UNIVERSITÉ DE
SHERBROOKE

Projet de session : Algorithmes pour la bio-informatique

Qu'est-ce qui n'est pas un motif ?

Octobre - Décembre 2022

Rapport de projet de bio-informatique

Livrable 3

Étudiants :
Benjamin JURCZAK
Thomas BALDUZ

CIP :
jurb1001
balt1201

Responsable :

Manuel Lafond

Table des matières

1 Introduction **1**

2 Revue de littérature **1**
2.1 Deux articles sur les motifs et l'apprentissage automatique 1
2.2 Résumé d'un article scientifique 2

3 Méthodologie **4**
3.1 Elaboration de l'ensemble d'entraînement 4
3.2 Les classificateurs 6
3.2.1 SVM 6
3.2.2 Régression logistique 7
3.2.3 Réseaux de neurones multicouches 7
3.3 Le code 8
3.3.1 Patron de conception contrôleur 8

4 Résultats **8**
4.1 Démarche pour chaque méthode 8
4.2 SVM 9
4.3 Logistic Regression 10
4.4 Neural Network Classifier 12

5 Comparaison des classificateurs binaires **13**

6 Conclusion **15**

1 Introduction

Le but de ce projet est de mettre l'apprentissage machine au service de la bio-informatique en réalisant un classificateur binaire. La mise en oeuvre d'un tel modèle aura pour but de spécifier si une séquence d'entrée est ou n'est pas un **motif**, terme défini dans le sujet comme "une séquence qui a une structure similaire à d'autres séquences connues qui ont un rôle biologique important".

La découverte de motifs ainsi que leur détection/extraction dans des grandes séquences constituent des enjeux majeurs en biologie. En effet, trouver des motifs dans une séquence permet d'obtenir des renseignements sur la séquence observée et en déduire par exemple son rôle biologique. Le succès que connaît l'apprentissage machine ces dernières années pousse les chercheurs à tenter de l'appliquer au domaine de la recherche de motifs.
Il existe de nombreux motifs connus et de nombreuses databases qui les répertorient, le plus souvent en fonction des espèces (par exemple pour les motifs de l'Homo Sapiens). D'un autre côté, il y aussi et surtout des séquences qui ne sont pas des motifs, c'est-à-dire qui ne traduisent aucun rôle biologique particulier ; mais dans notre cas, ces séquences ont autant d'importance que des motifs. Il ne suffit pas d'entraîner un modèle de machine learning avec des instances de motifs, il faut aussi avoir quelques mauvais exemples, c'est-à-dire des séquences qui ne sont pas des motifs.

Dans ce projet d'élaboration d'un classificateur binaire, un certain nombre de problématiques se posent. En effet, pour parvenir à l'élaboration de notre modèle final, il va falloir passer par plusieurs étapes : la création de notre ensemble d'entraînement (motifs + non motifs), choisir le bon type de classificateur (machine à vecteur de support, réseaux de neurones, ...), ou encore appliquer les bons pré-traitements sur les données avant l'entraînement. Finalement, le but du projet est, étant donné un ensemble de séquences (des motifs et des non motifs), que notre classificateur ait une erreur de classification la plus faible possible sur cet ensemble (qu'un maximum de séquences soient bien classées).

2 Revue de littérature

2.1 Deux articles sur les motifs et l'apprentissage automatique

La lecture des articles proposés dans le sujet nous a permis de nous familiariser avec les algorithmes de machine learning déjà existant pour le problème d'extraction de motifs.

Ces deux articles traitent des différentes méthodes pour identifier et extraire les motifs à partir de séquences d'ADN ou d'ARN. Concernant l'article "A survey on deep learning in DNA/RNA motif mining", il nous apprend que la recherche de motifs est divisée en deux principales approches : la méthode d'énumération et la méthode probabiliste. Ce n'est que grâce à la nouvelle méthode de séquençage ChIP-Seq publiée en 2007 que les techniques de machine learning ont pu être appliquées à la recherche de motifs. L'ensemble de données que nous choisirons doit donc être de type ChIP-Seq. L'article présente ensuite les résultats d'une étude appelée DeepRam qui a pour but de comparer une dizaine d'algorithmes de deep learning sur leur capacité à identifier des motifs. Il apparaît que c'est l'algorithme ECBLSTM qui est le plus efficace, et que plus le modèle est complexe, plus il est efficace. Il faut néanmoins nuancer et dire que la complexité de l'algorithme doit dépendre du volume de données traitées pour éviter l'over-fitting (capacité d'un modèle à apprendre par coeur les données d'entraînement). Le pré-traitement des données joue également un rôle important car il augmente la

robustesse du modèle et sa capacité à généraliser. Deux méthodes de pré-traitements existent : le one hot encoding et le sequence embeding. Il nous appartiendra de choisir parmi ces deux approches .
Le deuxième article commence par présenter le séquencage HTS qui a permis d'appliquer des méthodes de deep learning à des problèmes de biologie. Les parties intéressantes pour notre projet sont les chapitres 2 et 4. Le chapitre 2 s'intéresse à l'algorithme de recherche de séquence MEME qui est de type probabiliste, et présente la méthode EXTREME qui consiste à appliquer l'algorithme MEME avec une implémentation en ligne de l'algorithme Expectation-Maximization. La différence de performance provient en partie du fait que, contrairement au batch EM algorithm, chaque itération du online EM algorithm opère sur une observation X_i et non pas sur tout le dataset X. Le chapitre 4 présente les CNN (réseaux de neurones convolutifs) qui, grâce à leurs couches de convolution, peuvent identifier les motifs. La différence avec l'algorithme EXTREME est que les CNN correspondent à une approche supervisée et non pas non-supervisée comme EXTREME. Le modèle DanQ est ensuite étudié car il présente de bons résultats et peut s'expliquer avec des arguments biologiques.

Ainsi, en bioinformatique, il existe plusieurs méthodes de classification de motifs. Ces méthodes permettent de reconnaître et d'identifier des séquences de nucléotides ou d'acides aminés dans les génomes et protéomes. L'apprentissage automatique est une méthode couramment utilisée ces dernières années pour classer les motifs. Cette technique utilise des algorithmes d'apprentissage pour détecter des motifs dans les données, en utilisant des exemples précédemment annotés pour guider l'apprentissage du modèle. L'avantage de l'apprentissage automatique est qu'il peut être utilisé pour traiter des quantités importantes de données, ce qui en fait une option attrayante pour l'analyse de grandes bases de données génomiques ou protéomiques.

Pour revenir à notre projet, dans notre cas, nous devons, à partir d'une séquence d'entrée et dire si c'est ou non un motif via un classifieur binaire. Bien qu'il s'agisse aussi de motifs, notre problème se démarque assez de ce que nous avons décrit plus haut. En effet, dans la littérature, on trouve surtout (voire presque exclusivement) des méthodes qui permettent l'**extraction** de motifs sur une séquence donnée et pas de la classification. Pour nous, l'enjeu sera la tâche de prédiction, car il ne suffira pas de parcourir la séquence pour y trouver ou non des motifs, mais bien de prédire si ladite séquence est elle-même un motif. Il faudra notamment passer par les étapes délicates de création de l'ensemble d'entraînement et de mise en place des modèles prédictifs.

2.2 Résumé d'un article scientifique

L'article scientifique en rapport avec notre sujet que nous avons décidé de résumer est le suivant : "A Contrastive Learning Pre-Training Method for Motif Occupancy Identification" rédigé par Ken Lin, Xiongwen Quan , Wenya Yin et Han Zhang. Nous avons fait le choix de cet article car il traite également de l'élaboration d'un classificateur binaire, mais s'intéresse aux liens entre motifs et facteurs de transcription. Il pourra ainsi nous aider à voir le problème sous un autre angle.
Il est disponible à l'adresse cliquable suivante : `https://www.mdpi.com/1422-0067/23/9/4699/htm#`

Les facteurs de transcription (TF) sont des protéines qui peuvent se lier à des séquences d'ADN spécifiques et contrôler la transcription de l'ADN en ARN messager. Identifier l'occupation des motifs par ces protéines est une tâche de classification binaire qui peut être résolue en utilisant des méthodes basées sur les séquences d'ADN. Plusieurs de ces méthodes ont été proposées ces dernières années, mais elles ont toutes des limites en termes de performance et d'interprétabilité biologique.

Dans ce travail, les chercheurs ont proposé une méthode de pré-entraînement basée sur l'apprentissage contrastif et la similarité d'édition de séquences pour encoder des séquences d'ADN. Ils utilisent cette méthode pour entraîner un encodeur sur un grand jeu de données de séquences d'ADN, puis pour l'ajuster sur une tâche ultérieure, comme l'identification de l'occupation des motifs. Les résultats expérimentaux montrent alors que leur méthode est plus performante et plus interprétable que les méthodes existantes sur plusieurs ensembles de données de référence.

Les chercheurs de cette étude ont utilisé des séquences d'ADN de la base de données ENCODE (Encyclopedia of DNA Elements (1)), obtenues à partir d'expériences ChiP-seq sur 422 facteurs de transcription (TF) (2). Ils ont utilisé 40 de ces ensembles de données pour l'entraînement et l'évaluation de leur modèle, qui utilise l'apprentissage contrastif et la similarité d'édition des séquences pour pré-entraîner un codeur de séquence permettant d'identifier la liaison des TF aux motifs d'ADN. Les séquences d'ADN de l'ensemble de données ont toutes la même longueur et sont constituées de 101 nucléotides. Les séquences sont des sous-séquences centrales d'instances de motifs d'ADN, et l'étiquette d'une séquence d'ADN est déterminée par le fait qu'elle se lie à une TF. Les nucléotides des séquences d'ADN sont codés sous la forme d'un « one-hot vector » où chaque nucléotide est représenté par un vecteur à quatre dimensions. Les séquences d'ADN d'entrée sont alors représentées comme des tenseurs Lx4, où L est la longueur de la séquence.

Dans cette étude, les chercheurs ont développé deux modèles d'apprentissage contrastif pour la représentation (CLR) : editCLR (auto-supervisé) et supCLR (supervisé), qui ont la même structure mais diffèrent dans leur stratégie d'étiquetage des échantillons contrastifs et dans leur définition de la fonction de perte. Les deux modèles se composent d'un encodeur CNN et d'un classificateur DNN, inspirés respectivement des architectures DeepSEA et DeepBind. L'encodeur CNN extrait les informations de sous-séquences de la séquence d'ADN d'entrée à l'aide de noyaux de taille fixe et d'une couche de mise en commun, tandis que le classificateur DNN se compose de deux couches entièrement connectées et d'une couche de suppression. Ces modèles sont conçus pour pré-entraîner le codeur CNN en utilisant l'apprentissage contrastif, ce qui lui permet d'apprendre les règles associées à l'identification de la liaison TF et aux motifs d'ADN et d'aider le classificateur dans l'inférence finale. Les auteurs ont également introduit une méthode de calcul de la similarité des séquences basée sur l'algorithme de Needleman-Wunsch et la distance d'édition, qu'ils utilisent pour étiqueter les échantillons contrastifs dans le modèle auto-supervisé editCLR. La similarité entre deux séquences d'ADN est définie comme leur score de Needleman-Wunsch divisé par la longueur des séquences.

Dans cet article, il est aussi question du concept d'augmentation des données : une méthode utilisée dans l'apprentissage contrastif auto-supervisé pour générer des échantillons positifs et négatifs. Ici, l'augmentation des données est effectuée sur des séquences d'ADN par l'algorithme de Needleman-Wunsch. Ces opérations sont utilisées pour créer des séquences augmentées qui sont similaires à la séquence originale, avec une similarité d'au moins 90%. Pour un mini-lot de données de taille n, ce processus génère 2n séquences augmentées. Ces séquences sont ensuite utilisées pour créer des échantillons positifs et négatifs pour l'apprentissage contrastif.

EditCLR est une méthode qui utilise la similarité entre des paires de séquences d'ADN pour déterminer si elles doivent être considérées comme "positives" ou "négatives" dans un processus d'apprentissage. Pour ce faire, EditCLR compare la similarité entre les paires de séquences à des seuils prédéfinis de similarité positive et négative. Lorsque la similarité entre les séquences est supérieure au seuil de similarité positive, elles sont considérées comme positives, et lorsqu'elle est inférieure au seuil

de similarité négative, elles sont considérées comme négatives. Cette méthode détermine ces seuils en analysant la similarité entre toutes les paires de séquences possibles dans un ensemble de données, et en déterminant la proportion de paires positives pour différents intervalles de similarité. Selon les résultats de cette analyse, le seuil de similarité négative est fixé à 0,7 et le seuil de similarité positive à 0,8. La fonction de perte contrastive auto-supervisée est ensuite utilisée pour entraîner le modèle à regrouper les paires similaires et à séparer les paires dissemblables dans un espace de représentation. Dans supCLR, les paires de séquences d'ADN sont étiquetées comme positives ou négatives sur la base de leurs étiquettes de classe plutôt que de leur similarité. La fonction de perte contrastive supervisée est utilisée pour entraîner le modèle à regrouper les paires ayant la même étiquette de classe et à séparer les paires ayant des étiquettes de classe différentes.

Pour ce qui est des résultats, ils montrent que les deux modèles améliorent les performances de classification par rapport à l'utilisation des séquences d'ADN brutes ou d'un CNN. Les modèles sont évalués en utilisant des métriques comme l'exactitude, la précision, le rappel, le score F1 et l'AUC sur 40 ensembles de données. Les résultats indiquent que les modèles editCLR et supCLR sont plus performants que le modèle de base dans toutes les mesures, le modèle supCLR obtenant les meilleurs résultats. Le modèle editCLR avec un seuil positif de 0,8 est également plus performant que le modèle editCLR avec un seuil positif de 0,7 et le modèle SimCLR. Dans l'ensemble, l'utilisation de l'apprentissage contrastif dans le pré-entraînement des codeurs de séquences permet d'améliorer les performances de la tâche de classification en aval. Les chercheurs constatent également que leurs méthodes sont plus robustes face à des échantillons de petite taille et qu'elles fonctionnent bien avec des ensembles d'entraînement petits et grands.

3 Méthodologie

Nous construirons notre ensemble de données d'entraînement et de test à partir de databases de motifs déjà existantes, et de séquences dont on sait qu'elles ne sont pas des motifs. L'enjeu sera ensuite de comparer les options de classification binaire existantes. Cela nous permettra de choisir la méthode de classification la plus pertinente pour faire la distinction entre un motif et quelque chose qui n'en est pas un. Les résultats finaux consisteront à tester notre classificateur sur un ensemble de séquences test.

3.1 Elaboration de l'ensemble d'entraînement

Cette première étape est la plus cruciale dans notre projet. En effet, c'est l'ensemble d'entraînement qui va conditionner l'apprentissage et donc nos résultats de prédiction. Pour rappel, comme nous chercher à effectuer une classification binaire (motifs (M) ou non motifs (NM)), nous devons avoir dans notre ensemble d'entraînement des données/séquences qui couvrent les deux classes possibles. Ainsi, il nous faut des séquences qui sont des motifs et des séquences qui n'en sont pas.

Pour intégrer des séquences motifs à l'ensemble d'entraînement, nous avons utilisé une base de données de motifs de régulation des facteurs de transcription du génome humain (`http://compbio.mit.edu/encode-motifs/`). Cela nous a permis d'avoir une liste d'environ un millier de séquences, ou plutôt, de motifs. Cependant, ces séquences issues d'un fichier au format fasta (3), avaient la particularité de contenir des caractères spéciaux, qui ne sont pas des nucléotides : par exemple, W, B, X,... Ces caractères permettent en fait de décrire le fait qu'on a plusieurs possibilités pour cette position

(par exemple un W veut dire qu'on peut avoir un A, un T ou un U). De plus, dans le fichier, fasta, les probabilités de chaque nucléotide sont données. Ainsi, à partir de ces séquences motifs, nous allons générer toutes les séquences possibles en remplaçant les caractères "spéciaux" par ce qu'il représente. Cependant, les nucléotides candidats à la conservation seront ceux dont la probabilité d'apparition est supérieure à 25%. Une séquence pouvant posséder plusieurs caractères spéciaux, le nombre de nouvelles séquences motifs que nous créons grandit très vite et nous générons au final plusieurs millions de motifs.

Pour intégrer des non-motifs, c'est plus délicat, en effet, on retrouve à cette étape tout l'enjeu du sujet : qu'est-ce qui n'est pas un motif ? Pour répondre à cette problématique, nous avons fait le choix d'une séquence d'ADN humain (`http://useast.ensembl.org/info/data/ftp/index.html/`) parmi laquelle nous allons extraire des sous-séquences de toutes tailles (entre la taille min et max des tailles des motifs). Ces sous séquences seront comparées une à une aux motifs de même taille grâce à la distance d'Hamming (fonction *random_selection* de notre code). Si dans notre liste de motifs, il n'existe aucun motif qui est à distance d'Hamming de moins de $0.2 \times longueur\_seq$ de la séquence qu'on teste, alors cette séquence est considérée comme non-motif (fonction *est_motif* de notre code). En effet, cette séquence est considérée comme suffisamment distante de tous nos motifs, on la considère alors comme un non-motif. On va sélectionner de cette manière autant de non-motifs qu'on le souhaite (nous en prenons 10000 pour notre ensemble d'entraînement, ainsi que 10000 motifs).
Cette manière de faire semble pertinente car nous comparons des fragments d'ADN humain (qui contient donc forcément des motifs car cet ADN a un rôle biologique) à des motifs. Certains fragment vont donc être reconnus comme des motifs, mais si il y en a un qui n'est pas reconnu comme proche d'un motif, on peut naturellement supposer qu'on a trouvé une sous-séquence qui n'a pas de rôle biologique et que ce n'est donc pas un motif.

Grâce à ces étapes, nous avons en notre possession des séquences des deux classes en grand nombre. Nos modèle ne pouvant pas apprendre sur des string (les séquences sont des en effet des chaînes de caractères), nous procédons à une transformation de chaque séquence : A devient 1, C devient 2, G devient 3 et T devient 4. Ensuite, nous plaçons les données dans un fichier csv à deux colonnes : 'sequence' et 'etiquette'. La première colonne contient toutes nos séquences (qui sont désormais des suites de chiffres) et la deuxième associe à la séquence une étiquette de classe, 0 pour un motif, 1 pour un non-motif.
Une pratique courante en apprentissage automatique est de normaliser les données, cependant, nous n'avons pas jugé nécessaire de le faire ici. En effet, bien que nos séquences soient devenues des suites de chiffres, et donc vues comme des nombres par nos modèles, normaliser n'aurait aucun sens car c'est bien la suite de chiffre qui donne à chaque séquence sa particularité. Obtenir un nouveau nombre (entre -1 et 1 par exemple) aurait alors pour effet de détruire l'information que contenait la suite de chiffre.

Pour aller plus loin Ayant manqué de temps pour le faire, voici trois approches que nous aurions voulu mettre en place pour potentiellement améliorer notre ensemble d'entraînement et obtenir de meilleurs résultats dans nos prédictions.

- Le premier point aurait été d'utiliser des one-hot vector au lieu de suite de chiffres pour modéliser les séquences de notre ensemble d'entraînement comme dans l'article présenté en revue de littérature (4). En effet, cette pratique aurait peut-être permis à nos modèles de mieux apprendre,

et d'acquérir plus d'informations et de mieux faire la distinctions entre les éléments de chaque classe. Cependant, il aurait fallu adapter nos modèles pour qu'ils apprennent via des one-hot vector, ce que nous avons pas eu le temps d'implémenter. Pour rappel, dans cette modélisation, une position d'une séquence est vu comme un tuple de 4 éléments, rempli de 0 sauf une position contient un 1 : celle correspondant au nucléotide effectivement présent. Ainsi, avec comme ordre A,C,G,T dans les tuples, on aurait pour la séquence ATGC la représentation suivante : $((1,0,0,0),(0,0,0,1),(0,0,1,0),(0,1,0,0))$.

· Un autre point que nous aurions voulu mettre en place aurait été d'utiliser une autre distance que celle de Hamming pour comparer les séquences aux motifs. En effet, nous avions pensé à utiliser par exemple la distance de Levenshtein ou la distance d'édition.

3.2 Les classificateurs

Dans cette section, nous présentons les différents modèles de classification que nous avons décidé d'utiliser ainsi qu'une esquisse de la manière dont nous les avons implémentés. En effet, après avoir construit l'ensemble de nos données d'entraînement comme énoncé précédemment, il faut utiliser ces données pour que les modèles apprennent à différencier les deux classes (motifs/non-motifs).
Pour pouvoir comparer plusieurs résultats de prédiction, nous avons mis en place 3 modèles : les SVM (Support Vector Machine), la régression logistique et les réseaux de neurones multicouches. Nous avons pour chacun d'eux, tenté d'entraîner un modèle sur l'ensemble de données d'entraînement que nous avons créé. Nous avons également cherché à évaluer nos modèles en calculant leur "accuracy", qui indique à quel point les prédictions faites sont bonnes ; ce sera l'objet de la section "Résultats".

3.2.1 SVM

Les machines à vecteur de support (SVM) sont des algorithmes d'apprentissage supervisé utilisés dans le domaine de la classification. Ils sont largement utilisés dans la reconnaissance d'images, la reconnaissance de la parole et la classification de textes. Les SVM fonctionnent en trouvant une séparation linéaire des données dans un espace à plusieurs dimensions, de sorte que les éléments appartenant à des classes différentes soient séparés par une marge la plus grande possible. Pour cela, les SVM utilisent une fonction dite noyau qui permet de transformer les données d'entrée en un espace à plus grande dimension dans lequel une séparation linéaire est possible. Un des avantages des SVM est qu'ils peuvent être utilisés pour résoudre des problèmes complexes grâce à l'utilisation de noyaux. De plus, ils sont généralement efficaces pour traiter des jeux de données de grande taille. En revanche, ils peuvent être lents à entraîner pour des jeux de données très volumineux.

IMPLEMENTATION Pour l'implémentation, nous avons utilisé la fonction *SVC* de sklearn (5) qui implémente un classificateur à vecteurs de support (SVC). Nous cherchons à optimiser le paramètre 'C' qui correspond à la "force" que nous allons donner à la régularisation du modèle. Cette force est inversement proportionnelle à C et elle permet de réduire le sur-apprentissage qui rendrait notre modèle bon uniquement sur les données d'entraînement et donc mauvais pour les prédictions : à éviter absolument. Il va donc falloir trouver le bon compromis pour la valeur de C, c'est le principe de notre recherche d'hyper-paramètre optimal. De plus, pour la fonction noyau, c'est celle par défaut qui sera utilisée, soit 'rbf' (noyau gaussien).

3.2.2 Régression logistique

Le classificateur Logistic Regression est un algorithme de machine learning utilisé pour résoudre des problèmes de classification. Il est largement utilisé dans de nombreux domaines, notamment la reconnaissance d'images, la reconnaissance d'écriture manuscrite et la classification de documents.
Le principe de Logistic Regression est de modéliser la probabilité d'appartenance d'un exemple à l'une des classes en utilisant une fonction logistique. La fonction logistique est une fonction mathématique qui transforme une valeur en une autre valeur comprise entre 0 et 1, ce qui correspond à la probabilité d'appartenance à l'une des classes. Pour résoudre des problèmes de classification multi-classe, Logistic Regression utilise généralement une stratégie dite "One-vs-Rest" où un modèle de régression logistique est entraîné pour chaque classe par rapport à toutes les autres classes.
L'avantage de Logistic Regression est que le modèle est simple à mettre en œuvre et qu'il donne des résultats rapides, même avec de grandes quantités de données. Cependant, il est souvent moins précis que d'autres algorithmes, en particulier lorsque les données sont très corrélées ou non linéairement séparables.
IMPLEMENTATION Comme pour le SVM, nous choisissons de faire la recherche d'hyper-paramètre sur C qui est l'inverse de la force de régularisation. Là encore, des valeurs plus petites indiquent une régularisation plus forte. Nous faisons varier ce paramètre entre 0.001 et 50. Sa valeur par défaut est fixée à 1.

3.2.3 Réseaux de neurones multicouches

Les réseaux de neurones multicouches (ou réseaux de neurones en profondeur) sont un type de modèle de classification dans le domaine de l'apprentissage automatique. Ils sont inspirés du fonctionnement du cerveau humain et sont composés de plusieurs couches de "neurones" connectés entre eux. Chaque couche prend en entrée les sorties des neurones de la couche précédente, et produit des sorties qui sont utilisées comme entrées pour la couche suivante. Le nombre de couches et le nombre de neurones dans chaque couche peuvent être ajustés pour améliorer les performances du modèle (hyper-paramètres).
Le but d'un réseau de neurones multicouches est de prédire une cible (également appelée "étiquette") à partir d'un ensemble d'entrées. Pour cela, il utilise un processus d'apprentissage supervisé, où un ensemble de données d'entraînement est utilisé pour "enseigner" au modèle les relations entre les entrées et les étiquettes. Le modèle est ensuite capable de faire des prédictions sur de nouvelles données en utilisant les relations qu'il a apprises.
En général, les réseaux de neurones multicouches sont considérés comme des modèles très puissants et sont souvent utilisés pour résoudre des problèmes complexes de classification. Ils peuvent être entraînés pour traiter un large éventail de données, y compris des images, du texte, des séries temporelles et plus encore.

IMPLEMENTATION Pour mettre en place la méthode de prédiction par réseau de neurones multicouches, nous avons utilisé la fonction *MLPClassifier* de sklearn (Multi-Layer Perceptron). L'hyper-paramètre que nous cherchons à optimiser est le nombre de couches cachées *hidden_layer_sizes* que l'on accorde au modèle pour apprendre sur les données afin de prédire les nouvelles données. De même que précédemment, l'entraînement du modèle sur les données va permettre de trouver le nombre optimal de couches de neurones à insérer dans le réseau. Il faudra trouver un compromis, car un nombre de couches trop elevé mène très vite à du sur-apprentissage.

3.3 Le code

La partie implémentation de ce projet a été codée en python et nous avons eu recours au gestionnaire de version de code GitHub. On peut retrouver notre dépôt à l'adresse suivante `https://github.com/jurczakB/bio-info-2022`.
Comme le sujet du projet demande de créer un classificateur binaire, nous avons pour cela utilisé trois méthodes qui ont des comportements communs, comme le fait de devoir être entraînées, de devoir faire des prédictions, ou encore d'avoir un certain nombre d'hyper-paramètres à optimiser. Afin de pouvoir gérer au mieux ces trois types de classificateurs, nous avons décidé d'implémenter un patron de conception de type contrôleur.

3.3.1 Patron de conception contrôleur

Ce patron de conception définit donc des contrôleurs par le biais de classes, afin d'être responsable et de gérer les classes créant les instances des méthodes de classification. En outre, chaque classe de méthode est associé à un contrôleur. La classe méthode est responsable d'instancier le classifieur à partir de la liste des hyper-paramètres du classifieur visé. Elle définit également les fonctions d'entraînement, de prédiction et d'évaluation de la méthode. De plus, nous utilisons également un autre de type de classe permettant d'effectuer la visualisation des performances des méthodes. De surcroît, l'équipe utilise une classe de gestion de donnée permettant d'effectuer tous les pré-traitements sur les données nécessaires pour leur utilisation par les autres classes. Enfin le fichier permettant d'exécuter le code est le fichier nommé *main.py*.

4 Résultats

4.1 Démarche pour chaque méthode

La démarche entreprise pour chaque méthode est identique. Nous procédons en deux temps : avec et sans la recherche d'hyper-paramètres. A chaque fois, nous commençons par entraîner notre modèle puis nous obtenons son accuracy, c'est-à-dire son exactitude, à l'aide de la validation croisée. Le principe de la validation croisée en apprentissage automatique consiste à diviser les données d'entraînement en plusieurs parties, puis à entraîner le modèle en utilisant une partie des données et à le tester sur les autres. Cela permet de mesurer de manière plus précise la performance du modèle, car il est testé sur des données qu'il n'a jamais vues pendant l'entraînement. La validation croisée est utilisée à cette étape pour obtenir une vision fidèle de la performance de notre modèle en donnant une moyenne d'accuracy sur les k-folds, mais aussi une mesure de la variance du modèle.

Nous visualisons ensuite la courbe d'apprentissage du modèle qui nous permet d'évaluer l'accuracy du modèle en fonction de l'ajout de données d'entraînement lors de l'entraînement du modèle. Nous traçons de mêmela performance du modèle, c'est-à-dire l'accuracy en fonction du temps en seconde. Ces deux graphes permettent d'apprécier la façon dont chaque modèle apprend. Enfin, nous dressons un tableau qui récapitule les différents métriques du modèle obtenus selon les différents tests.
Concernant les métriques d'évaluation utilisés, nous avons chnoisis l'accuracy et la logloss. La logloss, ou perte logarithmique, est définie comme la moyenne négative des probabilités logarithmiques des résultats du modèle. Plus le modèle est proche de zéro, meilleur est l'ajustement. L'accuracy, ou exactitude, est la proportion de données à avoir bien été classés. Plus il est proche de 100, meilleure est l'accuracy du modèle car 100% des données ont donc été bien classées. Pour obtenir ce score, nous devons avoir les classes cibles, ce qui est le cas avec notre ensemble de données. Pour obtenir

des modèles satisfaisants, nous estimons qu'ils soivent avoir une 'accuracy' d'au moins 50%. En effet, si ce n'était pas le cas, un processus de calssification aléatoire (lancer une pièce de monnaie) serait alors plus performant que notre modèle !

4.2 SVM

Sans tuning La méthode des SVM, en ne s'intéressant pas à l'optimisation des hyper-paramètres, nous donne des résultats convenables. On fixe le paramètre C à 1 qui est sa valeur par défaut. Stable avec une très faible variance de 1%, ce modèle atteint une accuracy de 66% sur les données de test. Comme le montre la figure 1, le score d'accuracy sur les données d'entraînements stagne à partir de 4000 données, ce n'est donc pas un problème de quantités de données mais bien du modèle même des SVM qui n'arrive pas à mieux classer ces données. Il est aussi très long avec une durée de 12 secondes pour s'entraîner sur notre dataset composé de quelques milliers de données.

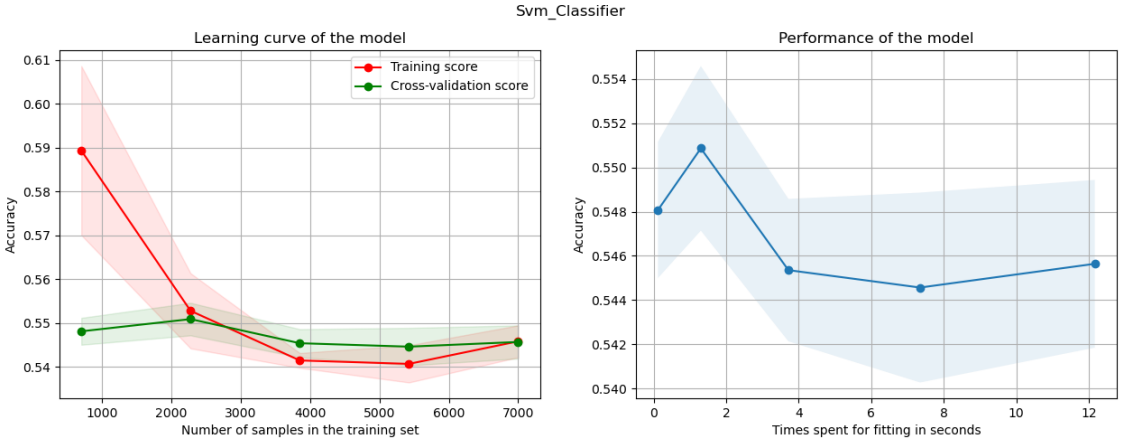

FIGURE 1 – Courbe d'apprentissage de l'algorithme SVM sans tuning

Avec tuning La figure 2 montre l'accuracy du modèle en fonction de la valeur de C et on remarque que l'optimal est atteint à partir de $C = 5$, le paramètre 1 fixé plus haut était donc trop faible. Le paramètre C optimal est égal à $\mathbf{C = 372}$.

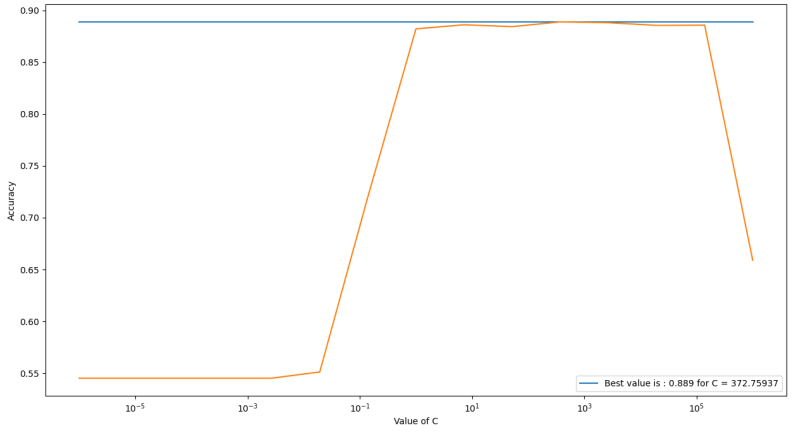

FIGURE 2 – Accuracy du modèle en fonction de l'hyper-paramètre

Nous avons donc des nouvelles courbes d'apprentissage en figure 3 avec cette nouvelle valeur de C optimale. Nous observons que le modèle obtient de bien meilleurs résultats avec une accuracy de 89% sur les données de test. La courbe d'apprentissage nous indique que notre modèle, à partir de 4000 données, sait reconnaître les motifs des non-motifs.

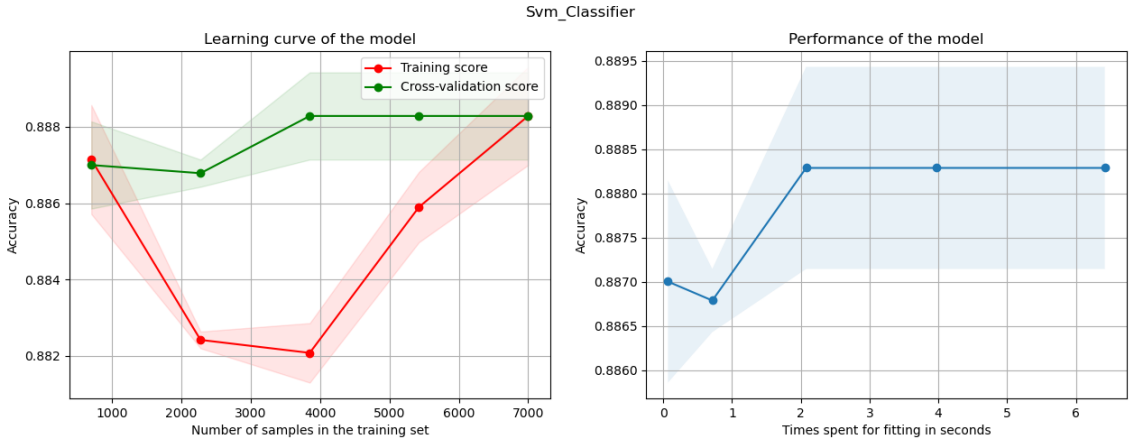

FIGURE 3 – Courbe d'apprentissage de l'algorithme SVM sans tuning

La table 1 recense les valeurs recueillies d' accuracy et de LogLoss dans le cas de l'utilisation de la méthode SVC pour la classification des motifs.

	TrainSet		TestSet	
	Sans tuning	Avec tuning	Sans tuning	Avec tuning
Accuracy	0.5385 (var : 0.01)	0.889 (var : 0.008)	0.66	0.8915
LogLoss	0.628	0.373	0.63	0.46

TABLE 1 – Métriques de performance pour le modèle SVM

En conclusion, cette méthode est un bon classificateur binaire à condition de procéder en amont à une otpimisation des hyper-paramètres. Il y a en effet une différence de 23 points pour un paramètre C bien choisi. Ce classificateur arrive donc à classer correctement 89% des motifs/non-motifs, avec une perte de 0.46 ce qui est très faible. Malgré ces bons résultats, ce modèle est très long à entraîner et on peut ainsi se demander s'il est utilisable en conditions réelles avec des dataset beaucoup plus grands. On s'intéresse alors à la régression logistique qui a l'avantage d'être très rapide.

4.3 Logistic Regression

Sans tuning Les résultats obtenus sur l'ensemble d'entraînement, sans tuning, sont les suivants :

— Accuracy : 0.54 (var : 0.01)

— Logloss : 0.68

Ces métriques sont assez faibles. Le modèle n'est supérieur que de 4 points par rapport au hasard. Son seul avantage est qu'il est très rapide car il classe les données de test en moins de 0.003 secondes comme on peut le voir sur la figure 4. On observe également sur cette figure que ce n'est pas un problème de quantités de données car à partir de 2000 données le modèle atteint son seuil de justesse.

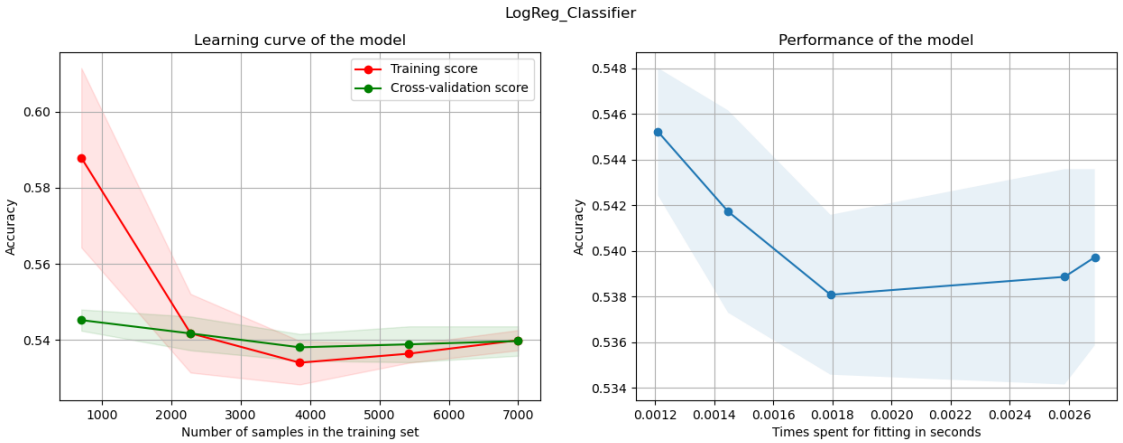

FIGURE 4 – Courbe d'apprentissage de l'algorithme Logistic Regression sans tuning

Avec tuning Après la recherche de l'hyper-paramètre C, nous obtenons la courbe présentée par la figure 5. Nous pouvons voir que l'accuracy du modèle ne change pas en fonction de l'hyper-paramètre. En effet, même s'il y a une chute, celle ci n'a qu'une amplitude de l'ordre de 10^{-5} sur l'accuracy du modèle. Le modèle obtient d'ailleurs exactement les mêmes résultats, ce qui le rend peu performant pour la classification binaire des séquences.

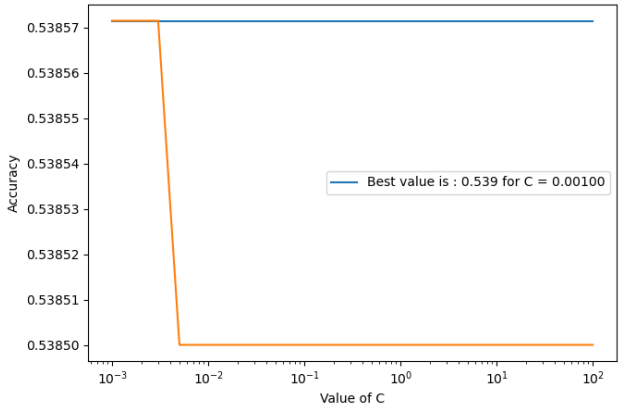

FIGURE 5 – Accuracy du modèle en fonction de l'hyper-paramètre C

Les résultats obtenus présentés par le tableau 2 montrent que Logistic Regression ne donne pas de bons résultats. Il est certes très rapide mais il n'arrive pas suffisamment à identifer les motifs des non-motifs.

	TrainSet		TestSet	
	Sans tuning	Avec tuning	Sans tuning	Avec tuning
Accuracy	0.54 (var : 0.009)	0.54 (var : 0.01)	0.541	0.54
LogLoss	0.68	0.69	0.68	0.69

TABLE 2 – Métriques de performance pour le modèle Logistic Regression

En conclusion, Logistic Regression est un algorithme simple et rapide qui n'est pas adapté à notre problème. Même avec une recherche d'hyper-paramètres, celui-ci n'est meilleur que de 4

points par rapport au hasard. Peut-être que ce modèle est trop simplise. Intéressons-nous alors à un classificateur plus complexe qui est le Neural Network Classifier.

4.4 Neural Network Classifier

Sans tuning Dans le cas sans tuning, le nombre de couches cachées *hidden_layer_sizes* est fixé à 100. Pour notre problème, le choix de cet hyper-paramètre permet d'avoir de très bons scores de classification pour l'apprentissage et pour la classification comme le montre la figure 6. On observe que le modèle n'a besoin que de 2000 données pour être capale de classer les motifs. Cependant, il est assez lent : il a besoin de 16 secondes pour s'entraîner sur notre dataset.

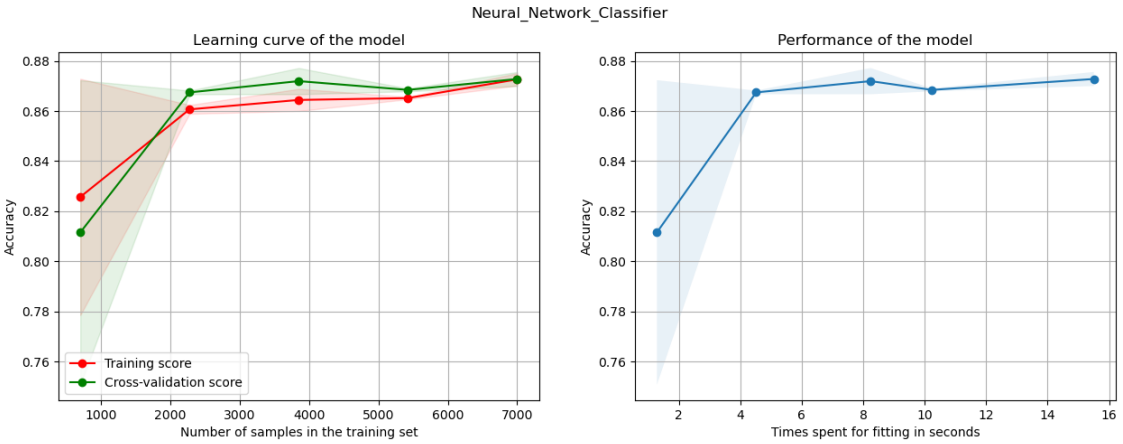

FIGURE 6 – Courbe d'apprentissage de l'algorithme NN sans tuning

Avec tuning La recherche du meilleur hyper-paramètre conduit au graphe de la figure 7. Nous voyons qu'un "pic" d'accuracy est présent pour un valeur de 300 couches cachées, avant que la courbe d'accuracy ne redescende légèrement.

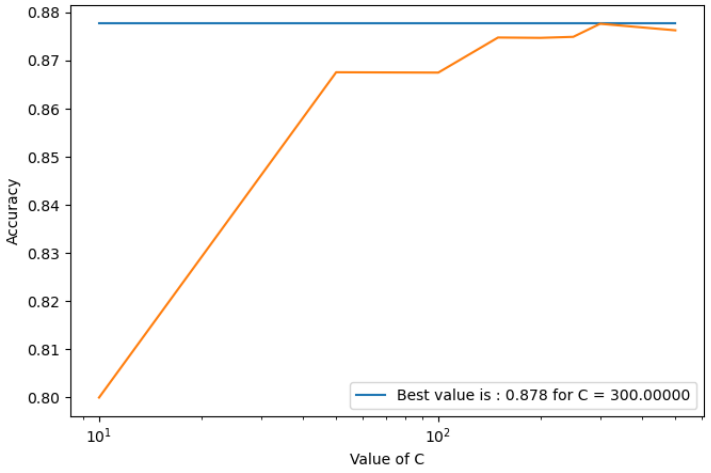

FIGURE 7 – Accuracy du modèle en fonction de l'hyper-paramètre

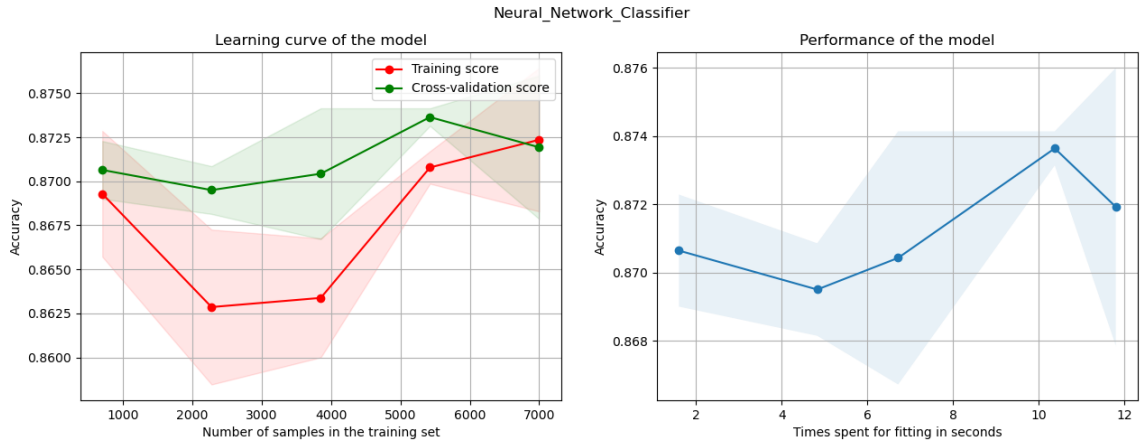

FIGURE 8 – Courbe d'apprentissage de l'algorithme NN avec tuning

En fixant à 300 le nombre de couches cachées dans le modèle, on arrive donc à gagner un peu d'accuracy dans nos prédictions. On a en effet une augmentation de 1% sur l'accuracy, même si la loss est identique. On observe aussi qu'il est plus rapide de 4 secondes.

	TrainSet		TestSet	
	Sans tuning	Avec tuning	Sans tuning	Avec tuning
Accuracy	0.878 (var : 0.007)	0.875 (var : 0.01)	0.87	0.88
LogLoss	0.34	0.33	0.4	0.4

TABLE 3 – Métriques de performance pour le modèle Neural Network

Comme on peut le voir dans le tableau **??**, le Neural Network Classifier est un classificateur qui obtient de très bons résultats pour classer nos séquences. Il est cependant assez lent, et comme les SVM on peut se demander s'il est utilisable en pratique.

5 Comparaison des classificateurs binaires

Dans ce projet, nous avons utilisé trois classificateurs différents pour rechercher des motifs dans le génome : le classificateur Logreg, le classificateur SVM et le classificateur Neural Network Classifier. Ces deux modèles se démarquent avec un score de justesse aux alentours de 88% et un temps d'entraînement équivalent.
L'étude de ces modèles a permis de souligner l'importance des hyper-paramètres. De bons hyper-paramètres ont permis au modèle SVM de passer de 66% à 89% d'accuracy et au modèle Neural Network Classifier d'être plus rapide. On peut voir la différence de scores avec et sans tuning avec la figure 9.

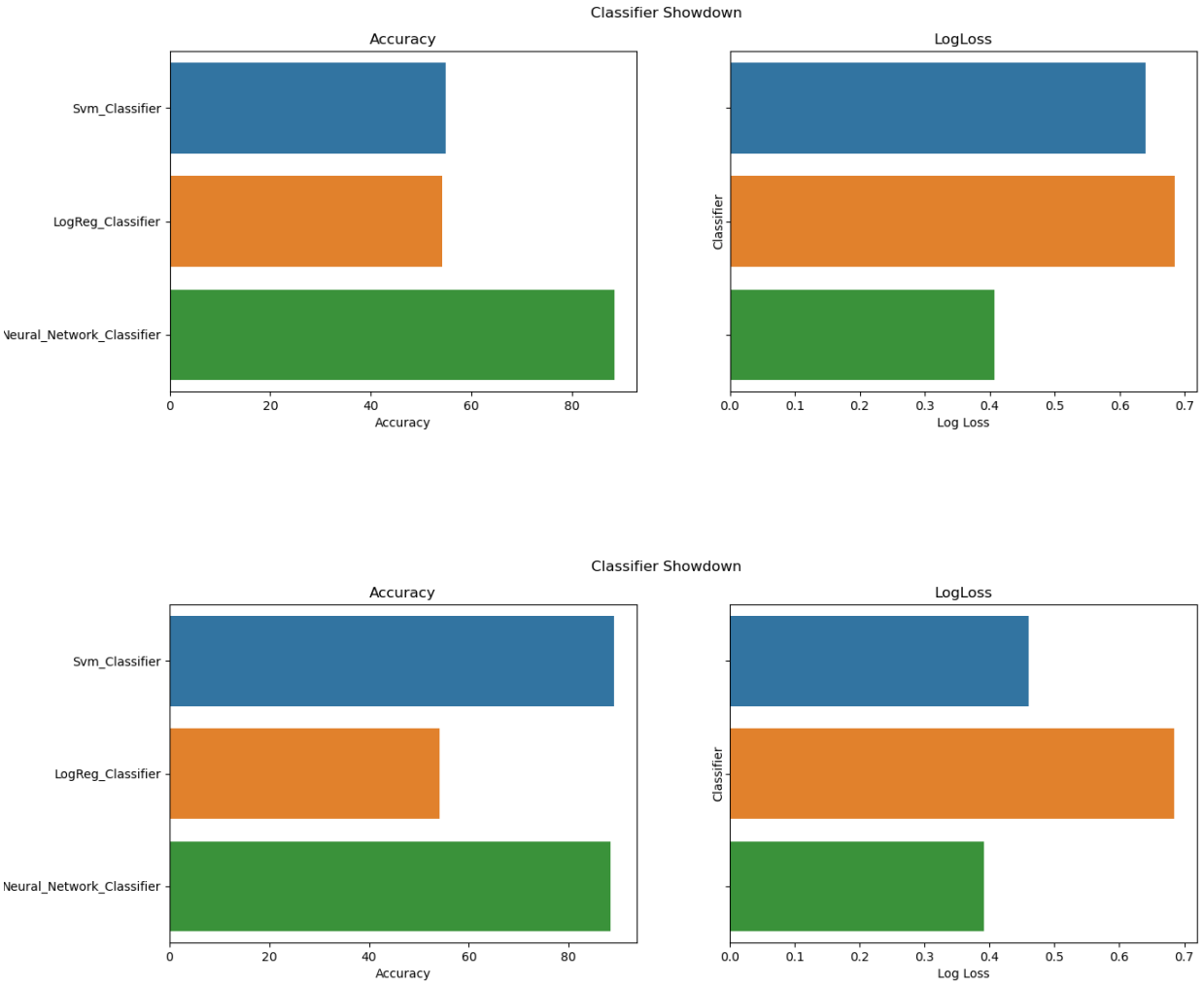

FIGURE 9 – Comparaison des performances des modèles avec et sans tuning

Finalement, le tableau 4 récapitule tous les résultats des prédictions de nos modèles. Les meilleurs classificateurs sont donc SVM et Neural Network Classifer avec des scores équivalents. On peut remarquer que c'est le NN Classifier qui se démarque avec une log plus faible.

	Accuracy		LogLoss	
	Sans tuning	Avec tuning	Sans tuning	Avec tuning
SVC	0.66	0.89	0.37	0.46
Logistic Regression	0.54	0.54	0.69	0.69
Neural Network Classifier	0.88	0.88	0.33	0.4

TABLE 4 – Récapitulatif de toutes les métriques des 3 modèles étudiés sur les données d'entraînement

6 Conclusion

Ainsi, ce projet avait pour but de trouver un moyen d'apprendre à un classificateur binaire à faire la distinction entre un motif et quelque chose qui n'en est pas un grâce à l'apprentissage machine. Lors de cette tâche de prédiction, l'un des plus grands enjeux aura été de créer l'ensemble d'entraînement. En effet, tout modèle d'apprentissage automatique nécessite d'être entraîné sur des données de toutes les classes considérées : ici des séquences motifs, mais aussi des séquences non-motifs. Bien que de nombreuses bases de données de motifs soient disponibles en ligne, c'est pour les non-motifs qu'il aura fallu être inventif afin de les générer et de les intégrer à l'ensemble d'entraînement.
Cette étape effectuée, nous avons ensuite pu tester 3 classificateurs différents et bien connus en apprentissage machine : les SVM, la régression logistique et les réseaux de neurone. Tous ont donné des résultats meilleurs qu'une décision aléatoire par 50/50, ce qui constitue des résultats encourageants et prouvent que les modèles peuvent différencier des motifs de non-motifs, dans une certaine limite.
Plusieurs axes d'améliorations sont à explorer pour obtenir des résultats encore meilleurs. Parmi eux, on peut citer l'utilisation de la distance de Levenshtein au lieu de Hamming pour générer des non-motifs, la modélisation des séquences par 'one-hot vector', ou finalement simplement d'agrandir encore plus l'ensemble d'entraînement pour couvrir un maximum de cas et que le modèle puisse mieux différencier les classes.

Références

[1] An integrated encyclopedia of dna elements in the human genome. [Online]. Available : https://www.nature.com/articles/nature11247

[2] Whole genome chromatin ip-sequencing (chip-seq) in skeletal muscle cells. [Online]. Available : https://europepmc.org/article/med/28842899

[3] Fasta (format de fichier. [Online]. Available : https://fr.wikipedia.org/wiki/FASTA_(format_de_fichier)

[4] A contrastive learning pre-training method for motif occupancy identification. [Online]. Available : https://www.mdpi.com/1422-0067/23/9/4699/htm#

[5] scikit-learn, machine learning in python. [Online]. Available : https://scikit-learn.org/stable/