Projet Final

Polytech Sorbonne

Mathématiques appliquées et informatique

Normalisation de concepts médicaux de comptes-rendus médicaux

Réalisé par :
Mohamed Djerrab
Sophia Hakam
Romaric Kanyamibwa
Boussad Merhane
Kayim Said Bacar

Encadrants :
Xavier Tannier
Coordinatrice :
Frédérique Charles

23 octobre 2019

Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué au succès de ce projet et qui nous ont aidées lors de la rédaction de ce rapport. Tout d'abord, nous tenons à remercier notre enseignant et encadrant, M. Xavier TANNIER (LIMICS), qui nous a consacré du temps et qui nous a été d'une aide précieuse. Son écoute et ses conseils nous ont permis de comprendre les différentes problématiques du sujet.

De plus, nous tenons à remercier vivement Mme. Frédérique CHARLES (LJLL), responsable de la formation et coordinatrice de ce projet final, sans qui la réalisation du projet n'aurait pas été possible. Enfin, nous tenons à remercier toutes les personnes qui nous ont conseillé et relu lors de la rédaction de ce rapport.

Table des matières

Remerciements **1**

1 Pre-processing **4**
1.1 Base de données UMLS 4
1.2 Recherche de la catégorie d'une expression donnée 4
1.3 Méthode 1 : Régression Logistique 4
1.3.1 Méthode 5
1.3.2 Résultat 6
1.4 Méthode 2 : Multi Layer Perceptron (MLP) 7
1.4.1 Concept du perceptron 7
1.4.2 MLP : Multi Layer Percepton 7
1.4.3 Résultats 9

2 Partie 1 : Recherche des codes associés à une expression **10**
2.1 Une première approche : Convolutional Neural Network (CNN) 10
2.1.1 CNN & NLP : *"Comment les CNNs s'appliquent-ils au NLP ?"* . . . 10
2.1.2 Résultats 11
2.2 Construction d'un dictionnaire de codes associés aux termes "stemmisés" . . 13
2.2.1 TF-IDF 13
2.2.2 Stemming 14
2.2.3 Application 14
2.3 Recherche du code le plus proche à partir des similarités fourni par Gensim . 15
2.3.1 Méthodologie 15
2.3.2 Principe du Word2Vec 16
2.3.3 Trouver l'expression la plus proche 17
2.3.4 Résultats 19

Bibliographie **22**

Introduction

Quotidiennement, les hôpitaux prennent en charge des patients avec ou sans antécédents médicaux. Ils recensent les diagnostics réalisés, leur pronostic, leur prescription et les actes médicaux pratiqués sous la forme d'un compte-rendu médical. Ces comptes-rendus médicaux sont répertoriés dans un dossier-patient qui sera transmis d'un praticien à un autre, peu importe sa spécification. Le cumul de comptes-rendus médicaux peut être pénible à la lecture pour les professionnels de santé et entraîner potentiellement des erreurs médicales avec de lourdes conséquences.

Le but de ce projet est de simplifier cette lecture en synthétisant les comptes-rendus médicaux en mots-clés. On parle de normalisation de concepts médicaux de comptes-rendus médicaux. Cette normalisation en concept crée un lien entre les mentions de concepts issues des textes et ceux d'une base de connaissances.

Dans ce projet, nous allons travailler sur une base de connaissances de l'UMLS (Unified Medical Language System). Ce dernier est un recueil de listes exhaustives de termes biomédicaux utiliser pour le développement de systèmes informatiques capables de comprendre le vocabulaire spécialisé utilisé en santé afin de traduire différentes terminologies et faciliter ainsi le traitement du langage naturel. Cette base de données contient donc un ensemble de termes médicaux organisé de manière hiérarchique très complexe et distinguable par un code.

Nous allons ainsi travailler avec différents modèles d'apprentissage statistique afin de détecter les codes associés à une expression et pouvoir par la suite utiliser ces relations dans d'autres applications dans un futur proche, *i.e.* la "dataification" des comptes-rendus médicaux, les recherches Data Science sur de grandes cohortes de patients. Identifier ces relations reste un travail assez fastidieux et demande un pré-traitement assez important (*i.e.* documentation en NLP, pré-traitement dus expressions, stemming, tfidf, etc.) afin de nous rapprocher progressivement d'une solution.

1 Pre-processing

1.1 Base de données UMLS

La base de données UMLS est répartie en différentes catégories en anglais :

— Signes et Symptômes,
— Partie anatomique,
— Éléments chimiques,
— Organismes vivants,
— Maladie,
— Troubles,
— Concept et idées.

Dans chacune de ces bases, nous avons choisi de les partitionner de la manière suivante :

— **Code CUI** (Concept Unique Identifiers) : unique, composé de la lettre C suivie de sept chiffres, correspondant à un ou plusieurs expressions associées à un concept médical ;
— **Nom** : correspondant à une expression ou à un groupe de mots tel qu'il sera trouvé dans un compte-rendu médical rédigé selon le lexique UMLS ;
— **Catégorie** : correspondant à un groupe de concepts médicaux.

Chaque partie comporte des dizaines de milliers de données. Mettre en place une classification semble être la méthode la plus naturelle. Pour ce faire, nous allons ainsi transformer nos données afin de nous focaliser sur les groupes de mots correspondant aux différentes expressions associées à un concept.

1.2 Recherche de la catégorie d'une expression donnée

1.3 Méthode 1 : Régression Logistique

Afin de mettre en place notre classification et d'entraîner notre modèle, nous fusionnons nos jeux de données dans des proportions équilibrées et leur ajoutons un label indiquant l'origine du fichier.

Nous représentons les mots d'une expression sous la forme d'un vecteur dont la taille est le nombre de mots composant le nouveau jeu de données à l'aide du TF-IDF, une méthode de la bibliothèque scikit learn. Le vecteur est composé d'indices correspondant aux mots le composant avec la valeur du TF-IDF pour le distinguer. Le jeu de données devient ainsi une grande matrice très creuse que nous stockons au format *sparse matrix*, méthode issue de la bibliothèque python *numpy*. Nous avons décidé de n'utiliser que le "Nom" pour prédire

l'origine du fichier. Pour ce faire, nous mettons en place une régression logistique.

1.3.1 Méthode

La régression logistique est un algorithme d'apprentissage statistique très connue. Elle est souvent utilisée pour différentes étapes de classification. L'algorithme de la régression logistique utilise une équation linéaire (1) avec des paramètres θ_i indépendants pour prédire une valeur z.

$$z = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_3 + \theta_4 \cdot x_4 \ldots \tag{1}$$

Nous avons besoin que la sortie de l'algorithme soit une variable de classe, *i.e.* 0-non, 1-oui. Par conséquent, nous écrasons la sortie de l'équation linéaire sur l'intervalle [0,1]. Pour "écraser" la valeur prédite dans cette intervalle, nous utilisons la fonction sigmoïde (2).

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

Pour cela, nous prenons l'équation (1) et donnons à la fonction $g(x)$, qui renvoie une valeur écrasée h, la valeur h sera comprise entre 0 et 1.

Toutefois, dans notre cas, nous n'avons pas seulement deux classes mais plusieurs classes représentant les différents concepts. Nous allons donc utiliser l'algorithme de régression logistique comme un classifieur multivarié. Pour ce faire, le principe de "one vs all" est utilisé. Le concept est simple : il consiste à découper le problème de classification multi-classes en une multitude de problèmes de classification binaires.

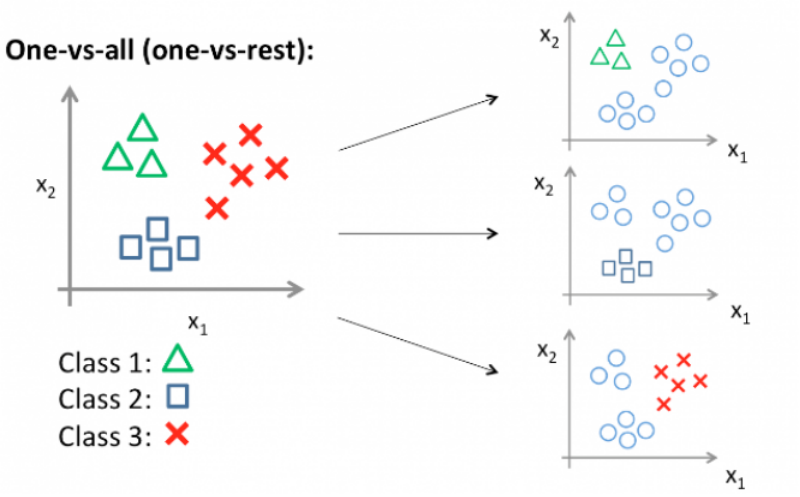

FIGURE 1 – Principe du "one vs all"

Chacune des fonctions de prédiction $g^1(z)$, $g^2(z)$, $g^3(z) \dots g^n(z)$ nous donnera la probabilité que z soit de la classe Y_i. La bonne classe de l'observation x est celle pour laquelle on a obtenu la plus grande probabilité.

En d'autres termes, $z = \underset{i \in \{1,2,\dots,K\}}{max} g^i(x)$, avec K le nombre de classes possibles.

1.3.2 Résultat

Nous avons choisi de travailler avec des données équilibrés pour maximiser la précision globale et non pas en la concentrant uniquement sur une ou deux classes.
Afin de valider notre modèle, nous avons ensuite découpé nos données en deux selon la répartition suivante : un jeu de données d'apprentissage (80%) et un jeu de données test (20%) pour valider notre modèle.

	precision	recall	f1-score	support
0	0.97	0.95	0.96	5346
1	0.98	0.99	0.98	5647
2	0.95	0.94	0.95	5514
3	0.95	0.91	0.93	5480
4	0.85	0.93	0.89	5721
5	0.93	0.89	0.91	5594
micro avg	0.94	0.94	0.94	33302
macro avg	0.94	0.94	0.94	33302
weighted avg	0.94	0.94	0.94	33302

Chaque chiffre obtenu correspond à une des catégories (selon l'UMLS). Nous pouvons aussi observer :

— la `précision` assignée à chaque classe (très élevée) ;
— le `recall` (rappel en français) : correspondant aux nombres d'expressions affectés à la bonne classe parmi tous ceux appartenant réellement à la classe (VRAIS POSITIFS).

La précision de nos résultats correspond au nombre d'expressions correctement attribués à la classe i parmi ceux qui ont été affectés à la classe i.

1.4 Méthode 2 : Multi Layer Perceptron (MLP)

1.4.1 Concept du perceptron

Le perceptron est un algorithme de classification supervisée linéaire. Il est constitué d'un unique neurone qui répond à l'équation suivante :

$$Y = b + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + w_4 \cdot x_4 \cdots = b + \sum_{i=1}^{n} w_i \cdot x_i \tag{3}$$

Un perceptron produit une seule sortie basée sur plusieurs entrées à valeur réelle associées des poids d'entrée formant ainsi une combinaison linéaire. Nous pouvons faire passer la sortie par une fonction d'activation non linéaire.

Soit mathématiquement :

$$y = \phi(\sum_{i=1}^{n} w_i x_i + b) = \phi(W^t x + b) \tag{4}$$

où,

W le vecteur des poids,
x le vecteur des entrées,
b le biais,
ϕ est la fonction d'activation non linéaire.

1.4.2 MLP : Multi Layer Percepton

Le modèle MLP est un algorithme de Deep Learning utilisé en classification supervisée. Il est composé plusieurs perceptrons. Chaque perceptron est lui-même composé d'une couche d'entrée pour recevoir le signal, d'une couche de sortie qui prend une décision ou une prédiction sur l'entrée, et entre les deux, un nombre arbitraire de couches cachées qui sont le véritable moteur de calcul du MLP. Grâce à leur couche cachée, ils sont capables de se rapprocher de n'importe quelle fonction continue.

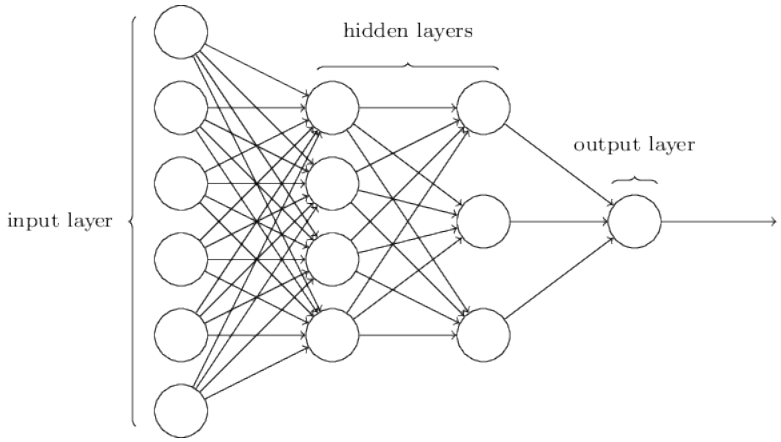

FIGURE 2 – Schéma du modèle MLP

Les MLP sont souvent appliqués à des problèmes d'apprentissages supervisés : ils s'entraînent sur un ensemble de paires entrées-sorties et apprennent à modéliser la corrélation (ou les inter-dépendances) entre ses entrées-sorties. Afin de minimiser l'erreur, la formation des perceptrons résultera de l'ajustement des paramètres, des poids ou des biais du modèle. Ainsi, la rétropropagation est utilisée pour effectuer ces ajustements de pondération et de biais relatifs à l'erreur, et l'erreur elle-même peut être mesurée de diverses façons, notamment par l'erreur quadratique moyenne quadratique (EQMR).

Vers l'avant, le flux de signaux se déplace de la couche d'entrée à la couche de sortie en passant par les couches cachées, et la décision de la couche de sortie est mesurée par rapport aux étiquettes de vérité de terrain. Vers l'arrière, en utilisant la rétropropagation et la règle de la suite de calcul, les dérivées partielles de la fonction d'erreur des différents poids et biais sont rétropropagés par le MLP. Cet acte de différenciation nous donne un gradient, ou une plage d'erreur, au cours duquel les paramètres peuvent être ajustés au fur et à mesure que le MLP se rapproche d'un minimum d'erreur. Ceci peut être fait avec n'importe quel algorithme d'optimisation basé sur le gradient tel que la descente stochastique du gradient. Le réseau continue à jouer au tennis jusq'à ce que l'erreur ne puisse plus baisser. Cet état est connu sous le nom de convergence.

1.4.3 Résultats

Nous avons choisi ici de travailler aussi avec des données équilibrés pour maximiser la précision globale et non pas en la concentrant uniquement sur une ou deux classes.
Afin de valider notre modèle, nous avons ensuite découpé nos données en deux selon la répartition suivante : un jeu de données d'apprentissage (80%) et un jeu de données test (20%) pour valider notre modèle. Nous avons mis en place un MLP à 4 couches de neurones avec la fonction ReLu en guise de fonction d'activation.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1000)              15001000
_________________________________________________________________
activation_1 (Activation)    (None, 1000)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_2 (Dense)              (None, 200)               200200
_________________________________________________________________
activation_2 (Activation)    (None, 200)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_3 (Dense)              (None, 20)                4020
_________________________________________________________________
activation_3 (Activation)    (None, 20)                0
_________________________________________________________________
dropout_3 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_4 (Dense)              (None, 9)                 189
_________________________________________________________________
activation_4 (Activation)    (None, 9)                 0
=================================================================
Total params: 15,205,409
Trainable params: 15,205,409
Non-trainable params: 0
_________________________________________________________________
Train on 26164 samples, validate on 2908 samples
```

FIGURE 3 – Modèle MLP

Nous obtenons une précision de 84%, ce qui est plus faible que celle donnée par la régression logistique. Pour la suite du projet, nous avons donc décidé de travailler avec la régression logistique.

```
Test accuracy: 0.8380795191465
```

FIGURE 4 – Taux de prédiction

2 Partie 1 : Recherche des codes associés à une expression

2.1 Une première approche : Convolutional Neural Network (CNN)

Lors de ce projet, les CNNs nous ont semblé être une piste d'apprentissage à explorer. Les réseaux neuronaux convolutifs (CNNs en anglais) sont des réseaux neuronaux artificiels profonds qui servent principalement à classer les images, à les regrouper par similarité (*e.g.* recherche de photos) et à effectuer de la reconnaissance d'objets. Ce sont des algorithmes qui permettent d'identifier les visages, les individus, les panneaux de signalisation, les tumeurs et de nombreux autres aspects de données visuelles.

Les CNNs utilisent la reconnaissance optique de caractères (ROC) pour numériser le texte et permettre le traitement du langage naturel sur des documents analogiques et manuscrits, où les images sont des symboles à transcrire. Ils peuvent également être appliqués au son lorsqu'il est représenté visuellement sous forme de spectrogramme. Plus récemment, les CNNs ont été appliqués directement à l'analyse de texte ainsi qu'aux données graphiques.

2.1.1 CNN & NLP : *"Comment les CNNs s'appliquent-ils au NLP ?"*

La plupart des tâches du NLP est saisie sous forme de phrases ou de documents représentés sous forme de matrice. Chaque ligne d'une matrice correspond à un token, généralement un mot, mais il peut également s'agir d'un caractère. Autrement dit, chaque ligne est un vecteur représentant un mot.

Typiquement, comme le `Word2Vec`, ces vecteurs sont des embedding (des représentations de faible dimension) des mots, mais ils peuvent aussi être des vecteurs *one-hot encoded* qui indexent le mot dans un vocabulaire. Par exemple, pour une phrase de 10 mots utilisant un encastrement de 100 dimensions, nous aurions une matrice de 10×100 en entrée représentant notre "image".

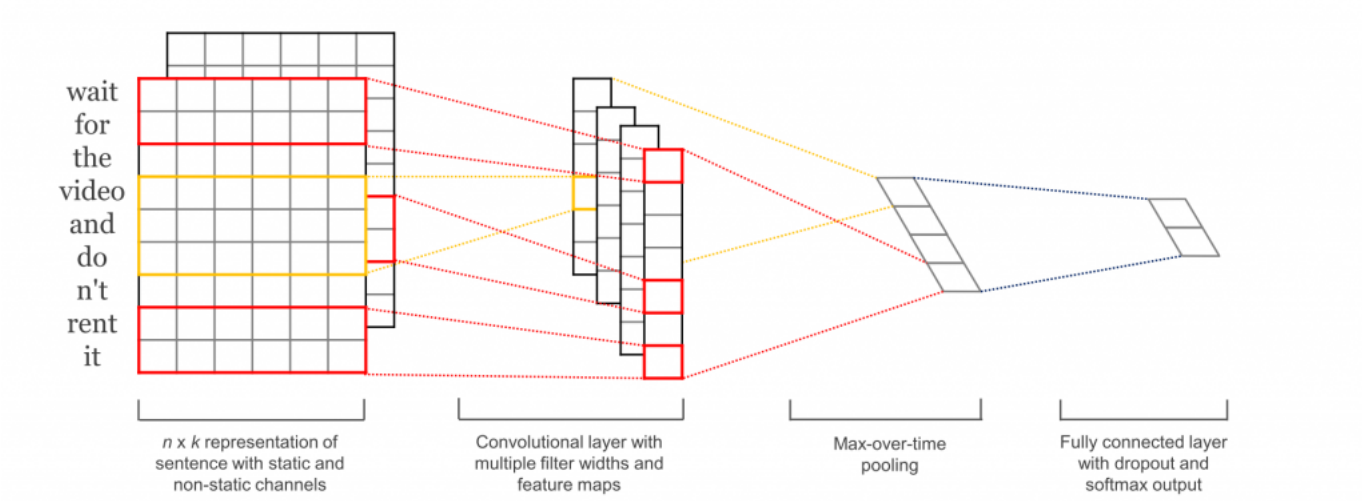

FIGURE 5 – CNN pour le NLP

Cependant, les réseaux neuronaux récurrents (RNN) sont l'approche la plus intuitive. Ils ressemblent à la façon dont nous traitons le langage ou du moins la façon dont nous pensons traiter le langage, *i.e.* une lecture séquentielle de gauche à droite. Néanmoins, notre problème n'est pas adapté à un tel traitement de par la taille de données et le nombre de classes qui ne nous permet pas d'implémenter des modèles complexes comme les RNN.
Ainsi, les CNNs restent des modèles plus simples, applicables et efficaces en termes de représentation et faciles à implémenter (au niveau matériel sur les GPU).

2.1.2 Résultats

Avec un CNN de la forme décrite dans la section précédente, nous allons essayer de prédire les codes associés à un problème médical donnée (*Disorders*). Pour réaliser notre classification, nous avons tout d'abord créé un `Word2Vec` des données "Disorders" et, par la suite, crée un CNN à une couche de convolution. Ensuite on lui applique un *maxpooling*.Max pooling est un processus de discrétisation basé sur des échantillons. L'objectif est de sous-échantillonner une représentation d'entrée (image, matrice de sortie à couches cachées, etc.), en réduisant sa dimension et en permettant de faire des hypothèses sur les caractéristiques contenues dans les sous-régions sélectionnées. Enfin on fait un *fully-connected* avec 2 couches ayant comme fonction d'activation une *relu* et une *softmax* respectivement.Un *fully-connected* est un MLP qui nous permet de 2D de passer en une dimension et d'apprendre des caractéristiques avec des fonctions d'activation.

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 30, 2, 64)	19200
max_pooling2d_4 (MaxPooling2	(None, 15, 1, 64)	0
flatten_4 (Flatten)	(None, 960)	0
dense_7 (Dense)	(None, 3000)	2883000
dense_8 (Dense)	(None, 22582)	67768582

Total params: 70,670,782
Trainable params: 70,670,782
Non-trainable params: 0

FIGURE 6 – CNN

A la phase d'entraînement, ce CNN nous donne en moyenne une précision de 80% avec une "validation accuracy" d'environ 3%. Lorsque nous testons notre modèle avec différentes tailles de notre *train et test set*, nous obtenons le tableau suivant :

Catégorie de fichiers	Train Size	Test Size	Nombre de code unique	Précision en %
Disorders	19473	499	18372	4.0081
Disorders	18474	1498	18372	3.5381
Disorders	15981	3991	18372	2.3051
Disorders	16978	4992	20080	2.8446
Disorders	18976	5989	22582	3.6233

Nous remarquons que la précision est très loin des 80% obtenus en entraînement mais qu'en revanche, elle est très proche de la "validation accuracy" précdemment obtenue. Autrement dit, notre modèle fait du surapprentissage (*overfiting*). Il marche bien sur les données d'entraînement mais le modèle ne peut pas être généralisé.

FIGURE 7 – Surapprentissage

Après discussion avec notre encadrant, nous nous sommes vite rendu compte que ce résultat n'est pas si étonnant que ça. En effet, en regardant le tableau précédent, nous remarquons que nous voulons effectuer une classification de 20000 données sur 18300 classes (Codes). La base de données contient des dizaines de milliers d'expressions et de codes. En moyenne, à un code donné correspond environ trois expressions rendant un apprentissage correct impossible. Donc nous observons très rapidement que la structure du problème ne nous permet pas de faire une classification CNN classique.

Ainsi, nous avons dû nous orienter vers d'autres méthodes pour pouvoir associer un code à une expression :

— créer un dictionnaire où à un radical correspond plusieurs codes ;

— rechercher du code le plus proche à partir des similarités fourni par Gensim.

2.2 Construction d'un dictionnaire de codes associés aux termes "stemmisés"

Une expression comporte un ensemble de mots. Certains mots peuvent être commun à plusieurs expressions. Or, dans un compte-rendu médical, des mots de même sémantiques peuvent ne pas avoir le même suffixe. De plus, nous ne nous intéresserons qu'aux termes importants, i.e. sans les conjonctions de coordination, les pronoms, les articles, etc. Ainsi, afin d'associer une expression à un code, il nous faut construire un dictionnaire dans lequel un radical serait associé à un ou plusieurs codes. Il nous faut donc dans un premier temps extraire les termes importants et les transformer en gardant leur radical.
Pour ce faire, nous avons utilisé deux méthodes de la bibliothèque python scikit-learn et nltk respectivement : le TF-IDF et le stemming.

2.2.1 TF-IDF

TF-IDF est l'acronyme de "Term Frequency-Inverse Document Frequency". Cette méthode vise à définir l'importance d'un mot-clé ou d'une phrase dans un document. La fréquence d'un terme (TF), comme son nom l'indique, représente la fréquence d'un terme dans un document. Il peut être calculé en divisant le nombre d'utilisations d'un mot-clé dans un document par le nombre total de mots d'un même document.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}} \tag{5}$$

Quant à la fréquence de document inverse (IDF), ce dernier indique la fréquence d'utilisation d'un mot-clé dans un ensemble de documents. Il s'agit d'une mesure corrective visant à minimiser l'importance des articles et de diverses fonctions d'un mot afin de donner plus d'importance aux mots ayant du sens.

$$idf(w) = \log(\frac{N}{df_t}) \tag{6}$$

Le produit de ces deux métriques nous donne la formule du TF-IDF qui indique la pertinence d'un mot-clé pour un document sous la forme d'un score notée w.

$$w_{i,j} = tf_{i,j} * \log(\frac{N}{df_t}) \tag{7}$$

où,

$tf_{i,j}$ nombre d'occurrences de i dans j,
df_t nombre de documents contenant i,
N nombre total de documents.

Plus la valeur du TF-IDF est grande, plus le mot-clé du document est important.

2.2.2 Stemming

Pour des raisons grammaticales, les documents vont utiliser différentes formes de mots. De plus, il existe des familles de mots liés par dérivation avec des significations similaires. Dans de nombreuses situations, comme la nôtre, il est utile de rechercher un de ces mots pour renvoyer des documents contenant un autre mot de l'ensemble. Ainsi, la recherche de racines est importante pour la compréhension du langage naturel (NLU) et le traitement du langage naturel (NLP).

Le stemming, signifiant "création d'un radical", est un processus de réduction d'un mot à son radical qui s'appose sur des suffixes et des préfixes, ou sur des racines de mots et ainsi réduire les formes dérivées d'un mots en un radical commun à ces entités.

L'algorithme le plus courant pour le stemming d'une collection en anglais qui semble à maintes reprises être empiriquement le plus efficace d'après divers champs d'application est l'algorithme de Porter (Porter, 1980). Cette algorithme est beaucoup trop long et trop complexe pour être présenter dans ce rapport. Ainsi, nous le présenterons de façon succincte. Il est composé de 5 phases de réduction de mots, appliquées de manière séquentielle. Au cours de chaque phase, diverses conventions sont appliquées. Par exemple : SSES $\rightarrow$ SS ; IES $\rightarrow$ I ; SS $\rightarrow$ SS ; S $\rightarrow$ S.

Ces règles utilisent des méthodes de "mesure" de mots indiquant le nombre de syllabes d'un mot et vérifiant si ce dernier est assez long pour qu'il soit considéré comme un suffixe plutôt qu'une partie d'un radical d'un mot. Par exemple, "replacement" $\rightarrow$ "replac" ; "cement" $\rightarrow$ "c".

2.2.3 Application

Chaque catégorie possède son dictionnaire. On teste simplement pour la catégorie `Living Beings` pour l'expression test `"cuban treefrog"` que l'on stemmise. On recherche dans le

dictionnaire de cette catégorie les codes associées aux radicaux extraits. On compte leur occurrence et on choisit le maximum à l'aide de la méthode `Counter` de la bibliothèque python `collections`.
Par exemple, dans ce cas-ci : `Counter('C0327060': 2, 'C0327064': 1, 'C0327082': 1, 'C0327066': 1, 'C0327062': 1, 'C1066366': 1, 'C1296229': 1, 'C0327061': 1, 'C0326165': 1, 'C1275609': 1, 'C0319727': 1, 'C0327907': 1, 'C0327063': 1, 'C0327065': 1, 'C0327683': 1, 'C0327053': 1, 'C1828306': 1, 'C0327058': 1, 'C0327059': 1, 'C1269456': 1, 'C1269457': 1)`

Dans d'autres cas, il peut arriver qu'aucun code ne se distingue des autres. Les scores du TF-IDF seront utiles pour mettre en exergue le code correspondant à l'expression la plus proche de l'expression test.

2.3 Recherche du code le plus proche à partir des similarités fourni par Gensim

2.3.1 Méthodologie

L'apprentissage classique ne pouvant fonctionner, il a fallu réfléchir à une autre méthode. Ainsi, nous nous sommes orientés vers une autre possibilité : si l'on ne peut apprendre les codes, on peut tout de même chercher à déterminer l'expression la plus "proche" de notre jeu de données d'entraînement de celle q'on cherche à évaluer.
Par exemple, si notre jeu de données se composent des phrases suivantes :

"Le patient est atteint de troubles de la vision."
"L'enfant possède des difficultés à se concentrer."
"L'homme est atteint de blemmophobie[1]*."*

Posons l'expression test suivante : *"La jeune femme est myope".*

Nous recherchons une similarité maximale entre la première expression de notre jeu d'entraînement et celle de l'expression test, *i.e.* une similarité faible par rapport aux deux autres.

"Comment pouvons-nous quantifier cette proximité ?"
Nous avons vu précédemment q'il était possible de "vectoriser" des mots en utilisant le TF-IDF. Ce dernier témoigne de l'importance d'un terme pour une expression. Dans le cas d'une analyse de similarité, on veut traiter chaque terme avec la même importance pour évaluer leur proximité. Cependant, il faut pouvoir évaluer le contexte. En effet, des mots similaires seront souvent utilisés dans des contextes semblables et donc accompagnés des mêmes termes. Ainsi, la différence principale réside dans une comparaison de vecteurs et non de scores, *i.e.* une recherche de similarités entre les mots plutôt q'à leur importance syntaxique.

1. La phobie du regard des autres qui concerne plus spécifiquement le corps et le regard porté, par autrui, sur celui-ci.

2.3.2 Principe du Word2Vec

Afin de comprendre le principe du `Word2Vec`, parlons tout d'abord des *Word Embedings*. Succintement, les *Word Embedings* sont les textes convertis en nombre. Il peut y avoir différentes représentations numériques du même texte. Actuellement, des algorithmes de Machine Learning et de Deep Learning sont incapables de traiter des chaînes de caractères ou des textes bruts. Les inputs sont nécessairement des nombres q'il s'agisse de classification ou de régression par exemple. Afin de lancer un tel algorithme, il nous faut impérativement transformer un texte en chiffre.

La méthode `Word2Vec` et basé sur la prédiction. En attribuant des probabilités aux mots dans un espace vectoriel, gardant ainsi l'analogie et la similarité des mots. Des chercheurs de Google ont même réussi à avoir le résultat `King - Man + Woman = Queen`.
`Word2Vec` est la combinaison de deux techniques :

— CBOW (Continuous bag of Words)

— Skip-Gram model

Les deux techniques utilisent des réseaux de neurones qui relient les mots à une cible qui est elle-aussi un mot. Les deux techniques apprennent des poids qui représentent le vecteur du mot.

`Word2Vec` utilise une seule couche (*hidden layer*), entièrement connectée au réseau comme dans la figure (8) ci-dessous. Tous les neurones du *hidden layer* sont des neurones linéaires. La couche d'entrée est configurée pour avoir autant de neurones q'il y a de mots dans le vocabulaire d'entraînement. La taille du *hidden layer* est définie en fonction de la dimension des vecteurs de mots résultants. La taille de la couche de sortie est identique à celle de la couche d'entrée. Ainsi, en supposant que le vocabulaire d'apprentissage des vecteurs de mots consiste en V mots et N comme étant la dimension des vecteurs de mots, l'entrée aux connexions des *hidden layer* peut être représentée par une matrice WI de taille `V x N` avec chaque ligne représentant un mot du vocabulaire. De même, les connexions du *hidden layer* à la couche de sortie peuvent être décrites par une matrice WO de taille `N x V`. Dans ce cas, chaque colonne de la matrice WO représente un mot du vocabulaire donné. L'entrée du réseau est codée en utilisant la représentation *"1-out of V"*, ce qui signifie qu'une seule ligne d'entrée est mise à 1 et que les autres lignes d'entrée sont mises à 0.

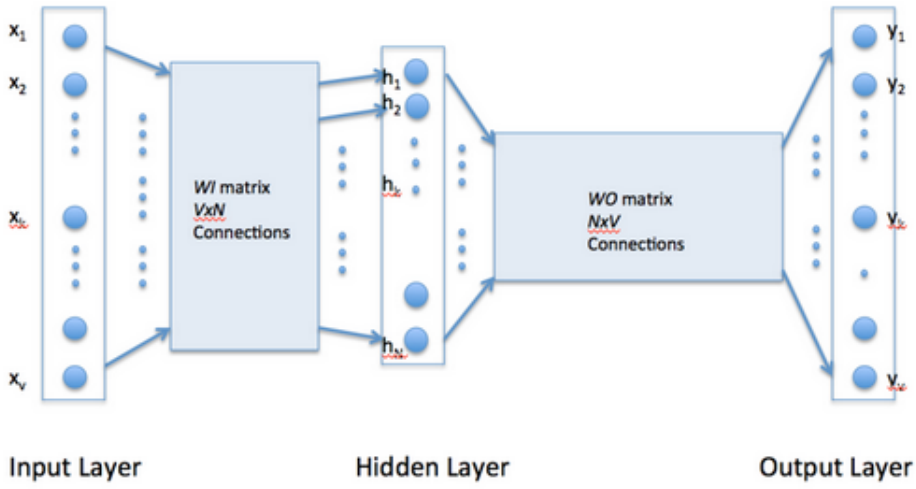

FIGURE 8 – Réseau de neurones du `Word2Vec`

2.3.3 Trouver l'expression la plus proche

Maintenant que les principes du `Word2Vec` sont posés, nous allons détailler comment nous l'avons appliqué.

Il nous a été fourni, un modèle de `Word2Vec` réalisé à partir de MIMIC [2], une base de données accessible à tous, élaboré par le laboratoire de physiologie computationnelle du MIT, qui comprend des données sur la santé associées à environ 40 000 patients en soins intensifs. Il comprend des données démographiques, sur les signes vitaux, les tests de laboratoire, les médicaments et plus encore.

Une fois le modèle appréhendé, la méthode consistait en plusieurs étapes.

En premier, nous utilisons notre technique évoquée précédemment pour récupérer l'origine de notre expression. Une fois cela réalisée, nous créons un dictionnaire à partir du jeu de données de l'origine de notre expression qui auront pour clé les différents codes et pour valeur la liste des expressions associés à ce code. Nous réalisons cela pour faciliter la recherche du code à partir de l'expression qui sera la plus proche.

Nous parcourons ensuite notre dictionnaire pour créer une liste de vecteurs composée de la moyenne de chaque mot. En effet, pour représenter une expression sous la forme d'un vecteur, nous avons choisi de faire la moyenne des coefficients de chaque mot ce qui parait être l'opération la plus logique. Il est également important de noter que nous ne considérons

2. https://mimic.physionet.org/

que les expressions qui font intégralement partie de notre vocabulaire sinon cela fausserait les résultats.

Une fois, notre liste réalisée, nous souhaitons pour une expression donnée retrouver le code associée. Ainsi, nous utilisons la méthode des k-plus proche voisins.

En apprentissage statistique, cette méthode permet de placer les échantillons labellisés dans un espace vectoriel et pour trouver la classe d'un nouvel échantillon donné. On le place dans notre espace et on va regarder les k éléments les plus proches de celui-ci. La classe qui se retrouvera le plus d'occurrences parmi ces k échantillons sera alors désignée comme la classe de notre test.

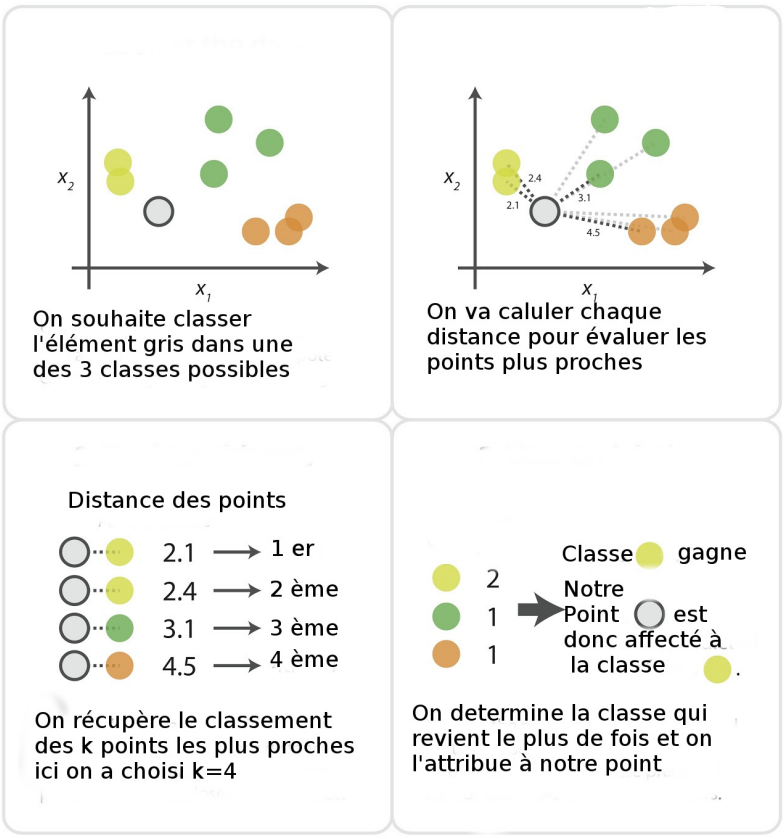

FIGURE 9 – Illustration de la méthode des plus proches voisins

Ainsi, cette méthode peut sembler être une redite du `Word2Vec`. Cependant, nous en avions besoin car le travail effectué est sur les moyennes. Nous avons fixé k à 1 car au vu du nombre de classe il n'est pas intéressant d'aller plus loin. Grâce à cela, nous récupérons le vecteur le plus proche de notre expression et comme nous avons gardé en mémoire les vecteurs via un indice les connectant à leur code, on l'associe à notre expression.

2.3.4 Résultats

Notre modèle nous permet donc d'avoir des résultats pour chaque catégorie de fichier. Pour réaliser les tests, nous découpons le jeu de données en données d'apprentissage qui vont constituer le voisinage à étudier et un jeu de données test sur lequel nous allons chercher le code le plus proche.

Voyons les résultats pour quelques catégories de fichiers à nombre d'échantillon égaux, avec un jeu de données test correspondant à 10% de l'ensemble :

Catégorie de fichiers	Précision en %	Nombre de code unique
Troubles	79.01	10250
Éléments chimiques	70.31	11412
Concept et Idées	51.58	11936
Signes et Symptômes	50.20	12300
Partie anatomiques	35.12	15606

Ici, la précision indique le nombre d'échantillon parmi le jeu de données test ayant eu leur code bien prédit. On voit directement la corrélation entre le nombre de code unique et la précision, ce qui est assez cohérent car moins on a de codes pour un nombre fixe de données et plus on aura d'occurrences du même code, ce qui est bénéfique pour un système basé sur les plus proches voisins et plus généralement pour un modèle de classification.

Conclusion

Au travers de notre projet, nous avons exploré une multitude de méthodes de classification nous ayant offert des résultats assez variés. Classifier des données ayant énormément de labels avec peu de données tests est un problème difficile et d'autres pistes que nous n'avons pas considéré aurait pu être exploité.

Combiner des méthodes d'apprentissage, lors de la mise en place la méthode de nos plus proche voisins, il aurait pu être intéressant de considérer d'autres opérations que la moyenne des vecteurs, notamment en utilisant les CNNs, le problème résidant dans l'expression de la nouvelle fonction de coût à minimiser pour réaliser une classification optimale.
Ce genre de problème aurait pu également être appréhendé par des méthodes basées sur le Zero Shot Learning qui correspond à un ensemble de techniques permettant la classification de données jamais vu auparavant. Ceci peut être intéressant dans la mesure où notre modèle est souvent face à de nouvelles données. Il faudrait donc étudier ces techniques afin de pouvoir les adapter à notre problème.

En ce qui concerne notre travail effectué, nous pensons que nous aurions pu aller plus loin dans le test des modèles et les confronter à des jeux de données différents que celui du test, dans un objectif de vérification mais aussi d'amélioration.
En dernier lieu, il aurait pu être intéressant d'étudier les NLP dans un autre contexte, celui de la segmentation de texte afin de séparer des rapports médicaux en expressions afin de pouvoir appliquer notre modèle directement sur de vrais rapports. Néanmoins, cela aurait demandé un travail différent mais qui apparaît comme la suite naturelle du projet en parallèle des améliorations liées aux modèles.

Répartition des tâches

On a tous commencé par se documenter sur les différentes notions abordés dans ce rapport.

Kayim s'est chargé de mettre en place le modèle de régression logistique sur les données et aussi d'exploiter le `Word2Vec` fourni pour réaliser la méthode des k plus proches voisins afin de retrouver les codes.

Mohamed et Sophia se sont tous deux occupés principalement de la partie pre-proccessing pour pouvoir bien retrouver la catégorie d'une expression donnée et de la création d'un dictionnaire par catégorie.

Romaric et Boussad ont travaillé conjointement sur la partie CNN et `Word2Vec` pour essayer de trouver une bonne implémentation du CNN à la recherche des codes associés à une expression et éventuellement trouver une méthode hybride avec le KNN ou la régression logistique.

Enfin, pour la rédaction du rapport, la répartition a été équitable sachant que nous avions connaissance des notions abordées par chacun dans la réalisation de sa partie.

Bibliographie

[1] Y. Goldberg and O. Levy, "word2vec explained : deriving mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv :1402.3722*, 2014.

[2] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv :1408.5882*, 2014.

[3] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv :1404.2188*, 2014.

[4] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, pp. 649–657, 2015.

[5] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242, pp. 133–142, 2003.

[6] H. C. Wu, R. W. P. Luk, K. F. Wong, and K. L. Kwok, "Interpreting tf-idf term weights as making relevance decisions," *ACM Transactions on Information Systems (TOIS)*, vol. 26, no. 3, p. 13, 2008.

[7] X. Sun, X. Liu, J. Hu, and J. Zhu, "Empirical studies on the nlp techniques for source code data preprocessing," in *Proceedings of the 2014 3rd International Workshop on Evidential Assessment of Software Technologies*, pp. 32–39, ACM, 2014.

[8] A. G. Jivani *et al.*, "A comparative study of stemming algorithms," *Int. J. Comp. Tech. Appl*, vol. 2, no. 6, pp. 1930–1938, 2011.

[9] T. Roska and L. O. Chua, "The cnn universal machine : an analogic array computer," *IEEE Transactions on Circuits and Systems II : Analog and Digital Signal Processing*, vol. 40, no. 3, pp. 163–173, 1993.

[10] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, classifiaction," 1992.