POLYTECH SORBONNE

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

PROJET FINAL

MAIN5

Analyse automatique de textes

Rapport final

Ben ali Hala
Boina Assef
Ly Sylviane
Mellouki Anass
Nguyen Quoc Son
Touil Soumaia

Encadrement : Xavier Tannier
Clients : Benjamin Granger
Aude Goldsztejn
(Département Qualité, Gestion des Risques et Expérience Patients)

AP-HP. Sorbonne Université

2019-2020

Table des matières

1 Introduction **3**

2 Concepts utiles **4**

3 Etat de l'art **6**

3.1 Contexte de travail 6

3.2 Les approches courantes 6

3.3 Terminologie du problème 7

4 Démarche **8**

5 Labellisation **8**

6 Nettoyage des verbatims **11**

6.1 Nettoyage simple 11

6.2 Retrait des stop-words 11

6.3 Correction orthographique 11

6.4 Lemmatisation 12

6.5 Exemple de transformation 12

7 Les modèles **12**

7.1 Scikit learn 13

7.1.1 Transformation de mots et extraction des vecteurs de caractéristiques 13

7.1.2 Scikit learn : les classifieurs 16

7.2 Optimisation des performances : les paramètres 17

7.3 CNN 18

8 Analyse des résultats **20**

8.1 Métriques de performance 20

8.2 Comparaison des modèles 20

8.3 Étude des faux négatifs 22

8.4 Gestion du multi-labels 22

9 Le Livrable **23**

9.1 Présentation 23

9.2 Temps d'éxecution 23

9.3 Résultats des premiers verbatims 24

10 Conclusion **25**

11 Bibliographie **26**

1 Introduction

Chaque trimestre, l'AP-HP soumet un questionnaire à ses patients (à la sortie de leur hospitalisation par exemple, après les rendez vous ...). Le but de ces questionnaires est de connaître les points positifs et négatifs rencontrés (par exemple à propos de la disponibilité du personnel, la compétence, l'information autour du soins, le relationnel...), afin de cibler des pistes d'amélioration. Cependant ces verbatims sont traités manuellement. En effet la tâche n'est pas coordonnée car il y a peu de données dans chaque service mais en même temps il y a énormément de données au niveau de l'AP-HP globalement.

Ce rapport présente une analyse automatique des verbatims patients de l'AP-HP. Un précédent travail a été effectué grâce à la technique LDA qui a permis d'explorer les sujets évoqués dans les messages. Ainsi l'AP-HP en a déduit 4 classes principales. Notre rôle est de poursuivre la modélisation du problème. Nous allons produire un outil de classification supervisée qui prendra en entrée un texte plus ou moins court et nous donnera en sortie la ou les classes thématiques concernées par le texte.

Le NLP (Natural Language Processing) est une branche de l'intelligence artificielle visant à comprendre, interpréter et manipuler le langage humain. Ainsi, il doit conjuguer les disciplines informatiques et linguistiques afin de faire cohabiter la compréhension de la machine et la communication humaine. Cette discipline a connu une croissance importante ces dernières années grâce aux avancées récentes en intelligence artificielle et est maintenant appliquée dans plusieurs domaines. De nombreuses entreprises et chercheurs en linguistique informatique s'intéressent à l'analyse automatique du contenu. Cette discipline se retrouve au cœur des débats avec l'avènement des médias sociaux et le Big data.

Par exemple, les outils de traduction pour traduire en fonction du contexte et non terme à terme est un champ du NLP, tout comme les assistants personnels intelligents qui utilisent la reconnaissance vocale ou les moteurs de recherche. La classification de textes est l'une des applications de traitement du langage naturel (NLP) les plus utilisées. Par exemple, prédire la catégorie (thématique) d'un article informatif donné , d'un post sur un forum, ou la détection de spam sont des problèmes usuels.

Notre projet s'inscrit donc dans ce vaste domaine puisqu'il vise à prédire (classer) en fonction du contexte médical donné par un verbatim. La difficulté réside dans la complexité et les nuances du langage humain.

2 Concepts utiles

Machine learning :

Est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d' «apprendre» à partir de données.
L'enjeu est l'utilisation des données et de leurs structures sous-jacentes. Les principaux algorithmes sont les régressions (linéaire, logistique), Naive Bayes, k-means, les arbres de décision, Random forest et les SVM.

Deep Learning :

S'inspire du fonctionnement du cerveau (avec des réseaux de neurones) pour combler les faiblesses du machine learning. L'algorithme va estimer les paramètres qu'ils jugent utiles ou non. Il utilise des couches successivement. Chaque couche prend en entrée la sortie de la précédente.
Des exemples de réseaux de neurones : le perceptron, les réseaux de neurones convolutifs (CNN), récurrents (RNN) etc.

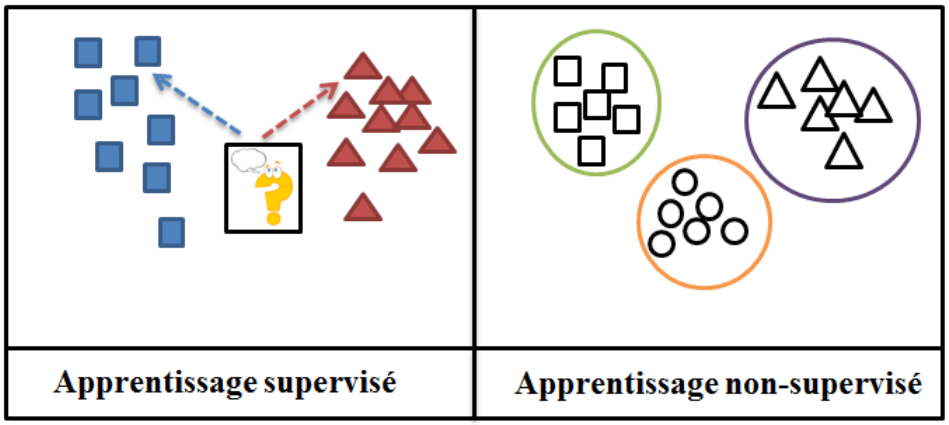

FIGURE 1 – Classification supervisée VS Non supervisée, source `https://fr.wikipedia.org/wiki/Apprentissage_non_supervis%C3%A9`

Apprentissage non supervisé :

contrairement à son homologue supervisé. On cherche à construire une fonction de prédiction à partir d'exemples qui ne sont pas annotés.
On dispose d'éléments non classés et on souhaite faire apparaître d'éventuelles structures. Le but est donc d'extraire des classes ou groupes d'individus présentant des caractéristiques communes, c'est ce qu'on appelle le clustering et les classes sont les clusters. Ni l'appartenance des individus à un groupe, ni le nombre de classes n'est connu, et on n'a aucune connaissance préalable sur celles-ci. C'est ce que l'on cherche à obtenir. L'objectif est donc exploratoire, on veut distinguer des classes pour mieux comprendre nos données.
Des exemples de problème non supervisé : les systèmes de recommandation (Netflix, Spotify, Youtube, Amazon pour les plus connus) afin de proposer des oeuvres proches de celle que l'on a déjà vu pour des vidéos/films, entendu pour de la musique ou de produits d'achat.

Apprentissage supervisé :

tâche de l'apprentissage automatique, et du domaine de l'intelligence artificielle qui consiste à apprendre une fonction de prédiction à partir d'exemples qui sont annotés. On dispose d'éléments dont on connaît déjà la classe d'appartenance, c'est l'échantillon d'apprentissage. A partir de ces exemples, on entraîne le modèle à choisir un groupe. L'objectif est de classer un nouvel élément, c'est-à-dire de décider dans quel groupe l'affecter à partir de ses prédicateurs ou variables explicatives. On distingue les problèmes de régression et les problèmes de classification.
Des exemples de problème supervisé : on connaît les symptômes présentés par un patient, nous voulons prédire un risque médical (survie/mort), ou lorsque se pose le problème de classification de mails (spam/non-spam) selon le sujet/thème du message. Notre travail est de produire un outil, se basant sur un jeu de données préalablement labellisé (training set) et dont on connaît le/les classes d'appartenance, capable de classer une nouvelle entrée. Nous sommes donc dans de la classification supervisée.

LDA (Linear Discriminant Analysis) :

fait partie des techniques d'analyse discriminante prédictive. Technique de classification adaptée lorsqu'il y a plus de deux classes (multi-classes). Il s'agit d'expliquer et de prédire l'appartenance d'un individu à une classe prédéfinie à partir de ses caractéristiques mesurées à l'aide de variables prédictives. Il réduit le nombre de dimensions dans un jeu de données tout en conservant le plus d'informations possibles. Par exemple, supposons que nous avons tracé la relation entre deux variables où chaque couleur représente une classe différente.

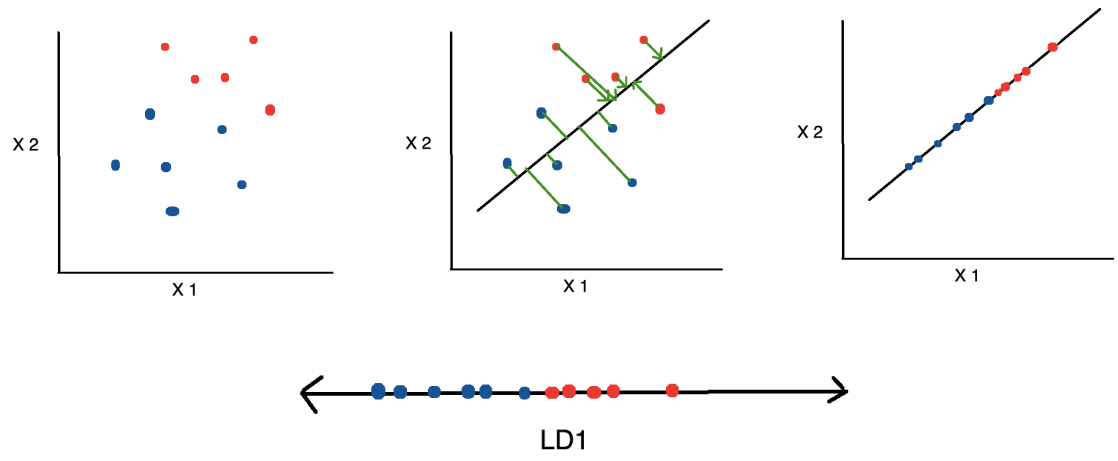

FIGURE 2 – Principe du LDA

Dans l'optique de réduire à 1 le nombre de dimension, on peut essayer de projeter les données sur l'axe X. Seulement, on perdrait toute information utile donnée par l'autre axe. En revanche, LDA utilise les informations des deux features pour créer un nouvel axe et projette les données sur le nouvel axe de manière à minimiser la variance (dispersion) et maximiser la séparation (la distance entre les moyennes) entre les deux classes.

Classifieur :

algorithme de classement dont le but est de classer dans des groupes les échantillons ayant des propriétés similaires. Un classifieur linéaire est un type particulier de classifieur, qui calcule la décision par combinaison linéaire des caractéristiques des échantillons.

Surapprentissage :

Le surapprentissage ou overfitting désigne le fait que le modèle prédictif produit par l'algorithme de Machine Learning s'adapte bien à un jeu d'entraînement en particulier, et même trop, dans le sens qu'il lui "colle" trop. Il obtiendra donc de bons résultats sur ces données, mais il aura du mal à se généraliser sur de nouvelles observations et donc à les prédire efficacement. Cela peut être la cause des mauvaises performances obtenues par le modèle.
On peut par exemple trouver une analogie dans l'éducation, lorsqu'un enfant va apprendre par coeur une leçon, mais ne va pas pouvoir répondre à des questions ou faire des exercices qui sortent de son confort, car ce sont des exemples qu'il n'a pas rencontrés lors de la phase d'apprentissage et/ou il n'a pas compris.

3 Etat de l'art

3.1 Contexte de travail

Pour débuter le projet, nous disposons de 3 fichiers comprenant un large nombre de verbatims non annotés provenant de différents hôpitaux de l'AP-HP. Les verbatims ont été séparés entre ceux qui faisaient suite à une consultation et ceux qui provenaient d'un questionnaire d'hospitalisation.

Ensuite, les verbatims sont organisés par hôpital et par service (ex : oncologie, hépatologie, chirurgie etc). Enfin, les verbatims sont organisés par polarité : une colonne est dédiée à ce qui a été trouvé positif d'après les patients, une autre à ce qu'ils ont trouvé négatif. Nous avons fusionné les différents fichiers pour ne pas tenir compte de la provenance des verbatims (séjour ou consultation). Nous avons également mélangé les verbatims pour ignorer leur polarité et leur origine (hôpital et service). Au final, nous avons un jeu de données harmonisé.

Ainsi, notre travail sera réutilisable sans contraintes liées au service ou au type d'intervention médicale. Cela nous a semblé plus logique car le langage est commun à tous les patients quel que soit le service où ils ont été pris en charge.

3.2 Les approches courantes

Tout d'abord, nous nous sommes intéréssés aux différentes approches pour la classification de textes courts. Nous avons recherchés des algorithmes qui s'appliqueraient dans notre cas. Ils sont notamment utilisés dans les réseaux sociaux, type Twitter avec les Tweets des utilisateurs qui sont des messages courts.
Les deux approches sur lesquelles nous nous sommes basé pour créer nos modèles :

- Approche Machine Learning
- Approche Deep Learning

Nous avons donc étudié les différents algorithmes de Machine learning ainsi que l'outil dédié comme la librairie Scikit Learn.
Ensuite nous nous sommes interessés aux différents types de réseaux de neurones ainsi que les outils qui nous permettent de mettre en place un apprentissage profond. Ces outils sont PyTorch-Keras pour le deep learning.

Des tutoriels sur l'analyse de sentiments ou la classification de textes existent expliquant la façon dont est traité le texte afin de pouvoir être interpreté par la machine. Il s'agit de la vectorisation de documents. L'idée est que les textes, vus comme des chaînes de caractères, sont transformés en représentation mathématique(numérique) reconnu par le modèle .
Nous verrons dans la suite du rapport les méthodes de transformation et vectorisation de textes :

- Bag of word
- Word embedding

En effet, ce sont les mots, unité principale d'un texte qui vont nous intéresser, et notamment des mots clés. On ne va pas prendre en compte les articles comme "de","la","les" etc car ils ne sont pas pertinents, pas assez informatifs (les stopwords). Pour ce genre de considération, nous allons mener une réflexion dans la partie pré-traitement de texte .

3.3 Terminologie du problème

La classification multilabels (ou multi-étiquettes) est le problème d'assigner à chaque instance plusieurs labels. Ainsi, une instance peut être classée dans une ou plusieurs classes. Dans notre cas, un verbatim peut appartenir à un ou plusieurs labels.
Il ne faut pas la confondre avec la classification multi-classes qui est le problème de classifier une instance parmi trois classes ou plus, et non deux, qui est de la classification binaire. Le multilabels est donc une généralisation.

Après cette distinction, nous avons conclu que notre projet est un problème de classification multi-classes (nous avons quatre classes, détaillées dans la partie Labellisation), multilabels (un verbatim peut appartenir à plusieurs classes), or certains algorithmes n'autorisent pas l'intronisation de plus de deux classes de par leur nature binaire ce qui nous pousse à adapter les approches utilisés à notre problème.

Néanmoins des techniques existent pour passer de la classification multi-classes à de la classification binaire. Dans notre cas, nous avons décidé pour simplifier de considérer le problème comme un problème de classification binaire pour chaque classe qui la discrimine contre tout le reste des autres .
La question qui revient alors pour chacun de nos couples instance/classe est donc la suivante : cette instance appartient-elle ou non (1 ou 0) à cette classe ? On peut traiter chaque classe individuellement car elles sont indépendantes les unes des autres. Cela revient donc à poser quatre classifieurs indépendants. Nous sommes face à un

Problème de Classification supervisée multi-classes et multilabels

4 Démarche

Les étapes que nous avons suivi sont brièvement présentées ci-dessous et détaillées ci-après.

FIGURE 3 – Processus

1. Etape de labellisation : durant cette étape nous avons labellisé des verbatims conformément à l'apprentissage supervisé car les fichiers que nous avons reçu n'indiquaient pas à quelle classe appartenait un verbatim. Les verbatims peuvent être uni-classe, ou multi-classes, c'est-à-dire qu'un texte donné peut appartenir à différentes classes. Par exemple :
   " La qualité des repas ... et le professionnalisme du médecin ..." , s'inscrit dans la classe environnement par la mention de la "qualité des repas" et dans la classe du relationnel en parlant du "professionnalisme du médecin".
2. Etape de nettoyage des données : dans cette partie nous avons préparé les données afin d'avoir un texte clair sans fautes pour l'étape d'entrainement.
3. Etape d'entrainement des modèles : enfin, une fois les données labellisées et nettoyées, nous avons entraîné les modèles choisis avec cet échantillon d'apprentissage.
4. Analyse des résultats : cette étape nous permet de comparer les résultats obtenus avec les différents modèles entraînés.

5 Labellisation

L'apprentissage supervisé tient son essence dans l'association des couples (inputs, targets). Dans notre cas les inputs sont les verbatims et les targets sont les différentes classes d'appartenance de chaque verbatim. Dans ce sens afin de pouvoir entraîner des modèles de classification, il est indispensable d'annoter ces verbatims afin de créer ces targets.
Cette annotation repose sur le fait de classer les différents verbatims dans une une ou plusieurs classes en fonction du sens du message. Ces différentes classes sont au nombres de 4 et nous ont été données au départ du projet. Ces dernières ont été obtenus par l'application de la méthode LDA. Nous pouvons voir ci dessous ces différentes classes accompagnées de leur description.

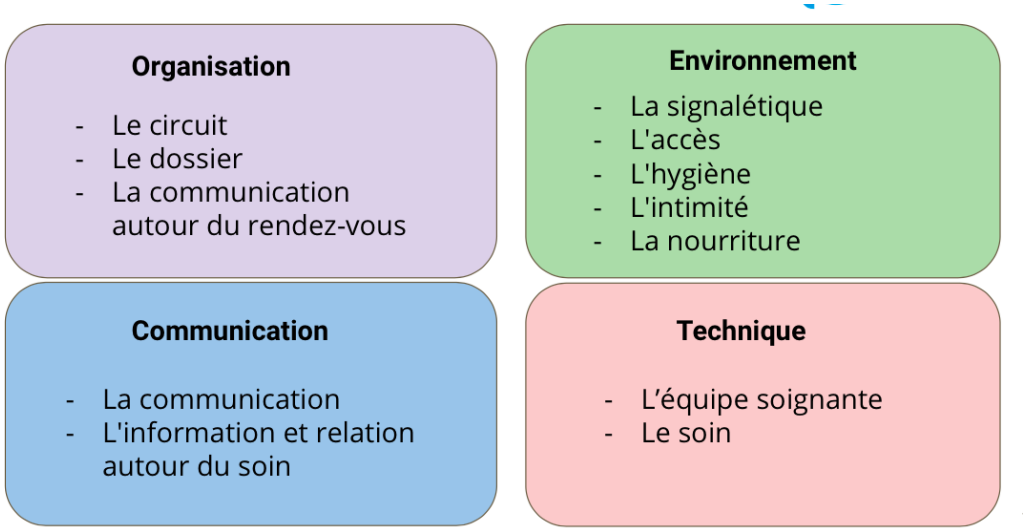

FIGURE 4 – Les 4 classes obtenues via la methode LDA

Il faut savoir que nous étions libre de changer ces classes ou de les subdiviser si l'une d'elle nous paraissait trop vaste. Nous avons uniquement décidé de renommer une classe : à la place de Communication était employé le terme relationnel qui nous était assez général. La classe Organisation concerne la prise en charge dans sa globalité (le circuit) notamment au niveau administratif.

Dans un premier temps pour bien cerner le problème et la difficulté de cette tâche nous avons annoté une centaine de verbatims afin d'établir une guideline que chacun devra suivre par la suite lorsqu'il annotera de façon autonome. Nous nous sommes alors imposés de prendre en compte en particulier l'idée principale du verbatim. En effet, comme nous l'avons précisé précédemment un bon nombre de verbatims mentionnent des plusieurs sujets ce qui pourrait nous encourager fortement à les classer fréquemment dans une multitude de classes. Cette mesure que nous avons pris nous permettait alors de classer ces verbatims uniquement dans les classes qui se dégageait explicitement de ce message. Dans un second temps, nous avons alors attribué à chaque membre du groupe 400 verbatims à annoter sachant qu'il y avait un recouvrement avec les autres membres du groupes. Cet overlap nous permettra de calculer l'accord inter-annotateurs. Ce dernier nous apporte une garantie de régularité sur la labellisation ainsi que sur la qualité du jeu de données d'entrainement. On devait alors s'arranger afin que notre overlap soit significatif. En conclusion, si l'accord inter-annotateurs est faible c'est que les données ont été mal annotées. Deux raisons peuvent expliquer cela :

— Une ou deux personnes ont mal annoté
— La tâche est difficile

Ce processus est itératif et plusieurs itérations sont nécessaires afin de s'assurer une cohérence entre les annotations des différents membres du groupe. A chaque itération une phase d'adjudication est importante afin de communiquer sur les verbatims qui ont posé problème et de se mettre d'accord sur le choix à prendre en fonction du cas rencontré. En outre pour comprendre les éventuelles difficultés des annotateurs, nous avons réalisé une matrice inter-annotateurs permettant de voir si les annotations sont les mêmes d'une personne à une autre et donc de voir si un membre du groupe appréhende les annotations de façon différente. Au bout de la deuxième itération, nos annotations étaient assez homogènes et cohérentes d'une personne à l'autre. Dans la figure ci-dessous, voici la heatmap issue de la seconde itération :

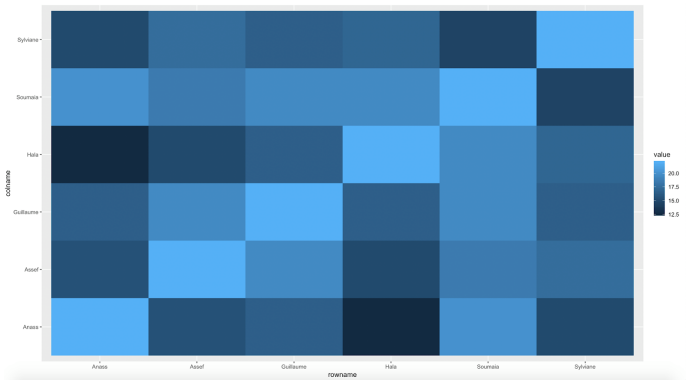

FIGURE 5 – Heatmap issu de la 2e itération du processus de labellisation

Lors de cette itération nous avons obtenu un minimum de 80% de similitudes dans les annotations pour chaque paire. On a alors estimé que ce pourcentage était satisfaisant afin de conserver une cohérence dans nos annotations. En effet, un manque de cohérence dans la labellisation ne permettrait pas à nos différents modèles d'apprentissage de saisir les liens entre un verbatim et sa classe d'appartenance durant la phase d'entraînement. L'efficacité du modèle en serait alors grandement affaiblit. Nous avons donc annoté, à raison de 400 verbatims par personne, un total de 2400 verbatims. Néanmoins, chacun des membres devait tacher de se prémunir d'un autre biais qui pouvait impacter la qualité de nos annotations, en l'occurrence le biais intra annotateur. Ce biais intra annotateur peut être causé par un changement de condition pendant la phase de labellisation, par exemple annoter une partie un jour et une autre le lendemain. En effet, les labellisations peuvent être différentes selon des facteurs tels que la fatigue ou selon la présence de verbatims complexes.

Ce jeu de données composés de 2400 verbatims devrait être suffisant pour bien entraîner nos modèles d'apprentissage statistique.
Après avoir labellisé l'ensemble des données, nous avons étudié les proportions des verbatims par classes. Nous avons constaté que la proportion de verbatims dans la classe communication est plus importante. Cependant la proportion des autres classes reste importante et nous permet d'appliquer tout de même nos modèles d'apprentissage sans ce soucier d'éventuels déséquilibres.

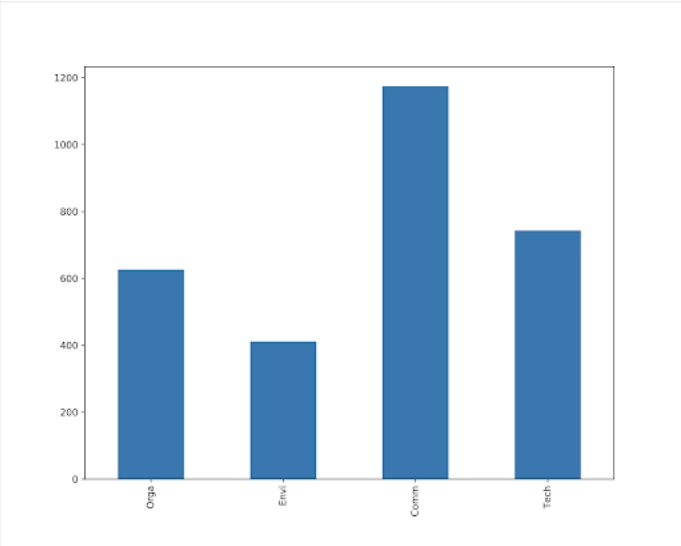

FIGURE 6 – Proportions des classes au sein du jeu de données d'entrainement

6 Nettoyage des verbatims

Les verbatims sont des commentaires laissés par des patients. Ces textes bruts ne sont pas facilement exploitables pour l'analyse de textes. Il est donc préferable de les nettoyer et de les normaliser avant de les étudier.

6.1 Nettoyage simple

Premièrement, nous nous sommes concentrés sur les choses facilement corrigeables. Par exemple, des chaînes de caractères <LF> (Line feed pour le retour à la ligne) ou encore <SEQ> dans les textes ont été retirées.

6.2 Retrait des stop-words

Les stop-words sont les mots qui n'apportent pas d'informations importantes quant au sujet de la phrase. Exemple de stop-words :"à","de","ta","ce" etc..

Il est donc intéressant de les retirer car ils peuvent influencer négativement les résultats. Pour les supprimer, il faut d'abord séparer les textes en mots grâce à la librairie Spacy puis retirer les mots qui font partie de la liste des stop-words de Nltk.

6.3 Correction orthographique

Une fois les stop-words retirés, on corrige les fautes d'orthographe des verbatims grâce à un système de distance. On télécharge une liste des mots de la langue française et on compare chaque mot à ceux du verbatim. C'est le processus le plus long du nettoyage mais également l'un des plus importants.

Exemple de correction de verbatim :

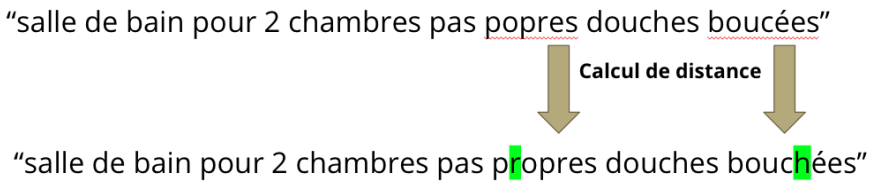

FIGURE 7 – illustration de correction d'orthographe

6.4 Lemmatisation

La lemmatisation est un processus qui permet de transformer un mot en un autre, résumant le sens global de celui-ci. Par exemple, les marques de pluriel, de genre ou encore de conjugaison sont enlevés. Lemmatiser nos verbatims permet alors de rassembler les mots d'une même famille et donc de réduire drastiquement le nombre de mots présent dans notre jeu de données, ce qui permettra à notre modèle statistique d'être beaucoup plus efficace.

Exemple de Lemmatisation de verbatim :

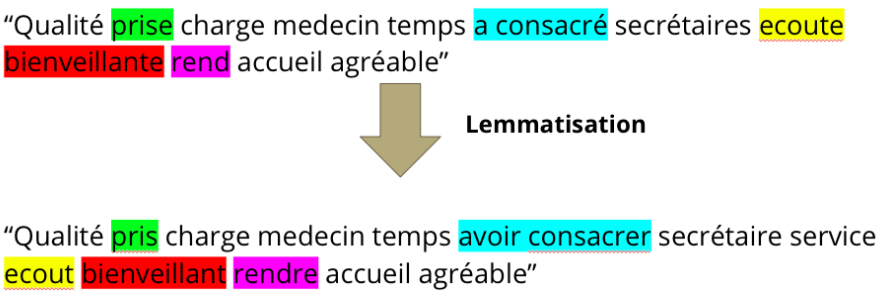

FIGURE 8 – illustration de la lemmatisation

6.5 Exemple de transformation

"LES EXPLICATIONS ET LE SUIVIT DU TRAITEMENT SUR MA MALADIE"

Devient
"explication suivre traitement maladie"

7 Les modèles

Une fois les données labellisées et nettoyées, nous nous sommes interessés aux différentes méthodes de classification supervisée.

Les deux approches que nous avons testées sont les suivantes :

- Approche Machine Learning : Consiste à utiliser différents algorithmes de machine learning pour l'apprentissage supervisé à l'aide de la librairie Python **Scikit learn**.
- Approche Deep Learning : Consiste à utiliser l'apprentissage profond (Réseau de neurones de convolution) pour un problème de classification supervisé à l'aide de la librairie Python **Keras**.

7.1 Scikit learn

Scikit learn est une bibliothèque d'apprentissage statistique en Python.
C'est le moteur de beaucoup d'applications de l'intelligence artificielle et de la science des données. Elle met à disposition des algorithmes de classifications pour plusieurs configurations. Nous l'avons utilisé pour construire nos différents modèles.

Dans cette section, nous verrons les différentes étapes du processus d'apprentissage :

- Chargement du contenu du fichier et des catégories.
- Transformation des mots et extraction des vecteurs de caractéristiques adaptés à l'apprentissage automatique.
- Création du modèle pour effectuer la catégorisation.

7.1.1 Transformation de mots et extraction des vecteurs de caractéristiques

Pour utiliser les données en apprentissage machine, il est nécessaire de leur trouver une représentation mathématique et les adapter pour les modèles. Typiquement une transformation en vecteurs sera plus appréciable qu'une chaîne de caractères pour les algorithmes.
Certaines données s'y prêtent directement, comme par exemple les images, qui engendrent des vecteurs riches en information, encodant toutes les nuances et les couleurs qui les composent.
Les mots, quant à eux, sont des éléments d'information isolés, et certaines représentations rudimentaires se limitent à un simple identifiant par mot.
Par exemple le mot « chat » sera encodé par un seul identifiant arbitraire. C'est une représentation discrète.
La librairie Scikit learn dispose de plusieurs outils pour effectuer cela. Un outil parlant est le **"bag of words model"**, une méthode qui transforme les données textuelles en une représentation discrête selon le nombre d' occurences du mot dans un articles de mots.

Un problème réside dans le fait que cette représentation est relativement pauvre. Elle ne permet pas de comparer deux mots entre eux. Utilisons les prolongements lexicaux qui représentent un mot par un vecteur de nombres réels.
Par exemple, "Homme" sera représenté par le vecteur [0,43 0,88 0,98 1,3]. Si l'on encode tous les mots d'un dictionnaire ainsi, il devient alors possible de comparer les vecteurs des mots entre eux, par exemple en mesurant **l'angle entre les vecteurs**.
Une bonne représentation de mots permettra alors de trouver que le mot « Femme » est plus

relativement proche du mot « Homme ».

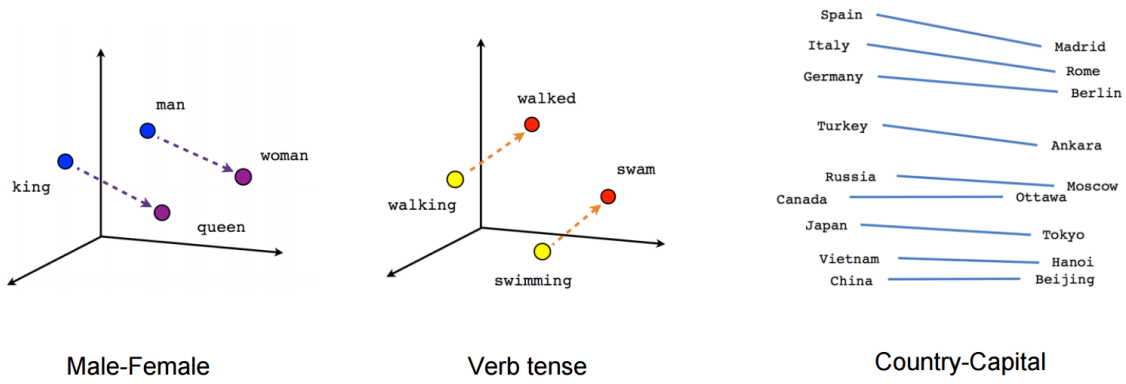

FIGURE 9 – Illustration de la proximité des mots

Ces représentations permettent d'espérer que, dans l'espace vectoriel où le prolongement est fait, on aura les équations suivantes .

$$\begin{cases} roi - homme + femme = reine \\ Madrid - Spain + Germany = Berlin \end{cases}$$

=> Nous traiterons dans cette partie les méthodes de "bag of words" et "word embeddings" que nous avons utilisés pour adapter notre jeu de données aux modèles que nous voudrons créer.
Ces deux méthodes nous ont permis de transformer les chaînes de caractères contenue dans nos fichiers en matériel compréhensible pour l'ordinateur.

=> Ce sont des techniques de traitement de texte (natural language processing) très utilisées dans la recherche d'informations (information retrieval).

• **Bag of words**

Cette méthodes consiste à :

1. Attribuer un identifiant d'entier fixe à chaque mot apparaissant dans n'importe quel verbatim de l'ensemble d'apprentissage (par exemple en créant un dictionnaire de mots avec des indices entiers).

2. Pour chaque donnée i , comptez le nombre d'occurrences de chaque mot w et stockez-le comme valeur de fonctionnalité où est l'index du mot dans le dictionnaire.

=> Pour avoir cette structure de données, les verbatims doivent être découpés en mots, c'est ce qu'on appelle la tokenisation, une étape indispensable dans la NLP.

• CountVectorizer() : aussi appelé one hot encoding, une fonction de scikit learn qui créé un dictionnaire de fonctionnalités avec tous les mots du corpus de texte et compte le nombre d'occurence de chaque token.

=> nous avons utilisé CountVectorizer pour le prétraitement du texte, la tokenisation et le filtrage des mots vides dans notre jeu de données.
En sortie cette fonction nous renvoie un vecteur pour chaque verbatim. Donc au final nous aurons une matrice avec en colonnes les différents mots et en lignes chaque verbatim.

Se baser sur l'occurence des mots peut être une solution dans un premier temps mais il y a un problème : les documents plus longs auront des valeurs de comptage moyennes plus élevées que les documents plus courts, même s'ils peuvent parler des mêmes sujets.
Pour éviter ces divergences potentielles, il suffit de remplacer l'occurence du mot par sa fréquence en divisant le nombre d'occurrences de chaque mot dans un verbatim par le nombre total de mots dans le même verbatim.
Ces nouvelles fonctionnalités sont appelées tf «Term Frequency».
Un autre raffinement, au-dessus de tf consiste à modifier le poids des mots qui apparaissent dans de nombreux verbatim et sont donc moins informatifs que ceux qui ne se produisent que dans une plus petite partie des verbatims.
Cette réduction d'échelle est appelée tf – idf «Term Frequency Frequency Inverse Document Frequency».

Les deux tf et tf-idf black peuvent être calculées en utilisant TfidfTransformer

- TfidfTransformer() : $TF-IDF = TF(termfrequency) * IDF(inversetermfrequency)$. Cette technique se base sur la fréquence d'apparition d'un mot pour lui accorder une meilleure importance qu'un simple compte comme avec CountVectorizer().

=>Dans notre travail nous avons transformé notre matrice de comptage en une représentation tf-idf.

Nous avons choisi de calculer la fréquence d'un mot pour éviter que les longs documents masquent les plus petits. En effet face à un long paragraphe, un mot sera sûrement comptabilisé plusieurs fois tandis que dans un court texte le même mot n'apparaîtra qu'une seule fois. Et pourtant il peut être très important mais la longueur de l'autre document cache cela. La fréquence ramène donc les mots proportionnellement à la taille du document pour ne pas fausser les résultats de prédiction.

De plus les nombreux articles ou mots de liaisons (stopwords) auront une grande fréquence car ils sont très utilisés mais apportent peu au sens de la phrase. C'est pour cela que l'on calcule l'inverse de la fréquence. Ces nombreux mots auront un poids plus faible dans l'apprentissage de la catégorisation. Des mots-clés pourront être mis en évidence.

- Hashing vectorizer() : les textes sont transformés grâce à une fonction (dite de hachage). Une taille de vecteur (pour représenter notre texte) est fixée, cela permet de ne pas avoir des vecteurs ou des matrices de taille trop grande ce qui peut limiter les calculs.

- **Word embedding**

Word embedding : représente les mots sous forme de vecteurs de telle sorte que deux mots sémantiquement proches auront des vecteurs proches dans l'espace.

Il est donc possible d'obtenir de nouveaux liens entre les mots comme leur contexte , des statistiques .
Ces techniques qui utilisent le deep learning sont disponibles via la librairie Gensim sous python.
Il existe plusieurs méthodes de prolongement (word embeddings) sous python tels que :

- Word2vect.
- Glove.
- asText.

Dans la suite de notre travail nous avons continué avec Word2Vect.

Word2Vect : algorithme de word embedding. Il se compose de modèles qui sont des réseaux neuronaux à deux couches peu profondes ayant une couche d'entrée, une couche cachée et une couche de sortie. Il cherche à apprendre les représentations vectorielles des mots composant un texte, de telle sorte que les mots qui partagent des contextes similaires seront représentés par des vecteurs numériques proches. Word2Vec possède deux architectures neuronales, appelées CBOW et Skip-Gram.

CBOW reçoit en entrée le contexte d'un mot, c'est-à-dire les termes qui l'entourent dans une phrase, et essaye de prédire le mot en question. Skip-Gram fait exactement le contraire : elle prend en entrée un mot et essaye de prédire son contexte.

7.1.2 Scikit learn : les classifieurs

Après la transformation de mots et extraction des vecteurs de caractéristiques, nous avons testé plusieurs classifieurs scikit Learn indépendamment sur nos données :

- MultinomialNB : utilisé pour la classification de textes vectorisés par comptage des occurences. Les paramètres sont le alpha, le class prior et le fit prior.

- LinearSVC : pour la classification, accepte des classes déséquilibrées, des matrices creuses. Fait parti de la famille des SVM (support vector machine) avec un noyau linéaire.

- LogisticRegression() : méthode de régression pour la classification binaire .

- Linear Support Vector Classification (fait parti des SVM)] : utilise la descente stochastique de gradient (SGD).

- RandomForestClassifier : algorithme des forêts d'arbres aléatoires, le premier paramètre est le nombre d'arbres dans la forêt(n_estimators), la profondeur maximale de l'arbre, une mesure de l'impureté d'un noeud (criterion). Par défaut, c'est l'indice de Gini qui est utilisé comme critère de séparation. Les arbres aléatoires sont une amélioration des arbres de décision, introduisant de la variabilité grâce au bootstrap.

=> Ces classifieurs sont des classifieurs binaires appliqués que dans le cas de deux classes.
Une solution pour notre cas avec 4 classes nous avons 4 classifieurs indépendants du même type, chacun discrimine une classe par rapport à toutes les autres classes .

7.2 Optimisation des performances : les paramètres

GridSearchCV est la fonction qui va nous permettre de faire varier les paramètres (figure 10), afin de trouver la meilleure combinaison (hyperparamater tuning), tout en faisant de la validation croisée k-fold. On rappelle que son principe est de diviser l'échantillon original en k échantillons, puis on sélectionne un des k échantillons comme ensemble de validation et les k-1 autres échantillons constitueront l'ensemble d'apprentissage. On recommence le processus k fois en sélectionnant un autre parmi les k-1 échantillons . On prend une moyenne des k scores comme score final.

Évidemment, les paramètres ne vont pas être les mêmes selon le classifieur que l'on choisit. L'objectif sera de trouver la meilleure combinaison et le meilleur classifieur parmi ceux cités en 7.1.2.

```
    parameters = {
    'tfidf__use_idf': (True, False),
    'clf__C': [0.5,1,1.5,2,2.5],
    'clf__class_weight':['balanced', None],
}
```

FIGURE 10 – Listes de paramètres de LinearSVC fournies à GridSearchCV

Le nombre de combinaisons correspond au produit des cardinaux de chaque paramètre. Dans l'exemple ci-dessus, il est de $2 * 5 * 2 = 20$. On peut obtenir un classement des combinaisons en fonction du scoring, et définir le scoring (precision, recall etc).

=> Cette méthode nous a montré que le classifieur LinearSVC avec des paramètres {tfid_use_idf : False , clf_C : 0.5 , clf_class_weight : None} est le meilleur parmis tous les autres classifieurs.

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf__C	param_clf__class_weight	param_tfidf__use_idf	params	mean_test_score	std_test_score	rank_test_score
0	0.048047	0.001261	0.013040	0.003995	0.5	balanced	True	{'clf__C': 0.5, 'clf__class_weight': 'balanced', 'tfidf__use_idf': True}	0.752578	0.051618	11
1	0.054848	0.010601	0.011926	0.001895	0.5	balanced	False	{'clf__C': 0.5, 'clf__class_weight': 'balanced', 'tfidf__use_idf': False}	0.744875	0.046358	13
2	0.052935	0.002864	0.012088	0.000269	0.5	None	True	{'clf__C': 0.5, 'clf__class_weight': None, 'tfidf__use_idf': True}	0.809058	0.052749	2
3	0.058827	0.004894	0.013973	0.001188	0.5	None	False	{'clf__C': 0.5, 'clf__class_weight': None, 'tfidf__use_idf': False}	0.815746	0.055393	1
4	0.081547	0.006237	0.017096	0.000814	1	balanced	True	{'clf__C': 1, 'clf__class_weight': 'balanced', 'tfidf__use_idf': True}	0.748059	0.054072	12

FIGURE 11 – Les performances de LinearSVC selon les différentes combinaisons

7.3 CNN

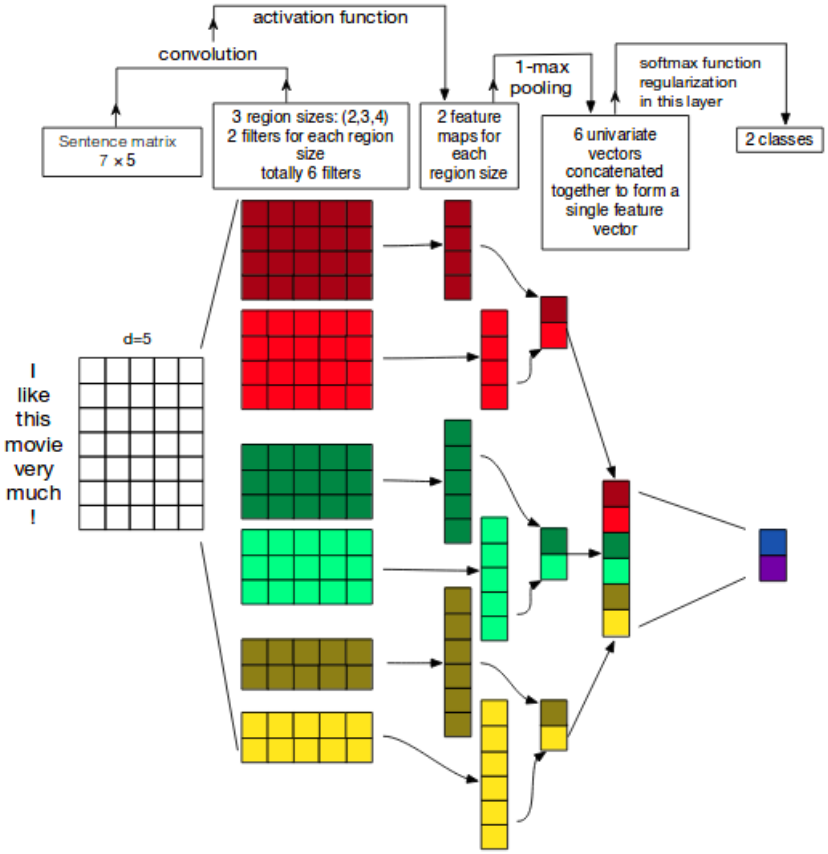

FIGURE 12 – Architecture CNN

Pour cette partie le but était de faire de la classification de texte sur une architecture CNN. Nous avons trouvé un modèle en open source d'un CNN, utilisant l'API keras. Après avoir nettoyé et séparé nos données à 70% pour le train et 30% pour le test, nous utilisons word2vec afin de vectoriser les mots dans le verbatim d'entrée. En effet, le CNN recoit en entrée les verbatims sous forme de vecteurs pour la phase d'apprentissage ainsi que pour la phases de prédiction.

Dans un premier temps une transformation des textes en vecteurs est effectuée, suivie d'une étape d'apprentissage des vecteurs sur un réseau de neurones convolutionnel. En effet, le modèle convolutionnel permet l'extraction des principales caractéristiques des textes représentés à l'aide de mot incorporé.

Les réseaux avec des couches de convolution et de regroupement sont utiles pour les tâches de classification dans lesquelles nous nous attendons à trouver des indices locaux forts concernant l'appartenance à une classe, mais ces indices peuvent apparaître à différents endroits dans une phrase. Les textes sont lus par le réseau mot par mot (Word Embedding) :

Le “Word Embedding” est une représentation distribuée de mots où différents mots ayant une signification similaire (en fonction de leur utilisation).

Ainsi, chaque mot se voit attribuer un entier et cet entier est placé dans une liste. Comme toutes les phrases d'apprentissage doivent avoir la même forme d'entrée, nous remplissons les phrases. Par exemple, si nous avons cette phrase «Comment fonctionne le texte pour la séquence et le remplissage». Chaque mot se voit attribuer un numéro. Nous supposons "comment" = 1, "fonctionne" = 2, "le" = 3, "texte" = 4, "pour" = 5, "la" = 6, "séquence" = 7... Une fois cette attribution effectuée, notre phrase ressemblera à [1, 2, 3, 4, 5, 6, 7...].

Suite à cette étape d'attribution, nous transformons les mots en "vecteurs" à partir du modèle Word2Vec de Google Actualités et ils sont enregistrés en fonction du numéro de séquence que nous avons attribué à chaque mot. Ensuite le texte sous forme de vecteur est transmis à un CNN.

Une fois l'apprentissage effectué, notre modèle est capable de prédire si le verbatim donné en entrée fait partie ou non de la classe qu'on souhaite vérifier. Autrement dit, nous avons un classifieur pour chaque classe, et ainsi chaque CNN est capable de détecter si le verbatim fait partie de sa classe .

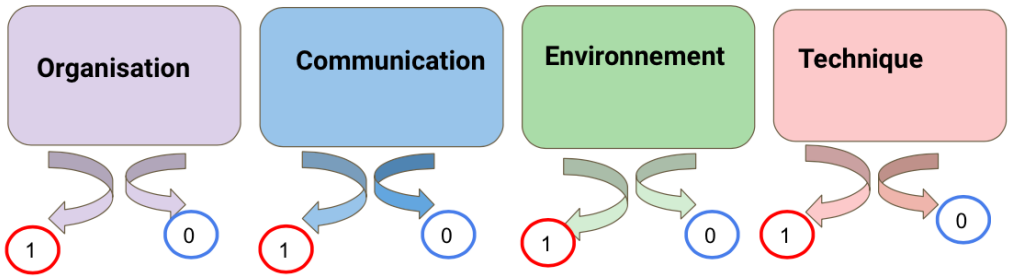

FIGURE 13 – Les classifieurs CNN

Pour déterminer la classe d'un verbatim, il faut lancer le CNN pour chaque classe (Organisation, Environnement, Communication et Technique).

8 Analyse des résultats

8.1 Métriques de performance

`Scikit learn` propose de nombreuses métriques parmi lesquelles nous utiliserons :

Matrice de confusion : présente les bonnes prédictions et les mauvaises prédictions d'après la véritable classe

`accuracy_score` : donne la proportion de bonnes classifications

Precision : Est la proportion des items pertinents parmi l'ensemble des items proposés, elle représente la fraction des données qui ont été bien reconnues parmi les positifs

$$P = \frac{T_p}{T_p+F_p} \text{ où } T_p \text{ sont les vrais positifs et } F_p \text{ sont les faux positifs}$$

Recall : représente la fraction des positifs correctement prédits c'est-à-dire les vrais positifs et les vrais négatifs sont classifiés comme tel

$$R = \frac{T_p}{T_p+F_n} \text{ où } F_n \text{ sont les faux négatifs.}$$

F-score : moyenne pondérée entre precision et recall $2 * \frac{(precision*recall)}{(precision+recall)}$

8.2 Comparaison des modèles

Dans la suite nous allons comparer les 4 classifieurs binaires avec différents modèles : Scikit Learn avec et sans word embedding et les classifieurs CNN (convolutional neuronal network).

On rappelle que le classifieur utilisé pour Scikit Learn est LinearSVC().

	Scikit Learn sans word embedding	Scikit learn avec word embedding	CNN avec word embedding
Accuracy	0,92 / 0,89 / 0,82 / 0,84	0,88 / 0,93 / 0,81 / 0,85	0,88 / 0,93 / 0,85 / 0,82
f1-score	0,73 / 0,77 / 0,81 / 0,71	0,74 / 0,78 / 0,80 / 0,74	0,77 / 0,84 / 0,82 / 0,70
Recall	**0,62** / 0,71 / 0,80 / 0,63	**0,68** / 0,70 / 0,79 / 0,66	**0,77** / 0,82 / 0,84 / 0,60

FIGURE 14 – Résultats de l'entrainement des modèles
Classes : Organisation / Environnement / Communication / Technique

Ce tableau présente les résultats obtenus pour 3 modèles différents et pour les 4 classes. Nous obtenons de bons résultats d'après l'accuracy et f1-score de chaque modèle. Ce n'est pas ce qui nous a permis de choisir un modèle. Lorsque l'on affiche les rapports de classification qui donnent precision, recall... pour les 0 et les 1 de chaque classe, on se rend compte que la mesure disciminante est le recall.

Comparons la mesure recall pour les prédictions de 0, c'est-à-dire que le verbatim est prédit comme n'appartenant pas à la classe. Prenons les négatifs de Organisation (par exemple). Nous avons **0,62** contre **0,68** contre **0,77** pour Scikit Learn sans word embedding, Scikit Learn et CNN avec word embedding respectivement. Cela signifie en outre que 62% des

verbatims sont correctement prédits comme ne faisant pas partie de Organisation. Par conséquent 38% des verbatims sont dit comme n'appartenant pas à Organisation alors qu'ils y appartiennent. On obtient le meilleur ratio pour le CNN. Il en est de même pour les classes Environnement et Communication. Seule la classe Technique a une moins bonne performance avec le CNN qu'avec les autres modèles.

Les figures ci-dessous permettent de montrer la performance des 4 classifieurs pour un verbatim. En effet un verbatim peut appartenir à 1 ou plusieurs classes. La légende indique "parfait" lorsque toutes les classes ont été correctement prédites, 1 erreur lorsqu'un classifieur n'a pas trouvé la bonne classe 1 fois sur 4 et ainsi de suite. "2 erreurs" signifie que 2 classifieurs se sont trompés.

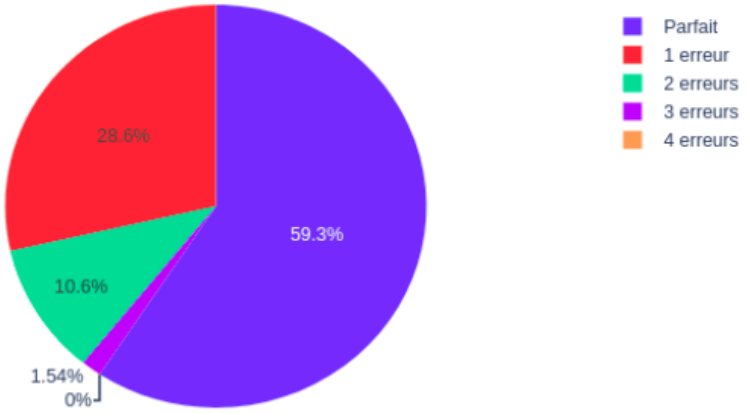

FIGURE 15 – Pourcentages d'erreurs pour Scikit Learn

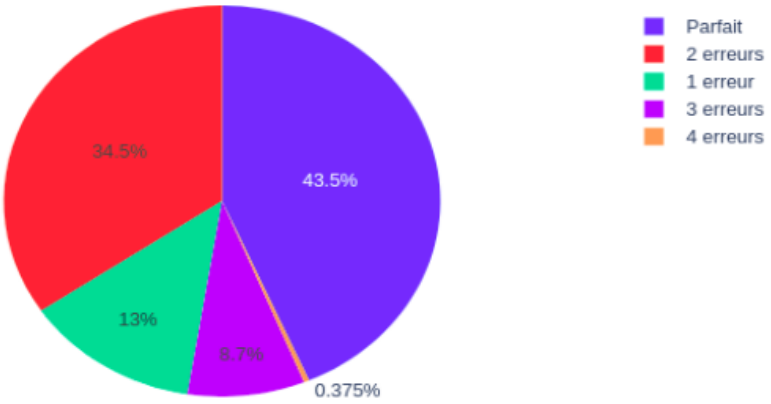

FIGURE 16 – Pourcentages d'erreurs pour le CNN

On observe des pourcentages assez proches. Néanmoins ils ne concernent pas les mêmes erreurs. Le CNN se trompent une fois dans 13% des cas tandis que le modèle avec Scikit Learn se trompe une fois dans 28,6% des cas.

De plus alors qu'il n'y a pas d'erreurs sur les 4 classes avec Scikit Learn, on a 0,375% de verbatims qui ne sont pas correctement prédits du tout. On pourrait penser que cela est

mauvais, cependant lorsque l'on examine plus attentivement les verbatims en question, on comprend pourquoi les classifieurs n'ont pas su affecter le verbatims à une classe. Ce sont des verbatims tels que "rien, tout etre tres bien" (dont le message d'origine est "Rien tout était très bien").

8.3 Étude des faux négatifs

Après avoir observé les recall des différents modèles il nous a paru important d'observer les faux négatifs de chaque classe afin de comprendre si deux classes sont étroitement liées. Cette observation nous permettra alors de comprendre quelles classes notre modèle a du mal à distinguer l'une de l'autre. Ceci nous amène à savoir qu'est ce qu'un faux negatif ? Un faux négatif est un verbatim qui a été prédit comme n'appartenant pas à une classe alors qu'il devrait. Ainsi nous observerons dans quelles classes ces faux négatifs ont été placé par erreur. Illustrons ceci avec la répartition des faux négatifs dans la classe Organisation.

- 13,89% des faux négatifs sont des verbatims classés dans Technique au lieu de Organisation
- 63,03% sont prédits comme appartenant à la classe Communication au lieu de Organisation
- 23,08% devraient appartenir à Organisation au lieu ou en plus de Environnement parmi les faux négatifs de Organisation

Dans cet exemple, au vu des pourcentages, le modèle a du mal à distinguer la classe Communication de la classe Organisation.

8.4 Gestion du multi-labels

Une possibilité pour prendre en compte le fait qu'un verbatim puisse appartenir à plusieurs classes aurait été d'utiliser un seul modèle qui nous renverrait en sortie une prédiction sur les 4 classes simultanément. Avec le CNN qui nous donne de bons résultats, cela n'est pas une tâche aisée c'est pourquoi nous avons 4 classifieurs binaires indépendants. Le défaut de cette méthode est de se retrouver avec des verbatims qui ne sont classés dans aucune classe car chaque classifieur a prédit 0. Nous avons alors observer pour nos différents modèles le pourcentage d'instance qui ont été déclaré comme fausse simultanément dans les 4 classes. Il faut savoir que le pourcentage réel d'instances non classées est negligéable car inférieur à 1%.

Scikit learn sans word embedding	Scikit learn avec word embedding	CNN avec word embedding
11,82%	9,39%	6,98%

D'après le tableau ci-dessous le CNN présente le pourcentage d'instances non classées le plus faible, en l'occurrence un peu moins de 7% quand dans le même temps le modèle Scikit Learn sans word embedding possède le pourcentage le plus haut avec quasiment 12%. Ceci nous montre les bienfaits des words embedding car les modèles l'utilisant possèdent un taux de verbatims non classé bien inférieur. Néanmoins au vu de ces taux il serait intéressant de forcer nos différents classifieurs à placer ces instances dans au moins une classe. il existe plusieurs méthodes afin de déterminer dans quelle classe ces verbatims vont être placés. Nous avons

tâché de réaliser cela pour le modèle CNN, nous avons utilisé le softmax comme indicateur. Ainsi lorsque l'instance n'est placée dans aucune classe nous comparons les softmax de chaque classe et optons pour la classe possédant le softmax le plus élevé.

9 Le Livrable

9.1 Présentation

Le livrable rendu au client est composé de plusieurs éléments :

- Du CNN déjà entrainé et stocké. La structure de ce CNN est enregistrée au format JSON et les poids sont enregistrés au format H5.
- Du programme python à executer pour classer de nouveaux verbatims.
- D'un csv dans lequel sont classés tous les verbatims fournis par le client au début du projet.
- D'un README.md dans lequel sont écrits les informations nécessaires à la bonne exécution du code python.

Le livrable est donc un .zip dans lequel le client se retrouvera facilement, le script pourra être lancé de manière régulière à condition d'avoir installé au préalable les librairies python nécessaires (indiquées dans le README.md)

9.2 Temps d'éxecution

Au cours de notre projet, nous avons rencontré des problèmes de temps d'exécution. Nous avons donc fait en sorte que le livrable ne rencontre pas ce même problème.

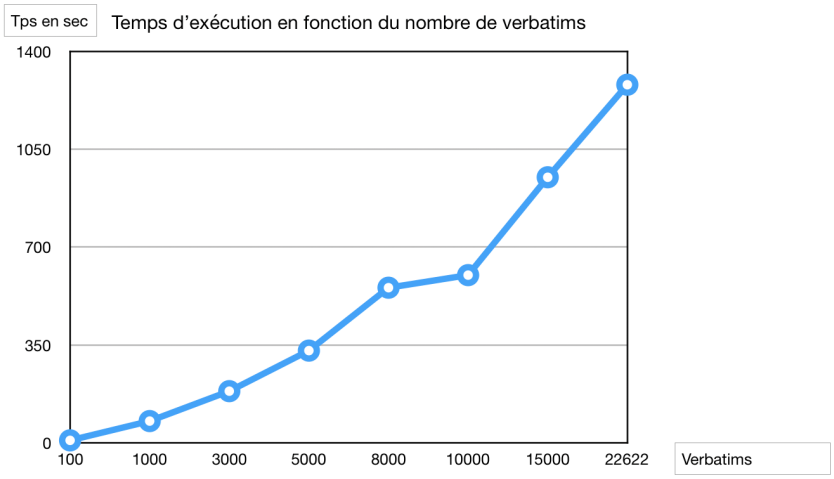

FIGURE 17 – Temps d'exécution du livrable

Nous avons donc fait une petite étude du temps d'exécution du livrable. Celui-ci est plus ou moins linéaire, à raison d'environ 900 verbatims classés par minutes

9.3 Résultats des premiers verbatims

Nous avons, dans ce livrable, la classification des 22622 verbatims fournis par le client au début du projet.

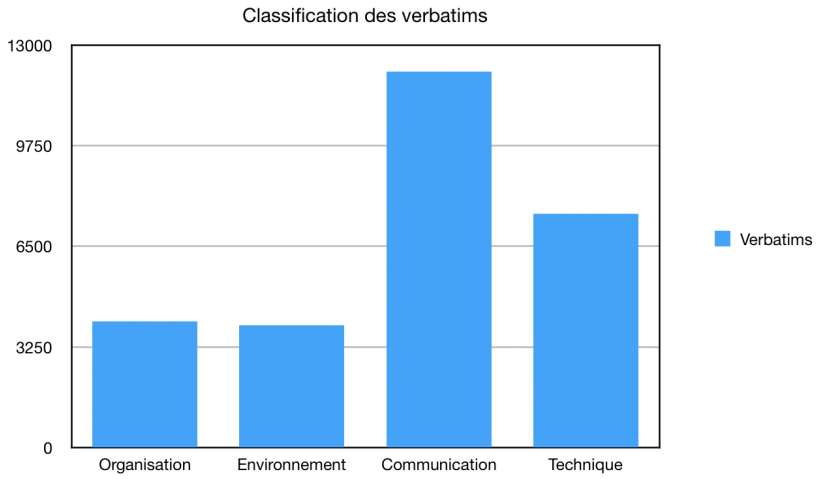

FIGURE 18 – Répartition des verbatims fournis par le client

La plus grande partie de ces verbatims ont été classé dans la catégorie COMMUNICATION mais toutes les catégories sont assez bien représentées.

Certains verbatims sont classés dans plusieurs catégories.

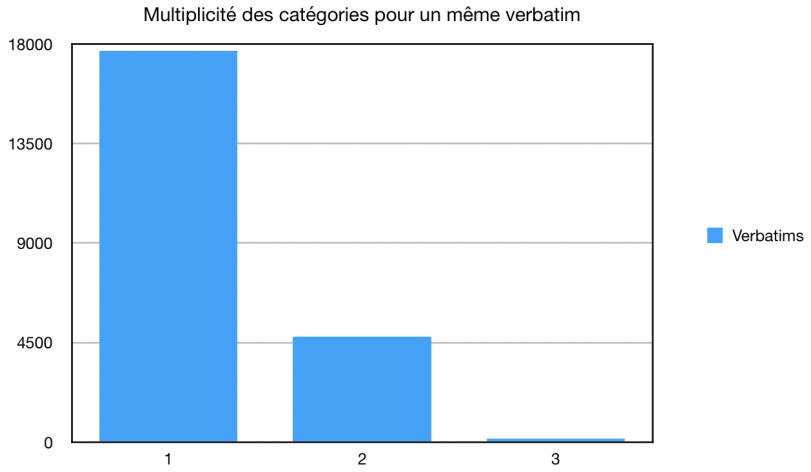

FIGURE 19 – Nombre de catégories pour un verbatim

La grande majorité des verbatims sont classés dans une seule catégorie mais certains appartiennent à deux d'entre elles. Il se peut même, dans de très rare cas, que le verbatim soit classé dans 3 classes en même temps.

10 Conclusion

Au terme de ce projet qui consistait à classifier de courts témoignages de patients afin de mettre en avant des pistes d'amélioration de la qualité des services, nous avons testé différentes approches pour traiter ces textes. En conclusion, utiliser un CNN par classe s'est avéré intéressant et a donné de très bons résultats (de l'ordre des 80% de bonnes prédictions).

Afin de simplifier notre étude, nous n'avons pas pris en compte l'origine des verbatims, c'est-à-dire le service concerné par les témoignages ni les différents hôpitaux dans lesquels ils ont été recueillis. Nous avons aussi fait le choix de regrouper tous les verbatims car le questionnaire se compose d'une partie "qu'avez-vous retenu de positif lors de votre consultation ?" ainsi que d'une partie "qu'avez-vous retenu de négatif lors de votre consultation ?".

Nous avons vu qu'en labellisant plus, on obtenait de meilleurs résultats (precision et recall). Il serait intéressant de voir la limite entre le moment où arrive le sur-apprentissage et le nombre de labellisations effectuées.

Par ailleurs, la correction orthographique prend énormément de temps (3 heures avec la correction pour 3000 verbatims contre 2 minutes sans) et n'apporte aucun gain sur les résultats. Il serait peut-être judicieux de l'enlever lors de l'étape de nettoyage (partie 6) pour des soucis de temps d'exécution, dans l'optique d'une mise en production. Cet aspect est à discuter avec les clients lors de la réunion finale.

Enfin, il est intéressant de voir sur le temps les résultats des prédictions. Est-ce-qu'ils seront toujours aussi fiables dans 1 an ? Se pose ainsi la question de re-entraîner le modèle. Une piste à explorer est le drift analysis pour que les entrées soient toujours conformes.

11 Bibliographie

Wikipedia

CNN :
https://towardsdatascience.com/cnn-sentiment-analysis-1d16b7c5a0e7

Documentation Scikit Learn :
https://scikit-learn.org/stable/index.html
https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html