الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

كلية علوم الطبيعة والحياة
Faculté des Sciences de la Nature et de la Vie

جامعة الإخوة منتوري قسنطينة 1
Université Frères Mentouri Constantine 1

Département de Biologie Appliquée

قسم البيولوجيا التطبيقية

Mémoire en vue de l'obtention du Diplôme de Master en :
Bioinformatique

THÈME

Nouvelle approche de prédiction des classes protéiques issues d'un séquençage NGS par Deep Learning

Présenté par :

ALIOUANE Salah Eddine
BENDAHMANE Abdelhafedh

Soutenu le : 17 - 09 - 2020

Devant le jury :

Président du jury : Dr. MENACER Rafik

Co-Encadreur : Pr. HAMIDECHI M. Abdelhafid

Co-Encadreur : Dr. CHEHILI Hamza

Examinateur : Dr. GHERBOUDJ Amira

Année universitaire 2019-2020

REMERCIEMENTS

Nous exprimons nos profonds remerciements à nos encadreurs et directeurs de mémoire le professeur Mohamed Abdelhafid HAMIDECHI et le docteur Hamza CHEHILI.

Nous remercions le professeur Mohamed Abdelhafid HAMIDECHI, notre premier directeur de mémoire, de nous avoir accepté au sein de son équipe de recherche, de nous avoir encadré et orienté dans les travaux de recherche ou dans la vie en général.

Nous tenons à remercier le professeur Mohamed Abdelhafid HAMIDECHI également pour sa gentillesse, sa disponibilité, sa patience, ses judicieux conseils, ses orientations, son aide à écrire ce mémoire et son soutien pour mener à terme notre master.

Nous sommes profondément reconnaissant au docteur Hamza CHEHILI, notre deuxième directeur de mémoire, de nous avoir encadré durant notre master et nous avoir permis de réaliser ce travail.

Nous remercions également le docteur Hamza CHEHILI pour sa présence, sa disponibilité, ses réflexions pertinentes, ses orientations, ses précieux conseils, sa patiente, son implication et pour son aide à écrire ce mémoire.

Les mots sont insuffisants pour décrire notre reconnaissance et notre gratitude au professeur Mohamed Abdelhafid HAMIDECHI et au docteur Hamza CHEHILI. Merci encore.

Nos vifs remerciements vont également aux membres du jury Dr. MENACER R. et Dr. GHERBOUDJ A. pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'évaluer notre mémoire de fin d'études et de l'enrichir par leur contribution.

Nous remercions également tous nos enseignants qui ont assuré notre formation tout au long de ces années. Nous soulignons que leurs efforts nous ont été bénéfiques pour effectuer ce travail.

Nous voudrions que ce petit bout de chemin que nous avons accompli ensemble en étant étudiants de master donne ses fruits et que nous puissions élaborer et accomplir notre chemin par un projet de recherche de Doctorat.

Un grand merci à nos amis pour leur sympathie et leurs soutiens.

Nous remercions nos parents, nos frères et sœurs, sans eux, ce travail n'aurait jamais pu être réalisé. Un énorme merci à nos familles pour leur soutien indéfectible et leurs encouragements tout au long de l'élaboration de notre travail.

RÉSUMÉ

Le but de ce travail est de développer une approche DeepProt visant à simuler les phases du processus de la post-génomique à partir d'une séquence nucléique d'un séquençage NGS. Cette approche est basée sur deux axes de l'intelligence artificielle (IA) : le traitement automatique du langage naturel (NLP) et l'apprentissage profond (DL). Avec un apprentissage sur 43 familles et un taux de précision de 88% obtenus dans les tests effectués, les résultats ont montré l'efficacité de cette approche, notamment la phase de prédiction basée sur le NLP et le DL. Ces deux outils, combinés, ont donné un modèle d'une grande capacité à extraire les connaissances des données protéiques afin de prédire et classer celles-ci. Le modèle proposé apprend grâce à un entrainement intensif par exploitation des séquences protéiques. Ce travail a permis de mettre en évidence l'apport de cette approche à améliorer la précision de la classification des protéines.

Mots clés : Assemblage ; Classification ; Prédiction ; Intelligence Artificiel ; NLP ; NGS.

ABSTRACT

The Purpose of this work is to develop DeepProt approach aimed at simulating the phases of the post-genomics process from a nucleic acid sequence of an NGS sequencing. This approach is based on two axes of artificial intelligence (AI): natural language processing (NLP) and deep learning (DL). With a training on 43 families and an accuracy rate of 88% obtained in the tests carried out, the results showed the effectiveness of this approach, in particular the prediction phase based on the NLP and the DL. These two tools, combined, gave a model with a great capacity to extract knowledge from protein data in order to predict and classify them. The proposed model learns through intensive training by exploiting protein sequences. This work made it possible to highlight the contribution of this approach for improving the precision of protein classification.

Key words : Assembly ; Classification ; Prediction ; Artificial intelligence ; NLP ; NGS.

الملخص

الهدف من هذا العمل هو تطوير منهج DeepProt يهدف إلى محاكاة مراحل عملية ما بعد الجينوم من تسلسل الحمض النووي لتسلسل الجيل القادم يعتمد هذا المنهج على محورين للذكاء الاصطناعي : المعالجة التلقائية للغة الطبيعية والتعلم العميق. التعلم العميق من خلال التدريب على 43 عائلة ومعدل دقة 88٪ تم الحصول عليه في الاختبارات التي أجريت، حيث أظهرت النتائج فعالية هذا المنهج، لا سيما مرحلة التنبؤ القائمة على البرمجة اللغوية العصبية والتعلم العميق بحيث أعطت هاتان الأداتان مجتمعتان نموذجًا ذا قدرة كبيرة على استخراج المعرفة من بيانات البروتين من أجل التنبؤ بها وتصنيفها. يتعلم النموذج المقترح من خلال التدريب المكثف من خلال استغلال تسلسل البروتين .أتاح هذا العمل تسليط الضوء على مساهمة هذا المنهج في تحسين دقة تصنيف البروتين.

الكلمات المفتاحية : التجميع ؛ التصنيف ؛ التنبؤ ؛ الذكاء الاصطناعي ؛ البرمجة اللغوية العصبية ؛ تسلسل الجيل القادم.

LISTES DES FIGURES

LISTE DES TABLEAUX

ACRONYMES

- CNN : Convolutional Neural Network (Réseau neuronal convolutif)
- CSV : Comma-Separated Values
- DL : Deep Learning (Apprentissage profond)
- HPC : High-Performance Computing, qui veut dire calcul intensif
- IA : Intelligence artificielle, (Artificial Intelligence)
- ML : Machine Learning (Apprentissage Automatique)
- NGS : Séquençage nouvelle génération (Next Generation Sequencing)
- NLP : Natural Language Processing (Traitement automatique du langage naturel)
- OLC : Overlap-Layout-Consensus (Chevauchement-mise en page-consensus)
- PCR : Polymerase Chain Reaction (réaction de polymérisation en chaîne)
- SCS : Shortest Common Superstring

TABLE DE MATIÈRES

TABLE DES MATIÈRES

Introduction

INTRODUCTION

Depuis l'avènement des techniques de séquençage, l'identification des séquences d'acide nucléique est devenue un outil essentiel dans le domaine de la biologie et plus précisément, depuis l'apparition des technologies de nouvelles générations de séquençage (NGS). Ces techniques capables de séquencer tout un génome en très peu de temps, ont fini par générer de grandes masses de données. La bioinformatique est au cœur de ces données : c'est le domaine offrant les différents outils et méthodes afin de traiter et d'interpréter ces données. Afin de modéliser les problèmes biologiques, la bioinformatique se base sur les mathématiques, les statistiques et l'informatique.

L'un des problèmes traités par la bioinformatique est la classification des protéines, parfois séquencées *in vitro*, mais plus souvent traduites à partir des gènes nouvellement séquencés. Au début, la solution était simple, une simple comparaison avec les bases de données protéiques suffisait à déterminer la classe protéique par homologie. Mais vu la hausse de données, et l'acquisition consécutive de nouvelles molécules biologiques de différents organismes, les méthodes heuristiques traditionnelles tel que blast ont perdu en matière d'efficacité. Donc une évolution des méthodes de prédiction s'impose. Le but est d'initier de nouvelles approches visant à augmenter la rapidité des traitements et à améliorer la précision des prédictions.

En parallèle, l'intelligence artificielle, un domaine établie dans les années 50, mais qui connait une nouvelle ère ces dernières années. Ce domaine donne lieu à de nombreux espoirs dans différents domaines, grâce à de nouveaux algorithmes, ainsi qu'à la multiplication des jeux de données et le décuplement des puissances de calcul. Les méthodes de l'intelligence artificielles telles que l'apprentissage automatique ou l'apprentissage profond, ont apporté de nouvelles solutions à différents problèmes de la biologie (Deitel et Deitel, 2018; Lee *et al.*, 2017; Paladino *et al.*, 2017; Wang et Wong, 2020; Zhavoronkov, 2018).

Notre contribution consiste à proposer une nouvelle approche, dont l'intelligence artificielle est le pilier. Elle consiste à utiliser le traitement automatique du langage naturel avec l'apprentissage profond dans le but de classifier les protéines traduites à partir de gènes séquencées.

Le manuscrit est organisé en quatre chapitres. Le premier concerne :

- Le séquençage avec ses différentes étapes.
- Les différentes technologies de séquençage à haut débit

- L’assemblage et la traduction
- Les différentes classifications des protéines
- Les méthodes de prédiction *in silico.*

Le deuxième chapitre aborde les différents outils de l’intelligence artificielle nécessaires à la compréhension de l’approche proposée, en commençant par les concepts généraux de l’apprentissage automatique, suivie par l’apprentissage profond et le traitement automatique du langage naturel.

Le troisième chapitre représente la partie pratique de ce travail et consiste à implémenter les méthodes utilisées.

Le quatrième chapitre est une discussion des résultats obtenus comparés aux travaux cités dans les productions scientifiques.

PARTIE 1 :

RECHERCHE BIBLIOGRAPHIQUE

CHAPITRE 1 : LA POST-GÉNOMIQUE

INTRODUCTION

Le développement du séquençage était une révolution dans le domaine biologie, car il a permis pour la 1ère fois l'identification de la séquence nucléique. Peu de temps après c'était la naissance de la bioinformatique, la discipline qui va se charger d'analyser et traiter les données obtenues par le séquençage. Plusieurs années plus tard, de nouvelles technologies de séquençages voient le jour : les NGS. Ces technologies, beaucoup trop rapide, sont capables de séquencer un génome entier en peu de temps, mais jamais en une seule lecture.

Entre la quantité énorme et la nature de données obtenues comme résultat du séquençage, plusieurs problèmes sont apparus : l'assemblage des petits fragments séquencés pour reconstituer le génome/chromosome d'une manière correcte, analyser ces séquences nucléiques, ou encore annoter les protéines traduites à partir des séquences des gènes.

Depuis l'avènement du séquençage, plusieurs méthodes ont été découvertes et plusieurs technologies ont été développées. On peut classer ces découvertes et technologies en plusieurs générations :

La 1ère génération, Marquée par la technique de Sanger est basée sur la synthèse d'un brin complémentaire. Elle utilise une amorce marquée radio-activement. Le séquençage est réalisé à l'aide de didésoxyribonucléotides (ddNTP) associés aux 4 types de nucléotides. Une fois incorporés dans le brin d'ADN, l'élongation est bloquée au niveau des nucléotides associés au didésoxyribonucléotide incorporé.

Chacune de réactions produit un ensemble de fragments d'ADN de tailles différentes. Il faut ensuite procéder à une électrophorèse afin de séparer les fragments selon leurs tailles. Une ligne indépendante sur la plaque de gel de l'électrophorèse est utilisée pour chaque réaction. La lecture de la séquence se fait à partir des bandes sur la plaque de gel, qui révèlent les positions des différents nucléotides de la séquence, là où la ddNTP a été incorporé (Morisse, 2019).

La 2ème génération est capable de traiter des milliers de séquences d'une taille qui varie de 50 à 400 pb (Besser *et al*., 2018). Elle s'est distinguée de la première en matière de vitesse. L'année 2005 fut marquée par l'apparition du séquençage à haut débit et plusieurs technologies ont été développées : Illumina, Life Technologies et Roche… mais leurs principes sont presque les mêmes :

- PCR par émulsion ou PCR par phase solide.

- Plusieurs cycles de « Wash & Scan », c'est-à-dire des nucléotides qui sont incorporés dans la chambre de réaction, lavage de la chambre puis capture d'image afin de déterminer les nucléotides incorporés (Metzker, 2010).

Contrairement à la 2ème génération, la troisième génération s'est symbolisée par le séquençage d'une portion d'ADN (Morisse, 2019) et non la totalité du génome. Elle s'est distinguée par ses longs Reads, atteignant une taille de 1 à 100 kb (Besser *et al*., 2018).

Les acteurs majeurs de cette génération sont : Pacific Biosciences et Oxford Nanopore Technologies :

- Pacific Biosciences : leur technique repose sur une ADN polymérase fixée, un colorant fluorescent différent pour chaque type de nucléotides. A chaque fois qu'un nucléotide est incorporé, l'étiquette fluorescente est clivée, et le signal qu'elle renvoie est capté par un détecteur jusqu'à la fin de l'élongation (Rhoads et Au, 2015).
- Oxford Nanopore : comme son nom l'indique cette technologie repose sur des nanopores, qui sont constitués de protéines, le fragment d'ADN traverse, provoquant une variation dans le courant ionique du nanopore. Chaque type de nucléotide provoque une variation différente, et donc c'est grâce à cette variation qu'on peut déterminer la séquence du fragment d'ADN (Kono et Arakawa, 2019).

1. ÉTAPES BIOINFORMATIQUES DE LA POST-GÉNOMIQUE

Ce sont les parties fondamentales bioinformatiques qui succèdent à tout séquençage haut débit. En effet, l'on distingue :

1.1. Fragmentation du génome

Avant de procéder au séquençage il faut passer par une étape cruciale : la préparation des échantillons à séquencer (Ignatov *et al.*, 2019), qui consiste à diviser le génome en petits fragments. Ce sont ces petits fragments à séquencer. Il existe deux méthodes principales de fragmentation (Kasoji *et al.*, 2015) :

1.1.1. Fragmentation mécanique

Il s'agit d'une transmission focalisée d'énergie acoustique à haute fréquence et à courte longueur d'onde appliquée sur l'échantillon d'ADN. La taille des fragments générés varie de 150 à 5000 pb selon l'intensité et la durée d'application des ondes acoustiques (Biolabs, 2017).

1.1.2. Fragmentation enzymatique

La fragmentation enzymatique consiste à utiliser des enzymes de restriction. Cette technique est souvent utilisée lors d'un séquençage ciblant une région précise. La taille de fragments varie selon les enzymes utilisées (Jenkins *et al*., 2002).

1.2. Assemblage

L'assemblage du génome est le processus bioinformatique visant à reconstruire un génome nouvellement séquencé, en utilisant les Reads des différents fragments de ce génome comme entré, et en se basant sur les chevauchements entre ces Reads (Kalyanaraman, 2011). Il existe principalement trois approches distinctes d'assemblage : Algorithme Greedy SCS, Overlap-Layout-Consensus (OLC) et le Graphe de De Bruijn (Choudhuri, 2014).

Dans un exemple d'utilisation de l'algorithme de Greedy SCS, on suppose la séquence suivante : AGATTATGGC. Lors de son séquençage, les Reads : AGAT, GATT, TTAT, TGGC sont obtenus. La figure suivante (Figure 1) résume le processus général d'assemblage.

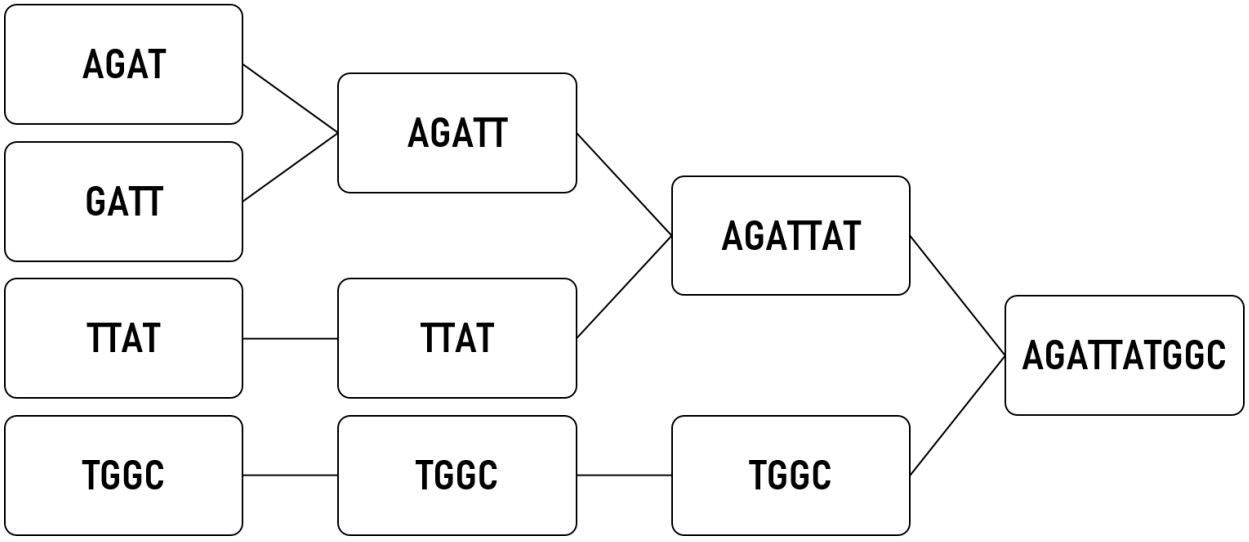

Figure 1 : Exemple démonstratif de l'utilisation de l'algorithme Greedy SCS

1.3. Traduction

Pour avoir la séquence protéique *in silico*, il suffit de simuler le processus de traduction, en se basant sur le code génétique. Cette simulation de traduction fera le passage entre l'assemblage d'un gène et la prédiction de la classification de sa protéine.

2. PRÉDICTION DE LA CLASSIFICATION D'UNE PROTÉINE

C'est une étape post-génomique et bioinformatique importante pour la vérification des étapes initiales (Fragmentation, Assemblage et Traduction) qui auront suivi le séquençage haut débit. En effet la séquence primaire de la protéine obtenue après traduction va être annotée afin de déterminer sa structure tertiaire et sa fonction. Cette annotation est majoritairement dépendante de la structure fine de la protéine (structures 2D et 3D) et de sa fonction physio-enzymatique.

2.1. Généralités sur les protéines

Les protéines sont des macromolécules essentielles du vivant. Elles peuvent comporter des milliers d'atomes et représentent la moitié du poids sec des cellules (Levinthal, 1966). Elles sont constituées de longues chaînes d'acides aminés reliées par des liaisons peptidiques (P.C *et al.*, 2006). La succession des résidus amino-acides constitue la structure primaire de la protéine. Les différentes interactions entre ces résidus permettent le repliement 2D de cette protéine. On distingue des repliements donnant des hélices formées par quelques résidus (une dizaine), des

feuillets (une dizaine) et des coudes (5 à 6 résidus). La conformation 3D est un repliement de la structure 2D sur elle-même.

Les protéines assurent les fonctions nécessaires pour la vie des cellules. Ce sont les seules macromolécules biologiques qui possèdent ces fonctions multiples et variées, accumulées durant des millions d'années d'évolution (Tiwari, 2010) :

- Catalyseurs qui maintiennent les processus métaboliques dans la cellule.
- Récepteurs qui transmettent des informations entre le milieu extérieur et intérieur.
- Élément structurel à l'intérieur et l'extérieur de la cellule.
- Élément essentiel dans des processus tels que la réplication de l'ADN.
- Élément important dans la réponse immunitaire coordonnée et organisée (anticorps et immunoglobulines) (Laroum, 2011).

2.2. Classification des protéines

Il existe plusieurs critères selon lesquelles se fait la classification. Les classifications les plus courantes se font suivant la structure de la protéine, ou suivant son rôle biologique.

2.2.1. Suivant la structure

Les protéines présentent différents niveaux de structure:

- Structure primaire : correspond à la séquence en acides aminés de la protéine autrement dit à l'ordre de ces acides aminés le long de la chaîne polypeptidique. Elle est le résultat direct de la traduction de l'ARNm en séquence protéique (Rashid *et al.*, 2015).
- Structure secondaire : correspond à la conformation locale des peptides. Elle consiste à définir les repliements réguliers et répétitifs : hélices, feuillets, coudes et boucles. Ces structures sont majoritairement dictées par les liaisons hydrogènes (Rashid *et al*., 2015).
- Structure tertiaire : correspond au repliement de la chaîne polypeptidique en trois dimensions. Elle correspond également à la compaction et l'agencement stable dans l'espace des structures secondaires de la protéine (Breda *et al*., 2007). La structure tertiaire d'une protéine est maintenue par différentes forces de cohésion:
    - Les interactions covalentes comme les ponts disulfures entre les atomes de souffre des résidus cystéines

  - Les interactions électrostatiques comme les liaisons ioniques (par exemple entre la chaine latérale de l'arginine et de l'aspartate) ou les interactions de type "liaison hydrogène"
  - Les interactions de van der Waals
  - Les interactions avec l'environnement de la protéine (solvant, ions, lipides, ...)
- Structure quaternaire : est caractérisée par l'assemblage de plusieurs chaînes polypeptidiques possédant chacune sa propre structure tertiaire. Chaque monomère est appelé sous-unité et l'agencement tridimensionnel des sous-unités est stabilisé le plus souvent par le biais d'interactions non covalentes. On distingue plusieurs types de structures quaternaires (Breda *et al.*, 2007) :
  - Une structure homodimérique quand les sous-unités sont identiques (Rashid *et al*., 2015).
  - Une structure est hétérodimérique quand les sous-unités sont différentes (Rashid *et al.*, 2015).

La structure 3D divise les protéines en structure fibreuse et globulaire :

- Protéines fibreuses ou scléroprotéines : Les chaînes polypeptidiques de ces protéines sont allongées et enroulées autour d'un axe sous une forme hélicoïdale. Ces protéines jouent un rôle structurel et constitutif, elles sont insolubles dans l'eau, et sont dotés d'une grande stabilité. Elles peuvent être extracellulaires**,** auront une fonction de protection comme elle peut être intracellulaire. Les principales protéines fibreuses connues sont la kératine (cheveux, ongles et poils), la fibroïne (la soie), l'élastine (la peau), le collagène (les tendons), la myosine et la tropomyosine (cellules musculaires) (Shen, 2019).
- Protéines globulaires ou sphéroprotéines : sont des protéines contenant des nombres substantiels d'hélices α et de feuillets β pliés en une structure compacte sphérique ou globulaire, qui est stabilisée par des interactions polaires et non polaires. Le plus souvent, les chaînes latérales d'acides aminés hydrophobes sont enterrées, encapsulées de façon fermée, à l'intérieur d'une protéine globulaire, hors de contact avec l'eau. Les chaînes latérales d'acides aminés hydrophiles se trouvent à la surface des protéines globulaires exposées à l'eau. Par conséquent, les protéines globulaires sont généralement très solubles dans les solutions aqueuses (Shen, 2019).

2.2.2. Suivant la fonction

Les protéines peuvent assurer diverses fonctions au sein de la cellule ou de l'organisme tel que les protéines de structure, de défense, de transport, de stockage, de régulation, des protéines

motrices (Whisstock et Lesk, 2003), ainsi que la famille des enzymes dans laquelle on distingues six classes enzymatiques (Robinson, 2015) :

1) Oxydoréductases
2) Transférases
3) Hydrolases
4) Lyases
5) Isomérases, synthétases
6) Ligases

2.3. Annotation des protéines et méthodes traditionnelles des études de fonctions

Il existe plusieurs méthodes et techniques d'annotation *in silico* des protéines que les bioinformaticiens utilisent pour attribuer des rôles biologiques aux protéines. Les protéines et leurs fonctions sont prédites après l'analyse des données de séquençage génétique. Les prédictions nécessitent des méthodes de calculs de données à forte intensité. Ces méthodes distinguent deux types d'approches : des approches basées sur la séquence, et d'autres basées sur la structure, comme indiqué ci-dessous dans (la Figure 2).

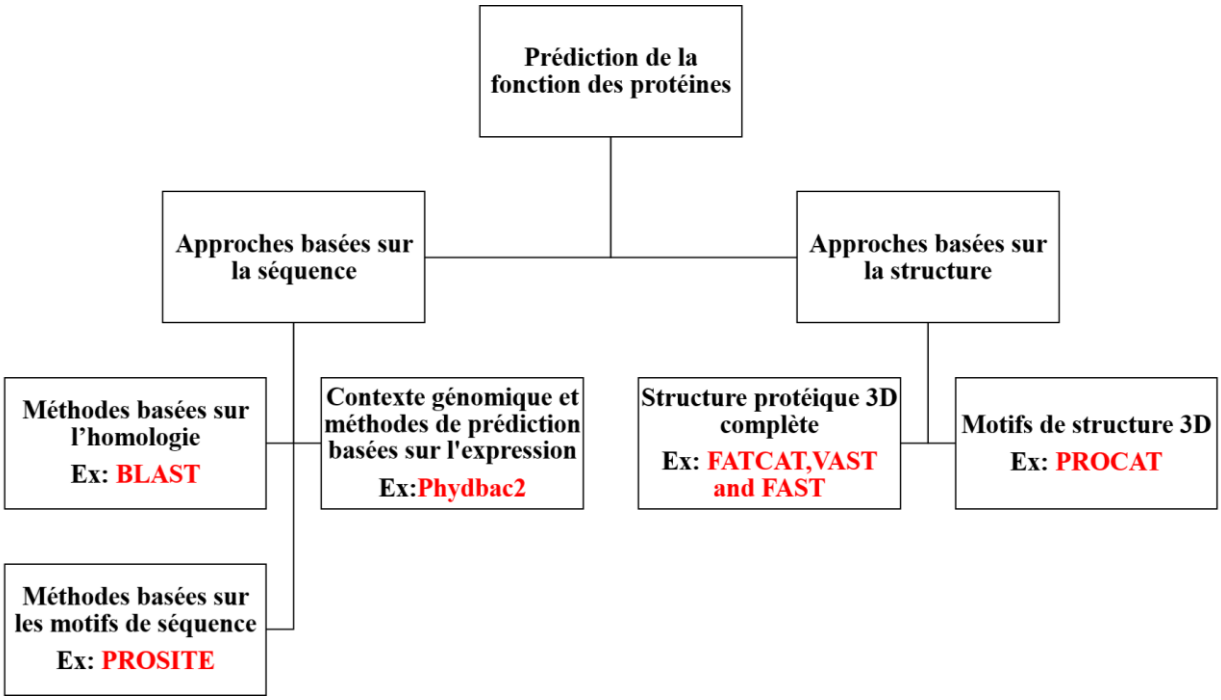

Figure 2 : Aperçu schématique des méthodes de prédiction de la fonction d'une protéine *in silico* (Sleator et Walsh, 2010)

2.3.1. Méthodes basées sur l'homologie

Les protéines ayant des séquences similaires sont généralement homologues et peuvent avoir la même fonction biologique (Reeck *et al.*, 1987). Ainsi, les protéines d'un génome nouvellement séquencé sont systématiquement annotées en utilisant des séquences protéiques similaires dans des génomes apparentés. Cependant, ils ne partagent pas toujours les mêmes fonctions (Whisstock et Lesk, 2003).

Par exemple les protéines de levure *Gal1* et *Gal3* sont des paralogues (73% d'identité et 92% de similarité) et possèdent des fonctions très différentes (Platt *et al.*, 2000) :

- *Gal1* étant une galactokinase.
- *Gal3* étant un promoteur (inducteur) de transcription.

Certains outils utilisés dans cette méthode sont :

- ClustalW2 : est un programme d'alignement de séquences multiples d'ADN ou de protéines à usage général pour trois séquences ou plus ("ClustalW2 EMBL-EBI," 2020).
- PSI-BLAST (Position-Specific Iterative Basic Local Alignment Search Tool): est une méthode de recherche de profil de séquence protéique qui s'appuie sur les alignements générés par une exécution du programme BLASTp (Altschul *et al.*, 1997).
- BLAST (Basic Local Alignment Search Tool): algorithme heuristique de comparaison de séquences de protéines ou d'ADN (Altschul *et al.*, 1990). Une recherche BLAST compare une séquence requête aux séquences de la bases de données et récupère celles qui ressemblent à la séquence requête, au-dessus d'un certain seuil (Wiltgen, 2019).

2.3.2. Méthodes basées sur les motifs

Pfam (Protein Families Database) est l'une des bases de données de domaines protéiques les plus développées parmi ses pairs (Finn *et al.*, 2010). Elle permet de découvrir des domaines connus dans les séquences de requêtes et de fournir des preuves de fonctions potentielles. Les domaines protéiques contiennent des motifs c'est-à-dire de courtes signatures connues qui sont associées à des fonctions particulières (Sleator et Walsh, 2010). Il existe d'autre bases de données dédiées à la recherche de motifs telle que PROSITE (Hulo *et al.*, 2008), qui dispose de deux type de signatures pour identifier les régions conservé qui sont les patterns et les profiles protéiques (Sigrist *et al.*, 2010).

Certains outils utilisés dans cette méthode sont :

- Multiple Expression motifs for Motif Elicitation (MEME) est un outil pour découvrir des motifs dans un groupe de séquences d'ADN ou de protéines apparentées (Bailey, 2011).
- Gapped local alignment of motifs (GLAM 2) est un outil pour découvrir des motifs espacés dans un groupe de séquences d'ADN ou de protéines (Frith *et al.*, 2008).
- Find Individual Motif Occurrences (FIMO) est un outil permettant de rechercher des instances de motifs dans une base de données de séquences (Grant *et al.*, 2011).

2.3.3. Méthodes basées sur la structure 3D

Étant donné que la structure 3D des protéines est fréquemment mieux conservée que les séquences protéiques, la similitude structurelle est un bon indicateur d'une fonction similaire dans les protéines (Sleator et Walsh, 2010). En effet, la plupart des plis protéiques connus sont associés à une fonction ou à un milieu fonctionnel particulier (Todd *et al.*, 2001). Du coup des programmes bioinformatiques d'analyse de structure 3D ont été créer dans le but de comparer une structure requête à une base de donnée de structures (Sleator et Walsh, 2010).

Certains outils utilisés dans cette méthode sont :

- Structurally Aligned Local Sites of Activity (SALSA) : est une nouvelle méthode de prédiction de la fonction protéique basée sur une correspondance structurelle locale au site catalytique ou de liaison prédit (Wang *et al.*, 2013).
- MODELLER est l'un des outils de calcul largement utilisés pour prédire les structures 3D des protéines à l'aide de la modélisation d'homologie (Webb et Sali, 2014).
- MaxMod, PyMod et PRIMO sont d'autres méthodes / serveurs récents pour la modélisation d'homologie (Gromiha *et al.*, 2019).

CHAPITRE 2 : CONCEPTS D'INTELLIGENCE ARTIFICIELLE

INTRODUCTION

L'intelligence artificielle, abrégée IA, a toujours constitué une frontière, incessamment repoussée. L'IA est fondé autour d'un objectif ambitieux : comprendre comment fonctionne la cognition humaine et comment la reproduire afin de créer des processus cognitifs comparables à ceux de l'être humain. Ce domaine est extrêmement vaste, que ça soit en ce qui concerne les procédures techniques utilisées ou bien les disciplines convoquées tel que les mathématiques, l'informatique et les sciences cognitives. Les méthodes de l'IA sont très nombreuses et diverses mais elles ne sont pas nouvelles : beaucoup d'algorithmes utilisés aujourd'hui ont été développés il y a plusieurs dizaines d'années.

La recherche en IA a donné lieu à des vrais succès et a nourri largement l'histoire des mathématiques et de l'informatique, beaucoup de disciplines ont d'ailleurs profité de cette avancée.

Parallèlement, les outils de calcules qui était efficaces auparavant, se voient dépassé de jour en jour. Les outils biologiques n'ont pas échappé à cet avènement, y compris les méthodes traditionnelles de classifications de protéines. Ce qui nous amène à penser à de nouvelles approches, notamment des approches d'IA par apprentissage automatique.

Ce chapitre comporte les concepts principaux nécessaires à la compréhension des techniques d'apprentissage, commençant par l'apprentissage automatique qui est la base, allant à l'apprentissage profond puis au traitement automatique du lagunage naturel. Ces 2 dernières techniques représentent la pierre angulaire de ce travail.

1. APPRENTISSAGE AUTOMATIQUE

L'apprentissage automatique, ou Machine Learning en anglais, abrégé (ML) : est l'un des sous-domaines les plus prometteurs de l'IA. Le ML est un sujet vaste et complexe qui utilise une variété de techniques statistiques, probabilistes et d'optimisation et qui soulève de nombreux problèmes subtils (Cruz et Wishart, 2007).

L'objectif du ML est de rendre la machine ou l'ordinateur capable d'apporter des solutions à des problèmes compliqués, en lui permettant d'apprendre des exemples passés représentés par une quantité astronomique de données, afin de détecter des modèles difficiles à discerner. Cela offre ainsi une possibilité d'analyser et de mettre en évidence les corrélations qui existent entre deux ou plusieurs situations données, et de prédire leurs différentes implications (Cruz et Wishart, 2007; "Machine learning definition," 2016).

1.1. Exemple de l'apprentissage automatique

Il existe certaines tâches dont nous ne disposons pas d'algorithme efficace, par exemple, pour distinguer les spams des e-mails légitimes. Nous connaissons l'entrée qui est : un document de courrier électronique, qui dans le cas le plus simple est un fichier de caractères. Nous connaissons ce que doit être la sortie: une sortie "oui" indiquant que le message est du spam ou une sortie "non" indiquant qu'il ne l'est pas. Nous ignorons comment transformer l'entrée en sortie car ce qui peut être considéré comme du spam change dans le temps et d'un individu à l'autre. Tout simplement parce que nous manquons de connaissances, et par conséquent il faudra compenser en matière de données. Nous pouvons facilement compiler des milliers d'exemples de messages dont certains que nous savons être du spam. Notre objectif c'est d'apprendre ce qui constitue le spam de leur part. En d'autres termes, nous aimerions que l'ordinateur extrait automatiquement l'algorithme de cette tâche (Alpaydin, 2009).

1.2. Processus de l'apprentissage automatique

On peut diviser le processus du ML en deux phases distinctes :

1.2.1. Phase d'apprentissage

Dans cette phase, la machine doit construire un modèle, qui sera son système de raisonnement sans qu'on lui impose les règles de la création de ce modèle. Pour y arriver il faut fournir à la machine une masse d'exemples qu'elle va analyser afin qu'elle puisse comprendre la logique du modèle quelle doit intégrer et déterminer l'algorithme de transformation ("Quels sont les usages du Machine Learning (apprentissage automatique) ?," 2019).

1.2.2. Phase de prédiction

Une fois que le raisonnement et l'algorithme du problème en question intégré à la machine, le programme du ML acquiert la capacité de déterminer la finalité d'une situation donnée. Plus l'apprentissage de la machine est complet, plus les prédictions obtenues par cet outil seront précises ("Quels sont les usages du Machine Learning (apprentissage automatique) ?," 2019).

1.3. Différents champs d'applications de l'apprentissage automatique

Le ML est actuellement très utilisé pour le BIG DATA dans différents domaines. La figure suivante (Figure 3) démontre donc les champs d'applications les plus courant du ML.

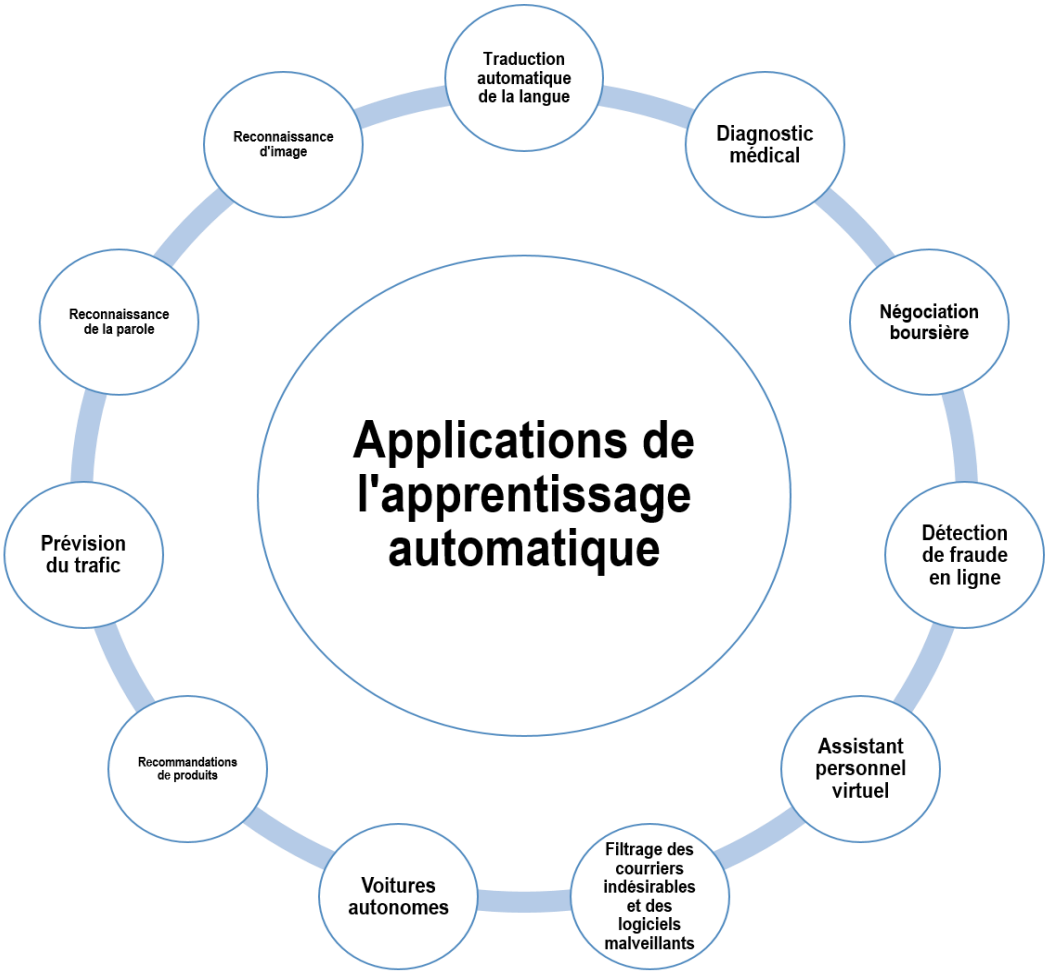

Figure 3 : Différents champs d'applications du ML ("Applications of Machine Learning - Javatpoint," 2018)

1.4. Types d'apprentissage automatique

Le ML dispose principalement de deux classes : le ML supervisé et le ML non supervisé (voir la Figure 4). La différence majeure entre les deux c'est que le premier fonctionne avec des données étiquetées et le second fonctionne avec des données non étiquetées. La figure suivante

représente une illustration des classes principales du ML, leurs types et quelques exemples de leurs applications. Seul l'apprentissage supervisé sera développé étant donné que c'est la classe utilisée dans notre travail.

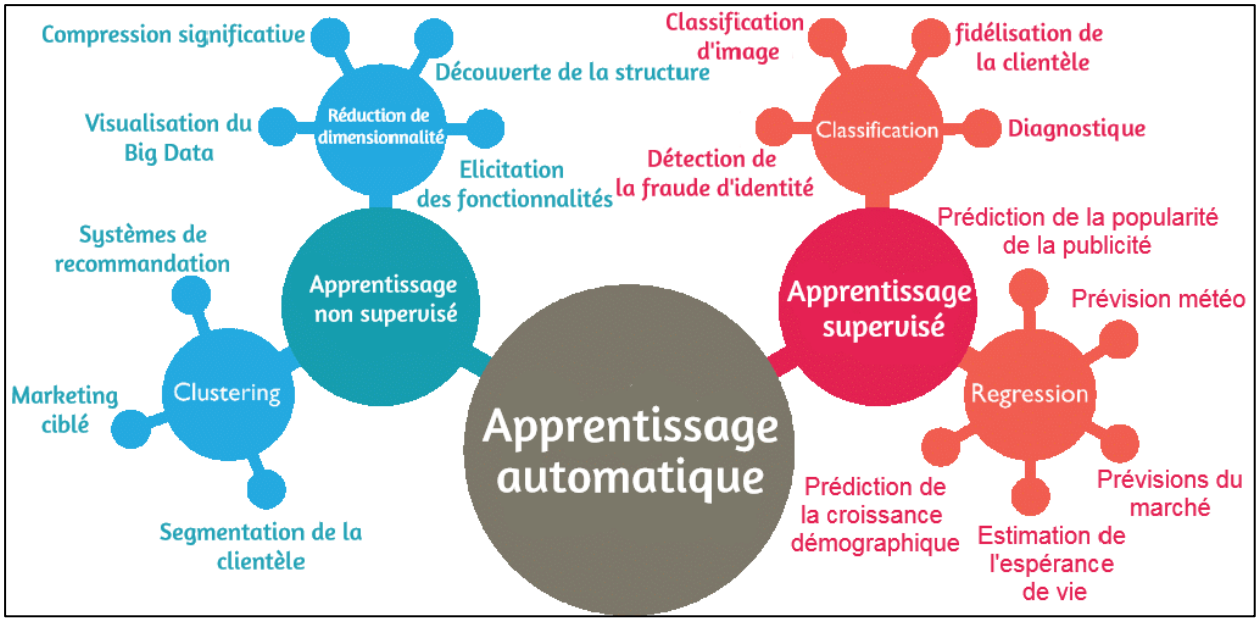

Figure 4 : Illustration des classes principales du ML, leurs types et leurs applications ("Types of Machine Learning Algorithms," 2020)

1.4.1. Apprentissage automatique supervisé

Les algorithmes du ML supervisé sont entraînés à l'aide d'un ensemble d'exemples étiquetés. Chaque exemple est composé d'une ou de plusieurs entrées accompagnées par la sortie correcte correspondante. L'algorithme apprend en comparant ses propres sorties avec les sorties correctes pour trouver les erreurs. Il modifie ensuite le modèle en conséquence (Dey, 2016) grâce à des méthodes telles que la classification ou la régression qu'on va voir ensuite.

Afin d'évaluer la précision du modèle, l'apprentissage supervisé prédit les valeurs de l'étiquette sur des données supplémentaires non étiquetées. Ce type d'apprentissage est couramment utilisé dans les applications où les données historiques prédisent des événements futurs probables (Ongsulee, 2017).

Le ML supervisé se divise à son tour en deux types: la classification et la régression, mais seule la classification sera développée étant donné que c'est le type d'apprentissage utilisé dans notre travail.

1.4.2. Classification

La classification est la tâche consistant à attribuer une classe à une donnée qu'on veut classer, par le biais d'un modèle dont le but est de rapprocher une fonction (f) des variables d'entrée (X) aux variables de sorties (Y), et c'est cette fonction qui prédit la classe d'une observation donnée.

Les modèles de classification prédisent souvent une valeur continue comme la probabilité qu'un exemple donné appartienne à chaque classe de sortie. Une probabilité prédite peut-être convertie en valeur de classe en sélectionnant l'étiquette de classe qui a la probabilité la plus élevée. Il faut savoir aussi que (Bouguelia, 2015) :

- Un problème de classification nécessite que les exemples soient classés dans une ou plusieurs classes.
- Une classification peut avoir des variables d'entrée à valeur réelle ou discrète.
- Un problème avec deux classes est souvent appelé un problème de classification binaire.
- Un problème avec plus de deux classes est souvent appelé un problème de classification multi-classes.

Afin d'y voir un peu plus clair nous allons revenir sur notre exemple d'e-mails à classifier entre spam et non spam (voir la Figure 5):

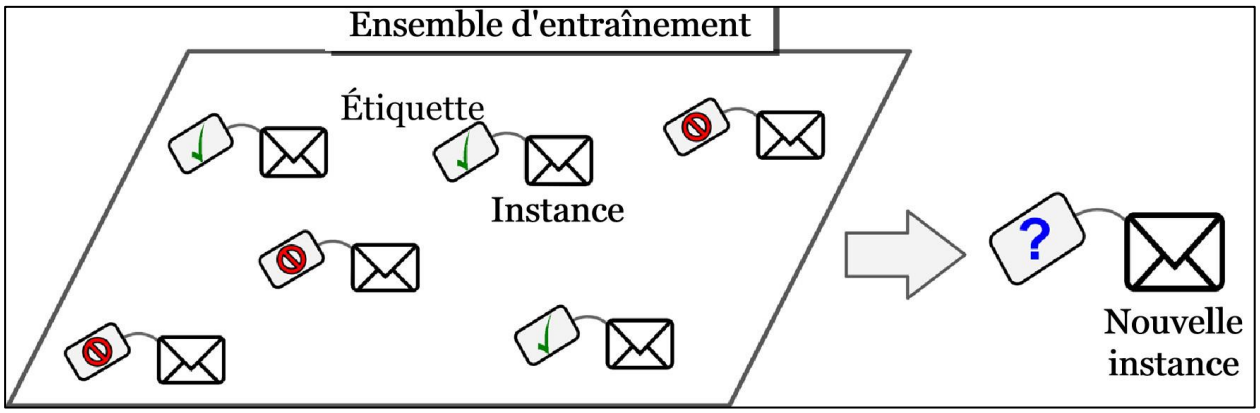

Figure 5 : Ensemble d'e-mails classifiés pour un apprentissage supervisé (Géron, 2017)

Un e-mail spécifique contenant du texte peut se voir attribuer les probabilités de 0,1 comme étant « spam » et 0,9 comme « non spam ». Nous pouvons convertir ces probabilités en une étiquette de classe en sélectionnant l'étiquette « pas de spam » car elle a la probabilité prédite la plus élevée (Géron, 2017).

Il existe de nombreuses façons pour estimer la compétence d'un modèle prédictif de classification, mais la plus courante est de calculer la précision de la classification.

La précision de la classification est le pourcentage d'exemples correctement classés parmi toutes les prédictions faites.

Par exemple, si un modèle prédictif de classification faisait cinq prédictions et que trois d'entre elles étaient correctes et que deux d'entre elles étaient incorrectes, alors la précision de classification du modèle basée uniquement sur ces prédictions serait (Brownlee, 2017a):

Précision = prédictions correctes / prédictions totales * 100

Précision = 3/5 * 100

Précision = 60%

2. APPRENTISSAGE PROFOND

La première chose à savoir est que l'apprentissage profond, ou Deep learning en anglais, abrégé (DL), c'est du ML. Plus précisément, le DL est considéré comme une évolution du ML (voir la figure 6). Il utilise un réseau neuronal programmable qui permet aux machines d'extraire par elles-mêmes les caractéristiques significatives à l'apprentissage, contrairement au ML qui repose sur le choix du data scientist afin de sélectionner les attributs de l'apprentissage ("La vraie différence entre Machine Learning & Deep Learning | Jedha Bootcamp," 2020; Patterson et Gibson, 2017).

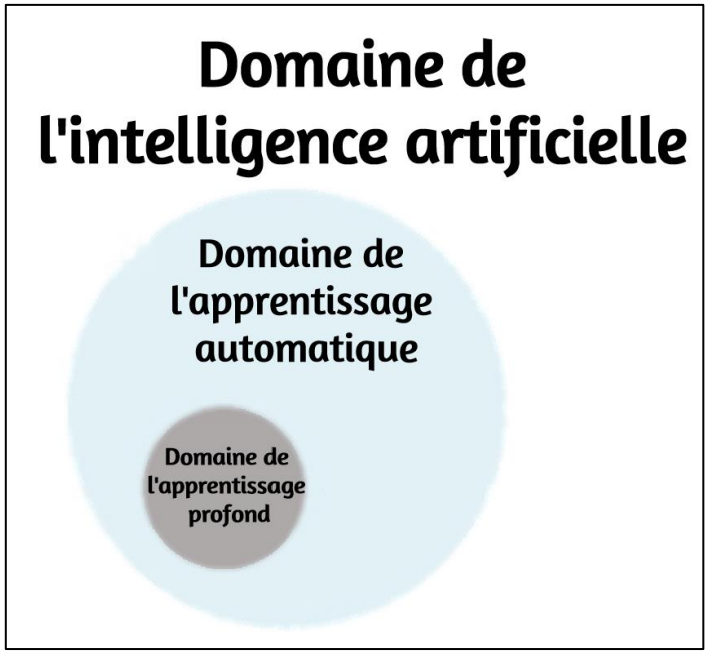

Figure 6 : Relation entre l'IA, ML et DL (Patterson et Gibson, 2017)

Bien que les modèles du ML de base s'améliorent progressivement, quelle que soit leur fonction, ils ont encore besoin de conseils. Si un algorithme renvoie une prédiction inexacte, on se trouve contraint d'intervenir et d'effectuer des ajustements. Avec un modèle de DL, un algorithme peut déterminer seul si une prédiction est précise ou non via son propre réseau de neurones.

Prenons l'exemple de la lampe de poche afin d'y voir plus clair: on peut la programmer pour s'allumer dès qu'elle reconnaît le signal sonore de quelqu'un prononçant le mot « sombre ». Au fur et à mesure que le programme continue à apprendre, il peut éventuellement s'allumer avec n'importe quelle phrase contenant ce mot. Maintenant, si la lampe de poche avait un modèle d'apprentissage en profondeur, elle pourrait comprendre qu'elle devrait s'allumer avec les signaux « Je ne peux pas voir » ou « l'interrupteur d'éclairage ne fonctionne pas », peut-être en tandem avec un capteur de lumière (Grossfeld, 2020).

Un modèle d'apprentissage en profondeur est capable d'apprendre grâce à sa propre méthode de calcul, une technique imitant la façon dont le cerveau perçoit et comprend les informations, capturant ainsi implicitement les structures complexes de données à grande échelle donnant l'impression d'avoir son propre cerveau (Voulodimos *et al.*, 2018).

2.1. Réseau de neurones artificiels

Un algorithme d'apprentissage de réseau de neurones artificiels, ou un réseau de neurones, est un système d'apprentissage qui utilise un réseau de fonctions pour comprendre et traduire une entrée de données d'une forme en une sortie souhaitée. Le concept de réseau neuronal artificiel a été inspiré par la biologie humaine (voir la Figure 7) et la façon dont les neurones du cerveau humain fonctionnent ensemble pour comprendre les apports des sens humains (Deitel et Deitel, 2018).

Les réseaux de neurones ne sont que l'un des nombreux outils et approches utilisés dans les algorithmes de DL. Le réseau neuronal lui-même peut être utilisé comme élément dans de nombreux algorithmes de DL pour traiter des entrées de données complexes dans un espace que les ordinateurs peuvent comprendre (Voulodimos *et al.*, 2018).

Les réseaux de neurones sont appliqués à de nombreux problèmes de la vie réelle aujourd'hui, notamment la reconnaissance de la parole et des images, le filtrage des courriers indésirables, la finance et le diagnostic médical…

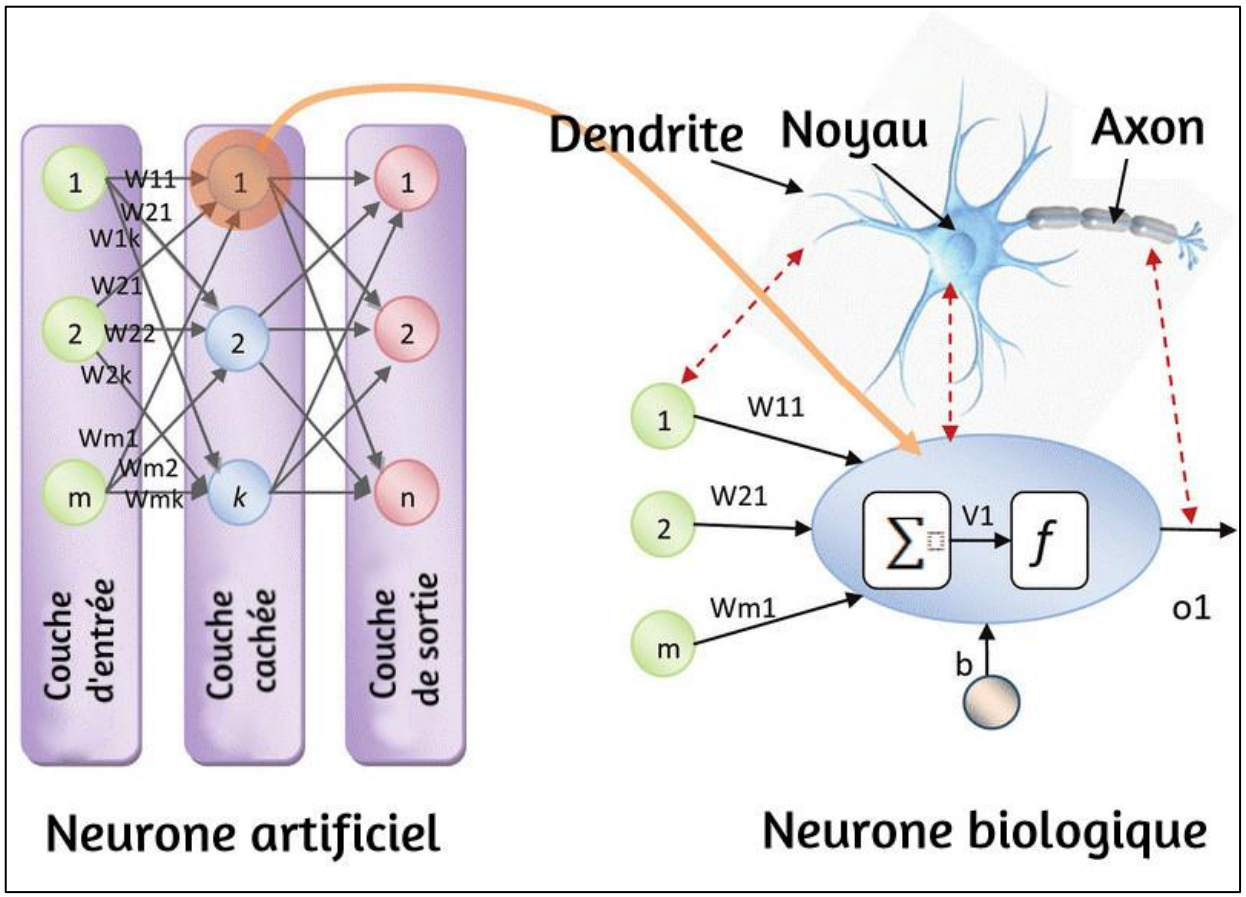

Figure 7 : Comparaison d'un réseau de neurone artificiel avec un neurone biologique (Lee *et al.*, 2012)

Selon la constitution et la fonction (voir Figure 8), trois éléments d'un modèle de neurone artificiel sont identifiés comme suit (Zhao, 2019) :

2.1.1. Poids

Pour un réseau neuronal biologique, deux neurones sont reliés par une synapse qui peut être entraînée par des activités. Pour un neurone artificiel, l'effet d'une synapse est incarné par le poids entre deux neurones.

2.1.2. Fonction de combinaison

Pour un neurone biologique, les signaux impulsionnels d'entrée sont additionnés et comparés au seuil. Pour un neurone artificiel, les entrées sont pondérées puis additionnées dans une fonction de combinaison.

2.1.3. Fonction d'activation

Pour un neurone biologique, une fonction de seuil est utilisée pour déterminer si le neurone est activé ou non. Pour un neurone artificiel, le seuil ou d'autres fonctions non linéaires peuvent être utilisés comme fonctions d'activation.

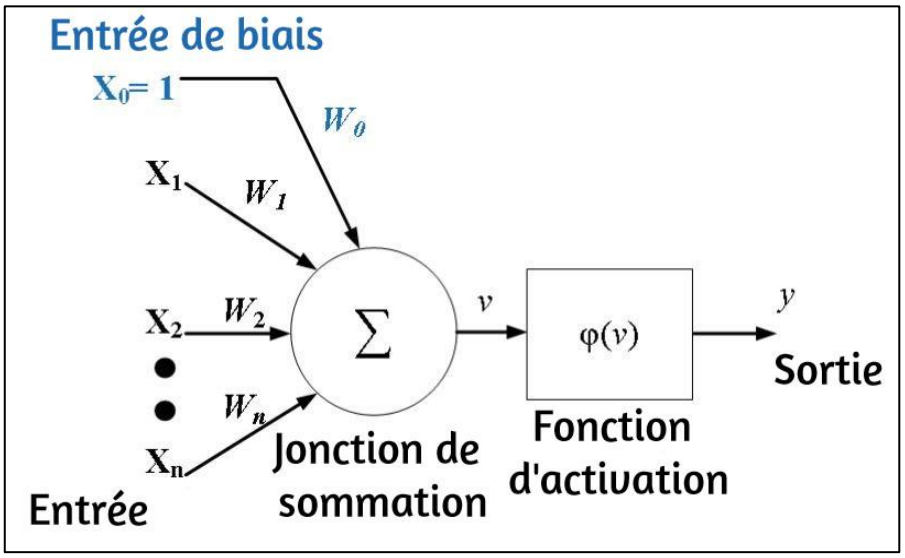

Figure 8 : Représentation d'un neurone artificiel (Zhao, 2019)

Un modèle de neurone artificiel est représenté dans la figure, où x1, x2… xn sont des entrées, x0 est une entrée de biais. w0, w1, w2 … wn sont les poids correspondants aux entrées. La valeur de la fonction de combinaison (v) est calculée comme suit (Voir figure 9):

$$v = \sum_{i=1}^{n} x_i \times w_i + x_0 \times w_0$$

Figure 9 : Équation de la fonction de combinaison (Zhao, 2019)

Ensuite, la valeur de la fonction de combinaison (v) est introduite dans la fonction d'activation (φ (x)) (Voir figure 10), qui déterminera la sortie du neurone artificiel (y), exprimée comme :

$$y = \varphi(v)$$

Figure 10 : Équation de la fonction d'activation (Zhao, 2019)

Il existe plusieurs types de réseaux de neurones, le réseau utilisé dans ce travail est de type convolutif.

2.2. Réseau de neurones convolutif

Le réseau de neurones convolutif, ou Convolutional Neural Network en anglais, abrégé CNN, a été conçu à l'origine pour effectuer un apprentissage pour les tâches de vision par ordinateur et s'est révélé très efficace. Il s'est avéré que cette approche fonctionne également bien pour le traitement automatique du langage naturel ("Using Convolutional Neural Networks for Sentence Classification," 2020), qui sera développé dans la suite du travail.

Il s'agit de l'un des types de réseaux de neurones artificiels les plus couramment utilisés. Il a une méthodologie similaire à celle des méthodes traditionnelles d'apprentissage supervisé : il reçoit des données en entrée, détecte les caractéristiques de chacune d'entre elles, qu'elles soient simples ou complexes, puis entraîne un classifieur dessus (Schmidhuber, 2015).

Cependant, ces caractéristiques sont apprises automatiquement car le CNN réalise lui-même l'extraction des caractéristiques pendant la phase d'entraînement, l'erreur de classification est minimisée afin d'optimiser les paramètres du classifieur et des caractéristiques.

C'est l'extraction et la hiérarchisation automatiques des caractéristiques s'adaptant aux problèmes donnés, qui constituent la force du CNN.

Le CNN se compose principalement de 4 couches :

2.2.1. Couche de convolution

Elle représente la composante principale du CNN, et constitue toujours au moins la première couche. Son rôle est de repérer les caractéristiques des données reçues en entrée.

2.2.2. Couche de pooling

Ce type de couche est souvent placé entre deux couches de convolution : elle reçoit en entrée les sorties de la couche de convolution, et applique à chacune d'entre elles l'opération de pooling, qui consiste à réduire la taille des données, tout en préservant leurs caractéristiques importantes.

2.2.3. Couche de correction ReLU

ReLU (Rectified Linear Units) désigne la fonction réelle non-linéaire définie par $\mathrm{ReLU}(x)=\max(0,x)$.

La couche de correction ReLU remplace donc toutes les valeurs négatives reçues en entrées par des zéros. Elle joue le rôle de fonction d'activation.

2.2.4. Couche fully-connected

La couche fully-connected constitue toujours la dernière couche d'un réseau de neurones, convolutif ou non, elle n'est donc pas caractéristique d'un CNN. Ce type de couche reçoit un vecteur en entrée et produit un nouveau vecteur en sortie. Pour cela, elle applique une combinaison linéaire puis éventuellement une fonction d'activation aux valeurs reçues en entrée, et c'est à elle que revient le rôle de classifier les données en entrée du réseau.

2.3. Processus de l'apprentissage profond

Le processus du DL se divise en huit phases :

2.3.1. Collection de données

C'est la première étape du processus, elle consiste à récolter et assembler les données nécessaires pour le lancement du processus de DL selon l'objectif désiré. Généralement ces données se trouve dans un état brut, et donc elles sont inutilisables ("Collecte de données étiquetées - Amazon Machine Learning," 2020).

2.3.2. Préparation de données

Une fois les données récoltées, place au prétraitement. Les données doivent être préparées, normalisées, dé-dupliquées et les erreurs et les biais doivent être supprimés. La visualisation des données peut être utilisée pour rechercher des modèles et des valeurs aberrantes pour voir si les bonnes données ont été collectées ou si des données sont manquantes ("Frameworks for Approaching the Machine Learning Process," 2018).

2.3.3. Choix du modèle

Une fois que les données soient prêtes, il faudra choisir le bon modèle, car il existe un grand nombre de modèle, qui diffère les uns des autres selon le type d'apprentissage, le types de données qu'ils peuvent traiter, ou même selon le type de sorties ou les résultats désirés (Rijmenam, 2019).

2.3.4. Entrainement du modèle

L'entrainement du modèle est une étape cruciale, dont le rôle est d'utiliser les données afin d'améliorer progressivement la prédiction du modèle. L'entrainement ne s'applique pas sur toutes nos données, mais plutôt sur une portion uniquement, le reste des données est utilisé plus tard pour évaluer le modèle. Généralement on utilise 80% des données pour l'entrainement, et les 20% restantes pour l'évaluation (Deitel et Deitel, 2018; Rijmenam, 2019).

2.3.5. Évaluation du modèle

Une fois l'entrainement achevé, il faudra évaluer l'efficacité de notre apprentissage, c'est-à-dire l'appliquer sur les 20% de données non utilisée lors de l'entrainement (Brownlee, 2017b; Deitel et Deitel, 2018).

2.3.6. Réglage des paramètres

Après avoir évaluer le modèle, place aux tests des paramètres initiaux du modèle, c'est-à-dire modifier ces encore et encore, et relancer les phases d'entrainement et d'évaluation dans le but d'améliorer la précision de la prédiction (Rijmenam, 2019).

2.3.7. Enregistrement et chargement du modèle

Une fois les réglages des paramètres du modèle terminé, et que le modèle soit prêt à être utilisé, il faudra l'enregistrer, afin de pouvoir le mettre en œuvre("How to predict new samples with your Keras model?," 2020).

2.3.8. Prédiction

Une fois que tout le processus est achevé, et que notre apprentissage atteint une précision satisfaisante, notre programme devient prêt à être enregistré et à faire de la prédiction sur de nouvelles données (Rijmenam, 2019).

3. TRAITEMENT AUTOMATIQUE DU LANGAGE NATUREL

Le traitement automatique du Langage Naturel, en anglais Natural Language Processing, abrégé (NLP), est une sous discipline de l'IA. C'est l'un des domaines de recherche les plus actifs en science des données actuellement. Il représente l'interaction entre les ordinateurs et les langues humaines (Machiraju et Modi, 2018). Son objectif est d'extraire des informations, voire une signification d'un contenu textuel (Fabien, 2019).

Les techniques et approches modernes du NLP sont basées sur le ML et le DL étant donné qu'ils disposent de modèles et d'algorithmes efficaces en extraction de caractéristiques (Ayyappan, 2017; Graeme, 2017)

3.1. Applications du traitement automatique du langage naturel

Le traitement automatique du langage naturel dispose de plusieurs techniques, celle utilisée dans ce travail est la classification des textes.

3.1.1. Classification des textes

La classification de texte est un processus de classification de morceaux de texte en différentes catégories. C'est l'une des tâches NLP les plus simples mais les plus largement utilisées. Par exemple, le filtrage anti-spam est un type de classification de texte. Il classe les e-mails en deux catégories: spam ou non. C'est pourquoi on reçoit très peu de spams lorsque on utilise Gmail qui est un service de Google.

3.1.2. Opinion mining/ analyse de sentiments

L'opinion mining ou en anglais sentiment analysis, est un type particulier de classification de textes, sa particularité réside dans sa capacité à identifier automatiquement des informations subjectives, telles que des opinions, des émotions ou des sentiments dans le texte. L'une des tâches les plus élémentaires de l'opinion mining est la classification de la polarité, c'est-à-dire de classer si l'opinion exprimée est positive, négative ou neutre (Boullier et Lohard, 2012).

Il existe bien évidemment beaucoup d'autres applications et tâches accomplie par le NLP telles que : la traduction automatique, la correction de l'orthographe et des erreurs grammaticale, la Génération automatique de textes (Soni et Thakur, 2018).

3.1.3. Relation entre le traitement automatique du langage naturel et l'apprentissage profond

La figure 11 ci-dessous montre graphiquement comment la NLP est liée au DL.

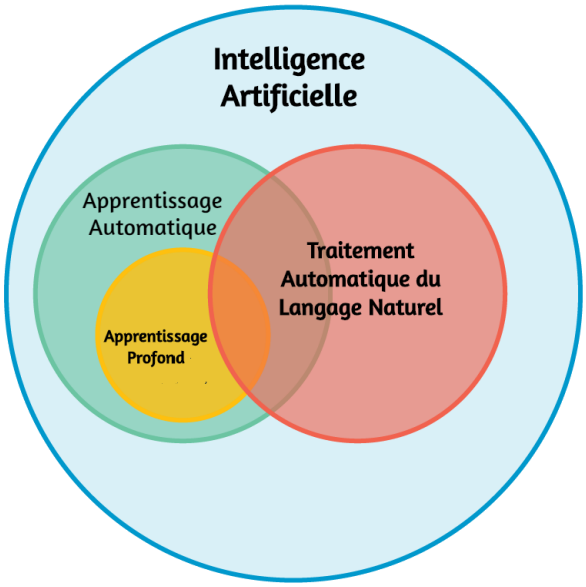

Figure 11 : Relation entre DL, ML et NLP (Tayiz, 2020)

Le NLP, accompagnés par les modèles adéquats du ML et du DL, en imitant la réflexion humaine, permet de créer des systèmes intelligents, capables de comprendre l'intention derrière le texte d'un utilisateur et de leur fournir une réponse appropriée (Hagiwara, 2020).

Les séquences protéiques sont représentées par des chaines de caractères ou chaque caractère représente un acide aminé. Les séquences protéiques ne sont donc qu'un simple texte d'un point de vue purement informatique, ce qui offre la possibilité de faire usage des techniques du NLP afin d'étudier les protéines. Par ailleurs ce travail est basé sur l'une des techniques du NLP qui est l'opinion mining.

PARTIE 2 :

MATÉRIEL ET MÉTHODES

1. MATÉRIEL

1.1. Données biologiques

Deux types de données ont été utilisés : séquences nucléiques utilisées pour la phase post-génomique de la fragmentation, de l'assemblage et de la traduction, et des séquences protéiques pour la phase post-génomique prédictive.

1.1.1. Données nucléiques

La séquence du gène (Ribokinase A) à fragmenter a été téléchargée sur la banque de données GenBank sous format de fichier fasta. Le format fasta est utilisé pour stocker des séquences nucléiques ou protéiques. La séquence dans le fichier fasta a été utilisée par un code python afin de générer un fichier fastq. Voici l'exemple[1] de la séquences fasta du gène (Ribokinase A) utilisé.

```
>NC_007795.1:c260822-259908 Staphylococcus aureus subsp. aureus NCTC 8325
chromosome, complete genome
ATGACCAACAAAGTTGTTATTTTAGGTTCAACGAATGTCGATCAATTTTTAACAGTTGAAAGATATGCAC
AACCAGGCGAAACATTACATGTTGAAGAAGCACAAAAAGCATTCGGCGGAGGTAAAGGTGCCAACCAGGC
TATTGCCACTGCACGCATGCAAGCAGACACAACATTTATTACTAAAATTGGCACTGATGGCGTTGCTGAT
TTCATCTTAGAAGATTTTAAAGTAGCTCATATTGATACATCATATATTATCAAAACAGCTGAAGCAAAAA
CGGGCCAAGCCTTTATCACTGTGAATGCAGAAGGACAAAACACCATCTATGTTTATGGTGGTGCGAATAT
GACGATGACACCTGAAGATGTTATTAACGCAAAAGACGCTATAATCAATGCAGACTTTGTCGTTGCACAA
TTAGAAGTACCCATCCCGGCTATTATATCTGCATTTGAAATTGCCAAGGCACATGGTGTGACGACAGTAT
TAAATCCTGCACCAGCGAAAGCATTACCTAATGAATTATTATCATTAATCGATATTATTGTGCCAAACGA
AACAGAAGCCGAATTGTTATCTGGGATTAAAGTAACTAATGAACAATCTATGAAAGACAATGCCAATTAC
TTTTTATCTATAGGCATTAAGACTGTTTTGATTACGCTAGGTAAGCAAGGTACATATTTTGCTACTAAAA
ATCAAAGCCAACACATCGAAGCTTATAAAGTAAATGCGATTGATACAACTGCTGCAGGCGACACATTTAT
TGGTGCATTTGTCAGTCGCTTAAACAAGTCGCAAGATAACTTAGCAGATGCTATTGATTTTGGTAATAAA
GCGAGCTCACTCACTGTACAAAAACACGGCGCGCAAGCATCTATTCCTCTACTAGAAGAAGTAAATCAAG
TTTAA
```

Le fichier fastq a été utilisé pour l'assemblage. C'est le format le plus utilisé dans l'analyse des séquences nucléiques, et souvent c'est le format des fichiers de séquençage généré par les NGS (Voir le tableau 1). Le format fastq se compose de 4 lignes :

- La 1ère ligne est l'en-tête de séquence et commence par un @ : a commercial, suivi des identifiants et de la description de la séquence.
- La 2ème ligne représente le Read.
- La 3ème ligne commence par un « + » contiens la suite de description de la séquence, mais se trouve souvent vide.
- La 4ème ligne représente les scores de qualité de chaque nucléotide.

1 - https://bit.ly/33fdjEI

Tableau 1 : Données utilisées pour la phase post-génomique

Données	Caractéristiques
Extension	- fasta - fastq
Taille	- NC_007795.1.fasta (1 Ko) - NC_007795.1.fastq (11 Ko)

1.1.2. Données protéiques

Les données utilisées ont été récupérées sur le site Kaggle[2], qui est spécialisé dans les sciences des données et le DL. L'origine de ces données est la banque de données protéique PDB (Protein Data Bank).

Ces données se présentent en deux fichiers CSV (Comma-separated values), et contiennent beaucoup d'attributs. Ceux qui ont été utilisées sont : l'entrée, la séquence protéique et la classification (Voir le tableau 02).

pdb_data_no_dups.csv contient des métadonnées sur les protéines qui incluent des détails sur la classification des protéines, les méthodes d'extraction, etc.

pdb_data_seq.csv contient 346 321 séquences de structure protéique, ainsi que d'autres molécules.

Tableau 2 : Données utilisées pour l'apprentissage

Données	Caractéristiques
Extension	Comma-separated values, connu par le sigle CSV
Taille	- pdb_data_no_dups.csv (26,03 Mo) - pdb_data_seq.csv (120,14 Mo)

1.2. Configuration de la machine

La machine utilisée est un simple ordinateur, dont les caractéristiques sont détaillées dans le tableau 3 :

2 - https://www.kaggle.com/

Tableau 3 : Tableau représentant la configuration du matériel informatique utilisé lors de l'apprentissage

Ordinateur	Caractéristiques
Processeur	Intel i5-4670K @3.40GHz (4 Cores, 4 Threads)
Mémoire installée RAM	8 Go DDR3 @ 2133Hz
Stockage	Western Digital Blue Desktop 1 To SATA 6Gb/s 64 Mo
Système d'exploitation	Windows 10 professionnel
Type de système	Système d'exploitation 64 bits
Version du système	1903/ 18362.476

1.3. Outils et bibliothèques

1.1.1. Environnement de travail

– Python

Python est un langage de programmation open source créé par le programmeur Guido van Rossum en 1991. Il tire son nom de l'émission Monty Python's Flying Circus.

Il s'agit d'un langage de programmation interprété, qui ne nécessite donc pas d'être compilé pour fonctionner. Un programme interpréteur permet d'exécuter le code Python sur n'importe quel ordinateur. Ceci permet de voir rapidement les résultats d'un changement dans le code. En revanche, ceci rend ce langage plus lent qu'un langage compilé comme le C.

En tant que langage de programmation de haut niveau, Python permet aux programmeurs de se focaliser sur ce qu'ils font plutôt que sur la façon dont ils le font. Ainsi, écrire des programmes en python prend moins de temps que dans un autre langage ("Welcome to Python.org," 2020).

– Anaconda

Anaconda est une distribution adaptée pour Windows, Linux et MacOS qui est libre et open source des langages de programmation Python et R appliqué au développement d'applications dédiées à la science des données et au ML (traitement de données à grande échelle, analyse prédictive, calcul scientifique), qui vise à simplifier la gestion des paquets et de déploiement ("Anaconda Navigator — Anaconda documentation," 2020).

– Jupyter notebook

Jupyter Notebook est une application Web open source qui permet de créer et de partager des documents contenant du code en direct, des équations, des visualisations et du texte narratif. Les utilisations comprennent: le nettoyage et la transformation des données, la simulation

numérique, la modélisation statistique, la visualisation des données, le ML et bien plus encore (“Project Jupyter,” 2020).

1.1.2. Bibliothèques Python utilisées

– Tkinter

Le package tkinter (Tk) est l'interface graphique de Python standard de la boîte à outils Tk GUI. Ils sont disponibles sur la plupart des plates-formes Unix, ainsi que sur les systèmes Windows (“Tkinter,” 2020).

– Biopython

Biopython est le package de bioinformatique le plus grand et le plus populaire pour Python. Il contient un grand nombre de modules pour les différentes tâches bioinformatiques courantes, tel que les alignements d’ADN, d’ARN et de protéines, ou la recherche de motifs dans les séquences protéiques…(“Biopython,” 2020).

– NumPy

NumPy est un projet open source visant à permettre le calcul numérique avec Python. Il a été créé en 2005, en s'appuyant sur les premiers travaux des bibliothèques Numerical et Numarray. Il sera toujours 100% open source, gratuit pour tous. Il est développé à l'air libre sur GitHub, grâce au consensus de NumPy et de la communauté scientifique Python au sens large (“NumPy,” 2020).

– Pandas

Pandas est une bibliothèque écrite pour le langage de programmation Python permettant la manipulation et l'analyse des données. Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles (“pandas - Python Data Analysis Library,” 2020).

– Matplotlib

Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous formes de graphiques. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy. Matplotlib est distribuée librement et gratuitement (“Matplotlib,” 2020).

– Sklearn

Sklearn est un module Python intégrant des algorithmes de ML classiques dans le monde très uni des packages scientifiques Python (numpy, scipy, matplotlib). Il vise à apporter des solutions simples et efficaces aux problèmes d'apprentissage, accessibles à tous et réutilisables

dans divers contextes : le ML en tant qu'outil polyvalent pour la science et l'ingénierie (“scikit-learn: machine learning in Python,” 2020).

Les fonctionnalités fournies par scikit-learn incluent :

- Régression, y compris la régression linéaire et logistique
- Classification, y compris K-Nearest Neighbours
- Clustering, y compris K-Means et K-Means ++
- Prétraitement, y compris la normalisation Min-Max

– TensorFlow

TensorFlow est une plate-forme Open Source dédiée au ML et au DL. Elle propose un écosystème complet et flexible d'outils, de bibliothèques et de ressources communautaires permettant aux chercheurs d'avancer dans le domaine de l’IA, et aux développeurs de créer et de déployer facilement des applications qui exploitent cette technologie (“TensorFlow,” 2020).

Le tableau suivant (Tableau 4) représente les différents outils et bibliothèques avec les versions utilisées pour notre travail.

Tableau 4 : Caractéristiques des différents outils informatiques utilisés

Outils / bibliothèques	Versions
Biopython	biopython 1.78
Anaconda	anaconda 3 -2020.07
Jupyter Notebook	jupyter 6.0.3
Python	python 3.7.5
Tensorflow	tensorflow 2.2.0
Pandas	pandas 1.0.5
Numpy	numpy 1.18.5
Matplotlib	matplotlib 3.2.2
Sklearn	scikit-learn 0.23.1
Tkinter	Tkinter 8.6

2. MÉTHODES

Cette partie décrit les méthodes utilisées afin d'aboutir à l'objectif de l'approche.

2.1. Description globale

L'approche globale peut se diviser en deux phases distinctes :

2.1.1. Phase post-génomique

Le fichier fasta récupéré sur GenBank a été fragmenté par un code python, simulant la méthode de fragmentation. La méthode utilisée pour assembler un fichier fastq est l'algorithme Greedy SCS (Voir figure 12). Le fichier fastq doit être chargé et lu par l'implémentation de l'algorithme écrite en python. Seules les séquences nucléiques seront prises en considération par le programme d'assemblage. Le programme compare chaque Read avec tous les Reads du fichier fastq afin d'identifier les contigs par chevauchement. Puis il combine les fragments ayant les plus longs chevauchements, et ainsi de suite jusqu'à l'obtention de la séquence complète.

Ensuite, la séquence nouvellement assemblée est traduite en séquence protéique pour prédire sa classification en utilisant l'approche mentionnée dans la phase 2 du travail.

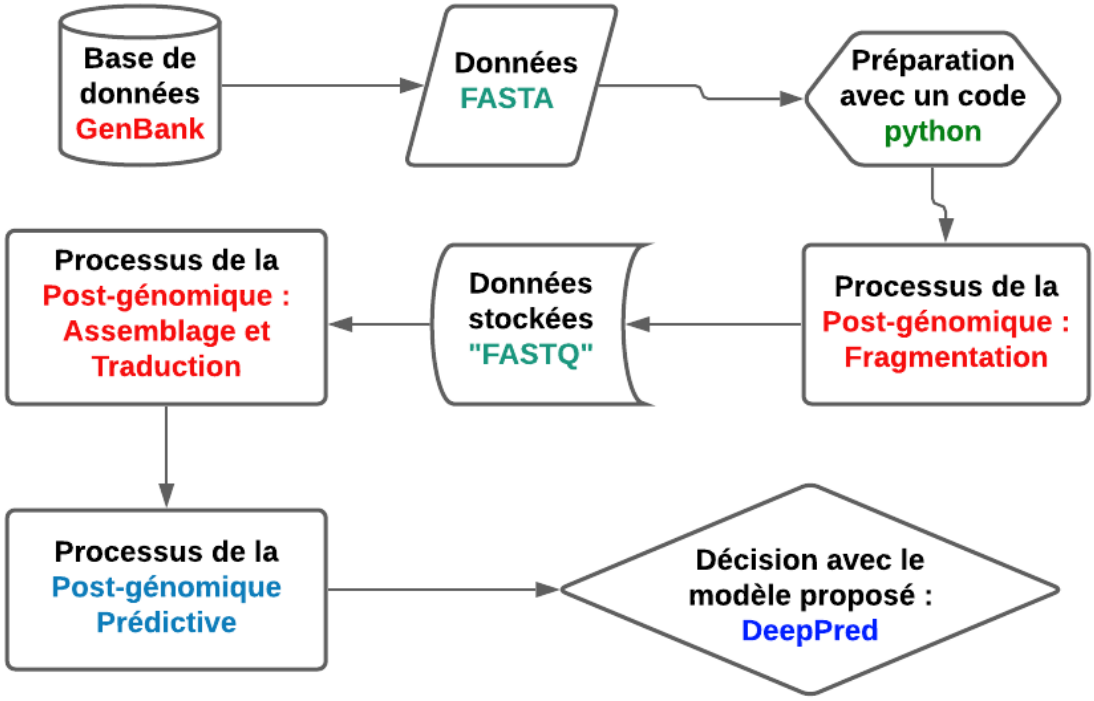

Figure 12 : Processus de l'approche DeepPred

2.1.2. Phase post-génomique prédictive

Dans cette section, nous présentons l'approche globale, qui est présentée également dans la (Figure 13) :

1) Prétraitement des données

Les données utilisées ont été récupérées sur le site web Kaggle, sous forme de 2 fichiers csv. Les fichiers ont été fusionné selon l'entrée, pour avoir toutes les informations d'une entrée sur le même enregistrement. Ensuite les données ont été nettoyées en supprimant les enregistrements n'étant pas des protéines puis en supprimant que les entrées ayant la séquence protéique ou la classification manquante.

2) Apprentissage

Afin de pouvoir procéder à l'entrainement, les données doivent être transformé, c'est-à-dire convertir les séquences protéiques et leurs classes en données numériques traitables par le réseau de neurones convolutif. Ensuite ces données sont divisées en deux échantillons, le 1er pour l'apprentissage et il représente 80 % de données, et le second pour l'évaluation du modèle et qui représente les 20% restantes. Ensuite, le modèle est créé, entrainé et évaluer, et on modifie à chaque fois les paramètres du modèle jusqu'à l'obtention d'un résultat satisfaisant. Une fois les paramètres optimaux du modèle trouvé, le modèle est enregistré.

3) Prédiction

Une fois le modèle enregistré, il suffit de le charger pour faire des prédictions des classes protéiques, y compris celle de la protéine traduite à partir du gène qu'on a assemblé.

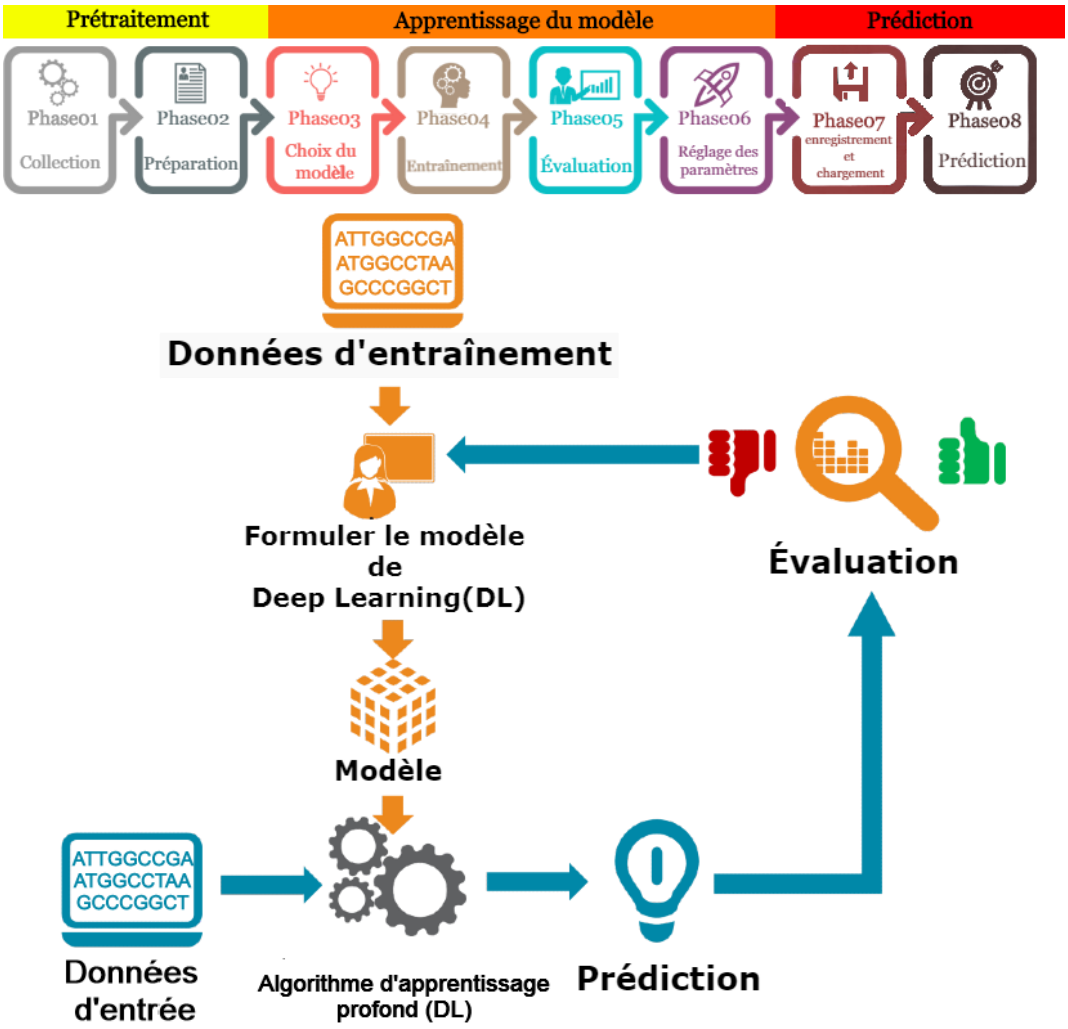

Figure 13 : Processus de l'approche globale

2.2. Description détaillée des méthodes

L'approche proposé distingue deux phase, la 1ère phase post-génomique composé de fragmentation, d'assemblage et de traduction. La 2ème est la phase post-génomique prédictive.

2.2.1. Phase post-génomique

La phase post-génomique se compose des cinq étapes suivantes :

1) Fragmentation d'une séquence nucléique

Afin de simuler le processus de fragmentation, on récupère une séquence nucléique sur le banque de données GenBank. Le fichier étant sous format fasta. La fragmentation se fait par un script python (voir figure 14), générant un fichier fastq semblable à celui généré par les NGS.

```
import sys
fastqname = input ("Tapez le nom du fichier output '.fastq': \n")
file_name=open('fasta.fasta','r')
read=file_name.read()
...
...
Reads=[]
while j < len(read)+1:
    ...
    ...
    aa1=Reads.append(read[i+0:j])
    aa2=Reads.append(read[k1+0:k2])
    aa3=Reads.append(read[h1+0:h2])
    i=j
    j=j+100
    ...
    ...
    k1=k2
    k2=k2+100

sys.stdout = open(fastqname, 'w')

for i in Reads:
    print("@")
    print(i)
    print("+")
    print("score")

sys.stdout.close()
```

Figure 14 : Code python effectuant la Fragmentation

2) Chargement du fichier fastq

Voici le script python permettant le chargement du fichier fastq (Voir figure 15)

```
filename = "NC_007795.1.fastq"
reads = []
with open(filename) as fh:
    while True:
        fh.readline()
        seq = fh.readline().rstrip()
        fh.readline()
        fh.readline()
        if len(seq) == 0:
            break
        reads.append(seq)
```

Figure 15 : Aperçu du code permettant l'ouverture du fichier fastq

3) Fonction Overlap

Cette fonction va calculer la longueur du chevauchement entre deux séquences avec un minimum de longueur d'un caractère. La fonction Find de python sera utilisée. Overlap renvoie un 0 s'il n'y a pas de chevauchement, sinon elle renvoie la longueur du chevauchement maximal (Voir figure 16).

```
def overlap(a, b, min_length=1):
    start = 0
    while True:
        start = a.find(b[:min_length], start)
        if start == -1:
            return 0
        if b.startswith(a[start:]):
            return len(a)-start
        start += 1
```

Figure 16 : Aperçu du code de la fonction cherchant les tailles de chevauchements

4) Fonction allOverlapPairs

Cette fonction se charge de rassembler par paires les Reads ayant les plus longs chevauchements (Voir figure 17).

```
def allOverlapPairs(reads, k=30, min_overlap=1):
    kIndex = {}
    pairs = []

    for read in reads:
        for i in range(len(read) - k + 1):
            if kIndex.get(read[i:i+k]) == None:
                kIndex[read[i:i+k]] = set([read])
            else:
                kIndex[read[i:i+k]].add(read)
    for read in reads:
        for candidate in kIndex[read[-k:]]:
            if candidate != read:
                olen = overlap(read, candidate, min_overlap)
                if olen >= min_overlap:
                    pairs.append([olen, read, candidate])
    pairs.sort(reverse=True)
    return pairs
```

Figure 17 : Aperçu du code de la fonction allOverlapPairs

5) Assemblage

Voici le Script python (Voir figure 18) responsable de l'assemblage des Reads récupérés par la fonction allOverlapPairs, il assemble les Reads se chevauchant jusqu'à l'assemblage complet du gène.

```
k=30
min_overlap =1
pairs = allOverlapPairs(reads, k, min_overlap)
olen, reada, readb = pairs[0][0], pairs[0][1], pairs[0][2]

while olen > 0:
    reads.remove(reada)
    reads.remove(readb)
    reads.append(reada + readb[olen:])

    pairs = allOverlapPairs(reads, k, min_overlap)
    if len(pairs) == 0:
        print("No more overlapping pairs remain.")
        break
    olen, reada, readb = pairs[0][0], pairs[0][1], pairs[0][2]

genome_readsIndexed = ''.join(reads)
```

Figure 18 : Code python effectuant l'assemblage des Reads

2.2.2. Phase post-génomique prédictive

Cette phase distingue trois étapes principales :

1) Prétraitement des données

Pour que l'apprentissage puisse avoir lieu, il faut que les données soient correctement préparées.

i. Chargement des bibliothèques python nécessaires

Pour initier le travail, les commandes affichées dans la (Figure 19) importent les bibliothèques nécessaires.

```
import tensorflow as tf
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing import text, sequence
```

Figure 19 : Chargement des bibliothèques python nécessaires

ii. Chargement des données

Le chargement des données consiste à ouvrir les fichiers contenant les données nécessaires au prétraitement, la fonction read_csv de la bibliothèque pandas permet ce chargement, comme mentionné dans la (Figure 20).

```
df_seq = pd.read_csv('pdb_data_seq.csv')
df_char = pd.read_csv('pdb_data_no_dups.csv')
```

Figure 20 : Chargement des données à partir des fichier CSV

iii. Exploration et visualisations des données avec pandas

Etant donné que les entrées nécessaires pour l'apprentissage doivent être des séquences protéiques, il faut procéder à un filtrage pour supprimer toute autres molécules, comme le montre le code dans la (Figure 21).

```
protein_char = df_char[df_char.macromoleculeType == 'Protein']
protein_seq = df_seq[df_seq.macromoleculeType == 'Protein']
```

Figure 21 : Filtrage et traitement des données chargées

Seuls les attributs nécessaires à l'apprentissage dans chacun des deux fichiers doivent être chargés, les autres doivent être supprimés (Voir figure 22).

```
protein_char = protein_char[['structureId','classification']]
protein_seq = protein_seq[['structureId','sequence']]
```

Figure 22 : Sélection les attributs nécessaires à l'apprentissage

Joindre les deux ensembles de données dans une seule DataFrame selon leurs 'structureId' qui représente l'entrée de chaque enregistrement. (Voir Figure 23)

```
df = protein_char.set_index('structureId').join(protein_seq.set_index('structureId'))
df.head()
```

Figure 23 : Fusion des deux ensembles de données dans une seule DataFrame

Visualisation d'une partie des données afin de s'assurer du résultat de la fusion des deux ensembles de données (Voir figure 24).

structureId	classification	sequence
101M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
102L	HYDROLASE(O-GLYCOSYL)	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNAAAKSE...
102M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...
103L	HYDROLASE(O-GLYCOSYL)	MNIFEMLRIDEGLRLKIYKDTEGYYTIGIGHLLTKSPSLNSLDAAK...
103M	OXYGEN TRANSPORT	MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDR...

Figure 24 : Visualisation d'une partie des données après la fusion

iv. Préparation des données

L'apprentissage nécessite deux attributs uniquement : la séquence et sa classification. Par conséquent il faut procéder à la suppression des enregistrements avec des valeurs manquantes (voir figure 25).

```
df = df.dropna()
print('%d is the number of proteins that have a classification and sequence' %df.shape[0])
```

Figure 25 : Suppression des enregistrements contenant des valeurs manquantes

Résultat : Le nombre de protéines qui ont une classification et une séquence est 346321.

Ensuite, il faudra compter le nombre de protéines appartenant à chaque classification (Voir figure 26).

```
counts = df.classification.value_counts()
print(counts)
```

Figure 26 : Calcul du nombre des classes protéiques

La figure 27 représente le nombre de protéine des différentes classes protéiques.

```
HYDROLASE                          46336
TRANSFERASE                        36424
OXIDOREDUCTASE                     34321
IMMUNE SYSTEM                      15615
LYASE                              11682
                                   ...
CARBOHYDRATE KINASE                    1
ATTRACTIN                              1
RECOMBINATION, REPLICATION             1
PROHORMONE                             1
YJQ8WW DOMAIN                          1
Name: classification, Length: 4468, dtype: int64
```

Figure 27 : Nombres des protéines de chaque classe

Les données sont filtrées pour ne garder que les classes ayant 1000 protéines ou plus, ce qui nous donne 43 familles (Voir figure 28).

```
# Get classification types where counts are over 1000
types = np.asarray(counts[(counts > 1000)].index)

# Filter dataset's records for classification types > 1000
df = df[df.classification.isin(types)]
```

Figure 28 : Filtrage des enregistrements

Le nombre final de protéines avec lesquelles se fera l'apprentissage est de 278866 protéines.

La figure 29 est une visualisation d'une partie échantillon des cinq premiers enregistrements afin de s'assurer des résultats du filtrage :

structureId	classification	sequence
10GS	TRANSFERASE/TRANSFERASE INHIBITOR	PPYTVVYFPVRGRCAALRMLLADQGQSWKEEVVTVETWQEGSLKAS...
10GS	TRANSFERASE/TRANSFERASE INHIBITOR	PPYTVVYFPVRGRCAALRMLLADQGQSWKEEVVTVETWQEGSLKAS...
117E	HYDROLASE	TYTTRQIGAKNTLEYKVYIEKDGKPVSAFHDIPLYADKENNIFNMV...
117E	HYDROLASE	TYTTRQIGAKNTLEYKVYIEKDGKPVSAFHDIPLYADKENNIFNMV...
11AS	LIGASE	MKTAYIAKQRQISFVKSHFSRQLEERLGLIEVQAPILSRVGDGTQD...

Figure 29 : Visualisation d'une partie des données

Visualisation de deux graphes : le 1er représente le nombre de séquences par famille, le second représente le nombre de séquences par rapport à leurs tailles (Voir les figures 30 et 31).

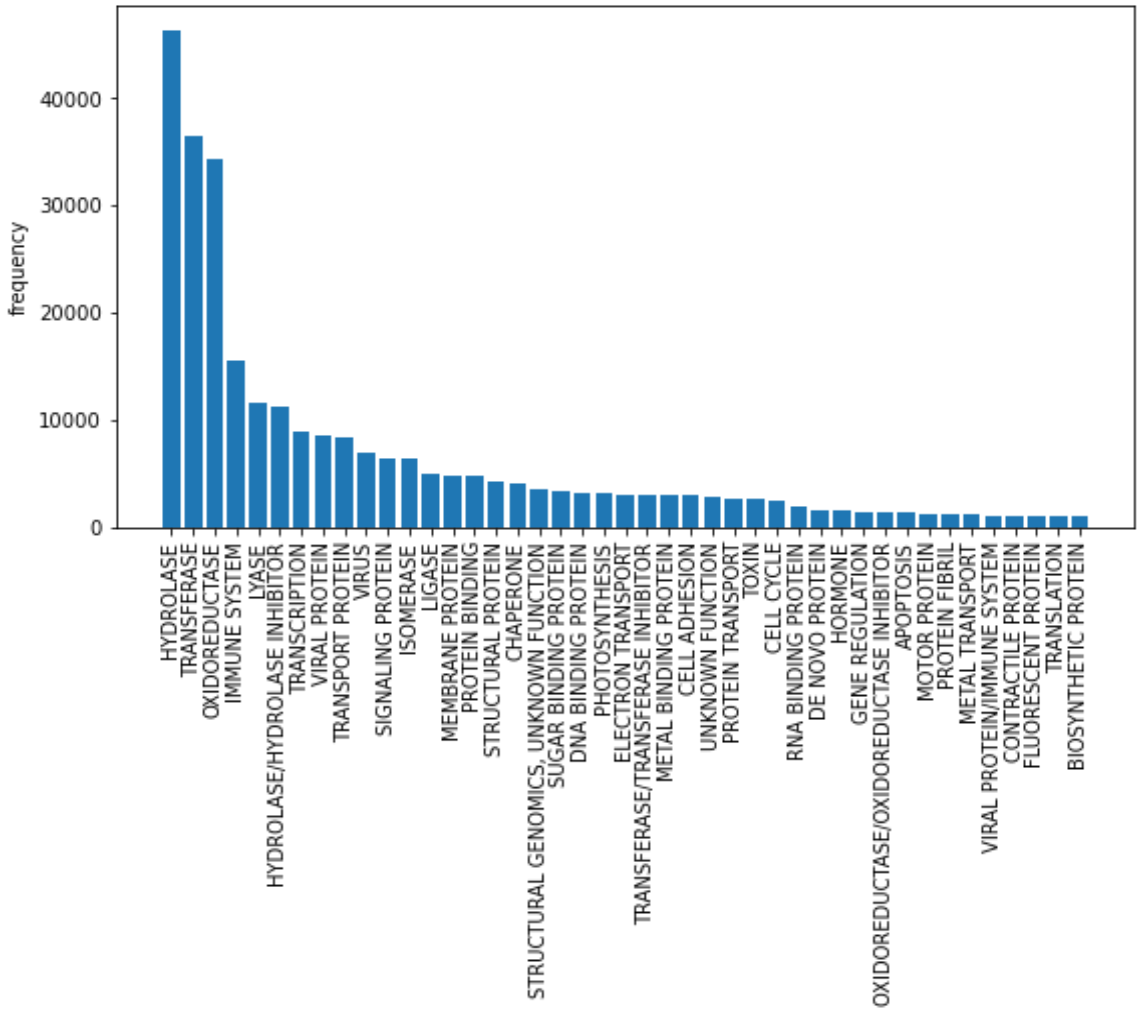

Figure 30 : Représentation graphique du nombre de séquences par famille

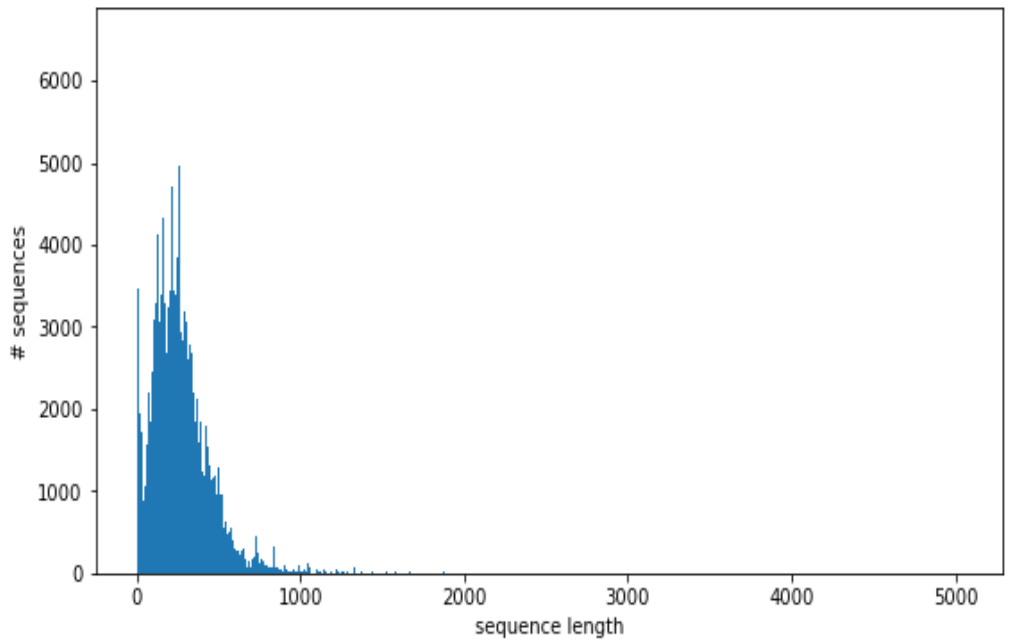

Figure 31 : Représentation graphique du nombre des séquences par rapport à leurs tailles

2) Apprentissage

Cette étape permet l'entrainement du modèle sur les données.

i. Transformation des données

Stockage des classes protéiques du DataFrame dans une matrice numpy afin de pouvoir les numériser et les préparer au processus du DL, en utilisant la fonction LabelBinarizer de la bibliothèque sklearn (Voir figure 32).

```
from sklearn.preprocessing import LabelBinarizer
classes = np.array(df.classification)
# Transform labels to one-hot
label_binarizer = LabelBinarizer()
Y = label_binarizer.fit_transform(classes)
```

Figure 32 : Transformation des classes en utilisant LabelBinarizer

Stockage des séquences protéiques du DataFrame dans une matrice numpy (voir figure 33) afin de pouvoir les numériser et les préparer au processus du DL, en utilisant la fonction Tokenizer de la bibliothèque Tensorflow (Voir figure 34).

```
sentences = np.array(df['sequence'])
```

Figure 33 : Stockage des données du DataFrame dans une matrice numpy

```
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
X = tokenizer.texts_to_sequences(sentences)
X = sequence.pad_sequences(X, maxlen=max_length)
```

Figure 34 : Transformation des séquences protéique en utilisant la fonction Tokenizer

ii. Fractionnement des données pour l'apprentissage et l'évaluation

Les données (278866 protéines) doivent être en deux parties, la 1ère représente 80% des données, servira à entraîner le modèle, et la 2nde représente les 20% restantes servira à tester le modèle et évaluer sa précision.

La fonction train_test_split de la bibliothèque sklearn, est utilisée pour diviser les données correctement, et que les entrées des 2 parties soient prisent d'une façon aléatoire (Voir figure 35).

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=.2)
```

Figure 35 : Fractionnement des données via la fonction train_test_split

Le nombre de séquences protéiques qui serviront de données à l'apprentissage est 223093, et les 55773 des séquences restantes serviront à l'évaluation du modèle.

iii. Création du modèle et du réseau des neurones convolutif

Le modèle défini est un modèle TensorFlow Keras séquentiel tf.keras.Sequential. Il se compose d'une couche conv1D, dédié aux traitements des vecteurs et à l'extraction des caractéristiques, suivie d'une couche MaxPolling1D dont le but est de réduire la taille des caractéristiques apprises, les consolidant uniquement aux éléments les plus essentiels. Après les couches conv1D et MaxPooling1D, les caractéristiques apprises sont aplaties en un vecteur long et passent à travers une couche fully-connected avant la couche de sortie utilisée pour faire la prédiction. La couche fully-connected fournit idéalement un tampon entre les caractéristiques apprises et la sortie dans le but d'interpréter les caractéristiques apprises avant de faire une prédiction. La version Adam de la descente de gradient stochastique sera utilisée pour optimiser le réseau, et la fonction de perte Categorical_crossentropy sera utilisée étant donné que le problème traité est une classification multi-classes. (Voir figure 36)

Le modèle se compose de six couches:

```
embedding_dim = 32
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(len(tokenizer.word_index)+1,
                              embedding_dim, input_length=max_length),
    tf.keras.layers.Conv1D(filters=128, kernel_size=2,
                           padding=padding_type, activation='relu'),
    tf.keras.layers.MaxPooling1D(pool_size=1),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(types), activation='softmax'),
])
model.compile(loss='categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])
print(model.summary())
```

Figure 36 : Création du modèle et d'un réseau de neurones convolutif

Lors de la compilation, le modèle vérifiera que les options choisies sont compatibles les unes avec les autres. (Voir figure 37)

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 1024, 32)          832
_________________________________________________________________
conv1d (Conv1D)              (None, 1024, 128)         8320
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 1024, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 131072)            0
_________________________________________________________________
dense (Dense)                (None, 128)               16777344
_________________________________________________________________
dense_1 (Dense)              (None, 43)                5547
=================================================================
Total params: 16,792,043
Trainable params: 16,792,043
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 37 : Récapitulation du réseau de neurone convolutif (RNC)

iv. Apprentissage et évaluation du modèle

Le modèle peut désormais être instancié et entraîné. La fonction fit est utilisé pour lancer l'apprentissage. (Voir figure 38)

```
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
                    epochs=50, batch_size=128)
```

Figure 38 : Utilisation de la fonction fit pour entrainer le modèle

Et voilà l'entrainement débute: (Voir figure 39)

```
Epoch 1/50
1743/1743 [==============================] - 811s 465ms/step - loss: 1.4084
- accuracy: 0.6385 - val_loss: 0.9309 - val_accuracy: 0.7581
Epoch 2/50
1743/1743 [==============================] - 817s 469ms/step - loss: 0.6795
- accuracy: 0.8185 - val_loss: 0.7364 - val_accuracy: 0.8162
Epoch 3/50
1743/1743 [==============================] - 821s 471ms/step - loss: 0.4768
- accuracy: 0.8672 - val_loss: 0.6658 - val_accuracy: 0.8426
Epoch 4/50
1743/1743 [==============================] - 803s 461ms/step - loss: 0.3808
- accuracy: 0.8918 - val_loss: 0.6340 - val_accuracy: 0.8507
Epoch 5/50
1743/1743 [==============================] - 820s 471ms/step - loss: 0.3256
- accuracy: 0.9054 - val_loss: 0.6562 - val_accuracy: 0.8546
```

Figure 39 : Premières itérations de l'apprentissage du modèle

Une fois le modèle entraîné, et vu le code qu'on a utilisé il affichera, en plus des résultats de son entrainement, les résultats de son évaluation.

v. Enregistrement

Le modèle est enregistré via l'une des fonctions que TensorFlow fournit. (Voir figure 40)

```
# Save the entire model as a SavedModel.
model.save('my_model.h5')
#model.save('saved_model/my_model')
```

Figure 40 : Enregistrement du modèle

vi. Visualisation des résultats

Pour afficher les résultats, les 2 bibliothèques Matplotlib & sklearn.metrics ont été utilisées :

– Matplotlib

La fonction plt de matplotlib offre un bon affichage permettant de visualiser l'état d'avancement de l'apprentissage. Le code utilisé est dans la (figure 41).

```
import matplotlib.pyplot as plt

def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```

Figure 41 : Affichage des données de résultats avec Matplotlib

L'affichage des résultats des données est dans le chapitre suivant : Résultats et discussion.

– Sklearn

Les fonctions confusion_matrix, accuracy_score et classification_report de sklearn couplées à plt de matplotlib offrent, un excellent outil pour visualiser les résultats d'évaluation d'un modèle de classification qui est la matrice de confusion (Voir figure 42 et figure 43).

– Importation des bibliothèques

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

Figure 42 : Chargement des bibliothèques pour afficher la matrice de confusion

– Visualisation de la matrice de confusion

```
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)
print("train-acc = " + str(accuracy_score(np.argmax(y_train, axis=1),
                                          np.argmax(train_pred, axis=1))))
print("test-acc = " + str(accuracy_score(np.argmax(y_test, axis=1),
                                         np.argmax(test_pred, axis=1))))

# Compute confusion matrix
cm = confusion_matrix(np.argmax(y_test, axis=1),
                      np.argmax(test_pred, axis=1))

# Plot normalized confusion matrix
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
np.set_printoptions(precision=2)
plt.figure(figsize=(10,10))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(label_binarizer.classes_))
plt.xticks(tick_marks, label_binarizer.classes_, rotation=90)
plt.yticks(tick_marks, label_binarizer.classes_)
#for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
#    plt.text(j, i, format(cm[i, j], '.2f'), horizontalalignment="center", color="white"
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

print(classification_report(np.argmax(y_test, axis=1),
                            np.argmax(test_pred, axis=1),
                            target_names=label_binarizer.classes_))
```

Figure 43 : Code d'affichage de la matrice de confusion

3) Prédiction

La prédiction représente l'étape ou le modèle devient fonctionnelle.

i. Chargement du modèle

Il est possible désormais de charger directement le modèle. La fonction load_model de tensorflow permet un chargement rapide du modèle (Voir figure 44).

```
model = tf.keras.models.load_model('my_model.h5')
#model = tf.keras.models.load_model('saved_model/my_model')
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 1024, 32)          832
_________________________________________________________________
conv1d (Conv1D)              (None, 1024, 128)         8320
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 1024, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 131072)            0
_________________________________________________________________
dense (Dense)                (None, 128)               16777344
_________________________________________________________________
dense_1 (Dense)              (None, 43)                5547
=================================================================
Total params: 16,792,043
Trainable params: 16,792,043
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 44 : Affichage du modèle chargé

ii. Prédiction en utilisant le modèle

Le modèle est désormais prêt à être utilisé pour prédire les fonctions des protéines nouvellement séquencées (Voir figure 45).

```
Protein = ["MTNKVVILGSTNVDQFLTVERYAQPGETLHVEEAQKAFGGGKGANQAIATARMQADTTFITKIGTDGVADFILED
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(Protein)
tokenizer.word_index = {'l': 1,'a': 2,'g': 3,'v': 4,'e': 5,'s': 6,'d': 7,'t': 8,'i': 9,
X = tokenizer.texts_to_sequences(Protein)
X = sequence.pad_sequences(X, maxlen=max_length)
prediction = model.predict(X)

result = pd.DataFrame(prediction)
pd.set_option('display.max_columns', 500)
result=result.apply(lambda x:round(x,2))
classes = ['APOPTOSIS', 'BIOSYNTHETIC PROTEIN', 'CELL ADHESION', 'CELL CYCLE',
       'CHAPERONE', 'CONTRACTILE PROTEIN', 'DE NOVO PROTEIN',
       'DNA BINDING PROTEIN', 'ELECTRON TRANSPORT', 'FLUORESCENT PROTEIN',
       'GENE REGULATION', 'HORMONE', 'HYDROLASE',
       'HYDROLASE/HYDROLASE INHIBITOR', 'IMMUNE SYSTEM', 'ISOMERASE',
       'LIGASE', 'LYASE', 'MEMBRANE PROTEIN', 'METAL BINDING PROTEIN',
       'METAL TRANSPORT', 'MOTOR PROTEIN', 'OXIDOREDUCTASE',
       'OXIDOREDUCTASE/OXIDOREDUCTASE INHIBITOR', 'PHOTOSYNTHESIS',
       'PROTEIN BINDING', 'PROTEIN FIBRIL', 'PROTEIN TRANSPORT',
       'RNA BINDING PROTEIN', 'SIGNALING PROTEIN',
       'STRUCTURAL GENOMICS, UNKNOWN FUNCTION', 'STRUCTURAL PROTEIN',
       'SUGAR BINDING PROTEIN', 'TOXIN', 'TRANSCRIPTION', 'TRANSFERASE',
       'TRANSFERASE/TRANSFERASE INHIBITOR', 'TRANSLATION',
       'TRANSPORT PROTEIN', 'UNKNOWN FUNCTION', 'VIRAL PROTEIN',
       'VIRAL PROTEIN/IMMUNE SYSTEM', 'VIRUS']
```

Figure 45 : Prédiction en utilisant le modèle

2.2.3.Création d'une interface graphique avec tkinter

Pour faciliter l'utilisation du modèle de DeepProt, une interface graphique a été programmée avec tkinter (voir la figure 46).

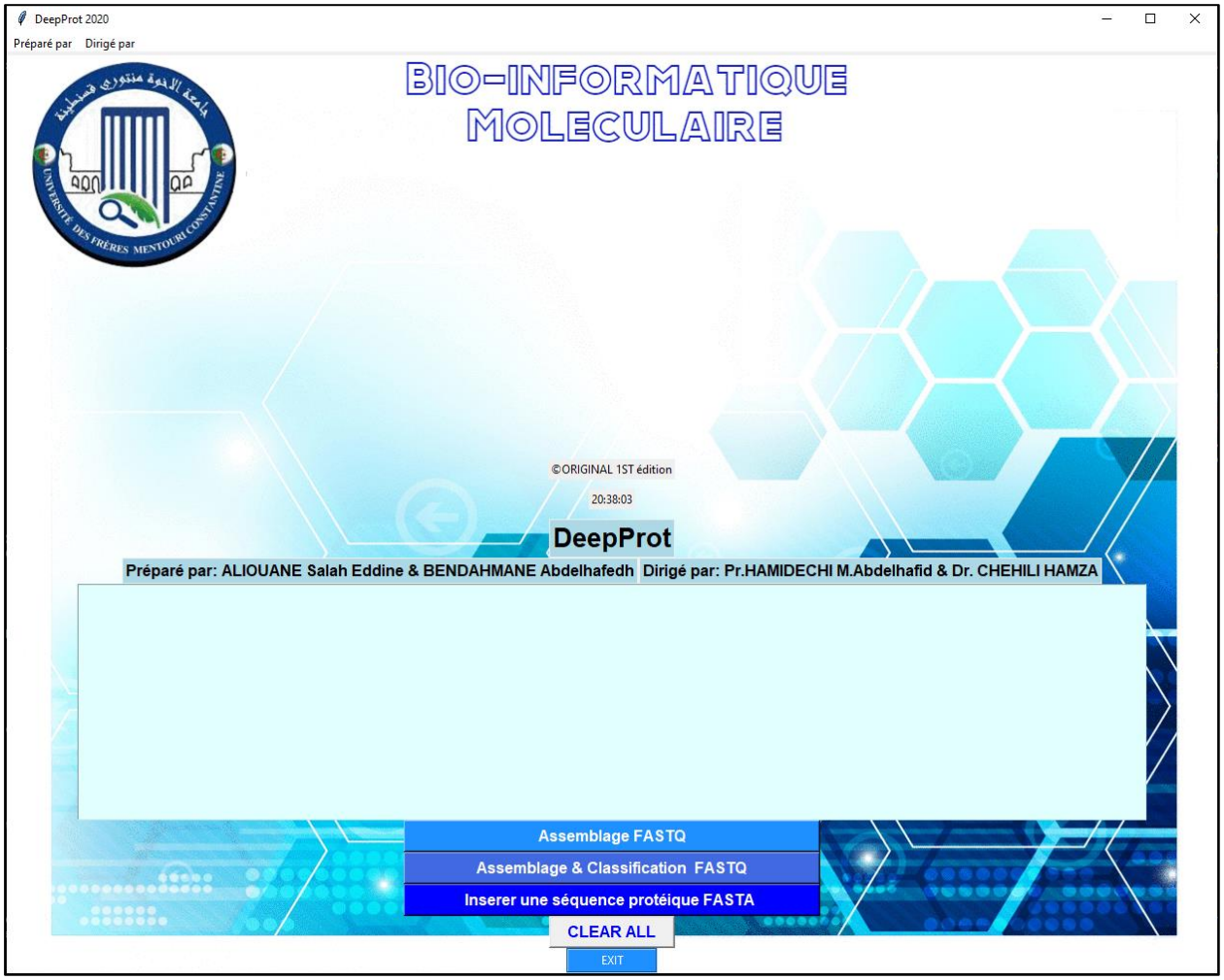

Figure 46 : Représentation de l'interface graphique de DeepProt

PARTIE 3 :

RÉSULTATS ET DISCUSSION

RÉSULTATS

Ce projet est divisé en deux étapes : la première partie consiste à l'implémentation de l'algorithme Greedy SCS afin d'assembler les Reads d'un gène puis traduire ce gène en une protéine. La seconde partie consiste à prédire la classification de la protéine traduite, et c'est dans cette partie que la contribution a été apportée.

Résultats de la phase 01: Fragmentation, Assemblage et Traduction

Dans cette étape, le résultat de fragmentation d'une séquence nucléique à partir d'un fichier fasta sera présenté (voir figure 47), ensuite le résultat d'assemblage du gène via le programme utilisant l'algorithme Greedy SCS sera présenté, suivi de sa traduction en protéine. La protéine traduite sera ensuite utilisée dans la seconde phase afin de prédire sa classification via le modèle de prédiction créé.

Après l'exécution de l'algorithme implémenté, la séquence assemblée est affichée. (Voir figure 47).

```
print(fastq)

@r0
TTAGAAGTACCCATCCCGGCTATTATATCTGCATTTGAAATTGCCAAGGCACATGGTGTGACGACAGTATTAAATCCTGCACCAGCGAAAGCATTACCTA
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@r1
TCATATTGATACATCATATATTATCAAAACAGCTGAAGCAAAAACGGGCCAAGCCTTTATCACTGTGAATGCAGAAGGACAAAACACCATCTATGTTTAT
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@r2
TTATATCTGCATTTGAAATTGCCAAGGCACATGGTGTGACGACAGTATTAAATCCTGCACCAGCGAAAGCATTACCTAATGAATTATTATCATTAATCGA
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
@r3
TTTTTATCTATAGGCATTAAGACTGTTTTGATTACGCTAGGTAAGCAAGGTACATATTTTGCTACTAAAAATCAAAGCCAACACATCGAAGCTTATAAAG
+
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII

print(genome_readsIndexed)

ATGACCAACAAAGTTGTTATTTTAGGTTCAACGAATGTCGATCAATTTTTAACAGTTGAAAGATATGCACAACCAGGCGAAACATTACATGTTGAAGAAGC
GGAGGTAAAGGTGCCAACCAGGCTATTGCCACTGCACGCATGCAAGCAGACACAACATTTATTACTAAAATTGGCACTGATGGCGTTGCTGATTTCATCTT
GCTCATATTGATACATCATATATTATCAAAACAGCTGAAGCAAAAACGGGCCAAGCCTTTATCACTGTGAATGCAGAAGGACAAAACACCATCTATGTTTA
ACGATGACACCTGAAGATGTTATTAACGCAAAAGACGCTATAATCAATGCAGACTTTGTCGTTGCACAATTAGAAGTACCCATCCCGGCTATTATATCTGC
GCACATGGTGTGACGACAGTATTAAATCCTGCACCAGCGAAAGCATTACCTAATGAATTATTATCATTAATCGATATTATTGTGCCAAACGAAACAGAAGC
ATTAAAGTAACTAATGAACAATCTATGAAAGACAATGCCAATTACTTTTTATCTATAGGCATTAAGACTGTTTTGATTACGCTAGGTAAGCAAGGTACATA
CAAAGCCAACACATCGAAGCTTATAAAGTAAATGCGATTGATACAACTGCTGCAGGCGACACATTTATTGGTGCATTTGTCAGTCGCTTAAACAAGTCGCA
GCTATTGATTTTGGTAATAAAGCGAGCTCACTCACTGTACAAAAACACGGCGCGCAAGCATCTATTCCTCTACTAGAAGAAGTAAATCAAGTTTAA
```

Figure 47 : Aperçu des fragments du fichier fastq et du gène assemblé

Afin de s'assurer de la précision de l'algorithme, on procède à une comparaison avec une séquence fasta contenant le gène de référence qu'on a utilisé pour générer le fichier fastq. La comparaison se fait grâce à une fonction python qui applique l'algorithme Needleman et Wunsch. Le score de match utilisé est 1, étant donné la longueur des 2 séquences qui est de 915 pb, le score calculé est maximal et par conséquent les 2 séquences sont identiques (Voir figure 48).

```
from Bio import pairwise2
from Bio.pairwise2 import format_alignment
print("la taille de la séquence assemblé est de : ", len(genome_readsIndexed))
print("la taille de la séquence de référence est de : ",len(gene_ref))
score = pairwise2.align.globalxx(genome_readsIndexed, gene_ref, score_only=True)
print("le score d'alignement global entre la séquence assemblée et la séquence de référence est de : ", score)
```

```
la taille de la séquence assemblé est de :  915
la taille de la séquence de référence est de :  915
le score d'alignement global entre la séquence assemblée et la séquence de référence est de :  915.0
```

Figure 48 : Code de l'alignement global qui vérifie le résultat de l'assemblage

Après la vérification du résultat de l'assemblage par alignement global en utilisant le Bio.pairwise2 de biopython, la séquence nucléique est traduite en protéine (voir figure 49)

```
from Bio.Seq import Seq
coding_dna = Seq(genome_readsIndexed)
Translated_DNA = (coding_dna.translate(to_stop=True))
Protein = ''
for i in range (len(Translated_DNA)):
    Protein = Protein + (Translated_DNA[i])
print(Protein)
```

Figure 49 : Code de la traduction du gène en protéine

Résultats de la phase 02: Prédiction de la classification

C'est cette phase qui représente la contribution apportée. Comme mentionné précédemment, une fois que le modèle termine la phase d'apprentissage, il faudra le tester sur un échantillon non étiqueté puis comparer les résultats avec les étiquettes qui conviennent à l'échantillon afin de s'assurer de son efficacité.

Résultats de test

Pour évaluer l'efficacité du modèle, il est primordial de calculer deux valeurs de précision:

La premier est la précision de l'entraînement du modèle appliqué aux 80% de l'échantillon des protéines et la seconde est la précision de l'entraînement du modèle appliqués aux 20% de l'échantillon de protéines laissé pour la phase d'évaluation, afin d'évaluer le modèle de classification des protéines.

Il faut noter que les taux de précision doivent être aussi élevé que possible.

Après avoir effectué les tests, voici les résultats obtenus:

- Train-acc = 0.9495454789952127 = **95%**
- Test-acc = 0.8794241044214149 = **88%**

Matrice de confusion

Nous avons utilisé la matrice de confusion (qui est une matrice qui mesure la qualité d'un système de classification) pour évaluer la qualité de la sortie du classificateur sur l'ensemble des données protéique afin de faciliter la lecture des résultats des tests obtenus. La ligne diagonale représente le nombre de points auxquels la classification attendue des protéines correspond à la classification réelle, tandis que les éléments n'appartenant pas à la ligne sont les protéines qui sont mal classées par le classificateur (Voir figure 50).

Plus les valeurs diagonales de la matrice de confusion sont élevées, mieux c'est, et cela prouve que la classification était efficace, indiquant de nombreuses prédictions correctes.

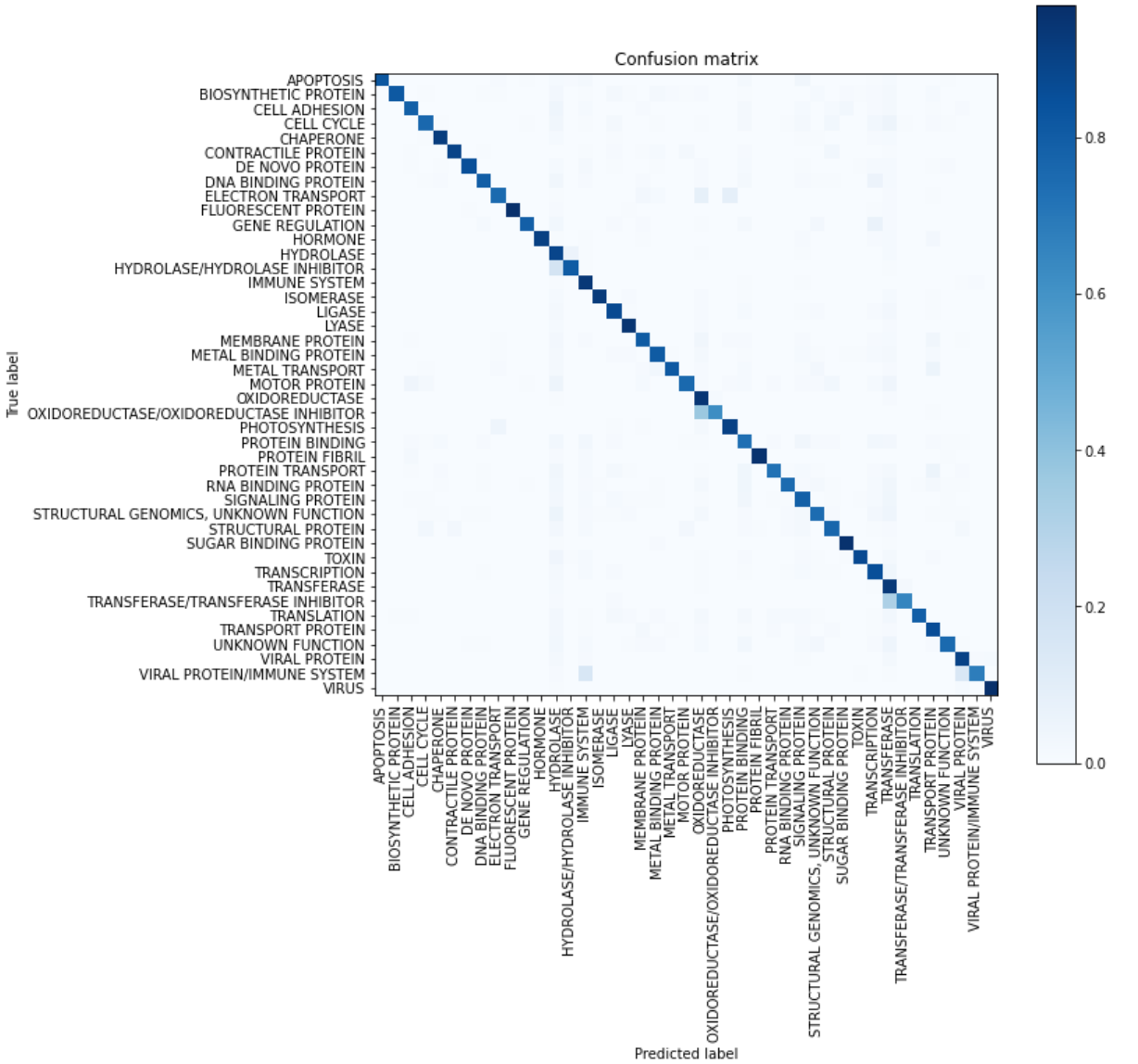

Figure 50 : Matrice de confusion obtenue après les tests

Temps d'exécution

Il est important de noter que dans tous les scores de performance qui ont été pris en compte dans le DL, en particulier en termes de temps et de résolution du modèle, seuls les temps d'exécution du noyau sont mesurés. Par conséquent, nous excluons le temps de chargement ou de compilation de ceux-ci ainsi que les transferts de mémoire.

Dans les tableaux 5 et 6 suivants, nous montrerons les temps itératifs du modèle lors de son apprentissage:

Tableau 5 : Durées des 15 premières répétitions de notre modèle pendant son apprentissage

Itération	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Temps (s)	813.466	829.476	809.464	822.472	812.466	804.461	911.522	823.472	859.493	829.476	802.460	807.463	860.493	808.463	807.463

Tableau 6 : Durées des 15 dernières répétitions de notre modèle pendant son apprentissage

Itération	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Temps (s)	832.477	833.478	809.464	805.462	848.487	808.464	803.461	814.467	830.476	794.455	795.456	791.454	794.456	798.458	799.458

À partir des deux tableaux, nous remarquons que le temps d'itération d'apprentissage de notre modèle sont très proches, car nous avons enregistré une moyenne de temps pour toutes les itérations: **818.7361** secondes, ce qui équivaut à **13** minutes et **38** secondes à titre approximatif pour chaque répétition.

Le graphe suivant (Voir figure 51) montre le changement de temps enregistré pendant l'entraînement du modèle et montre également la corrélation linéaire entre chaque répétition.

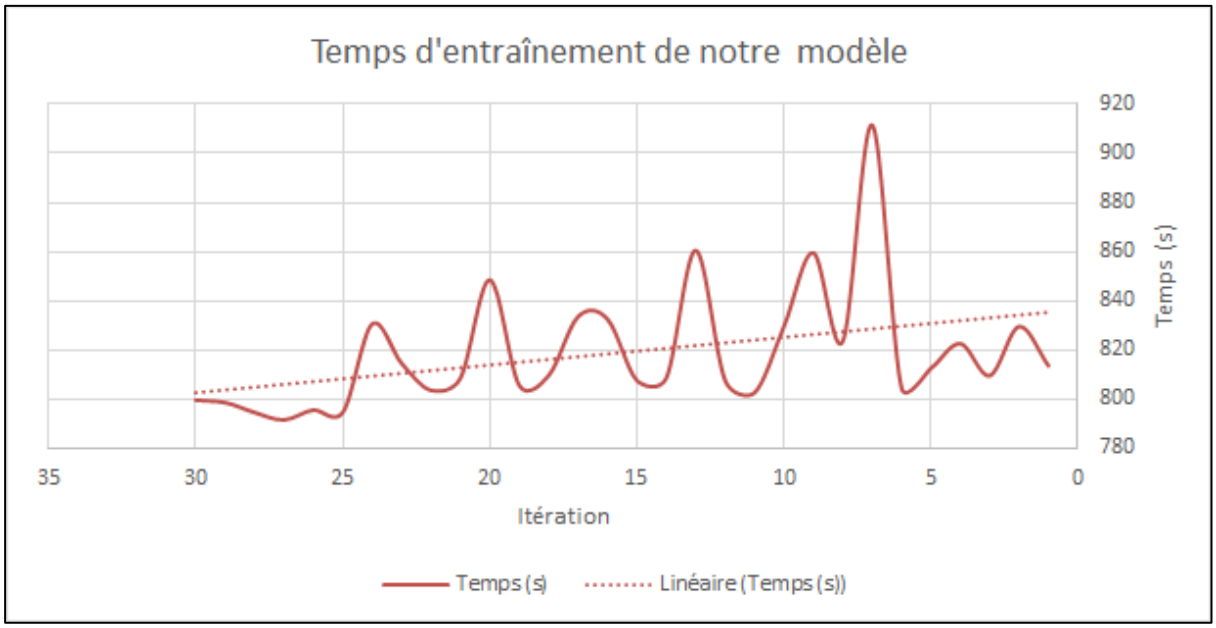

Figure 51 : Graphe montrant le temps de 30 itérations de notre modèle pendant l'entraînement

Enfin, on trace l'historique de la formation pour voir l'évolution, et pour comparer la formation et les prédictions de test.

À la fin des 30 époques, nous avons une précision pour l'ensemble d'apprentissage 0.9452 et 0.8794 pour l'ensemble d'essai. La perte de validation diminue à chaque époque, mais les résultats sont là (Voir les figure 52 et 53).

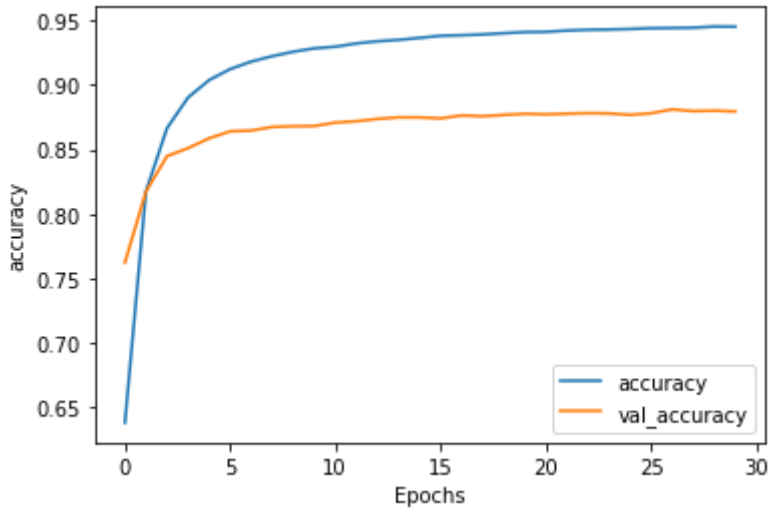

Figure 52 : Précision pour le premier classificateur pendant 30 itérations

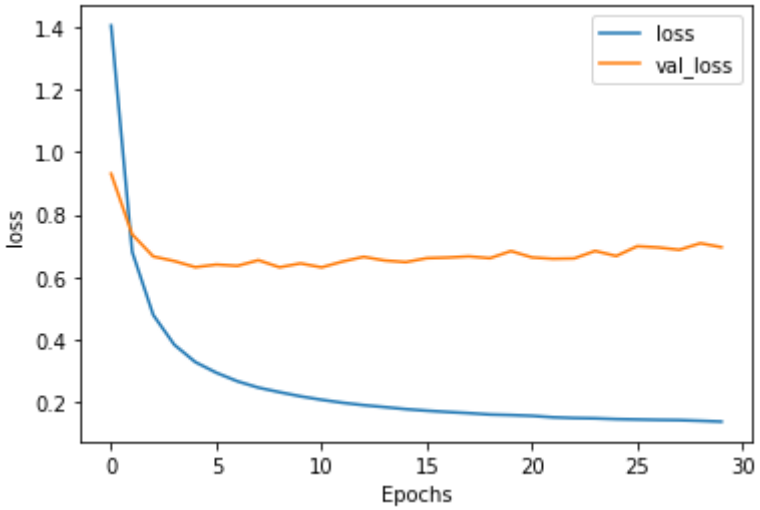

Figure 53 : Perte pour le premier classificateur pendant 30 itérations

La figure 54 ci-dessous représente un détail statistique démontrant l'efficacité des tests appliqués sur 43 familles de protéines.

	precision	recall	f1-score	support
APOPTOSIS	0.88	0.79	0.83	281
BIOSYNTHETIC PROTEIN	0.88	0.90	0.89	208
CELL ADHESION	0.90	0.79	0.84	625
CELL CYCLE	0.76	0.78	0.77	467
CHAPERONE	0.94	0.91	0.92	887
CONTRACTILE PROTEIN	0.84	0.89	0.86	237
DE NOVO PROTEIN	0.85	0.82	0.84	343
DNA BINDING PROTEIN	0.85	0.81	0.83	620
ELECTRON TRANSPORT	0.79	0.78	0.78	619
FLUORESCENT PROTEIN	0.98	0.96	0.97	222
GENE REGULATION	0.90	0.82	0.86	313
HORMONE	0.93	0.92	0.92	318
HYDROLASE	0.88	0.90	0.89	9381
HYDROLASE/HYDROLASE INHIBITOR	0.80	0.78	0.79	2254
IMMUNE SYSTEM	0.91	0.93	0.92	3040
ISOMERASE	0.93	0.93	0.93	1247
LIGASE	0.91	0.88	0.89	953
LYASE	0.95	0.95	0.95	2275
MEMBRANE PROTEIN	0.83	0.79	0.81	951
METAL BINDING PROTEIN	0.82	0.80	0.81	615
METAL TRANSPORT	0.85	0.82	0.84	216
MOTOR PROTEIN	0.78	0.67	0.72	220
OXIDOREDUCTASE	0.93	0.95	0.94	6846
OXIDOREDUCTASE/OXIDOREDUCTASE INHIBITOR	0.69	0.65	0.67	281
PHOTOSYNTHESIS	0.86	0.86	0.86	653
PROTEIN BINDING	0.70	0.70	0.70	1002
PROTEIN FIBRIL	0.93	0.97	0.95	213
PROTEIN TRANSPORT	0.91	0.72	0.80	543
RNA BINDING PROTEIN	0.76	0.77	0.77	379
SIGNALING PROTEIN	0.72	0.80	0.76	1254
STRUCTURAL GENOMICS, UNKNOWN FUNCTION	0.85	0.76	0.80	692
STRUCTURAL PROTEIN	0.88	0.78	0.83	836
SUGAR BINDING PROTEIN	0.94	0.93	0.93	669
TOXIN	0.91	0.85	0.88	508
TRANSCRIPTION	0.82	0.84	0.83	1736
TRANSFERASE	0.89	0.93	0.91	7308
TRANSFERASE/TRANSFERASE INHIBITOR	0.76	0.60	0.67	620
TRANSLATION	0.89	0.84	0.86	206
TRANSPORT PROTEIN	0.90	0.87	0.88	1722
UNKNOWN FUNCTION	0.88	0.80	0.84	596
VIRAL PROTEIN	0.90	0.90	0.90	1740
VIRAL PROTEIN/IMMUNE SYSTEM	0.77	0.79	0.78	258
VIRUS	0.97	0.98	0.98	1420
accuracy			0.88	55774
macro avg	0.86	0.84	0.85	55774
weighted avg	0.88	0.88	0.88	55774

Figure 54 : Résultats obtenus de l'évaluation du modèle

On peut constater que le champ d'efficacité du test était confiné entre deux valeurs limites. La valeur limite minimale obtenue est égale à 64% et elle est liée à la famille des INHIBITEUR D'OXIDOREDUCTASE / OXIDOREDUCTASE et la valeur limite maximale est de 100%, qui est une valeur liée à la famille des PROTÉINES FLUORESCENTES.

Prédiction de la classification du gène assemblé et traduit :

Une fois que le modèle d'apprentissage est prêt à l'emploi, il est utilisé afin de prédire la classification de la séquence protéique traduite à partir de la séquence nucléique assemblé (voir figure 55).

```
result = result.loc[:, (result != 0).any(axis=0)]
result
```

	CHAPERONE	ELECTRON TRANSPORT	OXIDOREDUCTASE	TRANSFERASE
0	0.01	0.01	0.07	0.91

Figure 55 : Résultat de la prédiction de la classification

L'affichage des résultats représente les probabilités d'appartenance de la protéine aux 43 familles. Uniquement les valeurs supérieures à zéro sont affichées. La plus grande probabilité est de 0,91. Notre modèle prédit donc que la Ribokinase A utilisée est une transférase.

Discussion

Les fonctions biologiques des protéines sont très importantes, mais leurs études en laboratoire sont très coûteuses. Étant donné que la génération de données de protéines nouvellement séquencées est en augmentation considérable, les chercheurs de bioinformatiques ont eu recours à la prédiction automatique de leurs fonctions principalement à travers des méthodes de modélisation informatique.

Les approches traditionnelles reposent sur des algorithmes d'alignement qui s'adaptent généralement au moins linéairement à la taille de la requête et de la base de données. Cette complexité temporelle n'est pas en mesure de suivre la taille actuelle et les taux de croissance exponentielle des bases de données actuelles sur les protéines. Par exemple les méthodes basées sur K-mer et le modèle de Markov caché par profil (pHMM). Toutefois ces méthodes restent limitées :

- Les méthodes basées sur k-mer ne sont pas suffisamment précises pour attribuer des fonctions protéiques.
- Le pHMM n'est pas assez rapide pour gérer un grand nombre de séquences de protéines issues de nombreux projets génomiques.

Les méthodes classiques sont généralement déduites de méthodes bien motivées mais restent toujours heuristiques tel que l'outil de recherche d'alignement local de base BLAST qui recherche dans une base de données les protéines homologues d’une protéine de requête donnée, via un alignement de séquences multiples, et qui, par la suite affecte à une séquence protéique requête la fonction de la protéine la plus similaire dans sa base de données.

Par conséquent, une méthode de prédiction de la fonction des protéines qui soit plus précise et plus rapide est nécessaire.

- Les méthodes de DL, et en particulier les algorithmes auto-supervisés issus du NLP, sont des approches prometteuses dans ce sens.
- Les méthodes sans alignement (alignment-free) tel que notre approche DeepPred, peuvent extraire des informations fonctionnelles directement de la séquence sans avoir besoin d'alignement multiple.

Le travail accompli et les résultats obtenus prouvent que le DL joue un rôle efficace dans les prédictions et les classifications, notamment dans la classification des protéines traduites *in silico*. Mais à une condition : le séquençage et l'assemblage doivent être convenablement réalisées. Car la séquence d'un gène est le substrat de l'annotation fonctionnelle. Ce travail propose une solution concrète afin de remédier au problème de l'annotation fonctionnelle, qui est l'un des problèmes majeurs de la bioinformatique.

La prédiction a été faite sur les 43 plus grandes familles, contenant au moins 1000 protéines quelles qu'elles soient (fonctionnelles, structurelles, ou membranaires ... etc.), le résultat obtenu par le modèle est 88% de précision, il existe quelques travaux voisins, avec différents avantages et différents inconvénients, une comparaison directe des résultats sera peu significatif étant donné la différence entre les approches et données utilisées, mais on considère tout de même ces travaux comme complémentaires les uns aux autres.

La comparaison entre les différents travaux peut se faire selon différents critères (Voir le tableau 07) :

Le 1er critère : la nature de données utilisées pour l'apprentissage et la prédiction:

- DeepProt utilise comme entrée les séquences protéiques entières. Mais l'apprentissage se limite aux 43 plus grandes classes protéiques.
- DeepFam utilise comme entrée les domaines protéiques. L'apprentissage se limites aux motifs, généré par le logiciel MEME à partir des donnés des bases de données : COG et GPCR
- UDSMprot utilise comme entrée les séquences protéiques entières, mais l'apprentissage se limite aux enzymes uniquement.
- DeepFunc utilise comme entrée les séquences protéiques entières. Mais se limite aux séquences ayant au maximum 1002 acides aminés. L'apprentissage s'est fait sur les données de SwissProt sauf que ces derniers ont été filtrés afin de supprimer toutes les séquences protéiques longues (supérieur à 1002), et toutes les protéines contenant des acides aminés considérés comme ambigus : B, O, J, U, X, et Z.

Le 2ème critère : étendue de la prédiction

- DeepProt est apte à prédire la classe de n'importe quelle protéine faisant partie des 43 plus grandes familles protéiques.

- DeepFam est incapable de faire une prédiction de classification sans passer par l'outil d'extraction de motifs MEME.
- UDSMprot est incapable de réaliser une prédiction d'une famille hormis les enzymes.
- DeepFunc est incapable de réaliser une prédiction correcte si la séquence protéique contient l'un des acides aminés suivant : B, O, J, U, X, et Z.

Le 3ème critère : l'indépendance à d'autres outils ou approches

- DeepProt est totalement indépendant, ne nécessite aucun autre outil avant de procéder à la prédiction.
- DeepFam doit passer impérativement par une extraction de motif par le logiciel MEME, donc il se repose sur un travail déjà accompli qui consiste à identifier la zone le plus significative d'une protéine qui n'est d'autre que le motif : l'élément influant sur la classification.
- UDSMprot totalement indépendant, mais limité à 6 familles uniquement.
- DeepFunc dépend des données fournit par le réseau d'interaction protéine-protéine au quelle il est connecté, sans ce réseau, il perd sa précision.

Tableau 7 : Comparaison de DeepProt avec travaux voisins

Critère / Travail	Nature de données de l'apprentissage et de la prédiction	Indépendance de l'apprentissage et la prédiction	Etendue de la prédiction
DeepFam	Motifs protéiques uniquement	Dépendant du logiciel MEME qui extrait les motifs	Toutes les familles
UDSMprot	Séquences protéiques entières	Totalement indépendant	Enzymes uniquement
DeepFunc	Longues séquences exclues, Protéines ayant les acides aminés suivant : (B, O, J, U, X, et Z) sont exclues.	Réseau d'interaction protéine-protéine	Toutes les familles
DeepProt (notre approche)	Séquences protéiques entières	Totalement indépendant	43 plus grandes familles

CONCLUSION

CONCLUSION

L’apparition des techniques de séquençage à haut débit a représenté un vrai défi pour les différentes disciplines en rapport avec le séquençage. Que ça soit en matière de traitement ou d’interprétation, la bioinfomatique demeure la discipline la plus concernée par cet avènement. L’un des défis rencontrés est l’annotation des gènes, que ce travail prend partiellement en charge, à commencer par l’assemblage d’un gène nouvellement séquencé, puis le traduire en séquence protéique pour finir par prédire sa classification.

L’approche abordée par ce travail consiste à assembler un gène nouvellement séquencé à partir d’un fichier fastq généré par un programme, mais qui est présumé simuler le rôle d’un vrai fichier généré par un appareil de séquençage à haut débit. Le gène assemblé par l’implémentation de l’algorithme Greedy SCS a été traduit en protéine par une fonction biopython, qui à son tour été utilisée par un modèle de prédiction basé sur le PNL et sur le DL, afin de prédire sa classification.

Le modèle de prédiction représente la partie majeure de la contribution. Néanmoins, malgré l’efficacité et la précision prouvées par tout le processus depuis l’assemblage à la prédiction, ce travail modeste demeure incomplet, et mérite des recherches plus approfondie.

Ainsi, les perspectives futures sont :

- Faire usage des techniques d’IA pour développer un modèle de correction des Reads d’assemblage génomique.
- Utilisation d’un HPC qui permettra d’exploiter de nouvelles options, de réduire les temps de calculs.
- Faire l’apprentissage sur de plus grandes base de données.
- Permettre au modèle de réaliser un apprentissage sur un plus grand nombre de familles.

RÉFÉRENCES BIBLIOGRAPHIQUES

RÉFÉRENCES

Alpaydin, E., 2009. Introduction to Machine Learning, Second Edition | The MIT Press, Second Edition. ed. The MIT Press.

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J., 1990. Basic local alignment search tool. J. Mol. Biol. 215, 403–410. https://doi.org/10.1016/S0022-2836(05)80360-2

Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J., 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. 25, 3389–3402.

Anaconda Navigator — Anaconda documentation [WWW Document], 2020. URL https://docs.anaconda.com/anaconda/navigator/ (accessed 9.10.20).

Applications of Machine Learning - Javatpoint [WWW Document], 2018. . www.javatpoint.com. URL https://www.javatpoint.com/applications-of-machine-learning (accessed 9.8.20).

Ayyappan, J., 2017. Natural Language Processing (NLP) for Dummies – Anexinet. URL https://www.anexinet.com/blog/natural-language-processing-nlp-dummies/ (accessed 9.1.20).

Bailey, T.L., 2011. DREME: motif discovery in transcription factor ChIP-seq data. Bioinformatics 27, 1653–1659. https://doi.org/10.1093/bioinformatics/btr261

Besser, J., Carleton, H.A., Gerner-Smidt, P., Lindsey, R.L., Trees, E., 2018. Next-generation sequencing technologies and their application to the study and control of bacterial infections. Clin. Microbiol. Infect. 24, 335–341. https://doi.org/10.1016/j.cmi.2017.10.013

Biolabs, N.E., 2017. Improving Enzymatic DNA Fragmentation for Next Generation Sequencing Library Construction | NEB [WWW Document]. URL https://international.neb.com/tools-and-resources/feature-articles/improving-enzymatic-dna-fragmentation-for-next-generation-sequencing-library-construction (accessed 9.4.20).

Biopython [WWW Document], 2020. URL https://biopython.org/ (accessed 9.11.20).

Bouguelia, M.-R., 2015. Classification et apprentissage actif à partir d'un flux de données évolutif en présence d'étiquetage incertain (phdthesis). Université de Lorraine.

Boullier, D., Lohard, A., 2012. Chapitre 5. Détecter les tonalités : opinion mining et sentiment analysis, in: Opinion mining et Sentiment analysis : Méthodes et outils, Sciences Po │ médialab. OpenEdition Press, Marseille.

Breda, A., Valadares, N.F., Souza, O.N. de, Garratt, R.C., 2007. Protein Structure, Modelling and Applications, Bioinformatics in Tropical Disease Research: A Practical and Case-Study Approach [Internet]. National Center for Biotechnology Information (US).

Brownlee, J., 2017a. Difference Between Classification and Regression in Machine Learning. Mach. Learn. Mastery. URL https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/ (accessed 9.8.20).

Brownlee, J., 2017b. How to Evaluate the Skill of Deep Learning Models. Mach. Learn. Mastery. URL https://machinelearningmastery.com/evaluate-skill-deep-learning-models/ (accessed 9.8.20).

Choudhuri, S., 2014. Chapter 7 - Additional Bioinformatic Analyses Involving Nucleic-Acid Sequences**The opinions expressed in this chapter are the author's own and they do not necessarily reflect the opinions of the FDA, the DHHS, or the Federal Government., in: Choudhuri, S. (Ed.), Bioinformatics for Beginners. Academic Press, Oxford, pp. 157–181. https://doi.org/10.1016/B978-0-12-410471-6.00007-4

ClustalW2 EMBL-EBI [WWW Document], 2020. URL https://www.ebi.ac.uk/Tools/msa/clustalw2/ (accessed 9.6.20).

Collecte de données étiquetées - Amazon Machine Learning [WWW Document], 2020. URL https://docs.aws.amazon.com/fr_fr/machine-learning/latest/dg/collecting-labeled-data.html (accessed 9.8.20).

Cruz, J.A., Wishart, D.S., 2007. Applications of machine learning in cancer prediction and prognosis. Cancer Inform. 2, 59–77.

Deitel, P., Deitel, H., 2018. Python for Programmers: with Big Data and Artificial Intelligence Case Studies. Prentice Hall, Boston, MA.

Dey, A., 2016. Machine Learning Algorithms: A Review 7, 6.

Fabien, M., 2019. Traitement Automatique du Langage Naturel en français (TAL / NLP) [WWW Document]. Stat4decision. URL https://www.stat4decision.com/fr/traitement-langage-naturel-francais-tal-nlp/ (accessed 9.9.20).

Finn, R.D., Mistry, J., Tate, J., Coggill, P., Heger, A., Pollington, J.E., Gavin, O.L., Gunasekaran, P., Ceric, G., Forslund, K., Holm, L., Sonnhammer, E.L.L., Eddy, S.R., Bateman, A., 2010. The Pfam protein families database. Nucleic Acids Res. 38, D211–D222. https://doi.org/10.1093/nar/gkp985

Frameworks for Approaching the Machine Learning Process, 2018. . KDnuggets. URL https://www.kdnuggets.com/a-general-approach-to-the-machine-learning-process.html/ (accessed 9.8.20).

Frith, M.C., Saunders, N.F.W., Kobe, B., Bailey, T.L., 2008. Discovering Sequence Motifs with Arbitrary Insertions and Deletions. PLOS Comput. Biol. 4, e1000071. https://doi.org/10.1371/journal.pcbi.1000071

Géron, A., 2017. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 1st Edition. ed. O'Reilly Media, Beijing ; Boston.

Graeme, H., 2017. Neural Network Methods in Natural Language Processing, Neural Network Methods in Natural Language Processing. Presented at the Neural Network Methods in Natural Language Processing, Morgan & Claypool.

Grant, C.E., Bailey, T.L., Noble, W.S., 2011. FIMO: scanning for occurrences of a given motif. Bioinformatics 27, 1017–1018. https://doi.org/10.1093/bioinformatics/btr064

Gromiha, M.M., Nagarajan, R., Selvaraj, S., 2019. Protein Structural Bioinformatics: An Overview, in: Ranganathan, S., Gribskov, M., Nakai, K., Schönbach, C. (Eds.), Encyclopedia of Bioinformatics and Computational Biology. Academic Press, Oxford, pp. 445–459. https://doi.org/10.1016/B978-0-12-809633-8.20278-1

Grossfeld, B., 2020. Deep learning vs machine learning [WWW Document]. Zendesk. URL https://www.zendesk.com/blog/machine-learning-and-deep-learning/ (accessed 9.8.20).

Hagiwara, M., 2020. Real-World Natural Language Processing: Practical applications with deep learning. Manning Publications, S.l.

How to predict new samples with your Keras model?, 2020. . MachineCurve. URL https://www.machinecurve.com/index.php/2020/02/21/how-to-predict-new-samples-with-your-keras-model/ (accessed 9.8.20).

Hulo, N., Bairoch, A., Bulliard, V., Cerutti, L., Cuche, B.A., de Castro, E., Lachaize, C., Langendijk-Genevaux, P.S., Sigrist, C.J.A., 2008. The 20 years of PROSITE. Nucleic Acids Res. 36, D245-249. https://doi.org/10.1093/nar/gkm977

Ignatov, K.B., Blagodatskikh, K.A., Shcherbo, D.S., Kramarova, T.V., Monakhova, Y.A., Kramarov, V.M., 2019. Fragmentation Through Polymerization (FTP): A new method to fragment DNA for next-generation sequencing. PLoS ONE 14. https://doi.org/10.1371/journal.pone.0210374

Jenkins, G.J.S., Williams, G.L., Beynon, J., Ye, Z., Baxter, J.N., Parry, J.M., 2002. Restriction enzymes in the analysis of genetic alterations responsible for cancer progression. BJS Br. J. Surg. 89, 8–20. https://doi.org/10.1046/j.0007-1323.2001.01968.x

Kalyanaraman, A., 2011. Genome Assembly, in: Padua, D. (Ed.), Encyclopedia of Parallel Computing. Springer US, Boston, MA, pp. 755–768. https://doi.org/10.1007/978-0-387-09766-4_402

Kasoji, S.K., Pattenden, S.G., Malc, E.P., Jayakody, C.N., Tsuruta, J.K., Mieczkowski, P.A., Janzen, W.P., Dayton, P.A., 2015. Cavitation Enhancing Nanodroplets Mediate Efficient DNA Fragmentation in a Bench Top Ultrasonic Water Bath. PLOS ONE 10, e0133014. https://doi.org/10.1371/journal.pone.0133014

Kono, N., Arakawa, K., 2019. Nanopore sequencing: Review of potential applications in functional genomics. Dev. Growth Differ. 61, 316–326. https://doi.org/10.1111/dgd.12608

La vraie différence entre Machine Learning & Deep Learning | Jedha Bootcamp [WWW Document], 2020. URL https://www.jedha.co/blog/la-vraie-difference-entre-machine-learning-deep-learning (accessed 9.8.20).

Laroum, S., 2011. Prédiction de la localisation des protéines membranaires : méthodes méta-heuristiques pour la détermination du potentiel d'insertion des acides aminés (phdthesis). Université d'Angers.

Lee, C., Benjamin, N.W., Pokharel, S., 2012. Managing Uncertain Inventory in Supply Chain with Neural Network and Radio Frequency Identification (RFID). https://doi.org/10.4018/978-1-60960-585-8.CH010

Lee, J., Freddolino, P.L., Zhang, Y., 2017. Ab Initio Protein Structure Prediction, in: J. Rigden, D. (Ed.), From Protein Structure to Function with Bioinformatics. Springer Netherlands, Dordrecht, pp. 3–35. https://doi.org/10.1007/978-94-024-1069-3_1

Levinthal, C., 1966. Molecular model-building by computer. Sci. Am. 214, 42–52. https://doi.org/10.1038/scientificamerican0666-42

Machine learning definition [WWW Document], 2016. . Digit. Insid. URL https://digitalinsiders.feelandclic.com/machine-learning-definition (accessed 9.8.20).

Machiraju, S., Modi, R., 2018. Natural Language Processing, in: Machiraju, S., Modi, R. (Eds.), Developing Bots with Microsoft Bots Framework: Create Intelligent Bots Using MS Bot Framework and Azure Cognitive Services. Apress, Berkeley, CA, pp. 203–232. https://doi.org/10.1007/978-1-4842-3312-2_9

Matplotlib [WWW Document], 2020. URL https://matplotlib.org/ (accessed 9.10.20).

Metzker, M.L., 2010. Sequencing technologies — the next generation. Nat. Rev. Genet. 11, 31–46. https://doi.org/10.1038/nrg2626

Morisse, P., 2019. Correction de données de séquençage de troisième génération (phdthesis). Normandie Université.

NumPy [WWW Document], 2020. URL https://numpy.org/about/ (accessed 9.10.20).

Ongsulee, P., 2017. Artificial intelligence, machine learning and deep learning. 2017 15th Int. Conf. ICT Knowl. Eng. ICT KE 1–6. https://doi.org/10.1109/ICTKE.2017.8259629

Paladino, A., Marchetti, F., Rinaldi, S., Colombo, G., 2017. Protein design: from computer models to artificial intelligence. WIREs Comput. Mol. Sci. 7, e1318. https://doi.org/10.1002/wcms.1318

pandas - Python Data Analysis Library [WWW Document], 2020. URL https://pandas.pydata.org/about/ (accessed 9.10.20).

Patterson, J., Gibson, A., 2017. Deep Learning: A Practitioner's Approach. O'Reilly Media.

P.C, T., A.G, M.L., A.D, B., M.R.H, W., 2006. Biologie Moléculaire. Berti editions.

Platt, A., Ross, H.C., Hankin, S., Reece, R.J., 2000. The insertion of two amino acids into a transcriptional inducer converts it into a galactokinase. Proc. Natl. Acad. Sci. U. S. A. https://doi.org/10.1073/pnas.97.7.3154

Project Jupyter [WWW Document], 2020. URL https://www.jupyter.org (accessed 9.10.20).

Quels sont les usages du Machine Learning (apprentissage automatique) ? [WWW Document], 2019. . Intell. Artif. Data Anal. URL https://ia-data-analytics.fr/machine-learning/usages/ (accessed 9.8.20).

Rashid, M.A., Khatib, F., Sattar, A., 2015. Protein preliminaries and structure prediction fundamentals for computer scientists. ArXiv151002775 Cs Q-Bio.

Reeck, G.R., de Haën, C., Teller, D.C., Doolittle, R.F., Fitch, W.M., Dickerson, R.E., Chambon, P., McLachlan, A.D., Margoliash, E., Jukes, T.H., 1987. "Homology" in proteins and nucleic acids: a terminology muddle and a way out of it. Cell 50, 667. https://doi.org/10.1016/0092-8674(87)90322-9

Rhoads, A., Au, K.F., 2015. PacBio Sequencing and Its Applications. Genomics Proteomics Bioinformatics, SI: Metagenomics of Marine Environments 13, 278–289. https://doi.org/10.1016/j.gpb.2015.08.002

Rijmenam, D.M. van, 2019. 7 Steps to Machine Learning: How to Prepare for an Automated Future [WWW Document]. Medium. URL https://medium.com/dataseries/7-steps-to-machine-learning-how-to-prepare-for-an-automated-future-78c7918cb35d (accessed 9.8.20).

Robinson, P.K., 2015. Enzymes: principles and biotechnological applications. Essays Biochem. 59, 1–41. https://doi.org/10.1042/bse0590001

Schmidhuber, J., 2015. Deep Learning in Neural Networks: An Overview. Neural Netw. 61, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

scikit-learn: machine learning in Python [WWW Document], 2020. URL https://scikit-learn.org/stable/ (accessed 9.10.20).

Shen, C.-H., 2019. Chapter 4 - Gene Expression: Translation of the Genetic Code, in: Shen, C.-H. (Ed.), Diagnostic Molecular Biology. Academic Press, pp. 87–116. https://doi.org/10.1016/B978-0-12-802823-0.00004-3

Sigrist, C.J.A., Cerutti, L., de Castro, E., Langendijk-Genevaux, P.S., Bulliard, V., Bairoch, A., Hulo, N., 2010. PROSITE, a protein domain database for functional characterization and annotation. Nucleic Acids Res. 38, D161–D166. https://doi.org/10.1093/nar/gkp885

Sleator, R.D., Walsh, P., 2010. An overview of in silico protein function prediction. Arch. Microbiol. 192, 151–155. https://doi.org/10.1007/s00203-010-0549-9

Soni, M., Thakur, J.S., 2018. A Systematic Review of Automated Grammar Checking in English Language. ArXiv.

Tayiz, B.C., 2020. Alternative NLP Method [WWW Document]. Medium. URL https://becominghuman.ai/alternative-nlp-method-9f94165802ed (accessed 9.9.20).

TensorFlow [WWW Document], 2020. URL https://www.tensorflow.org/?hl=fr (accessed 9.10.20).

Tiwari, A., 2010. Essentials Of Cell Biology.

Tkinter [WWW Document], 2020. URL https://docs.python.org/3/library/tkinter.html (accessed 9.12.20).

Todd, A.E., Orengo, C.A., Thornton, J.M., 2001. Evolution of function in protein superfamilies, from a structural perspective11Edited by A. R. Fersht. J. Mol. Biol. 307, 1113–1143. https://doi.org/10.1006/jmbi.2001.4513

Types of Machine Learning Algorithms [WWW Document], 2020. . 7wData. URL https://www.7wdata.be/visualization/types-of-machine-learning-algorithms-2/ (accessed 9.8.20).

Using Convolutional Neural Networks for Sentence Classification [WWW Document], 2020. . MissingLink.ai. URL https://missinglink.ai/guides/convolutional-neural-networks/using-convolutional-neural-networks-sentence-classification/ (accessed 9.9.20).

Voulodimos, A., Doulamis, N., Doulamis, A., Protopapadakis, E., 2018. Deep Learning for Computer Vision: A Brief Review [WWW Document]. Comput. Intell. Neurosci. https://doi.org/10.1155/2018/7068349

Wang, L., Wong, A., 2020. COVID-Net: A Tailored Deep Convolutional Neural Network Design for Detection of COVID-19 Cases from Chest X-Ray Images. ArXiv200309871 Cs Eess.

Wang, Z., Yin, P., Lee, J.S., Parasuram, R., Somarowthu, S., Ondrechen, M.J., 2013. Protein function annotation with Structurally Aligned Local Sites of Activity (SALSAs). BMC Bioinformatics 14, S13. https://doi.org/10.1186/1471-2105-14-S3-S13

Webb, B., Sali, A., 2014. Protein structure modeling with MODELLER. Methods Mol. Biol. Clifton NJ 1137, 1–15. https://doi.org/10.1007/978-1-4939-0366-5_1

Welcome to Python.org [WWW Document], 2020. . Python.org. URL https://www.python.org/about/ (accessed 9.10.20).

Whisstock, J.C., Lesk, A.M., 2003. Prediction of protein function from protein sequence and structure. Q. Rev. Biophys. 36, 307–340. https://doi.org/10.1017/S0033583503003901

Wiltgen, M., 2019. Algorithms for Structure Comparison and Analysis: Homology Modelling of Proteins, in: Ranganathan, S., Gribskov, M., Nakai, K., Schönbach, C. (Eds.), Encyclopedia of Bioinformatics and Computational Biology. Academic Press, Oxford, pp. 38–61. https://doi.org/10.1016/B978-0-12-809633-8.20484-6

Zhao, R., 2019. Development of a CMOS pixel sensor with on-chip artificial neural networks (phdthesis). Université de Strasbourg.

Zhavoronkov, A., 2018. Artificial Intelligence for Drug Discovery, Biomarker Development, and Generation of Novel Chemistry. Mol. Pharm. 15, 4311–4313. https://doi.org/10.1021/acs.molpharmaceut.8b00930

Soutenue le :	Présenté par :
17/09/2020	ALIOUANE Salah Eddine & BENDAHMANE Abdelhafedh

Thème :

Nouvelle approche de prédiction des classes protéiques issues d'un séquençage NGS par Deep Learning

Mémoire Présenté en vue de l'obtention du Diplôme de Master en :

Bioinformatique

Domaine : Science de la nature et la vie

Département de Biologie Appliquée

Le but de ce travail est de développer une approche DeepProt visant à simuler les phases du processus de la post-génomique à partir d'une séquence nucléique d'un séquençage NGS. Cette approche est basée sur deux axes de l'intelligence artificielle (IA) : le traitement automatique du langage naturel (NLP) et l'apprentissage profond (DL). Avec un apprentissage sur 43 familles et un taux de précision de 88% obtenus dans les tests effectués, les résultats ont montré l'efficacité de cette approche, notamment la phase de prédiction basée sur le NLP et le DL. Ces deux outils, combinés, ont donné un modèle d'une grande capacité à extraire les connaissances des données protéiques afin de prédire et classer celles-ci. Le modèle proposé apprend grâce à un entrainement intensif par exploitation des séquences protéiques. Ce travail a permis de mettre en évidence l'apport de cette approche à améliorer la précision de la classification des protéines.

Mots clés : Assemblage ; Classification ; Prédiction ; Intelligence Artificiel ; NLP ; NGS.

Jury d'évaluation:
Président du jury : Dr. MENACER Rafik
Co-Encadreur : Pr. HAMIDECHI M. Abdelhafid
Co-Encadreur : Dr. CHEHILI Hamza
Examinateur : Dr. GHERBOUDJ Amira