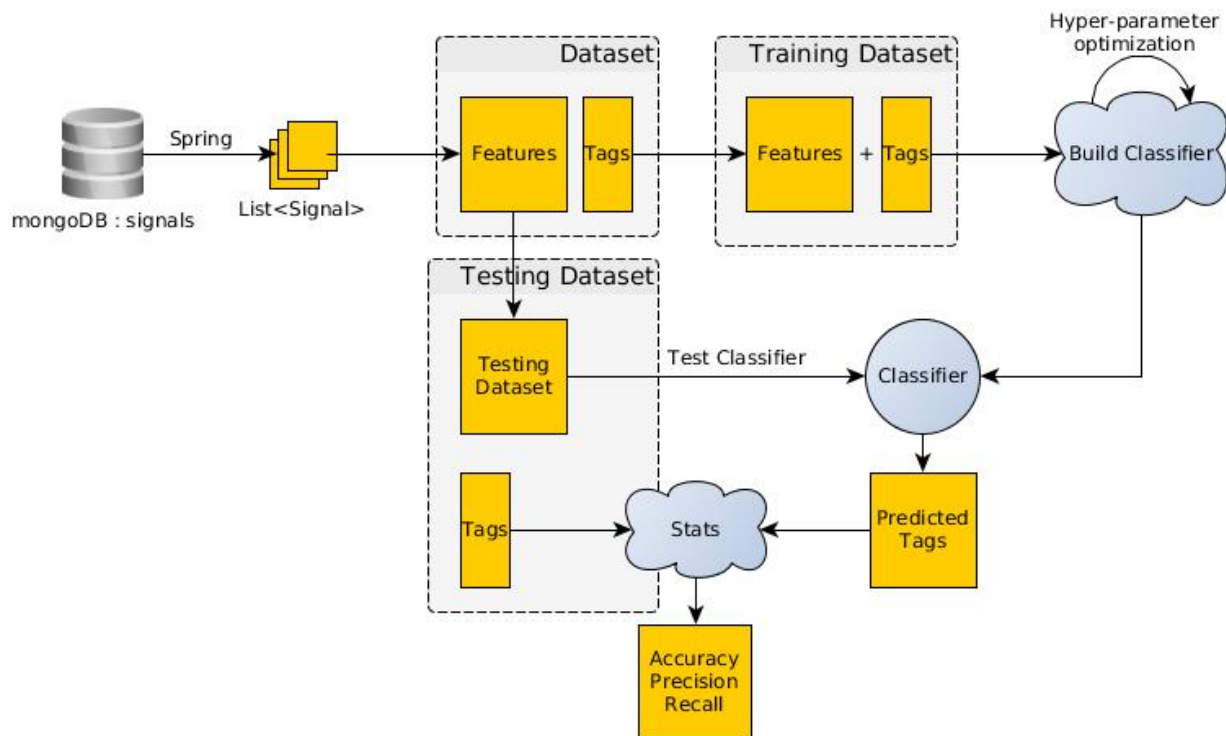


Figure 3.2. – La construction du classifieur.

mots (*bag of words*). Dans ce modèle, un texte (une phrase ou un document) est représenté comme un sac (*bag*), un ensemble (au sens mathématique) de ses mots, sans se préoccuper de la grammaire ou de l'ordre des mots, mais en gardant la multiplicité. Ce modèle est communément utilisé en classification de document, quand la fréquence, l'occurrence des mots est utilisé comme caractéristique, attribut. Ce qui est le cas du modèle naïf bayésien. Ainsi, un signal est caractérisé par la liste des occurrences des mots formant son titre et son contenu.

Exemple : Voici deux signaux que l'on va modéliser à l'aide du sac de mots :

- ¹ *Offre d'emploi : Ingénieur d'études et développement Java.*
- ² *Offre de stage : Classification de signaux entreprises Java ou Python.*

À partir de ces deux signaux, une liste de mot est construite comme suit :

```
[ "Offre", "d'", "emploi", ":", "Ingénieur", "études", "et", "développement",
  "Java", ".", "de", "stage", "Classification", "signaux", "entreprises",
  "ou", "Python" ]
```

Celle-ci contient 17 mots distincts. En utilisant son index, on peut représenter chaque signal comme un vecteur de taille 17 :

- ¹ [1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
- ² [1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1]

Chaque ième composante du vecteur représente le nombre de fois que le ième mot de la liste est présent dans le signal. Par exemple, dans le premier vecteur (qui représente le premier signal), les deux premières composantes sont « 1, 2 ». La première composante correspond au mot « Offre » qui est le premier mot de la liste, et sa valeur est « 1 » car « Offre » est présent qu'une fois dans le premier signal. De la même façon, la deuxième composante correspond au

mot « d' » qui est le deuxième mot de la liste et sa valeur est « 2 » car il est présent deux fois dans le signal. Cette représentation vectorielle ne préserve pas l'ordre des mots originel.

Pour mon premier classifieur binaire naïf bayésien, l'ensemble de données a été construit suivant ce modèle, sur la base des 1426 signaux validés (présentés dans le dernier paragraphe de la partie 2.3), en considérant seulement les 123 signaux validés **JOB** comme intéressant.

Enfin, aucun pré-traitement des données n'était mis en place, hors mis le fait de garder les mots dans le sac de mots final apparaissant un minimum de cinq fois. Ceci afin de supprimer les mots rares qui n'apportent pas d'informations lorsqu'on catégorise les signaux de type **JOB** : les noms propres, les mots mal orthographiés ou les mots très spécifiques.

Construction du classifieur :

Une fois le problème de la construction de mon ensemble de données résolu, j'ai construit mon classifieur binaire naïf bayésien. Pour cela, j'ai divisé mon ensemble de données en deux ensemble de données dans les proportions suivantes :

- 75% de l'ensemble de données pour l'ensemble d'apprentissage ;
- 25% de l'ensemble de données pour l'ensemble de test.

À noter que la méthode de la bibliothèque, permettant de diviser l'ensemble de données en deux ensembles, ne garantis pas que la proportion des classes soit conservée dans les deux ensembles résultants. De plus, l'ensemble n'est pas mélangé à chaque nouvelle construction du classifieur, il y donc un risque de sur-apprentissage. Ces deux points, je ne les avais pas pris en compte directement car cela n'était pas explicitement expliqué dans la documentation de la bibliothèque. En effet, il a fallu que je m'aperçoive que les ensembles de données étaient toujours les mêmes en phase d'apprentissage et de test pour corriger ce point. Ainsi, j'ai du lire de plus près le code source de la méthode pour ré-implémenter le bon comportement.

Optimisation des hyper-paramètres par validation croisée :

Pour cette étape, une méthode de la bibliothèque se charge d'optimiser l'hyper-paramètre de mon classifieur naïf bayésien par validation croisée. Mon classifieur ayant été construit sur la base de l'ensemble d'apprentissage. Je l'ai donc utilisé sans aller plus loin dans les détails car la documentation n'en disait pas plus sur celui-ci et les performances en étaient bonifiées.

Évaluation de la qualité du classifieur :

Enfin pour évaluer l'erreur de généralisation de mon classifieur, je les testais sur un ensemble de test, que le classifieur n'avait pas « vu » jusque ici. J'ai calculé la précision et le rappel obtenu par la classe **JOB** et par la classe **USELESS** représentant le reste n'appartenant pas à **JOB**. Ceux-ci sont visible en table 3.1.

	JOB	USELESS
Précision	0,84	0,99
Rappel	0,91	0,98

Table 3.1. – Performances du classifieurs.

Remarques : Il est important de noter que lorsqu'on fait ce type de classification, on met l'accent sur le fait d'avoir une précision très élevée quitte à avoir un rappel un peu diminué, car quand l'un augmente l'autre diminue et vice versa.

En effet, la précision mesure le nombre de fois où on a bien classifié un document. Alors que le rappel mesure le fait qu'on ait bien trouvé tout les documents d'une classe. Ainsi, on préfère se tromper très rarement dans notre classification (précision élevée) quitte à rater des signaux que l'on jugerait inutile alors que ça n'était pas le cas (rappel moyen). Ce qui est logique : on préfère qu'un utilisateur voit moins de signaux correctement classifié plutôt qu'il voit beaucoup de signaux plus ou moins correctement classifié.

Premier bilan :

Le classifieur construit est prometteur mais il ne s'agit que d'un classifieur binaire. La généralisation à plusieurs classes, va amener à faire de nouveau choix : Faut-il adopter une stratégie de type :

- One versus all ;
- One versus One.

D'autres questions peuvent être soulever comme :

Est-ce que le modèle naïf bayésien va bien se généraliser en multi-classe ?

Un prétraitement global n'est-il pas nécessaire ?

Un prétraitement spécifique à chaque classe n'est-il pas nécessaire ?

3.2.4. Amélioration de ma première application et prétraitement

Agrandissement de mon ensemble de signaux validés :

Avant de pouvoir passer à un classifieur multi-classe, il fallait que mon ensemble de signaux validés grandissent. En effet, celui-ci souffre d'un fort déséquilibre entre les classes : il y a beaucoup plus de signaux inintéressants qu'intéressants. Je n'avais donc pas assez d'exemples de signaux intéressants pour pouvoir considérer les éventuelles prédictions d'un classifieur construit sur la base de ces données comme correctes. Il a donc été nécessaire d'en valider d'autres manuellement.

Le QA ou Quality Assessment : L'objectif du QA est de demander la contribution d'un maximum de personnes sur une tâche de validation manuelle pénible.

Durant mon stage, j'ai organisé plusieurs QA pour approfondir l'ensemble des signaux d'apprentissage et de test.

Proportion de signaux intéressants : La quantité de signaux n'ayant pas d'intérêt est énorme (plus de 80% sur environ 300.000). De ce fait, une pré-sélection des signaux à valider est nécessaire. Pour cela, j'ai réutilisé le premier classifieur implémenté en Python de Samuel Charron, construit sur la base des 1426 signaux. Ce classifieur a permis de classer des signaux non validés. Ce sont ces signaux classifiés par le classifieur Python qui ont été sélectionnés pour être validés manuellement. Notamment ceux appartenant potentiellement aux catégories EVENT, JOB et PRODUCT.

Pour ceux appartenant aux catégories MONEY et PEOPLE, ils ont été sélectionnés pour être validés à l'aide d'expressions rationnelles pour faire ressortir des termes tels que « levée de fonds », « chiffre d'affaire », « nommer », « nomination », etc.

De cette manière 2574 nouveaux signaux ont été validés manuellement. J'ai conscience que ce type de méthodes biaise la dispersion des classes de signaux intéressantes par rapport à celle sans intérêt, mais autrement il mettait impossible d'obtenir plus d'exemple.

Au 27.07.2015, il y avait donc 4000 signaux validés manuellement par un humain :

- 488 catégorisés EVENT soit 12,2%
- 258 catégorisés JOB soit 6,4%
- 118 catégorisés MONEY soit 3%
- 83 catégorisés PRODUCT soit 2,1%
- 49 catégorisés PEOPLE soit 1,3%
- 3004 validés mais considérés comme inintéressant soit 75%

Le passage au multi-classe :

Pour le passage au multi-classe, la bibliothèque de Stanford ne propose pas de stratégie *One vs All* ou *One vs One*. Je ne sais donc pas laquelle des deux est choisie. En ce qui est du modèle de classifieur, deux options s'offraient à moi :

- Un modèle génératif et notamment un classifieur naïf bayésien ;
- Un modèle discriminatif et notamment un classifieur qui maximise l'entropie.

Le modèle génératif maximise la vraisemblance de la probabilité jointe $P(\text{classe}, \text{donnée})$ alors que le modèle discriminatif maximise la vraisemblance de la probabilité conditionnelle $P(\text{classe}|\text{donnée})$. Jusque là, j'avais opté pour le modèle bayésien car c'est ce qui ressortait le plus souvent dans la littérature. J'ai donc repris mon application précédente et j'ai construit un classifieur naïf bayésien multinomial sur la base de mes 4000 signaux taggés, en employant la même méthode de construction de mon ensemble de données. Les performances obtenues (visibles en table 3.2) avaient grandement baissées (la classe *PRODUCT* n'y figure pas car elle n'était pas suffisamment représentée).

	JOB	EVENT	PEOPLE	MONEY
Précision	0,63	0,51	0,46	0,44
Rappel	0,91	0,88	0,63	0,87

Table 3.2. – Performances du classifieur naïf bayésien multinomial.

Le modèle discriminatif maximisant l'entropie

Par curiosité, j'ai changé le modèle pour le modèle maximisant l'entropie car je ne connaissais pas et je voulais voir ce que ça donnerait. À ma grande surprise, le modèle discriminatif maximisant l'entropie obtint de bien meilleures performances que le modèle bayésien dans les mêmes conditions (construction de l'ensemble de données identique, prétraitements identiques, etc). Les performances sont visibles en table 3.3.

	JOB	EVENT	PEOPLE	MONEY
Précision	0,96	0,82	0,73	0,81
Rappel	0,73	0,60	0,50	0,49

Table 3.3. – Performances du classifieur maximisant l'entropie.

La régression logistique binomiale : Elle consiste à prédire une valeur parmi deux valeurs possibles (vrai ou faux), telle que :
 $\text{logit}(p(x = \text{vrai})) = \text{une combinaison linéaire d'un vecteur de poids et d'un vecteur de traits}$;
Cela correspond à une classification binaire qui maximise le log de vraisemblance

Le *MaxEnt* ou *Maximum Entropy* : Il s'agit de la généralisation de la régression logistique au cas multinomial. Dans ce cas, maximiser la vraisemblance revient à maximiser l'entropie. Il s'agit ni plus ni moins que du passage du cas binaire au cas N classes.

Les cours de l'université de Stanford :

Afin de perfectionner mon application, j'ai lu les cours de l'université de Stanford concernant le *Natural Language Processing*, accessibles librement sur Internet. Ces cours proviennent du livre *Introduction to Information Retrieval*.¹

En parallèle, j'ai visionné sur coursera les vidéos que cette même université avait diffusé suite à un MOOC sur le *Natural Language Processing*. Grâce à ces cours, j'ai pris conscience de toute l'importance du travail de prétraitement nécessaire à mettre en place, afin de bien normaliser et formater les données textuelles avant d'en faire quelque chose.

Cette remarque prend tout son sens quand les données manipulées en plus d'être textuelles, ne sont pas normalisées (ou pas structurées).

Durant ma formation au *Natural Language Processing*, j'ai également lu les livres *Natural Language Processing with Python*² et *Python 3 Text Processing with NLTK 3 Cookbook*³, ainsi que les pages internet *Introduction to Information Retrieval*.⁴

3.2.5. Les travaux de prétraitement :

1. Hinrich Schütze Christopher D. Manning Prabhakar Raghavan. Information Retrieval. url : <http://nlp.stanford.edu/IR-book/html/htmledition/contents-1.html>.

2. Edward Loper Steven Bird Ewan Klein. Natural Language Processing with Python. 2009. isbn : 978-0-596-51649-9.

3. Jacob Perkins. Python 3 Text Processing with NLTK 3 Cookbook. 2010. isbn : 978-1-78216-785-3.

4. Christopher D. Manning, cf. note 1.

4. Conclusion

5. Résumé

Bibliographie

- Christopher D. Manning Prabhakar Raghavan, Hinrich Schütze. Information Retrieval. url : <http://nlp.stanford.edu/IR-book/html/htmledition/contents-1.html>.
- Perkins, Jacob. Python 3 Text Processing with NLTK 3 Cookbook. 2010. isbn : 978-1-78216-785-3.
- Steven Bird Ewan Klein, Edward Loper. Natural Language Processing with Python. 2009. isbn : 978-0-596-51649-9.

A. Annexes

A.1. Titre de l'annexe

Ce stage obligatoire s'effectue en fin de quatrième année. Au cours de ce stage l'étudiant devra mettre en pratique les connaissances acquises au cours de sa formation et devra approfondir son savoir-faire au sein de l'entreprise. Il faudra qu'à la fin de son stage l'étudiant réalise un rapport écrit. La validation du stage dépend de la qualité du travail réalisé, du rapport et de la fiche d'évaluation du tuteur industriel. Ce stage obligatoire s'effectue en fin de quatrième année. Au cours de ce stage l'étudiant devra mettre en pratique les connaissances acquises au cours de sa formation et devra approfondir son savoir-faire au sein de l'entreprise. Il faudra qu'à la fin de son stage l'étudiant réalise un rapport écrit. La validation du stage dépend de la qualité du travail réalisé, du rapport et de la fiche d'évaluation du tuteur industriel. Ce stage obligatoire s'effectue en fin de quatrième année. Au cours de ce stage l'étudiant devra mettre en pratique les connaissances acquises au cours de sa formation et devra approfondir son savoir-faire au sein de l'entreprise. Il faudra qu'à la fin de son stage l'étudiant réalise un rapport écrit. La validation du stage dépend de la qualité du travail réalisé, du rapport et de la fiche d'évaluation du tuteur industriel. Ce stage obligatoire s'effectue en fin de quatrième année. Au cours de ce stage l'étudiant devra mettre en pratique les connaissances acquises au cours de sa formation et devra approfondir son savoir-faire au sein de l'entreprise. Il faudra qu'à la fin de son stage l'étudiant réalise un rapport écrit. La validation du stage dépend de la qualité du travail réalisé, du rapport et de la fiche d'évaluation du tuteur industriel. Ce stage obligatoire s'effectue en fin de quatrième année. Au cours de ce stage l'étudiant devra mettre en pratique les connaissances acquises au cours de sa formation et devra approfondir son savoir-faire au sein de l'entreprise. Il faudra qu'à la fin de son stage l'étudiant réalise un rapport écrit. La validation du stage dépend de la qualité du travail réalisé, du rapport et de la fiche d'évaluation du tuteur industriel. Ce stage obligatoire s'effectue en fin de quatrième année.

Ce stage obligatoire s'effectue en fin de quatrième année. Au cours de ce stage l'étudiant devra mettre en pratique les connaissances acquises au cours de sa formation et devra approfondir son savoir-faire au sein de l'entreprise. Il faudra qu'à la fin de son stage l'étudiant réalise un rapport écrit. La validation du stage dépend de la qualité du travail réalisé, du rapport et de la fiche d'évaluation du tuteur industriel. Ce stage obligatoire s'effectue en fin de quatrième année. Au cours de ce stage l'étudiant devra mettre en pratique les connaissances acquises au cours de sa formation et devra approfondir son savoir-faire au sein de l'entreprise. Il faudra qu'à la fin de son stage l'étudiant réalise un rapport écrit. La validation du stage dépend de la qualité du travail réalisé, du rapport et de la fiche d'évaluation du tuteur industriel. Ce stage obligatoire s'effectue en fin de quatrième année. Au cours de ce stage l'étudiant devra mettre en pratique les connaissances acquises au cours de sa formation et devra approfondir son savoir-faire au sein de l'entreprise. Il faudra qu'à la fin de son stage l'étudiant réalise un rapport écrit. La validation du stage dépend de la qualité du travail réalisé, du rapport et de la fiche d'évaluation du tuteur industriel. Ce stage obligatoire s'effectue en fin de quatrième année. Au cours de ce stage l'étudiant devra mettre en pratique les connaissances acquises au cours de sa formation et devra approfondir son savoir-faire au sein de l'entreprise. Il faudra qu'à la fin de son stage l'étudiant réalise un rapport écrit. La validation du stage dépend de la qualité du travail réalisé, du rapport et de la fiche d'évaluation du tuteur industriel. Ce stage obligatoire s'effectue en fin de quatrième année. Au cours de ce stage l'étudiant devra mettre en pratique les connaissances acquises au cours de sa formation et devra approfondir son savoir-faire au sein de l'entreprise. Il faudra qu'à la fin de son stage l'étudiant réalise un rapport écrit. La validation du stage dépend de la qualité du travail réalisé, du rapport et de la fiche d'évaluation du tuteur industriel. Ce stage obligatoire s'effectue en fin de quatrième année.

INSA Rouen
Campus du Madrillet
685 avenue de l'Université – BP 08
76801 SAINT-ÉTIENNE-DU-ROUVRAY cedex
www.insa-rouen.fr



RENSEIGNEMENTS
Département ASI
02 32 95 97 79
asi@insa-rouen.fr

Membre de



Normandie Université

Financeurs institutionnels



MINISTÈRE
DE L'ÉDUCATION NATIONALE,
DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE

