Table des matières

Liste des tableaux

Listes des images

Nomenclature

Machine learning

$Classe$ Valeur rattachée à une instance pour un certain label (ex: rouge, noir, blanc)

$Classification$ Processus de prédiction de la classe d'une instance donné

$Embeddings$ Processus de transformation de texte en vecteur

$Instance$ Objet unique a partir duquel le modèle va apprendre ou sur lequel un modèle va être utilisé (par exemple pour la prédiction)

$label$ Catégorie à prédire (ex: couleur de la voiture)

$Modèle$ Algorithme entraîné à reconnaître des valeurs / paternes dans les instances

$Multi-classes$ Quand une instance peut être associée a un label parmis un ensemble de classes

$Multi-labels$ Quand une instance peut être associée à plusieurs labels

$Poids$ Ensemble de valeurs, au format vectoriel ou matriciel, permettant d'influencer les données d'entrées afin de calculer les prédictions

Symboles romains

$\hat{y}$ Prédiction d'une instance

W, V, U Poids d'un modèle

x Donnée d'entrée (instance)

y Valeur réelle d'une instance

1 Introduction

1.1 Introduction

Les nouvelles technologies, en particulier dans le domaine informatique, ont pris depuis quelques années une place importante au sein de notre société. Elles ont été mises en pratique dans le but d'automatiser les processus métier les plus redondants, simplistes pour commencer, étonnament complexes maintenant. A l'époque, il était très compliqué d'automatiser des processus complexes étant donné les performances modérées des ordinateurs. Cependant, suivant la loi de Moore [1], ces derniers se sont développés à une vitesse fulgurante pour, aujourd'hui, réussir à créer des ordinateurs plus puissants et bien plus petits que tous ceux de l'époque: les smartphones.

Ce rapide développement du matériel informatique a permis à de nouvelles technologies de voir le jour. Certaines sont aujourd'hui tellement efficaces qu'on aurait cru cela impossible il y a quelques dizaines d'années. L'une de celles qui a pu le plus facilement s'améliorer, se développer et devenir actuellement l'une des technologies informatiques les plus en vogue est le "Machine Learning" [2]. Cette dernière a vu le jour dans la seconde moitié du 20ème siècle et est devenue tellement puissante qu'elle est aujourd'hui l'un des atouts majeurs des plus grandes entreprises pour améliorer leurs ventes. Les GAFAs (Google, Apple, Facebook, Amazon, etc) ont investi des sommes astronomiques pour développer et mettre en oeuvre de nouveaux algorithmes. Ces derniers, appelés "modèles", sont aujourd'hui la norme du Machine Learning.

La seconde raison de l'accroissement exponentiel de l'utilisation du Machine Learning est la quantité et la facilité d'accès des données. En effet, la capacité, les performances et le prix du matériel s'améliorant de plus en plus vite, le stockage de données en tout genre et en grandes quantités est facilité. Cet accroissement rapide du nombre de données disponibles a créé ce que nous appelons à présent le "Big Data". En apprenant à exploiter ces données, il est possible d'entraîner des modèles dont la performance de prédiction dépasse celle d'un être humain. Certains processus, jusqu'ici impossible à automatiser, le sont aujourd'hui. La reconnaissance d'images en temps réel, par exemple, est utilisé de manière intensive dans certains pays [3].

D'autres domaines d'études restent extrêmement compliqués à automatiser, même avec l'accroissement du nombre de données disponibles et l'amélioration du matériel informatique. Il est, par exemple, très compliqué d'automatiser les processus dans le domaine de la médecine. En effet, le secret médical oblige, le nombre de données est beaucoup plus faible et, en général, très difficile d'accès. De plus, les processus médicaux sont extrêmement complexes et nécessitent des modèles très performants. En effet, contrairement à d'autres domaines, une erreur logicielle peut avoir des conséquences humaines.

A cause de leur complexité, leur nombre important et leur diversité, il est difficile de stocker toutes les informations d'un patient dans une base de données structurée. La plupart des données médicales sont alors écrites sous forme de texte brut puis stocké, s'il le faut, de manière numérique [4]. Une grande quantité des données médicales, disponibles au format textuel, sont très difficiles à comprendre et à utiliser en Machine Learning. Afin de pouvoir exploiter ces données, il est nécessaire de les transformer préalablement en un format compréhensible par l'ordinateur. Ce domaine du Machine Learning est appelé NLP (Natural Language Processing) [5, 6].

Dans la seconde moitié du 20ème siècle, les algorithmes NLP[6, 5] étaient pour la plupart basés sur des règles complexes écrites à la main. Il était nécessaire d'avoir des règles spécifiques au problème à résoudre et donc non généralisables. Pour chaque problème, de nouvelles règles devaient être créées ce qui demandait des moyens importants. Depuis les années 2010, avec l'avancée technologique du Machine Learning, différents modèles de NLP ont vu le jour, bien plus performants et généralisables que les anciennes règles faites à la main. Ces premiers modèles sont entraînés avec des dictionnaires de millions de mots et de textes anglais puisés dans le Big Data. Ils sont alors très performants dans le langage de tous les jours mais ont plus de difficultés à transformer des textes médicaux sans perdre d'informations. Par la suite, des équipes ont dérivé ces modèles et les ont entraînés sur des données provenant du domaine médical.

Ces derniers modèles permettent actuellement de transformer la plupart des informations textuelles médicales en un format compréhensible par l'ordinateur de manière performante. Il devient donc possible d'automatiser une partie des processus extrêmement complexes de ce domaine. Cependant, il est encore impossible de faire entièrement confiance à un ordinateur et un médecin doit toujours vérifier les prédictions faites par le modèle. Il est possible néanmoins d'alléger son travail en traitant préalablement les informations pour en réduire au maximum la quantité.

1.2 Le projet

Dans le domaine plus spécifique des essais cliniques, chaque étape est décrite de manière très précise dans un protocole. Ce protocole est la marche à suivre afin de réussir l'essai clinique. Dans ce protocole, il est décrit, entre autres, des critères d'inclusion et d'exclusion destinés à sélectionner les personnes pouvant faire partie ou non de l'essai clinique. Cette étape est appelée sélection de cohorte.

La bonne sélection des patients (ou cohorte) est un élément essen tiel de l'essai clinique [7]. Il faut que l'échantillon choisi soit le plus homogène possible afin d'assurer que les effets imaginés du médicament (ou autre élément médical) testé proviennent bien de ce dernier et non d'une cause humaine ou environnementale. Cette sélection se base sur des critères d'inclusion et des critères d'exclusion définis par les chercheurs qui ont imaginé l'essai clinique et écrit le protocole. Actuellement, le choix des patients se fait à la main et demande beaucoup de temps, en raison de la diversité

et du nombre croissant de patients à vérifier.

En se basant sur les critères d'inclusion et d'exclusion, le plus souvent textuels, et sur les données des patients, il est, techniquement, possible de créer un modèle de Machine Learning permettant de faire un tri préliminaire des patients dans le but d'alléger la charge des chercheurs.

1.3 Objectifs

Les objectifs de ce travail sont les suivants:

1. Étudier et comprendre les données du jeu de données 2018 N2C2 shared task (track 1)
2. Rechercher le meilleur modèle d'embeddings pour ce problème
3. Rechercher le meilleur modèle de classification pour ce problème
    - Modèles classiques de Machine Learning: KNN, DecisionTree, RandomForest, Régression logistique, etc
    - Modèles de Deep-learning: MLP, CNN, RNN

1.4 Contenu de ce travail

Ce travail est composé de 6 chapitres. Le premier a pour objectif d'introduire le Machine Learning et son utilisation dans le domaine médical ainsi que d'expliquer les objectifs de ce travail.

Le deuxième chapitre décrit l'état de l'art en commentant ce qui a déjà été fait dans le domaine. Il permet également de se faire une idée générale sur les résultats potentiellement atteignables.

Le troisième chapitre a pour objectif d'expliquer les données utilisées. Cela permet de mieux comprendre pourquoi l'utilisation de données médicales rend l'utilisation d'algorithmes de Machine Learning bien plus compliqué.

Le chapitre numéro quatre donnera une explication détaillée des différents modèles utilisés dans ce travail ainsi que la manière dont ils ont été évalués.

C'est dans le cinquième chapitre que l'ensemble des résultats sont annoncés et expliqués pour chaque modèle.

Pour finir, le chapitre 6 donne un résumé général de l'ensemble du travail et des résultats tout en proposant des recommandations pour de futures recherches dans le domaine de la sélection de cohorte.

2 État de l'art

Dans la recherche biomédicale, il est essentiel de tester de nouveaux processus ou médicaments sur un groupe de personnes qui partagent les mêmes caractéristiques physiques ou physiologiques. Il est beaucoup plus simple d'isoler les effets biologiques d'un traitement dans une population homogène [7]. Cette population homogène est appelée une cohorte. Dans les essais cliniques, définir cette cohorte et sélectionner les patients est un processus crucial demandant beaucoup de temps aux chercheurs en raison de la variété de cas possibles et du grand nombre de dossiers de patients qui doivent être traités manuellement [8].

Grâce aux récentes découvertes technologiques, des essais d'automatisation du processus de sélection de cohorte ont été menés. Cela est un énorme défi en raison des nombreuses manières dont les données ont été récupérées et stockées. Il est possible de formuler le problème de sélection de patient comme un problème de classification multi-labels dont l'objectif est de déterminer quels critères correspondent à chaque patient [8]. Avec les avancements récents dans le traitement du langage naturel (NLP) [6], il devient possible de fournir un accès rapide aux dossiers des patients qui correspondent le mieux à l'étude. Actuellement, quelques algorithmes basés sur des règles [6, 9] sont capables de faire correspondre des patients à des critères avec une précision acceptable mais ne peuvent pas être généralisés aux autres études. En effet, le nombre limité et la complexité des données rendent la généralisation des modèles très compliquée.

D'autres études ont essayé différentes approches utilisant l'apprentissage machine (Machine Learning). Segura Debmar et cie [10] ont essayé différents modèles d'apprentissage classiques et profond pour détecter si des patients sont atteints d'une maladie spécifique sur la base de leurs dossiers médicaux. Pour une maladie spécifique, ils ont réussi à obtenir un score F1 de 95.6% avec un modèle CNN et un score 95.8% avec un modèle SVM, dont la complexité de calcul est bien moindre. Comme vu précédemment, l'utilisation de l'apprentissage approfondi (Deep-Learning) permet d'obtenir de très bons résultats pour une tâche spécifique. Dans une seconde étude, Segura Debmar et Pablo Raez [8] ont essayé de faire correspondre des critères d'inclusion avec des dossiers patients, une tâche bien plus généralisée, en se basant sur les données de la tâche partagée N2C2 [11]. Ils ont utilisé un CNN, un RNN et un modèle hybride (CNN et RNN) et ont testé si l'ajout d'une couche de classification dense et l'initialisation aléatoire des poids avaient un impact positif sur les résultats. Avec leur meilleur modèle (hybride, pas de couche de classification et poids pré-entraînés), ils ont obtenu un score F1 de 78.56%. En général, l'utilisation d'une couche de classification améliore les résultats, sauf pour l'architecture hybride.

La difficulté du problème de sélection de cohorte réside dans la grande diversité des critères. Chaque domaine médical a ses mots et abréviations spécifiques. En raison de cette diversité, il est difficile

de créer un modèle généralisé à l'ensemble des domaines médicaux. Pour ajouter du contexte et de la compréhension aux critères, des connaissances médicales peuvent être injectées dans les textes. Cette architecture aide à identifier la sémantique du domaine médical [12]. Toutes les phrases qui n'ajoutent pas de contexte, basé sur un dictionnaire de mots importants, seront supprimées du texte. L'avantage de cette architecture est qu'elle peut être plus facilement généralisée, en fonction du dictionnaire utilisé. Le MKCNN (Medical Knowledge Convolutional Neural Network) a surpassé certains modèles en obtenant un score F1 de 86.1%.

Une autre approche consiste à ne pas utiliser un seul classificateur multi-labels mais un classificateur pour chaque critère [13]. Chaque classificateur utilise un modèle simple (BoW [14]) mais doit utiliser en contrepartie une approche de correspondance des modèles pour conserver les informations pertinentes du texte. Toutes ces informations sont réintégrées dans le texte sous forme de jetons lexicalement distinguables. Pour les critères comportant un nombre suffisant d'instances, Spasci et cie [13] ont utilisé de l'apprentissage surpervisé. Pour les autres, une approche basée sur des règles a été choisie. Cette architecture a permis d'obtenir un score F1 de 89.04%. Le modèle Gradient Tree Boosting semble être le modèle le plus consistant.

Le plus grand défi de la communauté médicale est la collecte de données. La quantité de données disponibles est très limité et ces données sont souvent incomplètes. La génération de données synthétiques [13, 15, 16, 17], peut résoudre ce problème. En outre, cela permet d'anonymiser les données. Une autre difficulté réside dans le type de données qui peuvent être collectées. Certaines données peuvent être catégoriques mais la plupart sont au format textuel. Il est alors nécessaire d'en extraire les éléments pertinents et de les transformer en un format vectoriel, compréhensible par l'ordinateur. Une approche consiste à utiliser des méthodologies basées sur des règles [18]. Cela prend beaucoup de temps mais donne de très bons résultats. Néanmoins, il est alors très compliqué de généraliser ces transformations à de nouveaux textes. Une seconde approche consiste à utiliser l'incorporation de mots / documents [19]. Ce système permet de créer une représentation unique d'un mot / document, permettant de généraliser le modèle. Il est également possible de combiner les deux approches [20]. De nombreux modèles d'embeddings existent, comme GloVe [21], ELMo [22], BERT [23], Word2Vec [19], etc. Ceux les plus performants actuellement sont basés sur l'architecture Transformers [24], imaginée en 2017. Cette dernière permet d'augmenter grandement le nombre de mots sur lesquels la contextualisation peut être appliquée. BERT [23] est le modèle d'embeddings basé sur l'architecture Transformers le plus connu.

Beaucoup de personnes ont essayé d'utiliser ces outils pour créer les meilleurs modèles de sélection de cohorte. La fondation I2B2 [25] a permis de tester de nombreuses approches dans le domaine de la médecine. Depuis que de nouvelles technologies ont vu le jour, la fondation a changé de nom pour y intégrer le traitement du langage naturel. N2C2 [11] est maintenant une fondation internationale essayant de résoudre des problèmes médicaux en se basant sur le NLP [6]. La tâche partagée créée

en 2018 avait pour but de résoudre le problème de sélection de cohorte. Quarante-sept équipes ont participé et la meilleure a réalisé un score F1 de 91% en utilisant un algorithme basé sur des règles. Les dix meilleures équipes ont toutes utilisé une architecture à base de règles et de modèles hybrides [26]. L'utilisation de ces algorithmes augmente considérablement les résultats mais supprime toute possibilité de généralisation à d'autres problèmes.

3 Les données

3.1 Description des données

Les données utilisées durant ce travail proviennent du challenge 2018 N2C2 shared task (track 1) [26]. Ce dernier a été créé pour résoudre le problème de sélection de cohorte lors d'essais cliniques. Il est composé de données relatives à 288 patients, 202 pour l'entraînement des modèles et 86 pour les tests. Chaque patient est décrit par 2 à 5 enregistrements textuels ainsi que par 13 critères, expliqués au tableau 3.1, pour un total de 1267 instances d'entraînement et 377 instances de test. L'ensemble des enregistrements d'un même patient sont classifiés de la même façon.

Tag	Critère
DRUG-ABUSE	Drug abuse, current or past
ALCOHOL-ABUS	Current alcohol use over weekly recommended limits
ENGLISH	Patient must speak English
MAKES-DECISIONS	Patient must make their own medical decisions
ABDOMINAL	History of intra abdominal surgery, small or large intestine resection or small bowel obstruction
MAJOR-DIABETES	Major diabetes-related complication
ADVANCED-CAD	Advanced cardiovascular disease
MI-6MOS	Myocardial infarction in the past 6 months
KETO-1YR	Diagnosis of ketoacidosis in the past year
DIETSUPP-2MOS	Taken a dietary supplement (excluding Vitamin D) in the past 2 months
ASP-FOR-MI	Use of aspirin to prevent myocardial infarction
HBA1C	Any HbA1c value between 6.5 and 9.5%
CREATININE	Serum creatinine > upper limit of normal

Table 3.1: Critères et tags associés du jeu de données N2C2 shared task (track 1)

Chaque critère est défini par une phrase et abrégé par un tag, comme il est possible de le voir dans le tableau 3.1. Chacun d'eux peuvent avoir comme valeur "met", qui indique que l'enregistrement correspond au critère, ou "no met" qui indique que l'enregistrement ne correspond pas au critère. La plupart des critères relèvent du monde médical mais d'autres, tel que "ENGLISH" ou "MAKES-DECISION" y sont moins spécifiques. Certains critères, par exemple MI-6MOS, ont une signification sur la durée (... in the past 6 months) et nécessitent donc de connaître la date de l'enregistrement. Cette date est indiquée sur la première ligne de chaque enregistrement. Toutes les dates présentes dans les enregistrements sont anonymisées (la date est modifiée) tout en gardant le bon espace temporel entre les différentes dates.

Tous les enregistrements sont écrits en anglais et peuvent prendre des formes diverses: email, rapport médical, ordonnance, etc.

Données	Nombre de caractères			Nombre de mots (séparés aux espaces)			Nombre de tokens (FlairTokenizer)		
	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean
Entraînement	435	21189	4380	74	2984	629	93	3795	808
Test	743	17055	4282	115	2250	618	139	2967	798

Table 3.2: Longueur des enregistrements patients du jeu de données

Comme décrit dans le tableau 3.2, l'enregistrement le plus long comporte 2984 mots pour 21'189 caractères. Le plus court fait 74 mots pour un total de 435 caractères. En moyenne, les enregistrements ont 615 mots pour 4'284 caractères. Bien qu'il y ait de grandes différences dans les longueurs des textes, la distribution de ceux-ci est plutôt égale entre les données d'entraînement et les données de test.

Toutes les données du N2C2 shared task (track 1) sont stockées au format XML. Un fichier est utilisé par patient. La structure de l'arbre XML de chaque fichier est décrite à la figure 3.1.

```
<?xml version="1.0" encoding="UTF-8" ?>
<PatientMatching>
    <TEXT><![CDATA[]]></TEXT>
    <TAGS>
        <ABDOMINAL met="met" />
        <ADVANCED-CAD met="met" />
        <ALCOHOL-ABUSE met="not met" />
        <ASP-FOR-MI met="met" />
        <CREATININE met="not met" />
        <DIETSUPP-2MOS met="met" />
        <DRUG-ABUSE met="not met" />
        <ENGLISH met="met" />
        <HBA1C met="not met" />
        <KETO-1YR met="not met" />
        <MAJOR-DIABETES met="not met" />
        <MAKES-DECISIONS met="met" />
        <MI-6MOS met="met" />
    </TAGS>
</PatientMatching>
```

Figure 3.1: Structure XML des fichiers du jeu de données 2018 N2C2 shared task (track 1). Les valeurs des critères (en bleu foncé) sont un exemple.

Comme visibile à la figure 3.1, la structure XML est découpée en 2 partie : TEXT et TAGS. L'ensemble des enregistrements sont écrits dans la balise TEXT, plus spécifiquement à l'intérieur de la sous-balise CDATA. Cette dernière permet d'éviter que certains caractères des enregistrements soient intérprétés en XML et détruisent la structure de l'arbre. Comme chaque patient peut avoir entre 2 et 5 enregistrements, ceux-ci sont séparés par une ligne contenant 100 astérisques.

Les critères relatifs au patient sont stockés dans la balise TAGS. Chaque critère dispose de sa propre sous-balise et leur valeur est décrite par l'attribut met, pouvant avoir la valeur met ou not met.

Avec un total de seulement 1644 instances, le jeu de données N2C2 shared task (track 1) est très limité en taille, surtout dans un domaine comme la médecine où les données sont très complexes.

3.2 Distribution des données

Une distribution équitable des données est essentielle pour obtenir de bons résultats. Si les données sont inégalement distribuées, il est possible que le modèle n'apprenne que des instances dont la classe est majoritaire et n'arrive finalement pas à prédire les valeurs des classes minoritaires, résultant en un apprentissage sub-optimal. Dans ce cas là, il est fort probable que le modèle prédise uniquement la classe majoritaire, quelles que soient les données d'entrées.

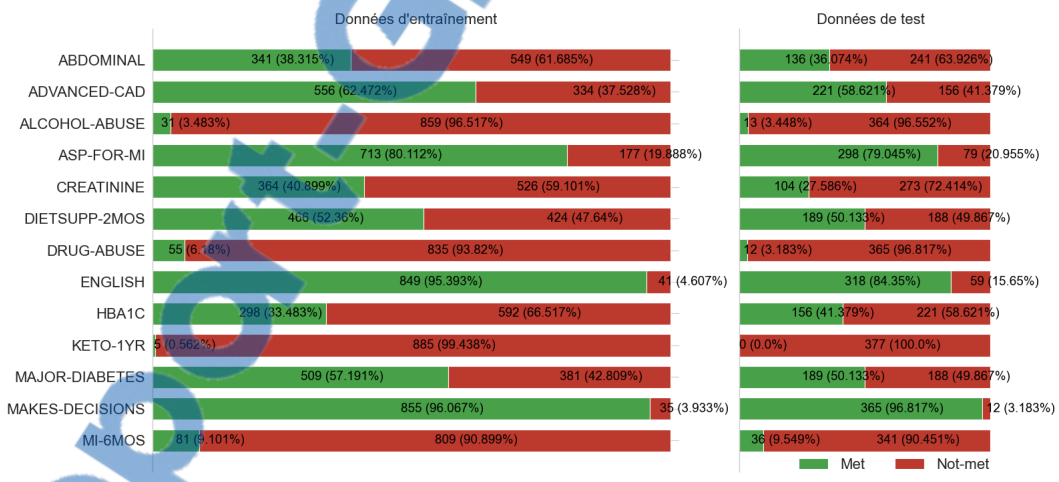

Figure 3.1: Distribution des enregistrements positifs et négatifs pour chaque critère, séparés en données d'entraînement et de test.

Une bonne distribution des données est donc essentiel. Malheureusement, comme il est possible de le voir dans la figure 3.1, la distribution du jeu de données N2C2 shared task (track 1) est très mauvaise. Sept critères (ALCOHOL-ABUSE, ASP-FOR-MI, DRUG-ABUSE, ENGLISH, KETO-1YR, MAKE-DECISIONS, MI-6MOS) sont inégalement distribués pour un total de 13 critères. Pour cinq de ces critères (ALCOHOL-ABUSE, DRUG-ABUSE, ENGLISH, MAKE-DECISIONS, MI-6MOS), la distribution est tellement inégale que la classe minoritaire comporte moins de 10% des instances.

Il est également intéressant de noter que les 2 critères qui ne font pas partie du domaine médical (ENGLISH, MAKE-DECISIONS) sont très inégalement distribués et, au vu de leur "simplicité", mériteraient d'être traités de manière séparée par le chercheur.

Néanmoins, bien que la distribution générale soit inégale, elle est très semblable entre les données d'entraînement et les données de test, comme il est possible de le constater dans la figure 3.1. Seul le critère KETO-1YR est spécial avec 100% d'instances négatives dans les données d'entraînement. De part à cette similarité, les résultats des modèles, même s'ils prédisent la classe majoritaire pour les critères inégalement distribués, ne seront pas mauvais.

Le second élément qui peut entrer en compte dans la classification est la longueur des enregistrements. Tout comme pour la distribution des critères, les résultats peuvent être différents pour des critères définis par des longueurs de textes différents.

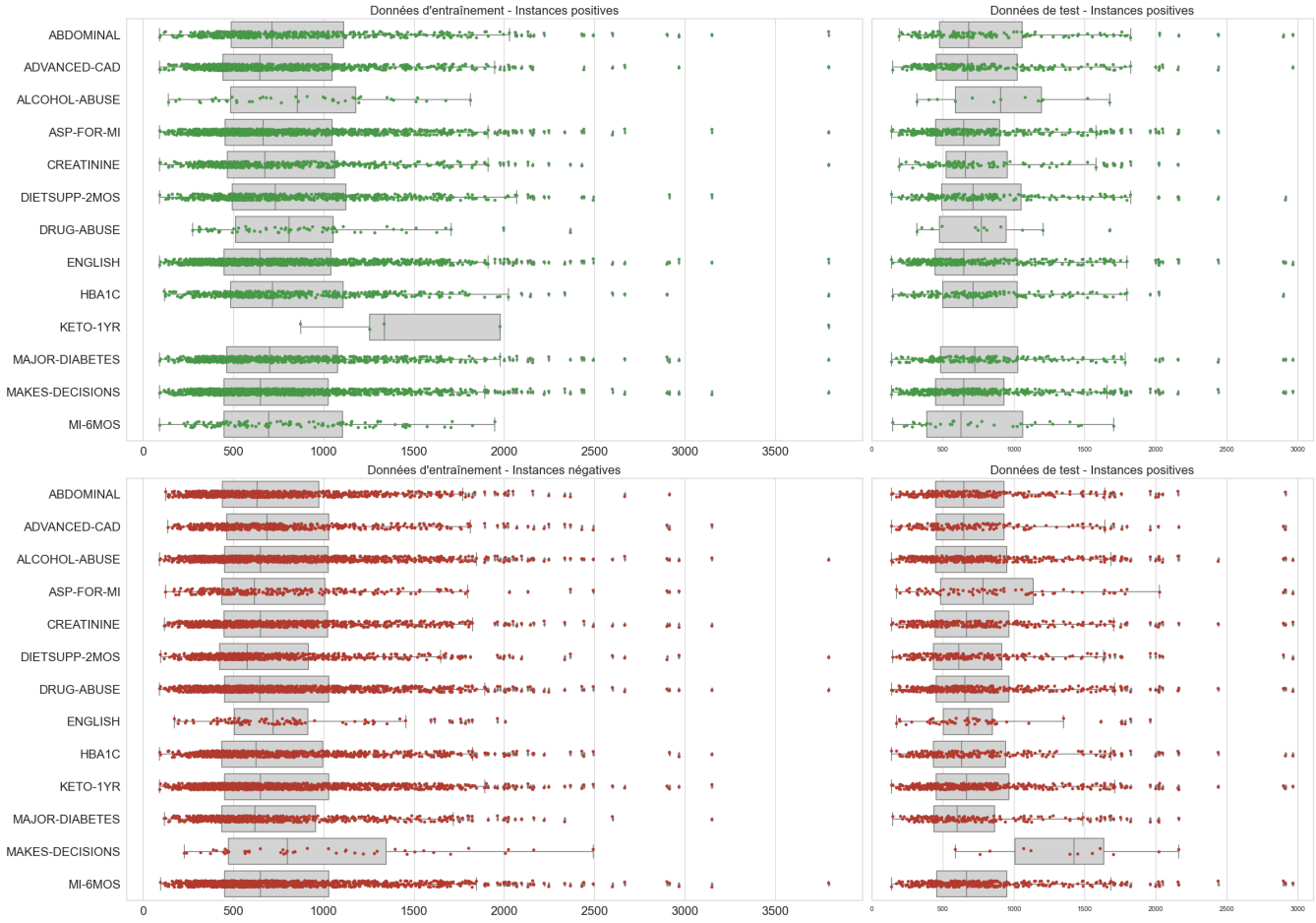

Figure 3.2: Distribution des longueurs des enregistrements positifs et négatifs pour chaque critère, séparés en données d'entraînement et de test.

Comme il est possible de le voir dans la figure 3.2, la longueur des enregistrements entre les différents critères est relativement égale. Les enregistrements ont une médiane de 684 tokens. La différence la

plus notable est le critère KETO-1YR ne disposant que de 5 enregistrements positifs d'entraînement et dont la longueur médiane est de 1335 tokens. Tout comme pour la distributions des critères, il est intéressant de noter que la distribution des longueurs des enregistrements est très équitable entre les données d'entraînement et les données de test et qu'elle ne devrait alors pas influencer les résultats.

Les données réelles, avec lesquels les modèles sont utilisés lorsqu'ils sont en production, sont en général également inégales. Il est alors nécessaire d'améliorer les modèles d'apprentissage machine pour prendre en compte cette contrainte. L'utilisation d'un jeu de données déséquilibré ne pose alors pas spécifiquement de problème pour autant qu'il y ait suffisamment d'instances d'entraînement. Le gros problème réside donc dans la collecte de ces données, surtout dans le domaine médicale.

3.3 Corrélation des critères

Le jeu de données 2018 N2C2 shared task (track 1) [26] provenant du domaine de la médecine, il est possible qu'un critère en induise un deuxième, comme une grippe est souvent accompagnée de fièvre et de toux. Il est alors important de vérifier que les différents critères du jeu de données ne sont pas corrélés. Afin d'appuyer l'hypothèse faite dans la section 3.2, il est également possible de calculer la corrélation entre la longueur des textes et les différents critères.

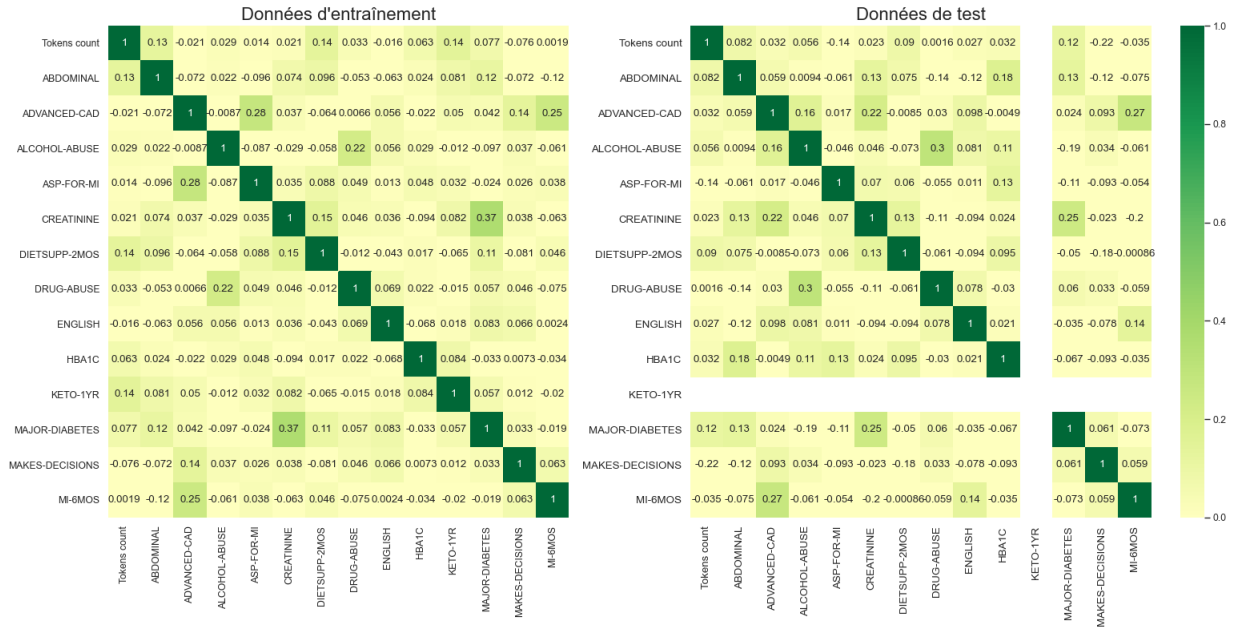

Figure 3.3: Corrélation des critères entre eux et avec la longueur des textes pour les données d'entraînement et de test.

Comme l'affiche le corrélogramme 3.3, il n'y a pas de corrélations fortes entre les différents critères. La corrélation la plus élevée est celle entre le critère MAJOR-DIABETES et CREATININE, qui semble normal, car une créatinine élevée s'observe chez les personnes souffrant d'un diabète. De même, il

est normal que le critère ASP-FOR-MI, qui est défini comme étant la prise d'aspirine pour prévenir des problèmes cardiaques, et le critère MI-6MOS, qui est défini comme un infection cardiaque, soient corrélés avec le critère ADVANCED-CAD, qui désigne des problèmes cardio-vasculaires. La combinaison DRUG-ABUSE et ALCOOL-ABUSE est également plus fortement corrélée, sans grande surprise. La longueur des enregistrements est également très peu corrélée aux critères, ce qui appuie l'hypothèse qu'elle n'influencera pas les résultats des prédictions.

4 Méthodologie

Afin de prédire les classes associées aux données, différents algorithmes / modèles existent. Chacun des modèles présentés ci-dessous peuvent être utilisés dans un problème de classification multi-label.

4.1 Modèles naïfs - baseline

Afin d'avoir une base sur laquelle comparer l'ensemble des modèles, il est essentiel d'utiliser préalablement des classificateurs naïfs. Ces derniers donnent des résultats calculés sur des statistiques de distribution des données et non sur un entrainement de ces dernières. Trois classificateurs naïfs sont intéressants dans une tâche de classification:

1. Random Guess Classifier: Assigne de manière aléatoire la première classe à 50% des instances et la seconde classe au reste des instances;
2. Majority Guess Classifier: Assigne la classe majoritaire à toutes les instances;
3. Weighted Guess Classifier: Assigne de manière aléatoire la première classe à x% (pourcentage d'instances classifiées positives) des instances et la seconde classe au reste.

4.2 Modèles classiques

4.2.1 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) [27] est l'un des algorithmes les plus simples et les plus utilisés dans des tâches de classification. C'est un algorithme non paramétré qui se contente de stocker l'intégralité des instances fournies lors de l'entraînement. Afin de prédire une nouvelle instance, il calcule la distance entre celle-ci et l'ensemble des instances stockées afin de définir la prédiction comme étant la valeur de l'instance la plus proche. La distance utilisée est, le plus souvent, la distance euclidienne, définie par l'équation suivante :

$$d(a,b) = d(a,b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + ... + (a_n - b_n)^2}, \tag{4.1}$$

$$d(a,b) = d(a,b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}, \tag{4.2}$$

où a est le vecteur à prédire et b un des vecteurs d'entraînement.

Dans l'exemple de la figure 4.1, l'instance la plus proche de celle à prédire (en noir) est l'instance entourée en rouge. La prédiction sera alors "violet".

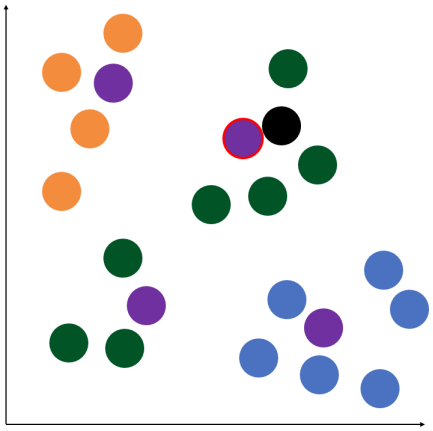

Figure 4.1: Exemple de classification KNN. En noir, l'instance à prédire et entouré en rouge l'instance la plus proche utilisée pour la prédiction.

Néanmoins, en analysant l'ensemble des données, l'instance aurait dû être prédite "vert" car entourée par plus d'instances vertes. L'algorithme KNN est très sensible aux données "parasites" et aura facilement tendance à incorrectement classifier une instance si ces derniers sont trop nombreux. Afin de palier à ce problème, il est possible de définir le nombre d'instances voisines prisent en compte lors du choix de la prédiction (paramètre k). En augmentant cette valeur, l'algorithme est obligé de prédire la classe majoritaire parmi l'ensemble des k instances les plus proches. Si deux classes ont le même nombre d'instances, la classe de l'instance la plus proche est sélectionnée.

En définissant k à 5, l'algorithme ne prédit plus la classe "violet" mais la classe "vert" (figure 4.2).

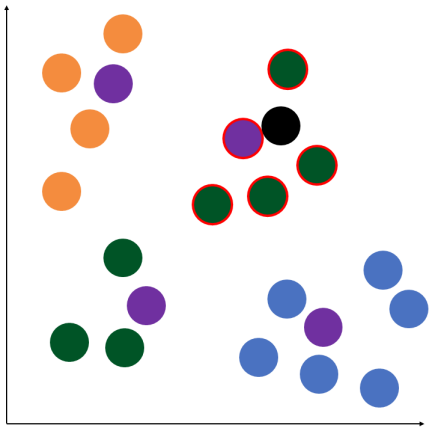

Figure 4.2: Exemple de classification KNN, $k = 5$. En noir, l'instance à prédire et entourées en rouge les instances sélectionnées pour la prédiction.

L'avantage important d'un algorithme non paramétré comme KNN est qu'il est possible de le rendre plus précis en augmentant le nombre d'instances d'entraînement au cours du temps.

4.2.2 Radius Neighbors

L'algorithme Radius Neighbors fonctionne de manière très semblable à l'algorithme KNN. Tout comme ce dernier, c'est un algorithme non paramétré qui se contente de stocker les instances d'entraînement. Il va également mesurer la distance euclidienne entre l'instance à prédire et les instances "d'entraînement" pour sélectionner les instances utilisées pour la prédiction. La prédiction sera également définie par la classe majoritaire de l'ensemble des instances sélectionnées.

La manière dont il sélectionne le nombre d'instances utilisées dans la prédiction fonctionne quant à elle de manière différente. Toutes les instances se trouvant dans un rayon centré sur l'instance à prédire et défini par le paramètre r seront utilisées. Dès lors, il est impossible de connaître le nombre d'instances d'entraînement utilisées pour la prédiction. L'équation suivante doit être vérifiée afin que l'instance soit prise en compte :

$$r > d(a, b), \tag{4.3}$$

$$r > \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}. \tag{4.4}$$

Dans l'exemple 4.3, seulement 4 instances seront utilisées pour la prédiction. La valeur de cette dernière sera "vert".

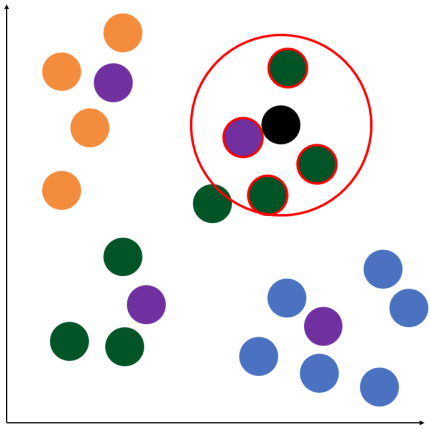

Figure 4.3: Exemple de classification Radius Neighbors. En noir, l'instance à prédire et entourées en rouge les instances sélectionnées pour la prédiction.

4.2.3 Arbre de décision

Basé sur des conditions simples enchaînées, l'arbre de décision (*Decision Tree*) [28] permet de résoudre des problèmes de classification multi-classes. Ce modèle fonctionne avec un système de noeuds conditionnels, mis les un après les autres sous la forme d'un arbre. Afin de définir chaque condition, le modèle doit analyser les données et trouver la condition permettant de les séparer en deux groupes. Plus les classes sont distinctes entre les deux groupes, meilleure est la condition. Les données sont ensuite séparées en suivant la condition trouvée et le processus se répète.

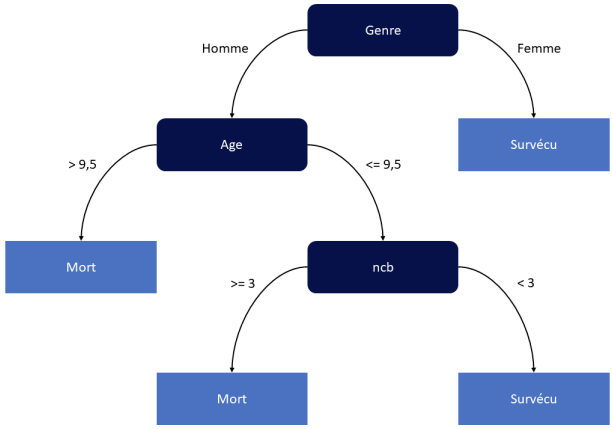

Figure 4.4: Exemple d'un arbre de décision représentant les survivants du Titanic (ncb représente le nombre de conjoints, frère ou soeur à bord) [29].

Afin d'évaluer la qualité d'une condition, il est possible de mesurer l'*information gain* ou la *gini impurty*. Cette dernière calcule la probabilité de classification incorrecte d'une instance. Elle est définie par la fonction suivante :

$$G(k) = \sum_{i=1}^{J} P(i) \cdot (1 - P(i)), \tag{4.5}$$

où $P(i)$ est la probabilité que l'instance soit classifiée dans une classe particulière.

Afin d'illustrer ceci, imaginons le jeu de données suivant :

Instance	Fatigué	Travail en retard	Grasse matinée ?
1	Oui	Oui	Non
2	Oui	Non	Oui
3	Oui	Oui	Non
4	Non	Oui	Non
5	Non	Non	Oui

Table 4.1: Jeu de données d'exemple - Arbre de décision

En utilisant la *gini impurty*, il est possible de calculer la probabilité qu'une nouvelle instance soit

incorrectement classifiée en se basant sur les données du tableau 4.1 :

$$G(\text{Grasse matinée ?}) = P(oui) \cdot (1 - P(oui)) + P(non) \cdot (1 - P(non)), \tag{4.6}$$

$$G(\text{Grasse matinée ?}) = \frac{2}{5} \cdot (1 - \frac{2}{5}) + \frac{3}{5} \cdot (1 - \frac{3}{5}), \tag{4.7}$$

$$G(\text{Grasse matinée ?}) = \frac{12}{25} = 0.48. \tag{4.8}$$

Dans 48% des cas, la nouvelle instance sera incorrectement classifiée.

Différents paramètres permettent de définir la taille de l'arbre, le nombre de conditions, le nombre de données minimum nécessaires dans chaque groupe distinct pour créer une condition, etc.

4.2.4 Random Forest

Très semblable à l'arbre de décision, le modèle Random Forest [30], comme son nom l'indique, se sert d'un ensemble d'arbres de décision, paramétrés différemment. Afin de définir la prédiction, il prend la classe majoritaire parmi les prédictions de l'ensemble des arbres de décision le formant. Le concept fondamental derrière ce modèle est appelé "la sagesse des foules". En effet, un grand nombre d'éléments individuels indépendants travaillant ensemble pourra toujours battre n'importe lequel des éléments individuels le constituant.

En d'autres termes, en créant plusieurs arbres de décision indépendants et en prédisant la classe la plus présente parmi eux, la précision sera statistiquement plus élevée qu'en en utilisant qu'un seul, quel que soit son paramétrage.

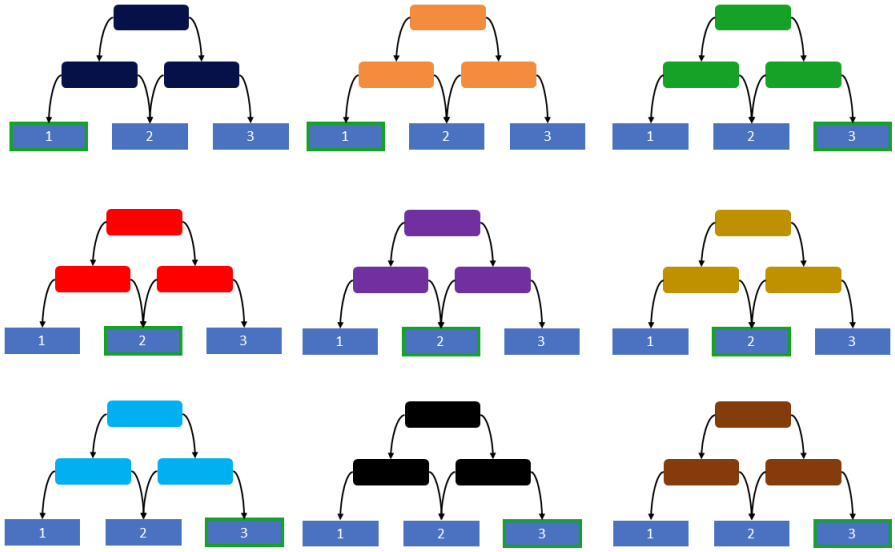

Figure 4.5: Exemple de Random Forest dont la prédiction finale est "trois"

4.2.5 Régression linéaire

La régression linéaire [31] est un modèle paramétré cherchant à définir une relation entre une variable x (variable explicative) et une variable y (variable expliquée).

Le modèle de régression linéaire le plus simple, dit affine, cherche à découvrir la droite expliquée par la fonction affine suivante :

$$f(x) = \alpha x + \beta, \tag{4.9}$$

$$\hat{y_i} = f(x_i). \tag{4.10}$$

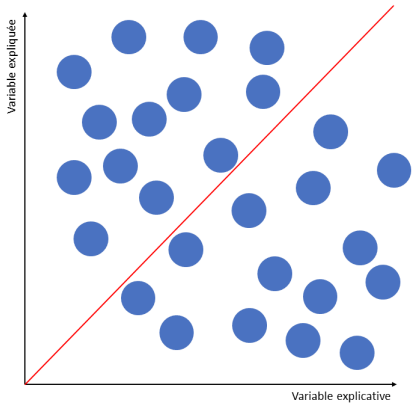

Figure 4.6: Régression linéaire affine. Les données sont représentées par des ronds bleus et la régression linéaire par une droite rouge

Une régression linéaire affine limite néanmoins les prédictions à une droite et donc à une seule variable explicative. Si la variable expliquée est décrite par une multitude de variables explicatives, le modèle devient un modèle linéaire multiple :

$$f(x) = \alpha_1 x_1 + \alpha_2 x_2 + ... + \alpha_i x_i + \beta, \tag{4.11}$$

$$f(x) = \sum_{i=0}^{n} (\alpha_i x_i) + \beta, \tag{4.12}$$

$$\hat{y_i} = f(x_i). \tag{4.13}$$

Il est possible de simplifier l'équation en la posant dans une notation vectorielle :

$$f(x) = x^T w, \tag{4.14}$$

$$\hat{y_i} = f(x_i), \tag{4.15}$$

où w représente l'ensemble des poids du modèle ($\alpha_1 ... \alpha_i$ et β) et où il est nécessaire d'ajouter un 1

au début du vecteur x pour intégrer β dans w ($\underline{1} \cdot \beta$).

Afin d'estimer les meilleurs paramètres (w), il est nécessaire de minimiser les erreurs entre la droite et les données. Une des méthodes d'estimation possible des erreurs est la méthode des moindres carrés (MSE) qui consiste à calculer la distance entre la valeur réelle (y) et la valeur prédite de cette dernière ($\hat{y}$) :

$$e(y, \hat{y}) = (y - \hat{y})^2, \tag{4.16}$$

$$e(y, \hat{y}) = (y - x^T w)^2. \tag{4.17}$$

Afin de réduire l'erreur, il convient de mettre à jour les poids utilisés dans le modèle linéaire en se basant sur les erreurs lors des prédictions avec les poids actuels. Pour ce faire, il faut d'abord calculer l'erreur moyenne du modèle (moyenne des moindres carrés) :

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{4.18}$$

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T w)^2. \tag{4.19}$$

Il est ensuite possible d'utiliser la méthode de descente stochastique de gradient pour mettre à jour les poids :

$$w_{t+1} = w_t + lr \cdot (-\nabla L(w_t)) \tag{4.20}$$

où w_{t+1} sont les nouveaux poids, w_t les poids actuels et lr le learning rate, permettant de donner plus ou moins d'importance au gradient par rapport aux poids actuels.

4.2.6 Régression logistique

La régression logistique [32] fonctionne de la même manière que la régression linéaire à la seule différence que la prédiction se trouve dans un espace limité entre 0 et 1 ou -1 et 1, comme le montre la figure 4.7.

Afin de restreindre l'espace de la prédiction, il est nécessaire d'utiliser une fonction d'activation sur la valeur fournie par la régression linéaire. Les fonctions d'activation les plus utilisées sont la fonction sigmoid (σ), qui réduit l'espace entre 0 et 1, et la tangeante hyperbolique (tanh), qui réduit l'espace entre -1 et 1.

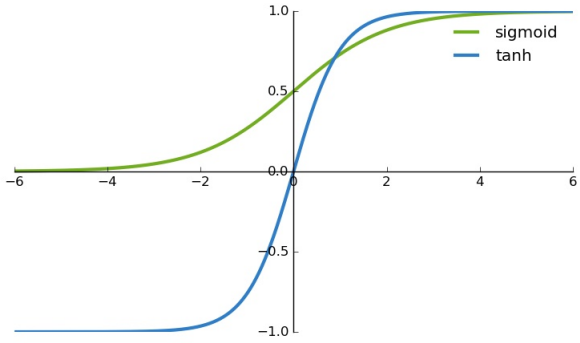

Figure 4.7: Différence entre sigmoid et tanh [33]

Dans notre cas, les valeurs à prédire sont binaires (0 ou 1) et suivent une distribution de Bernoulli:

$$P(X = x) = \begin{cases} p & \text{si } x = 1 \\ 1 - p & \text{si } x = 0 \end{cases} \tag{4.21}$$

La meilleure fonction d'activation pour le jeu de données N2C2 shared task est donc la fonction sigmoid, qui suit elle-même la distribution de Bernoulli et qui est définie par l'équation suivante:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{4.22}$$

$$\sigma(x) = \frac{e^x}{e^x + 1}. \tag{4.23}$$

Afin de prendre en compte cette ajout dans le calcul de la modifications des poids, il est nécessaire de modifier la fonction d'estimation des erreurs :

$$e(y, \hat{y}) = -y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}) \tag{4.24}$$

et donc de mettre à jour la fonction de calcul de l'erreur moyenne du modèle :

$$L(w) = -\frac{1}{n} \sum_{i=1}^{n} (y_i \cdot \log(\hat{y_i}) + (1 - y_i) \cdot \log(1 - \hat{y_i})), \tag{4.25}$$

$$L(w) = -\frac{1}{n} \sum_{i=1}^{n} (y_i \cdot \log(x_i^T w) + (1 - y_i) \cdot \log(1 - x_i^T w)). \tag{4.26}$$

4.3 Modèles de Deep Learning

4.3.1 Perceptron multicouches

Un perceptron multicouches (MLP) [34] est le plus basique des modèles de réseaux neuronaux. Il est constitué d'une suite de perceptrons, connectés les uns aux autres. Un perceptron est une unité du réseau de neurones chargé de détecter une valeur particulière, avec une architecture similaire à la régression logistique. Un vecteur x est fourni au perceptron qui va le multiplier à un vecteur de poids w. Par la suite, tout comme dans la régression logistique, une somme est appliquée sur l'ensemble du vecteur x puis une fonction d'activation permet de supprimer la linéarité, comme cela est visible dans la figure 4.8.

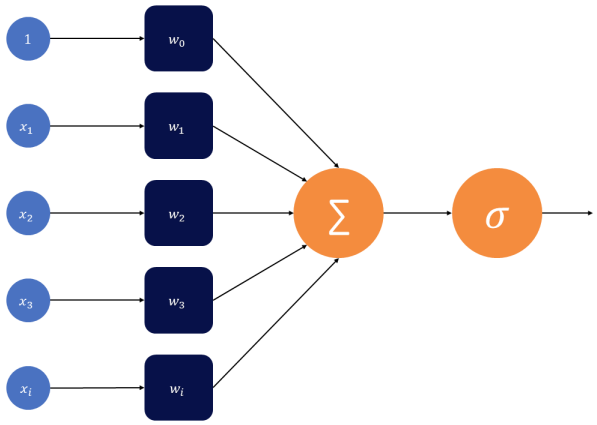

Figure 4.8: Perceptron. La fonction d'activation utilisée ici est sigmoid (σ).

Un MLP sera constitué de plusieurs perceptrons, appelés "couche" dans ce cas là. La première couche, d'entrée, aura automatiquement la même taille que le vecteur x tandis que la dernière couche, de sortie, aura la même taille que le nombre de classes à prédire. Les couches cachées intermédiaires peuvent avoir n'importe quelle taille. Afin de prendre en compte le fait qu'une couche t n'a pas forcement la même taille que la couche $t+1$, les poids ne sont plus stockés sous forme de vecteur mais sous forme de matrice (m, n), où m est la taille de la couche précédente (ou de x pour la première couche) et n la taille de la couche suivante (ou le nombre de classes à prédire pour la dernière couche). Chaque élément $(w_{m,n})$ calculé dans la couche t est passé à l'ensemble des éléments de la couche $t+1$ (voir figure 4.9).

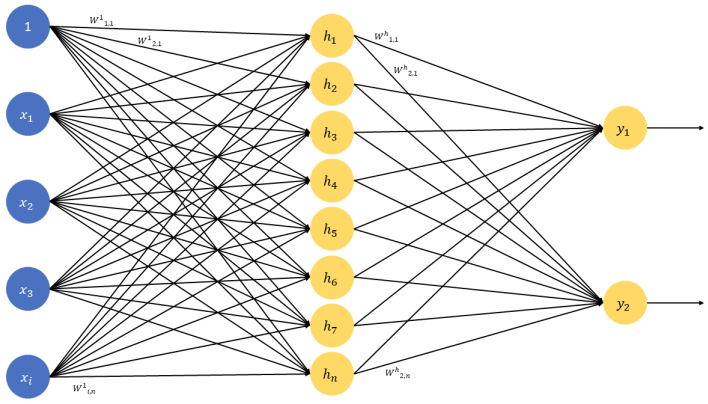

Figure 4.9: Perceptron multicouches

Lors du calcul de l'erreur et la mise à jour des poids, seules les instances incorrectement classifiées sont utilisées :

$$L(w) = \frac{1}{m} \sum_{i \in M} (y_i - \hat{y}_i)^2, \tag{4.27}$$

$$L(w) = \frac{1}{m} \sum_{i \in M} (y_i - (x_i^T w))^2, \tag{4.28}$$

où M est l'ensemble des indices des instances incorrectement classifiées.

Entre chaque couche, comme le perceptron le définit, une fonction d'activation est présente. La plus utilisée est la fonction ReLU, définie par l'équation suivante :

$$ReLU(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x <= 0 \end{cases} \tag{4.29}$$

Si le MLP est utilisé à des fins de classification binaire, il convient de mettre la fonction d'activation sigmoid dans la dernière couche à la place de ReLU.

Afin de mettre à jour les poids, la descente stochastique de gradient est toujours utilisée.

4.3.2 CNN

Les modèles de convolution (CNN) [35] sont un nouveau type de réseaux de neurones inventés dans le but de traiter des images mais également utilisables dans des tâches de classification. Le traitement d'images est bien différent au vu de son format. Une image est composée de milliers de pixels pouvant avoir des valeurs diverses selon la couleur à afficher. Ces valeurs là peuvent être récupérées et utilisées, au format matriciel, afin de représenter l'image.

Basé sur cette représentation matricielle, un modèle de convolution peut en retirer de l'information

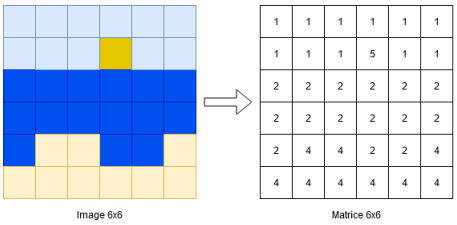

Figure 4.10: Exemple de transformation image à matrice

et la classifier. Afin de pouvoir récupérer seulement les informations voulues de l'image, un CNN fonctionne grâce à 3 étapes successives suivies d'un modèle de classification (régression logistique par exemple) :

1. Couche de convolution: applique un filtre (les poids) sur l'image pour récupérer seulement les informations voulues;
2. Fonction d'activation: supprime la linéarité dans le modèle;
3. Opération de pooling: réduit la taille de l'image.

Couche de convolution

Comme l'image est représentée par une matrice, le poids, appelé filtre dans le cas d'une convolution doit également être sous la forme d'une matrice. La couche de convolution va utiliser un filtre W de taille (m, n) sur la matrice d'entrée X. En commençant en haut à gauche de X, le filtre va se déplacer sur l'image et appliquer à chaque étape une multiplication $W \bullet X_{i-i+m,j-j+n}$.

Dans la figure 4.11, la première étape de convolution se calcule alors de la manière suivante:

$$W \bullet X_{0-3,0-3} = 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 2 + 1 \cdot 2 + 1 \cdot 2 \tag{4.30}$$

$$W \bullet X_{0-3,0-3} = 7 \tag{4.31}$$

Il est possible de voir ci-dessus que plus la partie de l'image analysée correspond au filtre utilisé, plus le résultat sera élevé.

La taille du filtre et la manière dont il se déplace (un par un, deux par deux, etc) sont définies par des paramètres. Il est possible de calculer la taille de la matrice de sortie, basée sur ces paramètres :

$$size = (w - f)/s + 1 \tag{4.32}$$

où w est la taille de la matrice d'entrée X, f la taille du filtre W et s le stride, qui contrôle le déplacement du filtre sur la matrice d'entrée.

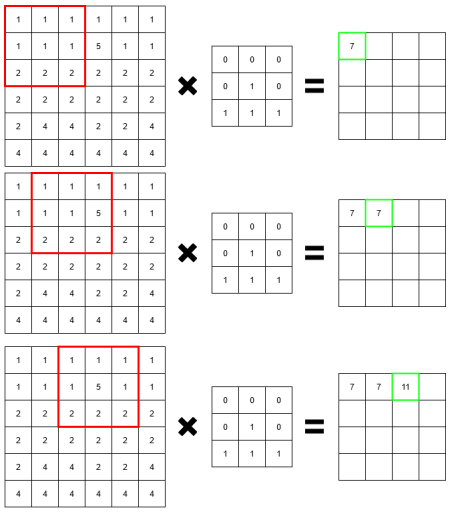

Figure 4.11: Exemple de convolution (w = 6, f = 3, s = 1)

Le passage complet du filtre sur la matrice X est décrit par l'équation suivante :

$$Z_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} X_{i+k,j+l} \cdot W_{k,l}, \tag{4.33}$$

où Z est la matrice de sortie complète de la convolution. Les indices des lignes et des colonnes de Z sont dénoté par i et j tandis que les indices des lignes et des colonnes du filtre W sont dénotés par k et l.

Dans la plupart des cas, une image en couleur dispose d'une matrice de dimension V x H par couleur. La matrice X a donc une taille de V x H x D où V est le nombre de pixels verticaux, H le nombre de pixels horizontaux et D le nombre de couleurs. Par conséquent, il est intéressant de modifier le format du filtre afin de le mettre à 3 dimensions également (M x N x D). De cette manière, chaque couche du filtre se chargera d'une couche de couleur.

En partant de ce principe, il est possible de s'imaginer un filtre contenant plusieurs centaines de couches servant chacune à capturer un élément particulier de l'image.

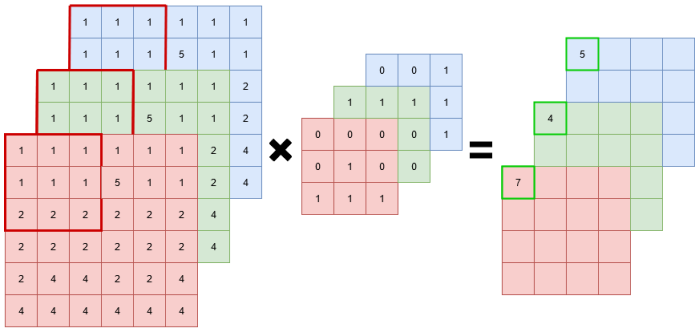

Figure 4.12: Exemple de convolution multi-channels

Bien évidemment, les filtres ne sont pas définis à la main par des chercheurs. Tout comme avec la régression linéaire, le perceptron ou le MLP, un système de calcul des erreurs et de mise à jour des filtres permettent de trouver la meilleure manière de classifier une image.

Opération de pooling

Après une ou plusieurs couches de convolution, et selon la taille de la matrice d'entrée et les paramétrages de la convolution, il est probable que la matrice Z ait une grande taille. Tout comme un MLP, un CNN cherche à réduire petit à petit la taille de la matrice de sortie afin de garder seulement les informations nécessaires. Un MLP le fait en réduisant petit à petit la taille des perceptrons, un CNN le fait en utilisant une couche intermédiaire de pooling. Le but d'une opération de pooling est de réduire la taille de la matrice en gardant uniquement les informations importantes. Il existe différentes méthodes de pooling mais la plus utilisée est le max-pooling. Comme vu précédemment, plus la partie de la matrice analysée correspond au filtre, plus le résultat de la multiplication sera élevé. Une opération de max-pooling va réduire la taille de cette dernière en ne gardant que les nombres les plus élevés et donc les informations les plus pertinentes. Tout comme avec la couche de convolution, il est possible de paramétrer la taille du pooling et la manière dont il se déplace sur la matrice.

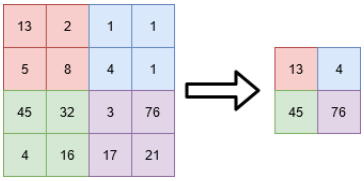

Figure 4.13: Exemple de max-pooling (taille de pooling = 2, stride = 2)

Classification

Lorsque toutes les informations pertinentes de la matrice ont été extraites, vient l'étape de classification. Pour ce faire, il est nécessaire d'applatir la matrice pour la transformer en vecteur. Ensuite, n'importe quel modèle de classification peut être utilisé, mais le plus souvent, un MLP ou un perceptron unique est utilisé. L'architecture finale d'un CNN ressemble à ceci :

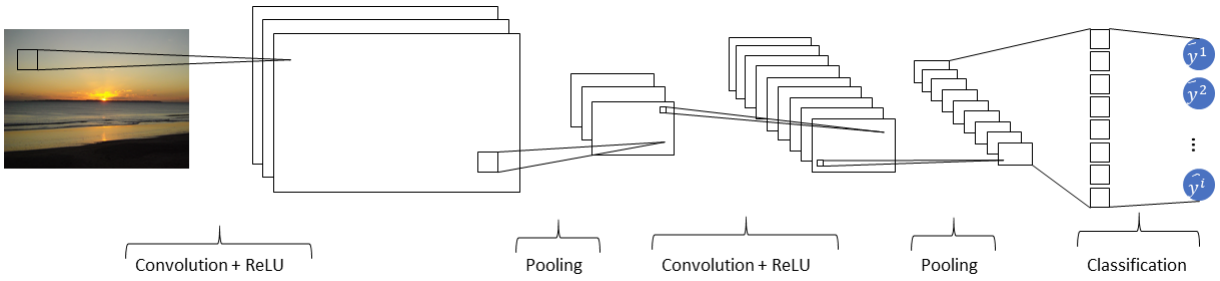

Figure 4.14: Exemple d'architecture complète CNN

Convolution à 1 dimension

Le modèle de convolution de base se pratique à 2 ou 3 dimensions, dans le cas où il y a plusieurs channel, mais il est également possible d'effectuer de la convolution à 1 dimension. En utilisant un filtre de taille 1 x M, la convolution s'aplatit et peut s'appliquer sur des données à 1 dimension tout en utilisant les mêmes calculs. Les convolutions à 1 dimension sont souvent utilisées pour des tâches d'analyse de texte et de séries temporels.

4.3.3 RNN

Un modèle récurrent (RNN) est une forme particulière de réseau de neurones imaginée pour traiter des séquences (phrase, texte, vitesse de déplacement d'un avion sur une plage horaire donnée, etc). Dans une séquence, le contexte et l'ordre des éléments a une place importante dans la compréhension de cette dernière. En utilisant des réseaux de neurones habituels, ceci n'est pas pris en compte, chaque donnée est indépendante. Le but d'un RNN est de se souvenir de l'ordre et du contexte de la séquence afin de prédire un élément qui suive logiquement cette dernière.

Pour garder une trace de la séquence, un RNN a une architecture spéciale en forme de boucle. Une boucle est appelée une cellule RNN et un modèle complet de RNN en empile plusieurs les unes sur les autres.

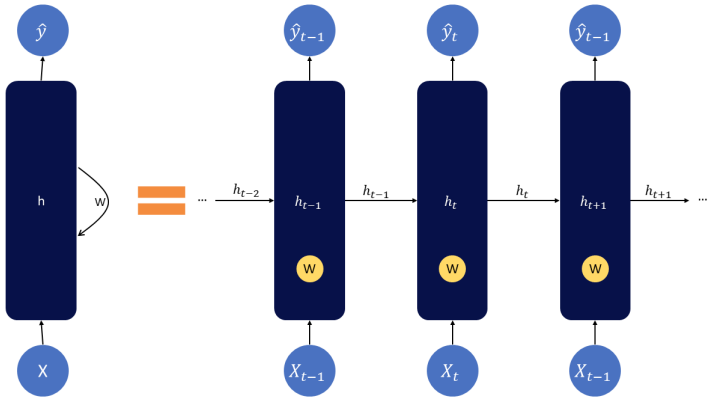

Figure 4.15: RNN "déroulé". Les poids (W) sont partagés entre l'ensemble des cellules.

Comme le montre la figure 4.15, un RNN complet est une suite de cellule RNN se partageant la même matrice de poids, permettant d'influencer les données. En plus de celle-ci, chaque cellule dispose d'un état caché. Cette dernière traverse l'ensemble des cellules et est modifiée à chaque passage. Le calcul de la prédiction se base alors sur les données d'entrée, les poids et cet état caché qui représente la séquence passée. Cet état caché agit alors comme une mémoire tampon et se calcule de la manière suivante:

$$h_t = f(Ux_t + Wh_{t-1}), \tag{4.34}$$

où h_t est l'état caché actuel, U et W sont les poids, x_t les données actuelles et h_{t-1} l'état caché précédent.

La matrice de poids doit alors prendre en compte cet état caché, en prenant le format (U, V, W) :

- U, qui représente le poids entre les données d'entrées et l'état caché;
- V, qui représente le poids entre l'état caché actuel et le résultat;
- W, qui représente le poids entre l'état caché précédent et l'état caché actuel.

Il existe plusieurs architectures de RNN, selon le problème à résoudre :

- One to One: Un seul élément en entrée, un seul élément en sortie;
- One to many: Un seul élément en entrée, une séquence en sortie;
- Many to One: Une séquence en entrée, un seul élément en sortie;
- Many to many: Une séquence en entrée, une séquence en sortie. La taille de la séquence d'entrée peut être différente de la taille de la séquence de sortie.

Le calcul d'une cellule RNN many to many peut être représenté par les équations suivantes :

$$a_t = \beta + W h_{t-1} + U x_t, \tag{4.35}$$

$$h_t = f(a_t), \tag{4.36}$$

$$\hat{y_t} = c + V h_t, \tag{4.37}$$

où β est le biais sur les données d'entrée, c le biais sur les résultats, h_{t-1} l'état caché précédent, h_t l'état caché actuel, f la fonction d'activation et W, U, V les poids.
Le fonctionnement d'une cellule RNN, défini aux équations 4.35, 4.36 et 4.37, est visualisable à la figure 4.16.

L'équation de calcul des erreurs sera égale à la somme des erreurs de chaque cellule :

$$L = \sum_{t}^{L(x_1,...,x_t,y_1,...,y_t)} L_t, \tag{4.38}$$

$$L = -\sum_{t} \log P_{model}(y_t | x_1, ..., x_t), \tag{4.39}$$

où x_t est un vecteur de données à un instant t et y_t l'erreur à un instant t.

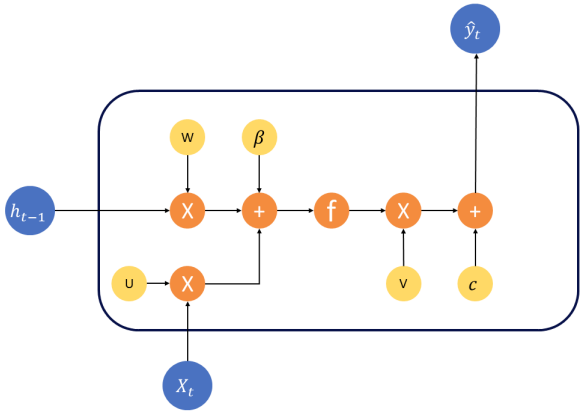

Figure 4.16: Cellule RNN. Les poids sont en jaune, les opérations et fonction d'activations en orange et les données, états cachés et prédictions en bleu.

Les architectures des RNN ont beaucoup évolué au fil des années. Celle de base (figure 4.16) a deux gros problèmes :

1. Il n'est pas possible de donner plus ou moins d'importance à l'état caché actuel par rapport à l'état caché précédent;
2. Plus le nombre de cellules augmente, plus le gradient vanishing sera important [36].

Afin de résoudre ces deux problèmes, de nouvelles architectures ont été proposées. L'architecture Gated Recurrent Units (GRU) [37] offre la possibilité d'oublier ou de donner plus d'importance à l'état caché précédent ou encore à l'état caché actuel et aux données, résolvant ainsi le problème 1. Pour ce faire, GRU va premièrement créer un état caché intermédiaire, basé sur l'état caché précédent et les données, tout comme la cellule RNN basique. La différence est que l'état caché réel de la cellule est une combinaison linéaire entre cet état caché intermédiaire et l'état caché précédent, permettant de donner plus ou moins d'importance à l'un ou l'autre. Les équations sont alors modifiées de la manière suivante:

$$r^t = \sigma(U^r X_t + W^r h_{t-1}), \tag{4.40}$$

$$\tilde{h}^t = \tanh(W h_{t-1} \circ r_t + U x_t), \tag{4.41}$$

$$z_t = \sigma(U^z x_t + W^z h_{t-1}), \tag{4.42}$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h_t}, \tag{4.43}$$

$$\hat{y_t} = c + V h_t, \tag{4.44}$$

où $\tilde{h_t}$ est l'état caché intermédiare et z_t et r_t (Update Gate et Reset Gate) les éléments nécessaires

au calcul de la combinaison linéaire sur l'état caché intermédiaire.

Afin de résoudre le problème de gradient vanishing [36], une autre architecture a été proposée : Long-Short-Term Memories (LSTM) [38]. Cette dernière reprend l'architecture GRU et ajoute un second état, l'état cellule. Tout comme l'état caché, l'état cellule traverse l'ensemble des cellules mais n'interagit que très faiblement avec celles-ci. De ce fait, les gradients de l'état cellule ne seront que peu impactés par un problème de gradient vanishing. De plus, contrairement à l'état caché qui est une combinaison linéaire et qui perd donc les informations à long terme, l'état cellule a la possibilité de stocker des informations à plus long terme, même si celles-ci ne sont plus apparentes dans l'état caché. Afin d'implémenter cette architecture, il convient de modifier les équations:

$$i_t = \sigma(U^i x_t + W^i h_{t-1}), \tag{4.45}$$

$$f_t = \sigma(U^f x_t + W^f g_{t-1}), \tag{4.46}$$

$$o_t = \sigma(U x_t + W h_{t-1}), \tag{4.47}$$

$$\tilde{c_t} = \tanh(W^c x_t + W^x h_{t-1}), \tag{4.48}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c_t}, \tag{4.49}$$

$$h_t = o_t \circ \tanh(c_t), \tag{4.50}$$

$$\hat{y_t} = c + V h_t, \tag{4.51}$$

où c_{t-1} est l'état cellule précédent, c_t l'état cellule actuel, $\tilde{c_t}$ l'état cellule intermédiaire et i_t, f_t et o_t (Input Gate, Forget Gate et Output Gate) les éléments nécessaire à la combinaison linéaire de l'état cellule.

4.4 Modèles d'embeddings

Un modèle est incapable d'utiliser du texte brut lors d'une classification. Il est nécessaire de le transformer préalablement en nombre, vecteur ou matrice. Cette opération de transformation est appelée "Embedding" [39]. Les valeurs du jeu de données 2018 N2C2 shared task (track 1) que nous voulons prédire étant uniquement du texte, l'ensemble du jeu de données doit être "embedded".

L'embedding est un algorithme ou un modèle de Machine Learning qui se charge de donner une représentation vectorielle la plus fidèle d'un mot. Chaque mot unique possède alors une représentation vectorielle unique. De plus, des mots au sens proche sont également représentés par des vecteurs similaires. Selon les modèles d'embedding utilisés, un mot peut être représenté par un vecteur d'une taille 50 à une taille de plus de 1000.

4.4.1 One-hot encoding embedding

Le modèles d'embedding le plus simple est le one-hot encoding embedding. Son fonctionnement consiste à compter le nombre de mots uniques dans le texte puis à créer un vecteur de taille identique

et à y assigner une colonne à chaque mot. Le vecteur sera constitué uniquement de 0, sauf à la colonne correspondante au mot, qui aura la valeur 1.

Mot	One-hot embedding						
HOMME	1	0	0	0	0	0	0
FEMME	0	1	0	0	0	0	0
GARCON	0	0	1	0	0	0	0
FILLE	0	0	0	1	0	0	0
SOEUR	0	0	0	0	1	0	0
FRERE	0	0	0	0	0	1	0
ADOLESCENT	0	0	0	0	0	0	1

Table 4.2: Exemple de one-hot embedding

Il y a malheureusement plusieurs problèmes avec ce type d'embedding :

- La plus grande partie des vecteurs est composée de 0 et ne donne donc aucune information;
- Comme la taille du vecteur est définie par le nombre de mots de notre dictionnaire, il augmente de manière linéaire avec la taille de ce dernier. Si le dictionnaire contient 30'000 mots, il faudra un vecteur de taille 30'000 pour représenter chaque mot, ce qui est bien trop important pour l'ordinateur;
- Chaque mot est représenté de manière indépendante. Le sens d'une phrase n'est alors pas pris en compte (le même mot, même avec deux sens différents dans une phrase sera représenté de manière identique);
- Il n'est pas possible d'analyser la matrice pour capturer les similarités de sens entre les mots (la distance euclidienne est la même entre tous les mots).

4.4.2 Réduire la dimensionalité de l'embedding

Comme décrit dans la section précédente, l'utilisation d'un one-hot embedding ne permet pas de transformer des mots en vecteurs de manière performante. Une meilleure manière de faire serait de représenter chaque colonne, non pas comme un mot de notre dictionnaire, mais comme une idée, un concept.

Mot	Fénimité	Jeunesse	Famille
HOMME	0	0	0
FEMME	1	0	0
GARCON	0	1	0
FILLE	1	1	0
SOEUR	1	1	1
FRERE	0	1	1
ADOLESCENT	0.5	0.5	0

Table 4.3: Exemple de réduction de la taille de l'embedding

L'utilisation de ce type d'embedding offre de nombreux avantages :

- Le vecteur contient beaucoup moins de 0. Il y a donc moins d'informations inutiles;
- La possibilité d'ajouter un nombre infini de mots dans le dictionnaire sans augmenter la taille de la représentation vectorielle;
- Les mots ayant une signification similaire auront une représentation vectorielle similaire. Il est possible de les comparer en utilisant la distance euclidienne ou la similarité cosine;
- La similarité entre deux mots est facilement interprétable. Par exemple, la valeur nécessaire pour passer de GARCON à FILLE est la même que pour passer de FRERE à SOEUR ([+1, +0, +0]).

Afin d'avoir une démonstration simple, seuls trois "concepts" ont été pris en compte, résultant en une représentation vectorielle de taille 1x3. Afin de pouvoir élargir le dictionnaire au plus de mots possibles, sans perdre trop d'informations, il est nécessaire d'augmenter cette dimensionnalité. Un modèle complet utilise des propriétés / concepts qui ne sont pas forcément compréhensibles pour un être humain. Le modèle apprend par lui même à "classifier" les mots, avec ses propres concepts. Actuellement, les modèles d'embedding les plus performants permettent de créer un vecteur d'une taille supérieure à 5000 pour représenter chaque mot.

De ce fait, des mots ayant une signification similaire, et donc également une représentation vectorielle similaire, vont se retrouver à des positions proches dans une visualisation à D espaces. Le plus souvent, la similarité est exprimée au travers de la distance euclidienne ou de la similitude de cosinus.

Les modèles d'embeddings les plus performants, dit contextuels, peuvent également créer des représentations vectorielles différentes pour un même mot, selon son contexte dans la phrase. En analysant les n mots avant et après le mot à transformer, il est possible d'intégrer son contexte dans l'embeddings.

De ce fait, "souris" aura une représentation différente si elle se réfère soit à l'ordinateur ou à l'animal, ce qui n'est pas possible avec un modèle non contextualisé.

Il est possible de visualiser une représentation vectorielle en la projetant sur une surface en 2 dimensions :

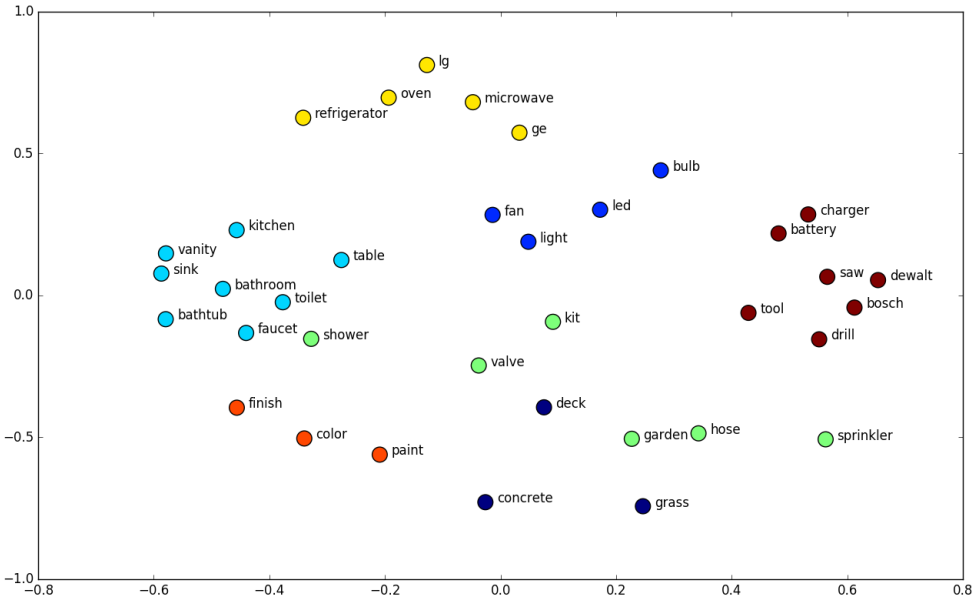

Figure 4.17: Représentation 2D d'un word embeddings [40]

Comme cela est visible sur la figure 4.17, les mots ayant une signification proche, représentés par des couleurs identiques, sont également situés à des positions rapprochées dans le graphique. Il est également possible de visualiser une représentation vectorielle en 3 dimensions sur le site https://projector.tensorflow.org/.

4.4.3 Embedder un document

Obtenir une représentation vectorielle d'un mot est intéressant, mais il est d'autant plus intéressant d'obtenir une représentation vectorielle d'un document entier. Il est difficile de concaténer les embeddings des mots les uns derrière les autres. En effet, le vecteur de sortie aurait une taille considérable et changeante, selon la taille du texte. Afin d'obtenir une taille d'embeddings fixe tout en ne perdant que le mimimum d'informations, deux solutions existent :

1. Utiliser une opération de pooling;
2. Utiliser un modèle récurrent (RNN, GRU, LSTM).

Opération de pooling

L'utilisation de l'opération de pooling, très semblable à celle du CNN, consite à transformer chaque mot en vecteur de manière individuelle puis à utiliser une opération de pooling sur ces vecteurs. Il existe 3 opérations de pooling :

1. Maximum: prendre la valeur maximum parmi l'ensemble des vecteurs pour chaque indice;
2. Minimum: prendre la valeur minimum parmi l'ensemble des vecteurs pour chaque indice;
3. Mean: prendre la valeur moyenne entre l'ensemble des vecteurs pour chaque indice.

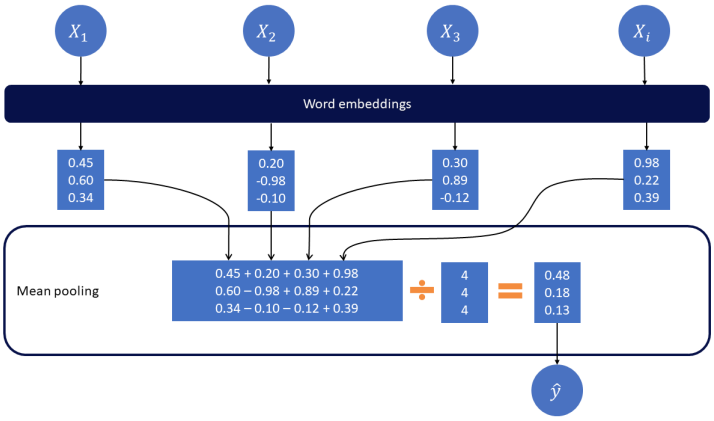

Figure 4.18: Document embeddings avec l'opération de pooling "mean"

Dans la figure 4.18, une opération de mean pooling est effectuée. L'ensemble des mots d'entrée x_i sont transformés individuellement en vecteurs. Puis, l'opération de mean pooling prend la moyenne des vecteurs pour obtenir une représentation de l'ensemble du document de même taille que celle des mots le composant.

Modèle récurrent

Très semblable à l'opération de pooling, l'utilisation d'un modèle récurrent permet d'obtenir une

représentation d'un document de même taille que celle des mots qui le compose. L'utilisation d'un RNN nécessite cependant la mise en place d'une phase d'apprentissage. Cette dernière permet d'améliorer les représentations faites par le modèle, ce qui est un avantage considérable.

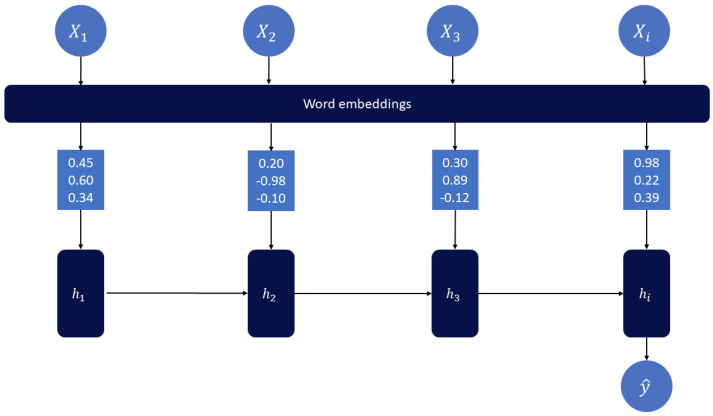

Figure 4.19: Document embedding avec un modèle récurrent

Les deux solutions fonctionnent sensiblement de la même manière. Chaque mot est transformé en vecteur individuellement. Ensuite, une opération (pooling ou modèle récurrent) est utilisée entre chaque mot pour garder une taille d'embedding constante tout en perdant le minimum d'informations.

4.4.4 Les modèles d'embeddings les plus connus

Parmi les différents modèles d'embeddings non contextualisés, les plus connus sont Word2Vec [41] et Glove [21]. Ces derniers ne prennent pas en compte l'ordre des mots et leur signification dans le texte. Pour transformer un mot en vecteur, il n'est pas nécessaire d'avoir le modèle complet mais seulement la liste des mots et leurs vecteurs associés. Cela rend la transformation d'un mot en vecteur très rapide mais limitée aux seuls mots compris dans cette liste.

Par la suite sont arrivés les modèles d'embeddings contextualisés. ELMo [42], l'un des plus connu, prend en compte la position et le contexte du mot dans la phrase pour créer la représentation vectorielle. La possibilité de contextualisé le mot engendre néanmoins un désavantage: il est indispensable d'utiliser directement le modèle pour embedder un mot, contrairement aux modèles non contextualisés, engendrant des temps de calculs plus élevés.

L'architecture la plus récente est l'architecture Transformers, introduite par Vaswani et cie [24]. Cette architecture a battu les derniers records de l'état de l'art. Elle est basée sur une couche "d'attention" qui augmente considérablement la longueur de la mémoire du modèle, permettant la contextualisation sur l'ensemble des mots d'une document. Contrairement à un modèle contextualisé basique, l'architecture Transformers utilise un système d'encoder / decoder, spécifiquement très efficace dans les tâches de traduction. L'encoder, entraîné à comprendre une langue particulière, crée une représentation vectorielle neutre d'un mot ou d'un document. Le decoder, entraîné également dans une langue

particulière (qui n'est pas forcément la même que celle de l'encoder), prend ce vecteur et le transforme afin d'obtenir le résultat attendu. Cette architecture permet de greffer n'importe quel décodeur sur l'encoder, le rendent plus facilement généralisable à toutes les langues. Cette étape du vecteur "neutre", que n'ont pas les autres architectures, oblige l'encoder à créer une représentation vectorielle la plus fidèle au mot, possible grâce à la couche d'attention. Actuellement, le modèle de Transformers le plus connu et utilisé est BERT [23].

Une différence notable entre ELMo et BERT, mis à part l'architecture de leur modèle, est leur gestion des mots. ELMo va considérer chaque mot directement, le contextualisant dans la phrase tandis que BERT va découper le mot en "sous-mots" (ex: sous-marin : sous et marin), si c'est possible, augmentant considérablement la taille du dictionnaire couvert. Ce système permet également de traiter des mots qui ne sont pas dans le dictionnaire d'entraînement et donc que le modèle n'a jamais vu, pour autant qu'il soit composé de "sous-mots" connus par ce dernier.

Cette architecture permet également de faire de la classification de texte. Seulement l'encoder doit alors être utilisé en amont d'une couche de classification basique (MLP par exemple) .

4.5 Métriques d'évaluation

Afin de pouvoir mesurer l'efficacité d'un algorithme ou d'un modèle, il est nécessaire d'utiliser des métriques d'évaluation. Ces dernières permettent de calculer les performances d'un modèle et donc, par la suite, de comprendre quels sont les éléments à changer pour l'améliorer. Le problème de cette thèse étant une tâche de classification (dire si un texte correspond à un critère ou non), les métriques les plus adaptées sont les suivantes :

- L'exactitude (*accuracy*);
- La précision (*precision*);
- Le rappel (*recall*);
- Le score F1.

Avant de pouvoir calculer ces quatre métriques, il faut préalablement comparer les résultats des prédictions avec les valeurs réelles des instances en comptant le nombre de prédictions justes ou fausses. Selon la valeur réelle de l'instance et le résultat de la prédiction, quatre cas peuvent apparaître dans un problème de classification binaire :

1. Le modèle a prédit **Positif** et la valeur réelle est **Positive** (TP);
2. Le modèle a prédit **Positif** et la valeur réelle est **Négative** (FP);
3. Le modèle a prédit **Négatif** et la valeur réelle est **Négative** (TN);

4. Le modèle a prédit **Négatif** et la valeur réelle est **Positive** (FN).

Instance	Valeur réelle	Valeur prédite	Résultat
1	1	1	TP
2	1	1	TP
3	1	0	FN
4	1	0	FN
5	0	1	FP
6	0	0	TN

Table 4.4: Exemple de classification

Dans le tableau d'exemple 4.4, chaque instance va être définie comme TP, TN, FP ou FN selon la valeur réelle et celle prédite.

4.5.1 Exactitude

L'exactitude (*accuracy*) est une métrique permettant de mesurer la quantité de résultats pertinents par rapport au nombre de résultats total. Elle se calcule de la manière suivante :

$$\text{accuracy} = \frac{|\text{Nombre de résultats pertinents}|}{|\text{Nombre de résultats}|}, \tag{4.52}$$

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{4.53}$$

Pour faire simple, l'exactitude est le nombre d'instances correctement prédites (TP+TN) divisées par le nombre d'instances totales (TP+TN+FP+FN). En reprenant les données du tableau 4.4, le calcul de l'exactitude est le suivant :

$$\text{accuracy} = \frac{2+1}{2+1+1+2} = \frac{3}{6} = 0.5. \tag{4.54}$$

4.5.2 Précision

La précision (*precision*) est une métrique permettant de mesurer la quantité de résultats pertinents par rapport à la réalité. Elle se calcule de la manière suivante :

$$\text{precision} = \frac{|\text{nombre de résultats pertinents}| \cap |\text{nombre de résultats}|}{|\text{nombre de résultats}|}, \tag{4.55}$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{4.56}$$

Pour faire simple, la précision est le nombre d'instances prédites positives et qui le sont réellement (TP) divisées par le nombre total d'instances prédites positives (TP+FP). En reprenant les données du tableau 4.4, le calcul de la précision est le suivant :

$$\text{accuracy} = \frac{2}{2+1} = \frac{2}{3} = 0.\overline{6}. \tag{4.57}$$

4.5.3 Rappel

Le rappel (*recall*) est une métrique permettant de mesurer la quantité de résultats pertinents qui devaient être récupérés et qui l'on effectivement été :

$$\text{recall} = \frac{|\text{nombre de résultats pertinents}| \cap |\text{nombre de résultats}|}{|\text{nombre de résultats pertinents}|}, \tag{4.58}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{4.59}$$

Pour faire simple, le rappel est le nombre d'instances prédites positives et qui le sont réellement (TP) divisées par le nombre total d'instances qui auraient dues être prédites positives (TP+FN). En reprenant les données du tableau 4.4, le calcul du rappel est le suivant :

$$\text{recall} = \frac{2}{2+2} = \frac{2}{4} = 0.5. \tag{4.60}$$

4.5.4 Score F1

Le score F1 est la moyenne harmonique de la précision et du rappel. Un score F1 de 1 induit une classification parfaite.

$$\text{F}_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}}, \tag{4.61}$$

$$\text{F}_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \tag{4.62}$$

$$\text{F}_1 = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}. \tag{4.63}$$

En reprenant les données du tableau 4.4, le calcul du score F1 est le suivant :

$$\text{F}_1 = 2 \cdot \frac{0.6 \cdot 0.5}{0.6 + 0.5} = 0.5. \tag{4.64}$$

Le score F1 est utilisé comme métrique principale pour mesurer la performance d'un modèle car elle

prend en compte la précision et le rappel en une seule métrique. Si, pour une tâche particulière, on désire donner une place plus ou moins importante au rappel, il est possible d'utiliser une variante du score F1 :

$$\mathsf{F}_\beta = (1+\beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall})}. \tag{4.65}$$

4.5.5 Micro vs Macro métriques

Il existe deux manières de calculer les métriques ci-dessus lorsque l'on travail sur un problème de classification multi-classes (plus de 2 classes) ou multi-label (2 labels ou plus avec minimum 2 classes par label). Le calcul Micro et Macro vont influencer les résultats de nos métriques selon la manière dont sont distribuées les données. Une métrique "Macro" va calculer la métrique de manière indépendante pour chaque classe et chaque label puis prendre la moyenne. De ce fait, chaque classe et chaque label sont traités de manière équitable. Une métrique "Micro", quant à elle, va prendre en compte la contribution de chaque label et chaque classe dans le calcul de la moyenne, ce qui signifie que la moyenne prend en compte la distribution des données.

Si les données sont inégalement distribuées, l'utilisation d'une métrique "Micro" est préférable [43].

Afin de bien illustrer ce concept, prenons comme exemple un jeu de données multi-classes inégalement distribué dont la prédiction donne les résultats suivants :

Classe	Nombre d'instances	Résultats			
		TP	FP	TN	FN
A	5	3	2	0	0
B	100	10	90	0	0
C	2	1	1	0	0

Table 4.5: Exemple micro / macro métriques

En utilisant les métriques précédemment décrites, nous pouvons calculer la précision de chacune des classes pour les données d'exemple du tableau 4.5 :

$$\text{precision}_\text{A} = 0.6, \tag{4.66}$$

$$\text{precision}_\text{B} = 0.1, \tag{4.67}$$

$$\text{precision}_\text{C} = 0.5. \tag{4.68}$$

En calculant la précision moyenne sur l'ensemble des classes en macro et micro, nous obtenons les résultats suivants:

$$\text{precision}_{\text{macro}} = \frac{0.6 + 0.1 + 0.5}{3} = 0.4, \tag{4.69}$$

$$\text{precision}_{\text{micro}} = \frac{3 + 10 + 1}{5 + 100 + 2} = 0.13. \tag{4.70}$$

Il y a une grosse différence entre le calcul "micro" et "macro". Il semble logique que la macro précision soit plus élevée car les bonnes précisions des classes A et C (0.6 et 0.5) ont autant de poids que celle de la classe B (0.1) bien que le nombre d'instances y soit bien plus faible. Cela permet de garder une précision moyenne plutôt haute. Néanmoins, celle-ci est faussée car, en globalité, un grand nombre d'instances ont été mal classifiées (93 sur 107, soit 86.92%). La moyenne "micro" quant à elle, va correctement prendre en compte la distribution inégale des classes. Son résultat sera alors plus proche de la réalité.

4.5.6 Informations importantes à propos des métriques

Il est très important de positionner les métriques utilisées dans leur contexte. L'exactitude, par exemple, est fortement influencée par la distribution des données. En effet, si une classe est fortement majoritaire, il est facile d'obtenir une exactitude de 75-80% juste en prédisant la classe majoritaire [44]. Il est alors nécessaire de comparer nos résultats avec des algorithmes de classification naïfs (aléatoires) pour vérifier que notre modèle ne prédise par uniquement la classe majoritaire. Comme décrit précédemment, les trois algorithmes suivants ont été utilisés :

1. Random Guess Classifier;
2. Weighted Guess Clasifier;
3. Majoritiy Class Classifier.

En dérivant algébriquement les résultats des métriques sur ces trois classifier [45], il est possible de dire que :

	Exactitude	Précision	Rappel
Random Guess Classifier	0.5	x	0.5
Weighted Guess Classifier	$x^2 + (1 + x)^2$	x	x
Majority Class Classifier	$1 - x$	0	0

Table 4.6: Performances statistiques des algorithmes naïfs [44]

En définisant que :

$$x = \text{Pourcentage d'instances classifiées positives}$$
$$(1 - x) = \text{Pourcentage d'instances classifiées négatives}$$

4.5.7 Test de Kolmogorov-Smirnov

Le test Kolmogorov-Smirnov (KS) [46, 47] permet de comparer 2 distributions. Il existe deux versions de ce test :

1. One-Sample KS test: permet de tester si un échantillon vient d'une distribution spécifique;
2. Two-Sample KS test: permet de tester si deux échantillons viennent de la même distribution.

Afin de comparer la distribution des prédictions avec la distribution des données réelles, la version Two-Sample est la plus adaptée. Elle se calculs de la manière suivante:

$$D_{n,m} = \sup_x |f_{1,n}(x) - f_{2,m}(x)|, \tag{4.71}$$

où $f_{1,n}(x)$ est la fréquence cumulative de la distribution du premier échantillon, $f_{2,m}(x)$ la fréquence cumulative de la distribution du deuxième échantillon et $\sup$ la fonction supremum [48].

Les deux distributions sont inégales si $D_{n,m}$ est plus grand que l'hypothèse nulle, défini par l'équation suivante:

$$D_{n,m} > c(\alpha)\sqrt{\frac{n+m}{n \cdot m}}, \tag{4.72}$$

où n et m sont les tailles des échantillons et α le risque d'erreur. $c(\alpha)$ est fourni par la table 4.7 ou l'équation 4.73.

α	0.20	0.15	0.10	0.05	0.025	0.01	0.005	0.001
$c(\alpha)$	1.073	1.138	1.224	1.358	1.48	1.628	1.731	1.949

Table 4.7: Résultats de la fonction $c(\alpha)$ pour les valeurs les plus communes de a.

Le résultat de la fonction c pour d'autres valeurs α peut être calculé en utilisant l'équation définie par:

$$c(\alpha) = \sqrt{-\ln(\frac{\alpha}{2}) \cdot \frac{1}{2}}. \tag{4.73}$$

En combinant les équations 4.72, 4.73, il est possible de définir que 2 distributions sont différentes si l'équation suivante est confirmée:

$$D_{n,m} > \frac{1}{\sqrt{n}} \cdot \sqrt{-\ln(\frac{\alpha}{2}) \cdot \frac{1+\frac{n}{m}}{2}}. \tag{4.74}$$

De manière générale, le risque d'erreur le plus utilisé est 5% (α = 0.05).

L'ensemble des tests KS ont été calculés en utilisant la librairie spicy [49]. Cette dernière retourne la valeur statistique $D_{n,m}$ ainsi qu'une P-Value, les deux entre 0 et 1. La valeur statistique $D_{n,m}$ définit si 2 distributions sont semblables ou non. Plus elle est petite, plus les distributions sont égales.

La P-Value permet de définir si l'hypothèse nulle est rejetée. Si elle est plus petite que la valeur du risque (0.05 par exemple), les deux distributions sont considérées comme statistiquement différentes. Par conséquent, la P-Value agit à peu près de la même manière que la valeur statistique $D_{m,n}$. Plus la P-Value est grande, plus les distributions sont égales.

5 Résultats

Ce chapitre se découpe en 5 sections. Premièrement, les résultats des algorithmes naïfs décrits dans la section 4.5.6 ont été calculés afin d'avoir une baseline sur laquelle comparer les différents modèles testés. Ensuite, différents modèles d'embeddings ont été analysés et testés afin d'établir lequel est le plus performant pour ce problème.

Chacun des modèles de Machine Learning ou de Deep Learning utilisés par la suite ont été entraînés avec les données provenant du meilleur modèle d'embeddings. Des algorithmes de classifications classiques ont été testés (RandomForest, DecisionTree, etc) puis différents modèles: MLP, CNN, RNN, LSTM. Pour finir, des modèles d'embeddings end-to-end, basés sur l'architecture Transformers, ont été testés. L'ensemble des paramètres des modèles ont été optimisés en utilisant Optuna [50].

5.1 Algorithmes naïfs

Comme discuté dans la section 4.5.6, il est nécessaire de vérifier que les résultats de nos modèles soient plus performants que ceux des classificateurs naïfs. Si c'est le cas, il est probable que le modèle apprenne des données et que ses prédictions ne soient pas aléatoires ou basées sur la classe majoritaire.

Avec les données du challenge N2C2 shared task (track 1), les résultats des algorithmes naïfs sont les suivants:

Classificateur	Macro-résultats			
	Exactitude	Précision	Rappel	F1
Random Guess	0.500	0.442	0.500	0.396
Weighted Guess	0.710	0.442	0.442	0.442
Majority Guess	0.550	0	0	0

Table 5.1: Résultats des classificateurs naïfs sur les données de N2C2 shared task (track 1). Ces résultats sont calculés mathématiquement en utilisant les formules des sections 4.5.6 et 4.5.4.

Les algorithmes naïfs, comme il est possible de le voir dans le tableau 5.1, permettent d'obtenir une exactitude supérieure à 70% en utilisant le classificateur Weight Guess. Cela s'explique assez facilement au vu de la distribution des données. En effet, le classificateur pondérant les résultats par rapport à cette distribution, ils pourront être satisfaisants dans les critères inégalement distribués tout comme pour les autres critères. Il est important de noter cependant que ces résultats sont basés sur des calculs et ne reflètent pas de réelles prédictions.

Les résultats du classificateur Majority Guess, quant à eux, semblent assez logiques du fait de la distribution des données. En effet, ils pourront se rapprocher de 100% d'exactitude dans les critères

inégalement distribués mais ne pourront jamais dépasser les 50% dans ceux qui sont bien distribués. Par conséquent, la macro-moyenne aura tendance à donner un résultat proche de 50%.

Critère	Random Guess Classifier			Weighted Guess Classifier			Majority Guess Classifier		
	Précision	Rappel	F1	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.383	0.500	0.433	0.383	0.383	0.383	0	0	0
ADVANCED-CAD	0.624	0.500	0.555	0.624	0.624	0.624	0	0	0
ALCOHOL-ABUSE	0.034	0.500	0.065	0.034	0.034	0.034	0	0	0
ASP-FOR-MI	0.801	0.500	0.615	0.801	0.801	0.801	0	0	0
CREATININE	0.408	0.500	0.449	0.408	0.408	0.408	0	0	0
DIETSUPP-2MOS	0.523	0.500	0.511	0.523	0.523	0.523	0	0	0
DRUG-ABUSE	0.061	0.500	0.110	0.061	0.061	0.061	0	0	0
ENGLISH	0.953	0.500	0.656	0.953	0.953	0.953	0	0	0
HBA1C	0.334	0.500	0.401	0.334	0.334	0.334	0	0	0
KETO-1YR	0.005	0.500	0.011	0.005	0.005	0.005	0	0	0
MAJOR-DIABETES	0.571	0.500	0.533	0.571	0.571	0.571	0	0	0
MAKES-DECISIONS	0.960	0.500	0.657	0.960	0.960	0.960	0	0	0
MI-6MOS	0.091	0.500	0.153	0.091	0.091	0.091	0	0	0
Moyenne	0.442	0.500	0.396	0.442	0.442	0.442	0	0	0

Table 5.2: Macro résultats détaillés des algorithmes naïfs. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

Ces réflexions se confirment sur les résultats détaillés de chaque critère. Dans le tableau 5.2, les critères dont la distribution est inégale ont des résultats très bons, proches de 90%. Par contre, les résultats des critères correctement distribués sont proches ou inférieurs à 50%. La valeur médiane du test statistique KS est de 0.209 et la P-Value médiane plus faible que le seuil critique de 5% (0.05) indiquant que la distribution des prédictions du classificateur Majority Guess est statistiquement différente de celle des données de test.

Afin de comparer les résultats des modèles sur ceux des classificateurs naïfs, il est essentiel de regarder la distribution des prédictions pour les critères inégalement distribués. En effet, il sera fort probable que les modèles prédisent la classe majoritaire, ce qui sera alors visible. Il est également indispensable de comparer les critères qui sont bien distribués. En les maximisant, les modèles doivent impérativement apprendre des données.

5.2 Analyse des modèles d'embeddings

L'entiéreté des valeurs utilisées dans la prédiction des critères sont textuelles. Il est donc obligatoire de transformer ces textes en vecteurs avant de pouvoir les traiter. Comme expliqué dans la section 4.4.4, plusieurs modèles d'embeddings existent, chacun ayant ses propres spécificités. Afin de savoir lequel est le plus adapté au jeu de données, ils ont tous été testés de la manière suivante:

1. Les textes ont été transformés en vecteur en utilisant le modèle d'embeddings à tester;
2. Les données embeddées sont passées à une régression logistique dont les paramètres sont les suivants:
    - Fonction de calcul des erreurs: MSE
    - Optimiseur: Adam
        i. Learning rate: 0.00001
        ii. Weight decay: 0.2
    - Batch size: 128
3. Les résultats de la régression logistique (compris entre 0 et 1) sont arrondis afin d'obtenir une prédiction binaire.

Les différents modèles d'embeddings testés ont été implémentés grâce à la librairie *flair* [51], développée par l'Université de Berlin. Cette dernière offre différents modèles pré-entraînés :

Modèle d'embeddings	Type
BytePairEmbeddings	Subword-level word embeddings
CharacterEmbeddings	Task-trained character-level embeddings of words
ELMoEmbeddings	Contextualized word-level embeddings
FastTextEmbeddings	Word embeddings with subword features
FlairEmbeddings	Contextualized character-level embeddings
OneHotEmbeddings	Standard one-hot embeddings of text or tags
PooledFlairEmbeddings	Pooled variant of FlairEmbeddings
TransformerWordEmbeddings	Embeddings from pretrained Transformers (BERT, XLM, GPT, RoBERTa, XLNet, DistilBERT etc.)
WordEmbeddings	Classic word embeddings

Table 5.3: Liste des modèles d'embeddings disponibles dans la librairie flair

L'équipe de la librairie *flair* a créé son propre modèle d'embeddings contextualisé (FlairEmbeddings) [52]. Ce modèle a battu les derniers records dans différents domaines tel que le Named Entity Recognition, Part-of-Speech tagging et Chunking [53]. Comme les données médicales sont très complexes, il est nécessaire d'utiliser des modèles performants et contextualisés afin de capturer un maximum d'informations du texte. Pour ce faire, trois modèles et leur dérivées ont été sélectionnés:

1. TransformerWordEmbeddings (BERT)

- En utilisant seulement la première couche
- En utilisant les 4 dernières couches
- En utilisant toutes les couches

2. FlairEmbeddings
    - News (entraîné avec un corpus de 1 milliard de mots)
    - Pubmed (entraîné avec 5% des données du site Pubmed [54])
3. ELMoEmbeddings
    - Large (entraîné avec un corpus de 1 milliard de mots)
    - Pubmed (entraîné avec des données provenant du site Pubmed [54])

Flair n'a pas réimplémenté les modèles ELMoembeddings et TransformerWordEmbeddings. La librairie n'est qu'une interface avec ceux déjà existants. Les différents modèles testés utilisent l'opération de pooling afin de pouvoir transformer un document entier en vecteur. Pour les modèles FlairEmbeddings et ElmoEmbeddings, les trois différentes opérations de pooling ont été testées. L'opération de pooling de BERT est spécifique et ne peux pas être modifiée. Comme la librairie *flair* le précise, afin d'obtenir de meilleurs résultats en utilisant les modèles FlairEmbeddings, il est conseillé d'utiliser également un WordEmbeddings "GloVe" en amont de ceux de *flair*.

L'ensemble des modèles de ce travail ont été testés uniquement avec les données provenant du modèle d'embeddings le plus performant.

Models	Micro-results			Macro-results		
	Precision	Recall	F1	Precision	Recall	F1
Bert - first layer	0.680	0.794	0.733	0.344	0.531	0.407
Bert - all layers	0.703	0.766	0.733	0.323	0.453	0.372
Bert - layers 1 to 4	0.710	0.750	0.730	0.356	0.418	0.371
Flair News - mean pooling	0.700	0.774	0.735	0.322	0.460	0.373
Flair News - min pooling	0.704	0.769	0.735	0.323	0.456	0.373
Flair News - max pooling	0.699	0.775	0.735	0.376	0.591	0.447
Flair Pubmed - mean pooling	**0.720**	0.725	0.722	0.324	0.419	0.359
Flair Pubmed - max pooling	0.698	0.775	0.735	0.381	0.453	0.386
Flair Pubmed - min pooling	0.699	0.775	0.735	0.322	0.461	0.373
ELMo large - mean pooling	0.707	0.770	0.737	0.324	0.457	0.374
ELMo large - min pooling	0.705	0.771	0.736	0.324	0.457	0.374
ELMo large - max pooling	0.705	0.771	0.736	0.403	0.517	0.409
ELMo Pubmed - mean pooling	0.688	**0.795**	**0.738**	**0.421**	**0.736**	**0.485**
ELMo Pubmed - min pooling	0.709	0.759	0.733	0.405	0.451	0.386
ELMo Pubmed - max pooling	0.709	0.763	0.735	0.361	0.514	0.390

Table 5.4: Résultats des modèles d'embeddings. En gras, les meilleurs résultats.

La première chose qu'il est possible de remarquer dans le tableau 5.4 est la différence minime de performance entre les différents modèles dans la métrique micro-F1. Tous les modèles tiennent entre 72.2% et 73.8%, soit une différence maximum de 1.6%. Il est par contre intéressant de noter que, bien que le micro-F1 ne change pas beaucoup, les métriques qui le composent (micro-précision et micro-rappel) peuvent varier beaucoup plus. La micro-précision a un écart maximum de 4% et le micro-rappel de 7% entre le moins bon et le meilleur modèle. Les métriques macro, quant à elles, ont une plus grande tendance à varier entre les différents modèles.

Une seconde chose intéressante à relever dans ce tableau est le fait que, pour la plupart des modèles, l'utilisation de l'opération de pooling "min" augmente légèrement le micro-F1. A l'inverse, c'est l'opération de pooling "max" qui augmente le plus le macro-F1.

Comme chaque modèle a une architecture et un entraînement différent, la taille des vecteurs de sortie est également différente. Il est possible que ce soit la taille de l'embedding qui influence les résultats.

Modèle d'embeddings	Taille d'embeddings
Bert - first layer	768
Bert - all layers	9984
Bert - layers 1 to 4	3072
Flair News	4196
Flair Pubmed	2400
ELMo large	3072
ELMo Pubmed	3072

Table 5.5: Taille des modèles d'embeddings testés

Comme il est possible de le voir dans le tableau 5.5, le meilleur modèle (ELMo Pubmed) a une taille de 3072 tandis que le moins bon (Flair Pubmed) a une taille de 2400. A première vue, il est possible d'imaginer que la performance est corrélée avec la taille de l'embedding. Plus il est grand, meilleures sont les performances. Néanmoins, deux autres modèles ont des vecteurs plus grands mais des performances plus faibles. On pourrait alors penser que la taille idéale se situe autour de 3100 mais ce n'est pas forcément le cas. Le modèle Bert avec les quatre premières couches a la même taille que celui d'ELMo mais obtient de moins bons résultats. Par conséquent, le seul élément qui peut entrer en compte est l'architecture elle-même du modèle et les données utilisées lors de l'entraînement.

D'après ce test, le meilleur modèle d'embedding pour le problème de classification du challenge N2C2 shared taks (track 1) est le modèle ELMo entrainé avec des données médicales provenant du site Pubmed.

5.3 Régression logistique

Un modèle logistique simple permet d'obtenir des résultats, à première vue, plutôt satisfaisants. Avec une micro-F1 de 73.8% et une macro-F1 de 40.7%, en utilisant les données embedées par le modèle d'embeddings ELMo Pubmed - mean pooling, le modèle logistique utilisé pour comparer les différents modèles d'embeddings parvient à surpasser les résultats des classificateurs naïfs. Une amélioration de 2% environ est visible.

En examinant les résultats détaillés (table 5.6), il est possible de voir qu'ils sont très bons pour les critères inégalement distribués et proche de 50% pour ceux qui sont bien distribués. Ces résultats ne sont que très légèrement supérieurs à ceux des classificateurs et laissent penser que la régression logistique agit de la même manière que les classificateurs naïfs, en prédisant la classe majoritaire. Afin de vérifier si, effectivement, le modèle apprend des données, il faut regarder la distribution des classes prédites par le modèle (figure 5.1).

Critère	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.618	0.618	0.618	0.485	0.496	0.428
ADVANCED-CAD	0.588	0.588	0.588	0.793	0.503	0.376
ALCOHOL-ABUSE	0.960	0.960	0.960	0.482	0.497	0.489
ASP-FOR-MI	0.790	0.790	0.790	0.395	0.500	0.441
CREATININE	0.729	0.729	0.729	0.864	0.509	0.440
DIETSUPP-2MOS	0.496	0.496	0.496	0.249	0.494	0.331
DRUG-ABUSE	0.957	0.957	0.957	0.483	0.494	0.489
ENGLISH	0.843	0.843	0.843	0.421	0.500	0.457
HBA1C	0.591	0.591	0.591	0.561	0.535	0.505
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.501	0.501	0.501	0.250	0.500	0.333
MAKES-DECISIONS	0.968	0.968	0.968	0.484	0.500	0.491
MI-6MOS	0.904	0.904	0.904	0.452	0.500	0.474
Moyenne	0.688	0.795	0.738	0.464	0.483	0.407

Table 5.6: Résultats détaillés de la régression logistique. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

Comme il est possible de le voir dans la figure 5.1, le modèle a prédit la classe majoritaire pour la plupart des critères, qu'ils soient inégalement distribués ou non. De manière intéressante, les deux critères les mieux distribués (DIETSUPP-2MOS, MAJOR-DIABETES) ont également été prédits par la classe majoritaire tandis que les critères ABDOMINAL et HBA1C n'ont pas totalement été prédits par la classe majoritaire, alors que leur distribution n'est que d'environ 35% / 65%.

En optimisant les paramètres, il est possible d'augmenter légèrement les résultats. La micro-F1 augmente de 0.055% et la macro-F1 de 0.398%. Bien que les métriques n'augmentent que légèrement,

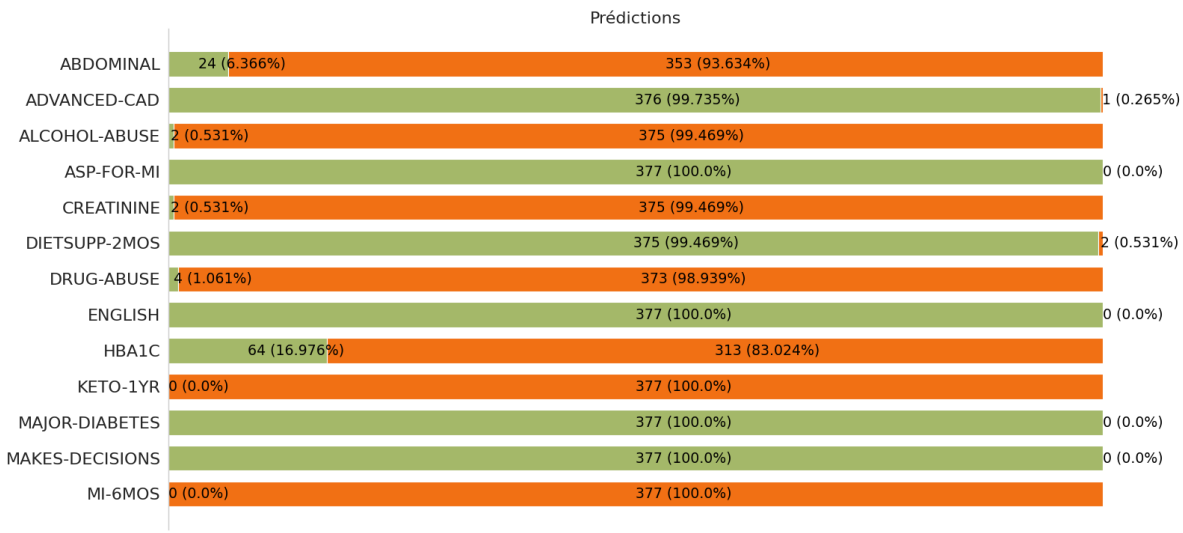

Figure 5.1: Distribution des prédictions - Regression logistique. En vert, les prédictions positives. En orange, les prédictions négatives.

la distribution des prédictions est par contre légèrement plus ressemblante à celle des données de test (figure 5.2), avec une valeur médiane du test de KS de 0.156 et une P-Value médiane de 0.00019, plus proche du seuil critique. Les deux distributions restent néanmoins statistiquement différentes. Les critères très fortement inégalement distribués continuent à être prédits par la classe majoritaire. Par contre, les critères ayant une distribution plus égale commencent à être prédit positif comme négatif. Il est intéressant de noter que pour ces critères là, le modèle va quand même prédire une majorité d'enregistrements positifs, même s'il y a plus d'enregistrements négatifs dans les données d'entraînement (ABDOMINAL, CREATININE, DIETSUPP-2MOS, HBA1C). Pour obtenir de meilleurs résultats tout en ayant une distribution des prédictions plus ressemblante à celle des données de test, la plupart des enregistrements ont été prédits dans la bonne classe, ce qui induit que le modèle a dû apprendre des données.

Ces résultats ont été obtenus avec les paramètres suivants:

- Fonction de calcul des erreurs: FocalLoss
  - alpha: 0.99
  - gamma: 3.5
- Optimiseur: Adam
  - Learning rate: 0.0036
  - Weight decay: 0.0111
- Batch size: 128

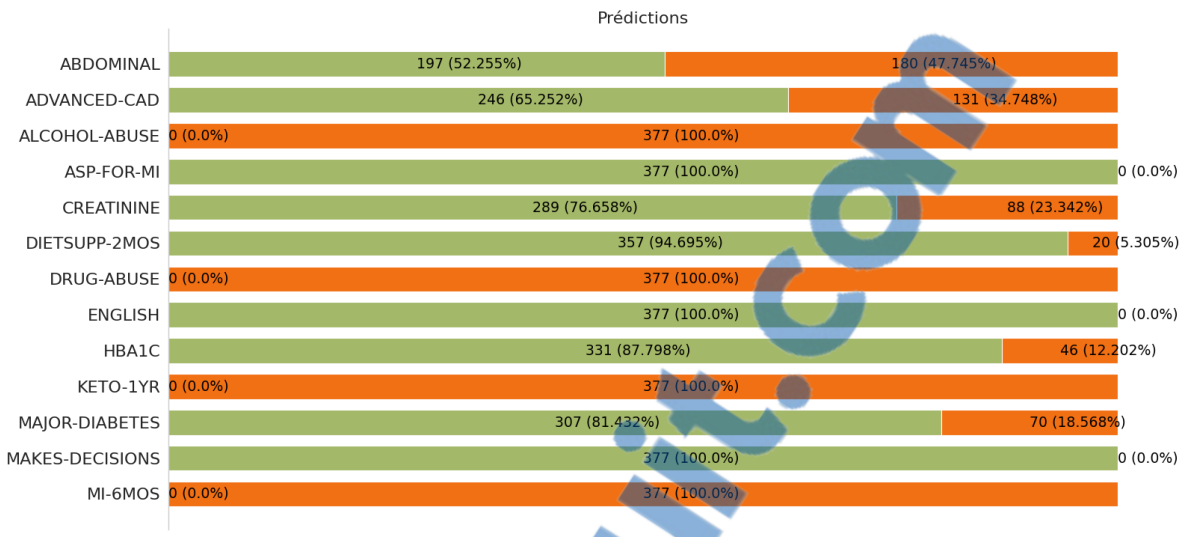

Figure 5.2: Distribution des prédictions - Meilleure régression logistique. En vert, les prédictions positives. En orange, les prédictions négatives.

5.4 KNN

L'algorithme KNN permet d'obtenir des résultats légèrements plus faibles que ceux d'un modèle de régression logistique correctement paramétré avec une micro-F1 de 72.4% et une macro-F1 de 43.8%. Comme il est possible de le remarquer dans le tableau 5.7, la différence de macro-F1 est plutôt faible entre les critères correctement distribués et ceux qui ne le sont pas. De manière générale, ce sont même ces premiers qui sont les mieux classifiés. Cela se voit sur la distribution des prédictions (figure 5.3), où le KNN fait mieux que la régression logistique avec une valeur médiane du test de KS de 0.095 et une P-Value médiane de 0.064, supérieur au seuil critique de 5%. Les deux distributions sont alors statistiquement égales. Cela peut s'expliquer par le fait que, dans une régression logistique, le même vecteur de poids permet de traiter l'ensemble des enregistrements, qu'ils soient positifs ou négatifs, correctement distribués ou non. Par conséquent, ce dernier se spécialise sur les données les plus présentes et a du mal à prédire les classes minoritaires. Dans le cas du KNN, il n'y a pas de vecteur de poids. L'algorithme se réfère aux instances les plus proches, sans préférence de classes et ne sera alors pas trop influencé par une mauvaise distribution.

Les paramètres utilisés pour obtenir ces résultats sont les suivants :

- Nombre de voisins (k): 15
- Distance utilisée: euclidienne
- Power (p): 1
- Leaf size: 35

Criteria	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.570	0.570	0.570	0.514	0.513	0.511
ADVANCED-CAD	0.599	0.599	0.599	0.576	0.540	0.505
ALCOHOL-ABUSE	0.965	0.965	0.965	0.482	0.500	0.491
ASP-FOR-MI	0.785	0.785	0.785	0.562	0.505	0.462
CREATININE	0.665	0.665	0.665	0.566	0.560	0.562
DIETSUPP-2MOS	0.511	0.511	0.511	0.512	0.511	0.502
DRUG-ABUSE	0.968	0.968	0.968	0.484	0.500	0.491
ENGLISH	0.843	0.843	0.843	0.421	0.500	0.457
HBA1C	0.572	0.572	0.572	0.538	0.528	0.513
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.519	0.519	0.519	0.522	0.519	0.501
MAKES-DECISIONS	0.968	0.968	0.968	0.484	0.500	0.491
MI-6MOS	0.904	0.904	0.904	0.452	0.500	0.474
Moyenne	0.692	0.759	0.724	0.420	0.471	0.438

Table 5.7: Résultats détaillés de KNN. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

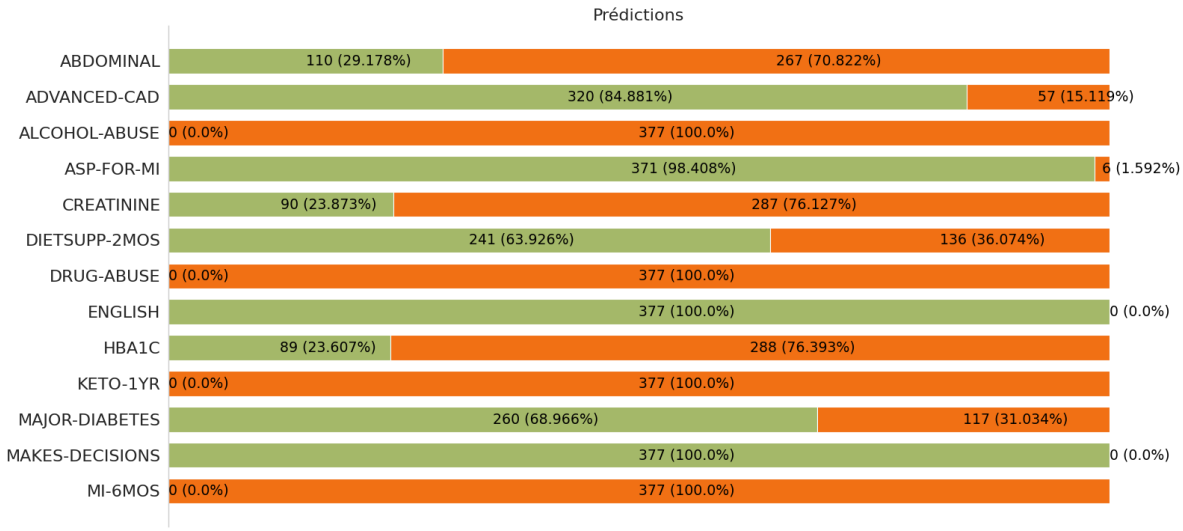

Figure 5.3: Distribution des prédictions - KNN. En vert, les prédictions positives. En orange, les prédictions négatives.

Il est intéressant de noter que, comme le démontre la figure 5.3, la distribution des prédictions ressemble beaucoup plus à celle des données de test que lors de la régression logistique. Bien que les métriques soient un peu plus faibles, il est possible d'affirmer qu'une grande partie des enregistrements ont été correctement classifiés. Les critères qui ont une distribution plus mauvaise que 30% / 70% dans les données d'entraînement continuent à être classifiés par la classe majoritaire tandis que le modèle arrive à apprendre des données des critères correctement distribués.

5.5 Radius Neighbors

L'algorithme Radius Neighbors est légèrement plus performant que KNN, avec un amélioration de 1.2% du micro-F1 pour atteindre 73.6% et une nette baisse de la macro-F1 de 6.5% pour atteindre 37.3%. Comme visible dans le tableau 5.8, les résultats macro pour les classes correctement distribuées sont bien plus faibles que ceux des classes inégalement distribuées. Cela s'explique assez facilement par le fait que le modèle prédise uniquement la classe majoritaire pour l'ensemble des critères. Cette hypothèse se confirme dans la figure 5.4, où la distribution des prédictions est clairement basée sur les classes majoritaires, avec une faible P-Value médiane plus faible que le seuil critique de 5%, indiquant que les deux distributions sont inégales. Une explication possible de cette baisse des résultats et de cette mauvaise distribution des prédictions est que le nombre d'instances voisines utilisées dans la prédiction n'est pas fixée, comme dans KNN. En effet, il est possible et probable même que, pour certaines prédictions, plusieurs dizaines d'instances d'entraînement sont sélectionnées tandis que pour d'autres prédictions, seuls quelques instances le sont. Cela induit une irrégularité potentiellement visible dans les résultats. Les paramètres utilisés pour obtenir ces résultats sont les suivants:

- Rayon (r): 9
- Distance utilisée: euclidienne
- Power (p): 3
- Leaf size: 45

Critère	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.639	0.639	0.639	0.319	0.5	0.389
ADVANCED-CAD	0.586	0.586	0.586	0.293	0.5	0.369
ALCOHOL-ABUSE	0.965	0.965	0.965	0.482	0.5	0.491
ASP-FOR-MI	0.790	0.790	0.790	0.395	0.5	0.441
CREATININE	0.724	0.724	0.724	0.362	0.5	0.420
DIETSUPP-2MOS	0.533	0.533	0.533	0.609	0.532	0.432
DRUG-ABUSE	0.968	0.968	0.968	0.484	0.5	0.491
ENGLISH	0.843	0.843	0.843	0.421	0.5	0.457
HBA1C	0.586	0.586	0.586	0.293	0.5	0.369
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.501	0.501	0.501	0.250	0.5	0.333
MAKES-DECISIONS	0.968	0.968	0.968	0.484	0.5	0.491
MI-6MOS	0.904	0.904	0.904	0.452	0.5	0.474
Moyenne	0.703	0.771	0.736	0.323	0.457	0.373

Table 5.8: Résultats détaillés de Radius Neighbors. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

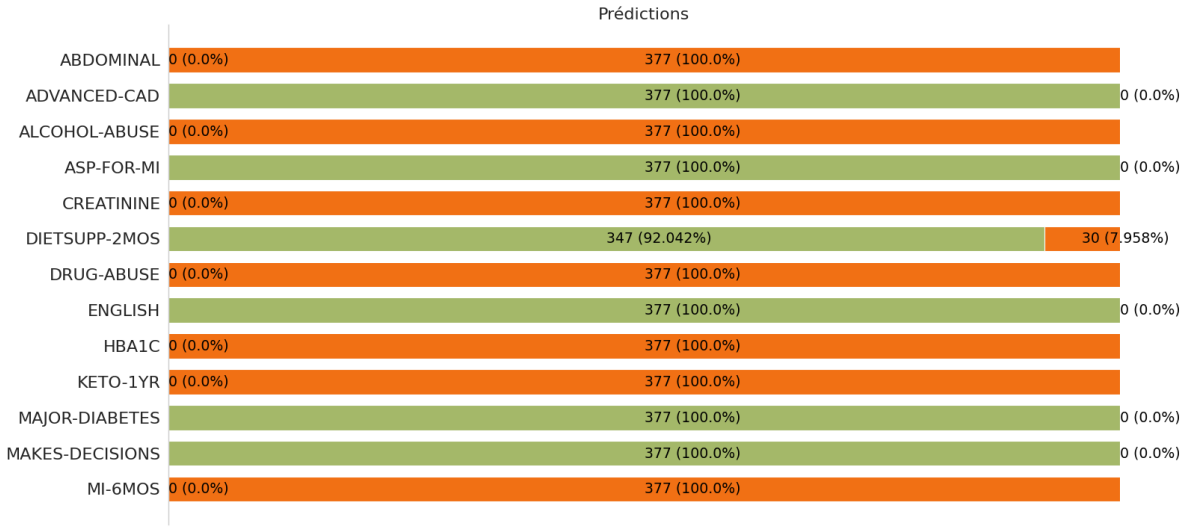

Figure 5.4: Distribution des prédictions - Radius Neighbors. En vert, les prédictions positives. En orange, les prédictions négatives.

5.6 Arbre de décision

L'arbre de décision se situe entre le KNN et le Radius Neighbors. Avec une micro-F1 de 73.5%, il surpasse légèrement celle du KNN. Néanmoins, sa faible macro-F1 de 41.6% donne l'impression qu'il est finalement moins performant que KNN. Comme cela est visible dans le tableau 5.9, les critères ont, de manière générale, les mêmes résultats que ceux des classificateurs naïfs. Par conséquent, cela implique que la majorité des prédictions sont basées sur les classes majoritaires. Les seules différences notables sont les critères ABDOMINAL et CREATININE qui ont tous les deux exactement 71 enregistrements prédits positifs, comme il est possible de le voir à la figure 5.5. Cela est assez spécial car ces deux critères ne sont pas corrélés, comme expliqué dans la section 3.3. Avec un médiane du test de KS de 0.156 et une P-Value médiane de 0.00019 la distribution des prédictions de l'arbre de décision ressemble légèrement plus aux données de test que celle du Radius Neihgbors, mais est moins bonne que celles du KNN. Assez étonnamment, elle est statistiquement équivalente à celle de la régression logistique.

Les paramètres utilisés pour obtenir ces résultats sont les suivants:

- Criterion : entropy
- Maximum features: number of features of the dataset
- Splitter: best random

- Maximum depth: 1

Critère	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.588	0.588	0.588	0.503	0.502	0.483
ADVANCED-CAD	0.586	0.586	0.586	0.293	0.500	0.369
ALCOHOL-ABUSE	0.965	0.965	0.965	0.482	0.500	0.491
ASP-FOR-MI	0.790	0.790	0.790	0.395	0.500	0.441
CREATININE	0.679	0.679	0.679	0.564	0.549	0.549
DIETSUPP-2MOS	0.501	0.501	0.501	0.250	0.500	0.333
DRUG-ABUSE	0.968	0.968	0.968	0.484	0.500	0.491
ENGLISH	0.843	0.843	0.843	0.421	0.500	0.457
HBA1C	0.586	0.586	0.586	0.293	0.500	0.369
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.501	0.501	0.501	0.250	0.500	0.333
MAKES-DECISIONS	0.968	0.968	0.968	0.484	0.500	0.491
MI-6MOS	0.904	0.904	0.904	0.452	0.500	0.474
Moyenne	0.679	0.801	0.735	0.379	0.496	0.416

Table 5.9: Résultats détaillés de l'arbre de décision. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

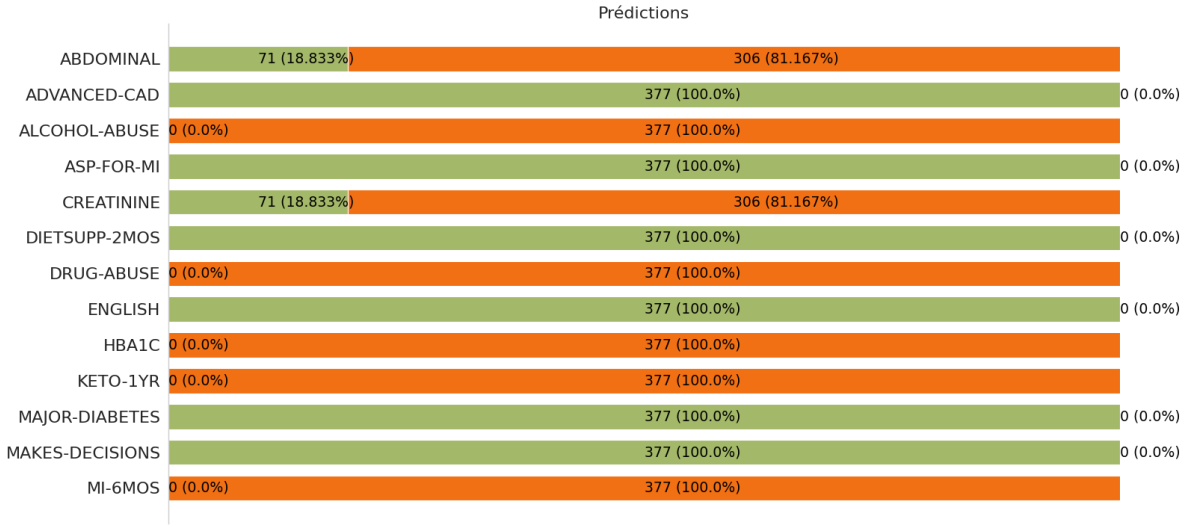

Figure 5.5: Distribution des prédictions - DecisionTree. En vert, les prédictions positives. En orange, les prédictions négatives.

5.7 Random Forest

Comme le Random Forest est une composition de plusieurs DecisionTree, il semble logique que les résultats soient également plus faibles que ceux de la régression logistique et du KNN. Par contre,

ils doivent normalement être plus élevés que ceux d'un DecisionTree seul. En effet, la micro-F1 est de 73.6%, soit une augmentation de 0.1% et la macro-F1 est de 41.7%, soit une augmentation de 0.1%. Bien que l'augmentation des résultats ne soit que très légère, la distribution des prédictions ressemble un peu plus à celle des données de test, comme visible à la figure 5.6. La valeur médiane du test de KS est de 0.132 et la P-Value médiane de 0.002. La distribution des prédictions est alors statistiquement différente de celle des données de test mais tout de même plus ressemblante que celle de l'arbre de décision. Les paramètres utilisés pour obtenir ces résultats sont les suivants :

- Profondeur maximum: 31
- Nombre d'estimateurs: 155
- Utilisation d'échantillons "bootstrap"

Critère	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.623	0.623	0.623	0.530	0.511	0.465
ADVANCED-CAD	0.596	0.596	0.596	0.577	0.528	0.472
ALCOHOL-ABUSE	0.965	0.965	0.965	0.482	0.500	0.491
ASP-FOR-MI	0.790	0.790	0.790	0.395	0.500	0.441
CREATININE	0.708	0.708	0.708	0.603	0.569	0.571
DIETSUPP-2MOS	0.564	0.564	0.564	0.569	0.564	0.556
DRUG-ABUSE	0.968	0.968	0.968	0.484	0.500	0.491
ENGLISH	0.843	0.843	0.843	0.421	0.500	0.457
HBA1C	0.596	0.596	0.596	0.598	0.520	0.436
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.557	0.557	0.557	0.591	0.556	0.508
MAKES-DECISIONS	0.968	0.968	0.968	0.484	0.500	0.491
MI-6MOS	0.904	0.904	0.904	0.452	0.500	0.474
Moyenne	0.720	0.753	0.736	0.442	0.456	0.417

Table 5.10: Résultats détaillés du Random Forest. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

5.8 MLP

Les perceptrons multi-couches (MLP), bien qu'ils aient la possibilité d'augmenter grandement le nombre de vecteurs de poids et donc le nombre d'informations pertinentes récupérables, ne sont pas meilleurs que les modèles basiques. Avec une micro-F1 de 74.2% et une macro-F1 de 41.8%, ces résultats sont très proches de la régression logistique, comme le démontre le tableau 5.11. Néanmoins, la distribution des prédictions, visible à la figure 5.7 permet de voir que le MLP classifie en se basant sur la classe majoritaire. Cette mauvaise distribution est également visible dans la P-Value

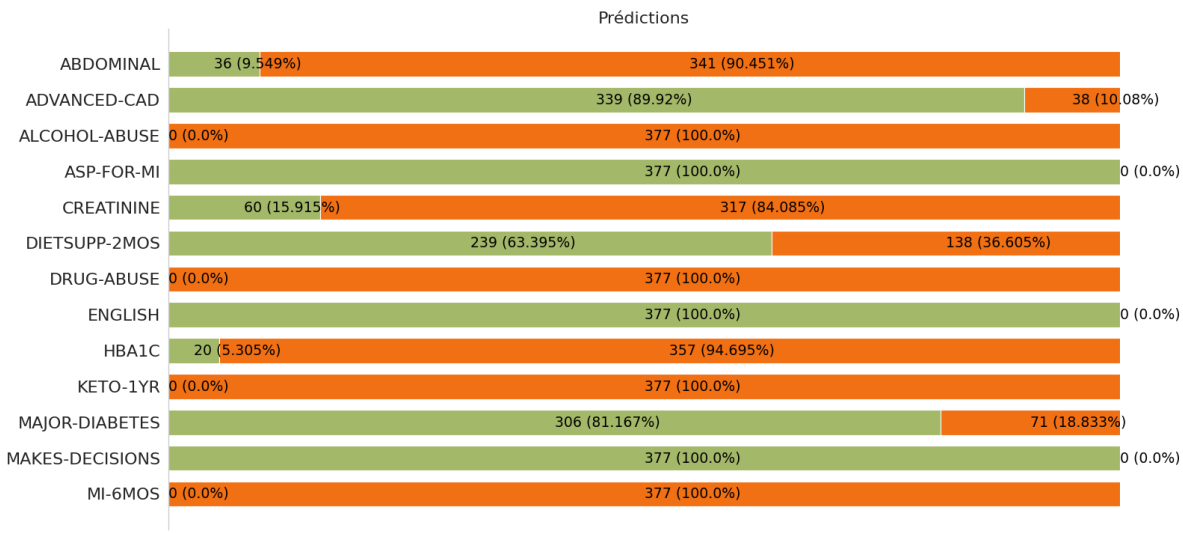

Figure 5.6: Distribution des prédictions - Random Forest. En vert, les prédictions positives. En orange, les prédictions négatives.

médiane qui est très largement en dessous du seuil critique, soit tout comme celle des classificateurs naïfs. Cela peut s'expliquer par le fait qu'en augmentant le nombre de poids, le modèle devient trop complexe.

Les paramètres utilisés pour obtenir ces résultats sont les suivants:

- Fonction de calcul des erreurs: FocalLoss
  - Alpha: 0.15
  - Gamma: 2.2
- Optimiseur: Adam
  - Learning rate: 0.0038
  - Weight decay: 0.0146
- Batch size: 128
- MLP:
  - Une couche d'entrée de taille 3072 $\rightarrow$ 9124
  - Une fonction d'activation RELU
  - Un dropout de 80%
  - Une couche de sortie de taille 9124 $\rightarrow$ 13

Critère	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.639	0.639	0.639	0.319	0.5	0.389
ADVANCED-CAD	0.586	0.586	0.586	0.293	0.5	0.369
ALCOHOL-ABUSE	0.965	0.965	0.965	0.482	0.5	0.491
ASP-FOR-MI	0.790	0.790	0.790	0.395	0.5	0.441
CREATININE	0.724	0.724	0.724	0.362	0.5	0.420
DIETSUPP-2MOS	0.501	0.501	0.501	0.250	0.5	0.333
DRUG-ABUSE	0.968	0.968	0.968	0.484	0.5	0.491
ENGLISH	0.843	0.843	0.843	0.421	0.5	0.457
HBA1C	0.413	0.413	0.413	0.206	0.5	0.292
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.501	0.501	0.501	0.250	0.5	0.333
MAKES-DECISIONS	0.968	0.968	0.968	0.484	0.5	0.491
MI-6MOS	0.904	0.904	0.904	0.452	0.5	0.474
Moyenne	0.657	0.852	0.742	0.354	0.538	0.418

Table 5.11: Résultats détaillés du MLP. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

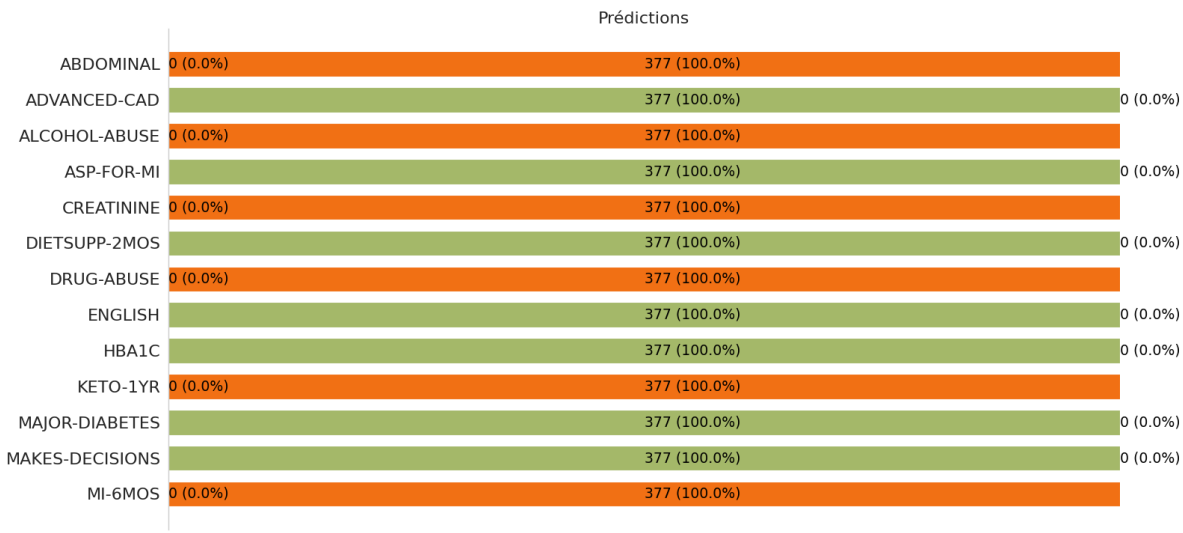

Figure 5.7: Distribution des prédictions - MLP. En vert, les prédictions positives. En orange, les prédictions négatives.

5.9 CNN

La capacité des CNN d'utiliser une multitude de filtres leur permettent de capturer des informations complexes dans les données. Avec une micro-F1 de 75.7% et une macro-F1 de 48.3%, la performance du CNN dépasse celle de tous les modèles testés jusqu'à maintenant. Comme il est possible de le voir dans les tableau 5.12, le CNN, tout comme le reste des modèles, a des performances très

élevées dans les critères très fortement inégalement distribués. Cela est explicable par le fait qu'il prédise la classe majoritaire. Néanmoins, le CNN arrive également à dépasser les 60% de F1 dans les critères correctement distribués, ce qui fait de lui le modèle le plus performant actuellement.

Ces performances accrues s'expliquent par la possibilité du CNN d'augmenter considérablement le nombre de poids à sa disposition pour capturer les informations pertinentes. De plus, la distribution des prédictions ressemble fortement à celle des données de test, comme il est possible de le voir à la figure 5.8, avec une médiane du test de KS de 0.042 et une P-Value médiane de 0.886. La distribution des prédictions et celle des données de test sont alors statistiquement égales. Ces résultats élevés combinés à cette distribution des prédictions induit que la grande partie des enregistrements a été correctement classifiée, qu'ils soient positifs ou négatifs.

Les paramètres utilisés pour obtenir ces résultats sont les suivants:

- Fonction de calcul des erreurs: FocalLoss
  - Alpha: 0.16
  - Gamma: 1.70
- Optimiseur: Adam
  - Learning rate: 0.00006
  - Weight decay: 0.0349
- Batch size: 128
- CNN :
  - Une couche de convolution de taille 512 avec une taille de filtre de 5 et un stride de 1
  - Une couche de normalisation
  - Un dropout de 20%
  - Une couche de max pooling adaptative de taille 512

Criteria	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.612	0.612	0.612	0.571	0.567	0.568
ADVANCED-CAD	0.623	0.623	0.623	0.650	0.556	0.507
ALCOHOL-ABUSE	0.965	0.965	0.965	0.482	0.5	0.491
ASP-FOR-MI	0.790	0.790	0.790	0.395	0.5	0.441
CREATININE	0.708	0.708	0.708	0.632	0.628	0.630
DIETSUPP-2MOS	0.501	0.501	0.501	0.250	0.5	0.333
DRUG-ABUSE	0.968	0.968	0.968	0.484	0.5	0.491
ENGLISH	0.843	0.843	0.843	0.421	0.5	0.457
HBA1C	0.591	0.591	0.591	0.583	0.585	0.583
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.596	0.596	0.596	0.626	0.596	0.570
MAKES-DECISIONS	0.968	0.968	0.968	0.484	0.5	0.491
MI-6MOS	0.904	0.904	0.904	0.452	0.5	0.474
Moyenne	0.685	0.847	0.757	0.440	0.553	0.483

Table 5.12: Résultats détaillés du CNN. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

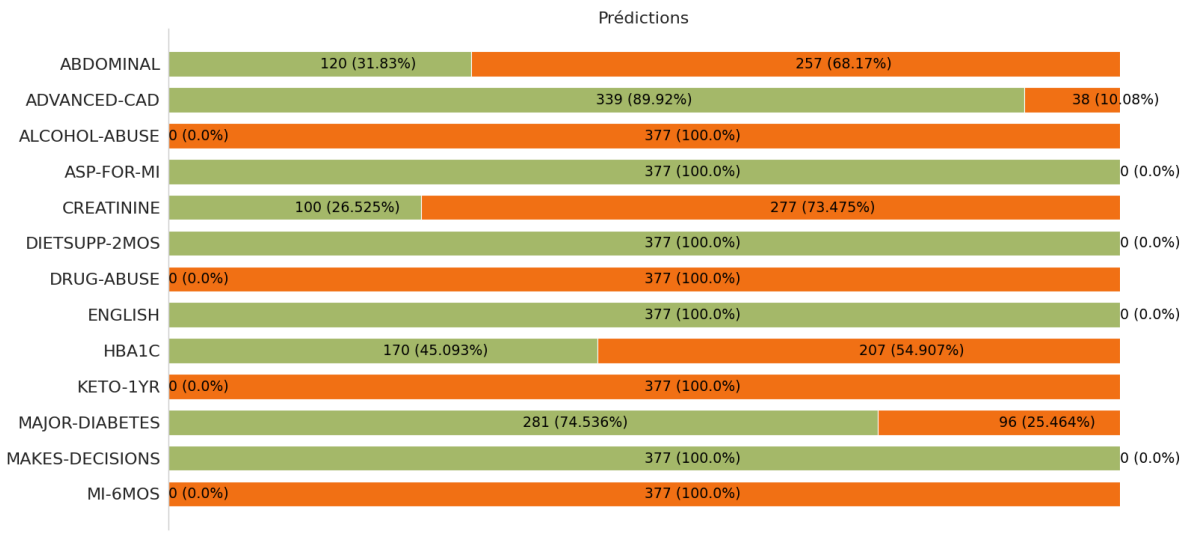

Figure 5.8: Distribution des prédictions - CNN. En vert, les prédictions positives. En orange, les prédictions négatives.

5.10 RNN

L'architecture RNN, spécialisée dans le traitement des séquences, est techniquement faite pour gérer les données textuelles comme celles du jeu de données N2C2 shared task (track 1). Comme expliqué dans la section 4.3.3, le RNN dispose de trois architectures: le RNN basique, le GRU et le LSTM. Chacune de ces architectures a ses avantages et ses inconvénients. Il est par conséquent nécessaire

de toutes les tester. C'est l'architecture GRU qui donne les meilleurs résultats avec une micro-F1 de 74.2% et une macro-F1 de 41.8% comme le démontre le tableau 5.13.

Architecture	Micro-F1	Macro-F1
RNN basique	0.7365	0.3742
GRU	0.7425	0.4186
LSTM	0.7422	0.4184

Table 5.13: Résultats des différentes architectures RNN

La différence de résultats entre les trois architectures est minime. Le RNN basique est le plus mauvais, que ce soit dans les métriques micro ou les métriques macro. L'architecture LSTM, la plus poussée, est deuxième, juste derrière l'architecture GRU. Cela est étonnant car c'est l'architecture la plus récente et la plus poussée. Elle permet, tout comme GRU, de pondérer les états cachés pour leur donner plus ou moins d'importance, tout en rajoutant un état cellule permettant de réduire le gradient vanishing et d'augmenter la longueur de la mémoire. L'architecture GRU est la plus performante dans le problème posé par le challenge N2C2 shared task (track 1), comme le démontre le tableau 5.13. Avec une micro-F1 de 74.25% et une macro-F1 de 41.86%, le modèle RNN (GRU) est l'un de moins performant parmi l'ensemble de ceux testés dans ce travail.

Tout comme les modèles ayant des résultats similaires, GRU classifie toutes les instances en se basant sur la classe majoritaire, comme le confirme la figure 5.9 et la P-Value médiane largement en dessous du seuil critique.

Les paramètres utilisés pour obtenir ces résultats sont les suivants :

- Fonction de calcul des erreurs: FocalLoss
  - Alpha: 0.27
  - Gamma: 4.4
- Optimiseur: Adam
  - Learning rate: 0.0121
  - Weight decay: 0.0110
- Batch size: 128
- GRU:
  - Taille de l'état caché: 256
  - Nombre de couches récurrentes: 1

– Dropout: 20%

Criteria	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.639	0.639	0.639	0.319	0.5	0.389
ADVANCED-CAD	0.586	0.586	0.586	0.293	0.5	0.369
ALCOHOL-ABUSE	0.965	0.965	0.965	0.482	0.5	0.491
ASP-FOR-MI	0.790	0.790	0.790	0.395	0.5	0.441
CREATININE	0.724	0.724	0.724	0.362	0.5	0.420
DIETSUPP-2MOS	0.501	0.501	0.501	0.250	0.5	0.333
DRUG-ABUSE	0.968	0.968	0.968	0.484	0.5	0.491
ENGLISH	0.843	0.843	0.843	0.421	0.5	0.457
HBA1C	0.413	0.413	0.413	0.206	0.5	0.292
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.501	0.501	0.501	0.250	0.5	0.333
MAKES-DECISIONS	0.968	0.968	0.968	0.484	0.5	0.491
MI-6MOS	0.904	0.904	0.904	0.452	0.5	0.474
Average	0.657	0.852	0.742	0.354	0.538	0.418

Table 5.14: Résultats détaillés de l'architecture GRU. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs)

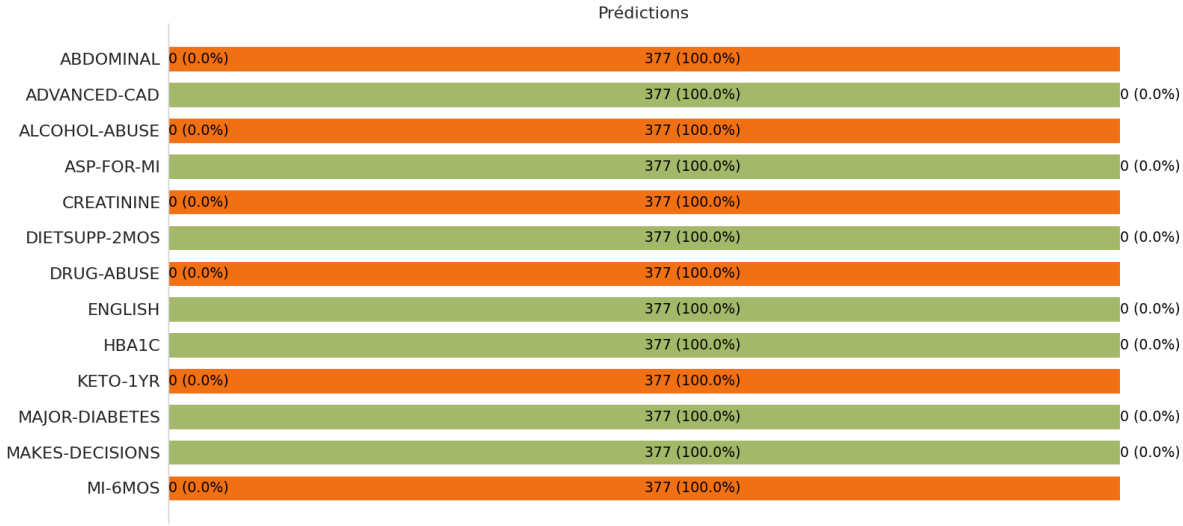

Figure 5.9: Distribution des prédictions - RNN (GRU). En vert, les prédictions positives. En orange, les prédictions négatives.

5.11 Classification end-to-end

La plupart des modèles d'embeddings proposent une solution de classification end-to-end. Le but est de donner le texte brut aux modèles d'embeddings et que ces derniers prédisent directement les classes associées sans avoir besoin de rajouter nous-même une couche de classification. Dans le cas du jeu de données N2C2 shared task (track 1), les enregistrements sont fournis au classificateur et ce dernier retourne une prédiction composée des valeurs associées à chaque critère. Tout comme la régression logistique, le classificateur retourne des valeurs entre 0 et 1 qu'il faut ensuite arrondir pour obtenir la prédiction réelle.

L'avantage de cette méthode de classification est qu'il est possible de mettre à jour les poids de la couche linéaire finale (régression logistique) tout comme ceux de la couche d'embeddings. Le modèle d'embeddings apprend alors des données et produit de meilleures représentations des mots.

Pour des raisons de temps et de performances machines, seul 3 modèles ont été testés avec l'architecture end-to-end :

1. BERT
2. Flair News
3. Flair Pubmed

L'entraînement de chacun de ces modèles a pris en moyenne 22 heures avec les paramètres suivants:

- Nombre d'epochs: 20
- Batch size: 16
- Learning rate: 0.01
- Patience: 2 (réduit le learning rate tout les 2 epochs n'améliorant pas les résultats)
- Facteur de réduction du learning rate: 0.5

BERT

Le modèle d'embeddings BERT a permis d'obtenir une micro-F1 de 72.6% et une macro-F1 de 37.4%. Comme visible dans le tableau 5.15, cette architecture arrive très bien à prédire les critères inégalement distribués mais a beaucoup de difficultés avec ceux qui le sont correctement. Ces résultats, tout comme certains modèles basiques, signifient que la classe majoritaire est utilisée pour la prédiction. Cela se confirme à la figure 5.10 qui montre la distribution des prédictions. BERT classifie néanmoins certains enregistrements positifs dans les critères ABDOMINAL et CREATININE tout comme d'autres sont classifiés négatifs dans les critères DIETSUPP-2MOS et MAJOR-DIABETES.

Avec une valeur médiane au test de KS de 0.156 et une P-Value médiane de 0.00019, BERT fait légèrement mieux que la plupart des modèles testés mais est bien loin du KNN et du CNN.

En disposant d'un jeu de données mieux distribué et comportant plus d'enregistrements, il est possible d'affirmer que BERT donnera de meilleurs résultats.

Critère	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.639	0.639	0.639	0.570	0.5010	0.396
ADVANCED-CAD	0.583	0.583	0.583	0.292	0.4974	0.368
ALCOHOL-ABUSE	0.965	0.965	0.965	0.482	0.5	0.491
ASP-FOR-MI	0.790	0.790	0.790	0.395	0.5	0.441
CREATININE	0.734	0.734	0.734	0.717	0.5286	0.483
DIETSUPP-2MOS	0.533	0.533	0.533	0.533	0.5327	0.530
DRUG-ABUSE	0.968	0.968	0.968	0.484	0.5	0.491
ENGLISH	0.843	0.843	0.843	0.421	0.5	0.457
HBA1C	0.586	0.586	0.586	0.293	0.5	0.369
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.488	0.488	0.488	0.450	0.4863	0.372
MAKES-DECISIONS	0.968	0.968	0.968	0.484	0.5	0.491
MI-6MOS	0.904	0.904	0.904	0.452	0.5	0.474
Moyenne	0.717	0.734	0.726	0.416	0.429	0.374

Table 5.15: Résultats détaillés du Transformers BERT. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

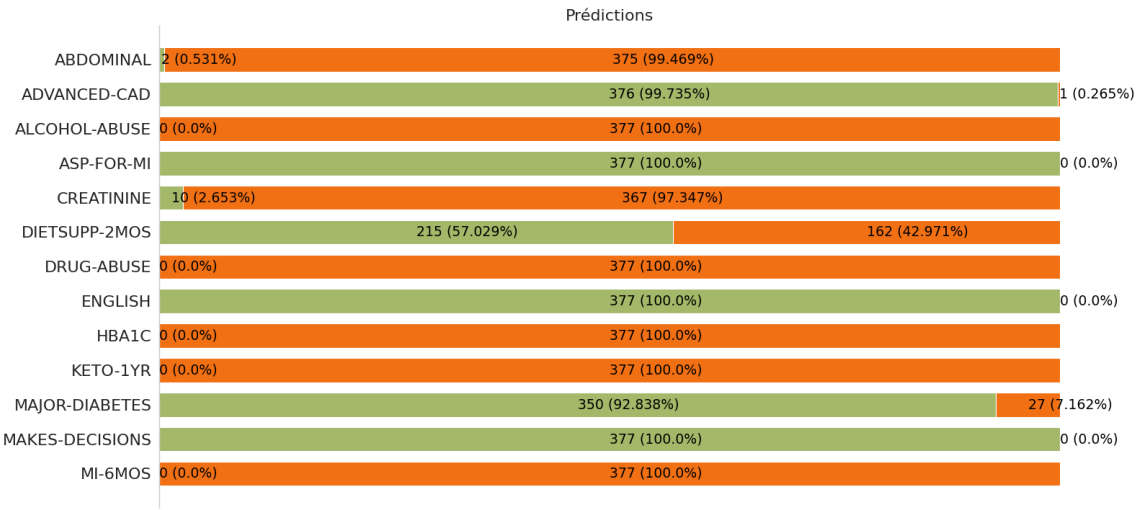

Figure 5.10: Distribution des prédictions - BERT. En vert, les prédictions positives. En orange, les prédictions négatives.

Flair News

Tout comme BERT, le modèle d'embeddings Flair News n'est pas très efficace en architecture end-

to-end avec les données du N2C2 shared task (track 1). Avec une micro-F1 de 73.3% et une macro-F1 de 37.1%, Flair News a des résultats similaires à ceux de BERT. Les résultats détaillés, visible au tableau 5.16 indiquent que le modèle a du mal à classifier les critères correctement distribués. Cela se confirme sur la figure 5.11 où la distribution des prédictions est totalement inégale et ne ressemble pas du tout à celle des données de test, avec une P-Value médiane bien en dessous du seuil critique de 5% (0.05). La distribution des prédictions est statistiquement différente de celle des données de test. Les seuls enregistrements classifiés différemment de la classe majoritaire sont ceux des critères DIETSUPP-2MOS et MAJOR-DIABETES. A contrario de BERT, Flair News n'a classifié aucun enregistrements positifs dans les CLASSES ABDOMINAL et CREATININE. De cet fait, il est impossible d'affirmer que le modèle serait plus efficace avec un meilleur jeu de données.

Critère	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.639	0.639	0.6396	0.319	0.5	0.389
ADVANCED-CAD	0.586	0.586	0.5861	0.293	0.5	0.369
ALCOHOL-ABUSE	0.965	0.965	0.9652	0.482	0.5	0.491
ASP-FOR-MI	0.790	0.790	0.7905	0.395	0.5	0.441
CREATININE	0.724	0.724	0.7244	0.362	0.5	0.420
DIETSUPP-2MOS	0.586	0.586	0.5861	0.597	0.585	0.572
DRUG-ABUSE	0.968	0.968	0.9687	0.484	0.5	0.491
ENGLISH	0.843	0.843	0.8430	0.421	0.5	0.457
HBA1C	0.586	0.586	0.5861	0.293	0.5	0.369
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.488	0.488	0.4886	0.358	0.486	0.336
MAKES-DECISIONS	0.968	0.968	0.9687	0.484	0.5	0.491
MI-6MOS	0.904	0.904	0.9041	0.452	0.5	0.474
Moyenne	0.717	0.750	0.7331	0.326	0.440	0.371

Table 5.16: Résultats détaillés du modèle Flair News. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

Flair Pubmed

Le dernier modèle d'embeddings testé en architecture end-to-end est le modèle Flair Pubmed. Ce dernier, entrainé avec des données médicales, permet d'obtenir une micro-F1 de 73.7% et une macro-F1 de 38.1%. Ces résultats sont légèrement moins bons que ceux de modèles d'embeddings BERT mais meilleurs que ceux de Flair News. Comme visible dans le tableau 5.17, Flair Pubmed souffre des mêmes problèmes que les autres modèles et a du mal à classifier les critères correctement distribués. Cette difficulté de classification est également visible à la figure 5.12 qui représente la distribution des prédictions, dont la P-Value médiane est largement en dessous du seuil critique. Néanmoins, Flair Pubmed classifie quelques enregistrements dans la classe minoritaire, tout comme

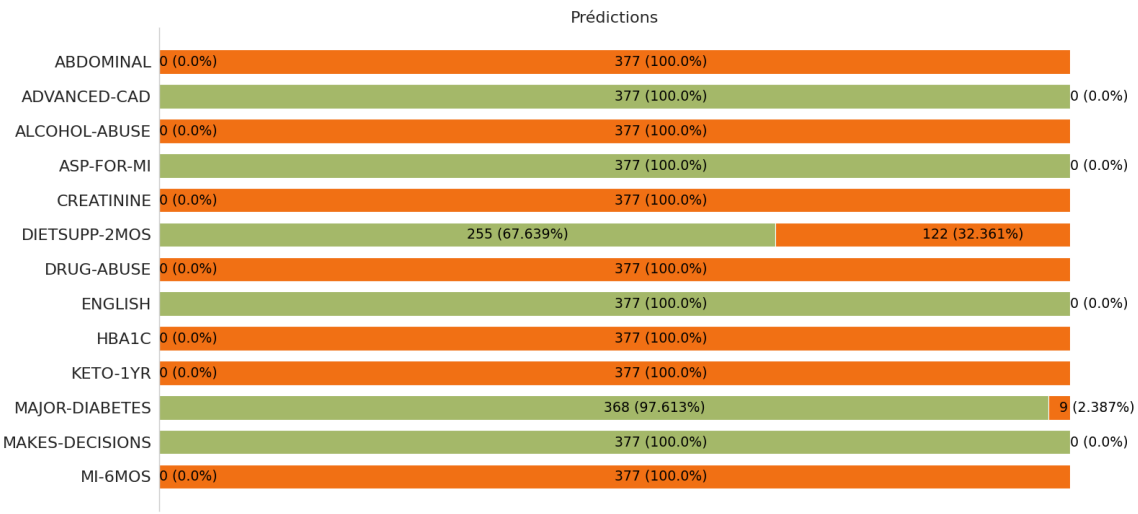

Figure 5.11: Distribution des prédictions - Flair News. En vert, les prédictions positives. En orange, les prédictions négatives.

BERT, ce qui permet d'imaginer qu'avec de meilleurs données et un temps d'entraînement plus important, le modèle pourrait augmenter ses résultats.

Critère	Micro			Macro		
	Précision	Rappel	F1	Précision	Rappel	F1
ABDOMINAL	0.644	0.644	0.644	0.821	0.507	0.405
ADVANCED-CAD	0.588	0.588	0.588	0.793	0.503	0.376
ALCOHOL-ABUSE	0.965	0.965	0.965	0.482	0.5	0.491
ASP-FOR-MI	0.790	0.790	0.790	0.395	0.5	0.441
CREATININE	0.734	0.734	0.734	0.865	0.519	0.459
DIETSUPP-2MOS	0.533	0.533	0.533	0.592	0.532	0.441
DRUG-ABUSE	0.968	0.968	0.968	0.484	0.5	0.491
ENGLISH	0.843	0.843	0.843	0.421	0.5	0.457
HBA1C	0.586	0.586	0.586	0.293	0.5	0.369
KETO-1YR	1	1	1	1	1	1
MAJOR-DIABETES	0.506	0.506	0.506	0.543	0.505	0.367
MAKES-DECISIONS	0.968	0.968	0.968	0.484	0.5	0.491
MI-6MOS	0.904	0.904	0.904	0.452	0.5	0.474
Moyenne	0.707	0.770	0.737	0.477	0.458	0.381

Table 5.17: Résultats détaillés du modèle Flair Pubmed. En couleur, les critères inégalement distribués (Vert: plus d'enregistrements positifs. Orange: plus d'enregistrements négatifs).

5.12 Discussion

Comme les résultats de chacun des modèles testés pendant ce travail se basent sur les données vectorisées par les modèles d'embeddings, ces derniers seront le premier élément de la discussion.

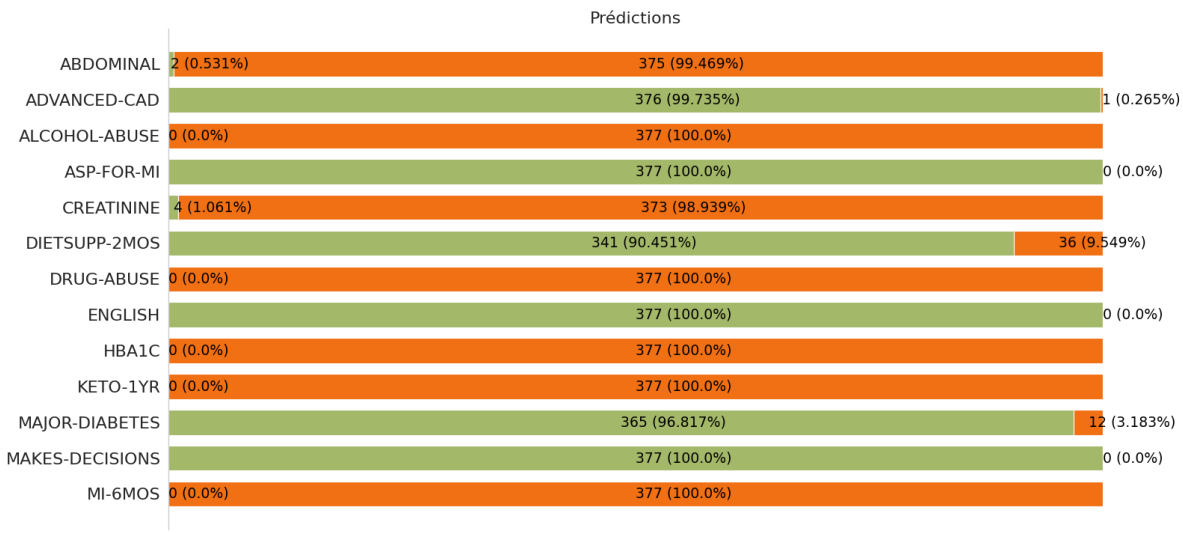

Figure 5.12: Distribution des prédictions - Flair Pubmed. En vert, les prédictions positives. En orange, les prédictions négatives.

De manière générale, l'ensemble des modèles d'embeddings présentent des résultats semblables. La différence de micro-F1 entre le meilleur et le moins bon est seulement de 1.5%, ce qui est insignifiant. La macro-F1 est beaucoup plus changeante avec des différences de l'ordre de 12.5%. La micro-F1 est généralement la métrique la plus utilisée lorsque les données sont inégalement distribuées. Néanmoins, dans le cas des données du N2C2 shared task (track 1), les données sont tellement inégalement distribuées que la micro-F1 donne toujours de bons résultats, même si le modèle prédit uniquement les classes majoritaires. Par conséquent, l'utilisation des macro métriques est ici plus adaptée. Cette différence de 12.5% entre le modèle ELMo, entraîné avec les données provenant du site Pubmed et le modèle Flair, également entraîné avec des données provenant du site Pubmed, est intéressant. Avec une taille d'embeddings plutôt proche (3072 pour ELMo et 4196 pour Flair) ainsi que des données d'entraînement similaires, la différence de résultats ne peut venir que de l'architecture des modèles ou de leur entraînement. ELmo est un modèle d'embedding basé sur les mots tandis que Flair est basé sur les caractères et n'a pas de connaissances explicites sur ce qu'est un mot [53]. Ces prédictions sont alors basées sur le premier et le dernier caractère de chaque mot. Cette différence peut engendrer un écart de résultats important entre les deux modèles. En effet, les données médicales étant très spécifiques, l'apprentissage du mot entier est indispensable. Le mot "Allogreffe" (greffe de cellule souche) est par exemple très différent du mot "Alopécie" (chute des cheveux) tout en commençant et finissant par les mêmes lettres. Il est par contre intéressant de voir que le modèle BERT, basé sur l'architecture Transformers, n'offre pas de meilleurs résultats qu'un modèle contextualisé simple. Bien que BERT ait été entraîné avec un nombre de mots considérable, ces derniers n'étaient pas du domaine médical. Cette différence d'entraînement peut expliquer les résultats plus faibles de BERT.

L'opération de pooling utilisée est également importante. Quel que soit le modèle, le changement de l'opération de pooling n'influence que très peu les résulats micro. A l'inverse, les macro métriques changent beaucoup plus. Pour l'ensemble des modèles, à l'exception d'ELMo Pubmed, l'utilisation de l'opération de pooling "max" permet d'obtenir les meilleurs résultats. Étonnament, c'est l'opération "mean" qui est la plus performante.

A partir de ces données, 5 modèles de Machine Learning basiques ont été testés: régression logistique, KNN, Radius Neighbors, arbre de décision et Random Forest. Comme vu dans la section 5.3, la régression logistique permet d'obtenir des résultats "acceptables" pour autant que les paramètres d'apprentissage du modèle soient bien choisis. La distribution des données confirme ces résultats avec une P-Value de 0.0001, supérieure à la P-Value de 1.168e-07 des algorithmes naïfs, visible dans le tableau 5.18. Les cinq critères très fortement inégalement distribués (ALCOHOL-ABUSE, DRUG-ABUSE, ENGLISH, MAKE-DECISIONS, Mi-6MOS) restent néanmoins prédits sur la base de la classe majoritaire.

Models	Micro-F1	Macro-F1	KS test median	P-Value median
Naive algorithms	-	0.44278^a	0.20954^b	1.168e-07^b
Logistic regression	73.8	**48.960**	0.15649	0.0001
KNN	72.4	43.8	0.095	0.064
Radius Neighbors	73.6	37.3	0.209	1.168-e07
Decision Tree	73.5	41.6	0.156	0.0001
Random Forest	73.6	41.7	0.132	0.002
MLP	74.2	41.8	0.209	1.168-e07
CNN	**75.7**	48.3	**0.042**	**0.886**
RNN (GRU)	74.2	41.8	0.209	1.168-e07
BERT Transformers	72.6	37.4	0.156	0.0001
Flair News Transformers	73.3	37.1	0.175	1.835-e05
Flair Pubmed Transformers	73.7	38.1	0.209	1.168-e07

Table 5.18: Résumés des résultats. En gras, les meilleurs résultats. ^{a} Résultats du Weighted Guess Classifier, ^{b} résultats du Majority Guess classifier.

D'une manière assez étonnante, l'algorithme KNN donne des résultats moins bons que ceux de la régression linéaire mais une P-Value plus élevée de 0.064 (tableau 5.18) indique une distribution des prédictions bien plus proche des réelles données que celle de la régression logistique. Tout comme cette dernière, il n'arrive pas à prédire correctement les enregistrements des cinq critères très inégalement distribués. Il est par contre intéressant de noter que l'algorithme KNN a prédit quelques instances du critère ASP-FOR-MI négatives, ce que n'a pas réussi à faire la régression logistique. La distribution n'est cependant pas améliorée dans l'ensemble des critères. KNN a, par exemple, prédit le critère ADVANCED-CAD plus souvent positif que la régression logistique, ce qui est moins bon. Une telle augmentation pourrait être due à la corrélation entre ce critère et les critères

ASP-FOR-MI et MI-6MOS. Cependant, comme vu précédemment, Mi-6MOS a la même distribution entre les deux modèles et celle du critère ASP-FOR-MI est très légèrement meilleure avec le modèle KNN. De manière générale, KNN arrive bien mieux à prédire des enregistrements négatifs que la régression logistique.

Basé sur le même principe que le KNN, l'algorithme Radius Neighbors à des résultats bien plus faibles avec une baisse de 6% dans la macro-F1. La distribution des prédictions est également moins bonne avec une P-Value de 1.168-e07, égale à celle des algorithmes naïfs (tableau 5.18). L'ensemble des critères, sauf DIETSUPP-2MOS, sont prédits en se basant sur la classe majoritaire. Une telle baisse dans les résultats s'explique assez facilement pour l'algorithme Radius Neighbors. En effet, la taille des vecteurs embeddés étant très grande (3072), la distance euclidienne entre ceux-ci est sûrement très élevée. Par conséquent, puisque l'algorithme se base sur un rayon pour sélectionner les instances de prédictions, il est fort probable que seulement une ou deux instances soient sélectionnées à chaque fois. Les prédictions ne sont alors pas faites par rapport à une majorité parmi les instances mais par rapport à une instance unique, les rendant moins précises.

L'arbre de décision n'a pas réussi à améliorer les résultats en produisant une macro-F1 de seulement 41%, moins bonne que celle des classificateurs naïfs (tableau 5.18). L'arbre de décision n'arrive pas à créer des conditions efficaces et prédit la classe majoritaire pour la plupart des critères, comme le démontre la P-Value de 0.0001. Cette dernière est égale à celle de la régression logistique alors que les métriques sont moins bonnes, démontrant que la plupart des critères classifiés autrement que par la classe majoritaire sont faux. Les deux seuls critères prédits différemment sont les critères ABDOMINAL et CREATININE. Étonnement, ces deux critères ont la même distribution, avec 71 instances prédites positives pour 306 prédites négatives. Les deux critères n'étant pas corrélés, l'explication de ces prédictions se trouve sûrement dans la manière dont l'algorithme a construit l'arbre.

Assez logiquement, le modèle Random Forest fait légèrement mieux que l'arbre de décision, avec une augmentation de la macro-F1 de 0.1% avec une macro-F1 de 41.7%. La distribution des prédictions est également plus ressemblante à celle des données de test (figure 5.6) que celle de l'arbre de décision, avec une P-Value de 0.002. De manière générale, le modèle Random Forest classifie un peu mieux les critères correctement distribués, tout en continuant à classifier les critères inégalement distribués en se basant sur la classe majoritaire.

L'utilisation d'un MLP améliore en général les résultats d'une simple régression. Ayant une multitude de vecteurs de poids, le MLP arrive techniquement mieux à capturer les informations pertinentes des données. Néanmoins, dans le cas du challenge N2C2 shared task (track 1), le MLP fait moins bien que les classificateurs naïfs avec une macro-F1 de 41.8% (table 5.18). Les données sont complètement prédites en se basant sur la classe majoritaire comme le démontre la P-Value de 1.168-e07, semblable à celle des classificateurs naïfs. Ces mauvais résultats peuvent s'expliquer par la com-

plexité des données. En effet, les vecteurs créés par les modèles d'embeddings sont très grands et composés uniquement de valeurs comprises entre -1 et 1. De ce fait, les poids devront réussir à capturer le plus d'informations sur des données très proches les unes des autres, rendant l'exercice d'autant plus complexe.

Le CNN arrive à résoudre cette problématique en proposant une macro-F1 de 48.3, juste derrière celle de la régression logistique qui est de 48.9% (tableau 5.18). Néanmoins, la micro-F1 du CNN est plus élevée de 1.9% que celle de la régression logistique. Ces bons résultats se voient également sur la distribution des prédictions qui ressemble le plus aux données de test parmi l'ensemble des modèles testés. De même, c'est le modèle qui a la P-Value la plus élevée avec 0.886. Ces très bons résultats peuvent s'expliquer par la possibilité du CNN d'utiliser une multitude de filtres sur les mêmes données (contrairement au MLP qui modifie les données après chaque perceptron via une opération linéaire), et donc de capturer l'ensemble des informations pertinentes sans altérer les données. Il est intéressant de noter que, tout comme les autres modèles, le CNN classifie les critères très inégalement distribués (ALCOHL-ABUSE, DRUG-ABUSE, ENGLISH, KETO-1YR, MAKES-DECISION, Mi-6MOS) en se basant sur la classe majoritaire. Le CNN est le seul modèle qui arrive à prédire les critères ABDOMINAL et HBA1C avec une marge d'erreur de seulement 5% dans la distribution des prédictions. D'une manière assez curieuse, le CNN arrive très bien à prédire les critères correctement distribués à l'exception du critère DIETSUPP-2MOS qui est pourtant celui qui est le mieux distribués (52% / 48 %). Dans ce cas là, le CNN a prédit la classe majoritaire. Aucune explication n'a été trouvée à ce comportement. De manière générale, augmenter le nombre de channels améliore également les prédictions jusqu'à la limite de 512 channels. Passé au dessus, les résultats chutent. L'augmentation du nombre de couches de convolution a également un effet négatif sur les résultats, de même que l'augmentation de la taille du filtre ainsi que son stride.

Le RNN, spécialisé dans le traitement des séquences, n'est malheureusement pas le modèle le plus performant, bien au contraire. Avec une faible macro-F1 de 41.8%, il se positionne au même niveau que le MLP (tableau 5.18). Quel que soit le type de RNN utilisé (RNN basique, GRU ou LSTM), les résultats sont très semblables. La distribution des prédictions est toujours la même avec l'ensemble des critères qui sont prédits sur la base de la classe majoritaire, d'où la P-Value de 1.168-e07, égale à celle des classificateurs naïfs. La seule exception est le critère HBA1C qui est tout le temps prédit positif alors que la classe majoritaire dans les données d'entraînement est la classe négative. La taille optimale de l'état caché est 256. Réduire ou augmenter cette valeur réduit considérablement les résultats, tout comme augmenter le nombre de couches du RNN.

En se basant sur l'état de l'art, l'utilisation d'un classificateur end-to-end basé sur l'architecture Transformers devrait permettre d'améliorer grandement les résultats. Malheureusement, avec le jeu de données N2C2 shared task (track 1), cela n'a pas été le cas. En se basant sur les métriques, le meilleur classificateur est Flair Pubmed avec une macro-F1 de 38.1%. Néanmoins, en regardant

la distribution des prédictions et la valeur de la P-Value (1.168-e07), il est assez facile de se rendre compte que Flair Pubmed obtient ces résultats en prédisant uniquement la classe majoritaire. Bien que la macro-F1 de BERT soit plus basse, sa distribution des prédictions est meilleure avec une P-Value de 0.0001. Bert a réussi a classifié quelques enregistrements des critères ABDOMINAL, CREATININE, DIETSUPP-2MOS et MAJOR-DIABETES dans la classe minoritaire. Ce modèle fait d'ailleurs mieux que le CNN pour le critère le mieux distribué DIETSUPP-2MOS. En utilisant un meilleur jeu de données, correctement distribué et contenant bien plus d'enregistrements, il est fort probable que l'utilisation d'un classificateur BERT surpasse les modèles CNN et KNN. Il est également fort probable que l'utilisation d'un modèle basé sur l'architecture Transformers et entraîné avec des données médicales augmente également les résultats.

5.12.1 Optimisation des paramètres

L'optimisation des différents modèles, que ce soit au niveau des paramètres, des fonctions de calculs des erreurs et des optimiseurs utilisées mettent aussi en avant un élément. Les paramètres utilisés et la fonction de calcul des erreurs ont un impact direct sur les résultats. Dans le cas du jeu de données 2018 N2C2 shared task (track 1), la fonction de calcul des erreurs FocalLoss [55] est à chaque fois celle permettant d'obtenir les meilleurs résultats. Cela s'explique assez facilement par le fait qu'elle est spécialisée dans le traitement des données dont la distribution est inégale. L'optimiseur de son coté à moins d'importance, ce qui n'est pas le cas de son paramétrage. Le choix des paramètres d'apprentissages ont une grande influence sur les résultats.

5.12.2 Impact des longueurs de texte

En se basant sur les résultats du meilleur modèle, le CNN, les résultats par longueur de texte ont également été calculés. Ces derniers démontrent que la longueur de texte a une certaine influence sur les résultats, avec une différence de F1 de 3.3% entre les textes de taille $(128, 403]$ tokens et ceux de taille $(844, 1188]$ tokens, comme cela est visible dans le graphique 5.13. Il est néanmoins important de ne pas utiliser des textes trop grands, comme le démontre la F1 plus faible de 2.3% obtenue avec des textes plus grands que 1188 tokens. De même, si le but est de maximiser la précision ou le rappel, les longueurs de texte idéales ne sont pas les mêmes, comme le démontre le graphique 5.13.

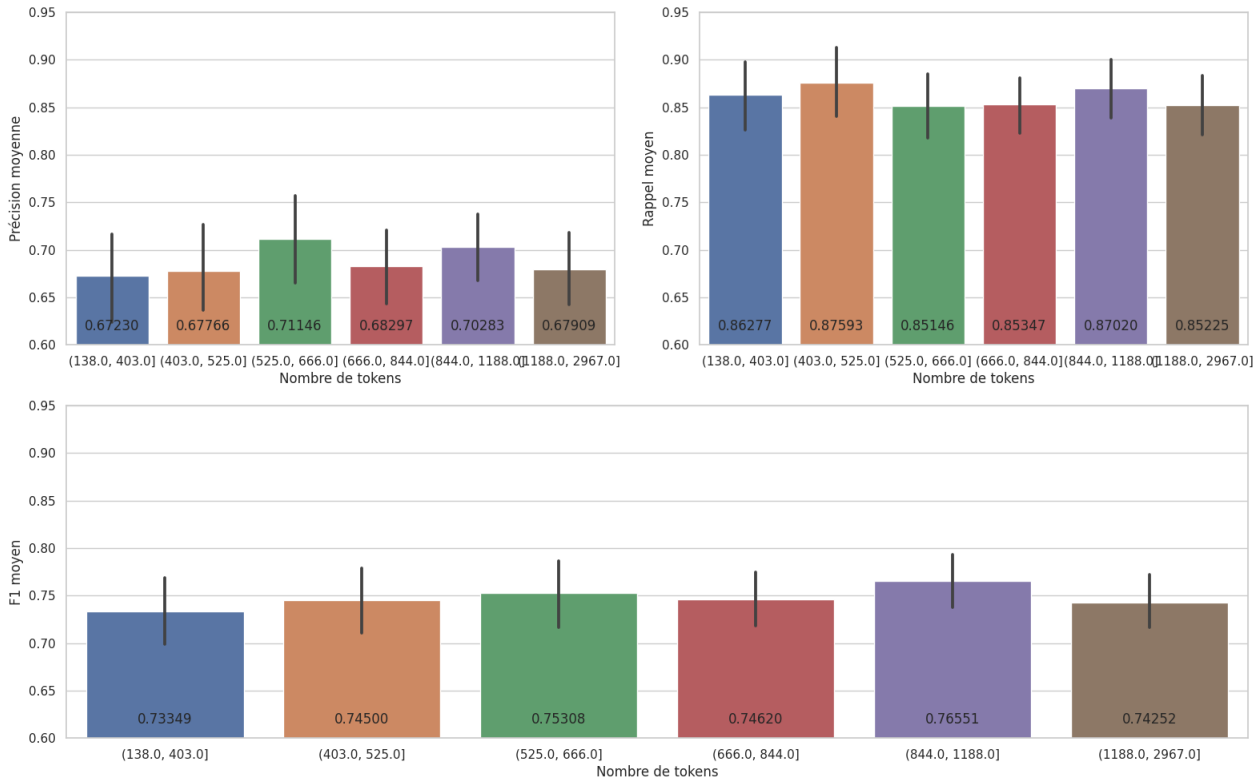

Figure 5.13: Résultats par longueurs de texte pour le modèle CNN

6 Conclusion

La réussite d'un essai clinique dépend de beaucoup d'éléments. L'un d'eux, la sélection des patients éligibles, appelée "sélection de cohorte", est d'autant plus importante qu'elle est déterminante dans le calcul de l'efficacité du traitement testé. Actuellement, cette tâche est faite par les chercheurs et demande énormément de temps et d'efforts. L'utilisation de modèles basés sur des règles écrites à la main permettent d'automatiser ce processus avec des performances proches de 100%. Ces règles sont néanmoins propres à chaque problème et ne peuvent donc pas être généralisées à un ensembles d'essais cliniques. C'est à cette tâche que ce travail a tenté de trouver une réponse en étudiant quel modèle de Machine Learning ou de Deep Learning permet la meilleure classification des patients, sans nécessiter la création de règles propre à chaque essai clinique. Cette tâche est très complexe compte tenu du nombre de données médicales très limitées disponibles pour l'entraînement. De plus, la plupart de celles-ci sont uniquement disponibles au format textuel et nécessite alors d'être transformées préalablement en vecteur.

Dans le cas des données du challenge N2C2 shared task (track 1), le meilleur modèle d'embeddings est le modèle ELMo, entraîné avec des données provenant du hub médical Pubmed. Il est ensuite possible d'utiliser ces données embeddées pour entraîner un classificateur. Ce dernier est chargé de prédire si un enregistrement correspond ou non à chaque critère d'inclusion. Le modèle CNN a permis d'obtenir les meilleurs résultats avec les données limitées à disposition. Etonnament, le modèle KNN se place juste derrière le CNN avec de très bons résultats également. Les résultats obtenus dans ce travail n'ont pas permis de devancer ceux présentés dans l'état de l'art [10]. Cependant, ils ont mis en avant le fait que l'ensemble des modèles de Deep-Learning ne sont pas forcément plus performants que des modèles basiques de Machine-Learning. D'autant plus intéressant, est le fait que l'augmentation de la complexité des modèles (plus de couches, vecteur de poids plus grands, etc) réduit les résultats par rapport à un modèle simple. Il est également nécessaire d'utiliser des paramètres d'entraînement adaptés aux données, comme le démontre le fait que l'ensemble des meilleurs résultats de ce travail ont été obtenus avec la fonction de calcul des erreurs FocalLoss. En comparant les résultats par longueur de texte, il a également été mis en évidence que la longueur du texte utilisé influence les résultats. De plus, ce travail a également permis de démontrer l'importance de la comparaison des résultats des modèles avec un classificateur naïf afin de s'assurer que ces derniers apprennent bien des données.

Bien évidemment, les résultats obtenus se basent sur les données limitées du jeu de données N2C2 shared task (track 1) et peuvent être sans doute améliorés en travaillant préalablement sur la récolte d'un meilleur jeu de données. Ce travail illustre bien l'importance d'avoir un jeu de données complet et correctement distribué.

L'utilisation d'un système informatisé permet de créer une liste classifiée des patients les plus aptes à participer à l'essai clinique, réduisant grandement la charge de travail des chercheurs. Ces derniers n'ont pas besoin de vérifier l'ensemble des patients mais seulement ceux qui ont la plus grande probabilité d'être éligibles.

Pour une future étude, il serait intéressant d'explorer une solution pour augmenter le nombre de données d'entraînement disponibles dans le domaine médical. En utilisant un modèle capturant la similarité entre plusieurs textes, il serait possible d'ajouter des données provenant d'une étude analogue sans avoir besoin de les annoter préalablement à la main, ce qui nécessite des connaissances médicales poussées. Il serait également judicieux de tester si l'efficacité de modèles de sélection de cohorte peut être augmentée en utilisant des données cliniques simples (âge, sexe, bmi, taille, etc) plutôt que des données complexes formulées textuellement.

Bibliographie

[1] Moore's law, . URL https://en.wikipedia.org/w/index.php?title=Moore%27s_law&oldid=967384337. Page Version ID: 967384337.

[2] D. Michie. "memo" functions and machine learning. *Nature*, 218:19–22, 1968.

[3] Y. Zeng, E. Lu, Y. Sun, and R. Tian. Responsible Facial Recognition and Beyond, Sept. 2019. URL https://deepai.org/publication/responsible-facial-recognition-and-beyond.

[4] H.-J. Kong. Managing unstructured big data in healthcare system. *Healthcare informatics research*, 25(1):1—2, January 2019. ISSN 2093-3681. doi: 10.4258/hir.2019.25.1.1. URL https://europepmc.org/articles/PMC6372467.

[5] A. SIDDHARTHAN. Christopher d. manning and hinrich schutze. foundations of statistical natural language processing. mit press, 2000. isbn 0-262-13360-1. 620 pp. $64.95/£44.95 (cloth). *Natural Language Engineering*, 8(1):91–92, 2002. doi: 10.1017/S1351324902212851.

[6] G. G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51–89, 2003. doi: 10.1002/aris.1440370103. URL https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440370103.

[7] T. P. S. University. 7.1 - defining the study cohort | STAT 509. URL https://newonlinecourses.science.psu.edu/stat509/node/60/.

[8] I. Segura-Bedmar and P. Raez. Cohort selection for clinical trials using deep learning models. *Journal of the American Medical Informatics Association*, 26(11):1181–1188, 09 2019. ISSN 1527-974X. doi: 10.1093/jamia/ocz139. URL https://doi.org/10.1093/jamia/ocz139.

[9] J. C. Kirby, P. Speltz, L. V. Rasmussen, M. Basford, O. Gottesman, P. L. Peissig, J. A. Pacheco, G. Tromp, J. Pathak, D. S. Carrell, S. B. Ellis, T. Lingren, W. K. Thompson, G. Savova, J. Haines, D. M. Roden, P. A. Harris, and J. C. Denny. PheKB: a catalog and workflow for creating electronic phenotype algorithms for transportability. *Journal of the American Medical Informatics Association*, 23(6):1046–1052, 03 2016. ISSN 1067-5027. doi: 10.1093/jamia/ocv202. URL https://doi.org/10.1093/jamia/ocv202.

[10] I. Segura-Bedmar, C. Colón-Ruíz, M. Ángel Tejedor-Alonso, and M. Moro-Moro. Predicting of anaphylaxis in big data emr by exploring machine learning approaches. *Journal of Biomedical Informatics*, 87:50 – 59, 2018. ISSN 1532-0464. doi: https://doi.org/10.1016/j.jbi.2018.09.012. URL http://www.sciencedirect.com/science/article/pii/S1532046418301874.

[11] N2c2: National NLP clinical challenges, . URL https://n2c2.dbmi.hms.harvard.edu/track1.

[12] C.-J. Chen, N. Warikoo, Y.-C. Chang, J.-H. Chen, and W.-L. Hsu. Medical knowledge infused convolutional neural networks for cohort selection in clinical trials. *Journal of the American Medical Informatics Association*, 26(11):1227–1236, 08 2019. ISSN 1527-974X. doi: 10.1093/jamia/ocz128. URL https://doi.org/10.1093/jamia/ocz128.

[13] I. Spasic, D. Krzeminski, P. Corcoran, and A. Balinsky. Cohort selection for clinical trials from longitudinal patient records: Text mining approach. *JMIR Med Inform*, 7(4):e15980, Oct 2019. ISSN 2291-9694. doi: 10.2196/15980. URL http://medinform.jmir.org/2019/4/e15980/.

[14] A. Sethy and B. Ramabhadran. Bag-of-word normalized n-gram models. In *INTERSPEECH*, 2008.

[15] B. Nowok, G. Raab, and C. Dibben. synthpop : Bespoke creation of synthetic data in r. *Journal of Statistical Software*, 74, 10 2016. doi: 10.18637/jss.v074.i11.

[16] M. Hittmeir, R. Mayer, and A. Ekelhart. A baseline for attribute disclosure risk in synthetic data. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, CODASPY '20, page 133–143, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371070. doi: 10.1145/3374664.3375722. URL https://doi.org/10.1145/3374664.3375722.

[17] N. Patki, R. Wedge, and K. Veeramachaneni. The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410. IEEE. ISBN 978-1-5090-5206-6. doi: 10.1109/DSAA.2016.49. URL http://ieeexplore.ieee.org/document/7796926/.

[18] D. D. Palmer. A trainable rule-based algorithm for word segmentation. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, ACL '98/EACL '98, page 321–328, USA, 1997. Association for Computational Linguistics. doi: 10.3115/976909.979658. URL https://doi.org/10.3115/976909.979658.

[19] Z. Gero and J. Ho. Pmcvec: Distributed phrase representation for biomedical text processing. *Journal of Biomedical Informatics: X*, 3:100047, 2019. ISSN 2590-177X. doi: https://doi.org/10.1016/j.yjbinx.2019.100047. URL http://www.sciencedirect.com/science/article/pii/S2590177X19300460.

[20] A. Ferré, L. Deléger, P. Zweigenbaum, and C. Nédellec. Combining rule-based and embedding-based approaches to normalize textual entities with an ontology. In N. C. C. chair), K. Choukri, C. Cieri, T. Declerck, S. Goggi, K. Hasida, H. Isahara, B. Maegaard, J. Mariani, H. Mazo,

A. Moreno, J. Odijk, S. Piperidis, and T. Tokunaga, editors, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 7-12, 2018 2018. European Language Resources Association (ELRA). ISBN 979-10-95546-00-9.

[21] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL http://aclweb.org/anthology/D14-1162.

[22] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. F. Liu, M. Peters, M. Schmitz, and L. Zettlemoyer. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2501. URL https://www.aclweb.org/anthology/W18-2501.

[23] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[25] i2b2: Informatics for integrating biology & the bedside, . URL https://www.i2b2.org/.

[26] A. Stubbs, M. Filannino, E. Soysal, S. Henry, and Uzuner. Cohort selection for clinical trials: n2c2 2018 shared task track 1. *Journal of the American Medical Informatics Association*, 26(11): 1163–1171, 09 2019. ISSN 1527-974X. doi: 10.1093/jamia/ocz163. URL https://doi.org/10.1093/jamia/ocz163.

[27] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer. Knn model-based approach in classification. In R. Meersman, Z. Tari, and D. C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 986–996, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39964-3.

[28] L. Rokach and O. Maimon. *Decision Trees*, pages 165–192. Springer US, Boston, MA, 2005. ISBN 978-0-387-25465-4. doi: 10.1007/0-387-25465-X_9. URL https://doi.org/10.1007/0-387-25465-X_9.

[29] Decision tree learning, . URL https://en.wikipedia.org/w/index.php?title=Decision_tree_learning&oldid=971585991. Page Version ID: 971585991.

[30] J. Ali, R. Khan, N. Ahmad, and I. Maqsood. Random forests and decision trees. *International Journal of Computer Science Issues(IJCSI)*, 9, 09 2012.

[31] G. A. F. Seber and A. J. Lee. *Linear Regression Analysis*. URL https://cds.cern.ch/record/1438185. ISBN: 9781118274422 Publisher: John Wiley & Sons.

[32] Q. Abdulqader. Applying the binary logistic regression analysis on the medical data. *Science Journal of University of Zakho*, 5(4):330–334, Dec. 2017. doi: 10.25271/2017.5.4.388. URL https://sjuoz.uoz.edu.krd/index.php/sjuoz/article/view/442.

[33] Ronny Restrepo. Ronny restrepo. URL http://ronny.rest/blog/post_2017_08_16_tanh/.

[34] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis. Multilayer perceptron and neural networks. *WSEAS Trans. Cir. and Sys.*, 8(7):579–588, July 2009. ISSN 1109-2734.

[35] K. Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119 – 130, 1988. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(88)90014-7. URL http://www.sciencedirect.com/science/article/pii/0893608088900147.

[36] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116, 1998. doi: 10.1142/S0218488598000094. URL https://doi.org/10.1142/S0218488598000094.

[37] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012. URL https://www.aclweb.org/anthology/W14-4012.

[38] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[39] shanelynn. Get busy with word embeddings – an introduction. URL https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/. Library Catalog: www.shanelynn.ie.

[40] S. Ramamoorthy. Chatbots with seq2seq. URL http://suriyadeepan.github.io/2016-06-28-easy-seq2seq//. Library Catalog: complx.me.

[41] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In Y. Bengio and Y. LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL http://arxiv.org/abs/1301.3781.

[42] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://www.aclweb.org/anthology/N18-1202.

[43] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing Management*, 45(4):427 – 437, 2009. ISSN 0306-4573. doi: https://doi.org/10.1016/j.ipm.2009.03.002. URL http://www.sciencedirect.com/science/article/pii/S0306457309000259.

[44] G. Shaheen. Is your classification model making lucky guesses? URL https://gallery.azure.ai/Experiment/Is-your-classification-model-making-lucky-guesses-1. Library Catalog: gallery.azure.ai.

[45] S. Gauher. shaheeng/ClassificationModelEvaluation. URL https://github.com/shaheeng/ClassificationModelEvaluation. Library Catalog: github.com.

[46] F. J. M. Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951. doi: 10.1080/01621459.1951.10500769. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1951.10500769.

[47] V. W. Berger and Y. Zhou. Kolmogorov–smirnov test: Overview. In *Wiley StatsRef: Statistics Reference Online*. American Cancer Society. ISBN 978-1-118-44511-2. doi: 10.1002/9781118445112.stat06558. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat06558. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118445112.stat06558.

[48] Infimum and supremum, . URL https://en.wikipedia.org/w/index.php?title=Infimum_and_supremum&oldid=972693886. Page Version ID: 972693886.

[49] SciPy.org — SciPy.org, . URL https://www.scipy.org/.

[50] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330701. URL https://doi.org/10.1145/3292500.3330701.

[51] A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59. Association for Computational Linguistics, . doi: 10.18653/v1/N19-4010. URL https://www.aclweb.org/anthology/N19-4010.

[52] A. Akbik, T. Bergmann, and R. Vollgraf. Pooled contextualized embeddings for named entity recognition. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 724–728. Association for Computational Linguistics, . doi: 10.18653/v1/N19-1078. URL https://www.aclweb.org/anthology/N19-1078.

[53] A. Akbik, D. Blythe, and R. Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649. Association for Computational Linguistics, . URL https://www.aclweb.org/anthology/C18-1139.

[54] PubMed, . URL https://pubmed.ncbi.nlm.nih.gov/.

[55] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017.