Modélisation Statistique et Stochastique

RAPPORT DE STAGE

S.A.S. MAKEZU

Traitement automatique du langage pour la recherche de prospect sur les réseaux sociaux

Master 2 mention Mathématiques Appliquées & Statistique
option Modélisation Statistique & Stochastique

14 avril - 11 septembre 2020

MAKEZU.IO

Hugo Lacauste

TABLE DES MATIÈRES

1 Le traitement automatique du langage **7**

1.1 L'analyse de texte . . . 7

1.2 Approches empiriques . . . 9

1.3 Le modèle Word2Vec . . . 11

1.4 Google Brain . . . 15

1.4.1 Les Transformers . . . 16

1.4.2 B.E.R.T. . . . 17

2 L'application Makezu **19**

2.1 Le produit Makezu . . . 19

2.2 L'application Flask-C.O.R.S. . . . 20

2.3 Les limites de l'A.P.I Twitter . . . 22

2.4 Le N.L.P. dans Makezu . . . 23

2.5 La description et les audiences . . . 25

3 Les données **29**

3.1 L'étiqueteur automatique de données . . . 29

3.2 La vie privée et les données sur Twitter . . . 30

4 Les moments de vie **32**

4.1 Vectorisation & approche non supervisée . . . 32

4.2 Vérifications d'hypothèses . . . 35

4.3 Les mesures de qualités . . . 36

4.4 Les modèles paramétriques . . . 38

4.5 Les modèles d'ensembles . . . 41

4.6 Synthèse des résultats . . . 43

5 Amélioration du ciblage produit **45**

5.1 Fichage automatique . . . 45

5.2 Modèle de B.E.R.T. 48
5.3 L'encodeur de phrases 49

A Les librairies Python utilisées **54**

B spaCy **61**

C Base de données **63**

Avant-Propos :

C'est avec beaucoup d'émotions et un peu de nostalgie, que je rédige ce rapport de stage, clôturant mes études. Je tiens donc tout d'abord à remercier mes parents, pour leur soutien inconditionnel que ce soit affectif, financier et dans la mesure du possible, technique, malgré quelques doutes quant à mes choix personnels. Ensuite, je tiens à remercier chaleureusement ma re-lectrice assidue et engagée.

Pour mon stage de fin d'études de master, j'ai eu l'opportunité de rejoindre la S.A.S. Makezu qui a décidé de me faire confiance en me recrutant malgré les conditions sanitaires exceptionnelles qui nous ont obligés à démarrer notre coopération à distance, je tenais donc à commencer ce rapport en exprimant ma gratitude à l'égard de Mesdames Hélène Lucien et Anne-Sophie de Gabriac.

Pour la lecture de ce rapport, j'ai mis en place pour chaque chapitre et sous-chapitre un court descriptif du contenu, le lecteur pourra donc avoir une idée claire du contenu de la section qu'il s'apprête à lire.

Introduction :

J'ai donc intégré cette toute jeune start-up parisienne, fondée en novembre 2019, par deux jeunes entrepreneuses : Hélène Lucien, forte d'une première expérience dans la création de start-up autour de Twitter, et Anne-Sophie De Gabriac, qui partage son temps entre Makezu et une thèse. L'objectif de mon stage est d'améliorer l'outil de traitement automatique du langage. En effet, le projet de Makezu, que l'on peut traduire du japonais par imbattable, est de permettre à des entreprises de toutes tailles et de tous moyens de gagner "intelligemment" de la visibilité sur les réseaux sociaux. Makezu est parti d'un constat ingénieux : la plateforme Twitter est forte d'une base de plus de 166 millions d'utilisateurs, dont 145 millions s'expriment quotidiennement [1], pour le réseau social dont le premier slogan était : *"What are you doing ?" (Qu'êtes-vous en train de faire)*. Et bien que Twitter se soit transformé en site d'informations, de par les pratiques des utilisateurs de la plateforme (avec par exemple, le relais des attaques terroristes à Bombay (2008), ou la répression du peuple syrien par le régime de Bachar El-Assad (2011)), il reste un réseau social très ouvert, dont les utilisateurs n'hésitent pas à partager leur vie quotidienne, ainsi que leurs ressentis. De ce constat, ainsi que d'une collaboration avec l'école CentraleSupElec, est née l'application Makezu, qui, à l'aide d'un outil d'apprentissage automatique, sélectionne des tweets qui expriment un besoin correspondant aux prérogatives de l'utilisateur de l'application. Mon rôle lors de ce stage était tout d'abord de me familiariser avec l'outil développé par les étudiants de Centrale, pour en résoudre les problèmes liés à l'utilisation, puis d'enquêter sur les innovations en traitement du langage pour améliorer "l'intelligence" de l'application. Mon rapport se décomposera de la façon suivante : il commencera par un large tour d'horizon de ce que l'on appelle, en apprentissage automatique, le traitement du langage. Avec une analyse chronologique des avancées dans ce sujet, qui a connu au cours de ces dernières années un regain d'intérêt impressionnant, grâce aux travaux des équipes de Google et Facebook. Je m'attarderai, ensuite, sur l'outil mis en place par Makezu, partageant avec vous les détails de l'utilisation de l'application ainsi que les choix qui ont été fait pour le traitement des données sur Twitter, analysant les résultats obtenus par cet outil. Ainsi, je terminerai par les différents axes de réflexion qui ont rythmé notre collaboration et qui ont convergé vers la mise en place d'un nouvel outil d'intelligence artificielle.

1. Ces chiffres proviennent du blog du réseau social Twitter https://blog.twitter.com/ (consulté le 12 août 2020)

CHAPITRE 1

LE TRAITEMENT AUTOMATIQUE DU LANGAGE

Ce chapitre essaye de parcourir de la manière la plus exhaustive possible les recherches autour des processus de traitement automatique du langage et s'attardera plus particulièrement sur les méthodes récentes. Il s'agit d'un rapport des connaissances théoriques qu'il a fallu que j'intègre pour mettre au point un outil fiable, et "intelligent".

1.1 L'analyse de texte

Dans cette section, on parcourt l'histoire des réflexions humaines sur l'analyse du langage et des langues[8] [9], et on définit l'ensemble des domaines d'utilisation de cette science.

Le sujet de traitement automatique du langage trouve son origine dans les années 50, durant la guerre froide, à un moment de l'histoire où la traduction automatique, notamment de textes russes pour les renseignements américains, devient un intérêt stratégique. Le 7 janvier 1954, l'Université de Georgetown, Kentucky, avec l'aide d'I.B.M. conduit une expérience qui portera ensuite le nom de *"Georgetown–IBM experiment"*, qui traduira automatiquement du russe à l'anglais, plus d'une soixantaine de phrases. Cette expérience fut basée sur 6 règles grammaticales, à l'aide de 250 éléments grammaticaux [1]. Cette expérience reçut un accueil largement enthousiaste de la part de la communauté scientifique, et on établit alors que dans les 3 à 5 ans, la traduction dans n'importe quelle langue serait automatisée.

Le rapport A.L.P.A.C. [2] de 1964 (9 ans plus tard), présidé par l'ingénieur américain John Robinson Pierce, conclut sur l'échec des objectifs fixés, ce qui entraina un arrêt du financement du programme de recherche. La même année, on assiste tout de même à la création d'*ELIZA*, un programme automatique qui simule un psychothérapeute, concluant à une dépendance émotionnelle de certains interlocuteurs du programme, et donnant

1. Mot, phrase, ou proposition, constituant une partie d'une construction grammaticale de taille supérieure.
2. Automatic Language Processing Advisory Committee, Comité consultatif sur le traitement automatique des langues en français

naissance à ce que l'on appellera *l'effet ELIZA*, une caractéristique humaine à donner plus de sens aux mots que ceux qu'ils n'ont réellement.

Mais c'est dans les années 80 que le traitement naturel du langage prend son véritable essor, d'une approche de compilation par plusieurs règles complexes manuelles, on intègre des processus d'apprentissage automatique dans le traitement du langage, possible par l'amélioration technologique des puissances de calculs informatiques[1][11]. On peut citer par exemple l'utilisation d'arbres de décisions avec leurs règles *if-then* (**si** on vérifie cette assertion **alors**...), qui est similaire à certaines règles de constructions grammaticales, par exemple **si** la phrase commence par les mots "Qui", "Quoi", "Comment", "Pourquoi", **alors** la phrase se terminera par : '?'. Ou bien encore, en grammaire, lorsqu'on rencontre dans une phrase des verbes transitifs directes (ou indirects), on aura toujours dans la phrase un complément d'objet direct (ou indirect). On remarque également, la mise en lumière de modèles de Markov cachés, dans des applications d'annotation grammatico-lexicales.

Dans le début des années 2000, les méthodes d'apprentissage se sont concentrées sur des algorithmes non supervisés, comme le regroupement de textes en fonction de leur sujet. On utilise pour cela, l'analyse sémantique latente[7], qui fait intervenir une matrice d'occurrences des termes dans un corps lexical (texte), puis par une réduction de rang (on supprime les termes qui n'apparaissent que de manière anecdotique, on regroupe les termes aux sens similaires, ...). Avec par exemple, l'algorithme T.F.-I.D.F., on introduit la vectorisation des mots, qui permet une représentation algébrique des documents et donc autorise une analyse mathématique. On crée un espace vectoriel du document, on identifie dans cet espace les relations de proximité des mots, en construisant une norme, donc une mesure et une distance dans cet espace vectoriel, par exemple la similarité cosinus entre les vecteurs :

$$cos(\alpha) = \frac{d_1 q}{||d_1|| * ||q||} \tag{1.1}$$

Ce sont ces techniques de vectorisation qui vont faire l'objet de recherches poussées depuis 2010, avec l'utilisation des réseaux de neurones (notamment récurent dans ce cas précis), et la possibilité technique de se servir d'apprentissage profond. Ces méthodes de vectorisation vont permettre l'émergence des techniques que l'on nomment en anglais le "word embedding", textuellement traduit par plongement, que l'on peut également appeler mondage, de mots, pour reprendre l'expression associée aux tomates, qui signifie décortiqué, faire apparaître le cœur. C'est une technique de vectorisation des mots dans laquelle on intègre le contexte de la phrase dans laquelle le mot "mondé" est présent. Cette méthode repose sur l'hypothèse grammaire distributionnelle[10] lexicale, théorisée par Leonard Bloomfield et Z. Harris, on peut également, citer John Ruppert Firth en 1957 : "Vous connaîtrez un mot par ses fréquentations" [3], c'est l'analyse sémantique latente. Par exemple, on peut décomposer la phrase en élément hiérarchique *Ex : [(L')ordinateur de (Toufik)] [n'est plus (tout jeune)]*. Cette hypothèse est contredite par les théories chomskyennes des années 60 et par l'analyse des dialectes natifs américains. Noam Chomsky développe la théorie syntaxique de grammaire générative et transformationnelle[3], travaux qui trouveront leurs échos dans les recherches associées de Google Brain et de l'université de Toronto, aboutissant à la rédaction de l'article "Attention Is All You Need"[18], le 6 décembre 2017 .

Ils rejettent les réseaux de neurones récurrents, et apportent des nouvelles techniques de vectorisation par l'apport d'un mondage de mots sur la position dans le texte. C'est l'invention des transformers. Pour finir, on en parlera plus en détails dans un sous-chapitre suivant, mais le 2 novembre 2018, l'équipe de Google AI Language, dirigée par les chercheurs Jacob Devlin et Ming-Wei Chang, publie B.E.R.T. acronyme de "Bidirectional En-

3. Analyse des co-occurrences récurrentes on trouve cette citation sur le site : www.sciencedirect.com (consulté le 22 juillet 2020)

coder Representations for Transformer" (Représentation d'encodeur à deux directions pour les transformers.). C'est un modèle pré-entrainé sur la base de données *texte* de Wikipédia.

En effet, pour l'analyse du langage, le premier facteur de perte d'efficacité vient de l'entrainement des modèles. Pour prendre un exemple simple, le mondage de mot pour "général", suivant l'emploi de ce nom dans une phrase , on peut lui trouver plusieurs sens. Les vectorisations de mots, avant B.E.R.T., nous donnent un même vecteur, qu'il soit dans la phrase "Le 18 juin le général De Gaule appela la France depuis Londres", ou dans la phrase "Suivant l'avis général des médecins, le président rend obligatoire le port du masque dans les lieux clos au 1er août".

Avec l'arrivée de B.E.R.T., on obtient un modèle qui identifie cette différence. Cette révolution a permis la création, par une étape de réglage particulier ("fine-tuning"), de modèles spécifiques aux langues. Par exemple en français, le résultat de chercheurs de l'INRIA, Facebook A.I. Research avec Sorbonne Université, s'est vu affubler du patronyme de CamemB.E.R.T.[14] [4]. Ce modèle est évalué sur le corpus de textes O.S.C.A.R. [5].

Quand on parle de langage, on ne parle pas seulement de vocabulaire, les langues sont soumises à des règles de construction, la grammaire. Le sens des phrases est changé suivant comment ces règles sont appliquées. En effet, les phrases : "vivre pour manger" et "manger pour vivre", bien qu'étant strictement composées des mêmes mots possèdent deux sens différents. Ces règles sont alors théorisées mathématiquement.

1.2 Approches empiriques

On a, au début du 20^{eme} siècle, l'apparition de la Loi de Zipf, qui est une observation empirique, décrivant la fréquence des mots dans un texte, elle est très proche de la loi de Pareto. Elle sert de base à la méthode de pondération T.F.-I.D.F. [6], fournissant une mesure d'importance d'un terme dans un corps de texte.

L'histoire[19] autour de sa création, vient de l'analyse du roman **Ulysse** de *James Joyce*. Elle veut que le mot qui apparait le plus souvent dans ce roman soit présent à 8000 reprises, le 10^{eme} mot le plus répété, 800 reprises, puis 80 fois pour le 100^{eme} mot, et 8 le 1000^{eme}. Ce résultat surprenant, bien qu'inexact, lance les travaux de George Kingsley Zipf, qui se concluent sur la déclaration d'une loi statistique, déterminée par une fonction de masse :

$$f(k) = \frac{1}{H_{N,s}} \frac{1}{k^s}, \text{ où } H_{N,s} = \sum_{n=1}^{N} \frac{1}{n^s} \tag{1.2}$$

On pose N le nombre d'éléments ($\in N^*$), k le rang (également $\in N^*$), et s(> 0), un paramètre de factorisation pour corriger le fait que l'on n'ait pas dans ces distributions un nombre infini d'éléments, et donc pour former une loi discrète de probabilité, dans le cas d'un nombre N fini d'éléments, le paramètre s a pour condition $s > 1$, on est alors dans le cadre d'une loi zêta.

$$\sum_{n=1}^{\infty} \frac{1}{n^s} = \zeta(s) \tag{1.3}$$

4. Les auteurs de ce modèle sont Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamé Seddah et Benoît Sagot. Un lien vers la page propre à ce modèle : https://camembert-model.fr (consultée le 17 juillet 2020)

5. Lien vers le site de ce corpus : https://traces1.inria.fr/oscar/(consulté le 17 juillet 2020) Open Super-large Crawled ALMAnaCH coRpus ou en français, l'almanach de corps de texte ouvert de taille astronomique, site consulté le 17 juillet 2020

6. "Term frequency-inverse document frequency", en français : fréquence du mot | fréquence inverse du document

On remarque qu'il s'agit de l'inverse des éléments d'une suite harmonique.

On a également la représentation graphique des fonctions de masse et répartition de la loi de Zipf [7] :

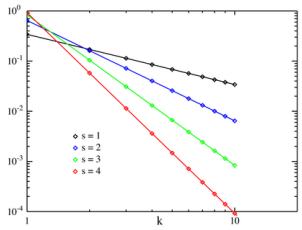

FIGURE 1.1 – Distribution de la loi de Zipf - source Wikipédia

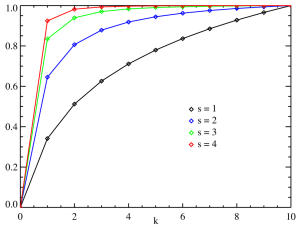

FIGURE 1.2 – Répartition de la loi de Zipf - source Wikipédia

Cette distribution légitime l'algorithme T.F.-I.D.F., dont les formules sont :

$$TF = \frac{Nombre\ d'occurrences\ du\ mot}{Nombre\ de\ mots\ dans\ le\ texte} \mid IDF = \log \frac{|D|}{|\{d_j : t_i \in d_j|\}} \tag{1.4}$$

Avec, $|D|$: nombre total de documents dans le corpus, et $|\{d_j : t_i \in d_j|\}$: le nombre de documents où le terme t_i apparaît.

On applique une échelle logarithme pour le calcul de la fréquence inverse du document, dont la base (2 ou 10), peut changer, pour obtenir des résultats de fréquences facilement manipulables (on évite de se retrouver avec des fréquences $\approx 10^{-7}$).

Pour résumer, l'algorithme T.F.-I.D.F. permet d'obtenir, pour un corps de documents donné, les fréquences d'utilisation de chaque mot pour chacun des documents. On obtient alors, une matrice de pondération (les nombres d'occurrences) de chaque mot d'un vocabulaire fini, pour un ensemble de documents donné. On décrit alors l'importance d'un mot dans un document par la valeur de son coefficient de pondération. De ce fait, si on cherche un document dont le sujet est, par exemple, la "*pêche*", on récupère la matrice de pondération. À la ligne du mot "*pêche*" dans le vecteur V (vocabulaire) on récupère les colonnes (document), où le coefficient est le plus élevé. Cet algorithme, bien qu'extrêmement simpliste, est toujours largement utilisé avec des variantes spécifiques dans les moteurs de recherche de tous types (web, documentaire, streaming).

Cependant, ce modèle possède des limites, le T.F.-I.D.F. brut ne permet pas de gérer les synonymes. Deux articles totalement équivalents peuvent alors sur la base d'un mot être très "éloignés" (dans l'espace vectoriel du T.F.-I.D.F.). De la même manière, les mots composés ne sont pas bien intégrés dans cette procédure de traitement du langage. T.F.-I.D.F. ne gère pas non plus la morphologie. Il n'interprète pas la place des mots dans la phrase, et on l'a remarqué plus haut, cela a son importance. Ce dernier point est particulièrement intéressant car il nous permet d'éclairer une hypothèse fondamentale de ce processus, qui repose sur l'indépendance de la représentation des mots, c'est-à-dire que tous les mots ont une chance équiprobable d'apparaitre dans un texte. Or, et ici je me permets de citer les résultats du compte rendu d'une étude québécoise de 1992, fourni par *Jean Baudot*[17], qui sur un recueil de textes divers et détaillés dans le compte rendu, a pu affirmer que le mot le plus utilisé, représente 7% de l'ensemble des mots, que 3000 éléments sont nécessaires pour former 90% de la totalité des textes (sur les 21700 mots que compte l'ensemble des corpus (13.8%)). Pour finir, 27.6% des occurrences sont uniques. Les résultats de ce rapport pourraient faire l'objet d'un test d'indépendance du χ^2, pour invalider l'hypothèse d'indépendance d'apparition de mots dans un texte.

Malgré tout, cet algorithme est à la base des méthodes de représentation vectorielles des mots et textes[16]. Il

7. Les graphes de distribution et répartition de la loi de Zipf ont été récupérés sur le site de Wikipédia : https://fr.wikipedia.org/wiki/Loi_de_Zipf

permet d'introduire des notions mathématiques de distance et proximité pour régir les langues. De nombreux modèles ont suivi cette approche, notamment en 2013 avec le protocole de vectorisation de texte "Word2Vec", qui est un réseau de neurones sac de mots continu ("continuous bag-of-words"). Il est couplé à un protocole "skip-gram", qui n'a pas vraiment de traduction en français, mais qui fait allusion à la méthode d'entrainement n-grammes, où n sont les éléments et $grammes$ leur poids, on pourrait donc traduire ça par le protocole "poids négligés".

1.3 Le modèle Word2Vec

Les années suivantes marquent différentes avancées sur des protocoles qui reprennent les codes de Word2Vec. Dans ce chapitre, on se donne l'ambition d'exposer son fonctionnement. Ce modèle est introduit le 16 janvier 2013, sous la supervision de Tomas Mikolov (Google)[15]. Pour commencer, la vectorisation ressemble à ce que l'on peut rencontrer avec T.F.-I.D.F.. On a toujours notre représentation d'ensemble de mots, que l'on nomme vocabulaire (noté V), de la plus petite taille possible (pour faciliter les calculs). On caractérise ce vecteur avec l'encodage "un-chaud" ("one-hot"), c'est-à-dire que chaque mot est un vecteur de la taille du vocabulaire, possédant des 0 à toutes les lignes sauf la i-éme qui, de ce fait caractérise le mot. On fait également en sorte d'exprimer la proximité des mots de contexte similaire, par exemple on peut s'attendre à ce que les vecteurs des mots "chien" et "chat" soient proches dans l'espace vectoriel sur lequel on les "plongent", parce que ce sont des mots qui sont "similaires", dans le sens où chacun de ces deux mots peut se retrouver de manière très probable dans la même phrase. En effet, les phrases : *"Benjamin adore jouer avec son chat de compagnie"* et *"Benjamin adore jouer avec son chien de compagnie"*, ont un sens lexical, alors que la phrase *"Benjamin adore jouer avec son char de compagnie"*, bien que plus proche de la phrase originale (dans le nombre de caractères commun), n'a pas de sens lexical.
Ici, on obtient le plongement des mots par un apprentissage profond sur deux types de réseaux de neurones, que l'on schématise rapidement comme : $X^{(input)} \; -> Réseau\ de\ Neurones \; -> Y_{(output)}$, Où X et Y, sont des vecteurs (ou matrices suivant le contexte) à valeurs dans $\{0;1\}$.

"Skip-gram" : On choisit un mot central i, et on essaye de retrouver les mots qui composent son voisinage. On passe par des couches cachées du neurone, qui se décomposent comme suit :

$$\text{L'Encodeur : } X \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} * W^{input} \begin{pmatrix} \dots \\ \vdots\ \vdots\ \vdots \\ \dots \\ \dots \\ \dots \\ \vdots\ \vdots\ \vdots \\ \dots \end{pmatrix} = h^t \begin{pmatrix} \dots\dots \end{pmatrix} \quad \text{Le Décodeur : } h \begin{pmatrix} \vdots \\ \vdots \\ \vdots \end{pmatrix} * W_{output} \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} = C \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} \quad (1.5)$$

La matrice C contient alors, pour chaque ligne, les probabilités d'avoir chacun des mots du vocabulaire. Pour

gérer cette probabilité, on utilise la généralisation de la fonction logistique, qui est le softmax ($\sigma(Z)$) :

$$\sigma(Z)_j = \frac{e^{Z_j}}{\sum_{k=1}^{K} e^{Z_k}} \text{ avec } \sum \sigma(Z) = 1 \tag{1.6}$$

Et donc lorsqu'on applique cela aux éléments de notre réseau de neurones, on obtient :

$$p(w_{Cj} = w_{O,C}|w_i) = y_{Cj} = \frac{e^{u_{Cj}}}{\sum_{j=1}^{V} e^{u_j'}} \tag{1.7}$$

Sac-de-mots : Masque un mot et essaye de le retrouver grâce à son voisinage. C'est-à-dire que l'on part d'un ensemble de mots dans une phrase que l'on note $\{x_0, ..., x_N\}$ avec une valeur manquante x_i pour $i \in \{0, N\}$ et on prédit le mot manquant y_i en faisant passer notre matrice de mot connu par un réseau de neurones (encodeur-décodeur).

$$X \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix} * W^{input} \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} = h^t \left(\dots\dots \right) \rightarrow h \begin{pmatrix} \vdots \\ \vdots \\ \vdots \end{pmatrix} * W_{output} \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} = C \begin{pmatrix} \vdots \\ \vdots \\ \vdots \end{pmatrix} \tag{1.8}$$

Le vecteur C est obtenu pour caractériser sur les N éléments de notre vocabulaire, qu'elle est la chance de retrouver le mot manquant x_i.

$$p(w_j|w_i, ..., w_C) = y_j = \frac{e^{u_j}}{\sum_{j=1}^{V} e^{u_j'}} \tag{1.9}$$

Dans ces deux réseaux de neurones, on définit la fonction de coup ("loss-function") qui permet l'entrainement du modèle puisqu'on va résoudre le problème d'optimisation qui réduit cette fonction. Une méthode qui simplifie les problèmes calculatoires, liés au grand nombre de données (mots), est le calcul de l'entropie-croisée ("cross-entropy").
L'entropie croisée de deux distributions q et p se mesure par $H(q, p) = -\sum_x q(x) \log p(x)$. On utilise la rétropropagation du gradient pour le calcul de descente du gradient stochastique sur chaque erreur par couches du réseau de neurones. Dans notre cas avec la représentation matricielle de nos méthodes, on obtient les formules de fonction de coûts suivantes :

$$E = -\log p(w_O|w_I) \quad | \quad E = -\log p(w_{O,1}, ..., w_{O,C}|w_I) \tag{1.10}$$

De cette manière, on exprime les coûts sur un nombre réduit de mots du vocabulaire, et non pas sur l'ensemble des mots (à la différence donc de T.F.-I.D.F.), ce qui est primordial dans les méthodes itératives. C'est cette réduction de dimension qui fait la force de la combinaison des modèles "skip-gram" et sac de mots. Par exemple, pour la traduction automatique, on procède maintenant sur une phrase et non plus mot-à-mot.
À la suite de cette découverte, d'autres méthodes ont vu le jour pour l'analyse automatique du langage et particulièrement sur la vectorisation de textes.

On peut citer par exemple *Doc2Vec*, qui vectorise l'ensemble des mots à la place d'un mot, *FastText*, de FacebookA.I., *VarEmbed*, qui prend en compte les morphèmes [8], ou encore *WordRank*, issu d'un projet Amazon, qui place le plongement de mot dans une analyse de rang, et calcule la fonction de coût pour l'apprentissage du réseau sur le rang des mots du vocabulaire.

Les méthodes de plongements de mots ont donc permis la prise en compte du contexte des mots avant de les vectoriser. De plus, en ajoutant une variable qualitative, on peut arriver à des méthodes d'analyse supervisée, par l'ajout de "*masque*". Par exemple, sans ajout d'un "*masque*", les mots "bon" et "mauvais", après la méthode de mondage, vont avoir des vecteurs proches, alors que si on cherche à faire ressortir un sentiment d'une phrase, on peut ajouter une nouvelle couche dans ce réseau qui prendra en compte l'appréciation.

Un enjeu de ce plongement de mot vient du fait que l'on s'intéresse au voisinage des mots que l'on essaye de retrouver. On prend en compte la proximité de notre objet avec les éléments qui lui sont autour. Ce type de réseau de neurones, porte le nom de récurrent. Ce sont des réseaux de neurones qui utilisent la sortie d'une couche pour en alimenter la suivante, d'où le terme de récurrent (ce type de réseaux excelle pour des données temporelles). Ils ont également la particularité de pouvoir gérer des entrées|sorties de taille variable, c'est à dire qu'à la différence des réseaux de neurones classiques, qui possèdent une entrée et une sortie, on peut intégrer dans le réseau plusieurs entrées, mais également plusieurs sorties.

Ce type de réseau souffre du problème de perte de gradient, ou de perte de mémoire, c'est-à-dire qu'au bout d'un certain nombre d'itérations, lorsque les séquences étudiées sont trop longues, les informations que l'on a en entrée, qui héritent des cellules de réseaux antérieurs, n'apportent pas d'avantage au modèle. Pour lutter contre ce problème, des architectures particulières ont été mises en place : on peut citer la longue mémoire à court-terme (L.S.T.M.[12]), ou l'unité récurrente à portes (G.R.U.[4]).

"L.S.T.M." :

Pour démarrer l'illustration de ce réseau de neurones, et puisqu'une image vaut mille mots, je me propose de reprendre le schéma du "L.S.T.M." [9].

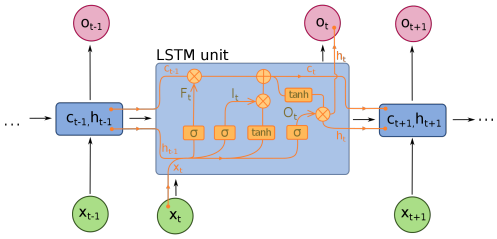

FIGURE 1.3 – Représentation d'une couche d'un LSTM - source Wikipédia

Sans autre explication, cette représentation n'a que peu d'intérêt. On se propose donc de décortiquer cette couche du réseau, pour bien comprendre de quoi il retourne. Pour bien démarrer, une première étape est de découper ce réseau en ce que l'on appelle des portes ("gates"). Ce réseau est composé de 3 portes, une porte d'oubli ("forget gate"), une porte d'entrée ("input gate"), et une porte de sortie ("output gate") [10].

8. La morphologie du mot, par exemple dans le mot "joueurs", on peut décomposer en jouer- [le fait], -eur- [le faiseur], -s [le pluriel], chacune de ces décompositions apporte de l'information sur le mot en lui-même, c'est la grammaire transformationnelle chomskyenne.

9. Cette représentation du réseau de neurone est issue de la page Wikipédia : https://en.wikipedia.org/wiki/Long_short-term_memory

10. Les schémas suivants ont été trouvés sur le site https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (consulté le 06 juillet 2020)

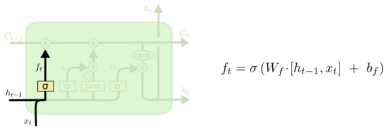

FIGURE 1.4 – Porte d'oubli - LSTM - source github colah

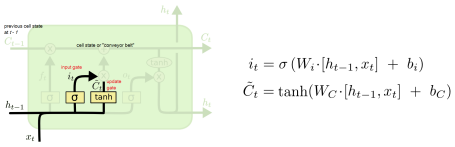

FIGURE 1.5 – Porte d'entrée - LSTM - source github colah

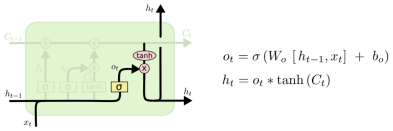

FIGURE 1.6 – Porte de sortie - LSTM - source github colah

On reconnaît tout de même deux types de fonctions d'activation : la sigmoïde (la fonction de répartition de la loi logistique.) [11], ainsi que la tangente hyperbolique [12]. Pour rappel :

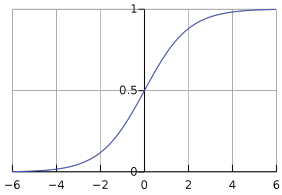

FIGURE 1.7 – Sigmoïde|Fonction Logistique - source Wikipédia

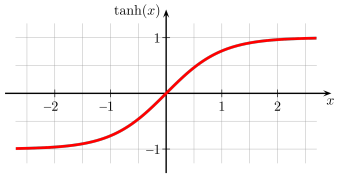

FIGURE 1.8 – Graphe de la fonction Tangente Hyperbolique - source Wikipédia

Il est donc important de comprendre l'usage de ces deux fonctions d'activation. Comme les figures 7 & 8 le montrent, la sigmoïde est à valeurs dans $[0\,;1]$, tandis que la tangente hyperbolique dans $[-1\,;1]$. On peut expliquer cette différence en décrivant la sigmoïde comme une expression de l'importance, c'est-à-dire que les valeurs de la fonction pour les vecteurs (ou matrices) d'entrée dans le réseau vont définir l'importance de conservation de l'information. En d'autres termes, si la $i^{éme}$ composante de σ est proche de 0, l'information est à oublier. À l'inverse pour une valeur à 1, il est important de conserver cette composante. Pour la fonction de tangente hyperbolique, cela correspond aux coefficients de description de l'information, c'est la partie du réseau qui apprend la quantification de l'importance de l'information (plus ou moins proche de 0), et le sens de corrélation (positive ou négative).

La combinaison de ces deux fonctions d'activation va nous donner la couche du réseau. Concentrons nous alors sur la porte d'oubli. On a deux entrées, qui sont composées de nos données à l'instant t, noté x_t, la sortie de la couche précédente h_{t-1}, et ce que l'on appelle la mémoire des couches antérieures C_{t-1}.

On applique la fonction σ à la concaténation des entrées h_{t-1} et x_t (on regarde donc l'importance relative de la sortie de la couche précédente par rapport aux nouvelles données d'entrées). La sortie de la fonction est ensuite multipliée par la mémoire C_{t-1}, ce qui nous permet d'oublier une partie des données (celles dont le résultat de la fonction sigmoïde est proche de 0). Prenons maintenant la porte d'entrée, on applique les fonctions σ et tanh, à la concaténation de h_{t-1} et x_t, ce qui nous donne les importances relatives ainsi que le poids respectif.

On multiplie le résultat des transformations obtenues par ces deux fonctions d'activation, et on obtient ce que l'on peut appeler la mémoire naïve à l'instant t, noté $\tilde{C}$. De ce résultat, ainsi que de ce que l'on récupère de la mémoire de la couche précédente, on somme les éléments, auxquels on applique une tangente hyperbolique, pour l'attribution des poids. On transforme en même temps, par une sigmoïde, la concaténation de nos vecteurs

11. Le graphe de la fonction logistique est issu du site Wikipédia : https://fr.wikipedia.org/wiki/Fonction_logistique_(Verhulst)
12. Le graphe de la fonction tangente hyperbolique est obtenu grâce au site Wikipédia : https://fr.wikipedia.org/wiki/Tangente_hyperbolique

(matrices) d'entrées h_{t-1} et x_t. La multiplication des résultats de ces deux computations, nous donne la sortie de notre couche du réseau à l'instant t. On a le détail des formules sur chacune des figures représentant les portes. Mais l'expression de chaque activation est donnée par :

$$f_i = A_i(W_i\,[h_{t-1}, x_t] + b_i) \tag{1.11}$$

où W_i sont les poids, b_i le biais, et A la fonction d'activation de la porte de la couche. Parfois on prend le vecteur de mémoire C à la place de la concaténation des vecteurs $[h_{t-1}, x_t]$.

"G.R.U." :

Le nombre élevé de transformations pour gérer l'importance de l'oubli dans les modèles du type L.S.T.M. a conduit à la création d'une version allégée. Le G.R.U. a été introduit pour simplifier les réseaux de neurones du type L.S.T.M.. On retrouve la dualité des fonctions d'activation σ et tanh. L'astuce vient de l'implémentation d'un nouveau critère de décision formulé :

$$\Gamma_u = \sigma(W_u\,[C_{t-1}, x_t] + b_u) \tag{1.12}$$

Qui nous permet de sélectionner la partie importante de la mémoire avec :

$$C_t = \Gamma_u * \tilde{C}_t + (1 - \Gamma_u)C_{t-1} \tag{1.13}$$

On représente la couche du neurone avec le schéma suivant à l'aide d'une image tirée de Wikipédia [13]

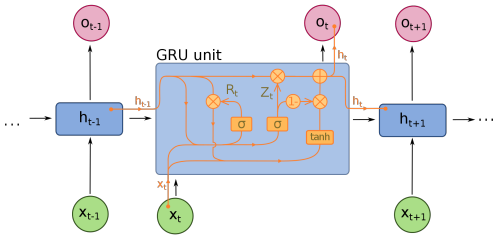

FIGURE 1.9 – Représentation d'une couche d'un G.R.U. - source Wikipédia

Malgré l'optimisation des réseaux "G.R.U.", ce type de réseaux de neurones est très lents à entrainer, et demande un imposant corpus de texte. On peut citer des modèles pré-entrainés tels que Google [14], spaCy [15], gluon [16] ou même N.L.T.K. [17]. Généralement, ces modèles utilisent des sources de données libres comme celles de Wikipédia [18], ou Twitter [19]. Mais l'apprentissage automatique des langues est toujours un sujet stratégique et au début des années 2010, une équipe va être à l'origine de transformations majeures dans ce domaine.

1.4 Google Brain

En 2011, Google Research, le pôle recherche et développement de Google, crée Google Brain, une équipe de chercheurs composés de Jeffrey Dean, Greg Corrado et Andrew Ng. C'est de ce laboratoire de recherche

13. L'image a été librement récupérée du site https://en.wikipedia.org/wiki/Gated_recurrent_unit consuté le 07 juillet 2020
14. Modèle Google disponible sur le site : https://radimrehurek.com/gensim/ (consulté le 08 août 2020)
15. Modèle spacy disponible sur le site : https://spacy.io/usage/vectors-similarity (consulté le 18 avril 2020)
16. Modèle gluon disponible sur le site : https://gluon-nlp.mxnet.io/examples/word_embedding/word_embedding.html (consulté le 08 août 2020)
17. Modèle NLTK disponible sur le site : http://www.nltk.org/nltk_data/ (consulté le 02 mars 2020)
18. Lien vers la base de données Wikipédia : https://dumps.wikimedia.org/
19. Lien vers le site développeur de Twitter : https://developer.twitter.com/en

qu'émerge ce qui deviendra l'outil de traitement du langage qui, aujourd'hui, est reconnu comme état-de-l'art ("state-of-art). On a tout d'abord, en juin 2017, l'introduction des transformers dans l'article "Attention Is All You Need", et l'année suivante, en février, la mise en pratique des encodeurs bi-directionnelles (ELMo). Puis le rassemblement de ces deux résultats abouti à la création de B.E.R.T, en octobre de la même année.

1.4.1 Les Transformers

Le problème de lenteur des réseaux de neurones récurrents (R.N.N.), et donc des réseaux L.S.T.M. et G.R.U., vient de l'obligation de traiter les informations séquence par séquence. Avec la puissance de calcul des G.P.U. [20], on a l'opportunité de travailler en parallèle avec des séquences entières. C'est de cet outil technique que sont apparus tous les traitements d'apprentissage profond et la démocratisation des tenseurs [21]. Les Transformers apparaissent dans le texte "Attention Is All You Need"[18]. On a la structure du modèle :

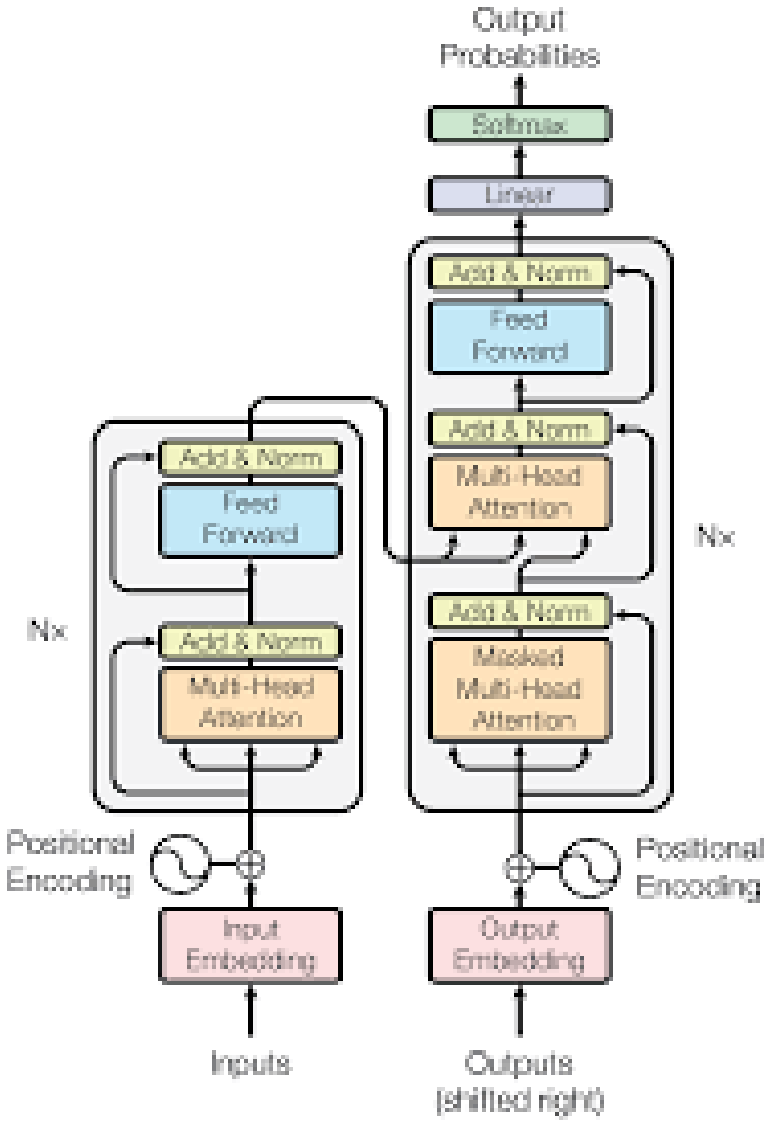

FIGURE 1.10 – Représentation d'un Transformer - source "Attention Is All You Need"

On repère ici, les couches parallélisées à l'endroit du *multi-head attention*, mais les innovations ne se situent pas seulement à cet endroit. L'aspect remarquable de ce traitement, vient de l'attention. On crée une couche du

20. Processeur graphique
21. Application multilinéaire

réseau qui va définir l'importance des mots dans la phrase.
On intègre cette couche qui nous permet de définir un réseau de neurones à convolution (C.N.N.), qui possède lui aussi la particularité d'être calculé parallèlement. La sortie de ce réseau nous donnera des valeurs "d'importance" des mots dans la phrase, mais également des valeurs "d'importance" d'un mot par rapport à un autre. Par exemple, dans la phrase : "*Ce film était extraordinaire.*" Les mots "*film*", et "*extraordinaire*", possèdent une grande importance dans la compréhension de la phrase (en comparaison par exemple du mot *Ce*), mais il y a également un lien fort direct entre ces deux mots ("*film*" <-> "*extraordinaire*"). Les Transformers permettent la mise en vecteurs de ces relations.
Les positions des éléments dans la phrase représentent alors un biais qu'il faut arriver à prendre en compte. C'est pour cela qu'un encodage de position est déclaré en entrée de l'encodeur et du décodeur.
On s'attarde un peu sur ce processus, dans l'article "Attention Is All You Need"[18]., l'encodeur de position nous est donné tel quel avec la justification que même si on utilise des méthodes d'apprentissage successives pour définir les valeurs de la matrice qui va nous servir pour coder le positionnement, on n'obtient pas de meilleur résultat que celui obtenu par cette matrice.

$$P_{i,j} = \begin{cases} sin(\frac{i}{10000^{\frac{j}{d_{model}}}}) & \text{si j pair} \\ cos(\frac{i}{10000^{\frac{j-1}{d_{model}}}}) & \text{si j impair} \end{cases} \quad \text{où i est la position du mot et j la position relative à i} \tag{1.14}$$

Pour résumer, on applique à plusieurs reprises à nos données d'entrée ces mécanismes d'attention. Le réseau s'entraine en calculant la probabilité de retrouver l'objet masqué en sortie. Dans tout ce processus les réseaux de neurones récurrents disparaissent.

1.4.2 B.E.R.T.

Dans ce chapitre, on dissèque l'encodeur bidirectionnel mis en place par les équipes de Google, et qui, depuis octobre 2019[5] (pour sa version anglaise et décembre en français) fait partie des outils d'aide à la compréhension des requêtes sur le moteur de recherche le plus célèbre. Il y a 3 notions importantes à définir lorsqu'on travaille avec B.E.R.T. : on intègre un marqueur de début et de fin de phrase, ainsi qu'un masque. Donc lorsqu'on alimente B.E.R.T. pour son apprentissage, on a la phrase suivante :
"[DEBUT] Le chat de Charlotte [MASQUE] le lait [FIN]"
Par convention, on parle plus de séparateur que de fin, parce qu'il peut arriver que notre séparateur soit autre chose que la fin d'une phrase. Un des travaux de B.E.R.T. sera d'être capable de savoir, le cas échéant, si les éléments suivants le jeton [SEPARATEUR], constitue bien la suite logique de notre texte.
Par exemple :
"[DEBUT] Le chat de Charlotte [MASQUE] le lait [SEPARATEUR] Il peut en [MASQUE] un bol par jour [SEPARATEUR]"
Pour justifier de cet apprentissage, on ajoute un élément de caractérisation qui définit pour chaque séparateur le caractère continu ou non du texte (en anglais, c'est une variable label qui prend les valeurs IsNext ou NotNext).
À la fin de son entrainement, B.E.R.T. est capable de traduire des textes, de répondre à des questions, de résumer des textes,... Mais est également capable de réaliser de la classification parfois mieux que des modèles spécifiques.
B.E.R.T. commence, ainsi, par vectoriser les mots qui lui sont fournis en entrée. On peut se servir des vecteurs

obtenus par Word2Vec, par exemple. On y ajoute l'encodeur de position défini par les Transformers, on obtient alors une matrice d'entrée de taille (la taille de la séquence * la dimension de la vectorisation). C'est ce qui constitue l'entrée de notre réseau.

La première étape consiste à décomposer chaque mot en 3 sous-éléments : Q, K, et V (une requête, une clé et une valeur), auxquels on associe une matrice de poids chacun, qui est évaluée par apprentissage. Donc chaque mot a ses propres matrices de poids apprises, et pour le calcul de l'attention du mot, on utilise alors un $softmax$:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1.15}$$

Qui nous donne des probabilités qui renforcent les mots pour lesquels clés et requêtes sont similaires. Une fois cette sélection faite, on utilise les attentions multiples en concaténant les résultats des attentions pondérées :

$$\begin{aligned} MultiHeadAttention(Q, K, V) &= Concat(head_1, head_2, ..., head_h)W_0 \\ \text{où} \quad head_1 &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{1.16}$$

Pour ne pas apprendre uniquement sur les attentions, on ajoute une étape d'action anticipative (feedforward). La phrase qui résume le mieux cet ajout est : "Apprend les relations entre les éléments de la séquence, mais n'oublie pas ce que tu sais déjà à propos de toi." [22]. Cette action anticipée prend la forme d'une fonction d'activation redresseur linéaire (ReLU) formalisée par :

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{1.17}$$

On a donc ici parcouru les étapes de l'encodeur du Transformer (partie gauche de la figure 1.10).

La particularité du décodeur est alors la suivante.

Le principe d'auto-régression du Transformer nous donne pour chaque passage dans le réseau un mot prédit supplémentaire, jusqu'à ce que la prédiction soit la fin de la phrase. Mais puisque nous entrainons sur une séquence dont on connaît déjà tous les mots, il faut "forcer" le réseau à prédire au premier passage le jeton [DEBUT], puis au second passage le premier mot, et ainsi de suite (au $i^{ème}$ passage on prédit le $(i-1)^{ème}$ mot) jusqu'à la fin de la phrase. On doit appliquer ce que l'on appelle un masque sur les mots suivants. On fait donc en sorte que la matrice obtenue par le $softmax$ possède une moitié supérieure droite de 0, ce qui revient à forcer le produit QK^T à être égal à $-\infty$ pour les éléments au-dessus de la diagonale.

Ce traitement permet donc l'apprentissage de nouveau poids qui seront liés aux facilités|difficultés à prédire les éléments masqués.

Pour finir, on applique à nouveau un $softmax$ pour obtenir les probabilités de prédictions.

B.E.R.T. se sert des avancées de représentations vectorielles contextualisées, établies avec ELMo, ainsi que des mécanismes d'attention, pour créer un état de l'art ("state-of-art") en traitement automatique du langage, notamment grâce à la simplicité d'adaptation du modèle à des utilisations spécifiques comme l'analyse de sentiment.

22. Référence issu du site : https://lesdieuxducode.com/blog/2019/4/bert–le-transformer-model-qui-sentraine-et-qui-represente (consulté le 15 août 2020)

CHAPITRE 2

L'APPLICATION MAKEZU

Dans ce chapitre, on se concentrera sur la description de l'outil de recherche de prospect développé par Makezu en collaboration avec l'école CentralSupElec, des détails de traitements et des améliorations de l'outil. L'idée est assez simple, il nous faut un outil de traitement du langage qui permet de détecter si un tweet exprime un besoin relatif au produit ou à la solution que propose l'entreprise qui est notre client.

2.1 Le produit Makezu

Makezu est un ensemble de deux applications :

— L'une, codée entièrement en Python, à l'aide du framework [1] Flask. Il nous fallait construire une application web, qui puisse être requêtée par n'importe quelle plateforme. Nous n'avons pas besoin de développer un site web de bout en bout, il nous faut uniquement un serveur web, auquel on soumettra des requêtes assez fréquemment. Les interactions seront donc rapides et ponctuelles. C'est pourquoi, l'application a été développée avec *C.O.R.S.* [2], qui permet la gestion des plateformes d'origines multiples *A.J.A.X.* [3]. Elle est hébergée sur une instance A.W.S. [4], qui est un serveur simple sans puissance de calcul graphique (On ne se servira donc de tenseurs et de deep learning qu'en cas de changement d'architecture). On utilise un serveur libre web nginx, pour la rapidité de traitement des requêtes H.T.T.P.. Pour ce qui est du service d'enregistrement des données, on utilise un autre service cloud disponible avec A.W.S., une base de données NoSQL, DynamoDB, qui est l'équivalent du gestionnaire Apache MongoDB.

— La seconde est une application web [5] sans ligne de code, qui ne demande pas de développeur (nocode). Elle est développée grâce à l'outil Bubble, une technologie importante pour les fondateurs de Makezu. Il permet la génération d'une application web dynamique, très esthétique, que l'on crée à l'aide d'une

1. Un framework est une bibliothèque construite dans un langage spécifique (ici, Python), qui permet de développer un produit informatique spécifique.
2. Cross-Origin-Resources-Sharing textuellement Partage de ressources aux origines multiples
3. Asynchronous JavaScript and XML, technique côté client qui permet la gestion de requête asynchrone, envoi-réception de requête sans création d'interface visible
4. Amazon Web Service, plateforme de services distants (cloud)
5. Lien vers l'application : https://app.makezu.io/

interface visuelle (W.Y.S.I.W.Y.G. [6]) intuitive. C'est sur cette application que les utilisateurs naviguent et utilisent l'outil. Elle permet la gestion des appels avec l'application hébergée sur A.W.S.. Un exemple ici avec la page tableau de bord pour un utilisateur, qui reprend quelques chiffres sur les campagnes qu'il a lancées (Le nombre de tweets envoyés, le nombre de clics, ...).

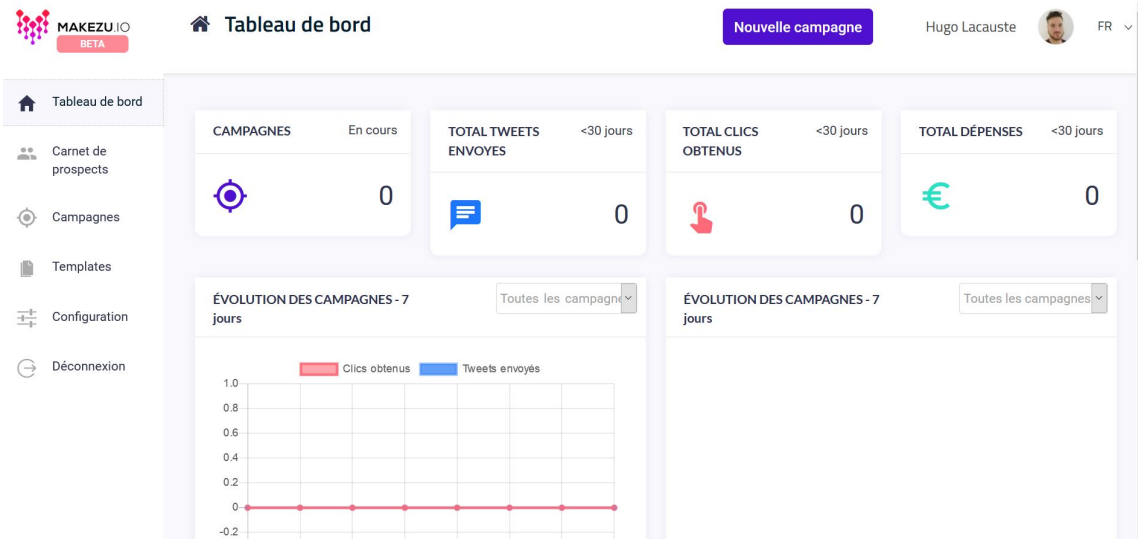

FIGURE 2.1 – Tableau de bord utilisateurs application Makezu

Pour finir, Makezu possède un site vitrine [7], qui expose notre application et permet de comprendre l'outil en lui-même, permettant aux visiteurs de s'inscrire sur la plateforme.

2.2 L'application Flask-C.O.R.S.

Nous allons nous intéresser plus particulièrement à l'outil en arrière-plan, qui concerne le traitement automatique du langage. Pour comprendre le fonctionnement de l'application, je prends le parti de suivre le parcours des interactions de l'utilisateur, tout au long du processus.

Les prochaines étapes vont être décrites sur un exemple en français, il faut également noter que le même schéma échoie en anglais.

Tout d'abord, comme dans toute application l'utilisateur s'enregistre en renseignant une adresse mail et un mot de passe. On lui demande l'autorisation de se connecter avec son compte Twitter, ce qui va nous permettre de communiquer avec le réseau social en qualité de l'utilisateur [8]. Une fois connecté, on lance une campagne, en choisissant parmi 3 profils de recherches :

- Un produit : on cherche les tweets qui expriment un besoin sur un produit ou service particulier (une voiture, une aide ménagère).
- Un intérêt : on cible une communauté de gens qui parle d'un sujet commun (intérêt pour un sport ou une série particulière).
- Un moment de vie : on choisit parmi des moments de vie représentatifs (mariage, naissance, décès).
- Une description : on récupère les personnes qui s'expriment sur leurs particularités (métiers, préférences alimentaires).

6. What You See Is What You Get en anglais ce que vous voyez est ce que vous obtenez.
7. Lien vers le site : https://makezu.io/
8. Ce type de connexion nous permet d'utiliser le compte Twitter de l'utilisateur de Makezu.

Lors de mon arrivée dans l'entreprise, seuls deux de ces champs étaient disponibles : la recherche de besoins exprimés sur un produit, ou sur un moment de vie. Cependant, les moments de vie n'étaient pas très efficaces, et ont étés rapidement abandonnés. La recherche d'intérêt pour un produit, en revanche, récupérait des tweets de manière satisfaisante, nous allons donc nous concentrer sur ce procédé.

Tout d'abord, l'utilisateur choisit un ensemble de verbes qui vont définir le type de besoin recherché (ex : "acheter", "louer", "réparer", "apprendre", "vendre"). Ensuite, on identifie le produit ou service (ex : "voiture", "mathématiques", "appartement"). L'ensemble de ces informations est alors enregistré dans la base de données A.W.S., dans la table *"Campagnes"*. Pour l'identification de tweets, on effectue alors l'appel à l'API de Twitter.

L'application commence par récupérer les informations de la campagne. Il y a en base les verbes de bases ("basic verbs") qui sont déjà renseignés ("vouloir", "chercher", etc), on récupère les verbes de compléments et on concatène chacun des verbes de chaque liste ensemble.

Exemple : verbes_basiques["veux", "aimerais"] + verbes_complements["acheter", "trouver"] = ["veux acheter", "veux trouver", "aimerais acheter", "aimerais trouver"]

On se rend alors compte que la multiplication de ces verbes va impliquer une augmentation importante du nombre de requêtes finales.

Une fois les verbes concaténés, on y ajoute les mots-clés, la cible du besoin, ainsi que le pronom personnel singulier "je" ou "j'". Ce qui nous crée une requête comme ["je veux acheter voiture", "je veux trouver voiture", ...]. On envoie cette requête à l'API de Twitter qui nous récupère les 100 tweets (au maximum) qui correspondent à chacune des requêtes ainsi créées.

Une fois cette récupération effectuée, on applique un modèle pré-entrainé (sur le corps de texte de Wikipédia) de traitement automatique du langage, c'est un module Python du nom de spaCy [9]. Cette librairie permet un bon nombre de traitements sur des données *texte*. Ce dont on va se servir, c'est ce que l'on appelle l'étiquetage des parties de textes (P.O.S. [10]), ainsi que la gestion des dépendances entre les mots d'une phrase (ce traitement a été rendu particulièrement efficace par le développement des transformers et des encodeurs bidirectionnels).

On applique une règle de décision qui va nous permettre de classer les tweets suivant si on les considère bons ou mauvais.

On peut résumer l'ensemble de ces interactions avec le schéma suivant :

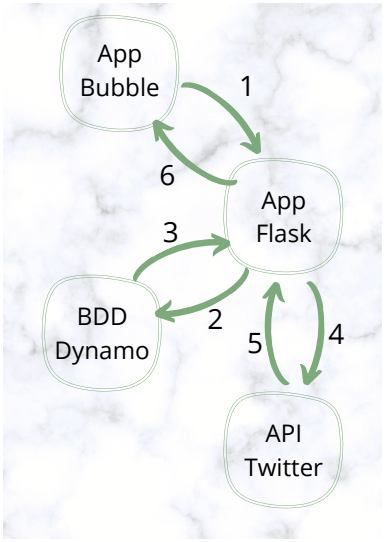

FIGURE 2.2 – Chemin d'une campagne

1	Création de la campagne
2	Enregistrement des détails de la campagne dans la base
3	Récupération des détails de la campagne
4	Envoi des requêtes à l'API de Twitter
5	Récupération des tweets
6	Tri et envoi des tweets classés positifs

TABLE 2.1 – Explication des relations

9. En annexe B se trouve un descriptif un peu détaillé de ce module.
10. Part-Of-Speech tagging c'est l'étiquetage morphosyntaxique.

Les requêtes entre l'application Bubble et l'application Flask (hébergée sur A.W.S.) sont des requêtes H.T.T.P. standards. Lors de la création d'une campagne, ou lors d'une demande de récupération de tweet, le corps de la requête est un fichier J.S.O.N., qui donne les caractéristiques de la demande.
Ces requêtes sont envoyées grâce au gestionnaire de connexions aux applications de Bubble. Pour ce qui est des enregistrements dans la base, on se sert d'un module Python pour communiquer avec les services cloud d'Amazon (boto3). Pour finir les appels à l'A.P.I. de Twitter sont aussi des requêtes H.T.T.P., elles sont cependant gérées par une librairie Python (tweepy), qui permet des interrogations simples par le biais d'une fonction. [11]

2.3 Les limites de l'A.P.I Twitter

La première amélioration apportée à notre outil vise à contrecarrer les erreurs de "time out" (délai épuisé), et de "rate limit" (limite atteinte), qui sont, on va le voir, assez liées. On commence par la plus aisée à appréhender, la limite atteinte. En effet, l'A.P.I. de Twitter définit des restrictions sur ce que l'on peut récupérer. Toutes les 15 minutes ces compteurs sont remis à 0. Ces limites sont de :

nombres de requêtes	18
nombres de tweets retournés	1500
nombres d'utilisateurs retournés	300
nombres de connexions simultanées	50

TABLE 2.2 – Limites de l'API de Twitter

On a donc un souci par rapport aux nombres de requêtes que l'on peut envoyer, et la multiplication de combinaisons que l'on effectue semble incompatible avec une telle restriction.
La seconde erreur est le délai épuisé. Pour éviter une surcharge du serveur web, on considère que lors d'une requête H.T.T.P.., si le délai de réponse dépasse une minute, c'est que la requête n'est pas bonne. La connexion est donc rompue et on se retrouve avec une erreur de délai dépassé.
Ici, le temps de réponse est proportionnel au nombre de tweets qui passe par notre outil de traitement du langage. Donc, plus on récupère de tweets, plus on va mettre de temps à les analyser.
Une idée assez classique pour se débarrasser de ce genre de problème, c'est de faire en sorte d'envoyer les requêtes par séquence, et non pas en totalité avec un seul appel. Or, si on fait en sorte d'attendre le temps de régénération de nos droits d'accès à l'A.P.I. de Twitter, c'est-à-dire 15 minutes, on aurait alors une erreur de délai ("time out).
Il y a également un second désagrément, qui est le stockage des requêtes déjà passées, afin d'éviter de renvoyer toujours les mêmes.
Une première correction mise en place a été de limiter la multiplicité des verbes, et des mots-clés. Mais on se rend alors compte que l'on passe à côté de nombreux tweets potentiellement intéressants.
On a donc décidé de mettre en place un petit traitement en amont de l'envoi de la requête à Twitter : on se décide à faire un tirage aléatoire de 12 requêtes parmi l'ensemble des possibles. On ne prend pas 18, le nombre maximal de requêtes par tranche de 15 minutes, car ce nombre conduisait trop souvent à un délai expiré. On justifie ce choix par un calcul distributionnel. On possède un total de 12 verbes basiques, en fonction du type de

11. Pour un détail complet des librairies et des modules nécessaires à la création de l'application, veuillez vous référer à l'annexe A, pour la structure de la base de données ainsi que les enregistrements des données il faut se référer à l'annexe C.

besoin recherché, parmi les 6 que nous avons identifiés (vendre, acheter, recruter, louer, réparer, se former). Le type de besoin qui possède le plus de verbes compléments en possède 27, ce qui nous donne 324 combinaisons pour tous ces verbes, on tombe donc à un tirage d'une loi uniforme discrète sur $[1\,;\,43]$.
On arrive donc avec un certain nombre d'itérations, au parcours total des combinaisons possibles. Ce problème nous a permis de pouvoir mettre en place des appels plus fréquents à l'A.P.I.. On peut, en effet, envoyer des requêtes toutes les 15 minutes avec cette solution. De plus, nous avons pu mettre en place un marqueur qui nous permet de ne remonter que les tweets plus récents que ceux récupérés par une requête ultérieure, ce qui nous permet de ne pas remonter à plusieurs reprises le même tweet.

2.4 Le N.L.P. dans Makezu

Pour ce chapitre, on va se servir d'exemples pour illustrer le chemin parcouru par les tweets une fois retournés par l'A.P.I. de Twitter.
On a donc déjà créé la requête qui a été envoyée à l'A.P.I. de Twitter. On prend ici l'exemple de la requête "Je cherche à acheter une voiture". Une fois la requête envoyée à l'API de Twitter, pour chaque résultat, on crée un processus de sélection. Un exemple de tweet récupéré par cette requête est : "Hé l'ami, Je cherche à acheter une voiture d'occasion... Quels sites me recommandes-tu ? Merci :)". Un premier traitement vient de retirer toutes les émoticônes, les mentions d'autres comptes Twitter, et les liens hypertextes. Une fois ce nettoyage effectué, on fait appel au module spaCy [12], pour l'application de jeton sur notre tweet.
Le résultat de ce traitement est un étiquetage morphosyntaxique. Pour cela, on doit charger les modèles pré-entrainés, une liste est disponible sur le site spaCy [13]. Une fois le modèle chargé, on l'applique à notre texte, c'est-à-dire qu'il va être vectorisé, et une analyse syntaxique va nous donner les détails suivants :

12. Pour le détail de la librairie spaCy, veuillez vous réferer à l'annexe B
13. La liste des modèles spacy propres aux langues est disponible au site : https://spacy.io/usage/models (consulté le 16 avril 2020)

Text	Lemma	POS	Dep	Shape	alpha	stop
Hé	hé	CCONJ	vocative	Xx	True	True
l'	le	DET	det	x'	False	True
ami	ami	PROPN	nmod	xxx	True	False
,	,	PUNCT	punct	,	False	False
Je	je	PRON	nsubj	Xx	True	True
cherche	chercher	VERB	ROOT	xxxx	True	False
à	à	ADP	mark	x	True	True
acheter	acheter	VERB	xcomp	xxxx	True	False
une	un	DET	det	xxx	True	True
voiture	voiture	NOUN	obj	xxxx	True	False
d'	de	ADP	case	x'	False	True
occasion	occasion	NOUN	nmod	xxxx	True	False
...	...	PUNCT	punct	...	False	False
Quels	quel	ADJ	det	Xxxxx	True	True
sites	site	NOUN	nsubj	xxxx	True	False
me	me	PRON	iobj	xx	True	True
recommandes	recommander	VERB	parataxis	xxxx	True	False
-	-	PUNCT	punct	-	False	False
tu	tu	PROPN	obl	xx	True	True
?	?	PUNCT	punct	?	False	False
Merci	merci	NOUN	ROOT	Xxxxx	True	True

TABLE 2.3 – Résultats du N.L.P. avec spaCy

Text :	Le mot dans sa forme d'origine
Lemma :	Le mot dans sa forme basique
POS :	La forme morpho-syntaxique simple
(Tag :)	La forme détaillée du mondage (masculin|feminin, singulier|pluriel, temps de conjugaison, actif|passif).
Dep :	Dépendance syntaxique
Shape :	Le spectre du mot
alpha :	Si le mot ne possède que des lettres
stop :	Si le mot est un mot stop, ce sont les mots les plus fréquemment utilisés dans la langue

TABLE 2.4 – Explication des résultats spaCy

On a donc une explication complète de ce que sont les mots du tweet, leurs importances et dépendances.

On peut également afficher les dépendances sur un graphe avec les jetons (tokens).

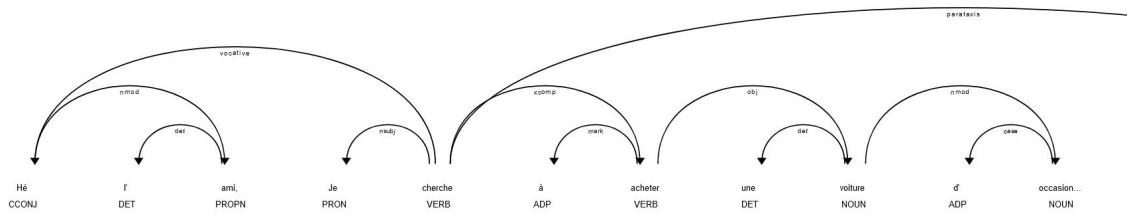

FIGURE 2.3 – Dessin des dépendances dans le début du tweet

C'est sur ces dépendances que va opérer notre règle de décision. On commence par la requête que l'on a soumise à l'A.P.I. de Twitter : "Je cherche à acheter voiture", et on regroupe les jetons qui constitueront notre groupe verbal : "cherche à acheter", on obtient alors :

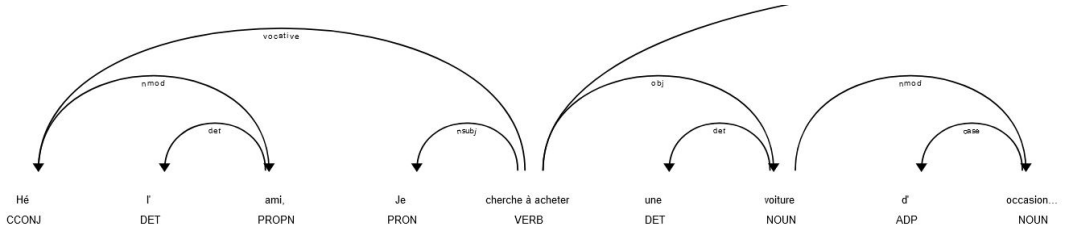

FIGURE 2.4 – Dessin des dépendances avec le groupe verbal

On se sert alors de la librairie Python networkX [14] qui permet la génération de graphe, et on trace alors le graphe des dépendances :

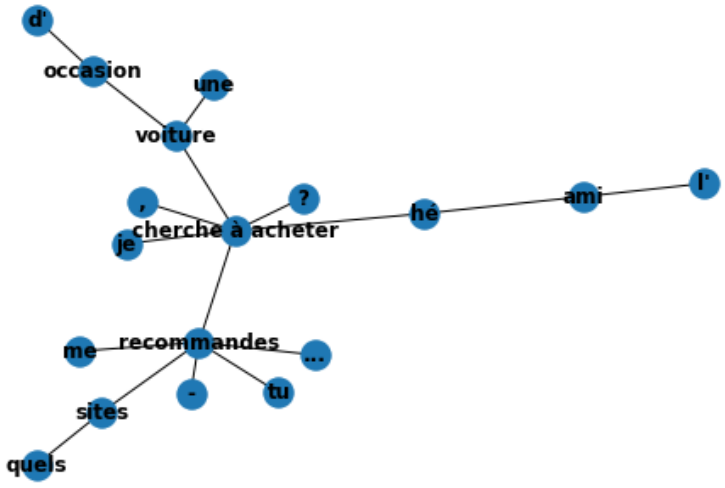

FIGURE 2.5 – Graphe des dépendances

Notre règle de décision est alors que la distance entre le groupe verbal et la cible produit soit ≤ 1. Dans notre cas, le tweet récupéré est donc classé positif.

La suite de ce rapport se concentrera donc sur les nouveautés apportées à notre outil sur les différents objectifs à atteindre.

2.5 La description et les audiences

Les retours d'expériences des testeurs de notre application sur les possibilités de l'outil nous ont amenés à penser deux nouvelles options à développer pour répondre à ce qui était attendu de Makezu. Ces deux besoins

14. Une liste des librairies utilisées et leurs descriptions est disponible en annexe A

sont regroupés dans le même chapitre car ils sont assez liés.
Une des volontés de Makezu est de pouvoir identifier les comportements des utilisateurs de Twitter, pour leurs proposer des mises en relation liées à leurs intérêts. C'est pour cette raison, qu'il nous est vite apparu qu'il serait important de relever les sujets autour desquels les utilisateurs du réseau social communiquent.
Dans ce chapitre, on va donc identifier deux champs d'action qui se focalisent sur des mots-clés qui seront au centre de nos recherches.

Objectifs et enregistrement des données

L'application Makezu stocke dans une base de données (Un détail de la base de données est disponible en annexe C), l'ensemble des tweets qui ont été récupérés par les différentes campagnes. Les données sont stockées avec : un id propre au tweet, le texte du tweet, une variable booléenne classified (qui informe si le tweet a été identifié comme adéquat), la langue, ainsi que les identifiants de campagne et d'utilisateurs de Makezu.
Dans un premier temps puisque notre méthode de récupération nous donne uniquement un tweet qui exprime clairement le désir d'un produit, aucune information supplémentaire ne nous est nécessaire. Cependant, avec les nouvelles demandes de nos utilisateurs, on a maintenant un intérêt à associer aux profils des utilisateurs Twitter remontés, la cible de la campagne.
On a donc mis en place un nouvel enregistrement dans la base, conservant cette fois-ci, l'identifiant de l'utilisateur, ainsi que le mot-clé. De cette manière, on pourra créer des listes d'utilisateurs aux centres d'intérêt communs sur lequel notre outil pourra apporter des solutions encore plus spécifiques. Par exemple, en regroupant par classes des individus proches, chaque caractéristique spécifique d'utilisateurs pourra être proposée au reste des membres de cette classe.

L'audience

On parle ici d'un type de recherche d'utilisateurs, uniquement autour d'un centre d'intérêt assez précis. Le résultat de la recherche en question ne demande pas vraiment de traitement du langage. On n'effectue aucun processus particulier de traitement après avoir récupéré les résultats de l'appel à l'A.P.I. de Twitter.
C'est un processus plus large, donc moins précis, et il est utilisé uniquement pour des usages particuliers. Par exemple, on s'intéresse ici à des tweets qui mentionnent une série télévisée, ou un club de sports. Il y aura peu d'intérêt pour nos utilisateurs de retrouver toutes les personnes qui parlent d'un sport ou de séries télévisées en général.
On effectue, tout de même, un traitement en amont de l'envoi : on propose à l'aide de notre application des mots dont le sens est proche. Pour ce faire, on se sert de l'implémentation de la base de données lexicale de l'université de Princeton WordNet, déployée sur Python par N.L.T.K., la boîte à outils en traitement du langage.
Allons donc un peu plus en profondeur sur l'outil WordNet. Cette base de données est née en 1998 dans le laboratoire de sciences cognitives de l'université de Princeton. Sur la page de la faculté [15] qui nous donne l'accès à cette base (elle est libre de droits), les contributeurs nommés sont : Christiane Fellbaum, Randee Tengi, Yan (Bill) Huang, Andrew ("Ferg") Ferg (PU '15), Collin Stedman (PU '15), Eric Tengi, Mark Tengi (PU '16), Gil Walzer (PU '16).
L'outil est maintenant hébergé au département d'informatique. Il est principalement développé en anglais, mais

15. Lien vers la page de l'outil sur le site de l'université : https://wordnet.princeton.edu/ (consultée le 7 juillet 2020)

par le biais du module N.L.T.K., d'autres langues sont disponibles [16].

Il permet la mise en relation des mots dans une même langue en fonction de leur signification. Pour cela, il utilise une classe Python, que l'on appelle *Synsets*. Cet objet répertorie le mot, les différents sens dans lequel le mot peut être employé, les synonymes de ce mot, les antonymes, ainsi qu'une définition pour chacun des sens du mot. Pour une précision sur le mot en particulier, on appelle l'attribut *synset* de la classe *Synsets*. Cet attribut prend trois options d'identifications de la forme suivante : "dog.n.01" avec le mot, le plongement lexicographique et le numéro, parmi les définitions possibles de la base de données WordNet.

L'interface WordNet pour Python de N.L.T.K. [17] permet ainsi de lister les synonymes des mots soumis à l'outil. On soumet donc cette liste à notre utilisateur pour enrichir les possibles mots-clés que l'on soumettra à l'A.P.I. Twitter.

En voici un petit exemple d'utilisation et les sorties associées :

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('dog')
[Synset('dog.n.01'), Synset('frump.n.01'), Synset('dog.n.03'), Synset('cad.n.01'),
Synset('frank.n.02'), Synset('pawl.n.01'), Synset('andiron.n.01'), Synset('chase.v.01')]
>>> print(wn.synset('dog.n.01').definition())
a member of the genus Canis (probably descended from the common wolf) that has been domesticated by
    man since prehistoric times; occurs in many breeds
```

L'auto-description

Pour l'auto-description, on reprend exactement le traitement dont on s'est servi pour l'expression d'un besoin envers un produit ou un service. La seule grande différence vient du fait que le groupe verbal sera composé du seul verbe *être*.

Par ce mode de recherche, on veut récupérer des utilisateurs de Twitter par rapport à leur façon de s'exprimer sur un sujet en particulier. Le but de ce traitement et de pouvoir identifier précisément les intérêts exprimés sur un sujet.

Pour reprendre des exemples classiques, le football, on aimerait arriver à identifier si l'utilisateur parle du fait de pratiquer le sport ou bien uniquement s'il parle du sport comme d'un sport qu'il regarde avec enthousiasme. De cette façon, on peut arriver par exemple à récupérer des groupes d'utilisateurs qui partagent des points communs et obtenir des bases de données qui soient une liste d'utilisateurs pour chaque description que l'on a établie.

Donc comme pour la recherche de produit, on crée une requête qui reprendra les termes d'un tweet qui exprimera un intérêt pour une pratique, par exemple : "Je suis végétalien". On envoi donc la requête à l'A.P.I. de Twitter et on effectue le tri sur les résultats de la requête.

16. C'est langues sont l'albanais, l'arabe, le catalan, le mandarin chinois, le danois, l'anglais, le basque, le perse, le finlandais, le français, le galicien, l'hébreu, l'indonésien, l'italien, le japonais, le norvégien, le polonais, le portugais, l'espagnol, et le thaïlandais, la liste des code I.S.O. 639 est disponible en détail à https://www.loc.gov/standards/iso639-2/php/code_list.php
17. Dont on peut avoir une prise en main sur le site : https://www.nltk.org/howto/wordnet.html

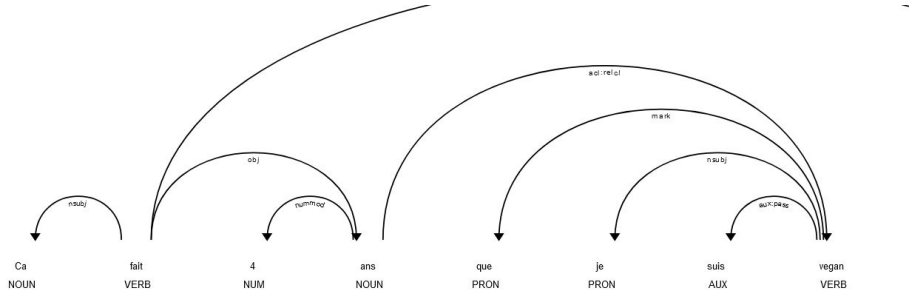

FIGURE 2.6 – Tweet d'auto-description

On utilise à nouveau la construction d'un graphe pour appliquer notre règle de sélection.

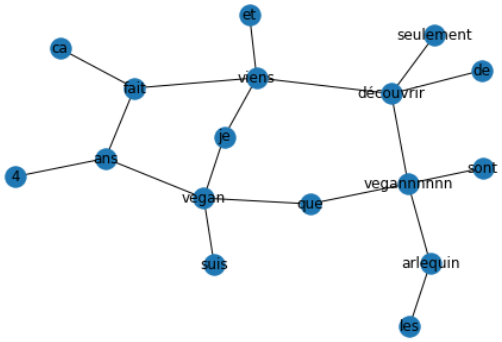

FIGURE 2.7 – Graphe auto-description

Grâce à ce procédé, on obtient alors les tweets qui définissent un utilisateur par rapport à la manière qu'il a de parler de ses préférences.

CHAPITRE 3

LES DONNÉES

Dans ce chapitre, on s'intéresse aux données que nous avons eues à traiter dans nos modèles et dans l'application Makezu. Il s'agit donc des textes des tweets sur lesquels nous allons travailler pour obtenir des informations sur la manière dont les besoins sont exprimés. On en profitera pour rappeler quelques règles sur les données des réseaux sociaux.

3.1 L'étiqueteur automatique de données

Dans cette section, on aborde un outil que nous avons développé chez Makezu.
Il s'agit d'une application Python, qui nous permet de classer manuellement des tweets parmi une liste définie de classes. Cet outil a été notre principale source d'alimentation de données pour la création de nos modèles. Il nous a permis de bâtir des outils automatiques de A à Z, et de faire de l'apprentissage par transfert (transfer learning), ou du réglage fin (fine-tuning) sur des modèles existants. C'est-à-dire que nous avons ajouté la spécificité de nos données sur des modèles déjà entrainés, pour illustrer nos propres caractéristiques.
L'application se présente de la façon suivante, tout d'abord, on vous demande le nombre de tweets que vous êtes prêt à traiter, ensuite chacun des tweets s'affiche à l'utilisateur qui appuie sur le bouton correspondant au type de besoin, pour finir un message de remerciements apparaît. La sortie de cet outil est un fichier *.csv*, qui est composé de 4 variables : l'identifiant du tweet, le texte du tweet, la date et le label sur lequel il a été marqué.

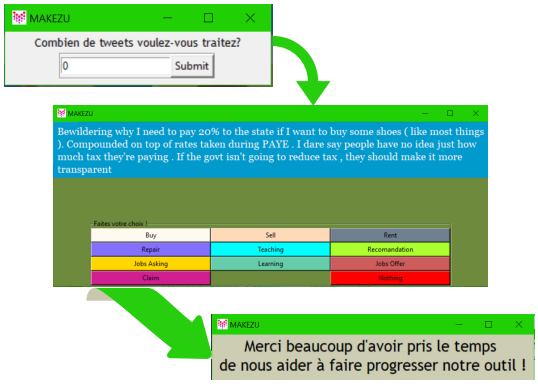

FIGURE 3.1 – Etiqueteur besoin exprimé

On peut comparer cela à l'outil de l'entreprise explosionA.I., prodigy [1].
Avec Python, il a même été possible d'en faire une application portable sur chacun des ordinateurs de l'équipe, pour que l'ensemble des travailleurs de Makezu puisse aider au renforcement de notre outil. On a donc un exécutable que l'on transfert sur n'importe quel P.C., et lorsqu'on lance l'application, un fichier *tweets.csv* est créé pour l'enregistrement du travail.
L'idée première était un outil de reconnaissance du type de besoin exprimé, parmi la liste ci-dessus. Mais il nous a servi d'étiqueteur automatique pour tous les domaines dans lesquels on a eu besoin de données avec un label. Dans un cadre plus simple, il nous fallait uniquement sur des requêtes spécifiques, déterminer si les tweets, qui nous étaient renvoyés, étaient conformes à ce que l'on attendait de notre requête.
On a le modèle spécifique des moments de vie :

FIGURE 3.2 – Etiqueteur moment de vie

Cet outil nous a permis d'obtenir beaucoup de données rapidement annotées, sans avoir de biais cognitif trop important lié au fait d'étiqueter automatiquement les tweets pour une requête vraiment spécifique à l'A.P.I. de Twitter. Notre idée de base était de questionner Twitter avec des phrases longues et spécifiques pour automatiquement ranger les tweets, cependant, on aurait subi le biais de notre requête dans notre règle de classification. Le fait de multiplier les utilisateurs nous a permis d'éviter les biais cognitifs individuels.
Pour finir spécifiquement sur les moments de vie, on peut récupérer par l'A.P.I. de Twitter, les tweets des 7 derniers jours qui répondent à la requête. Une fois par semaine, on peut relancer les étiqueteurs, qui sont déjà paramétrés pour chacune des requêtes spécifiques, et donc récupérer tous les tweets pour améliorer encore plus notre outil de classification.
Cet outil nous a permis de traiter des milliers de tweets pour l'expression des besoins et plus de 600 tweets pour chacune des requêtes de moments de vie.

3.2 La vie privée et les données sur Twitter

Dans cette partie, on aborde la particularité de l'utilisation des données personnelles sur les réseaux sociaux. Il m'est apparu important d'intégrer dans mon rapport de stage une section qui concerne l'éthique dans notre utilisation des données, notamment pour celles qui proviennent d'internet. Le cours proposé par Madame Gagnou nous a permis de nous sensibiliser sur ce type de problématique, et il me semble important que ses enseignements soient visibles dans nos rapports de stage.
Il faut tout d'abord savoir que le 27 janvier dernier la C.N.I.L. à fait paraître un communiqué pour rappeler à

1. Lien vers l'outil : https://prodi.gy/

partir de quand on peut identifier une donnée comme étant personnelle. Le communiqué est disponible sur le site de la C.N.I.L. [2]. Et pour ce qui concerne Twitter, il faut savoir qu'un pseudonyme peut être utilisé pour identifier des personnes, et est donc considérée comme donnée personnelle toutes données rattachées à un pseudonyme.

Puisque les tweets que nous récupérons et que nous envoyons sont des réponses à des tweets associés à un pseudonyme, notre base de données de l'application côté utilisateur est constituée de données personnelles provenant des milliers d'utilisateurs Twitter. La divulgation, ainsi que l'utilisation de celles-ci sont donc régies par la C.N.I.L.. Pour ce qui est de l'enregistrement de notre base de données hébergée en distant sur des serveurs Amazon, seule la table *Audience* [3] est constituée de données personnelles, les autres données ne possèdent pas de marqueur personnel. On a donc pris le parti, pour le travail de nos modèles, d'effectuer un traitement d'anonymisation des données, et lorsque nous faisons une récupération des données via l'étiqueteur automatique, nous ne conservons en aucun cas les données d'identification des utilisateurs.

Cette clarification m'a paru importante à la suite de la lecture du livre "Amour sous algorithme"[6] de Judith Duportail. Dans cet écrit, la jeune femme raconte comment elle a récupéré ses données personnelles de l'application Tinder, et fût surprise de recevoir un dossier de centaines de pages reprenant toutes les interactions qu'elle a eues avec l'application. Ce qui est intéressant à comprendre dans la construction de ce travail, ce n'est pas qu'il est naïf de penser que nos interactions ne sont pas enregistrées (tous les utilisateurs d'outils gratuits savent que leurs utilisations sont soumises à une observation), mais la quantité de données enregistrées ainsi que les utilisations faites de celles-ci sont souvent obscures et questionnent les utilisateurs.

Ce chapitre sert donc également de clarification sur l'usage que nous avons des données. Les données *texte* sont donc toutes traitées anonymement pour l'enrichissement de nos modèles et la création d'outils de détection de besoins exprimés. Les données personnelles stockées sont donc uniquement un carnet d'adresses pour entrer en communication avec vous, en ciblant (avec la table audience), les centres d'intérêt que vous auriez exprimés dans vos tweets.

Il m'est important de clarifier le "paradoxe de la vie privée", qui est la constatation que nous sommes de plus en plus critiques à l'idée que des personnes puissent se servir de nos données personnelles, tout en continuant d'en inonder internet par nos actions sur les réseaux sociaux. Je pense qu'il est important pour des étudiants ou des professionnels de la donnée d'avoir une sensibilité sur ces sujets.

2. Lien vers le comuniqué sur le site de la C.N.I.L. : https://www.cnil.fr/fr/identifier-les-donnees-personnelles
3. Pour le détail de ces informations, veuillez vous réferer à l'annexe C.

CHAPITRE 4

LES MOMENTS DE VIE

Pour la recherche d'utilisateurs qui correspondent à un moment de vie, nous avons décidé d'établir nous-même une liste de moments, ainsi qu'une liste de requêtes associées. Les moments de vies sont au nombre de 10, et on a environ 50 requêtes.

mariage	déménagement	premier emploi	recherche emploi	études supérieures
7	9	9	10	8
retraite	grossesse	décès	départ étranger	diplômes
6	10	6	6	6

TABLE 4.1 – Moments de vie et nombre de requêtes associées

Beaucoup de ces requêtes étant très spécifiques, le traitement du langage n'était pas nécessaire. Le nombre de tweets ne correspondant pas à notre recherche n'étant que de 1 mauvais tweet sur des centaines de bons.
On a créé un modèle sur 9 requêtes spécifiques, récupéré environ 600 tweets pour chacune des requêtes, avec une récupération hebdomadaire sur 4 semaines.
On a pu observer que les modèles s'amélioraient au fur et à mesure que le nombre de données augmentait. Il serait donc intéressant de poursuivre la récupération de tweets pour la mise à jour de nos modèles.

4.1 Vectorisation & approche non supervisée

Nos jeux de données sont le résultat de l'étiqueteur automatique pour toutes les requêtes qui ont nécessité la création d'un modèle. On se concentre sur les résultats pour une seule requête, et on consigne dans un tableau récapitulatif final, les valeurs des mesures de qualité de nos modèles.
Notre jeu de données est donc le résultat de la requête "nouvel appartement" auprès de l'A.P.I. de Twitter sur quatre semaines, elle est constituée de 690 tweets, dont 323 classifiés non conformes. On a donc une répartition équilibrée des tweets [1].

1. C'est-à-dire que le nombre de bons(367) est proche du nombre de mauvais(323)

La première étape pour le travail avec des données *texte*, c'est la vectorisation.
Dans notre cas, on démarre avec deux types :

— T.F.-I.D.F.

— spaCy tokenization

Comme on l'a vu au premier chapitre, le T.F.-I.D.F. parcourt l'ensemble des documents du corpus, ici les tweets de notre jeu de données, et attribue une valeur à chaque mot de vocabulaire.
On calcule alors la fréquence relative d'apparition des mots dans les tweets, que l'on redimensionne à l'aide d'un *log* en base 10, pour manipuler des chiffres plus facilement.
Le résultat de ce procédé pour nos trois premiers tweets nous donne un tableau avec les 690 tweets en lignes et les 2095 mots du vocabulaire en sortie, on a ici le vecteur réduit aux valeurs non nulles des deux premiers tweets :

tweet 1		tweet 2	
137	0.0834701206110526	137	0.06334123249315037
512	0.1820964326444259		
		681	0.26875937027061403
		708	0.16673825345501422
		929	0.18292827407565496
		1207	0.13269481724429869
1244	0.5664126846969743		
1288	0.11588201739651104	1288	0.08793697375723891
1351	0.2758376639309025		
		1421	0.27122218044913365
		1550	0.37228459756906446
1575	0.532861066396495		
		1717	0.3227745133223842
1944	0.2311336698426096		
1945	0.2863595604578741		
1992	0.36079045541314475		
		2010	0.7216719375548293

TABLE 4.2 – Résultats T.F.-I.D.F. sur deux tweets

On effectue donc une concaténation des résultats pour obtenir un tableau de données de taille (690, 2095).
La tokenisation de spaCy se sert des modèles de vectorisation du style Word2Vec, avec les mondages de mots et l'utilisation de l'attention pour la création de vecteurs. En appliquant le modèle pré-entrainé, la vectorisation se projette sur un espace de 96 dimensions. On obtient donc un tableau de données de taille (690, 96). On se propose de regarder une esquisse des 4 premiers tweets véctorisés :

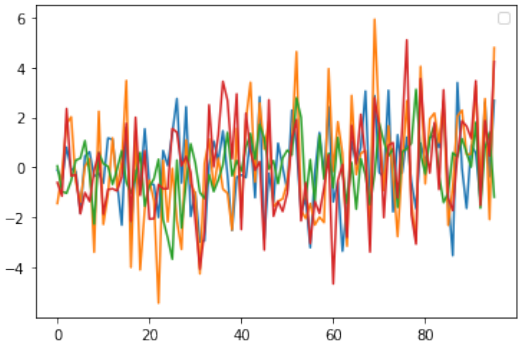

FIGURE 4.1 – Tweets & vecteurs spaCy associés

Texte du tweet	Label
pas une vie de rechercher un nouvel appartement à nantes	0
vite vite plus quune semaine et jai enfin mon nouvel appartement	1
je vais bientôt déménager je pense que cela me servira à équiper mon nouvel appartement	1
aujourdhui jai les clés de mon nouvel appartement	1

TABLE 4.3 – Tweets et leurs labels

Cette représentation nous sert uniquement à essayer de se rendre compte des valeurs visuellement. On se propose de jeter un œil aux **20** premiers tweets, pour lesquels on observe l'attribut de classe.

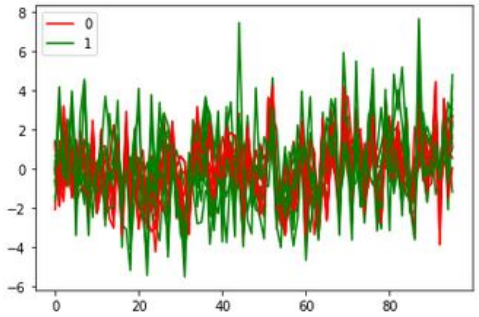

FIGURE 4.2 – Graphes bon|mauvais 20 premiers tweets vecteurs spaCy

Visuellement uniquement, il nous est impossible de trouver une règle de classification qui pourra nous permettre d'isoler les tweets bons des mauvais. On procède tout de même à une analyse sur les composantes principales pour nos deux modèles de vectorisations.

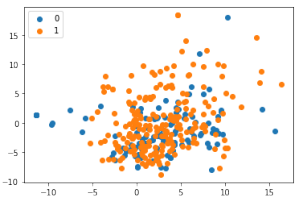

FIGURE 4.3 – Projection sur les composantes 1 & 2 - spaCy

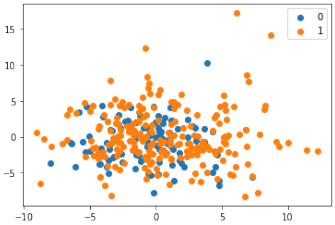

FIGURE 4.4 – Projection sur les composantes 3 & 4 - spaCy

	Composante 1	Composante 2	Composante 3	Composante 4
Variance Expliquée	3.27739225e-01	1.33598505e-01	7.98150208e-02	7.20109367e-02
Somme Variance Expliquée	3.27739225e-01	4,6133773-01	5,411527508e-01	6,131636875e-01

TABLE 4.4 – Variances expliquées sur les composantes - spaCy

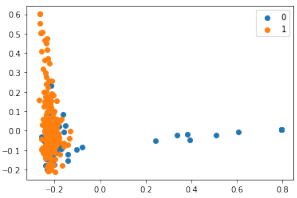

FIGURE 4.5 – Projection sur les composantes 1 & 2 - T.F.-I.D.F.

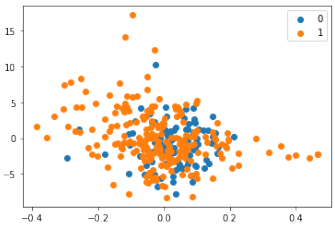

FIGURE 4.6 – Projection sur les composantes 3 & 4 - T.F.-I.D.F.

	Composante 1	Composante 2	Composante 3	Composante 4
Variance Expliquée	1.84818978e-01	1.73596913e-02	1.36426445e-02	1.15593192e-02
Somme Variance Expliquée	1.84818978e-01	2,021786693e-01	2,158213138e-01	2,27380633e-01

TABLE 4.5 – Variances expliquées sur composantes - T.F.-I.D.F.

Visuellement la vectorisation par T.F.-I.D.F. nous apporte des petites précisions sur des projections qui sépareront les bons tweets des mauvais. Cependant, la variance est plus vite expliquée par les composantes déduites des vecteurs obtenus par le modèle spaCy.
En plus de ces analyses en composantes principales, nous avons tracé des dendrogrammes pour l'observation de valeurs couperets pour la mise en place d'une règle de classification. Mais de la même façon qu'avec les A.C.P., les observations n'apportaient rien de concluant.

4.2 Vérifications d'hypothèses

On attaque donc maintenant les algorithmes de classifications supervisés, en commençant par les modèles linéaires. On a donc quelques hypothèses à vérifier. Tout d'abord, dans les règles basées sur un modèle, on a la famille des modèles bayésiens naïfs, qui demandent une hypothèse d'indépendance des variables. Vu le nombre élevé de variables utilisées pour la vectorisation, il est peu probable que l'hypothèse d'indépendance soit vérifiée. Cependant, on se propose de réaliser un test du χ^2 d'indépendances des variables aléatoires (des dimensions de la projection vectorielle des tweets). On obtient les résultats suivants :

	Statistiques	p-value
spaCy	0.9985717535018921	1.46754722449945e-22
T.F.-I.D.F.	0.06101804971694946	0.0

TABLE 4.6 – Résultats des tests du χ^2

On a également calculé les corrélations entre chaque variable, et on se rend compte que nombre d'entre elles se situent proches de 1 ou -1, ce qui valide le rejet de l'hypothèse d'indépendance.
Bien que l'hypothèse ne soit pas acceptée (ce qui arrive fréquemment), on utilise tout de même le bayésien naïf. D'autres modèles linéaires demandent une distribution gaussienne comme hypothèse de départ sur les données. On effectue alors un test de Shapiro-Wilk de distribution gaussienne, et on a décidé de tracer le graphe quantiles-quantiles pour toutes les variables et pour nos deux modèles de vectorisations :

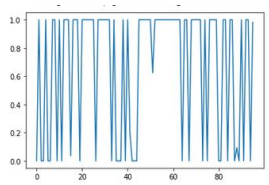

FIGURE 4.7 – Résultats des tests Shapiro-Wilk vecteur spaCy

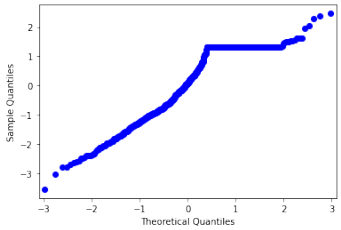

FIGURE 4.8 – Graphe quantiles-quantiles 1er vecteur spaCy

On a une variable sur deux, peut-être plus, qui, au vu des p-valeurs du test de Shapiro-Wilk, rejette l'hypothèse de distribution gaussienne, et lorsqu'on regarde les tracés des diagrammes quantile-quantile sur la première variable, on se rend compte que le tracé est linéaire, ce qui sous-entend une distribution gaussienne.

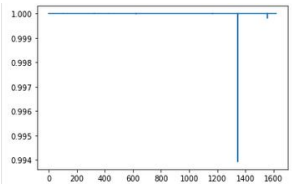

FIGURE 4.9 – Résultats des tests Shapiro-Wilk vecteur T.F.-I.D.F.

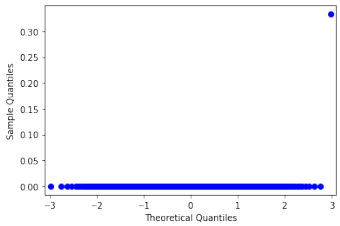

FIGURE 4.10 – Graphe quantiles-quantiles vecteur T.F.-I.D.F.

Dans ce cas-là, il y a potentiellement une variable qui nous donne une p-valeur pour le test de Shapiro-Wilk qui pourrait nous faire conclure à une distribution gaussienne. Si on observe le tracé du premier diagramme quantiles-quantiles, on se rend compte de la distribution : beaucoup de 0 et quelques valeurs non nulles. Cela vient du fait que la construction vectorielle extraite de nos données *texte* implique la création de nombreux 0. Bien que les hypothèses ne semblent pas validées (cela arrive lorsqu'on utilise le bayésien naïf où les hypothèses sont fortes), on utilise tout de même ces modèles.

4.3 Les mesures de qualités

Dans cette section on aborde les mesures de performances de nos différents modèles. On comparera toujours ces mesures pour se fixer sur le modèle le plus performant. C'est une liste rapide reprenant les calculs que l'on effectue pour obtenir la mesure.

Erreur d'apprentissage

La plus classique, la plus rapide, et celle qui nous permettra de juger aussi du sur-apprentissage éventuel. On divise notre jeu de données en deux sous-ensembles : un ensemble d'apprentissage et un de test. On entraine notre modèle sur les données du jeu d'apprentissage, puis on calcule les prédictions du modèle sur ce même jeu de données. On calcule le nombre d'erreurs de prédiction et on divise par le total.

$$Erreur_{apprentissage} = \frac{\text{Nombre d'erreur de prédiction sur le jeu d'apprentissage}}{\text{Cardinal de l'ensemble d'apprentissage}} \tag{4.1}$$

Justesse

Une mesure de qualité d'un modèle largement utilisé, et classique à prendre en compte, est l'erreur de test ou justesse ("accuracy"). Il revient au même calcul que l'erreur d'apprentissage, à l'exception du fait que l'on

calcule les prédictions sur le jeu de données test (celui qui ne nous a pas servi pour la création du modèle de classification).

$$Erreur_{test} = \frac{\text{Nombre d'erreur de prédictionsur le jeu de test}}{\text{Cardinal de l'ensemble de test}} \tag{4.2}$$

Précision et Rappel

On introduit ici la matrice, ou tableau, de confusion pour une règle de classification binaire.

Caractère de classification	prédiction positif	prédiction négatif
vrai valeur positif	vrai positif	faux négatif erreur de type 2
vrai valeur négatif	faux positif erreur de type1	vrai négatif

TABLE 4.7 – Matrice de confusion

On a deux formules, pour la précision et le rappel, qui permettent de juger de la qualité de notre règle de classification, et de détailler si elle est plus à même de juger les positifs ou négatifs.

$$Précision = \frac{\text{Vrais Positifs}}{\text{Vrais Positifs + Faux Positifs}} \tag{4.3}$$

$$Rappel = \frac{\text{Vrais Positifs}}{\text{Vrais Positifs + Faux Négatifs}} \tag{4.4}$$

Le F1-Score

La moyenne harmonique entre la précision et le rappel, le F1-score :

$$F1\ Score = 2 * \frac{\text{Précision} * \text{Rappel}}{\text{Précision + Rappel}} = \frac{\text{Vrais Positifs}}{\text{Vrais Positifs} + \frac{1}{2} * (\text{Faux Positifs + Faux Négatifs})} \tag{4.5}$$

Le 1 sert à avertir que l'on peut également appliquer une pondération, dans une situation où on voudrait privilégier les erreurs de premier ou de second type.

L'erreur L.O.O.

Dans ce calcul de mesures de performance de la règle de décision, notre jeu de données est découpé $n-1$ fois où n est le nombre d'éléments de notre jeu. On laisse de côté à chaque découpage un unique élément (leave-one-out : L.O.O.), et, pour chacun on prédit la valeur laissée de côté sur un modèle créé avec les autres observations.

$$Erreur_{LOO} = \sum_{1}^{n} \text{Erreur de prédiction} \tag{4.6}$$

Validation K-Folds

On découpe notre jeu de données k fois. Pour chacun des découpages, on calcule les erreurs de prédictions de la partie non prise en compte pour la création du modèle et on caclule la moyenne des erreurs de prédictions.

$$Erreur_{K-Folds} = \frac{1}{K} \sum_{1}^{K} \text{Erreur de prédiction} \tag{4.7}$$

La courbe R.O.C.

La courbe R.O.C. nous permet d'avoir une représentation graphique de la qualité d'une règle de décision, elle représente l'évolution du taux de vrais positifs sur le taux de faux positifs en fonction du seuil de décision. Une règle de classification inopérante donnerait une droite allant du point (0, 0) au point de coordonnées (1, 1) [2].

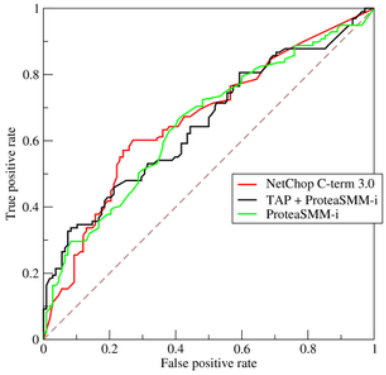

FIGURE 4.11 – Courbes R.O.C. - source Wikipédia

Coefficient de Matthews

Une autre mesure de qualité d'une classification binaire, est le coefficient de corrélation de Matthews. Cette mesure reprend la statistique de χ^2 pour un tableau de contingence de taille (2,2). Cette valeur est comprise entre [−1,1] : 1 étant la règle de classification parfaite, 0 signifiant que les prédictions sont aussi bonnes que la règle triviale de classification [3], et la valeur −1 pour des prédictions qui seraient toutes erronées.
Ce coefficient s'exprime de la façon suivante :

$$CoefficientMatthews = \frac{(\mathrm{VP} * \mathrm{VP}) - (\mathrm{FP} * \mathrm{FN})}{\sqrt{(\mathrm{VP}+\mathrm{FP})(\mathrm{VP}+\mathrm{FN})(\mathrm{VN}+\mathrm{FP})(\mathrm{VN}+\mathrm{FN})}} \tag{4.8}$$

Score de Jaccard

La dernière mesure de qualité pour nos modèles de classification binaire est le score de Jaccard.
C'est une mesure de similarité entre les objets, il est également utilisé en théorie de l'information comme un score de similarité entre deux mots.
On la définit de la manière suivante :

$$ScoreJaccard = \frac{\mathrm{VP}}{(\mathrm{VP}+\mathrm{FP}+\mathrm{FN})} \tag{4.9}$$

On a donc un ensemble de mesures de qualité qui vont nous permettre de juger et de choisir, parmi les différents modèles, lequel servira au mieux nos besoins.

4.4 Les modèles paramétriques

Dans ce paragraphe, on décrit brièvement les méthodes de classification utilisées ainsi que les résultats obtenus pour chacune d'entre elles.

2. Le graphe des courbes R.O.C. a été récupéré sur le site de Wikipédia : https://fr.wikipedia.org/wiki/Courbe_ROC
3. Choix de la classe majoritaire pour toutes les prédictions

Bayésien Naïf

Le bayésien naïf, bien que soumis à de nombreuses hypothèses, est une technique d'apprentissage supervisé qui fonctionne dans de nombreuses situations. Même lorsque les hypothèses ne sont pas toujours (presque jamais) vérifiées, d'où l'utilisation du mot naïf, cette méthode obtient de bons résultats.
C'est une approche indirecte, c'est-à-dire que l'on ne cherche pas à calculer la probabilité de distribution de nos variables, on cherche à calculer les probabilités sachant les valeurs de la variable qualitative.
On appelle cette méthode, bayésien naïf, car elle repose sur la formule de la loi a posteriori de Bayes, et la formule éponyme :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.10}$$

La construction du classifieur répond alors au problème d'optimisation sur la densité des lois a posteriori sachant la valeur de la variable cible Y.

$$g(x) = \underset{l\in\{1,\dots,K\}}{\operatorname{argmax}} \pi_l f(x|Y=k) \quad \text{où} \quad f(x|Y=k) \quad \text{densité de probabilité de } (X|Y) \tag{4.11}$$

Dans notre jeu de données, bien que l'hypothèse de normalité ne soit pas vérifiée, on va statuer sur l'inverse. Et on obtient les résultats suivants :

	App	Justesse	Précision	Rappel	F1-Score	LOO	10-Folds	Matthews	Jaccard
TFIDF	0.9718	0.7631	0.7619	0.6530	0.7032	0.2246	0.2796	0.5123	0.5423
scpaCy	0.7683	0.7982	0.7352	0.9090	0.8130	0.2637	0.2246	0.6151	0.6849

TABLE 4.8 – Tableau des résultats bayésien naïf gaussien

Analyse Discriminante

Là encore, les hypothèses de distribution gaussienne sur les variables sont obligatoires, et on cherche à estimer pour chacune des distributions des variables les espérances et variances de nos lois gaussiennes hypothétiques. On a tout de même un problème de colinéarité des variables, observé dans nos tests d'hypothèses. On peut donc tout de suite exclure l'analyse quadratique qui cherche à estimer la matrice de covariance de la loi multinomiale. On obtient tout de même les résultats pour une approche discriminante linéaire, qui répond au problème d'optimisation :

$$g(x) = \underset{l\in\{1,\dots,K\}}{\operatorname{argmax}} Q_l(x) \quad \text{où} \quad Q_l(x) = -\frac{1}{2}\log|\Sigma_l| - \frac{1}{2}(x-\mu_l)^T\Sigma_l^{-1}(x-\mu_l) + \log(\pi_l) \tag{4.12}$$

Dans le cas quadratique la matrice de covariance Σ est unique, il n'y a pas une matrice à recalculer pour chaque caractéristique l. Pour cela, il ne doit pas y avoir colinéarité des variables.

	App	Justesse	Précision	Rappel	F1-Score	LOO	10-Folds	Matthews	Jaccard
TFIDF	0.9978	0.8201	0.7435	0.8877	0.8093	0.1956	0.2057	0.6507	0.6796
scpaCy	0.9112	0.7675	0.7878	0.7090	0.7464	0.2086	0.2275	0.5354	0.5954

TABLE 4.9 – Tableau des résultats analyse discriminante linéaire

Machines à vecteurs de support

Les Machine à vecteurs de supports permettent de transférer les problèmes linéaires par des problèmes d'optimisation sur un panel plus large de fonctions. Il suffit pour cela de paramétrer la fonction noyau, qui déterminera quelles hypothèses correspondent le mieux à nos données. C'est un problème d'optimisation sous contraintes, qui se généralise en classification et régression suivant que la variable cible Y est à valeurs dans $\mathbb{Z}$ ou dans $\mathbb{R}$. L'équation simple à résoudre est le problème : $y = h(x)$.
$h(x)$ est alors la fonction qui séparera nos classes, le cas linéaire où $h(x) = 0$ nous donne un hyperplan séparateur.
On obtient pour les fonctions noyaux : euclidien, linéaire, polynomial (de degrés 2, & 3), et sigmoïde, les résultats suivants :

	Noyau	App	Justesse	Précision	Rappel	F1-Score	LOO	10-Folds	Matthews	Jaccard
TFIDF	Eucl	0.9913	0.8464	0.7368	1.0	0.8484	0.1623	0.1753	0.7337	0.7368
	Lin	0.9826	0.8552	0.7731	0.9387	0.8479	0.1507	0.1594	0.7245	0.7360
	Pol2	0.9956	0.8377	0.7259	1.0	0.8412	0.1782	0.1869	0.7206	0.7259
	Pol3	0.9978	0.8157	0.7	1.0	0.8235	0.2057	0.2144	0.6883	0.7
	Sig	0.9567	0.8596	0.7750	0.9489	0.8532	0.1507	0.1652	0.7349	0.7440
spaCy	Eucl	0.8549	0.8245	0.7611	0.9272	0.8360	0.1913	0.1985	0.6660	0.7183
	Lin	0.9329	0.7719	0.7685	0.7545	0.7614	0.2115	0.2420	0.5430	0.6148
	Pol2	0.8181	0.8201	0.7446	0.9545	0.8366	0.2028	0.2028	0.6680	0.7191
	Pol3	0.8203	0.8201	0.7346	0.9818	0.8404	0.2086	0.2086	0.6800	0.7248
	Sig	0.7380	0.7236	0.7422	0.6545	0.6956	0.2956	0.2855	0.4474	0.5333

TABLE 4.10 – Tableau des résultats machines à vecteurs support

Régression Logistique

On peut comparer la régression logistique à un bayésien naïf, où la distribution serait une Bernoulli.
On a la formule :

$$\mathbb{P}(Y = 1|X = x) = p \quad | \quad \mathbb{P}(Y = 0|X = x) = 1 - p \tag{4.13}$$

Avec l'hypothèse que la probabilité p soit une fonction logistique d'un score linéaire : $\sum \beta_i x_i$, et la fonction logistique :

$$f(u) = \frac{\exp(u)}{1 + \exp(u)} \tag{4.14}$$

On estime donc le score (les β) par log-vraisemblance conditionnellement à notre jeu de données. Cependant, comme il n'existe pas de calcul explicite du maximum de vraisemblance, on utilise des algorithmes pour l'estimation. Le plus utilisé de tous (celui que nous utilisons dans notre cas), est l'algorithme de Newton-Raphson.

	App	Justesse	Précision	Rappel	F1-Score	LOO	10-Folds	Matthews	Jaccard
TFIDF	0.9134	0.8464	0.7368	1.0	0.8484	0.1623	0.1739	0.7337	0.7368
scpaCy	0.9047	0.8114	0.8070	0.8	0.8036	0.2086	0.2159	0.6222	0.6717

TABLE 4.11 – Tableau des résultats régression logistique

Descente de gradient stochastique

On regroupe dans ce chapitre l'ensemble des fonctions coûts (d'objectif - loss function) pour lesquelles on utilisera l'algorithme de descente de gradient stochastique. On estime w pour la minimisation de la fonction de sommes :

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w) \tag{4.15}$$

Les i correspondent aux observations de la fonction en un point.

	Coûts	App	Justesse	Précision	Rappel	F1-Score	LOO	10-Folds	Matthews	Jaccard
TFIDF	hinge	0.9978	0.8289	0.750	0.9207	0.8266	0.1565	0.1753	0.6748	0.7045
	log	0.9978	0.8333	0.7560	0.9207	0.8303	0.1507	0.1594	0.6822	0.7099
	perc	0.9978	0.8289	0.7540	0.9108	0.8251	0.1811	0.1971	0.6719	0.7022
spaCy	hinge	0.9177	0.7280	0.7575	0.6637	0.7075	0.2275	0.2420	0.4589	0.5474
	log	0.9264	0.7324	0.7653	0.6637	0.7109	0.2347	0.2376	0.4683	0.5514
	perc	0.7640	0.6666	0.7846	0.4513	0.5730	0.2	0.2289	0.3650	0.4015

TABLE 4.12 – Tableau des résultats de descente de gradient stochastique

4.5 Les modèles d'ensembles

Dans ce chapitre on aborde les techniques de classification supervisée, qui sont des méthodes d'ensemble, qui ne demandent pas de vérification d'hypothèses, qui prennent tous types de variables pour établir des règles de classification.

On utilise la méthode du *Bagging*, qui est la concaténation de *Boostrap* et *Aggregating*, qui échantillonne plusieurs ensembles de données obtenus par tirage avec remises sur le jeu de données de base. On crée des modèles sur tous ces *ensembles boostrap*, et on concatène, *aggregate*, les résultats de ces modèles.

L'agrégation se fait par vote majoritaire ou par moyenne. C'est-à-dire que pour chaque élément de notre jeu de données, on analyse toutes les prédictions pour chacun des découpages. Puis, soit on prend la moyenne des prédictions, soit on compte le nombre de fois où la prédiction était de la classe k, et suivant laquelle des classes revient le plus souvent on statue sur cette prédiction.

K plus proches voisins

La méthode des k-plus-proches voisins est un processus en apprentissage supervisé. Elle sous-entend la définition d'une distance entre deux points dans notre espace des observations, et elle veut que la règle de classification soit le vote majoritaire parmi les k éléments les plus proches (au sens de la distance) de notre observation à prédire.

Puisque la création de nos vecteurs de texte nous fournit deux espaces vectoriels, on peut voir notre espace des observations comme un $\mathbb{R}^{96}$ pour la vectorisation avec spaCy et un $\mathbb{R}^{2095}$, pour la transformation T.F.-I.D.F..

On obtient donc les résultats suivants, pour 5, 10, et 25 voisins :

	V.	App	Justesse	Précision	Rappel	F1-Score	LOO	10-Folds	Matthews	Jaccard
TFIDF	5	0.8787	0.8552	0.7971	0.9565	0.8695	0.2420	0.1681	0.7249	0.7692
	10	0.8484	0.8596	0.7902	0.9826	0.8759	0.2014	0.1623	0.7414	0.7793
	25	0.8311	0.8552	0.7770	1.0	0.8745	0.2072	0.1724	0.7416	0.7770
spaCy	5	0.8528	0.7807	0.6846	0.9081	0.7807	0.2289	0.2333	0.5927	0.6403
	10	0.8354	0.7982	0.6884	0.9693	0.8050	0.2289	0.2347	0.6467	0.6737
	25	0.8333	0.7719	0.6619	0.9591	0.7833	0.2043	0.2072	0.6025	0.6438

TABLE 4.13 – Tableau des résultats k-plus-proches voisins

Arbres de décisions

Les arbres de décisions sont un outil de classification d'apprentissage supervisé très répandus car il n'y a aucune règle sur les données d'entrée pour la création de la règle de classification.

$$f(X) = y \forall X \tag{4.16}$$

Les variables peuvent être quantitatives ou qualitatives, tant qu'elles apportent de l'information sur la classe de notre observation. Elles sont prises en compte dans notre modèle pour être un lieu de séparation (nœud) entre deux sous-ensembles qui maximise l'indice de Gini (une autre mesure de section de nœud qui peut être utilisée est la mesure d'entropie.), qui mesure la pureté d'une division.

$$I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i) \tag{4.17}$$

On commence par observer les résultats de la construction d'un arbre de décisions sur les données :

	App	Justesse	Précision	Rappel	F1-Score	LOO	10-Folds	Matthews	Jaccard
TFIDF	0.9870	0.8333	0.8235	0.8521	0.8376	0.2260	0.1666	0.6669	0.7205
scpaCy	0.9891	0.7894	0.7049	0.8775	0.7818	0.2144	0.2362	0.5961	0.6418

TABLE 4.14 – Tableau des résultats arbres de décisions

Forêts aléatoires

Les forêts aléatoires en apprentissage supervisé, c'est la concaténation des arbres de décisions avec la méthode de sous ensemble. On utilise les données pour plusieurs créations d'arbres de décisions.

	App	Justesse	Précision	Rappel	F1-Score	LOO	10-Folds	Matthews	Jaccard
TFIDF	0.9978	0.8371	0.7954	0.9130	0.8502	0.1841	0.1710	0.6826	0.7394
scpaCy	1.0	0.7938	0.6861	0.9591	0.8	0.1971	0.2116	0.6353	0.6667

TABLE 4.15 – Tableau des résultats forêts aléatoires

Classifieurs ExtraTree

Ici, comme pour les forêts aléatoires, on entreprend plusieurs arbres de décisions sur des sous-sections de notre ensemble d'apprentissage de base. La différence vient de l'aléatoire dans la création des nœuds : on choisit aléatoirement la variable sur laquelle on va ordonner la règle de séparation des données.
On ne cherche pas sur quelle variable la section va augmenter la pureté d'un nœud mais on va tirer au hasard une variable, et faire la règle de séparation à cet endroit.
Dans le cas des forêts et au vu de la multiplication des arbres, cet aspect aléatoire va pouvoir nous faire gagner du temps de calcul sans perdre en qualité.

	App	Justesse	Précision	Rappel	F1-Score	LOO	10-Folds	Matthews	Jaccard
TFIDF	0.9978	0.8421	0.7615	1.0	0.8646	0.1971	0.1841	0.7204	0.7616
scpaCy	1.0	0.7587	0.6423	0.9897	0.7791	0.1927	0.1999	0.6012	0.6381

TABLE 4.16 – Tableau des résultats extra-forêts aléatoires

AdaBoost

Il s'agit comme le *boosting*, d'une méthode d'ensemble pour l'amélioration des classifieurs. Comme le *boosting*, on crée notre règle de classification sur des sous-sections de notre ensemble de base. Ici, les sous-sections sont faites de sorte que les éléments les plus difficiles à prédire sont tirés plus souvent. Le tirage avec remise est pondéré de sorte que les observations qui sont mal prédites ont plus de chances d'être tirées dans le prochain sous-ensemble de création de la règle.

	App	Justesse	Précision	Rappel	F1-Score	LOO	10-Folds	Matthews	Jaccard
TFIDF	0.9610	0.8245	0.8099	0.8521	0.83905	0.1956	0.1739	0.6498	0.7101
scpaCy	0.9481	0.7675	0.6667	0.9184	0.7725	0.1971	0.1999	0.5764	0.6294

TABLE 4.17 – Tableau des résultats AdaBoost forêts aléatoires

4.6 Synthèse des résultats

On se propose ici d'effectuer une synthèse des résultats sur une sélection de méthodes et avec quelques mesures de qualité de modèles pour prendre notre décision quant au modèle à mettre en place pour la sélection de nos tweets.
On peut tout d'abord remarquer que, pour notre travail de classification, on obtient de meilleurs résultats lorsqu'on utilise les techniques de vectorisation triviales de fréquences de termes. On peut expliquer cela par le fait que lors de l'apparition de certains termes, comme les négations, ou de prépositions comme "quand" pour l'utilisation du conditionnel, on se trouve dans des cas où le moment de vie n'est pas vraiment explicite. Donc la fréquence de termes nous apporte toute l'information nécessaire pour conclure sur nos classes.
On ne garde donc que les résultats pour la vectorisation T.F.-I.D.F..

TFIDF	Erreur-test	F1-Score	Erreur L.O.O.	Coefficient de Matthews
Extra forêts	0.8421	0.8646	0.1971	0.7204
SVM linéaire	0.8552	0.8479	0.1507	0.7245
SVM sigmoïde	0.8596	0.8532	0.1507	0.7249
SVM euclidien	0.8464	0.8484	0.1623	0.7337
Regression Logistique	0.8464	0.8484	0.1623	0.7337
10 plus proches voisins	0.8596	0.8759	0.2014	0.7414

TABLE 4.18 – Tableau de synthèses des résultats

On peut tout d'abord remarquer que les résultats de machines à support de vecteur avec noyau euclidien et de régression logistique sont en tout point égaux.
La fonction noyau euclidien est donnée par la formule suivante :

$$exp(-\|x - x'\|^2) \tag{4.18}$$

C'est la norme que l'on utilise pour le calcul des β_i de la probabilité a postériori des variables pour chacune des classes.
Une autre remarque intéressante est la proximité des résultats des S.V.M. pour une fonction noyau sigmoïde et linéaire. L'apport d'un type de fonction plus complexe (sigmoïde) n'apporte donc pas plus d'informations qu'un postulat linéaire.
On décide donc pour notre modèle d'implémenter un modèle de régression logistique sur une vectorisation T.F.-I.D.F.. On peut ainsi continuer chaque semaine à alimenter notre jeu de données annoté pour améliorer notre modèle et arriver à un jeu de données d'une dizaine de milliers de tweets.
Pour implémenter ce modèle dans notre application Flask, on se sert d'un outil de Python qui porte le nom de "pickle". C'est une librairie Python qui permet de sauvegarder en binaire n'importe quel objet Python. On a donc dans notre base de données stockée sur A.W.S., un champ booléen qui correspond à l'attribut "has_classifier" et dans les cas où cet attribut est vrai, on charge le modèle pré-entrainé T.F.-I.D.F. en régression logistique.
Cette petite astuce nous permet d'obtenir des traitements d'apprentissage automatique, dont seul le modèle est enregistré. On n'a donc pas besoin de charger les données à chaque traitement.

CHAPITRE 5

AMÉLIORATION DU CIBLAGE PRODUIT

Dans ce chapitre, on cherche à s'attaquer à une amélioration de notre outil de recherche d'utilisateurs de Twitter qui expriment un besoin. Tout d'abord, on aimerait ne pas subir le problème de ne récupérer qu'un certain nombre de tweets au maximum, limité par nos méthodes d'appel de l'A.P.I. de Twitter.
De plus, une volonté de Makezu est de prendre en compte les tweets autour d'un besoin exprimé de manière indirecte, plus implicite, dans des phrases qui ne ciblent pas vraiment le produit.
Par exemple, la phrase "J'ai encore l'air immonde sur cette photo", pourrait être récupérée par un vendeur de téléphone qui propose un superbe appareil photo, ou par un vendeur de produit de beauté.

5.1 Fichage automatique

Dans ce chapitre, on se propose d'étudier une idée pour récupérer l'ensemble des tweets qui exprimerait un besoin autour d'un sujet. On se propose de demander à Twitter des requêtes beaucoup plus larges. Ce qui nous permettrait de récupérer l'ensemble des tweets qui exprimerait un besoin d'achat, de vente, ou de location, ne renseignant aucun mot-clé cible, puis d'effectuer un traitement d'analyse morphosyntaxique étudiant la dépendance entre le verbe et l'objet et nous permettant de récupérer une liste de mots-clés cibles des besoins exprimés dans les tweets.
L'idée est d'obtenir une table de base de données avec la structure suivante :

id_tweet	text_tweet	label_tweet	date_tweet	user_name

TABLE 5.1 – Structure de la table pour le fichage automatique

On a donc analysé l'ensemble des résultats de requêtes sur plus de 10000 tweets pour observer toutes les dépendances entre le mot cible et le verbe. Prenons l'exemple du verbe "buy", pour formaliser ce processus de récupération.
On commence par donner quelques exemples de dépendances dans ces tweets :

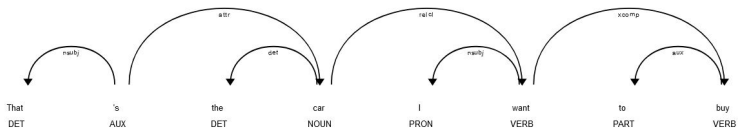

FIGURE 5.1 – Dépendances des mots clés avec le verbe buy

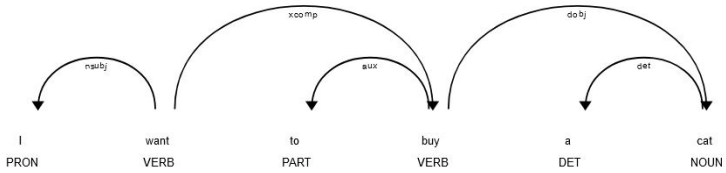

FIGURE 5.2 – Dépendances des mots clés avec le verbe buy - 2eme exemple

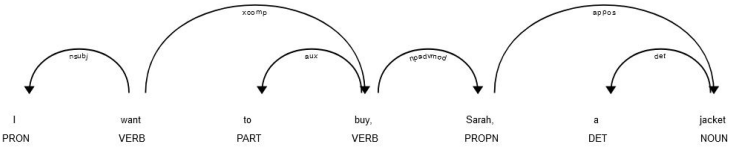

FIGURE 5.3 – Dépendances des mots clés avec le verbe buy - 3eme exemple

On a donc différents types de dépendances entre les verbes et le mot cible du besoin. Le but est donc d'intégrer chacune de ces dépendances dans un filtre qui ne récupèrera que le mot.

```
for token in doc:
   if token.text == "buy":
      for child in token.children:
         if child.tag_.find("NN") != -1 or child.tag_.find("VB") != -1:
            if child.dep_ == "dobj":
            babies = [baby for baby in child.children if baby.dep_ == "amod" or baby.dep_ ==
                "compound"]
            for baby in babies:
               keyword = keyword + " " + baby.text
               keyword = keyword + " " + child.text
            else:
               pass
         elif token.text == "need":
            for child in token.children:
               if child.tag_.find("NN") != -1 or child.tag_.find("VB") != -1:
                  if child.dep_ == "dobj":
                     babies = [baby for baby in child.children if baby.dep_ == "amod" or baby.dep_ ==
                         "compound"]
                     for baby in babies:
                        keyword = keyword + " " + baby.text
                        keyword = keyword + " " + child.text
                     else:
                        pass
```

On se rend déjà compte de la complexité du rendu lors de la mise au point de ce traitement même si une fois le modèle spaCy chargé, le temps de processus de l'ensemble de ces conditions est extrêmement rapide.
On obtient donc une liste de la taille : (le nombre de tweets qui sont passés par le filtre moins les quelques uns pour lesquels on n'a pas de dépendance), par exemple les tweets du type :

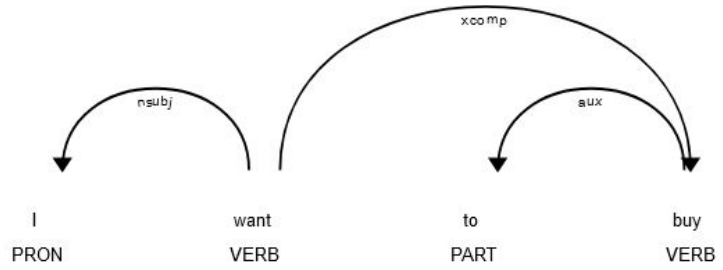

FIGURE 5.4 – Pas de dépendance avec le verbe buy

On a donc essayé d'effectuer un premier traitement sur cette liste, en regroupant les mots qui étaient présents à plusieurs reprises, et on a obtenu un tableau avec les mots-clés les plus présents et le compte de leurs occurrences, sur un ensemble de 10000 tweets :

Elément	Occurence
it	841
one	627
that	221
car	21
computer	14

TABLE 5.2 – Occurence des mots clés cibles

Le problème rencontré ici, est que l'on a beaucoup de mots qui apparaissent moins de 4 fois. On s'est donc proposé d'utiliser à nouveau la librairie Python N.L.T.K. et d'effectuer un regroupement par similarités sur les cibles que l'on a identifiées.
Dans l'implémentation WordNet de l'université de Princeton dans N.L.T.K., on a la possibilité d'utiliser des mesures de similarité entre les mots, en voici une petite liste :

- path_similarity : Une mesure basée sur la longueur du chemin issu d'un modèle de classification taxonomique [1] d'un mot et de ses synonymes|hypernymes à un autre et ses synonymes|hypernymes. C'est un score entre 0 et 1
- lch_similarity : Similarité de Leacock-Chodorow, qui mesure également la mesure du chemin, mais qui est réduite avec une fonction log. C'est une mesure ≥ 0
- wup_similarity : Similarité Wu & Palmer, encore une fois, on se sert de la longueur du chemin de texonomie, en prenant cette fois-ci en compte la profondeur de l'arbre créé.
- res_similarity : Similarité de Resnik, qui calcule la proximité de deux mots par le contenu d'information du premier mot commun. C'est une mesure ≥ 0.
- jcn_similarity : Distance de Jiang Conrath, reprend également les contenus d'informations du nœud ancêtre et des deux mots à comparer. $\frac{1}{(IC(s1)+IC(s2)-2*IC(lcs))}$

1. Modèle de classification similaire au dendrogramme de classification, très utilisé en sciences de la vie

— lin_similarity : Distance de Lin qui reprend les éléments de Jiang Conrath, seule la formule change. $\frac{2*IC(lcs)}{IC(s1)+IC(s2)}$

On se décide donc de prendre la similarité de chemin classique, et on définit le seuil de synonymie à un score de **0.3**, on prend un score plus large pour pouvoir s'il le faut le cas échéant refaire un tri a postériori.

On obtient alors un tableau de contingence avec :

Elément	Occurence
synset(it)	1876
synset(one)	1443
synset(car)	73
synset(computer)	229

TABLE 5.3 – Occurence des mots clés regroupés

On se rend compte que les mots tels que : "it", "that", "those", sont tous regroupés dans le *synset* des pronoms, sont très utilisés, et sont souvent des réponses à des tweets qui identifient un objet. Il faudrait donc effectuer un traitement pour trouver la cible de cette réponse pour pouvoir identifier la cible du besoin. On a également tous les chiffres "one", "two", "twice", "thousands", qui sont regroupés dans le même *synset*. Comme pour les pronoms, il faut effectuer un traitement pour se rendre compte de la cible du besoin d'achat exprimé.

Pour finir, le dernier résultat intéressant est autour du mot "computer", qui reprend tous les mots qui sont des outils technologiques. Il s'avère qu'il s'agit d'un domaine pour lequel nous avions un grand intérêt.

Par ce traitement on se rend tout de même rapidement compte que, premièrement on récupère uniquement les tweets qui expriment très clairement le besoin (on n'a pas d'amélioration sur les expressions moins directes de besoin), et deuxièmement que les résultats ne sont pas très aisés à utiliser.

Nous avons donc décidé de nous orienter vers d'autres méthodes pour l'optimisation de notre outil de récupération de tweets, en se servant notamment des résultats de notre étiqueteur automatique de tweets.

5.2 Modèle de B.E.R.T.

On l'a vu dans le premier chapitre, les modèles de B.E.R.T.[5] ont la particularité de pouvoir ajouter une couche supplémentaire dans le réseau d'apprentissage. Cela nous permet d'ajouter un traitement en plus pour une tâche spécifique.

Dans notre cas on cherche à créer un filtre qui prend en entrée un tweet et renvoie une valeur de probabilité que ce tweet exprime un besoin ou non.

On utilise l'implémentation en tenseur de B.E.R.T. sur Python par l'équipe de huggingface [2]. On doit donc créer un dossier comprenant notre vocabulaire, ainsi que la configuration de notre outil de transformation de texte en vecteur.

Ce dossier se crée automatiquement lors de l'apprentissage de notre modèle sur nos données.

Notre jeu de données est composé de **54.837** tweets, **22.934** expriment un besoin et **31.913** non. On se propose d'étudier tout d'abord la taille en nombre de mots de nos tweets :

2. Entreprise majeure du N.L.P., leurs équipes implémentent les solutions de Google dans Python https://huggingface.co/

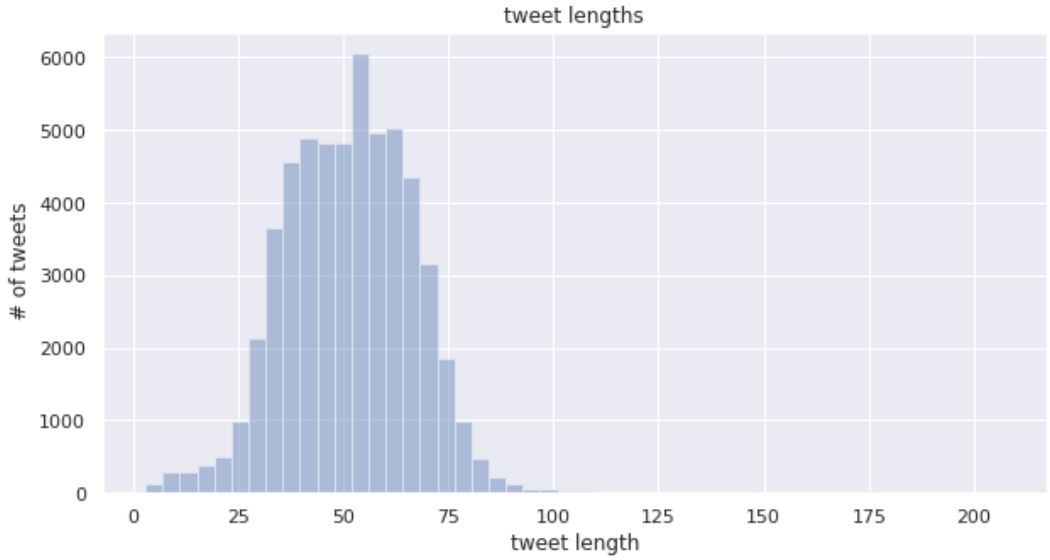

FIGURE 5.5 – Taille en nombre de mots des tweets

On remarque presque une distribution gaussienne de la taille de nos tweets.
On entraine donc notre réseau de neurones et on obtient les mesures suivantes de qualité de notre modèle.

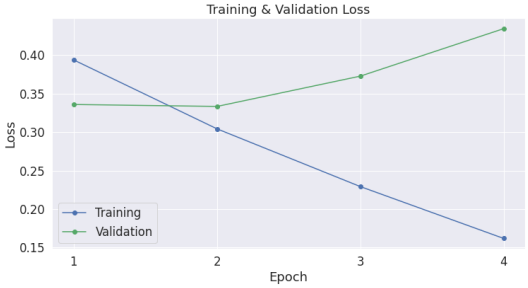

FIGURE 5.6 – Fonction de coûts sur nos jeux de données

On obtient rapidement un problème de sur-apprentissage, dès le moment où la courbe verte passe au-dessus de la courbe bleue, c'est-à-dire au bout de deux epochs, deux passages de notre jeu de données.
On applique alors notre modèle sur un jeu de données test qui n'a pas servi à notre modèle. C'est un jeu de données composé de **1532** tweets étiquetés. On obtient un taux d'erreur très faible de **44%**, ce qui signifie que notre modèle de classification n'apporte rien.

5.3 L'encodeur de phrases

Dans ce chapitre, on se propose d'utiliser un encodeur de phrases[2] qui est entrainé pour analyser les ressemblances sémantiques entre des phrases.
D'après la documentation disponible sur le site développeur de tensorflow qui fonctionne comme un hub [3], et qui promet la création de matrice de corrélation pour les ressemblances entre les phrases :

3. Une plateforme d'hébergement de code comme Github, source de nos deux figures des encodeurs de phrases : https://tfhub.dev/google/universal-sentence-encoder/1

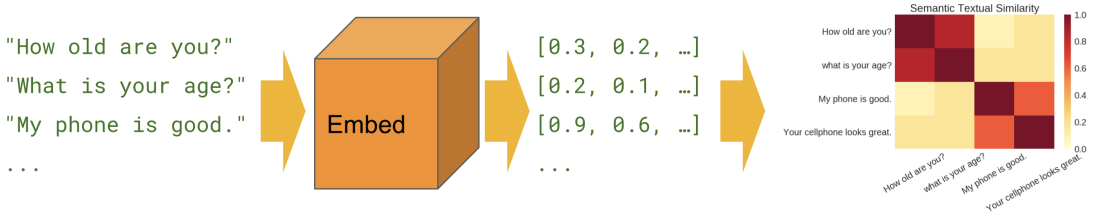

FIGURE 5.7 – Exemple de l'analyse de similarités - source TensorFlowHub

On se propose donc d'analyser un ensemble d'exemple de phrases dont on aimerait avoir des proximités pour savoir si on peut utiliser tel quel le modèle pré-entrainé.

On se sert de 8 phrases :

"Je cherche un vélo d'occasion" ; "J'aimerais acheter un vélo" ; "J'en ai marre de faire du vélo à Paris" ; "Je voudrais m'acheter une console" ; "J'aimerais acheter une voiture" ; "J'aimerais m'acheter un cabriolet" ; "Où puis-je trouver un bon vélo" ; "Comment allez-vous ?" ; "Ca va ?"

On obtient la matrice de similarités suivante :

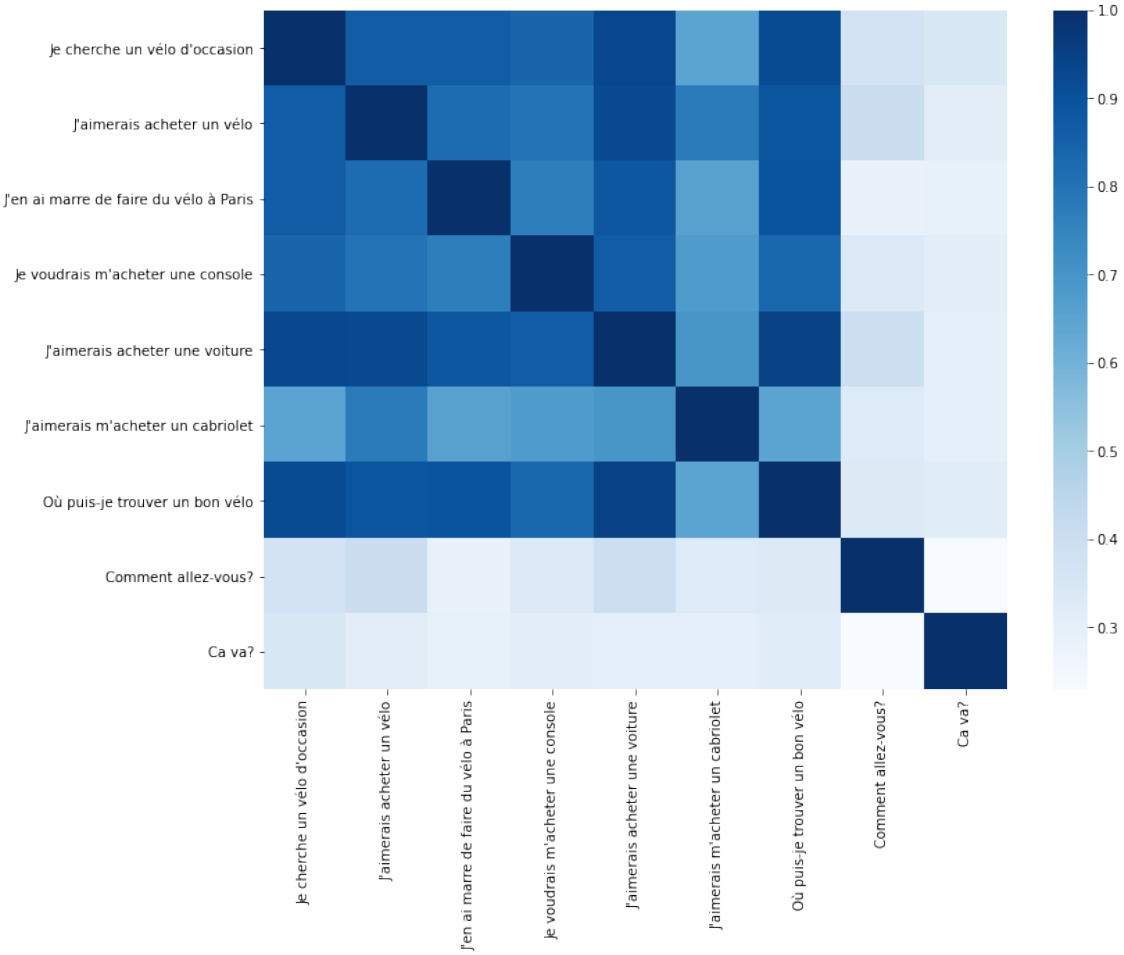

FIGURE 5.8 – Matrice des similarités

Et on se rend compte tout de suite que les 6 premières phrases sont toutes bien corrélées, on a donc un bon outil pour retrouver les phrases qui expriment un besoin. Cependant, on se rend compte qu'avec les deux dernières phrases qui sont deux questions qui expriment exactement la même chose, nous n'avons presque pas de similarité entre les deux phrases.

On va donc se servir de cet outil comme base pour la création de notre outil de filtrage. Car comme expliqué

dans la suite de la documentation, il est possible d'ajouter un élément de classification, et donc de faire du "fine tuning"[13] (du réglage fin de modèle.).

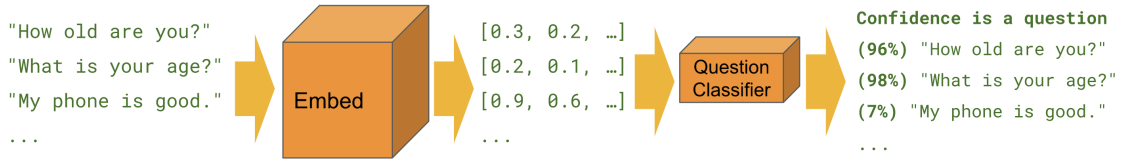

FIGURE 5.9 – Fine-tuning de l'encodeur de phrases - source TensorFlowHub

On entraine donc notre encodeur de classification sur un jeu de données annotées, fruit de notre travail collectif. On possède un jeu de données de **24488** tweets labelisés comme exprimant un besoin ou non. On effectue un entrainement du réseau sur 10 epochs (10 rappels à notre jeu de données), et on obtient les résultats suivants :

Train on 16406 samples

Epoch 1/10 : loss : 0.3387 - acc : 0.8499

Epoch 2/10 : loss : 0.2973 - acc : 0.8674

Epoch 3/10 : loss : 0.2808 - acc : 0.8774

Epoch 4/10 : loss : 0.2620 - acc : 0.8867

Epoch 5/10 : loss : 0.2425 - acc : 0.8968

Epoch 6/10 : loss : 0.2240 - acc : 0.9066

Epoch 7/10 : loss : 0.2038 - acc : 0.9172

Epoch 8/10 : loss : 0.1859 - acc : 0.9264

Epoch 9/10 : loss : 0.1683 - acc : 0.9373

Epoch 10/10 : loss : 0.1517 - acc : 0.9434

Et lorsqu'on applique nos résultats sur un jeu de données test :

Precision	Recall	F1-Score
1.00	0.34	0.51

TABLE 5.4 – Résultats de l'encodeur de phrases sur notre jeu de données test.

Il est donc nécessaire d'obtenir plus de données pour avoir un modèle de classification plus performant. Le fait d'avoir un si grand écart entre le nombre de données classées positivement **4683** et le nombre de données au total **24488**, peut nous conduire à ce type d'erreur. On se propose donc de revenir à ce type d'affinage de modèle lorsque nos données nous permettront un traitement plus efficace.

Conclusion :

Il est difficile de conclure un rapport de stage en entreprise, car il nous est souvent demandé d'apporter une conclusion scientifique et technique qui puisse résumer notre travail lors de ces 6 mois. Le stage faisant partie de notre cursus universitaire, il est normal qu'il représente un dernier défi académique. Cependant, un stage en entreprise, c'est également un apprentissage du monde du travail ainsi qu'une rupture avec l'université. C'est pourquoi, il me paraît important de reprendre également tout l'aspect humain et l'expérience qu'il représente.
Durant ce stage, j'ai eu le plaisir de collaborer avec Hélène Lucien, Youcef Kemiche et Théo Vanneufville, des profils d'horizons trés différents. Ensemble nous avons su faire en sorte que Makezu devienne un produit qui séduit des entreprises de toutes tailles.
À mon arrivée dans l'entreprise, il a fallu prendre connaissance de l'outil, pour avoir une vision d'ensemble, ainsi qu'une idée très précise de la volonté des fondatrices. Le travail sur l'interface utilisateur via l'outil Bubble a permis de structurer les attentes auxquelles l'application flask devait répondre. Dans la lisibilité de l'outil tout comme dans sa vision stratégique c'est cette dualité entre partie visible et cachée, qui rend complète et efficace notre application.
Par la mise en place d'un outil d'annotation de données, nous avons pu travailler directement sur des tweets et donc reprendre les termes et formules qui sont propres à ce réseau social. Avec l'apport de modèles mathématiques paramétriques ou non dans des problématiques d'apprentissage supervisé, nous avons mis en place des modèles de classification robuste et efficace avec comme témoin les mesures de qualité.
De nouvelles solutions ont été apportées à nos utilisateurs pour que Makezu réponde à leurs retours d'expériences : la possibilité de répondre par un tweet ou en message direct, la possibilité de cibler si le tweet s'adresse à un homme ou une femme,...
Il était pour moi très important de rejoindre une start-up lors de mon stage, et je ne regrette absolument pas ce choix. J'ai eu dans mon stage l'ambition de posséder plusieurs fonctions au sein du développement de cette application et je pense que j'aurais ressenti de la frustration s'il en avait été autrement.
Nous avons pu assister à un séminaire en ligne d'A.W.S., et suivre une dizaine de conférences sur un large champ de compétences et connaissances : du déploiement d'application web classique à la mise à l'échelle de projet d'apprentissage automatique ou de grosse donnée (big data), en passant par les architectures de stockages des données. Ce séminaire a permis d'explorer l'ensemble des métiers relatifs à la donnée, et nous a apporté des solutions pour que notre outil soit performant.
Cette confrontation au monde du travail m'a permis de mieux comprendre les problématiques liées à mon métier ainsi que les solutions qui existent pour y répondre. Par exemple, La lutte des hébergeurs distants ("cloud") pour le monopole des applications d'apprentissage automatique, ou encore la nécessité d'une puissance de calcul pour gérer des modèles de réseaux de neurones profonds : la mise en place d'outils utilisant les caractéristiques des cartes graphiques (G.P.U.) plutôt que celles des processeurs de calculs classiques (C.P.U.), l'invention de nouveau type de processeurs spécifique à l'utilisation des tenseurs (T.P.U.), et donc la création des objets que sont les tenseurs.

Cet ensemble de termes et concepts fait partie des axes de recherche pour proposer à Makezu un outil performant pour un coût optimal. Devoir construire et dimensionner toute une infrastructure et tout un projet dans ce stage a été une expérience formidable et exaltante. Tout comme l'a été de partager nos résultats avec une équipe intéressée et investie. Les expériences ratées ainsi que les impasses empruntées, comme la volonté d'avoir un outil qui récupère automatiquement les mots-cibles des tweets, ont été particulièrement formateurs.
Du point de vue de ma formation, j'ai pu me servir de mes enseignements en apprentissage supervisé et en data mining pour faire de l'analyse de données *texte*, mais j'ai également pu découvrir de nouvelles méthodes de classification très intéressantes, ainsi que des mesures de qualités de modèles qui m'ont apporté une approche plus tranchante pour la sélection de mes modèles. C'est dans ces découvertes que l'on prend conscience du spectre d'applications que peuvent revêtir les statistiques. Dans des domaines classiques comme l'informatique ou l'économie, comme dans la médecine ou la biologie, c'est cette vastitude de champs qui rend les mathématiques appliquées si intéressantes.
Le sujet de ce stage était novateur et d'actualité, les résultats les plus intéressants datent d'il y a quelques années, avec de nouveaux articles et de nouvelles pistes explorées chaque jour. C'est également un sujet qui mêle de nombreux champs de réflexion comme la linguistique, la traduction, les automates de paroles, mais aussi soulève des questions éthiques quant au caractère personnel des données des réseaux sociaux.
Bien que nombres de ces sujets méritent que je m'y attarde plus intensément, j'ai eu dans mon stage cette problématique de production d'un outil. Nous sommes à la genèse du projet Makezu et les pistes d'améliorations ne manquent pas, un des objectifs non atteint de mon stage aura été de reproduire ce que nous faisons, sur Twitter, sur d'autres plateformes comme LinkedIn ou Facebook.

ANNEXE A

LES LIBRAIRIES PYTHON UTILISÉES

Cette annexe parcours l'ensemble des modules et librairies utilisés de notre application, elle est constituée d'une brève déscription des actions de chacun d'entre eux, ainsi que de nombreux liens vers la documentation ou l'hébergeur de ceux-ci. On privilégiera donc une lecture sur un ordinateur.

FLASK : *Flask* est un utilitaire Python permettant la création d'application web. Il automatise la gestion du script Python pour éviter de définir à chaque fois les éléments de la page, en suivant la norme W.S.G.I.. Cette application permet la génération de sites web dynamique bien plus facilement, et par la même occasion étant un utilitaire Python, il peut également être associé à d'autres modules de Python. Dans notre cas, on associe des modules de N.L.P. , qui vont permettre à notre application la conversion de nos tweets et mots-clés dans les recherches. De nombreux tutos sont disponibles pour se former sur *Flask*, en anglais (https://www.tutorialspoint.com/flask/flask_application.html). Mais également en français (http://sdz.tdct.org/sdz/creez-vos-applications-web-avec-flask.html). Avec les exemples classiques : "Hello World", mais qui rapidement amène une utilisation du framework, pour la création d'A.P.I. plus élaborée. *Flask*, nous donne une facilité d'écriture pour répondre à la norme W.S.G.I..

Flask-C.O.R.S : *Flask-C.O.R.S.* est une extension C.O.R.S. de *Flask*, une documentation est disponible sur https://flask-cors.readthedocs.io/en/latest/, il permet la création de contenu web utilisant le mécanisme C.O.R.S., lorsqu'on développe une application web avec *Flask*.

FIRE : *Fire* est une librairie qui permet la génération automatique en ligne de code pour tout objet de Python. On en a quelques exemples dans le Github https://github.com/google/python-fire. On y voit par exemple un exemple sur un objet fonction ou encore sur un objet "class".

APPDIR : Le module Python, *appdir*, permet de faciliter le choix des accès et la génération automatique des répertoires en fonction du profil des intervenants dans une A.P.I.. L'exemple est écrit comme suit :

- user data dir
- user config dir
- user cache dir
- site data dir

— site config dir

— user log dir

On retrouve une démonstration du fonctionnement, sous différents O.S., dans le détail du projet disponible au lien https://pypi.org/project/appdirs/

ASTROID : Le module Python, *astroid*, est un module de représentation de code source Python utilisant les arbres syntaxiques ou arbres syntaxiques abstraits A.S.T., qui est une technique de découpage syntaxique en arbre (nœuds et feuilles) où les nœuds sont les opérateurs et les feuilles les opérandes. Il permet une génération par lignes de codes simplifiés.

ATTRS : A nouveau un lien vers la documentation du package est disponible au lien suivant : https://pypi.org/project/

Ce module est une aide au code Python, il permet la construction d'attributs pour une classe. Mais il permet également de récupérer la classe. Il faut bien faire attention lorsqu'on suit les étapes des différents tutos.

```
import attr
@attr.s
class Empty(object):
pass
```

Il faut avoir bien installé le package *attrs*, lors du chargement de la librairie avec

```
import attr
```

Sinon on obtient une erreur d'attribut (AttributeError : module 'attr' has no attribute 's') avec la commande *@attr*.s. En effet, il y a une confusion avec une autre librairie du nom de *attr*

BACKPORTS.CSV : Ici comme le nom l'indique, il s'agit d'un utilitaire de lecture|écriture au format *.csv*. Un lien vers la documentation Python se trouve au lien suivant : https://pypi.org/project/backports.csv/

BEAUTIFULSOUP :

BeautifulSoup4 : Cette section est sur la libraire, très populaire *Beautiful Soup*, avec une grande documentation, ainsi que de nombreux blogs, pour en avoir des tutoriels vraiment bien guidés. Mais à quoi sert donc cette librairie ? *Beautiful Soup* permet la récupération de données depuis des fichiers H.T.M.L. et X.M.L.. C'est-à-dire depuis toutes les pages web. La documentation sur la librairie est vraiment dense, et permet à des débutants de bien établir les bases pour la compréhension de comment les données peuvent être récupèrer depuis le web. Par des illustrations, et des astuces de compréhensions. Ici, une liste non exhaustive de blog qui fournissent des tutos avec l'utilisation de cette librairie :

— Un tuto avec l'ensemble des basiques en français : https://code.tutsplus.com/fr/tutorials/scraping-webpages-in-python-with-beautiful-soup-the-basics–cms-28211

— Récupération de vidéo sur D8 replay : http://sametmax.com/parser-du-html-avec-beautifulsoup/

— Un parser d'Instagram : https://edmundmartin.com/scraping-instagram-with-python/

L'ensemble de ces blogs fournit un catalogue intéressant de l'utilisation que l'on peut faire avec cet outil.

bs4 : Ce package est un backup de *Beautiful Soup*, mis en place pour, qu'en cas d'erreur dans la typographie, les utilisateurs de Python utilise le bon package, pour prevenir de developpeurs malveillants, qui utilisent la confusion.

SOUPSIEVE : Utilitaire de *Beautiful Soup*, pour lire les arguments en C.S.S., en plus des éléments textuellesc H.T.T.P.. d'une page web. Pour comprendre en profondeur ce module, on peut retrouver l'index

PyPI, un git, mais également un blog d'entrainement à l'adresse : https://facelessuser.github.io/soupsieve/api/ sur l'A.P.I..

BLACK : *black* est une aide à la réalisation de code. Il permet de rendre plus lisible l'écriture de tuples lorsque ceux-ci commencent à être de grande taille. *black* se voit comme une sous-section de la nomenclatutre PEP8 de Python.

BLIS : Ensemble de routines d'algèbre linéaire, indispensables pour le machine learning. Remarque : Il faut un traitement particulier pour l'installation de cette librairie dans un environnement Windows. En effet, la librairie demande l'installation de L.L.V.M., qui est un compilateur de code. *blis* est une extension C pour Python, qui demandera la création d'un fichier wheel, pour l'instalation dans un environnement Windows, d'où le besoin d'utiliser L.L.V.M.. *blis* utilise numpy, utilitaire de base Python pour le calcul, en arrière-plan. Mais revenons donc à notre outil *blis*, il se sert de la structuration de calcul du modèle B.L.A.S. pour Basic Linear Algebra Subprograms, qui sont 3 sous-programmes, pour 3 niveaux de calculs :

- niveau 1 : opérations entre vecteur et scalaire, ou vecteur et vecteur
- niveau 2 : opérations entre vecteur et matrice
- niveau 3 : opérations entre matrice et matrice

B.L.I.S. est alors l'acronyme de BLAS-like Library Instantiation Software, il s'agit donc d'un programme, qui reprend les règles de calculs en 3 niveaux de *B.L.A.S.*. *blis* apporte une optimisation calculatoire en comparaison de B.L.A.S., disponible dans le package *numpy*, un exemple de comparaison de vitesse de calcul est donné en exemple dans le depository de PyPI au lien suivant : https://pypi.org/project/blis/.

BOTO3 : *boto3* est l'utilitaire Python pour la gestion des services cloud d'Amazon Web Service, dans notre cas d'utilisation, il nous permet la gestion de l'instance (le serveur web), mais également les bases de données DynamoDB (NoSQL) et RedShift (S.Q.L.).

CATALOGUE : *catalogue* permets l'appel de fonctions ou données, un exemple donné dans les archives Python , au lien https://pypi.org/project/catalogue/, présente le cas d'un package Python que l'on a déjà créé, auquel on veut laisser un utilisateur du package ajouter ses propres données.

CERTIFI : Outil de vérification de certificat S.S.L. pour l'identité des hôtes T.L.S.. Gestion de clé de cryptage.

CHARDET : Utilitaire de détections de codage texte :

- ASCII, UTF-8, UTF-16 (2 variants), UTF-32 (4 variants)
- Big5, GB2312, EUC-TW, HZ-GB-2312, ISO-2022-CN (Traditional and Simplified Chinese)
- EUC-JP, SHIFT-JIS, CP932, ISO-2022-JP (Japanese)
- EUC-KR, ISO-2022-KR (Korean)
- KOI8-R, MacCyrillic, IBM855, IBM866, ISO-8859-5, windows-1251 (Cyrillic)
- ISO-8859-5, windows-1251 (Bulgarian)
- ISO-8859-1, windows-1252 (Western European languages)
- ISO-8859-7, windows-1253 (Greek)
- ISO-8859-8, windows-1255 (Visual and Logical Hebrew)
- TIS-620 (Thai)

CHEROOT : Un serveur H.T.T.P.. haute performance, supporté par *cheroot*.

CHERRYPY : On a donc vu que *CherryPy* est un framework, comme *Flask*.

On l'utilise dans la création d'application web, c'est une grosse machine, largement utilisés, rapides et éfficace.

CLICK : *Click* possède une superbe documentation au lien suivant : https://click.palletsprojects.com/en/7.x/. *Click* permet entre autres la génération automatique des aides, c'est une aide à la génération de code en ligne de commande, on appelle ça un kit C.L.I.. Il améliore également les procédures pour la génération de sous-commandes.

CYMEM : On rentre avec cette librairie dans un environnement plus vaste encore que le simple langage Python, en effet dans cette librairie, qui sert de modérateur de mémoire cache par la mise en place de "cycle de vie" d'objet Python. On utilise une extension Python qui permet la lecture de partie de programme en C, Cython.

Mais Cython permet également le dialogue dans les deux sens, en effet, il permet la lecture de C, ou de C++ en Python, mais également l'inverse. Ce qui permet d'utiliser des modules en C dans des applications Python. Mais revenons alors à *Cymem*, l'accès à des séquences de données, peut rapidement être ralenti avec les objets *list* ,de Python, un moyen plus rapide de traiter ces informations seraient de passer à C, avec les objets *C-array*, ou encore en C++, avec des objets, *C++vectors*.

Avec l'objet *Pool*, du package, on peut effectuer ce traitement. Qui permet également des liens entre la partie stockée sur des éléments C, ou C++, et l'appel que l'on en fait avec Python, par exemple la suppression de l'élément Python, supprimera le stockage en C.

DECORATOR : De manière générale, les "decorator pattern", sont des utilitaires qui permettent l'ajout de sous-sections à des modules sans altérer les modules eux-mêmes. Il permet la modification syntaxique du code sans génération d'erreurs.

DOCX : Le nom de cet outil nous renseigne déjà bien sur son utilisation, en effet *Pool* permet la génération, la lecture, ainsi que l'écriture de fichier Microsoft Word *.docx*.

FEEDPARSER : Il s'agit là encore d'un web parser, qui nous permet la navigation sur des URL pour récupérer du contenu organisé avec attributs et corps, suivant la structure R.S.S..

FUTURE : Dans ce module, il est question de compatibilité des versions Python 2 et 3. L'utilisation de ce module permet de générer du code compatible Python3 à partir de code en Python2 et inversement.

GOOGLETRANS : Module Python de Google Translate, pour intégrer le traducteur de Google dans un code Python. On y apprend que l'A.P.I. suit le protocole web A.J.A.X...

IDNA : Encodeur et décodeur de nom de domaine pour répondre au format I.D.N.s.

IMPORTLIB-METADATA : On a affaire ici à une librairie qui permet d'obtenir les métadonnées des librairies Python.

ISORT : Module Python de réorganisation de code.

ITSDANGEROUS : *itsdangerous* permet le cryptage par méthode de clé générée qui porte le nom de token, qui assure la sécurité des données transmises.

JARACO : Dans ce chapitre, on s'intéresse aux travaux de Jason R. Coombs [1]. Qui cherche l'optimisation de son temps de travail par la création d'une structuration qui permet l'évolution des objets Python pour suivre les différentes mises à jour, cette structuration est définie par le terme *skeleton*, introduit dans son blog, on va donc donner les liens des différents outils du *skeleton*, utilisé dans l'A.P.I. de Centrale. JARACO.CLASSES : https://pypi.org/project/jaraco.classes/

JARACO.COLLECTIONS : https://pypi.org/project/jaraco.collections/

JARACO.FUNCTOOLS : https://pypi.org/project/jaraco.functools/

1. Lien vers sa page personnelle : https://www.jaraco.com/

JARACO.TEXT : https://pypi.org/project/jaraco.text/

JINJA2 : *Jinja2* est un developpeur de template pour Python, sur les bases de ce que met en place Django.

LAZY-OBJECT-PROXY : C'est une librarie basée sur les objets Les librairies Python utilisées. Il permet le stockage des résultats de requêtes à des fonctions qui sont soumises à un travail sur une durée.

LOGURU : *Loguru* est une librairie pour simplifier le logging. On peut par exemple lui préciser si on veut mettre tous les logs dans un fichier, ce que l'on ne peut pas avec le logging de base de Python.

LXML :Utilitaire rapide et efficace d'intégration de X.M.L. et de H.T.M.L. en Python.

MARKUPSAFE : Outil très utile, *markupsafe*, permet la correction de l'interprétation de caractères spéciaux suivants les langages utilisés, en effet il arrive que la nomenclature Python arrive à un conflit d'interprétation avec une commande H.T.M.L., C ou bien X.M.L., c'est une erreur de programmation classique, que le module permet de corriger.

MCCABE : Pour ce plug-in, nous utilisons un élément de mesure de complexité de programme informatique, la complexité McCabe, ou encore complexité cyclomatique.
Le module peut également être utilisé en plug-in pour Flake8.

MORE-ITERTOOLS : Uu utilitaire de gestion des ittérateurs très complets, pour l'optimisation des boucles.

MURMURHASH : Utilitaire présent sur C, *Murmurhash* permet l'intégration en Python de la fonction de hachage éponyme. En bref, une fonction de hachage, permet la réduction de la mémoire nécessaire, par la transformation de l'objet de base (généralement de grande taille), en chaîne alphanumérique, que l'on nomme généralement empreinte, ou hash.

NETWORKX : Package d'analyse de réseaux complexes avec Python, récupération des chemins les plus courts entre deux nœuds distincts, analyse de graphe.

N.L.T.K. : Module de N.L.P. commun et robuste avec une grosse base d'utilisateurs.

NUMPY : Librairie Python de base pour le calcul. Sous licence B.S.D.. C'est un open-source largement répandu.

OAUTHLIB : Module de gestion des autorisations sous la forme OAuth. Permets donc la création de signature OAuth pour l'authentification des utilisateurs de l'A.P.I..

PACKAGING : Module de gestion des packages Python, il permet l'accompagnement pour le développement d'une application qui corresponde bien aux structures de module Python.

PATHSPEC : Comme son nom l'indique ce module permet la correspondance des *path* (chemins), avec le protocole Wildmatch.

PATTERN : ***Pattern*** est un outil multi-usage divisé en trois catégories :

- Web-Mining d'internet. Récupération de données et crawling. Avec pattern.web
- Analyse automatique du language. Avec l'outil pattern.en en anglais, mais disponible en 6 langues. Qui applique des annotations aux différents mots trouvés par types (ex : noms, verbes, adjéctifs), qui permet un traitement par découpage.
- Machine learning. Avec de l'analyse vectorielle (S.V.M., K.N.N.) et une représentation graphique avec pattern.graph

PDFMINER.SIX : Librairie Python de récupération de données sur des fichiers *pdf*.

PDFMINER3K : Comme PDFMINER.SIX, PDFMINER3K est un mineur de document *pdf*, qui se consacre uniquement sur le texte et sa position.

PICKLE : Dans ce chapitre on s'iteresse à un petit outil de stockge d'élément Python. *Pickle* permet de transformer en série les objets Python, ce qui peut s'avérer utile pour le stockage de fichier temporaire sans à chaque fois à avoir géré une grosse mémoire.

PILLOW : Gère en Python les fichiers photos P.I.L..

PLAC : *Plac* est un outil d'aide à la compilation de code, il permet de programmer orienté objet.

PLUGGY : C'est un manager de gestion de plugins Python.

PLY : *Ply* est une implémentation de Lex et Yacc en Python qui permet la génération automatique de code, et qui permet une analyse syntaxique.

PORTEND : Un nouveau module du programmeur Jason R. Coombs, qui permet la gestion des conexions T.C.P., il permet de monitorer suivant les différents statuts des ports.

PRESHED : Module en *Cython*, contraction de C et de Python, preshed permet le hash (A), suivant la méthode de Jeff Preshing.

PY : Librairie de support avec les différents outils :

— 'path', objet path

— 'apipkg' module d'aide à la gestion d'API

— 'iniconfig' segmentation ddu fichier ini.

— 'code' génération et correction dynamique de code.

PYCRYPTODOME : Librarie de cryptage, utilitaire plus complet pour l'amélioration de PyCrypto, avec une liste de méthodes de cryptage disponibles..

PYLINT : Analyseur de code statique.

PYPARSING : Permets une alternative au parsing Lex et Yacc, par des modélisations simples de grammaire.

PYSOCKS : Utilitaire de transfert à travers des protocoles H.T.T.P.. ou S.O.C.K.S.. C'est la version moderne de SoxksiPy

PYTEST : Librairie pour effectuer de petit test de manière à améliorer le texte codé.

PYTHON-DOTENV : Module d'appel pour le chargement d'un environemment depuis un dossier *.env* dans le répertoire actuel. Il lit l'expension POSIX.

PYTZ : Utilitaire de découpage en créneau horaire de la base de données de T.Z., il nous permet de récupérer le format ainsi que l'heure actuelle en fonction des régions.

REGEX : Librairie sur la programation régulière.

REQUESTS : Librairie H.T.T.P.., pour récupérer des URLs, avec les codes de requêtes, l'encoding, le texte ainsi qu'un fichier J.S.O.N. avec l'ensemble de la requête comme classe.

REQUEST-OAUTHLIB : Permets de créer une interface plus accessible des librairies *request*A et *oauthlib*A.

SIMPLEJSON : Encodeur et décodeur J.S.O.N. pour Python.

SIX : Un outil de relecture de code pour une compatibilité Python 2 et 3.

SKLEARN : La librairie de machine learning de Python, la plus complète, bâtie sur *numpy*, *scipy* et *matplotlib*. Avec des applications en apprentissage supervisé et non supervisé avec des méthodes de data mining.

SORTEDCONTAINERS : Editeur de classement rapide en full-python, sous licence linux Apache2, qui permet de ne pas se servir des utilitaires communiquant en C pour des procédés de tries performant sur de grandes bases.

SPACY : Librairie N.L.P. très performante, basé sur des modèles pré-entrainés, on parle de transfer-learning. Il supporte près de 50 langues, prend en compte la segmentation grammaticale. Permets l'extraction en vecteurs *numpy*.

SRSLY : Outil de mise en séries des packages Python pour condenser les applications de l'ensemble des packages et des versions de Python. Pour une optimisation du temps de procédure, lorsqu'un paquet C appelle un paquet Python, pour transmettre un fichier Java, les différents passages et les lectures|écritures doivent être optimisées, c'est ce que permet le package *srsly*.

TEMPORA : Module d'objets dates et temps mis en place par Jason R. Coombs(A).

THINC : Librairie apprentissage automatique, compatible avec *spaCy* (A). C'est également un utilitaire de N.L.P.. Il suit la trame : intégrer, encoder, assister, prévoir. Structurant les méthodes de langage. https://thinc.ai/.

TOML : Il s'agit à nouveau d'un parser vers une forme T.O.M.L..

TQDM : Génère la visualisation d'une barre de chargement lors du parcours d'une boucle dans un code ou d'un chargement, format largement supporté, toutes les plateformes Python l'utilisent.

TWEEPY : Un plug-in pour accéder facilement à l'A.P.I. Twitter.

TYPED-AST : Parser de Python3 supportant comme astroid (A) les A.S.T..

UNIDECODE : Traducteur unicode, à tout type de format vers un type de format A.S.C.I.I..

URLLIB3 : Client H.T.T.P.. pour Python.

WASABI : Librairie de colorisations des messages d'alertes a intégrer à une A.P.I.. Fruit du travail de Ines (https://ines.io/), une informaticienne de chez *spaCy* (A).

WAITRESS : Serveur Python W.S.G.I.. C'est un module stable sans aucune dépendance, qui tourne avec Cython, ou Python.

WCWIDTH : Permet la production de fichiers de sortie d'une A.P.I., ou d'un module qui doit être lu par un interprète donné.

WERKZEUG : Librairie web d'application W.S.G.I..

WRAPT : Wrapt Librairie objet proxy, complète est composée de nombreux plugins, avec notamment *decorator* (A).

ZC.LOCKFILE : Interpréteur de verrouillage de fichiers, pour la protection d'objets critiques dans l'A.P.I.. Il repose sur le navigateur de fichier *os*.

ZIPP : Un enrôleur de fichier archivé mis en place par Jason R. Coombs(A).

ANNEXE B

SPACY

On l'a parcouru souvent dans ce rapport mais la librairie spaCy a été primordiale dans la mise en place de nos modèles d'apprentissage automatique. Dans cette annexe, on développe un peu ce module pour faire le tour de ses usages et de sa construction.

Tout d'abord, pour avoir un aperçu de ce module, je ne peux que vous conseiller de naviguer sur le site du projet https://spacy.io/. On peut y trouver une documentation très complète ainsi que quelques vidéos pour mieux comprendre la construction de l'outil.

Pour utiliser spaCy, il faut donc commencer par installer le module, comme pour toute librairie Python. De plus on charge un modèle pré-entrainé qui nous permettra la vectorisation des textes une fois chargé. On charge un modèle avec les lignes de commande suivante *spacy load nom_du_modèle*. SpaCy fait partie de la famille des produits de explosionAI comme l'outil d'entrainement des données prodigy que l'on a abordé dans le chapitreIII, ou encore l'outil thinc qui permet la verfication des codes d'apprentissage profond (notamment l'usage des tenseurs). C'est un leader dans le domaine, et spaCy est l'outil de traitement automatique du langage. La puissance de spaCy est qu'il base ses modèles sur de l'entrainement bidirectionnel reprenant les architectures de B.E.R.T. pour les modèles de vectorisation. On a une possibilité d'utiliser spaCy dans 15 langues et il existe un traitement multi-langues [1]. Les modèles sont entrainés sur la base de données de textes de Wikipédia, qui fournit un accès ouvert à l'ensemble des textes que composent l'ensemble des articles. C'est donc une base de données qui reprend une grande partie du vocabulaire en concentrant bien les termes régulièrement utilisés.

SpaCy prend en charge la vectorisation des textes sans détruire la chaîne de caractères proposée en entrée, c'est-à-dire que lorsqu'on applique le modèle à une variable *texte* Python, on crée une classe qui possède tous les renseignements qu'apporte le modèle déployé avec spaCy en conservant le texte original. La classe possède de multiples attributs qui permettent de réaliser de nombreuses utilisations de traitement du langage comme l'étiquetage morphosyntaxique : la reconnaissance des noms, verbes, adjectifs, ... mais également des singuliers, pluriels, et genres des mots. On a la création des vecteurs (token), qui sont une projection sur 96 variables, des mots mais également des phrases entières. On peut donc suivant notre utilisation se servir de jetons de phrases

1. Ces langues sont le chinois, le danois, le néerlandais, l'anglais, le français, l'allemand, le grecque, l'italien, le japonais, le lituanien, le norvégien, le polonais, le portugais, le roumain, et l'espagnol.

ou de mots. On utilise également la représentation de dépendances. Je me permets de reprendre le schéma explicatif pour résumer l'ensemble des utilisations du module :

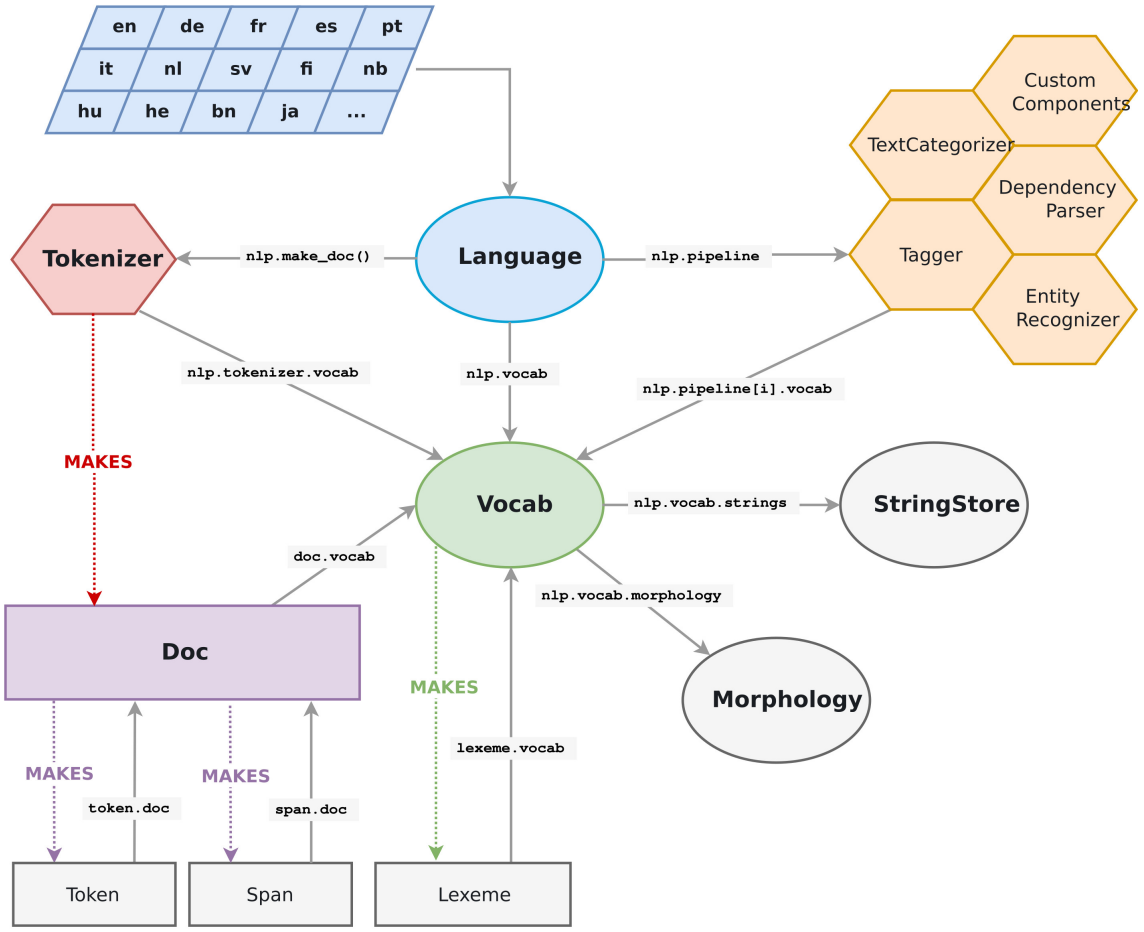

FIGURE B.1 – Utilisation spaCy - source site spaCy

Une dernière particularité de cette librairie qui fera sans doute l'objet d'une utilisation future de notre outil est la détection d'entités. Par exemple, les marques telles que Windows Microsoft ou Apple peuvent, suivant le contexte être identifiées, comme des entreprises et non pas une pomme (apple) ou une fenêtre (window).

ANNEXE C

BASE DE DONNÉES

Cette courte annexe reprend la structure de la base de données DynamoDB hébergée sur Amazon Web Service.

La base de données est composée de 4 tables, la première table reprend les éléments enregistrés des campagnes :

id_campagne	l'identifiant campagne
id_client	l'identifiant utilisateur
stratégies	un dictionnaire de booléen pour chaque strategie disponible
audience	un dictionnaire avec les éléments d'une campagne de type audience
life_moment	un dictionnaire avec les éléments d'une campagne de type moment de vie
product_targetting	un dictionnaire avec les éléments d'une campagne de type recherche de produit
self_description	un dictionnaire avec les éléments d'une campagne de type auto-description

TABLE C.1 – Table campagne

Pour une campagne sélectionnée, les champs des autres types de campagnes sont vides.

La seconde table est une table audience, qui nous donne les résultats des campagnes de types audience uniquement.

id_user	l'identifiant compte Twitter
audience	la cible de l'intérêt de l'utilisateur
lang	la langue de recherche
id_tweet	l'identifiant du tweet sur lequel on a identifié la cible
tweet_text	le texte du tweet sur lequel on a identifié la cible
user	le nom de l'utilisateur Twitter

TABLE C.2 – Table audience

On a également une table tweets qui répertorie les résultats des campagnes de types recherche de produit, auto-description, moments de vies.

id_tweet	l'identifiant du tweet sur lequel on a identifié la cible
classified	Booléen True pour un tweet qui repondait à la requête False sinon
id_campagne	l'identifiant campagne
id_client	l'identifiant utilisateur
lang	la langue de recherche
tweet_text	le texte du tweet sur lequel on a identifié la cible

TABLE C.3 – Table tweets

La dernière table est la table des requêtes des moments de vie.

name	le nom du moment de vie
eng	un dictionnaire reprenant les requêtes en anglais ainsi qu'un booléen pour signifier d'un modèle de N.L.P.
fra	un dictionnaire reprenant les requêtes en français ainsi qu'un booléen pour signifier d'un modèle de N.L.P.

TABLE C.4 – Table lifemomentqueries

TABLE DES FIGURES

1.1 Distribution de la loi de Zipf - source Wikipédia . . . 10
1.2 Répartition de la loi de Zipf - source Wikipédia . . . 10
1.3 Représentation d'une couche d'un LSTM - source Wikipédia . . . 13
1.4 Porte d'oubli - LSTM - source github colah . . . 14
1.5 Porte d'entrée - LSTM - source github colah . . . 14
1.6 Porte de sortie - LSTM - source github colah . . . 14
1.7 Sigmoïde|Fonction Logistique - source Wikipédia . . . 14
1.8 Graphe de la fonction Tangente Hyperbolique - source Wikipédia . . . 14
1.9 Représentation d'une couche d'un G.R.U. - source Wikipédia . . . 15
1.10 Représentation d'un Transformer - source "Attention Is All You Need" . . . 16

2.1 Tableau de bord utilisateurs application Makezu . . . 20
2.2 Chemin d'une campagne . . . 21
2.3 Dessin des dépendances dans le début du tweet . . . 25
2.4 Dessin des dépendances avec le groupe verbal . . . 25
2.5 Graphe des dépendances . . . 25
2.6 Tweet d'auto-description . . . 28
2.7 Graphe auto-description . . . 28

3.1 Etiqueteur besoin exprimé . . . 29
3.2 Etiqueteur moment de vie . . . 30

4.1 Tweets & vecteurs spaCy associés . . . 34
4.2 Graphes bon|mauvais 20 premiers tweets vecteurs spaCy . . . 34
4.3 Projection sur les composantes 1 & 2 - spaCy . . . 34
4.4 Projection sur les composantes 3 & 4 - spaCy . . . 34
4.5 Projection sur les composantes 1 & 2 - T.F.-I.D.F. . . . 35
4.6 Projection sur les composantes 3 & 4 - T.F.-I.D.F. . . . 35
4.7 Résultats des tests Shapiro-Wilk vecteur spaCy . . . 36

4.8 Graphe quantiles-quantiles 1er vecteur spaCy 36
4.9 Résultats des tests Shapiro-Wilk vecteur T.F.-I.D.F. 36
4.10 Graphe quantiles-quantiles vecteur T.F.-I.D.F. 36
4.11 Courbes R.O.C. - source Wikipédia 38

5.1 Dépendances des mots clés avec le verbe buy 46
5.2 Dépendances des mots clés avec le verbe buy - 2eme exemple 46
5.3 Dépendances des mots clés avec le verbe buy - 3eme exemple 46
5.4 Pas de dépendance avec le verbe buy 47
5.5 Taille en nombre de mots des tweets 49
5.6 Fonction de coûts sur nos jeux de données 49
5.7 Exemple de l'analyse de similarités - source TensorFlowHub 50
5.8 Matrice des similarités 50
5.9 Fine-tuning de l'encodeur de phrases - source TensorFlowHub 51

B.1 Utilisation spaCy - source site spaCy 62

LISTE DES TABLEAUX

2.1 Explication des relations 21
2.2 Limites de l'API de Twitter 22
2.3 Résultats du N.L.P. avec spaCy 24
2.4 Explication des résultats spaCy 24

4.1 Moments de vie et nombre de requêtes associées 32
4.2 Résultats T.F.-I.D.F. sur deux tweets 33
4.3 Tweets et leurs labels 34
4.4 Variances expliquées sur les composantes - spaCy 34
4.5 Variances expliquées sur composantes - T.F.-I.D.F. 35
4.6 Résultats des tests du χ^2 35
4.7 Matrice de confusion 37
4.8 Tableau des résultats bayésien naïf gaussien 39
4.9 Tableau des résultats analyse discriminante linéaire 39
4.10 Tableau des résultats machines à vecteurs support 40
4.11 Tableau des résultats régression logistique 40
4.12 Tableau des résultats de descente de gradient stochastique 41
4.13 Tableau des résultats k-plus-proches voisins 42
4.14 Tableau des résultats arbres de décisions 42
4.15 Tableau des résultats forêts aléatoires 42
4.16 Tableau des résultats extra-forêts aléatoires 43
4.17 Tableau des résultats AdaBoost forêts aléatoires 43
4.18 Tableau de synthèses des résultats 44

5.1 Structure de la table pour le fichage automatique 45
5.2 Occurence des mots clés cibles 47
5.3 Occurence des mots clés regroupés 48
5.4 Résultats de l'encodeur de phrases sur notre jeu de données test. 51

C.1 Table campagne 63
C.2 Table audience 63
C.3 Table tweets 64
C.4 Table lifemomentqueries 64

ACRONYMES

A.C.P. Analyse en Composantes Principales. 35

A.I. Artificial Intelligence. 9, 13, 30

A.J.A.X. Asynchronous JavaScript and XML. 19, 57

A.L.P.A.C. Automatic Language Processing Advisory Committee. 7

A.P.I. Application Programming Interface. 22, 23, 25–27, 30, 32, 45, 54, 56–58, 60

A.S.C.I.I. American Standard Code for Information Interchange. 60

A.S.T. Abstract Syntax Tree. 55, 60

A.W.S. Amazon Web Service. 19–22, 44, 52

B.E.R.T. Bidirectional Encoder Representation for Transformer. 8, 9, 17, 18, 48, 61

B.L.A.S. Basic Linear Algebra Subprograms. 56

B.L.I.S. BLAS-like Library Instantiation Software. 56

B.S.D. Berkeley Software Distribution. 58

C.L.I. Command Line Interface. 57

C.N.I.L. Commission Nationale de l'Informatique et des Libertés. 30, 31

C.N.N. Convolutional Neural Network. 17

C.O.R.S. Cross-Origin Resource Sharing. 19, 54

C.P.U. Computer Processing Unit. 52

C.S.S. Cascading Style Sheets. 55

G.P.U. Graphical Processing Unit. 16, 52

G.R.U. Gated Recurent Unit. 13, 15, 16

H.T.M.L. HyperText Markup Language. 55, 58

H.T.T.P. Hypertext Transfer Protocol. 19, 22, 55, 56, 59, 60

I.D.N.s International Domain Names. 57

J.S.O.N. JavaScipt Object. 22, 59

K.N.N. K-Nearest Neighbours. 58

L.L.V.M. Low Level Virtual Machine. 56

L.O.O. Leave One Out. 37

L.S.T.M. Long Short Time Memory. 13, 15, 16

N.L.T.K. Natural Language ToolKit. 15, 26, 27, 47, 58

NoSQL Not only Structured Query Language. 19, 56

O.S. Operating System. 55

O.S.C.A.R. Open Super-large Crawled ALMAnaCH coRpus. 9

P.C. Post Computer. 30

P.I.L. Public Image Limited. 59

P.O.S. Part-Of-Speech. 21

PEP8 Python Enhancement Proposal. 56

R.N.N. Recurent Neural Network. 16

R.O.C. Receiver Operating Characteristic. 38

R.S.S. Really Simple Syndication. 57

ReLU Rectified Linear Unit. 18

S.A.S. Société Anonyme Simplifiée. 1, 5

S.O.C.K.S. Secured Over Credential-based KerberoS. 59

S.Q.L. Structured Query Language. 56

S.S.L. Secure Sockets Layer. 56

S.V.M. Support Vector Machine. 44, 58

T.C.P. Transmission Control Protocol. 59

T.F.-I.D.F. Term Frequency - Inverse Document Frequency. 8–12, 33, 35, 41, 43, 44

T.L.S. Transport Layer Security. 56

T.O.M.L. Tom's Obvious, Minimal Language. 60

T.P.U. Tensor Processor Unit. 52

T.Z. Time Zone. 59

W.S.G.I. Web Server Gateway Interface. 54, 60

W.Y.S.I.W.Y.G. What You See IS What You Get. 20

WordNet Word Network. 26, 27, 47

X.M.L. eXtensible Markup Language. 55, 58

GLOSSAIRE

action anticipative Formulation des demandes ou à proposition des options ou des solutions tournées vers l'avenir.. 18

analyse sémantique latente Procédé de traitement des langues naturelles, dans le cadre de la sémantique vectorielle.. 8

annotation grammatico-lexicale Méthode d'attribution des caractéristiques lexicales et grammaticales.. 8

apprentissage par transfert champs de recherche de l'apprentissage automatique qui vise à transférer des connaissances d'une ou plusieurs tâches sources vers une ou plusieurs tâches cibles.. 29

apprentissage profond Ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires. 8, 11, 16, 61

bagging Meta-algorithme d'apprentissage ensembliste conçu pour améliorer la stabilité et la précision des algorithmes d'apprentissage automatique.. 41

biais cognitif Distorsion dans le traitement cognitif d'une information.. 30

cloud Accès à des services informatiques (serveurs, stockage, mise en réseau, logiciels) via Internet.. 19, 22, 52, 56

cognitive Qui se rapporte à la faculté de connaître.. 26

ELIZA Programme informatique, d'intelligence écrit par Joseph Weizenbaum, qui simule un psychothérapeute en reformulant la plupart des affirmations du patient en questions, et en les lui posant.. 7, 8

entropie croisée Mesure le nombre de bits moyen nécessaires pour identifier un événement issu de l' ensemble des événements.. 12

epoch Date initiale à partir de laquelle est mesuré le temps par les systèmes d'exploitation.. 49, 51

fonction d'activation Fonction mathématique appliquée à un signal en sortie d'un neurone artificiel.. 15, 18

fonctions coût Fonction qui sert de critère pour déterminer la meilleure solution à un problème d'optimisation.. 41

framework Ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel.. 19, 54, 56

grammaire distributionnelle Théorie syntaxique suivant laquelle le comportement humain serait totalement explicable, et on pourrait en étudier la mécanique, dans la création des langues.. 8

grammaire générative et transformationnelle Théorie syntaxique s'inscrivant dans le courant de la linguistique générative. Majoritairement présente en Amérique du Nord, elle s'est développée depuis 1957 sous l'impulsion de Noam Chomsky. Cette théorie tente de caractériser la connaissance de la langue qui permet l'acte effectif du locuteur-auditeur.. 8

label Étiquette ou marque spéciale créée par un syndicat professionnel et apposée sur un produit destiné à la vente, pour en certifier l'origine, en garantir la qualité et la conformité avec les normes de fabrication.. 17, 29, 30, 34, 45

lexicographique Qui concerne la discipline dont l'objet est l'élaboration des dictionnaires.. 27

Machine à vecteurs de support Ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de discrimination et de régression.. 40

nocode Environnements de développement intégrés visuellement.. 19

parser Mis en évidence la structure d'un texte.. 55, 57, 60

perte de gradient Empêche la modification des poids d'un réseau en fonction d'évènements passés, on a une perte d'information de la méthode de descente de gradient stochastique.. 13

proxy Composant logiciel informatique qui joue le rôle d'intermédiaire en se plaçant entre deux hôtes pour faciliter ou surveiller leurs échanges.. 60

réglage fin Situation où un certain nombre de paramètres doivent avoir une valeur très précise pour pouvoir rendre compte de tel ou tel phénomène observé. . 29, 51

rétropropagation du gradient Méthode pour calculer le gradient de l'erreur pour chaque neurone d'un réseau de neurones, de la dernière couche vers la première.. 12

softmax Généralisation de la fonction logistique.. 12

start-up Jeune entreprise innovante, notamment dans le secteur des nouvelles technologies.. 6, 52

tenseur Grandeur mathématique ayant plusieurs composantes et qui possède certaines propriétés d'invariance formelle dans un changement de base des espaces vectoriels sous-jacents. (Introduit en 1900 par C. G. Ricci et T. Levi-Civita, le calcul tensoriel est très utilisé en analyse et en mécanique classique ; il est indispensable en mécanique relativiste.). 16, 19, 48, 52, 61

théorie de l'information Théorie probabiliste permettant de quantifier le contenu moyen en information d'un ensemble de messages.. 38

twitter Service de microblogging créé aux États-Unis en 2006, qui permet aux utilisateurs d'envoyer et de recevoir des messages brefs, appelés Tweets.. 6, 15, 20–23, 25–27, 30–32, 45, 53, 60, 63, 67

un-chaud Consiste à représenter des états en utilisant pour chacun une valeur dont la représentation binaire n'a qu'un seul chiffre 1.. 11

vectorisation Adaptation d'un programme informatique en vue de son traitement sur un ordinateur.. 8, 9, 11, 12, 18, 33–35, 41, 43, 44, 61

vraisemblance Probabilité proche de la certitude.. 40

état-de-l'art Etat des connaissances dans tout domaine donné (scientifique, technique, artistique, médical, etc.) à un instant donné.. 16

étiquetage morphosyntaxique Processus qui consiste à associer aux mots d'un texte les informations grammaticales correspondantes comme la partie du discours, le genre, le nombre, etc. à l'aide d'un outil informatique.. 23, 61

BIBLIOGRAPHIE

[1] Fatiha BOUBEKEUR and Wassila AZZOUG. Pondération des concepts en recherche d'information sémantique. *CORIA*, 2013.

[2] Daniel CER, Yinfei YANG, Sheng-yi KONG, Nan HUA, Nicole LIMTIACO, Rhomni ST. JOHN, Noah CONSTANT, Mario GUAJARDO-CESPEDES, Steve YUAN, Chris TAR, Yun-Hsuan SUNG, Brian STROPE, and Ray KURZWEIL. Universal sentence encoder. In *Conference on Empirical Methods in Natural Language Processing*, 2018.

[3] Noam CHOMSKY. *Réflexions sur le langage.* Paris : Flammarion, 1981.

[4] Junyoung CHUNG, Caglar GULCEHRE, KyungHyun CHO, and Yoshua BENGIO. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Deep Learning and Representation Learning Workshop*, 2014.

[5] Jacob DEVLIN, Ming-Wei CHANG, Kenton LEE, and Kristina TOUTANOVA. Bert : Pre-training of deep bidirectional transformers for language understanding. *Cornell University*, 2019.

[6] Judith DUPORTAIL. *Amour sous algorithme.* Paris : Goutte d'or, 2019.

[7] John Ruppert FIRTH. *The Tongues of Men.* London : Watts et Co, 1937.

[8] Maurice GROSS. *Méthodes en syntaxe : régime des constructions complétives.* Paris : Hermann, 1975.

[9] Zellig S. HARRIS. *A grammar of English on mathematical principles.* New York : Wiley, 1982.

[10] Zellig S. HARRIS. *Langue et information.* Paris : CRL, 2008.

[11] Kris HEYLEN and Ann BERTELS. Sémantique distributionnelle en linguistique de corpus. *CAIRN*, 2016.

[12] sepp HOCHREITER and Jürgen SCHMIDHUBER. Neural computation. *Massachusetts Institute of Technology Press*, 1997.

[13] Jeremy HOWARD and Sebastian RUDER. Universal language model fine-tuning for text classification. In *56th Annual Meeting of the Association for Computational Linguistics*, 2018.

[14] Louis MARTIN, Benjamin MULLER, Pedro Javier ORTIZ SUÁREZ, Yoann DUPONT, Laurent ROMARY, Éric DE LA CLERGERIE, Djamé SEDDAH, and Benoît SAGOT. CamemBERT : a tasty French language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7203–7219, Online, July 2020. Association for Computational Linguistics.

[15] Tomas MIKOLOV, Kai CHEN, Greg CORRADO, and Jeffrey DEAN. Efficient estimation of word representations in vector space. In *the International Conference on Learning Representations (ICLR)*, 2013.

[16] Tomas MIKOLOV, Ilya SUTSKEVER, Kai CHEN, Greg CORRADO, and Jeffrey DEAN. Distributed representations of words and phrases and their compositionality. In *26th International Conference on Neural Information Processing Systems*, 2013.

[17] Hubert SÉGUIN. Compte rendu de [fréquences d'utilisation des mots en français écrit contemporain. jean baudot, 1992, les presses de l'université de montréal]. *Revue québécoise de linguistique*, 1993.

[18] Ashish VASWANI, SHAZEER Noam, Niki PARMAR, Jakob USKOREIT, Llion JONES, Aidan N. GOMEZ, Lukas KAISER, and Illia POLOSUKHIN. Attention is all you need. In *The 31 st International Conference on Neural Information Processing Systems*, 2017.

[19] Georges Kingsley ZIPF. *Human Behavior and the Principle of Least Effort : An Introduction to Human Ecology.* Eastford, U.S.A. : Martino Fine Books, 2012.