INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Baptiste O'Jeanson

Rapport de stage de spécialité

Image : http://www.data-publica.com/

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 Bancilhon [1], directeur général de Data Publica, et Christian Frisch [2], 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 encadré, suivi et conseillé durant mon stage, et qui ont enrichi mes connaissances du monde professionnel ainsi que mes connaissances en informatiques.

J'aimerais aussi remercier tout particulièrement Thomas Dudouet, qui a fait suite à ma candidature spontanée, et qui m'a donc offert cette chance de pouvoir m'entretenir avec Clément Chastagnol puis Christian Frisch.

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. Francois Bancilhon, présenté ici 1.2
2. Christian Frisch, présenté ici 1.1

Table des matières

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 tous 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 bénéficié 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 une 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 françaises 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 composé 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

(a) Christian F. (b) Thomas D. (c) Guillaume L. (d) Samuel C.

(e) Loïc P. (f) Clément C. (g) Clément D. (h) Vincent Y.

Figure 1.1. – L'équipe technique de Data Publica

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

- François Bancilhon, 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 « scrappé » 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 fond des yeux » [1] afin de voir leur fond mais aussi leur forme.

Le machine learning : Quand les données sont correctement formatées et normalisées, on peut construire des applications capables d'apprendre automatiquement de ces données, de les classifier. 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'ait à s'occuper que de l'application. Il prend en charge énormément de tâches 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.

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.

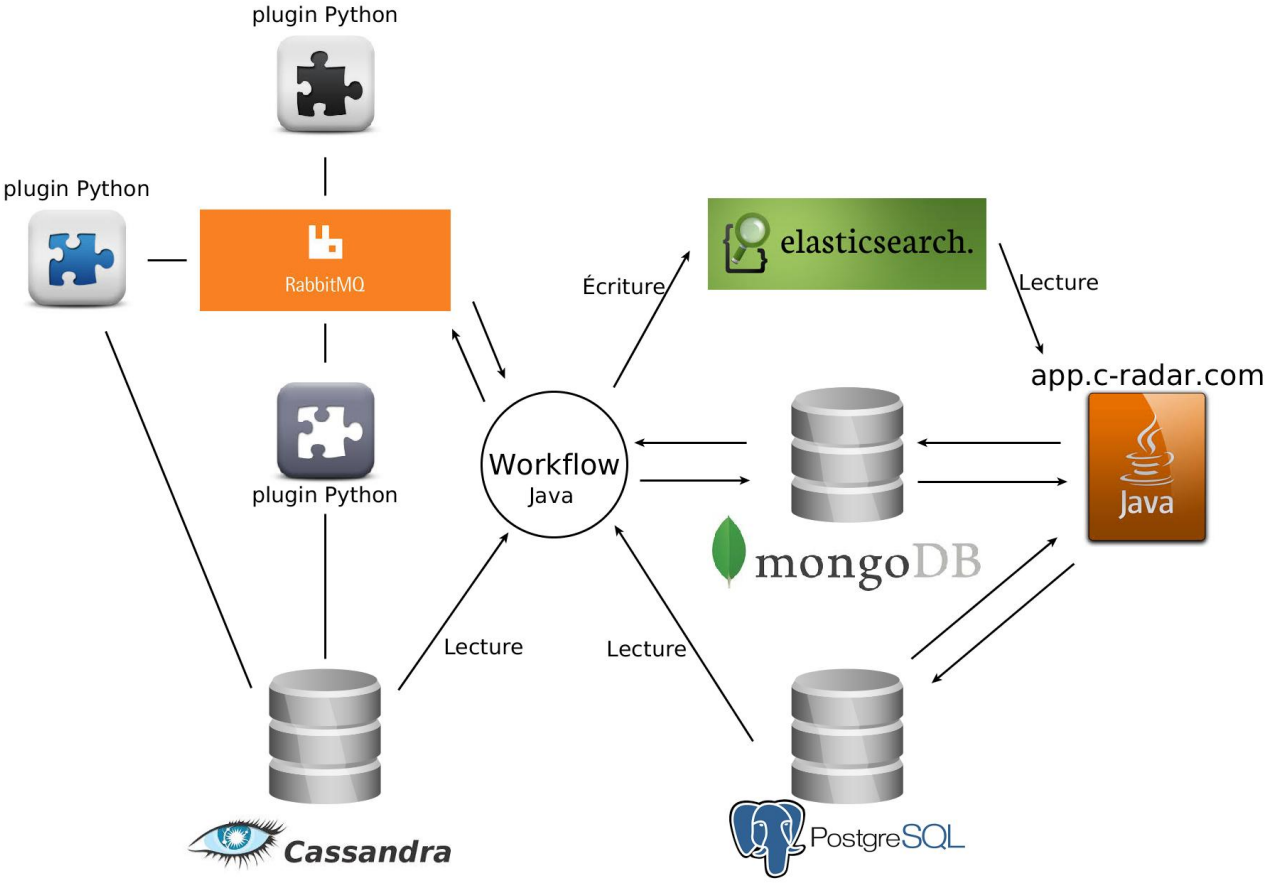


Figure 1.3. – L'architecture générale de C-Radar

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 une newsletter.

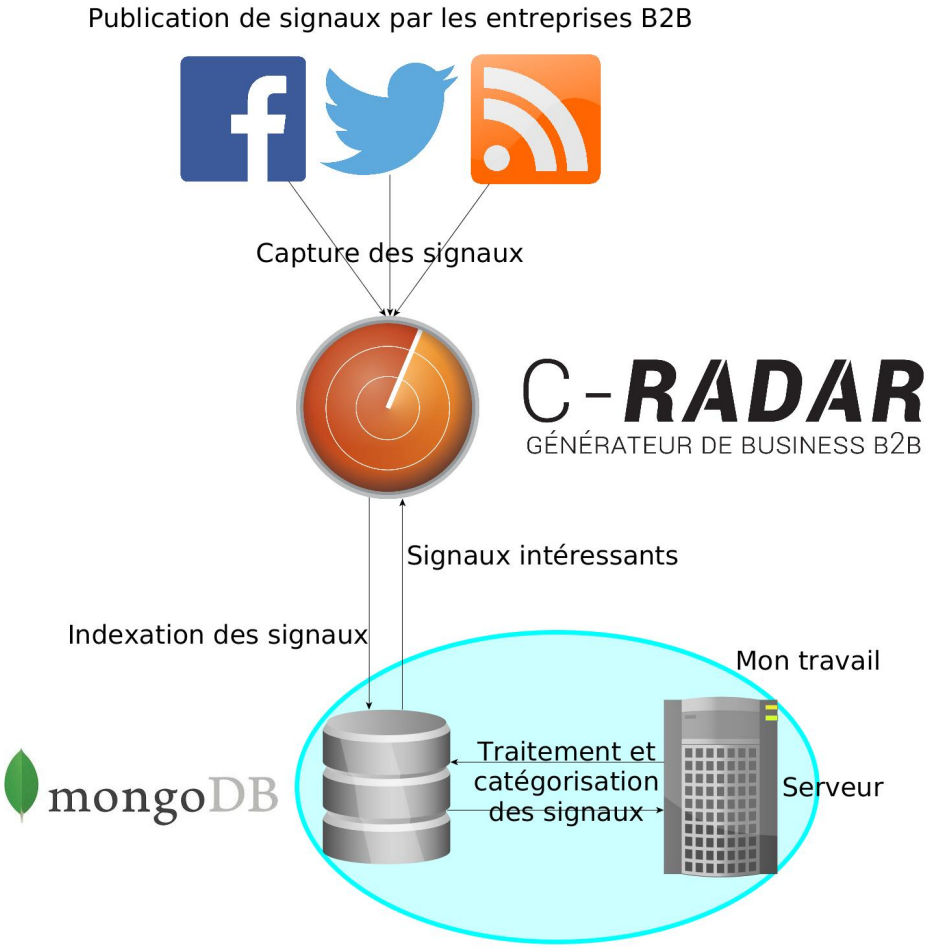


Figure 2.1. – Le fonctionnement général de cette fonctionnalité.

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 bel et bien ordonnée grâce à cette fonctionnalité. On peut désormais savoir quels sont les derniers postes à pourvoir chez Orange, par exemple, ou encore les dernières nominations qui ont eu lieux à la Banque Postale.

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

1. C-Radar capte tous 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 stockés dans une base de données Mongo.
2. Une fois ces signaux capturés et stockés, 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 catégorie respective.

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 porteuses d'information.

2.2.1. Inscription dans le domaine du *Big Data*

Les tâches qui découlent de ma mission, et notamment tous 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*.

Remarque : Un classifier est un outils d'apprentissage automatique qui prend des données et les classifie dans k classes.

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.

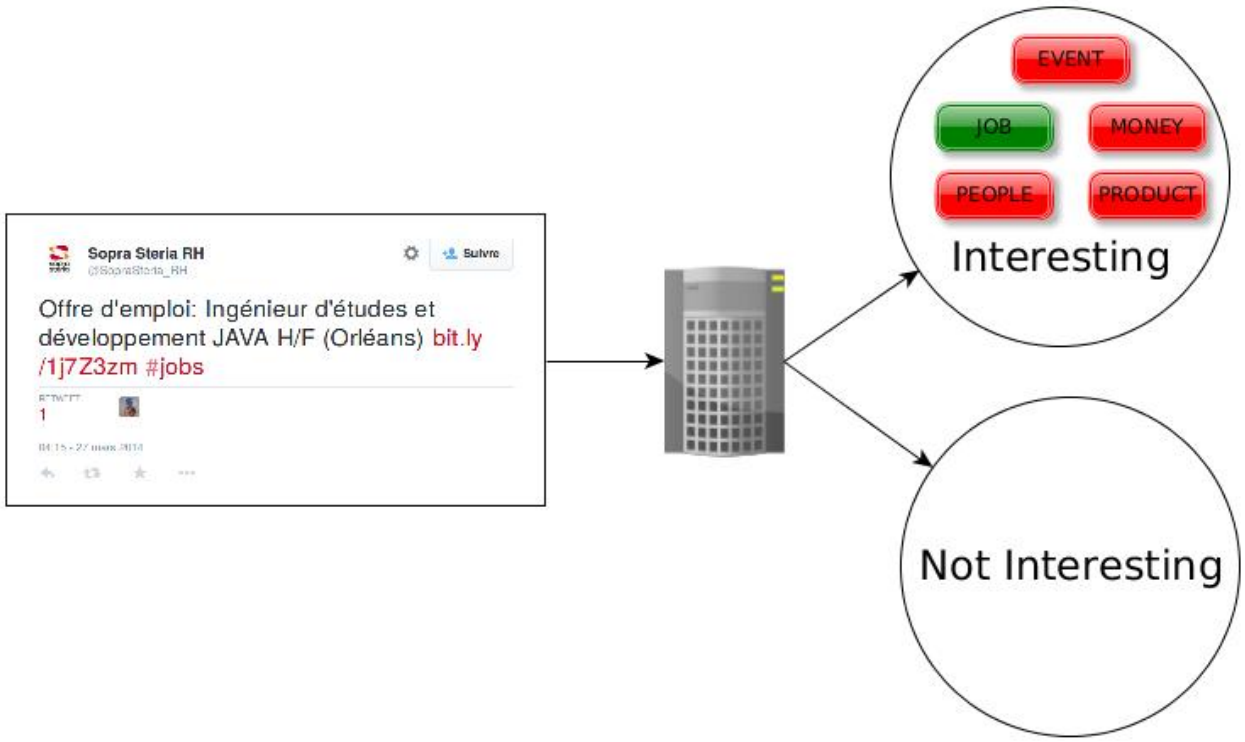


Figure 2.2. – La tâche du plugin Python.

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/LOvN8rLHIe #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 Fraisp ! »
— *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 multinomial 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 pour pouvoir l'intégrer à l'architecture de C-Radar.

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, algorithmes des K-Means et EM) ;
- Base de l'optimisation : méthodes du gradient et de Newton, introduction à 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.

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 **numérique**. De ce fait, une formation en *Text Mining* m'était donc indispensable avant de pouvoir commencer le développement d'une application.

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 3.3. Certain composants n'existaient pas en Python, je les ai donc ré-implémenté.

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 tout les traitements de type « computing » (calcul) sont réalisés en Python (cf partie 1.2.2). De ce fait, créer une application permettant de classifier les signaux, devrait être fait en Python sous la forme d'un plugin. 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 : 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 classifieur à partir des signaux validés récupérés depuis une base de données Mongo. Cette base de données contient mon ensemble de signaux permettant de construire et tester mon classifieur. Les 1426 signaux validés y sont stocké 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 *sourceId* ;
— la source *source* composé :
  — du type de réseau dont provient le signal *type* ;
  — de l'identifieur du publieur dans ce réseau *resourceId* ;
— un identifeur 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 interfacer 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. La documentation de Mongo disponible sur https://docs.mongodb.org/manual/ m'a également bien aidé. 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 me concentrer sur le traitement des signaux, et donc construire un 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 m'a conseillé de me former en *Text Mining* et en *Natural Language Processing* au travers de cette bibliothèque implémentée en Java. Celle-ci propose un ensemble d'outils pour le traitement automatique de la langue anglaise, chinoise et espagnole.

Les cours de l'université de Stanford sur le *Natural Language Processing*

Ces cours accessibles librement sur Internet, m'ont permis de découvrir de nombreux concepts (expliqués ci-après et en partie 3.2.3) :
— Le modèle du *bag of words* ;
— Le modèle *maximum entropy* ;
— Les prétraitements comme la lemmatisation.
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 l'importance de bien choisir son modèle de classifieur. De plus, j'ai également compris l'intérêt

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

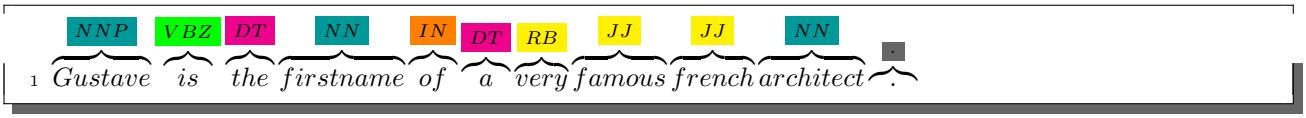
des prétraitements, permettant de normaliser et formater les données textuelles afin d'augmenter les performances et la capacité de généralisation du classifieur.

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 internets *Introduction to Information Retrieval.*[4]

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, le verbe et le complément. Cet outils est une sur-couche du *POS Tagger* puisqu'il réutilise son résultat.

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     (. .)))
```

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.

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.

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 *maximum entropy* et un naïf bayésien sont disponibles. Cet outils propose une classe (*ColumnDataClassifier*) qui permet de construire n'importe quel type de classifieur, simplement en lui fournissant un fichier de configurations, et des données d'apprentissage et de test. Le soucis de cette classe est que les données doivent se conformer à un format bien précis.

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, 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 permet de faire de la recherche morphologique telle que lemmatisation ou stemming. Le *POS Tagger* et le *Tokenizer* sont réutilisés pour ces traitements.

Lemmatisation et stemming :

La lemmatisation consiste à trouver les lemmes de chaque mots d'une phrase. Ce traitement est détaillé en partie 3.2.4. Ce traitement nécessite que la phrase soit préalablement découpée en token (les mots de la phrase) par un *Tokenizer* 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 :

```
1 No(no) better(good) example(example) than(than) these(this) could(can) have(have) been(be) found(find) .(.)
```

Le stemming : Le stemming est un traitement assez similaire à la lemmatisation. Celui-ci consiste à trouver les stems et non les lemmes. Ce traitement est détaillé en partie 3.2.4.

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

```
1 Les(le) chevaliers(cheva) chevauchent(cheva) leur(leur) chevaux(cheva) ,(,) le(le) cavalier(caval) son(son) cheval(cheva) .(.)
```

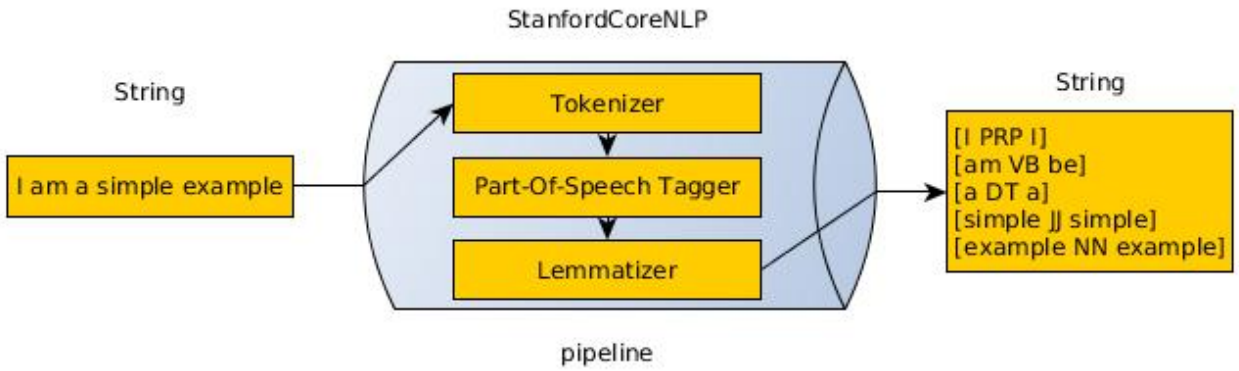


Figure 3.1. – Trois traitements sont assignés au pipeline (entité de la classe StanfordCoreNLP) : Tokenization, POS-Tagging et Lemmatization. En sortie, une chaîne de caractère présente le résultat de chaque traitement en colonne.

Premier bilan sur la bibliothèque

Dans un premier temps, mon objectif est seulement de réutiliser la partie *Stanford Classifier* de la bibliothèque pour créer un classifieur binaire. Celui-ci doit être capable de classifier les signaux relatifs aux offres d'emploi et de stage (soit le tag *JOB*). La classe *ColumnDataClassifier* va mettre d'une grande aide, même si elle exige des données dans un format particulier et qu'elle ne gère pas certain prétraitements.

Des traitements comme la lemmatisation ou le stemming seront utiles, lors de la normalisation des données. Pouvoir utiliser le *StanfordCoreNLP* pour normaliser et sélectionner les features utiles serait l'idéal, avant de les fournir au classifieur pour construire un modèle.

3.2.3. Première mise en pratique de la bibliothèque de Stanford

Première application Spring

J'ai construis 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 classifier 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

Quel type de classifieur construire ? Un SVM ? Une régression logistique ? Un modèle naïf bayésien ?

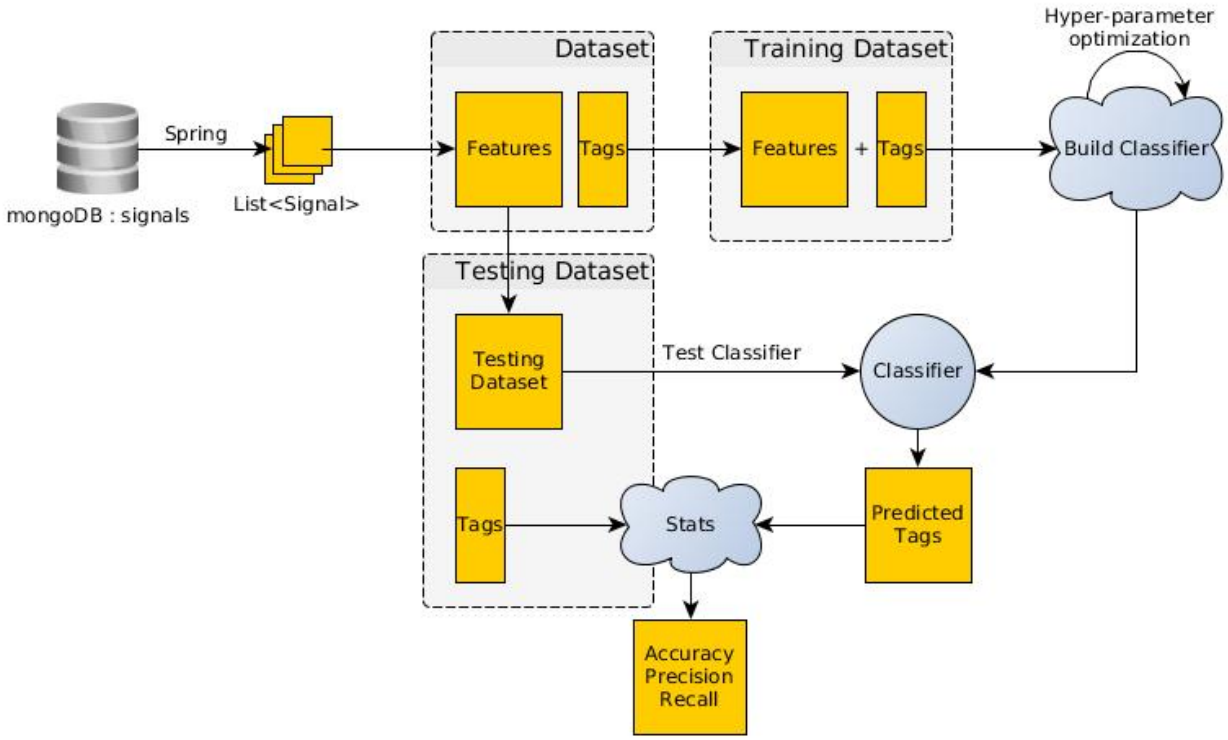


Figure 3.2. – La construction du classifieur.

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 poser 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 *bag of words*.

Dans ce modèle, un texte (une phrase ou un document) est représenté comme un sac (*bag*), un ensemble de ses mots (au sens mathématique), 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. C'est le cas du modèle naïf bayésien.

Ainsi, un signal est caractérisé par la liste des occurrences des mots de son titre et son contenu.

Exemple : Modélisation de deux signaux à l'aide du *bag of words* :

```
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 à l'aide d'un tokenizer :

```
[ "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 (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 une fois dans le premier signal. De la même façon, la deuxième composante correspond au mot « d' », le deuxième mot de la liste et sa valeur est « 2 » car il est présent deux fois dans le signal.

Quelque soit le classifieur que j'ai construis par la suite, l'ensemble de données est toujours construit suivant ce modèle. Pour mon premier classifieur binaire, les 123 signaux validés *JOB* de l'ensemble des 1426 signaux validés (présentés dans le dernier paragraphe de la partie 2.3), forment la classe *JOB* contre le reste.

Enfin, le seul pré-traitement des données mis en place est le fait de garder les éléments du *bag of words* apparaissant au moins cinq fois. Ceci afin de supprimer les mots rares qui n'apportent pas d'informations, quand on fait de la classification textuelle (les mots mal orthographiés, les mots très spécifiques, les noms propres, etc).

Construction du classifieur

Pour construire mon classifieur binaire naïf bayésien, j'ai divisé mon ensemble de données en deux ensembles 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. Christian Frisch me l'a fait remarqué, lors d'une réunion de suivi. J'ai donc repris le code source de la méthode pour ré-implémenter le bon comportement.

Optimisation des hyper-paramètres par validation croisée

Une méthode de la bibliothèque se charge d'optimiser l'hyper-paramètre du classifieur par validation croisée. Dans un premier temps, j'ai simplement réutilisé cette méthode, car les performances en étaient bonifiées.

Évaluation de la qualité du classifieur

Enfin pour évaluer la qualité de mon classifieur, je les testais sur un ensemble de test, pas utilisé jusque ici (les 25% de l'ensemble initial). J'ai calculé la précision et le rappel obtenu par la classe *JOB* et par la classe *USELESS* représentant le reste (pas *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 : Ici, on met l'accent sur le fait d'avoir une précision [5] très élevée quitte à avoir un rappel [6] un peu diminué (car quand l'un augmente l'autre diminue et vice versa). On préfère se tromper très rarement dans notre classification (précision élevée) quitte à rater des signaux (rappel moyen). Autrement dit, on préfère qu'un utilisateur voit peu de signaux intéressants plutôt qu'il voit beaucoup de signaux dont l'intérêt reste à démontrer.

Premier bilan

Le classifieur construit est prometteur. Cependant, la généralisation à plusieurs classes, va amener à faire de nouveaux choix :

— Faut-il adopter une stratégie de type *One versus All* ou *One versus One* ?
— Est-ce que le modèle naïf bayésien va bien se généraliser ?
— Faut-il un prétraitement global et/ou spécifique à chaque classe ?

3.2.4. Amélioration de ma première application et prétraitement

La démarche de construction du classifieur présentée dans la partie précédente (3.2.3) reste la même. Cependant, j'ai repris la classe principale *ColumnDataClassifier* de la bibliothèque *Stanford Classifier*, car elle intègre, notamment, un vrai processus de validation croisée en k plis. Je l'ai donc refactoré à plusieurs fins :

— Pour qu'elle récupère mes données directement depuis Mongo ;
— Pour qu'elle leur applique des prétraitements avant de construire l'ensemble de données ;
— Pour qu'elle réalise dix plis de validation croisée lors de l'apprentissage du classifieur.

Ce dernier point, me permet d'avoir une idée de la capacité de généralisation de mon classifieur.

Agrandissement de mon ensemble de signaux validés

Avant de pouvoir passer à un classifieur multi-classe, il fallait que mon ensemble de signaux validés grandisse. En effet, celui-ci souffre d'un fort déséquilibre entre les classes : il y a beaucoup plus de signaux inintéressants qu'intéressants. Ainsi, considérer les prédictions d'un classifieur construit avec ces données comme correctes était impossible. Il a donc été nécessaire d'en valider d'autres manuellement.

Durant mon stage, j'ai donc organisé plusieurs QAs [7] (Quality Assessment) pour approfondir l'ensemble des signaux d'apprentissage et de test.

Proportions des signaux : La proportion des signaux sans intérêt est énorme (plus de 80% sur environ 300.000).
Pour les QAs, j'ai donc fais une pré-sélection des signaux à valider. J'ai réutilisé le premier classifieur implémenté en Python de Samuel Charron, construit avec les 1426 signaux. Ce classifieur a classifié des signaux non validés. Ce sont les signaux appartenant potentiellement aux

5. Définition en annexe A.1.1
6. Définition en annexe A.1.2
7. Définition en annexe A.2.

catégories *EVENT*, *JOB* et *PRODUCT* qui ont été sélectionnés pour être validés manuellement. Pour les classes *MONEY* et *PEOPLE*, ils ont été sélectionnés dans Mongo à l'aide d'expressions rationnelles sur des termes comme « levée de fonds », « chiffre d'affaire », « nommer », « nomination », etc.

De cette manière, 2574 nouveaux signaux ont été validés. Ce type de méthodes biaise les proportions 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 :
— 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 impose une stratégie *One vs All*. Concernant le modèle de classifieur, deux options s'offraient à moi :
— Un modèle génératif[8] et notamment un classifieur naïf bayésien ;
— Un modèle discriminatif[9] et notamment un classifieur qui maximise l'entropie.
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 avec les 4000 signaux taggés de la même manière que précédemment. Les performances obtenues (visibles en table 3.2) avaient grandement baissées (la classe *PRODUCT* n'y figure pas car elle n'était pas suffisament 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. À ma grande surprise, ce modèle 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.

8. Définition en annexe A.3.1
9. Définition en annexe A.3.3

Le *MaxEnt* ou *Maximum Entropy* : Les modèles *maximum entropy* sont aussi connu sous le noms de *softmax classifiers* et sont équivalent aux modèles de régression logistique [10] multi-classe (avec des paramètres différents). 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.
Ce type de modèle est avantageux dans le cas où les données sont *sparse*. Ce qui est le cas, d'où des résultats meilleurs qu'avec le modèle bayésien.

Les travaux de prétraitement global

Les traitements expliqués ci-après sont réalisés dans la construction de l'application, en amont de la construction de l'ensemble de donnée selon le modèle du *bag of words*.

La segmentation ou tokenisation : La tokenisation consiste à découper une phrase en token, dans l'idéal représentant des mots. Cette tâche est très spécifique à chaque langue. Les difficultés principales sont surtout liées aux contractions. Il est nécessaire de s'attarder sur cette phase de la normalisation pour minimiser la perte sémantique, car beaucoup de traitement s'appuie sur la tokenisation. Dans mon application, j'ai utilisé le *PTB Tokenizer* disponible dans *Stanford Classifier*.

Exemple : Entrée : « Je n'ai pas d'argent. En as-tu ? »
Sorties possibles :
"Je", "n", "ai", "pas", "d", "argent", "En", "as", "tu"
"Je", "n", "'", "ai", "pas", "d", "'", "argent", ".", "En", "as", "-", "tu", " ?"
"Je", "n'", "ai", "pas", "d'", "argent", ".", "En", "as", "-", "tu", " ?"
Le résultat du *PTB Tokenizer* dans l'exemple précédant est le premier.

La casse et la ponctuation : Pour normaliser les mots, une manière simple consiste à réduire la casse de tout les mots. Ainsi, on réduit notre ensemble de mot. Cela est très facile à effectuer mais cela a quand même quelques défauts :

- Les noms propres perdent leur différences par rapport aux noms communs ;
- Les noms d'organisation (comme l'OTAN) perdent leur sens en minuscule.

De plus, supprimer la ponctuation permet aussi d'épurer les données car celle-ci n'apporte pas d'information. Encore une fois, c'est simple à réaliser mais certain mot perde leur sens.

- Les mots composés comportant des tirets perdent leur sens (exemple : « après-midi ») ;
- Les mots composés comportant des apostrophes perdent leur sens (exemple : « aujourd'hui ») ;
- Les acronymes comportant des points de séparation perdent leur sens (exemple : « U.S.A. »)

Les stopwords : Les stopwords sont les mots d'une phrase inutiles à la compréhension de celle-ci. Ils ne portent pas d'information et sont présents dans n'importe quels documents textuels. Les supprimer permet donc de réduire le nombre de feature à notre ensemble de mot (le sac de mot). Une liste de stopwords contient majoritairement des pronoms, des prépositions et des déterminants comme : « a », « au », « ce », « de », « le », « mon », etc.

10. Rappel en annexe A.3.2

La recherche morphologique : Enfin, les deux derniers moyens permettant de normaliser du texte sont la lemmatisation et le stemming. Ces deux traitements ont pour objectif de faire baisser le nombre de forme infléchie. Une forme infléchie est appelé un lexème en morphologie. Il faut savoir qu'un lexème est composé de différentes types de morphèmes : les stems et les affixes. Voici leur définition à l'aide d'un exemple :
Le lexème « chanteurs » est composé de trois morphèmes : « chant », « eur » et « s ». Parmi ces trois morphèmes, un est un stem « chant » et les deux autres des affixes « eur » et « s ».

Le stemming : L'objectif du stemming est de diminuer les lexèmes en les réduisant à leur stems : « chanteurs », « chanteuse » transformés en « chant ».
Un des désavantage du stemming est que les stems ne sont pas toujours des lemmes, c'est à dire la forme canonique d'un lexème (son entrée dans le dictionnaire), et donc pas un mot.
Exemple : le stem de « chercheur » est « cherch » (n'est pas un mot).

La lemmatisation : L'objectif de la lemmatisation est le même que celui du stemming à savoir diminuer le nombre de formes infléchies, de lexèmes. Pour cela, la lemmatisation consiste à réduire un lexème en lemme, la forme canonique de ce lexème. Le lemme d'un verbe correspond au verbe à l'infinitif, le lemme d'un nom commun est ce nom commun au masculin singulier, etc. Les lemmes correspondent aux entrées dans le dictionnaires de tout les lexèmes.

Exemple : Les lemmes de la phrase « Ils chantent leurs chansons préférées. » sont :
« Il chanter leur chanson préférer. »

Remarques : Étant donné que le stemming nous fait perdre plus d'information sur nos features que la lemmatisation, j'ai choisi d'appliquer cette dernière. De plus, avec le stemming on ne manipule plus des mots mais leur stem.
La bibliothèque de Stanford ne propose pas de Lemmatizer pour la langue française. De ce fait, j'ai utilisé une bibliothèque externe de Ahmet Aker [11] que j'ai rajouté dans le projet Spring grâce à Maven en ajoutant une dépendance.

Bilan sur les prétraitements : Dans l'idéal, j'aurais aimé pouvoir réaliser les prétraitements avec le *Stanford CoreNLP* mais il ne gère pas bien le français et il est très gourmand en mémoire.

En outre, les prétraitements permettent de normaliser le texte et de réduire le nombre de feature présent dans le sac de mot. Il est important de réduire la disparité des features pour pouvoir calculer leur fréquence par la suite. Un parallèle pourrait être fait entre la compression de données numériques par ACP (Analyse en Composantes Principales) et la lemmatisation par exemple. Il est très important de bien préparer les données afin d'obtenir les meilleures performances possibles par la suite.

Les travaux de prétraitement spécifique

En plus, des traitements présentés précédemment, il est possible de traiter nos données plus spécifiquement par classe, afin d'augmenter les performances du classifieur par la suite.
Pour cette tâche, il faut d'avantage s'attarder sur le fond des données de chaque classe, et essayer de voir si de l'information intéressante aurait pu être détruite par les traitement précédents.

11. http ://staffwww.dcs.shef.ac.uk/people/A.Aker/activityNLPProjects.html

	JOB	EVENT	PEOPLE	MONEY
Précision	0,96	0,82	0,94	0,81
Rappel	0,81	0,63	0,60	0,60

Table 3.4. – Performances du classifieur maximisant l'entropie.

Les signaux issus de Twitter : Les signaux twitter comportent souvent des références « @pseudo » et des mentions comme « #job ». Les références n'apportent pas d'information dans la majorité des cas. Les supprimer permettrait d'éliminer du bruit dans les features. Quant aux mentions comme « #job », celles-ci portent de l'information et dans un processus de normalisation, il serait bien de garder le mot suivant le dièse (#).

Les emails et les URLs À l'image des pseudonymes Twitter, les emails et les URLs (présent dans certain signaux) ne portent aucune information et ne sont pas bien tokenisés à cause de leur formes. Ainsi, il serait bien de les supprimer en amont de la tokenisation.

Les signaux de la classe *JOB* : Souvent dans les offres d'emploi ou de stage, la mention « H/F » signifiant « homme ou femme » est présente. Cependant, lors de la tokenisation, ce genre de feature est détruit du fait qu'il contient le caractère de ponctuation slash (/). Ainsi, il serait intéressant de pouvoir les détecter avant la tokenisation et de les remplacer par une chaîne de caractère qui ne serait pas détruit par la tokenisation comme le terme « hommeoufemme ».

Les signaux de la classe *MONEY* Une caractéristique des signaux de cette classe est que, souvent lorsque ces signaux parlent d'une levée de fonds, une somme est annoncée avec une devise. Exemple : « L'INSA a levée 5 k€ pour construire un amphithéâtre ».
Il serait donc intéressant de conserver l'unité et le symbole de la devise suivant le montant qui est caractéristique de ce type de signal (k€, m€, etc).

Mise en place de ces traitements et bilan : J'ai implémenté ces traitements assez facilement à l'aide d'expressions rationnelles. Ce travail d'inspection du contenu des signaux est important car c'est là que l'on détecte des informations caractéristiques.

Performances obtenues par mon application avec les prétraitements

Les performances présentés en table 3.5 ont été obtenues en construisant le classifieur comme dans la partie 3.2.3 avec l'application des précédents prétraitements en amont de la construction de l'ensemble de données.

Le problème des FP (faux positifs) : Les signaux qui peuvent avoir potentiellement plusieurs labels engendrent une quantité non négligeable de faux positifs, comme par exemple :

- Signal : "Venez découvrir nos nouveautés produits du 20 au 22 Mai 2014 au salon SEPEM, Hall 4 - Stand F13 http ://t.co/vS7gT2x4yp #marquagepermanent" Label : PRODUCT ou EVENT
- Signal : "Nous embauchons ! Étudiants de HEC Paris, nous sommes aujourd'hui au #CarrefoursHEC. Venez découvrir nos offres d'emploi dans les domaines du #digital #data #marketing" Label : JOB ou EVENT

Une solution potentielle serait d'autoriser le classifieur à attribuer plusieurs label. C'est ce qu'avait fait Samuel Charron en Python mais les résultats n'étaient pas suffisant.

	JOB	EVENT	PEOPLE	PRODUCT	MONEY
Précision	0,98	0,70	0,83	0,56	0,51
Rappel	0,92	0,80	0,97	0,75	0,95

Table 3.5. – Performances du classifieur sur des données non validées.

Le problème des FN (faux négatifs) : Les signaux ne sont pas toujours très succins et précis. De ce fait, parfois en plus d'un contenu intéressant le signal peut contenir du bruit, ce qui engendre une quantité non négligeable de faux négatifs. Ce qui baisse les rappels. Exemple :

- Signal : "Dans le cadre de son développement, EXCELIUM a fait l'acquisition du fonds de commerce de la société lyonnaise SES Vidéo, spécialisée dans l'installation de systèmes vidéo et d'alarme." Label : Aucun alors que c'est MONEY
- Signal : "A l'occasion de la fête de la gastronomie, nous vous invitons à visiter notre nouveau laboratoire à Téteghem samedi 27 septembre de 10 h à 18h. Visite, dégustation et chèques cadeaux offerts pour toute commande passée sur le site cette semaine !" Label : aucun alors que c'est EVENT

Performances du classifieur sur un ensemble de données non vus (obtenus grâce à un QA) : 341 718 signaux ont été tagué automatiquement par le classifieur entraîné sur 4000 signaux validés. (Certaines sources de signaux sont blacklistées et leurs signaux émis ne sont donc pas taguées par le classifieur : Les entreprises d'interim, les chasseurs de tête, les e-commerçants, etc). Parmi tout ces signaux, 388 ont été validé manuellement lors d'un QA. Ces signaux validés ont été sélectionnés aléatoirement parmi les 341 718 signaux tagués par le classifieur. Voici les résultats :

Malgré tout, grâce à tout ces prétraitements, mon classifieur a obtenu de meilleures performances, jugées suffisantes par mon maître de stage Samuel Charron pour que je puisse passer à l'implémentation en Python. Les classes *PRODUCT* et *MONEY* manque d'exemple représentatif, d'où leur mauvais résultat. Un QA supplémentaire sur ces classes leur permettrait de se bonifier.

3.3. Travaux réalisés en Python

À la suite de ces deux mois de formations au *Text Mining*, au *Naturel Language Processing* et à l'*Information Retrieval*, j'étais à présent capable d'identifier les tâches à réaliser en Python : Construire un module Python permettant de réaliser tout les prétraitements expliqués précédemment, reprendre le plugin Python de Samuel Charron afin d'appliquer ces prétraitement en amont de la construction de l'ensemble de données et enfin changer de modèle de classifieur.

3.3.1. Module de prétraitement spécifique aux signaux

Ce module de prétraitement (basé sur des expressions rationnelles) prend en charge la détection des patterns précédents à savoir : les emails, les URLs, les pseudonymes et les références Twitter, les mentions « H/F », les devises, la ponctuation et les stopwords ; et les traite spécifiquement :

- les emails, les URLs, les pseudonymes Twitter, la ponctuation et les stopwords sont supprimés ;
- les références Twitter perdent leur dièses (#) ;
- les mentions « H/F » sont remplacés par la chaîne de caractère « hommeoufemme » ;

— les devises (k€, m€) sont remplacés par les chaînes de caractère « millier d'euros », « millions d'euros ».

3.3.2. Module de lemmatisation

En Python 3, il n'existe pas de module permettant de faire de la lemmatisation de la langue française. Je me suis alors demandé comment le lemmatizer, que j'avais utilisé en Java, fonctionnait. Celui-ci couplait le *POS Tagging* des mots à un autre outils, le HFST (*Helsinki Finite State Transducer*), permettant de trouver les différents lemmes possibles d'un mot.

Exemple d'utilisation du HFST : Entrée : « suis »
Sortie :

```
1 être + verb + singular + indicative + present + firstPerson
2 suivre + verb + singular + imperative + present + secondPerson
3 suivre + verb + singular + indicative + present + firstPerson
4 suivre + verb + singular + indicative + present + secondPerson
```

Idée du lemmatizer Java :

1. Pour chaque phrase, calculer le *POS Tagging* de chaque mot via un POS Tagger ;
2. Pour chaque mot de chaque phrase, trouver tous ses potentiels lemmes via le HFST ;
3. Choisir le bon lemme grâce à l'information apporter par son *POS Tagging*.

Helsinki Finite-State Transducer Technology (HFST)

Le *Helsinki Finite-State Transducer software* est une implémentation de plusieurs outils d'analyse morphologique et d'autres outils basés sur les transducteurs (éventuellement pondérés) à état fini. Il s'agit d'un grand projet portant sur les transducteurs littéralement en anglais « transform reducer ».
Le HFST propose un module Python permettant de trouver tout les lemmes d'un mot grâce notamment à un modèle de transducteur du français.

Stanford POST Tagger

Pour le POS Tagging, il existe un wrapper Python du *Stanford POS Tagger*. Cet outils fournit également un modèle français pour taguer des phrases en français.

Qualité du POS Tagger : Pour vérifier la qualité de celui-ci, j'ai utilisé des corpus de texte français. Ceux-ci présentent un texte dans un format XML, sous forme de liste de paires mot/-fonction grammaticale. J'ai du transformé le corpus en un texte composé de phrases afin de pouvoir le passer à mon *POS Tagger* Python. J'ai également du transformé le corpus car les caractères de ponctuation étaient utilisé comme séparateur. Ceci créait un décalage entre la sortie de mon POS Tagging et cette liste de paires.
Une fois les problèmes de formatage résolus pour un texte du corpus, j'ai testé le *POS Tagger* dessus et celui-ci a obtenu un taux d'erreur jugé suffisant.

Lemmatiser Python :

J'ai donc mis au point Lemmatiser Python 3. Avec Clément Chastagnol, on a pu le déployé, mais un problème inattendu est survenu. En effet, à chaque appelle de la méthode de POS Tagging, derrière le wrapper Python lance la bibliothèque de Stanford dans une nouvelle JVM. Ceci ralenti énormément (facteur 100) le processus de traitement des signaux malgré une parallélisation des tâches (multithread).

3.3.3. Plugin de classification

Le plugin de classification permet de créer un modèle *maximun entropy*, une régression logistique multinomiale, *One vs All* à partir des 4000 signaux. Une fois le modèle créé, ce plugin est déployable et permet de taguer de nouveaux signaux. Celui-ci utilise les modules Python 3 scikit-learn et numpy.
Numpy est un module pour la gestion de tableau (matrice).
Scikit-learn est un module de machine learning construit au dessus de numpy pour faciliter la gestion des matrices.
L'utilisation de scikit-learn est relativement facile car la documentation est très bien faite mêlant explications aux exemples d'utilisation.

3.4. Bilan sur mes travaux

J'ai réussi à mettre au point un plugin Python capable de classifier les signaux captés par C-Radar.

Les pistes d'amélioration possibles sont les suivantes :
- Introduire un outils permettant de détecter la langue du signal (comme Tika) afin d'adapter les traitements en aval ;
- Utiliser un détecteur d'entités nommées *Named Entity Recognizer* afin de gérer autrement les noms d'entreprises, de personnes qui sont généralement caractéristiques des signaux *MONEY* (exemple : « Data Publica a levée 10 m€ en janvier auprès du fond d'investissement français constitué du groupe Lagardère... »). Pour le moment ces noms propres sont rejeté comme feature car tous sont isolés n'étant pas les mêmes.
- Redéfinir la classe *MONEY* en deux classes, comme suit, pour voir si le classifieur s'améliore :
  - *MONEY* : Déclarations de résultats financiers, de CA (chiffre d'affaire), de levée de fond ;
  - *BUSINESS* : Déclaration de Partenariat, de rachat ou d'aquisition d'entreprises, de « contrat gagné », etc.
- Se tourner vers des outils statistiques comme le traducteur de Google pour exécuter la lemmatisation.

Les difficultés majeures rencontrés sont le manque d'outils de *Naturel Language Processing* pour la langue française. En effet, pour l'anglais les outils sont très bien aboutis et disponible mais pour le français il est difficile de trouver de bon outils. Je pense notamment au Lemmatiser qui n'existe pas pour le français en Python 3. J'ai vu qu'il y avait des outils tel que clips disponibles mais j'avais une contrainte sur la version du Python utilisé. En effet, il existe de grande différence entre Python 3 et Python 2. Il existe des wrapers permettant de faire le pond entre les deux mais d'un point de vu développement c'est moyen.

4. Conclusion

Ce stage a été une expérience très enrichissante tant sur le plan humain, que sur le plan technique.

Sur le plan humain, j'ai eu la chance de rencontrer des personnes extrêmement sympathiques, compétentes et ouvertes à la discussion. L'équipe de Data Publica m'a toujours bien encadré. En effet, lorsque je rencontrais une difficulté, j'avais constamment quelqu'un de disponible à déranger pour mettre à l'épreuve son expérience.

En outre, j'ai découvert le travail dans une structure de type start-up, très favorable à la réussite, au partage et au dépassement de soi.
Ce stage a également été très riche, techniquement parlant, car certain domaines abordés m'étaient inconnus et se sont avérés passionnant.

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. Les métriques de mesure de la qualité d'une classification

A.1.1. La précision

La précision mesure le nombre de fois où on a bien classifié un document.

A.1.2. Le rappel

Le rappel mesure le fait qu'on ait trouvé tout les documents d'une classe.

A.1.3. La norme F1

A.2. 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.

A.3. Modèle de classifieur

A.3.1. Le modèle génératif

Le modèle génératif maximise la vraisemblance de la probabilité jointe $P(classe, donnée)$.

A.3.2. La régression logistique binomiale

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

A.3.3. Le modèle discriminatif

Le modèle discriminatif maximise la vraisemblance de la probabilité conditionnelle $P(classe|donnée)$.

A.4. Les transducteurs

Définition [1] tirée de Wikipedia.

1. .

En informatique théorique, en linguistique, et en particulier en théorie des automates, un transducteur fini (appelé aussi transducteur à états finis par une traduction maladroite de l'anglais finite state transducer) est un automate fini avec sorties. C'est une extension des automates finis. Ils opèrent en effet sur les mots sur un alphabet d'entrée et, au lieu de simplement accepter ou refuser le mot, ils le transforment, de manière parfois non déterministe, en un ou plusieurs mots sur un alphabet de sortie. Ceci permet des transformations de langages, et aussi des utilisations variées telles que notamment l'analyse syntaxique des langages de programmation, et l'analyse morphologique ou l'analyse phonologique en linguistique.

Des explications détaillées sont disponibles ici.

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

Financeurs institutionnels