RAPPORT FINAL : PROJET SCIENTIFIQUE COLLECTIF

Chimie et machine learning

CHI05

25 avril 2022

MAALEJ, EL-BOUKKOURI, BARTHES, GARANCINI

Tuteurs : Gilles FRISON, Benoit Da MOTA, Bastien NAY

Coordinateur : Alexis ARCHAMBEAU

TABLE DES MATIÈRES

1. INTRODUCTION

Grâce au Machine Learning et à l'informatique de manière générale, les chimistes ont désormais accès à de nombreux outils pour les accompagner dans leurs recherches. C'est cela qui nous a attiré dans ce PSC au premier abord, la possibilité de voir un travail informatique, la création d'un programme ou le développement d'un algorithme servir en pratique et pouvoir l'appliquer à des objets aussi concrets que des molécules ou des réactions.

C'est ensuite en voyant le PSC des X19 [1] que notre envie s'est concrétisée et s'est précisée. Nous étions passés d'une matière, d'un domaine, à l'approfondissement d'un sujet qui semblait lui-même passionnant. Voici pourquoi de nombreuses références à ce travail parsèment ce rapport . Il nous a inspirés, a guidés notre travail de loin et nous a permis de tracer le fil rouge de notre projet.

Une fois le sujet choisi, nous avons dû décider d'une ligne de conduite pour notre PSC. Les X19 ayant travaillé sur la synthétisabilité des molécules il nous semblait logique et pertinent d'étudier la rétrosynthèse. Ce concept, que nous détaillerons plus loin, fait suite de manière assez évidente au concept de synthétisabilité.

Notre objectif est alors **d'étudier les algorithmes existants, de les comprendre afin de les combiner, de les améliorer, voire d'implémenter notre propre algorithme de rétrosynthèse.**

Ce rapport présente la méthode que nous avons employée, les bases de donnée utilisées, les algorithmes développés ainsi que les résultats qui leur sont liés.

L'ensemble des codes évoqués dans ce rapport est disponible au github :
Github : https://github.com/fressharkk/PSC_CHI_05

2. État de l'art

2.1 La Rétrosynthèse

Notre PSC trouvant son origine dans celui des X19, nous nous sommes naturellement intéressés tout d'abord à la synthétisabilité. Ce concept, que l'on ne détaillera pas ici est celui étudié en profondeur dans le PSC des X19. Néanmoins, il convient d'en retirer l'utilité. Il s'agit d'aider les chimistes à trouver des réactions de synthèse. C'est pourquoi nous nous sommes intéressés par la suite à la rétrosynthèse. Il s'agissait de la suite logique du projet des X19 et c'est ce concept que nous avons étudié en détail.

La rétrosynthèse est la tâche qui consiste à prendre une molécule et à tenter de deviner quels réactifs pourraient donner cette dernière. Ce procédé, faisable humainement est néanmoins très chronophage et c'est pourquoi il semble intéressant à implémenter informatiquement. Nous nous sommes donc intéressés aux algorithmes de rétrosynthèses déjà présent dans la littérature open-source.

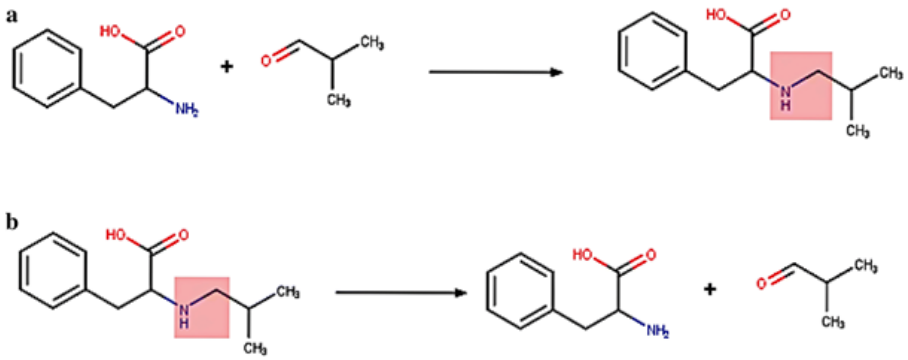


Figure 1 – a. Synthèse b. Retrosynthèse

2.2 Représentation(s) chimique(s) et informatique

L'objectif étant d'étudier et d'implémenter des algorithmes permettant d'effectuer de la rétrosynthèse, il nous a fallu étudier la façon de communiquer aux ordinateurs les informations pertinentes afin qu'ils puissent les appréhender et les analyser.

2.2.1 • Format SMILES

Le premier problème est d'exprimer une molécule de manière complètement déterminée dans un langage relativement simple afin qu'un programme puisse de manière assez rapide travailler dessus pour en retirer des données. Il ne s'agit pas de donner en entrée à un algorithme la formule brute d'une molécule, qui serait trop pauvre en informations, ni le dessin en 3D de cette molécule, qui serait alors bien trop complexe à analyser. Il faut trouver un entre-deux.

Un tel type de données existe : le format SMILES [2]. Il s'agit d'un format permettant de décrire en une ligne de caractère une molécule de manière quasiment unique. Le principe est le suivant : un des atomes est choisi comme atome de base puis chaque embranchement est noté par un parenthésage. Des notations spécifiques sont réservées pour les cycles (des entiers) ou les doubles et triples liaisons (resp. = et #).

Plusieurs variantes de ce format existent, permettant par exemple de différencier les stéréoisomères (isoSMILES), mais nous ne conserveront ici que le format classique pour des questions de simplicité.

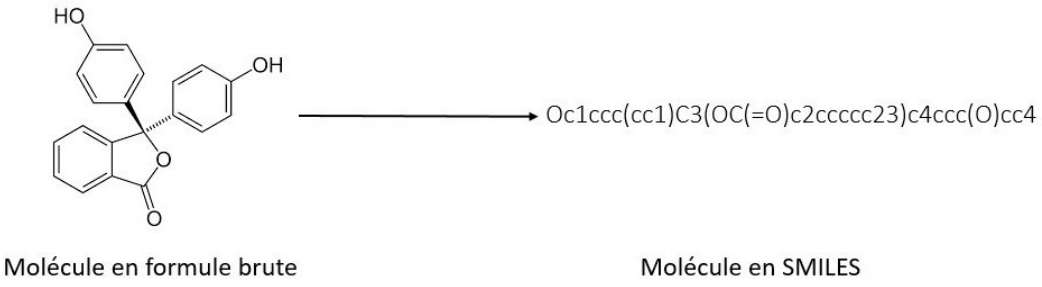


Figure 2 – Conversion de formule brute en SMILES

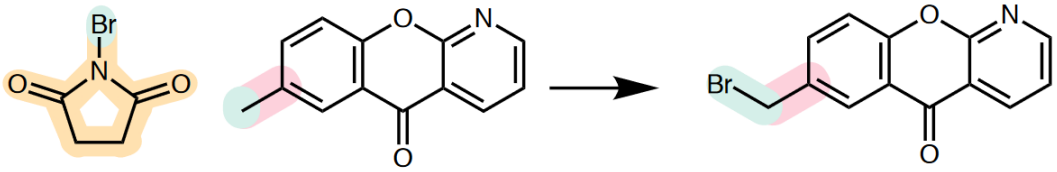
2.2.2 • Format SMARTS

Les outils informatiques d'aujourd'hui permettent la recherche de sous-structures, c'est-à-dire la recherche d'un motif particulier ou sous-graphe dans une molécule ou graphe. Ce processus est souvent utilisé dans tous les algorithmes de représentation numérique d'une molécule afin de mettre en évidence un certain groupe fonctionnel, de conception des médicaments et de chimie analytique.

Le format SMARTS [3] est une extension du format SMILES dans lequel les atomes ou les liaisons peuvent être remplacés par des "jokers". Ceci permet de mettre en évidence des structures communes et par exemple coder des modèles de réaction (reaction *templates*).

2.2.3 • Modèles de réactions (Reaction *templates*)

Nous avons vu comment nous pouvions représenter les molécules et les réactions mais un autre type de données doit aussi être représenté : les *templates*. Les *templates* sont des règles ou des schémas de réactions qui sont créés à la main ou bien générés à partir de bases de données de réactions et qui permettent ensuite de créer des réactions à partir de réactifs ou de produits. Ces *templates* ne sont pas uniques pour chaque réaction, plusieurs réactions partagent les mêmes. Ces templates peuvent être obtenus à partir de réactions grâce à l'algorithme RDChiral [4].




```
[Br;H0;D1;+0:1]-[CH2;D2;+0:2]-[c:3]>>
    O=C1-C-C-C(=O)-N-1-[Br;H0;D1;+0:1].[CH3;D1;+0:2]-[c:3]
```

Figure 3 – Template d'une rétro-réaction

2.2.4 • Morgan Fingerprints

Le format SMILES est très performant car il permet de donner presque l'ensemble des informations en un minimum de caractère mais il reste trop complexe pour un modèle de Machine Learning comme un réseau de neurone qui prend de manière classique un vecteur colonne en entrée. Il faut donc trouver une façon de transmettre certaines informations de la molécule en un tableau d'entiers. C'est là qu'entrent en jeu les Morgan Fingerprints [**Fingerprint**]. Donnés par la fonction de RDKit "Feature Morgan" (rdkit.Chem.GetMorganFingerprintAsBitVect), ce tableau est obtenu de la manière suivante :

A chaque atome est associé un identifiant en fonction de ses voisins, de la présence de cycles, de voisins non hydrogénés, etc... Puis par une méthode d'itération on met à jour les identifiants des atomes en fonctions de ses voisins directs puis indirects, des types de liaisons dont il fait partie. On vérifie ensuite qu'il n'y a pas de doublons, puis on convertit la liste de ces identifiant en une liste de booléen (classiquement de longueur 2048) : le FingerPrint.

Figure 4 – Conversion SMILES en FingerPrint

Une particularité est à préciser en ce qui concerne les FingerPrints d'une réaction. Le tableau correspondant est donné par la fonction suivante :

$$Fp(reaction) = Fp(product) - \sum Fp(reactants)$$

2.3 Algorithmes fournis par la littérature

Notre objectif étant de créer un algorithme de rétrosynthèse en s'appuyant sur des algorithmes existants, il nous a fallu faire des recherches dans la littérature *open-source*. En effet, plusieurs entreprises privées proposent des algorithmes faisant de la rétrosynthèse mais aucune

d'entre elles ne mettent à disposition leurs codes. Nous avons donc trouvé quelques algorithmes en *open-source* dont deux auxquels nous nous sommes particulièrement intéressés.

2.3.1 • LillyMol - RTSA

Lors de nos recherches nous avons trouvé ce premier algorithme *open-source* : Retrosynthesis Algorithm (RTSA) co-écrit par Ian A. Watson, Jibo Wang et Christos A. Nicolaou sous le nom de "LillyMol" [5].

Cet algorithme s'appuie sur une idée simple. Pour chercher la réaction de synthèse de molécule on tente de voir si on connaît des modèles de réaction qui peuvent s'appliquer à cette réaction. C'est ce qui est implémenté ici. L'algorithme travaille d'abord sur une base de données d'entraînement composée de réactions. Pour chaque réaction, il va tenter de la mettre dans une "catégorie", un modèle de réaction renversée ou RRT (*Reverse Reaction Template*). Pour cela, il se base sur les atomes centraux de la réaction et étudie la structure de leur voisinage pour reconnaître les *patterns* et ainsi classer la réaction.

Une fois cette liste de *templates* obtenue à laquelle est associé une liste de réactions types, l'algorithme peut commencer à prédire la rétro-réaction. Il prend la molécule passée en entrée et tente de lui appliquer les *templates* de la liste qu'il a constituée plus tôt. Si l'on trouve une correspondance, on aura trouvé une réaction donnant le produit souhaité.

Malheureusement, nous n'avons pas réussi à installer correctement cet algorithme et nous n'avons donc pas pu le faire fonctionner. Nous avons néanmoins décidé de l'évoquer dans ce rapport car il nous a permis, en lisant son code, de mieux comprendre les raisonnements généraux sur les traitements de bases de données, sur les *templates* ou sur la rétrosynthèse.

2.3.2 • AiZynthFinder

En continuant à chercher après notre échec avec LillyMol, nous avons trouvé le travail de Samuel Genheden, Amol Thakkar, Veronika Chadimová, Jean-Louis Reymond, Ola Engkvist et Esben Bjerrum [6]. Leur objectif était de faire un algorithme complètement *open-source* et suffisamment bien documenté pour être accessible par des débutants dans le domaine. Il est vrai que la documentation (**AiZynthFinder :** https://molecularai.github.io/aizynthfinder/) était très bien fournie et que c'est grandement grâce à ce travail de documentation que nous

avons pu comprendre l'algorithme dans le détail.

L'algorithme se décompose en plusieurs étapes :

1. Pour chaque molécule, on obtient à l'aide d'un réseau de neurone (*expansion policy*) entraîné sur des bases de réactions et de *templates* une liste d'au plus 50 réactions possibles.
2. Par la suite, un autre réseau de neurone (*filter policy*) entraîné sur des bases de fausses et vraies réactions, va filtrer ces réactions possibles pour n'en garder qu'une poignée.
3. L'algorithme itère ces opérations afin de construire un arbre de réactions.
4. Une recherche de Monte-Carlo est alors effectuée dans l'arbre afin de récupérer la meilleure route possible.

Ces étapes permettent donc en passant une molécule à synthétiser en entrée, de récupérer une route de réactions probable pour cette synthèse.

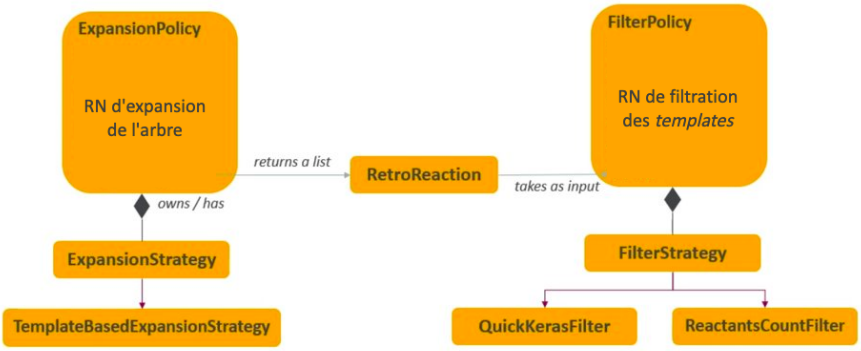


Figure 5 – Structure des Réseaux de neurones d'AiZynthFinder

Tentons d'appliquer AiZynthFinder à un exemple utile en médecine : l'Aspirine. L'Aspirine a pour formule brute $C_9H_8O_4$ et pour smiles $= CC(=O)OC1 = CC = CC = C1C(=O)O$. Le résultat obtenu avec l'algorithme AiZynthFinder (situé dans l'image ci-dessous) demande un peu plus d'analyse. En effet, cette réaction s'apparente à une réaction de saponification (transformation d'un ester en acide carboxylique). Néanmoins, pour la réaliser en vrai, il faut prendre garde au fait que le réactif (à gauche) possède deux fonctions esters pas très différentes. Il devient donc potentiellement difficile de faire une réaction sur l'un sans la faire sur l'autre. Nous aurions alors peut-être un rendement faible ou nul. Ainsi, cet exemple vient rappeler qu'il faut prendre un outil comme AiZynthFinder comme une aide et non comme un résultat fini et utilisable directement en soi.

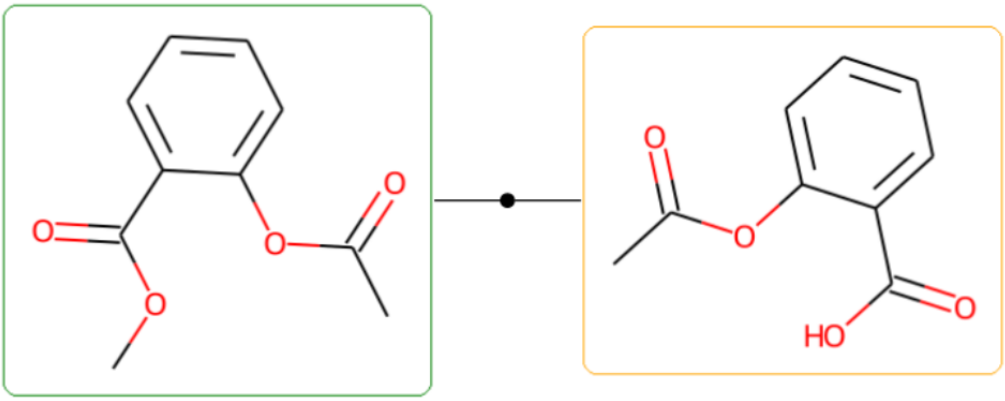


Figure 6 – Application de AiZynthFinder à l'Aspirine

3. NOTRE PROJET

Après avoir étudié les algorithmes existants nous avons pu décider d'une ligne directrice pour notre projet. Nous avons décidé d'améliorer AiZynthFinder. Ce projet étant très ambitieux il nous a fallu restreindre notre champ d'action. C'est en étudiant en profondeur le fonctionnement de l'algorithme que nous avons eu l'idée de spécialiser le filter d'AiZynthFinder. En effet, après l'étape d'expansion de l'arbre qui renvoie au maximum 50 *templates*, il faut filtrer ces *templates* afin de ne garder que les plus pertinents pour éviter d'avoir un arbre dont la taille exploserait. L'objectif est donc de fixer un "cutoff", i.e. un seuil, de telle sorte à ne garder que les réactions que le modèle prédit à une probabilité supérieure au "cutoff". Notre objectif au cours de ce PSC aura donc été de perfectionner cette étape de filtration afin de "battre" AiZynthFinder et donner ainsi une filtration plus adaptée pour établir des routes de rétrosynthèses. Nous avons donc tenté de développer différents modèles de Machine Learning et Deep Learning pour tenter de répondre à ce problème qui sont présentés plus loin dans ce rapport. Le modèle final adopté prend la forme d'une multitude de réseaux neurones que l'on applique aux réactions que l'on cherche à discriminer selon les fonctions chimiques des produits de ces réactions à chaque étape.

4. BASES DE DONNÉES ET PREPROCESSING

Un des travail important en Machine Learning est l'obtention et le traitement des bases de données. En effet, ces dernières vont considérablement influer sur la pertinence des modèles informatiques proposés. Les modèles s'appuient sur ces bases pour apprendre et "comprendre" comment analyser de nouvelles données afin d'effectuer de bonnes prédictions.

4.1 Bases de données déjà constituées

L'étape préliminaire à la préparation de bases de données destinées à l'apprentissage est la récupération de données dans la littérature. En effet, même si nous verrons plus tard que nous créerons des données nous-même, nous avons besoin de nous baser sur des réactions réelles. Nous avons pour cela récupéré une double base de données : la base "United States Patent and Trademark Office" (USPTO) [7]. Cette base de données se décompose en deux parties :

- une base de presque deux millions de réactions sous format SMILES en trois colonnes : une pour les réactifs, une pour le produit qui nous intéresse et une pour la réaction entière
- une base de même taille comprenant les *templates* correspondants toujours en format SMILES

Ces deux bases sont fondamentales pour nous car c'est à partir de celles-ci que nous allons développer l'ensemble de notre dataset et notamment obtenir une base de fausses réactions.

Dans le but de valider et comparer nos modèles, nous avons été confrontés à un autre problème. En effet, le modèle d'AiZynthFinder a été entraîné sur l'entièreté de la base USPTO (jusqu'au 31/04/2018) et nous ne pouvons donc pas le tester sur ce même ensemble. C'est pour cela que nous nous sommes procurés une autre base de réactions disponible en open source dont le seul but est de servir de données de "validation" et "comparaison" des modèles [8]. Nous avons ensuite retiré toutes les réactions en commun avec USPTO contenues dans cette base de donnée pour obtenir une base de réactions réalisables sur laquelle on peut tester AiZynthFinder
.

4.2 Construire une base de données de réactions peu probables

L'étape suivante dans la constitution de notre dataset est la création d'une base de fausses réactions. L'objectif étant d'entraîner un modèle permettant de filtrer les réactions selon la faisabilité. Il faut alors lui donner en entraînement des réactions faisables ET non-faisables. Malheureusement, pour des raisons évidentes il n'existe pas de bases de "fausses réactions" de large taille dans la littérature. Mais les programmeurs d'AiZynthFinder ont été confrontés au même problème et nous proposent plusieurs solutions qu'ils ont détaillées dans un autre article. [9]

L'idée générale est d'utiliser les bases de "vraies réactions" pour construire une base de "fausses réaction". Trois méthodes sont disponibles :

Strict : Il s'agit de la méthode la plus simple, le but est d'appliquer le *templates* d'une réaction à celle-ci et de supprimer le produit qui est dans la base USPTO afin de ne garder que les produits annexes et ainsi obtenir des réactions que l'on peut considérer comme "fausses"

Random : Cette méthode consiste à parcourir les réactifs de la base USPTO, à choisir des *templates* au hasard dans l'autre base et de les appliquer à ces réactifs jusqu'à obtenir quelque chose.

Recommender : Ici, on entraîne un réseau de neurones séquentiel qui prend en entrée des réactifs et qui renvoie un template de réaction qui a de grandes chances de pouvoir être appliqué à ces réactifs pour obtenir un produit à l'issue. On vérifie ensuite que la réaction totale obtenue n'est pas "réalisable" en la comparant aux restes des données de notre base de réaction.

Cela est résumé par le schéma suivant :

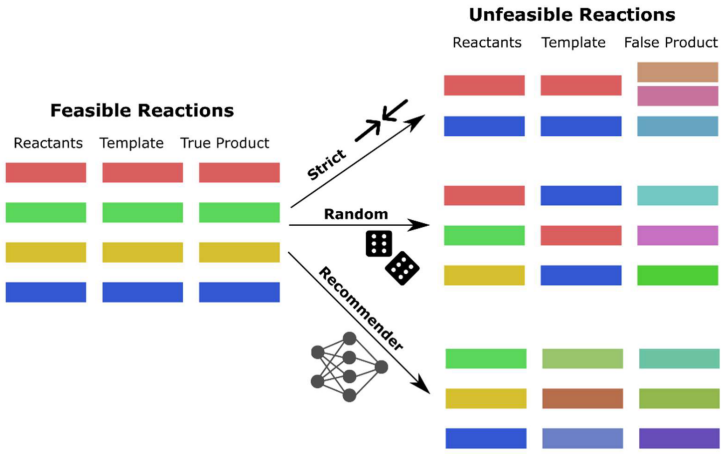


Figure 7 – Représentations des méthodes pour réaliser les "fausses réactions"

Les deux premières méthodes ne nécessitent pas de traitement de données particulier. Néanmoins, le **Recommender** étant un réseau de neurones, il est nécessaire de réaliser un pré-traitement en entrée. A partir des bases USPTO réactions et *templates* , nous allons réaliser trois bases de données appelées respectivement "Purified_*templates*.csv", "Purified_Reactions.csv" et "Purified_Fingerprints.csv". Plus précisément, ces données sont de même taille et contiennent les *templates*, réactions et sommes des fingerprints des réactifs qui correspondent ligne à ligne en ayant enlevé toutes les réactions potentiellement problématiques (celles dont le *templates* n'a pas pu être extrait par exemple). On obtient ainsi un jeu de données long d'environ 1 600 000 réactions qui servira d'entraînement à notre **Recommender** . Pour des raisons de temps d'entraînement et d'évaluation du modèle, nous avons néanmoins réduit ce jeu de données à environ 200 000 réactions auxquels correspondaient un total de 57959 *templates* uniques.

La méthode **Random** nous permet de réaliser environ 250 000 fausses réactions et une fois le modèle entraîné, le **Recommender** nous en donne environ 270 000, soit un total d'environ 520 000 fausses réactions. Nous n'avons pas cherché à appliquer la méthode **Strict**, estimant que les deux méthodes citées précédemment étaient suffisantes.

4.3 CheckMol et spécialisation

En cherchant à remplir notre objectif qui est d'améliorer le filtre d'AiZynthFinder, nous avons tenté de spécialiser les filtres entraînés. Ainsi, nous avons dû filtrer nos bases de données de réactions pour ne garder que celles possédant certains groupes caractéristiques chimiques.

Nous nous sommes confrontés à l'impossibilité de trouver de telles bases en "open source" facilement disponibles. Nous avons donc décidé de les constituer par nous-même et il nous a été conseillé d'utiliser le logiciel "CheckMol" [10] implémenté en Pascal. Ce logiciel permet à partir d'une molécule donnée en entrée sous format .mol de renvoyer les groupes caractéristiques chimiques présents dans la molécule et ce parmi une liste de plus de 200 groupes.

Ceci nous a donc permis d'avoir des bases de données de réactions dont les produits possédait certains groupes exclusivement. Le lien donné ici détaille l'ensemble des groupes chimiques gérés par l'application. Nous avons décidé d'en sélectionner 12 parmi les 200 : Aldéhyde, Cétone, Enamine, Alcool, Phenol, Ether, Amine, Halogène, Acide Carboxylique, Alcène, Urée et Lactone. Certaines familles comme les alcools, les éthers ou les amines ont été sélectionnées car elles reviennent souvent dans les molécules organiques. D'autres comme l'Urée ou la Lactone ont été sélectionnées car pouvoir faire la rétrosynthèse de molécules contenant ces groupes peut se révéler particulièrement utile que ce soit dans les domaines de l'agro-alimentaire ou du médical. Nous avons ensuite entraîné douze réseaux de neurones spécialisés correspondant à chacun des douze groupes nommés ci-dessus.

4.4 Préprocessing

Le préprocessing désigne toutes les étapes que l'on réalise sur nos données "basiques" afin d'obtenir des données numériques que l'on peut fournir au modèle étudié. Nous présentons donc ici l'architecture des différentes de bases de données que l'on a utilisées qu'elles aient servi d'intermédiaires ou non, (Architecture résumée par l'image fournie à la fin du paragraphe).

Quelles sont les données fournies en entrée aux modèles ?
Afin de comparer les modèles que nous développons avec ceux de AiZynthFinder , nous donnons

à nos modèles les mêmes données. Ainsi, en notant fp la fonction "Fingerprint", chaque modèle de filtre prend en entrée une réaction encodée de façon numérique sous la forme d'un *tuple* contenant "$fp(produit)$" et le reste de la réaction sous la forme "$fp(\text{produit}) - \sum(fp(\text{réactifs}))$" et renvoie en sortie la probabilité que la réaction soit faisable (entre 0 et 1). Le réseau de neurones **Recommender** prend quant à lui en entrée la somme $\sum fp(\text{réactifs})$ et renvoie une liste de 57959 probabilités. La valeur i dans cette liste correspond à la probabilité que les réactifs dont la somme des fingerprints a été passée en entrée soient compatibles avec le *template* i. (*Le dictionnaire faisant la correspondance entre les templates et les indices i est disponible sur le GitHub.*) Tout cela est résumé par le schéma suivant :

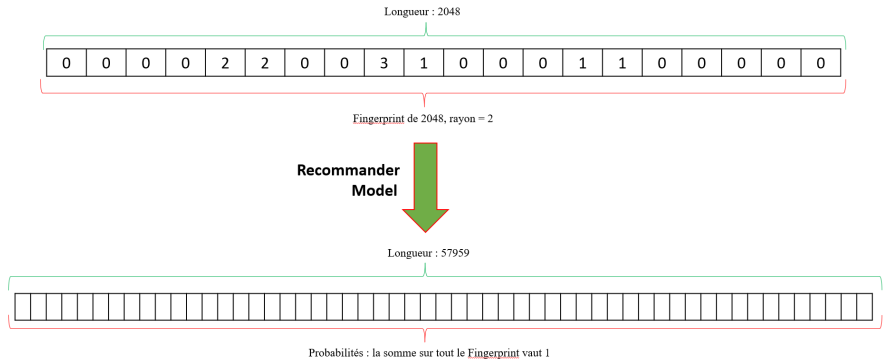


Figure 8 – Fonctionnement du "Recommender"

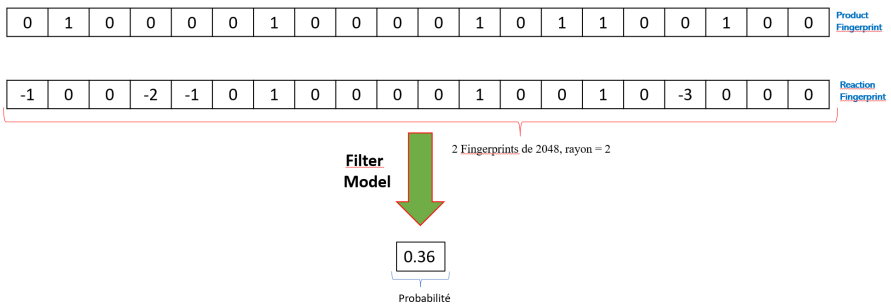


Figure 9 – Fonctionnement du "Filter Model"

Les bases de type "Groupes" contiennent pour chaque réaction l'ensemble des groupes caractéristiques contenus dans le produit de la réaction sous la forme d'une chaîne de caractères incluant la suite des groupes sous la forme "# + numéro du groupe en 3 chiffres".
De plus, le stockage des fingerprints ne peut pas se faire de manière naïve sans quoi les bases de donnée auraient des tailles bien trop importantes (il faudrait stocker 2 fois 2048, soit 4096, entiers par réaction). Nous avons ainsi mis en place un format visant à diminuer cette taille en

mémoire. Rappelons que nous stockons pour chaque réaction des sommes de fingerprints donc des tableaux qui peuvent contenir d'autres entiers que 0 et 1. Ainsi, pour chaque fingerprint fp de longueur 2048, nous ne stockons en mémoire que les indices i pour lequel $fp[i]$ est non nul ainsi que les valeurs $fp[i]$ correspondantes, ce qui permet de réduire considérablement la taille occupée en mémoire par les bases de données de type "Fingerprints".

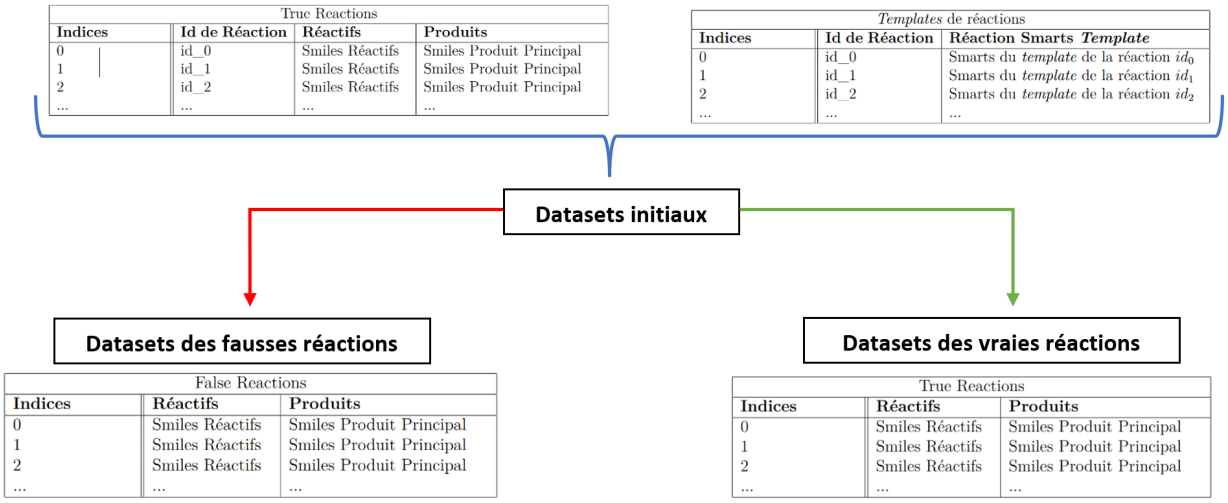


Figure 10 – Architecture générale de nos bases de données

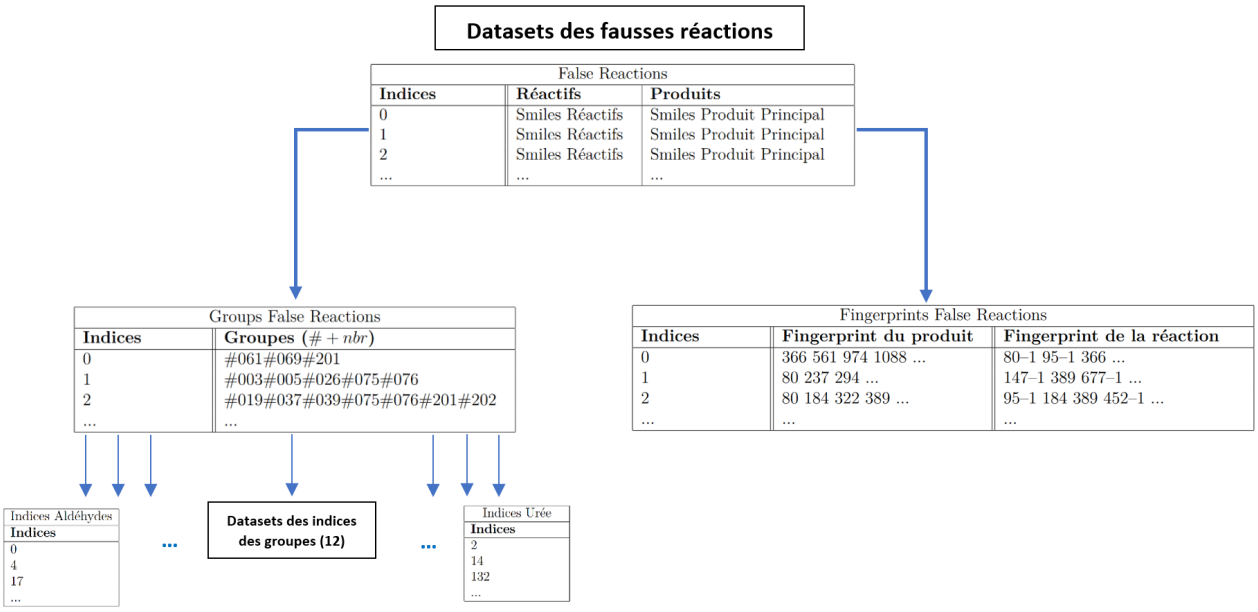


Figure 11 – Architecture des datasets pour les fausses réactions

La même architecture se retrouve pour le dataset des vraies réactions et le dataset de validation.

5. MÉTHODES ET ALGORITHMES DÉVELOPPÉS

5.1 Premiers modèles de filtres : Classifieur Bayésien Naïf, Régression Logistique et Random Forest

Dans un premier temps, nous avons tenté d'implémenter des modèles de Machine Learning classiques simples, ne nécessitant pas de réseaux de neurones, pour s'approprier les données et tenter de comparer leurs performances à celles de AiZynthFinder.

5.1.1 • Classificateur Bayésien Naïf

Le Classificateur Bayésien Naïf est un modèle très simple d'apprentissage supervisé. L'hypothèse principale au centre de ce modèle est que les descripteurs (données en entrées) sont indépendantes conditionnellement aux données de sorties. Comme son nom l'indique, cet algorithme repose essentiellement sur la formule de Bayes :

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

Nous avons choisi d'utiliser un algorithme de type "Complement Naïve Bayes" (CNB) pour lequel la probabilité recherchée est proportionnelle à :

$$argmin P(y) \prod \frac{1}{P(w|y)^{f_i}}$$

Ce choix s'explique par le fait qu'on donne en entrée de notre modèle une base de données déséquilibrée qui contient plus de vraies réactions que de fausses réactions. Contrairement aux réseaux de neurones développés ci-après, les algorithmes développés dans cette section ne prennent qu'une entrée. Nous avons donc décidé de prendre comme entrée pour ce modèle CNB la concaténation des listes des fingerprints du produit et de la somme des réactifs, soit un array total de 4096 valeurs. Nous récupérons alors la probabilité que la réaction soit faisable.

5.1.2 • Régression Logistique

Après le classificateur bayésien naïf, nous avons tenté d'implémenter une régression logistique. Ici, le modèle recherche à déterminer à partir des données fournies lors de l'apprentissage une façon de prédire et d'estimer la faisabilité des réactions en entrée du modèle. Plus précisément, une régression logistique est particulièrement utile lorsque la variable que l'on cherche à estimer ne prend que deux valeurs (binaire, 0 ou 1). La fonction de répartition du modèle de régression logistique est de la forme d'une sigmoïde à savoir :

$$f(x) = \frac{1}{1 + e^{-x}}$$

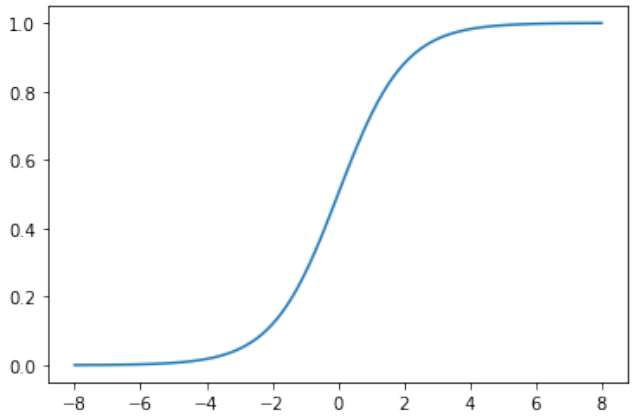


Figure 12 – Fonction Sigmoïde

Le modèle que nous avons développé prend en entrée la réaction sous la forme d'une concaténation de $(fp(\text{produit}) et \sum fp(\text{réactifs}))$ et renvoie une sortie sous la forme $[p, 1-p]$ où p désigne la probabilité pour la réaction d'être faisable. Nous pouvons ensuite fixer un seuil à partir duquel on considère la réaction comme faisable ou non. En raison des temps de calculs,

nous avons entraîné notre régression ici seulement sur une partie des réactions prise au hasard (200 000 environ).

5.1.3 • Random Forest

La Random Forest est aussi une méthode classiquement utilisée en machine learning. Ici, on choisit un nombre fini (noté $n\_estimators$ dans le code et fixé à 100) d'arbres de décision. Les arbres de décision permettent de découper un ensemble d'observations en groupe homogène en se basant sur des règles appliquées sur les variables descriptives d'entrée du modèle. Les algorithmes de type "Random Forest" permettent d'étendre les notions d'abres de décision afin de construire des modèles prédictifs plus stables.

Il est possible de construire un arbre de décision de la façon suivante. On crée d'abord la racine de l'arbre qui contient et représente toutes les observations d'apprentissage. Il faut ensuite choisir une méthode de segmentation des variables pour construire les fils successifs de la racine. La méthode CHAID (CHI-Squared Automatic Interaction Detection) est souvent utilisée. Elle est basée notamment sur l'utilisation d'une loi du $Chi - 2$. Nous ne la détaillons pas ici et renvoyons à l'article suivant pour plus d'informations [11]. Pour achever la construction d'un tel arbre de décision, il faut ensuite fixer une condition d'arrêt d'exploration afin d'obtenir toutes les feuilles de l'arbre. Une fois l'arbre construit, diverses méthodes (élaguage, ...) permettent de traiter la donnée obtenue.

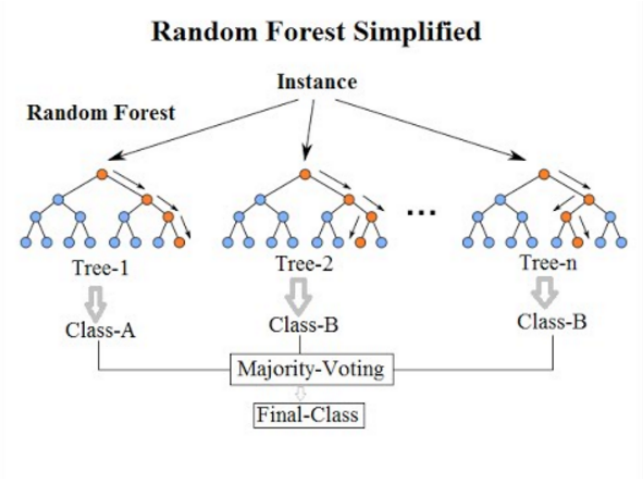


Figure 13 – Illustration du fonctionnement d'une Random Forest

Les algorithmes de type "Random Forest" proposent de construire plusieurs arbres de

décision à partir d'échantillons tirés de façon aléatoire dans les données d'entraînement. Une fois ces arbres de décision construits, le modèle "Forêt Aléatoire" interroge chacun des 100 arbres de décision et renvoie le résultat majoritaire. Notre algorithme ne prend en entrée qu'une partie des données (pour des raisons de temps d'entraînement) choisies au hasard dans la base de données totale. Il renvoie la probabilité que la réaction soit faisable selon le modèle sous la forme $[p, 1-p]$.

5.2 Réseaux de Neurones

Un autre type de modèle extrêmement fréquent en Machine Learning est le réseau de neurones. Comme son nom le suggère, le réseau de neurones s'inspire du fonctionnement du cerveau humain pour développer un modèle de Deep Learning qui répond à plusieurs besoins. Classiquement, et historiquement, l'élément de base du réseau de neurones est le perceptron. Un perceptron prend en entrée un certain nombre d'inputs et leur affecte un certain poids. On calcule ensuite la somme pondérée selon un seuil b_i et on renvoie la sortie après application d'une fonction dite fonction d'activation.

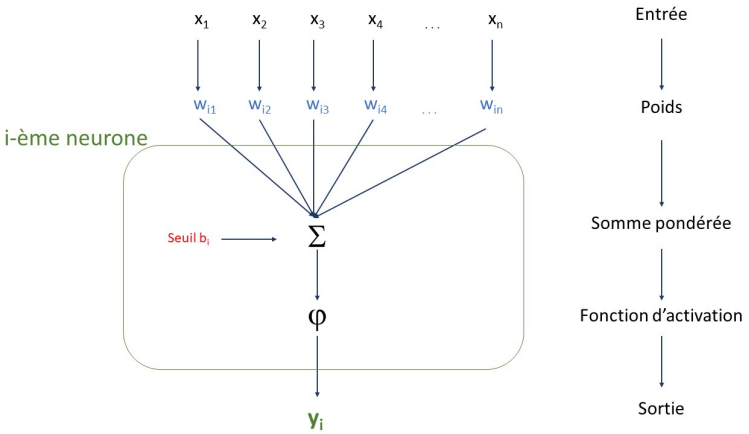


Figure 14 – Schéma fonctionnel d'un perceptron d'un réseau de neurones artificiels

Ces différents perceptrons sont connectés entre eux selon différents moyens afin de créer le réseau de neurones en lui-même. Ils interagissent de façon à modifier les poids des neurones adjacents.

Le propre des réseaux de neurones est leur capacité à apprendre et à s'entraîner par eux-

mêmes en enchaînant propagation en avant et back-propagation pour chercher à équilibrer les poids en accord avec les données d'entraînement fournies. Cela se fait par le biais de plusieurs fonctions de "monitoring" qu'on appelle des métriques. Cela peut être une fonction de perte, une fonction de précision, etc...

5.2.1 • Le **Recommender** : Un exemple de réseau de neurones

Le premier réseau de neurone que nous avons dû développer est le **Recommender** . En effet, nous avons dû créer une base de fausse réaction afin d'entraîner nos différents modèles de filtration. Il est compliqué de se procurer de telles bases de données dans la littérature. Ce réseau permet d'obtenir une telle base par la méthode **Recommender** .

Nous n'avons pas réussi à récupérer le **Recommender** créé par les auteurs de AiZynthFinder . Nous avons donc décidé de l'implémenter nous-même. Il s'agit d'un réseau de neurones de type "Sequential" (linéaire) implémenté à l'aide de la bibliothèque python tensorflow. Le réseau s'articule de cette façon :

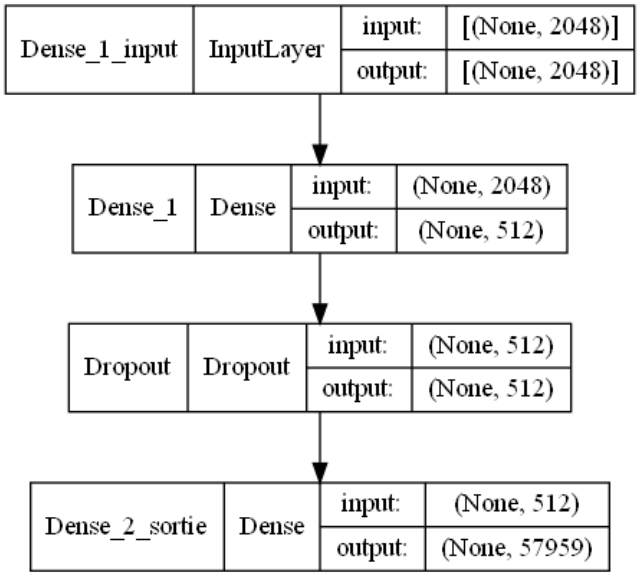


Figure 15 – Architecture du réseau **Recommender**

Explicitons un peu les différents layers utilisés. Nous commençons par un layer de type "Dense" qui récupère les inputs (2 Fingerprints de taille 2048) et qui les transmet à un deuxième layer Dense qui comporte 512 neurones (perceptrons). Rappelons qu'un layer de type Dense se nomme ainsi car il est connecté à toutes les sorties des layers qui le précèdent (ici 4096). Vient

ensuite un layer de type Dropout qui permet de ne conserver que la partie la plus significative des données. Ici, on ne garde que 40% de la donnée transmise par le premier layer Dense. Nous utilisons enfin un dernier layer Dense qui servira d'interface de sortie finale. Nous récupérons ainsi un tableau de longueur 57959 où chaque case contient la probabilité que les réactifs passés en entrée répondent positivement au template correspondant à l'entier $i \in \{0, 1, ..., 57958\}$.

Au sein du layer nommé "Dense_1", nous utilisons une fonction d'activation de la forme "ELU (Exponential Linear Unit). Le dernier layer utilise quant à lui une fonction de type softmax pour produire le vecteur des probabilités finales.

Après avoir entraîné ce modèle, nous l'avons testé sur les 20% des données initales qui n'ont pas servi à l'entraînement. En raison d'un manque de temps, nous n'avons pas pu réaliser plus de tests sur ce recommender. Une fois le recommender en main, nous l'utilisons pour créer notre base de fausses réactions en essayant pour chaque réaction dans la base de données les 20 templates les plus probables et en ne gardant que les réactions qui ne sont pas faisables.

5.2.2 • Reprise des modèles développés par AiZynthFinder

Filter Model d'origine

Notre objectif étant d'améliorer AiZynthFinder , nous avons étudié leur modèle de filtration. Il s'agit d'un réseau de neurone comportant deux *inputs* de longueur 2048, l'un pour le *fingerprint* du produit étudié et l'autre pour celui de la réaction. Ces deux entrées passent ensuite à travers un layer *dense* puis sont concaténées par un layer *dot* avant d'être renvoyés en sortie. Les développeurs d'AiZynthFinder ont stocké les paramètres du modèles dans le github avec la méthode save_model de Tensorflow. Grâce à cela, nous avons pu le récuperer avec la méthode load_model du même package. Ainsi, nous pourrons non seulement comparer notre futur modèle au leur mais aussi faire du Transfer Learning.

Intéressons-nous rapidement aux différents layers de ce modèle. Les deux layers Dense situés à la suite immédiate des deux inputs ont pour fonction d'activation la fonction "ELU". Le modèle utilise ensuite un layer de type dot qui réalise le produit scalaire des deux entrées pour renvoyer un réel qui passe ensuite par une sigmoïde, fonction d'activation du dernier layer de sortie.

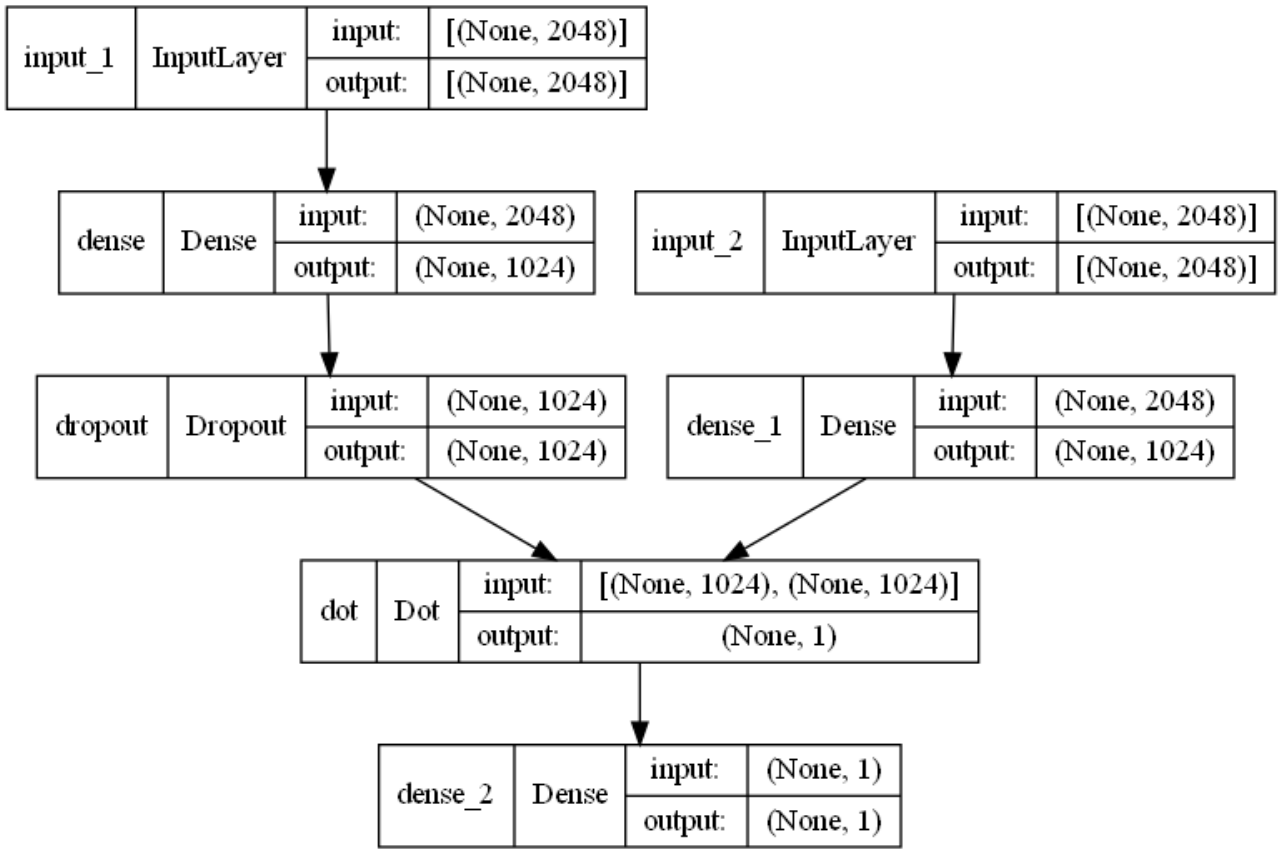


Figure 16 – Architecture du filter model d'AiZynthFinder

Spécialisation

Notre hypothèse est que notre modèle sera plus performant si nous restreignons son champ d'action à certains groupes caractéristiques chimiques. Pour cela, il existe plusieurs méthodes. Nous aurions pu créer un nouveau modèle de zéro et l'entraîner sur des bases restreintes mais, ayant sous la main un bon modèle général nous avons décidé de faire du Transfer Learning (TL).

Le Transfer Learning fonctionne de manière assez simple en quelques étapes :

1. Récupérer un modèle entraîné au préalable
2. "Geler" les paramètres de telle sorte à ce qu'ils ne soient plus modifiable même durant une phase d'entraînement future.
3. Supprimer le layer de sortie de ce modèle et le remplacer par un nouveau modèle qui n'a jamais été entraîné.

4. Lancer l'entraînement sur la nouvelle base de données spécialisée.

5. (Optionnel) Dégeler tous les paramètres et refaire une phase d'entraînement en baissant drastiquement le *learning_rate*

Une fois toutes ces étapes effectuées, on dispose théoriquement d'un nouveau modèle spécialisé.

En pratique, nous avons créé un nouveau modèle à deux entrées dont les deux branches avant de se concaténer sont les mêmes que celle du modèle d'AiZynthFinder avec les paramètres chargés depuis le github. Voici son schéma :

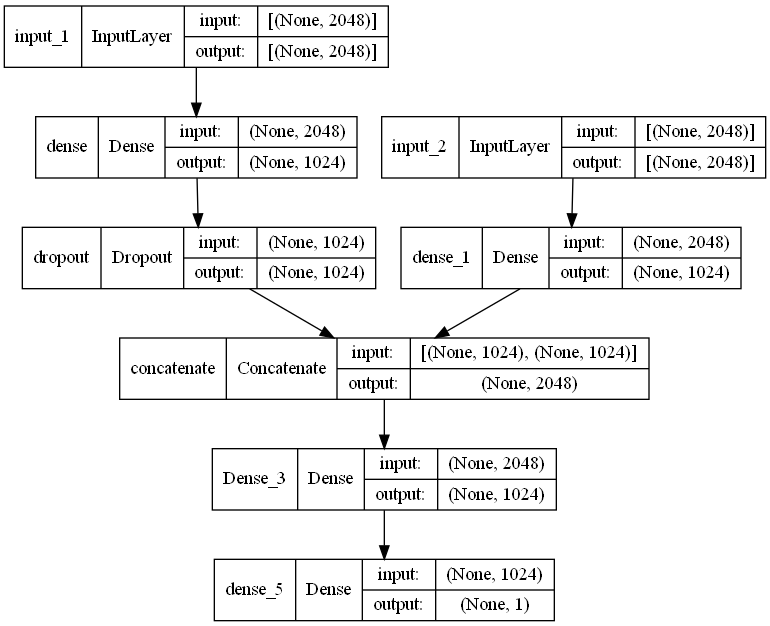


Figure 17 – Architecture de notre filter model

Ici, nous avons choisi comme fonction d'activation pour les deux layers dense respectivement une fonction d'activation de type RELU (Rectified Linear Unit) et une fonction sigmoïde. Tous les paramètres entraînables avant le layer concatenate sont mis en mode non-trainable pour qu'on ne puisse pas les entraîner. Nous entraînons ensuite 12 modèles pour chaque famille chimique parmi celles auxquelles on s'intéresse.

6. RÉSULTATS OBTENUS, DIFFICULTÉS RENCONTRÉES ET PISTE D'AMÉLIORATION

6.1 Méthodes utilisées

Avant d'analyser nos résultats il est primordial d'avoir une méthode rigoureuse d'évaluation des modèles.

Notre objectif étant d'améliorer AiZynthFinder nous allons devoir tester notre modèle et celui d'AiZynthFinder sur les mêmes bases. Le problème est que la base que nous avons utilisé pour entraîner notre modèle et que nous comptions utiliser pour l'évaluer est la base USPTO sur laquelle AiZynthFinder s'est entraîné. Hors il est impératif que l'évaluation d'un modèle se fasse sur un jeu de donnée qui n'a jamais été vu par le modèle en phase d'entraînement. Nous avons donc récupéré une autre base de données à laquelle nous avons retiré les réactions présentes dans USPTO. Ainsi, nous avions une base de données neuve qu'aucun des modèles, le notre ou celui d'AiZynthFinder , n'auront vu en phase d'entraînement. En sachant que les développeurs d'AiZynthFinder ont récupéré leurs bases en 2018, il est impossible que ces réactions aient été déjà vues même si nous tentons d'évaluer leur modèle sur des données postérieures.

L'évaluation d'un modèle, quel qu'il soit, se fait de la même façon. On scinde tout d'abord nos données en un jeu d'entraînement et un jeu de validation. Ce jeu de validation ne sera jamais vu par le modèle pendant la phase d'entraînement. Dans notre cas, nous n'avons pas besoin de le faire particulièrement pour notre base de vraies réactions car nous en utilisons une complètement différente. Néanmoins nous avons dû le faire pour les fausses réactions car, même si AiZynthFinder n'a jamais vu les fausses réactions que nous avons généré nous-même, notre modèle est susceptible de les avoir vu.

Une fois cela fait, nous pouvons entraîner notre modèle sur les bases d'entraînement en ajustant les hyper paramètres comme le nombre d'*epochs*, le *learning rate*, les layers ou encore les fonction de callbacks (notamment *Early_stopping* et *ReduceLROnPlateau*). Afin d'éviter de sur-ajuster les données, on s'entraîne en partitionnant notre base de données en une base de

test et une base d'entraînement. Nous avons choisi comme il est souvent fait dans la littérature une partition 80/20. Ainsi, après cette étape, nous avions un modèle entraîné, prêt à être évalué et comparé à AiZynthFinder .

Avant de pouvoir continuer dans l'évaluation des modèles, nous sommes obligés de faire un point sur les seuils de décisions. Un modèle de Machine Learning qui a pour but de classifier des données va renvoyer pour chaque prédiction un flottant entre 0 et 1 représentant la "probabilité" d'être ou non dans la classe. C'est ensuite au programmeur de décider à partir de quelle probabilité on considère que l'élément est dans la classe. Ce seuil est fixé humainement et dépend du problème et de la solution souhaitée. En effet, si le seuil est très élevé, il n'y aura presque aucun faux positif et, à l'inverse, s'il est très bas, ce sont les faux négatifs qui seront en très faible nombre comme on peut le voir dans la figure (METTRE LE NUMERO DE LA FIGURE).Les vrais positifs étant le nombre de données positives bien classées par le modèle et les faux positifs celles mal classées avec la même chose pour les données négatives.

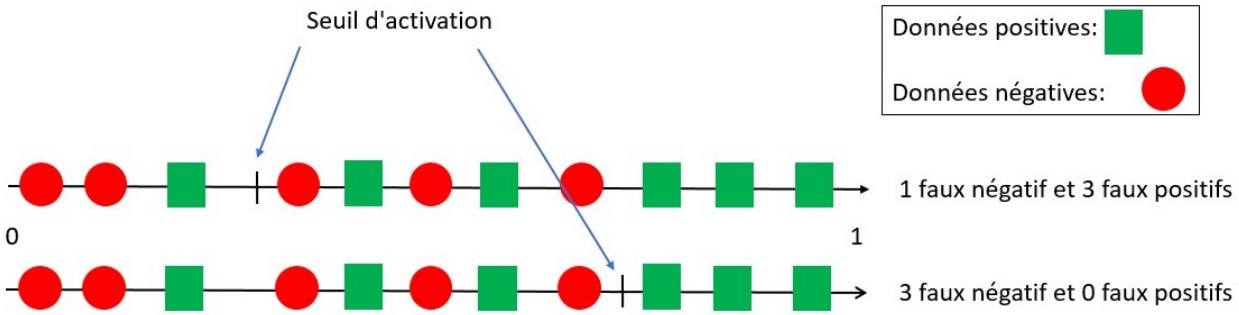


Figure 18 – Importance du seuil d'activation

Afin d'évaluer un modèle, il faut comparer la prédiction de ce dernier avec les valeurs attendues pour la base de test. On utilise plusieurs valeurs pour cela. Les valeurs numériques que nous avons décidé de regarder sont la précision, le rappel, le score f1 et l'aire sous la courbe ROC. Les trois premiers sont définis par les fonctions suivantes :

$$precision = \frac{vrais\,positifs}{vrais\,positifs + faux\,positifs}$$

$$rappel = \frac{vrais\,positifs}{vrais\,positifs + faux\,négatifs}$$

$$score\,f1 = \frac{2 \times precision \times recall}{precision + recall}$$

La courbe ROC est la courbe représentant le taux de vrais positifs en fonction du taux

de faux positifs. L'idéal étant qu'elle soit le plus proche de la fonction qui a x associe 1 pour $x > 0$ et 0 pour $x = 0$, nous pouvons ainsi définir un nouvel indicateur : l'aire sous la courbe ROC qui nous donnera la qualité du modèle en fonction de sa proximité avec 1.

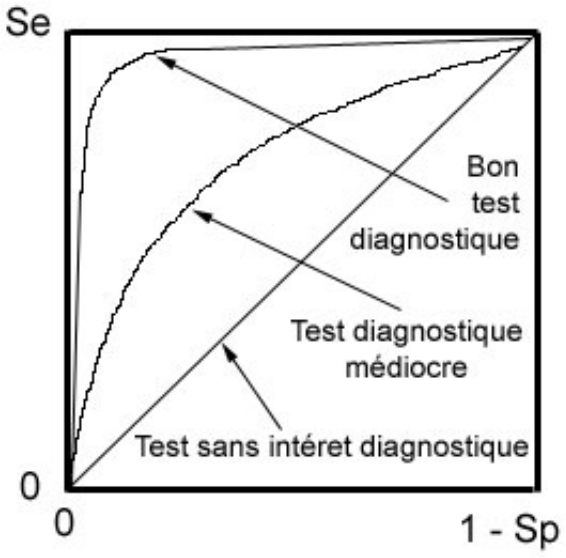


Figure 19 – Exemple de courbes ROC

Une fois nos hyper paramètres choisis, nous avons vérifié la pertinence de ce choix en faisant un apprentissage par k-folds. L'idée est de diviser la base de données en n puis de successivement apprendre sur une fraction égale à $\frac{n-1}{n}$ de la base de données et ensuite évaluer sur la fraction restante. Ceci permet de combler le risque d'avoir une hétérogénéité dans la base de données et d'avoir donc un apprentissage ou un test biaisé.

Ensuite, afin de valider notre modèle, nous faisons un dernier test qui est de tenter d'entraîner notre modèle sur une base de données avec des cibles qui ont été mélangées et qui n'ont donc aucun sens.

6.2 Présentation des résultats : résultats du filtre

Nous présentons ici les résultats de notre modèle de filtre, selon les différentes méthodes utilisées. Pour des questions de place et de lisibilité, nous ne présenterons ici qu'une partie des résultats obtenus.

6.2.1 • Modèles autres que les réseaux de neurones

Nous avions commencé par implémenter des modèles plutôt simples autres que les réseaux de neurones. Nous rassemblons ici l'ensemble des résultats les concernant. Pour rappel, nous avions un CNB (classifieur bayésien), une Regression Logistique et une Random Forest. Nous les comparons avec AiZynthFinder. Nous obtenons ainsi les résultats suivants :

Résultats sur la base de validation "globale"				
	CNB	**RL**	**RF**	**AZF**
TP	0.95	0.744	0.861	0.878
TN	0.027	0.0968	0.0817	0.077
FP	0.136	0.134	0.01853	0.0019
FN	0.072	0.0232	0.0383	0.043
Precision	0.969	0.969	0.957	0.953
Recall	0.846	0.846	0.979	0.997
Fscore	0.904	0.904	0.968	0.975

Le tableau ci-dessus résume rapidement certaines caractéristiques des algorithmes entraînés sur USPTO et testés sur la base de validation. Globalement, AiZynthFinder est un peu meilleur mais la Random Forest développée s'en rapproche bien.

Nous disposons aussi des courbes ROC suivantes qui font le même constat que le tableau précédent, à savoir que AiZynthFinder semble rester meilleur mais que la Random Forest n'est pas mauvaise non plus.

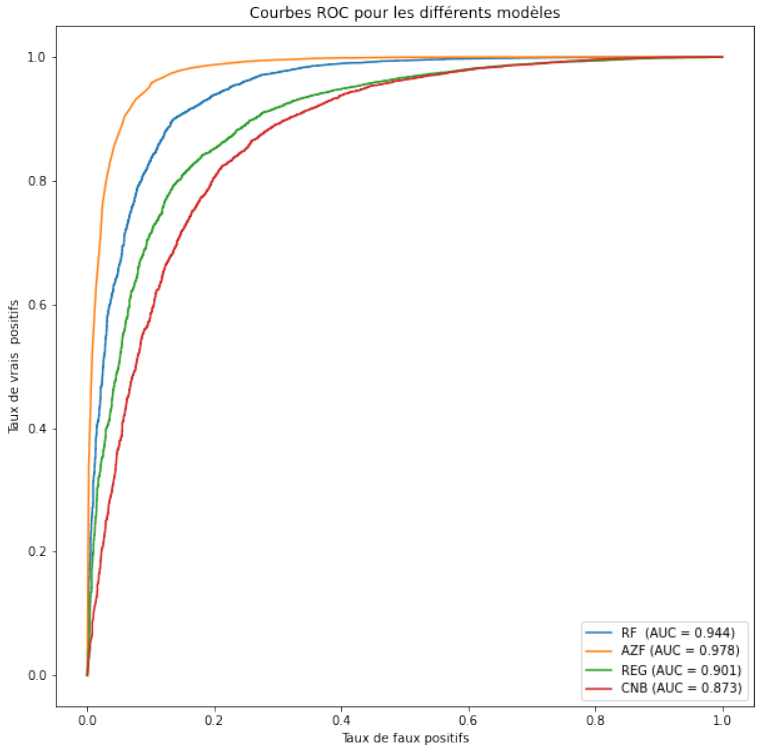


Figure 20 – Courbes ROC pour les modèles évoqués ci-dessus
Dans l'ordre : *random forest, AizynthFinder, Regression, CNB*

Dans notre volonté de développer un filtre encore plus adapté, il semble donc logique de nous tourner vers du transfer learning.

6.2.2 • Entraînement des réseaux de neurones

Pour suivre l'entraînement des réseaux de neurones, nous suivons l'évolution des métriques *accuracy* et *loss*. Lors de l'entraînement, une fraction du jeu de données est utilisée comme données de validation/test et n'est pas utilisée dans l'entraînement du modèle. Elle est représentée par la courbe orange. Nous avons limité le graphe à 40 epochs. Remarquons qu'on commence déjà à une accuracy plutôt élevée puisqu'on récupère une partie du modèle de AiZynthFinder .

Nous entraînons ainsi 12 réseaux de neurones, un par famille chimique étudiée. Puis,

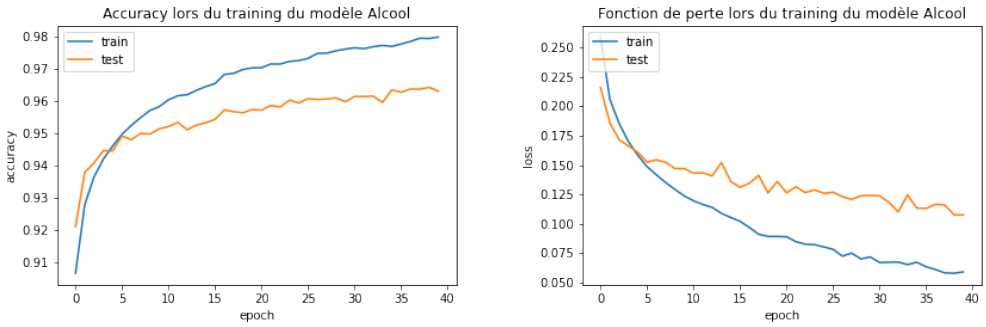


Figure 21 – Evolutions des métriques intéressantes au cours des epochs

nous comparons avec AiZynthFinder sur les bases de données spécialisées issues de la base de validation. Nous présentons certaines courbes ROC ci-après.

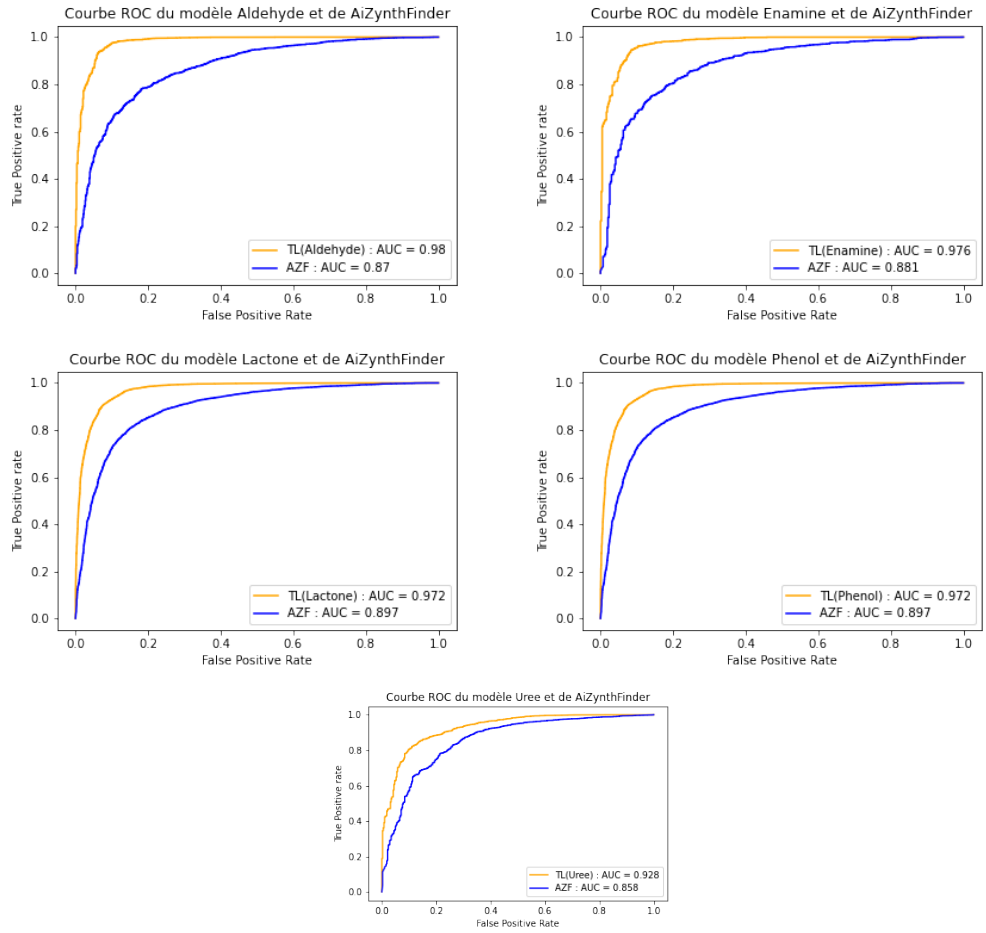


Figure 22 – Courbes ROC par comparaison avec AiZynthFinder testés sur des datasets spécialisés respectifs

Nous remarquons ainsi que notre modèle est légèrement meilleur que AiZynthFinder ici. Néanmoins, cela est à contraster car les modèles spécialisés ne sont meilleurs que sur les jeux de données dont la proportion taille de la base spécialisée / taille totale est faible. Ainsi, par exemple, la courbe ROC de *AiZynthFinder* est légèrement meilleure que la nôtre dans le cas des *Ether* puisque la fonction ester se retrouve dans un grand nombre de réactions.

6.2.3 • Tableau de détermination du "cutoff"

Pour chaque filtre, il faut encore déterminer un cutoff, i.e. un seuil optimal de fonctionnement. Pour cela, nous avons décidé d'imiter les auteurs de l'article d'*AiZynthFinder* et de réaliser pour chaque modèle un tableau de la forme suivante :

Tableau en fonction des thresholds		
Thresholds	**Recall**	**Specificity**
0.05	0.99	0.65
0.1	0.99	0.68
0.15	0.98	0.70
0.2	0.97	0.74
...	...	...
0.95	0.93	0.90

Dans notre cas, nous voulons rejeter les réactions infaisables. C'est donc le recall qu'il faut regarder en priorité bien qu'il ne faille pas négliger également la spécificité. A titre d'exemple, AiZynthFinder utilise un seuil de 0.05 pour son filtre.

6.2.4 • K-Fold

Nous avons évalué notre modèle en faisant du K-Fold comme expliqué dans la partie précédente. Nous pouvons voir dans la figure suivante le résultats d'un 5-folds évalué sur AiZynthFinder et sur notre modèle.

Au vu du *KFold* réalisé, il ne semble pas qu'il y ait un biais dans les données dû à une mauvaise "répartition" des données. Il n'y a pas une partie des données réelleement plus avantageuse que les autres.

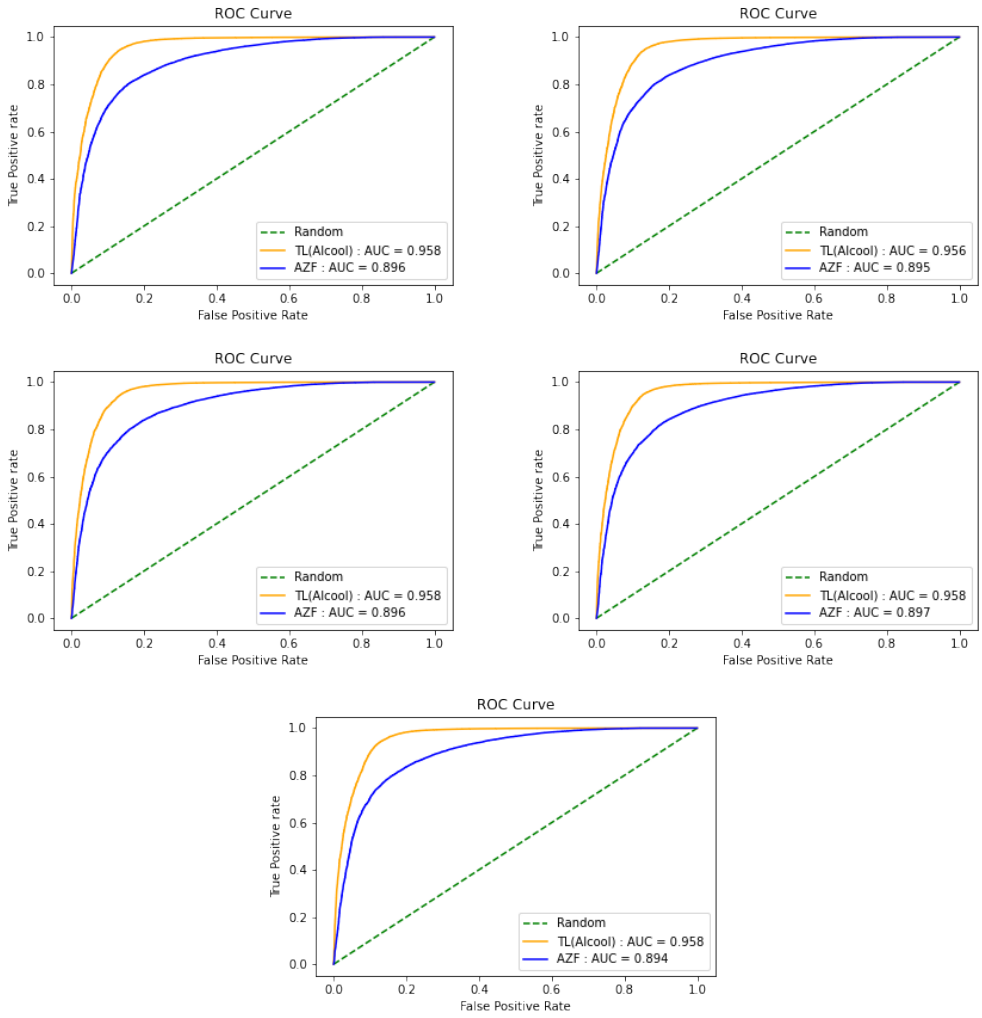


Figure 23 – Courbe ROC de notre modèle "Alcool" et de celui d'AiZynthFinder sur des Alcools et pour un 5-Folds

6.3 Application de notre modèle à la rétrosynthèse

Tentons d'appliquer les modèles développés à la molécule d'Aspirine prise en exemple dans la partie 2.3.2. Ici, on le fait "à la main". Nous récupérons la liste des groupes présents dans la molécule d'Aspirine grâce à CheckMol. Nous choisissons ensuite le filtre *AcideCarboxylique* (car l'aspirine contient ce groupement). Nous appliquons ensuite *AiZynthFinder* pour obtenir la réaction suivante :

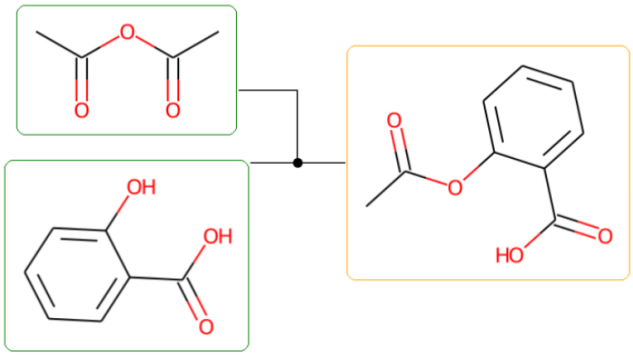


Figure 24 – Application de AiZynthFinder avec notre filter modele Acide Carboxylique

Cette réaction s'apparente à une estérification. Néanmoins, pour la réaliser en vrai, il faudrait certainement ajouter deux ou trois étapes de protection, déprotection et d'activation. Il est difficile de juger à "vue d'oeil" laquelle des deux réactions prédites (cf. partie 2.3.2) est la plus à même d'être réalisée en laboratoire sans analyse plus poussée.

6.4 Difficultés rencontrées

La première difficulté à laquelle nous avons été confrontés est la pluridisciplinarité de ce projet. En effet, les profils des membres du groupe sont très variés avec 3 personnes dont les compétences sont axées informatique et un chimiste. Il a dès le début été difficile de se coordonner pour faire un travail de groupe et répartir les tâches de façon pertinente. Néanmoins,

après une longue réflexion nous avons réussi à travailler en groupe et à communiquer de la meilleure des façons possible compte tenu des profils.

Notre second écueil a été l'obtention de base de données de fausse réaction. En effet, aucune base n'était disponible et nous avons dû créer un autre réseau de neurone (le **Recommender**) pour en faire une nous-même. De manière générale, les bases de données nous ont donnés du fil à retordre. En effet, leur taille était un vrai problème pour l'exécution des codes sur nos machines (à 32 Go de Ram) et nous avons dû utiliser des générateurs afin d'entraîner nos modèles au regard de notre faible mémoire vive. Par ailleurs, les bases avaient très souvent besoin d'être traitées pour enlever les réactions qui posaient problème (éléments manquants, problème dans la spécialisation ou dans la conversion en fingerprint).

La dernière grosse difficulté que nous avons rencontrée est l'obtention de bases "spécialisées". Nous avions initialement prévu d'utiliser les fingerprints mais nous nous sommes aperçus que cette méthode n'était clairement pas assez précise et ce sont nos tuteurs qui nous ont apporté la solution. Ils nous ont conseillé d'installer un logiciel codé en Pascal : CheckMol. C'est ce logiciel qui nous a permis d'obtenir ces bases spécialisées.

6.5 Pistes d'amélioration et d'exploration

6.5.1 • Comment améliorer ce que l'on a développé ?

Le modèle que l'on a développé est loin d'être parfait et est sujet à de nombreuses améliorations possibles. D'abord, il est évidemment possible de tenter de nombreux autres modèles, avec des layers différents (CNN, LSTM, RNN, ...), avec ou sans transfer learning, mais cela demande beaucoup de temps et de puissance de calcul. Dans le même genre d'idée, il peut être intéressant de réaliser un fine-tuning. Une fois le premier modèle obtenu par transfer learning obtenu, on peut dégeler la totalité ou une partie des paramètres que l'on avait gelés et réentrainer le modèle avec un learning rate très faible sur les mêmes données.

De plus, par manque de temps et de puissance de calcul, nous n'avons pas pu réaliser de tests approfondis du recommender. Une analyse du modèle que nous avons développé pourrait permettre d'avoir un recommender encore meilleur et d'avoir potentiellement une influence

positive sur la sélectivité du filtre.

Ensuite, il faudrait trouver une alternative pour réduire le temps de calcul au cours de l'algorithme AiZynthFinder . En effet, pour que les 12 modèles développés par spécialisation soient efficaces, il faut qu'on exécute CheckMol pour chaque molécule rencontrée par AiZynthFinder lorsqu'il établit son arbre. Or cela rajoute un temps considérable dans l'exécution de cette recherche. Ainsi, il serait bon de rechercher un équivalent plus rapide de CheckMol (l'appel au terminal via Python de Checkmol prend environ 0.05 secondes par molécule) ou de trouver un moyen de stocker en mémoire un grand nombre de groupes déjà calculés pour des molécules communes auxquelles on pourrait accéder en temps quasi-constant.

Enfin, en incluant ce travail dans une perspective encore plus ciblée, il serait possible d'ajouter encore de nouveaux modèles (pas seulement 12) qui ne sont plus spécialisés seulement sur des groupes chimiques, mais sur d'autres caractéristiques des molécules (chiralité, point de fusion, etc...)

6.5.2 • Des idées pour la suite

Le modèle développé ici ne constitue en réalité qu'une petite partie de AiZynthFinder et de l'algorithme de rétrosynthèse associé et de ce fait d'autres initiatives peuvent être adoptées pour améliorer cet algorithme. Une première idée serait d'essayer de spécialiser la partie dite "expansion_policy". Nous pourrions tenter d'adapter les mêmes idées de spécialisation des modèles par Transfer Learning au modèle qui régit l'expansion et la recherche de nouvelles routes de réaction afin de tenter d'obtenir des résultats plus ciblés, que ce soit sur un groupe caractéristique chimique ou sur une famille de molécules. Cela pourrait ainsi avoir des applications dans des domaines comme la recherche médicale ou la biochimie.

Par ailleurs, afin de se construire un vrai avis sur la pertinence de notre algorithme il pourrait être judicieux de le comparer avec d'autres algorithmes que celui d'AiZynthFinder .

Enfin, toujours pour donner plus d'applications concrètes à cet algorithme, il pourrait être utile d'implémenter d'autres façons d'attribuer les scores aux routes trouvées. Par exemple, une application extrêmement utile serait de pouvoir disposer d'une base avec les prix des composants chimiques mis en jeu dans la route de rétrosynthèse et de les faire intervenir pour faire ressortir davantage les routes les moins chères (en euros par exemple).

7. CONCLUSION

Pour rappel, nos objectifs, fixés en début d'année étaient les suivants :

— Étudier des articles sur le sujet et analyser les algorithmes existants : Se faire une idée de l'état de l'art afin de guider nos travaux. *Objectif réalisé. Après avoir analysé plusieurs articles et algorithmes, nous avons choisi de nous concentrer sur l'algorithme d'AiZynthFinder pour baser notre projet.*

— Tester ces algorithmes, comparer et analyser leurs résultats. *Objectif réalisé. Nous avons choisi AiZynthFinder pour leurs bons résultats, leur documentation fournie et la facilité d'usage de l'algorithme.*

— Étudier les propriétés chimiques régissant la synthèse afin de les implémenter dans notre algorithme. *Objectif réalisé. Nous avons utilisé les fingerprints pour représenter les propriétés des réactions.*

— En se basant sur les différents algorithmes existants, constituer notre propre algorithme. *Objectif partiellement réalisé. Bien qu'il ne soit pas complètement implémenté et parfaitement au point, nous avons un modèle de Machine Learning qui filtre les réactions en fonction de leur faisabilité.*

— Tester sur différentes bases. *Objectif réalisé. Pour le développer, notre algorithme a été testé sur la base USPTO puis dans une optique de test, sur une autre base de données.*

— Établir des critère d'amélioration de notre algorithme. *Objectif réalisé. Nous avons listé plus haut les pistes d'amélioration et d'exploration qui pourraient être suivie par la suite.*

— Comparer notre algorithme avec des méthodes déjà existantes. *Objectif partiellement utilisé. Nous avons comparé notre algorithme avec celui de AiZynthFinder mais pas avec d'autres algorithmes.*

— Exporter cet algorithme pour le rendre disponible et l'implémenter afin qu'il puisse prendre en compte les diverses bases de données. *Objectif non réalisé*

— Vérifier en laboratoire de chimie des résultats par l'expérience. *Objectif non réalisé*

Nous sommes relativement satisfait de nos résultats. En effet, bien qu'implémenter notre propre algorithme nous eût permis de conclure notre travail d'une façon satisfaisante, nous sommes parvenus à développer le coeur de cet algorithme en produisant des modèles entraînés pour chaque groupe chimique caractéristique.

RÉFÉRENCES

[1] Cayatte M., Guignon R., Lahiani M., Njoo C. et Theveneau M. *Développement d'un outil d'évaluation de la synthétisabilité*. (2020).

[2] Weininger D. « SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules ». In : *American Chemical Society* (1988).

[3] *A Language for Describing Molecular Patterns*. URL : `https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html`.

[4] et Jensen Klavs F. Coley Connor W. Green William H. « RDChiral : An RDKit Wrapper for Handling Stereochemistry in Retrosynthetic Template Extraction and Application ». In : *American Chemical Society* (2019).

[5] Watson I.A., Wang J. et C.A. Nicolaou. « A retrosynthetic analysis algorithm implementation. » In : *J Cheminform* 11.1 (2019).

[6] Genheden S., Thakkar A., Chadimová V., Reymond J-L., Engkvist O. et Bjerrum E. « A fast, robust and flexible open-source software for retrosynthetic planning. » In : *J Cheminform* 12.70 (2020).

[7] *USPTO DataBase, United States Patent and Trademark Office*. URL : `https://www.uspto.gov`.

[8] *ORD-Data, Open Reaction Database*. URL : `https://github.com/open-reaction-database`.

[9] Genheden S., Engkvist O. et Bjerrum E. J. « A Quick Policy to Filter Reactions Based on Feasibility in AI-Guided Retrosynthetic Planning. » In : *ChemRxiv* (2020).

[10] Haider N. « Functionality Pattern Matching as an Efficient Complementary Structure/Reaction Search Tool : an Open-Source Approach ». In : *Molecules* 15.5079-5092 (2010).

[11] J.J. McArdle G. Ritschard. « CHAID and Earlier Supervised Tree Methods ». In : *Contemporary Issues in Exploratory Data Mining in Behavioral Sciences*. New York, NY : Routeledge, 2013, p. 48-74.