



FACULTÉ DE SCIENCE

---

## IGL 691- Projet Informatique

---

<i>Auteur</i>	<i>Email</i>	<i>Id</i>
ABDALLAH AARABA	<a href="mailto:abdallah.aaraba@USherbrooke.ca">abdallah.aaraba@USherbrooke.ca</a>	19 143 080
NABIL BENJAA	<a href="mailto:nabil.benjaa@USherbrooke.ca">nabil.benjaa@USherbrooke.ca</a>	19 148 469

Présenté à :  
Aïda Ouangraoua

17 Avril 2020

## Table des matières

1	Introduction . . . . .	2
2	Contexte . . . . .	2
3	Revue de littérature . . . . .	3
4	Base de données . . . . .	3
5	Méthodes de classification . . . . .	3
5.1	Méthode1 : Classification basée sur les motifs . . . . .	3
5.1.1	Construction de la base des données . . . . .	3
5.1.2	Exploration de la base de données de tests . . . . .	4
5.1.3	Exploration de la base de données originale . . . . .	8
5.1.4	Pré-traitement de la base de données . . . . .	10
5.1.5	Application de plusieurs modèles de classifications . . . . .	13
5.1.6	Choix du modèle . . . . .	15
5.1.7	Évaluation des résultats . . . . .	21
5.1.8	Extraction des motifs . . . . .	22
5.1.9	Visualisation . . . . .	28
5.1.10	Application sur la base de données globale . . . . .	29
5.1.11	Continuité et évolution . . . . .	31
5.2	Méthode2 : Classification basée sur LSTM . . . . .	34
5.2.1	Un peu sur LSTM . . . . .	34
5.2.2	Concept de la méthode . . . . .	35
5.2.3	Modèles implémentés . . . . .	36
6	Conclusion . . . . .	42
7	Références . . . . .	42

## Table des figures

1	Une partie de la base de données de 20 familles . . . . .	4
2	Statistiques de la base de données . . . . .	4
3	ACP avec 10 composants . . . . .	5
4	ACP avec 10 composants . . . . .	5
5	ACP avec deux composantes pour 20 familles . . . . .	5
6	Différents score de test pour différents modèles . . . . .	6
7	Arbre de décision de notre base de données . . . . .	6
8	Une partie d'arbre de décision estimée . . . . .	7
9	Partie d'arbre sans application de PCA . . . . .	7
10	Base de données initiale . . . . .	8
11	Ratio des valeurs nulles dans notre dataset . . . . .	9
12	Statistiques des motifs (attribut classe a ignoré) . . . . .	9
13	Score du modèle . . . . .	10
14	Nombre d'occurrence total . . . . .	11
15	Nombre d'occurrence suivant chaque famille . . . . .	12
16	Calcule de occ =nombre d'occurrence total . . . . .	12
17	Résultat finale après les calculs . . . . .	12
18	Résultat après sélection . . . . .	13
19	Fréquence des motifs suivant chaque classe . . . . .	13
20	Cross validation . . . . .	14
21	Score obtenu pour chaque modèle . . . . .	14
22	Les différents résultats d'entraînement de test de différents k . . . . .	16
23	Les différents paramètres choisis . . . . .	17
24	Meilleure paramètres, score pour extra-tree . . . . .	18
25	Meilleure paramètres, score pour random-forest . . . . .	18
26	Recherche de grille pour random-forest . . . . .	18

27	Recherche de grille pour extra-tree . . . . .	19
28	Réultat des moyens score et leur écart-type . . . . .	19
29	Comparaison de la variance des scores obtenus par 5 cross validation . . . . .	19
30	Résultat d'entraînement et de test pour K-Neighbores . . . . .	20
31	Résultat d'entraînement et de test pour extra-tree . . . . .	20
32	Résultat d'entraînement et de test pour random-forest . . . . .	20
33	Résultat des scores de test et d'entraînement pour les différents modèles en utilisant les meilleurs paramètres . . . . .	21
34	Résultats des scores rappel, précision et f-mesure . . . . .	22
35	Scores pour les deux familles 292 et 289 . . . . .	22
36	L'importance de chaque motifs pour cette classification . . . . .	23
37	L'importance de chaque motifs . . . . .	23
38	Seuil d'importance . . . . .	24
39	Distribution normal des valeurs . . . . .	24
40	Boxplot des valeurs obtenues . . . . .	25
41	Longueur et importance des motifs . . . . .	25
42	Boxplot d'importance en fonction des longueurs des motifs . . . . .	26
43	Motifs importants . . . . .	26
44	Réultat ENTRAINEMENT / TEST . . . . .	26
45	Précision, rappel et F1-mesure . . . . .	27
46	Importance des motifs . . . . .	27
47	Boxplot d'importance en fonction des longueurs des motifs . . . . .	28
48	Nouvelle base de données . . . . .	28
49	classifieur de extra tree . . . . .	28
50	Une partie de la forêt d'arbre générée . . . . .	29
51	La forêt d'arbre totale . . . . .	29
52	Arbre de décision de cette classification . . . . .	29
53	Boxplot pour les scores de knn . . . . .	30
54	Les différents scores de knn . . . . .	30
55	Précision rappel et F1 mesure score . . . . .	31
56	résultats finaux . . . . .	31
57	Application de K-min pour la dataset de test . . . . .	32
58	Matrice de contingence . . . . .	32
59	Régions denses . . . . .	33
60	Régions denses 2 . . . . .	33
61	Détection des valeurs aberrantes . . . . .	34
62	Fonctionnement d'un LSTM . . . . .	35
63	Architectures Équivalentes LSTM . . . . .	35
64	architecture du modèle LSTM simple . . . . .	36
65	Paramétrage du modèle LSTM simple avec Keras . . . . .	37
66	Architecture du modèle LSTM avec bloc d'attention . . . . .	38
67	bloc attentionnel . . . . .	39
68	paramétrage du modèle de LSTM avec bloc d'attention . . . . .	40
69	Erreur d'entraînement et de test du modèle LSTM simple . . . . .	41
70	Erreur d'entraînement et de test du modèle LSTM avec bloc d'attention . . . . .	41

## 1 Introduction

L'Acide ribonucléique est un acide nucléique présent dans toutes les cellules vivantes. Son rôle principal est d'agir comme un messager portant les instructions de l'ADN pour contrôler la synthèse des protéines, bien que dans certains virus l'ARN plutôt que l'ADN porte l'information génétique. Les molécules d'ARN se replient en structures secondaires et tertiaires caractéristiques qui représentent diverses activités fonctionnelles. Beaucoup de ces structures d'ARN sont assemblées à partir d'une collection de motifs structuraux d'ARN. Ces blocs de construction de base sont utilisés à plusieurs reprises et dans diverses combinaisons pour former différents types d'ARN et définir leur structure unique. En effet Le domaine de la bio-informatique, a toujours été lié au développement des algorithmes qui utilisent l'informatique et les statistiques dans la gestion, le traitement, l'analyse et la modélisation des données afin de résoudre des problèmes en sciences de la vie tels que la biologie, l'agronomie et la médecine. L'analyse de séquences biologiques est l'un des sujets les plus intéressants de la bioinformatique. L'une des recherches importantes faites dans ce domaine est la classification des ARN suivant leurs structures secondaires qui sont représentée par des familles. Jusqu'à nos jours 3016 familles existent à savoir (ARNr, ARNm, ARNt, ARNSi, ARNmi, snARN...). Chacune de ses familles possède une structure ou une fonction particulière.

De nos jours des nouvelles méthodes ont été développées pour analyser ces séquences et classifier ces ARN suivant leurs structures secondaires. Parmi ces structures, on trouve la modélisation probabiliste (chaînes de Markov) méthodes issue de l'apprentissage automatique (machine Learning), de l'apprentissage profond (deep learning) et de l'intelligence artificielle. Ces méthodes ont été utilisées dans le but d'identifier et de classer les séquences. La recherche et le développement des techniques modernes sont devenues obligatoires afin de manipuler, traiter et interpréter un volume important de données génomiques de nature complexe. En effet les méthodes classiques requièrent plus de temps et plus de travail. Plusieurs méthodes de l'intelligence artificielle et de forage de données sont appliquées de plus en plus dans l'analyse de séquences biologiques en raison de leur rapidité, la possibilité de leur interprétation, leur adaptation ainsi que l'extension, et la disponibilité de leurs implémentations informatiques. En tant qui étudiant dans le domaine de l'intelligence artificielle et de la science de données ce sujet m'intéresse énormément.

Le présent projet s'intéresse à l'Évaluation de diverses représentations et méthodes de classification pour l'annotation de séquences d'Acide RiboNucléique (ARN) dans des génomes. La classification de ces séquences assigne une nouvelle séquence à un ensemble de séquences connues partageant des propriétés, des caractéristiques, des traits ou des fonctions similaires.

L'objectif de base de ce projet était l'implantation de modèles de classification des séquences ARN pour que par la suite appliquer l'un de ces modèles sur un génome et déterminer quelles sont les parties de ce génome qui sont des séquences ARN. Dans ce projet, on a pu atteindre la première partie de l'objectif de base qui est l'implémentation d'un ensemble de modèles qui peuvent être utilisée dans la classification des séquences. La deuxième partie de l'objectif n'est pas atteinte, faute de temps, et donc elle ferait l'objet de travaux futures. Les livrables de ce projet sont :

- Code source
- Jupyter notebooks
- Données transformées
- Rapport

Ces livrables seront trouvés dans le répertoire GitHub du projet.

## 2 Contexte

Comme il est présenté par le dogme central de la biologie, les acides ribonucléiques abrégés par ARN sont les agents de l'expression génétique nécessaire à la synthèse des protéines. Effectivement, par le biais des processus de transcription et traduction, l'informatique génétique se transforme de l'ADN à l'ARN en site de l'ARN aux protéines.

Les ARN ne jouent pas que le rôle d'intermédiaire messager de l'information génétique, mais aussi ils agissent en tant qu'enzyme nécessaire au maintien l'activité cellulaire.

Chaque ARN adopte une structure particulière qui lui permet d'exécuter sa fonction dans l'organisme cellulaire. De ce fait, étudier les structures d'ARN pour savoir auxquelles famille ils appartiennent est crucial pour comprendre leurs fonctions.

Dans notre projet, les ARN seront présentés par des chaînes de caractères représentant leur structure primaire. La base de données utilisée pour l'entraînement et le test de nos modèles est celle de Rfam [1]. Les données de cette base de données sont regroupées en 3016 familles. Ainsi, notre but est l'implémentation d'un modèle capable de prédire la famille d'une chaîne de caractères donnée en entrée à un modèle.

### 3 Revue de littérature

Nombreux sont les documents et les sites web sur lesquelles on s'est basé pour réaliser ce projet. Tout d'abord, on cite Le site web de la base de données Rfam qui fournit les données de l'entraînement et du test des modèles qu'on a implémenté. Parmi les méthodes de classification réalisées dans ce projet on trouve une méthode qui est complètement basée sur l'architecture LSTM réalisée par les auteurs Sepp Hochreiter et Jürgen Schmidhuber dans leur article "Long Short-Term Memory". En outre, les modèles réalisés dans ce projet sont implémentés à l'aide de la librairie Keras connue par sa simplicité pour l'utilisateur. Ian Goodfellow, Yoshua Bengio, et Aaron Courville ont pu écrire un livre très utile dans le domaine de l'apprentissage profond, et parmi les notions introduites dans ce livre on a le concept de l'entropie croisée qui est utilisée comme fonction de perte de la plupart de modèles de réseaux de neurones aujourd'hui. L'article "Brief Introduction to Attention Models" de Abhishek Singh publié dans le site web "Towards Data Science" explique clairement le mécanisme d'attention dans les réseaux de neurones. Dans le site de Medium on trouve un article intéressant qui explique le fonctionnement de l'algorithme k-nearest Neighbors par l'auteur Renu Khandelwal. On s'est basé sur l'article "Decision Trees Explained Easily" de Chirag Sehra qui explique comment les arbres de décisions fonctionnent. Dans le livre des auteurs Breiman et Leo intitulé "Statistical Modeling : The Two Cultures" on trouve une claire description du concept de bagging. Pour ce qui est du grid search, on a consulté l'article "Grid Search for model tuning" de Rohan Joseph dans le site web Towards Data Science

### 4 Base de données

### 5 Méthodes de classification

#### 5.1 Méthode1 : Classification basée sur les motifs

##### 5.1.1 Construction de la base des données

###### 1. Structure de données pour extraire les motifs communs :

Dans notre travail, nous utilisons un suffixe généralisé GST, qui est une structure de données arborescente de suffixes construite sur toutes les séquences d'entrée. Chaque nœud interne représente un motif commun pour son sous-arbre enraciné, ce qui signifie la sous-chaîne qui existe à différentes positions dans la séquence, d'où le nombre d'occurrences de ce motif. Avec GST, en plus du nombre d'occurrences dans la même séquence, chaque nœud interne représente un motif commun pour toutes les séquences de son sous-arbre.

###### 2. Matrice d'index des séquences et motifs communs :

Afin d'avoir rapidement accès à l'utilisation de ces motifs communs dans nos prochaines étapes, nous les indexons en fonction de leurs séquences. Pour cela, nous générerons une matrice de séquences index / motif commun, qui contient une liste de positions des motifs communs et le nombre de ses occurrences.

###### 3. Longueur des motifs communs :

Au début, nous générerons une matrice avec toute la longueur théorique possible du motif. Lorsque les motifs communs entre les séquences vont de 1 caractère à la longueur d'une séquence donnée, ce qui signifie que nous avons de nombreuses séquences identiques, ou qu'une séquence est une sous-chaîne d'une autre séquence, et ce n'est pas vrai. En conséquence, nous avons une énorme taille de matrice, seules quelques familles génèrent une énorme matrice. Ces matrices sont très clairsemées, car, les motifs longs n'existent généralement que dans sa séquence, de sorte que toutes les autres séquences en métrique ont la valeur 0. Pour cette raison, nous limitons la longueur des motifs acceptables entre MIN et MAX Limiter la longueur des motifs communs entre Min = 3 et Max de 5 à 30 nucléotides. Avoir un impact énorme sur la structure de données GST, car elle aura au niveau des mots max qui réduisent la mémoire utilisée. Cela peut avoir un impact énorme

sur la structure de données GST, car elle aura à des mots des niveaux max qui diminuent par rapport à l'espace mémoire utilisé.

### 5.1.2 Exploration de la base de données de tests

La base de données a été téléchargée depuis le site <https://rfam.xfam.org/> Contenant 3016 familles. Avant de générer le fichier CSV nous avons essayé de définir le motif (colonnes) de notre base de données en se basant sur un algorithme (.....). La génération de tous les motifs nous a retourné un nombre énorme de colonnes. Donc la taille devient beaucoup plus grandes, plus de 50Go pour toutes les familles. Alors en attendant la modification de cet algorithme, nous avons essayé de tester les modèles de classification pour une base de données de 20 familles. Exploration de la base de données de test :

index	seqIdInFam	familyId	AAAAAAAAAAAAAGAAA	AAAAAAAAAAAAAGAAA	AAAAAAAAAAAAAGAAA
0	0	0	0	0	0
1	1	0	0	0	0
2	2	0	0	0	0
3	3	0	0	0	0
4	4	0	0	0	0
...	...	...	...	...	...
1902	56	19	0	0	0

FIGURE 1 – Une partie de la base de données de 20 familles

```
RangeIndex: 1907 entries, 0 to 1906
Columns: 101117 entries, Classe to R
dtypes: int64(101117)
memory usage: 1.4 GB
```

FIGURE 2 – Statistiques de la base de données

Une base de données avec 1907 lignes et 101117 colonnes. Ce qui nous cause un vrai problème. La mémoire d'usage est d'environ 1.5Go pour seulement 20 familles ce qui explique la complexité de la base en termes de mémoire.

Pour appliquer les modèles de classification, le nombre de données doit être au moins significativement de la même grandeur qu'au nombre de colonnes, s'il n'est pas très grand.

Nous avons essayé d'appliquer une des méthodes de réduction de dimension "Analyse en composantes principales". Cette méthode nous permet de réduire les dimensions (colonnes) en un nombre de composantes déterminé à l'avance.

Cette approche est basée sur la projection des données sur les axes ayant la plus grande variance (qui contiennent le maximum d'informations). Ces axes correspondent aux vecteurs propres associés aux plus grandes valeurs propre de la matrice de covariance issues de la matrice des données.

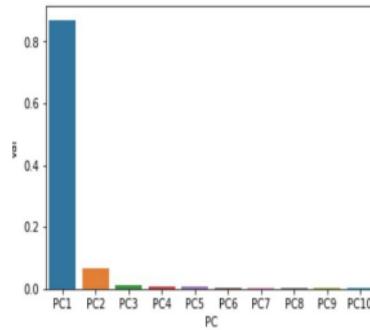


FIGURE 3 – ACP avec 10 composants

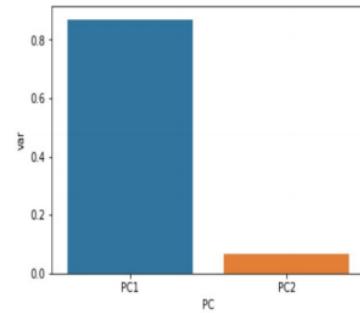


FIGURE 4 – ACP avec 10 composants

Avec 10 composantes, on constate que seule la première composante a presque 90% de quantité d'information gardé tandis que les autres composantes (de 3 à 10) ne représentent rien. Nous avons choisi donc de refaire ACP avec deux composantes cela nous aide aussi à visualiser les données et avoir une idée sur la distribution et la séparation de ces données.

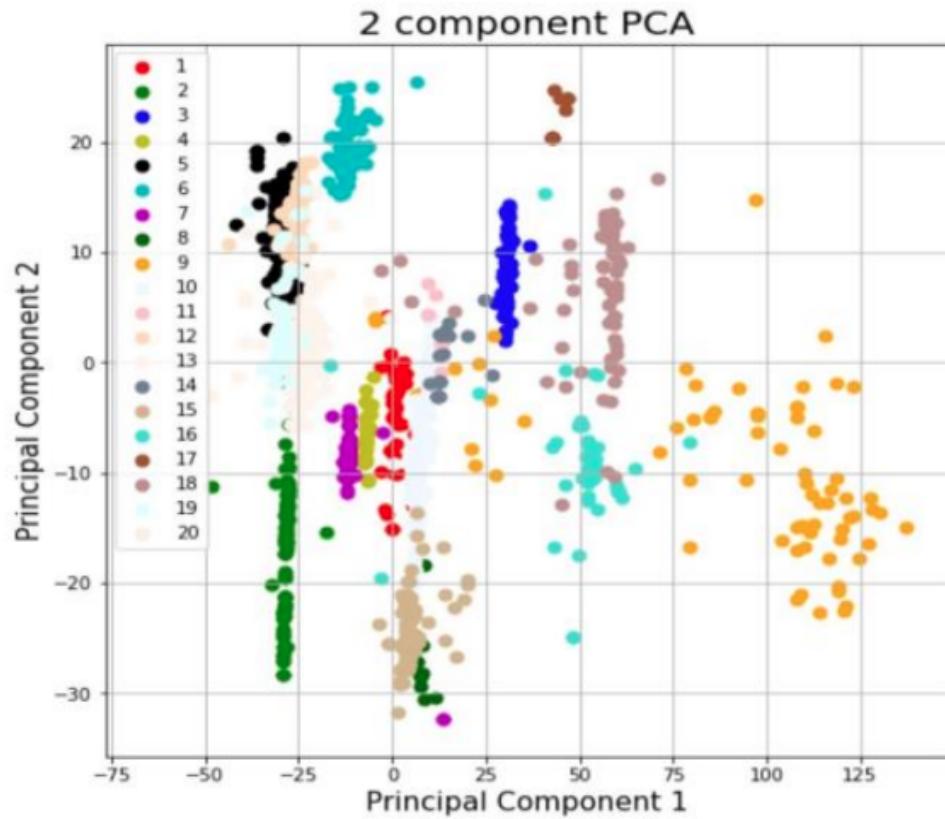


FIGURE 5 – ACP avec deux composantes pour 20 familles

Nous constatons, en appliquant cette transformation que les classes sont plus ou moins séparées les unes des autres suivant ces 2 composantes avec quelques chevauchements. Nous avons appliqué quelques modèles

pour tester si vraiment on arrive à avoir un bon score de classification. Nous avons ainsi trouvé de bons résultats pour différents modèles.

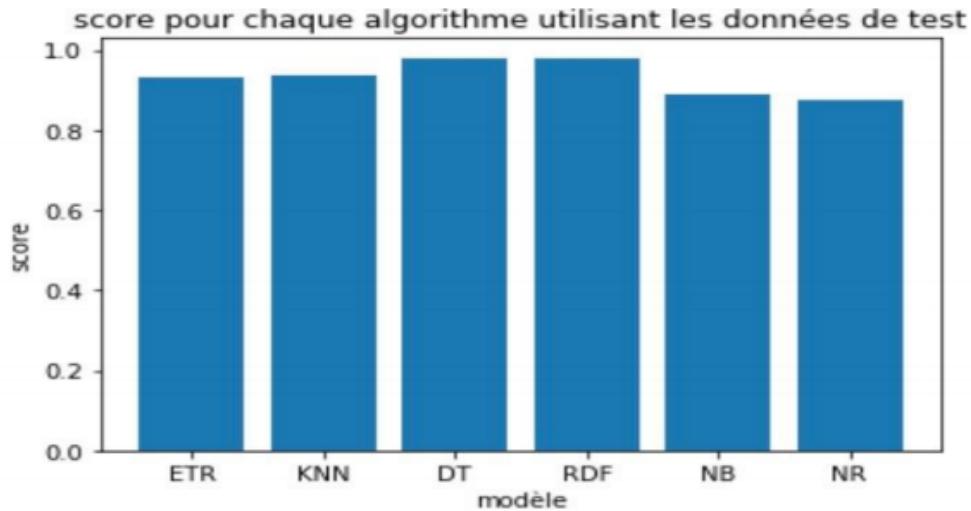


FIGURE 6 – Différents score de test pour différents modèles

On sait bien que l’arbre de décision gère bien les données catégoriques (surtout s’ils sont mieux séparés suivant leurs dimensions). Ceci explique les scores des deux modèles DT (arbre de décision) et RDF (forêt d’arbre de décision). Avoir un bon résultat avec tous les modèles se serait bien, mais est-ce-que cela est correct ? Tant que nous n’avons pas utilisé tous les données, on ne peut pas généraliser. D’autre part le problème des données Im balancées peut expliquer les résultats obtenues car peut être le test se fait sur une seule ou deux classes. Nous avons essayé de visualiser l’arbre de décision :

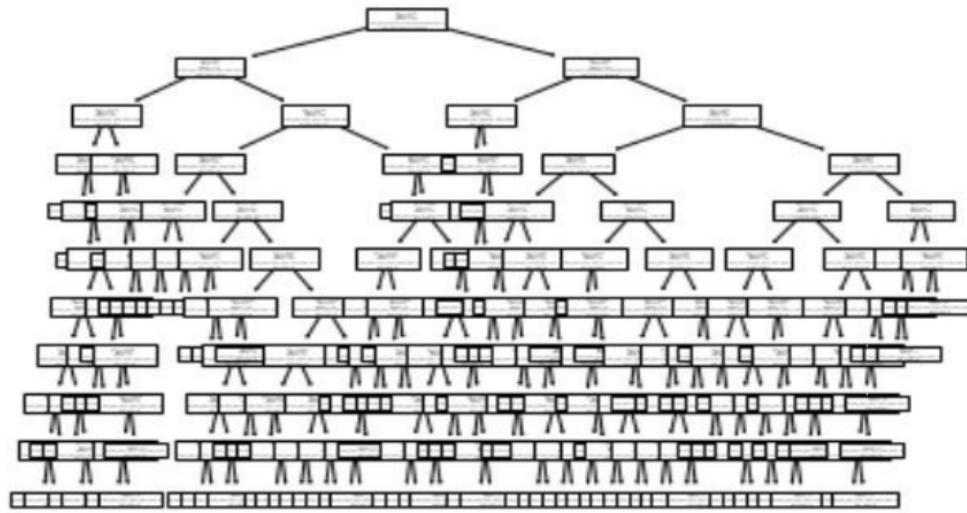


FIGURE 7 – Arbre de décision de notre base de données

Le résultat est un peu difficile à visualiser. Donc nous avons utilisé les bibliothèques ipywidgets et IPython.display pour une meilleure visualisation :

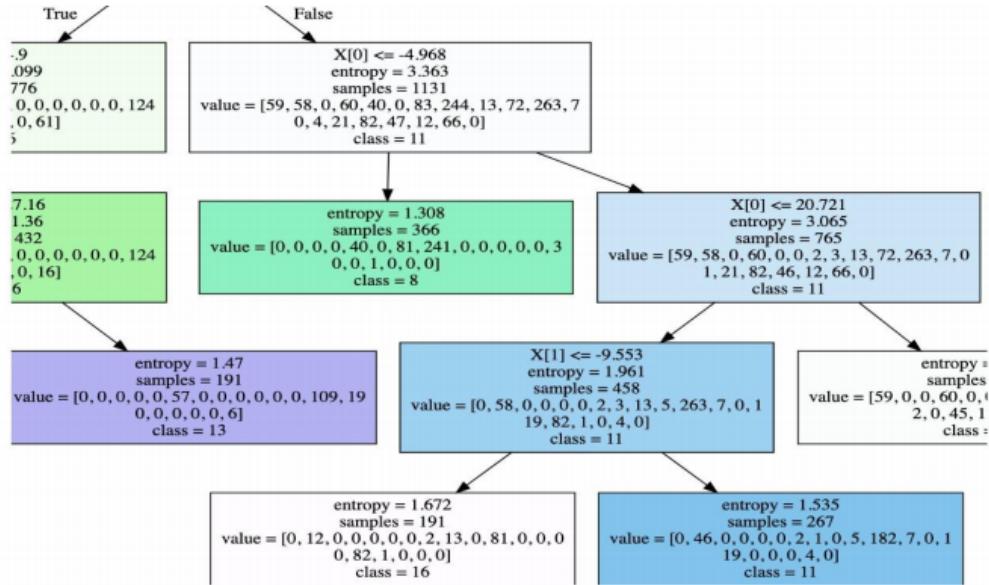


FIGURE 8 – Une partie d'arbre de décision estimée

Même si on a eu un bon résultat, lors de notre visualisation, nous constatons que l'utilisation de PCA supprime l'information des motifs. Alors que ces derniers sont obligatoires pour atteindre l'un des buts de ce projet.

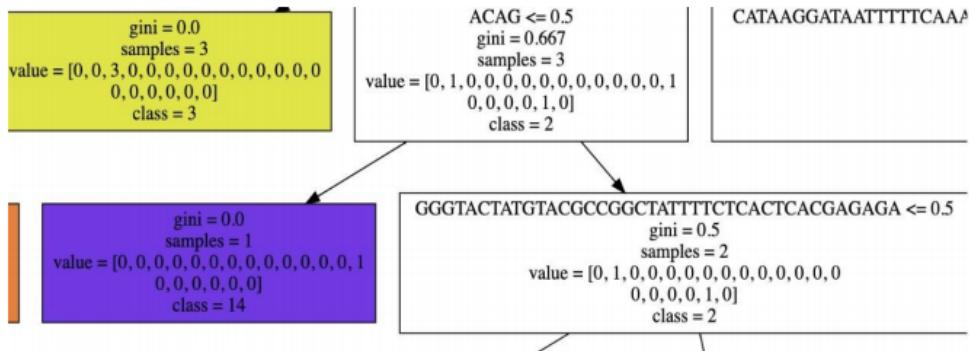


FIGURE 9 – Partie d'arbre sans application de PCA

On a choisi deux composantes, car utiliser plus n'ajoute pas grand-chose en termes d'informations (variance). D'après ce qui précède, on va suivre ce plan pour répondre au besoin de notre projet : D'une part à cause du volume de la base de données on a essayé de produire une base avec 300 familles en utilisant tous les motifs intéressants, pour notre étude. D'autre part on a générée une base avec toutes les familles mais en limitant les motifs a 20%, afin de tester nos résultats s'il est bon de généraliser a 3000 familles.

Notre étude va se baser par la suite sur les étapes suivantes :

1. Une exploration de la base de données
  2. Un pré-traitement de la base de données
    - (a) Une sélection d'attributs informatifs et représentatifs pour les classes
    - (b) Une gestion des données non balancées
    - (c) Une détection des incohérences
  3. Une application de plusieurs modèles de classifications

4. Un choix selon des caractéristiques des modèles les plus adaptés à notre base de données et aux résultats.
5. Une optimisation et un choix des paramètres pour les modèles choisis
6. Une analyse des résultats des modèles
7. Le dernier choix du bon modèle d'après ce qui précède.
8. Une Application de modèle en augmentant l'occurrence par un pré-traitement
9. Une Validation du modèle et un calcul des erreurs
10. La Généralisation sur une base de données de 3000 familles

### 5.1.3 Exploration de la base de données originale

index	seqIdInFam	familyId	AAAAAAAAAA	AAAAAAAAG	AAAAAAAC	AAAAAAA	AAAAAAAGA	AAAAAAAGG	...	CCCCCCCCG	CCCCCCCCC
0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0
2	2	2	0	0	0	0	0	0	0	0	0
3	3	3	0	0	0	0	0	0	0	0	0
4	4	4	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
7291	7291	22	300	0	0	0	0	0	0	0	0
7292	7292	23	300	0	0	0	0	0	0	0	0
7293	7293	24	300	0	0	0	0	0	0	0	0
7294	7294	25	300	0	0	0	0	0	0	0	0
7295	7295	26	300	0	0	0	0	0	0	0	0

7296 rows x 15192 columns

FIGURE 10 – Base de données initiale

Après une réduction du nombre de familles et un choix représentatif d'un nombre de séquences significatives, on a intégré la base de données. On a constaté alors quelques problèmes d'incohérences dans quelques colonnes (motif). On a trouvé des colonnes avec des valeurs nulles et des colonnes avec des noms incompréhensibles puisque le nombre de ces derniers est négligeable (environ 4 colonnes parmi 15192 motif). On a procédé à l'élimination de ces colonnes. Une colonne qui représente le numéro de séquence par famille et aussi supprimée puisqu'elle ne sert à rien.

La base de données contient 15190 motifs avec 300 familles et au total 7296 séquences. L'ordre de grandeur du nombre de motifs est supérieure au nombre de séquences, ce qui pose problème, donc on doit le gérer. Chaque élément ( $U_{ij}$ ) de cette matrice représente une fréquence du motif ( $e_i$ ) dans la séquence ( $s_j$ ). On vérifie les valeurs manquantes dans notre base de données

	Types	Nb manquants	Ratio manquants%
GCCGCCAGC	int64	0	0.0
GCCGCCAG	int64	0	0.0
GCCGCCCA	int64	0	0.0
GCCGCCGC	int64	0	0.0
GCCGCCCG	int64	0	0.0

FIGURE 11 – Ratio des valeurs nulles dans notre dataset

On visualise les statistiques descriptives :

	Classe	AAAAAAAAAA	AAAAAAA
<b>count</b>	7296.000000	7296.000000	7296.000000
<b>mean</b>	149.585389	0.017407	0.03591
<b>std</b>	87.277774	0.467562	0.57341
<b>min</b>	0.000000	0.000000	0.00000
<b>25%</b>	74.000000	0.000000	0.00000
<b>50%</b>	149.000000	0.000000	0.00000
<b>75%</b>	225.000000	0.000000	0.00000
<b>max</b>	300.000000	34.000000	37.00000

FIGURE 12 – Statistiques des motifs (attribut classe a ignoré)

Ce qui nous intéresse ici, c'est la valeur maximale qui nous donne une vue sur les poids des motifs suivant les séquences, les moyennes et les variances

On constate bien l'existence que quelques motifs qui ont des poids intéressants suivant quelques séquences. Dans la partie du test on a bel et bien montré l'importance de l'arbre de décision et la foret d'arbre de décision. Mais cette fois ci on a appliqué PCA sur la base de données de 300 familles et on a constaté que le score a chuté vers le 30%.

Deux explications sont possibles :

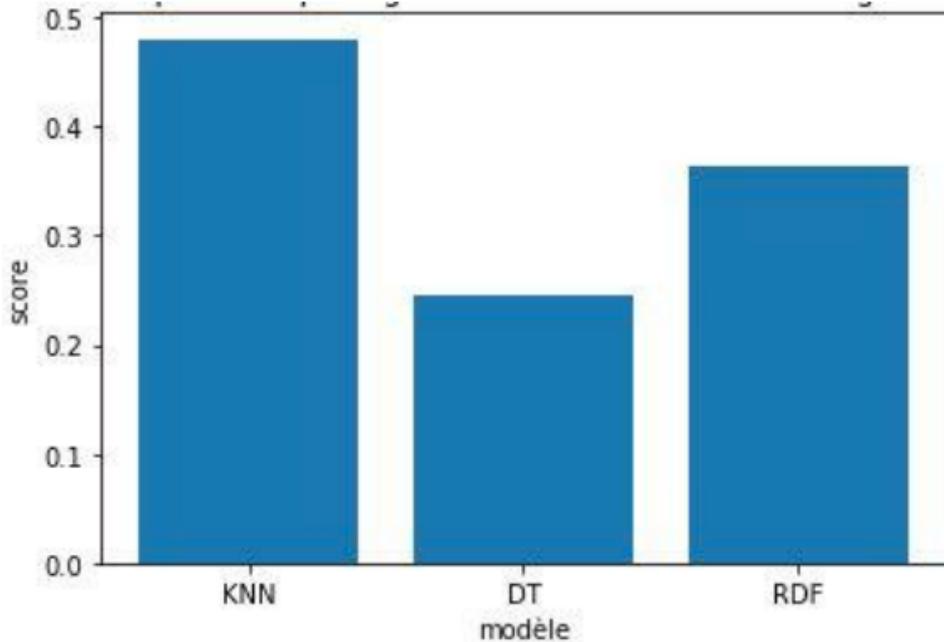


FIGURE 13 – Score du modèle

**Premièrement**, le nombre de données énormes et l'augmentation de nombre de familles peuvent réduire l'efficacité de ce modèle.

**Deuxièmement**, l'application de PCA peut vraiment nuire à la performance des modèles à base d'arbre (arbre de décision) ; puisque les motifs (colonnes) restent importants pour ce genre de modèles. Donc pour éviter l'application de PCA et appliquer les modèles de classification, nous devons réduire les motifs (dimension) inutiles pour notre base de données et régler le problème des données imbalancées. C'est ce que nous allons voir dans la partie suivants.

#### 5.1.4 Pré-traitement de la base de données

Dans cette partie nous nous intéressons à la suppression des motifs qui ne sont pas représentatifs pour notre base de données.

Il existe plusieurs approches pour la sélection d'attributs importants, à savoir l'utilisation des fonctions qui existent dans la bibliothèque sklearns. L'utilisation de ces fonctions nous permet de choisir les attributs avec un pourcentage .Or cette sélection peut supprimer des motifs importants pour la base de données .En effet on ignore le pourcentage de la sélection. Il existe aussi la sélection d'attributs avec l'arbre de décision et d'autres modèles qui dérivent de cette dernière. Mais le problème réside dans le modèle. Nous ne savons pas si ce modèle classifie bien nos séquences ; en plus chaque modèle à son propre approche pour le choix d'attributs. Pour éviter tout problème .on va implémenter nous même un algorithme pour la sélection des attributs (motifs) importants.

Soit  $n$  : le nombre de données de la base de données (nombre de séquences)

Soit  $n_s$  : la taille d'une famille.

Soit  $e$  : un motif (attribut)

Soit  $occ(e, C_S)$  : nombre d'occurrence de motif dans la famille  $C_S$

Soit  $occ(e, TD)$  : nombre d'occurrence de motifs  $e$  dans la base de données.

A savoir que quelle que soit la fréquence des motifs dans une séquence, l'occurrence de ce motif est mesurée indépendamment de cette fréquence .Donc l'apparition d'un motif dans une séquence est mesurée par  $occ=1$  quelle que soit sa fréquence dans cette séquence.

Notre but est de calculer  $WLF(e, C_S)$  et  $WGF(e, TD)$

Avec :

$$WLF(e, C_S) = \frac{occ(e, C_S)}{n_s} \times \frac{occ(e, C_S)}{occ(e, TD)}$$

Cette fonction mesure l'importance d'un motif à être spécifique pour une famille donnée.

Où :

- (a) :  $\frac{occ(e, C_S)}{n_s}$  : mesure l'importance d'un motif dans une famille. Une valeur près de 1 implique que ce motif est important pour cette famille
- (b) :  $\frac{occ(e, C_S)}{occ(e, TD)}$  mesure l'unicité de ce motif pour une famille de données .une valeur proche de 1 explique l'unicité de motif pour une famille

Alors si le produit de (a) et (b)  $WLF(e, C_S)$  proche de 1 explique l'importance et l'unicité de ce motif pour une famille donnée. (Sinon si il est proche de 0), cela peut être expliqué soit par l'existence de ce motif dans toutes les autres familles ( $occ(e, TD) >> occ(e, C_S)$ ), soit par la non présence de ce motif dans la famille. En effet ce motif sera négligeable devant les autres motifs caractérisant cette famille.

L'application de cette mesure va nous permettre de supprimer les rares motifs de grande taille (taille de chaîne de caractères) existants dans les familles.

$$WGF(e, TD) = \frac{occ(e, TD)}{n} \times (n - occ(e, TD) + 1)$$

Cette fonction mesure l'importance globale d'un motif dans la base de données

Avec :

- (c) :  $\frac{occ(e, TD)}{n}$  : Cette mesure qui dévalorise les motifs rares. Si  $occ(e, TD) << n$ , alors WGF tend vers 1, sinon  $WGF > 1$
- (d) :  $(n - occ(e, TD) + 1)$  : Cette mesure dévalorise les motifs trop fréquents, c'est à dire les motifs qui existent dans toutes les familles, parce qui ils ne vont pas être représentatifs. Cette mesure va nous permettre de supprimer les motifs de petites tailles de la chaîne de caractère.

Une seule limite se pose lors de l'application de cette mesure. En effet elle ignore la variation de la fréquence de ces petits motifs selon chaque groupe de séquences représentant une famille.

Ceci fait que cette variation peut contribuer à la classification. Ce problème va être pris en considération lors du choix du seuil appliquée à WGF.

Notre but ici est de supprimer les motifs non représentatifs et d'augmenter juste le temps d'exécution ainsi que l'attribution d'un biais. Voilà donc les visualisations après chaque mesure :

nb_count_all	
<b>ACACAGCCC</b>	12
<b>GCAGCGAGGC</b>	12
<b>GAGGACGC</b>	12
<b>CAAACCCCC</b>	12
<b>AGGCGCAGG</b>	12
...	...
<b>GC</b>	7265
<b>CA</b>	7268
<b>AC</b>	7290
<b>GA</b>	7294
<b>AG</b>	7296

FIGURE 14 – Nombre d'occurrence total

	nb_count_cluster	num_classe
<b>AAAAAAAAAA</b>	0	0
<b>GCCAAGGGG</b>	0	0
<b>GCCAAGGG</b>	0	0
<b>GCCAAGGCA</b>	0	0
<b>GCCAAGGCCA</b>	0	0
...	...	...
<b>CCCGG</b>	27	300
<b>CCGG</b>	27	300
<b>CA</b>	27	300
<b>AG</b>	27	300
<b>CC</b>	27	300

4571588 rows × 2 columns

FIGURE 15 – Nombre d'occurrence suivant chaque famille

	nb_count	num_classe	nombr_classe	OCC
<b>AAAAAAAAAA</b>	0	0	15188	33
<b>GCCAAGGGG</b>	0	0	15188	35
<b>GCCAAGGG</b>	0	0	15188	84
<b>GCCAAGGCA</b>	0	0	15188	30
<b>GCCAAGGCCA</b>	0	0	15188	17
...	...	...	...	...
<b>CCCGG</b>	27	300	15188	1978
<b>CCGG</b>	27	300	15188	4039
<b>CA</b>	27	300	15188	7268
<b>AG</b>	27	300	15188	7296
<b>CC</b>	27	300	15188	7171

FIGURE 16 – Calcule de occ =nombre d'occurrence total

	nb_count	num_classe	nombr_classe	OCC	OCCs3	Z	first_s	WGF	WLF
<b>AAAAAAAAAA</b>	0	0	15188	33	0	7264	0	32.855263	0.000000
<b>GCCAAGGGG</b>	0	0	15188	35	0	7262	0	34.836897	0.000000
<b>GCCAAGGG</b>	0	0	15188	84	0	7213	0	83.044408	0.000000
<b>GCCAAGGCA</b>	0	0	15188	30	0	7267	0	29.880757	0.000000
<b>GCCAAGGCCA</b>	0	0	15188	17	0	7280	0	16.962719	0.000000
...	...	...	...	...	...	...	...	...	...
<b>CCCGG</b>	27	300	15188	1978	19683	5319	104693877	1442.020559	0.000024
<b>CCGG</b>	27	300	15188	4039	19683	3258	64127214	1803.599507	0.000012
<b>CA</b>	27	300	15188	7268	19683	29	570807	28.888706	0.000007
<b>AG</b>	27	300	15188	7296	19683	1	19683	1.000000	0.000007
<b>CC</b>	27	300	15188	7171	19683	126	2480058	123.841283	0.000007

4571588 rows × 9 columns

FIGURE 17 – Résultat finale après les calcules

En observant manuellement les résultats, on a déterminé un seuil pour supprimer les motifs non significatifs de notre base de données :

```

final_data_result = final_data_result[final_data_result['WGF'] > 30]
final_data_result_new = final_data_result[final_data_result['WLF'] > 1e-05]
final_data_result_new

```

	nb_count	num_classe	nombr_classe	OCC	OCCs3	Z	first_s	WGF	WLF
ACGAAAGC	3	0	15188	44	27	7253	195831	43.740680	0.000013
ACGAGGCA	3	0	15188	31	27	7266	196182	30.872533	0.000019
CAAACCGA	3	0	15188	56	27	7241	195507	55.577851	0.000011
CAAGCCGG	3	0	15188	50	27	7247	195669	49.664200	0.000012
AGACCACG	3	0	15188	32	27	7265	196155	31.864035	0.000019
...	...	...	...	...	...	...	...	...	...
GACC	27	300	15188	3720	19683	3577	70406091	1823.799342	0.000013
GGCCG	27	300	15188	1866	19683	5431	106898373	1389.013980	0.000026
GACCC	27	300	15188	1685	19683	5612	110460996	1296.082785	0.000028
CCCGG	27	300	15188	1978	19683	5319	104693877	1442.020559	0.000024
CCGG	27	300	15188	4039	19683	3258	64127214	1803.599507	0.000012

69905 rows × 9 columns

FIGURE 18 – Résultat après sélection

Comme le nombre énorme de motif, les données imbalancées ,elle aussi nuisent à la performance des méthodes de classifications. Pour gérer ce problème, on a essayé de visualiser la fréquence des séquences pour chaque famille et de fixer un seul minimal de 20 séquences par famille

Classe	ACGAAAGC	ACGAGGCA	CAAACCGA	CAAGCCGG	AGACCACG	AGACACCC	AAGCCGGA	CGAGACAC	GAGCAGA	AACGAGGC	...	GCAACGGCA
150	30	30	30	30	30	30	30	30	30	30	...	30
61	30	30	30	30	30	30	30	30	30	30	...	30
223	30	30	30	30	30	30	30	30	30	30	...	30
201	30	30	30	30	30	30	30	30	30	30	...	30
188	30	30	30	30	30	30	30	30	30	30	...	30
...	...	...	...	...	...	...	...	...	...	...	...	...
141	20	20	20	20	20	20	20	20	20	20	...	20
53	20	20	20	20	20	20	20	20	20	20	...	20
137	20	20	20	20	20	20	20	20	20	20	...	20
49	20	20	20	20	20	20	20	20	20	20	...	20
157	20	20	20	20	20	20	20	20	20	20	...	20

301 rows × 8876 columns

FIGURE 19 – Fréquence des motifs suivant chaque classe

### 5.1.5 Application de plusieurs modèles de classifications

Dans la partie précédente, on a fait un pré-traitement de la base de données afin de réduire la dimensionnalité et de régler le problème des données imbalancées. Ainsi maintenant on possède une base de données qui ne contient que les motifs intéressants des données plus ou moins balancées pour chaque famille. On va traiter dans cette partie le reste de ce qu'on n'a pas traité dans la partie précédente.

Cette partie s'intitule le choix du modèle parmi ceux qu'on a testé avant. Les tests initiaux nous ont permis d'avoir une vue générale sur les modèles qui peuvent donner un bon résultat. Parmi les modèles qu'on a déjà testé, nous avons : « decision\_tree », « k\_neighbors », « gradient\_boosting », « gaussien NB » , « random\_forest », « logistique\_regression », « MLP (multi-layer) classifier »....

Parmi ces modèles, il existe ceux qui donnent un bon résultat, mais d'autres utilisent beaucoup de temps uniquement pour l'entraînement (plus que 3 heures). Alors imaginez si on les applique avec un cross validation (de 10 splits), le temps sera énorme d'où l'incapacité d'utiliser ces modèles actuellement à cause du confinement et du manque de matériels.

Nous avons ignoré les modèles de faibles résultats ainsi que les modèles longs à exécuter. Donc il nous reste

quatre modèles : « gaussian\_NB » , « k\_neighbors » , « random\_forest » et nous avons ajouté « extra-tree », puisqu'en général ce modèle ressemble à « random-forest » (modèle d'ensembles avec moyen variance) qui a donné un bon score contrairement l'arbre de décision (grande variance). Avant de définir chaque modèle on va procéder à leur première application pour vérifier notre choix.

L'application de ces modèles va se faire à l'aide d'une crosse validation [4] : Nous avons choisi la méthode cross validation pour éviter le problème des données imbalancées qu'on a manqué d'une part, et d'autre part pour utiliser toutes les données et avoir un score moyen plus général. La procédure est la suivante :

- a) diviser les données en m parties (part\_1, part\_2, part\_3,....)
- b) faire varier le nombre de partitions(k) et pour chaque k répéter pour i de 1 à m : entraîner et évaluer un model en utilisant part\_i comme donnée de test et le reste comme donnée d'entraînement, puis calculer la moyenne des scores pour les i itérations.

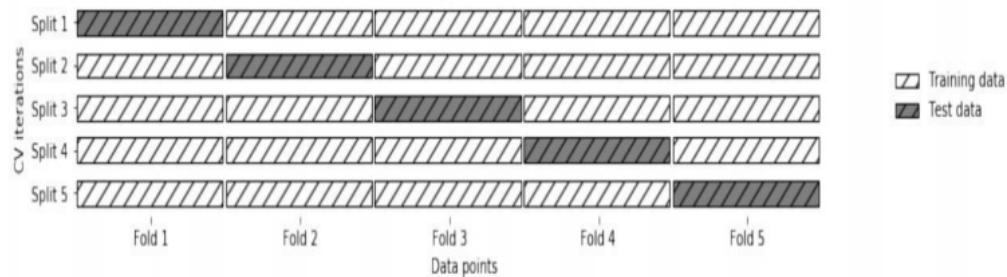


FIGURE 20 – Cross validation

Nous avons vu précédemment que KNN est très sensible au changement de K dans notre base de données. Donc nous avons choisi d'appliquer KNN trois fois avec des K différents.

Nous avons appliqué chaque modèle avec une cross validation de N splits, et nous avons trouvé les résultats suivant :

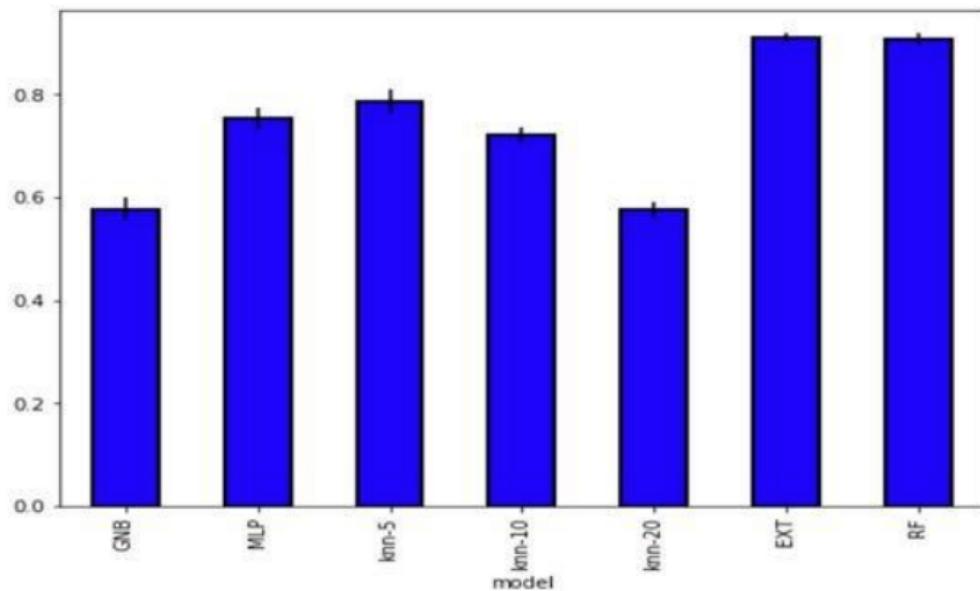


FIGURE 21 – Score obtenu pour chaque modèle

Comme c'était prévu, « extra-tree » a eu le meilleur score. et même plus que « random\_forest ». Pour « K-neighbors ». On a remarqué que le score diminue avec l'augmentation du k. Donc on va garder KNN espérant qu'il donnera un bon résultat lors du choix des paramètres.

### 5.1.6 Choix du modèle

Pour choisir le meilleur parmi les trois modèles. Nous allons tout d'abord procéder à leur para-métrisation. C'est à dire le choix du meilleur paramètre pour chaque modèle afin d'avoir un meilleur score ainsi que pour faire une comparaison finale des modèles et du choix le plus adopté.

Tout d'abords nous allons définir les trois modèles et la différence entre eux

**KNN** : L'algorithme des k plus proches voisins (k-NN) est certainement un des algorithmes les plus simples d'apprentissage automatique[6]. Il est motivé par l'idée que des entrées  $x_t$  semblables devraient avoir des cibles  $y_t$  semblables. Ainsi, pour bien définir un algorithme k-NN, il suffit de définir ce que veut dire "semblable" dans le contexte des entrées et de définir l'influence de ces voisins sur la prédiction de la cible pour une entrée de test. Donc, pour obtenir une prédiction de la cible pour une entrée de test x, il suffit de trouver les k plus proches voisins selon une métrique déterminant jusqu'à quel point des entrées sont semblables (par exemple, la distance euclidienne ou norme  $L^2$  2, ou de façon plus générale la norme  $L^p$  de Minkowski) et d'utiliser ces k plus proches voisins pour prédire la cible de x. Dans un problème de classification, la prédiction correspond à la classe majoritaire parmi les k plus proches voisins, i.e. que l'ensemble des k plus proches voisins votent pour la classe correspondant à leur cible respective et la classe recueillant le plus de vote est choisie en tant que prédiction par l'algorithme.

**Arbre de décision** : Un arbre de décision est un schéma représentant les résultats possibles d'une série de choix interconnectés. Il permet à une personne ou une organisation d'évaluer différentes actions possibles en fonction de leur coût, leur probabilité et leurs bénéfices. Il peut être utilisé pour alimenter une discussion informelle ou pour générer un algorithme qui détermine le meilleur choix de façon mathématique.

Un arbre de décision commence généralement par un noeud d'où découlent plusieurs résultats possibles. Chacun de ces résultats mène à d'autres noeuds, d'où émanent d'autres possibilités [6].

L'arbre de décision se base sur deux ou trois mesure en général ; son objectif et d'augmenter le gain pour chaque étape de découpage suivant un motif. Les mesures utilisé en général sont :

L'entropie :

$$E(P) = \sum_{i=1}^n p_i \log(p_i)$$

Et le gain :

$$Gain(X, T) = E(T) - E(X, T) = E(T) - \sum_{j=1}^n \frac{|T_j|}{|T|} E(T_j)$$

Ce modèle semble parfait pour notre base de donnée car, nous constatons qu'il existe un groupe de motif caractérisant chaque famille, or le nombre énorme de séquence et de famille peuvent poussée l'arbre de décision a un sur apprentissage.

Ce sur-apprentissage est expliqué par le faible biais et la grande variance du modèle. D'où la solution est le baggin.

**Bagging** :

- Génération de k échantillons « indépendants » par tirage avec remise
- Pour chaque échantillon, apprentissage d'un classifieur En utilisant le même algorithme d'apprentissage
- La prédiction finale pour un nouvel exemple est obtenue par un Vote majoritaire (classification).

L'intérêt du bagging réduit la variance quand les prédicteurs sont instables. D'où la réduction du problème de sur-apprentissage.

L'un des modèles qui utilise la stratégie de bagging dans le cas spécifique de l'arbre de décision sont random-forest et extra-tree.

**Random forest** : est un algorithme de classification qui réduit la variance des prévisions d'un arbre de décision seul, améliorant ainsi leurs performances. Pour cela, il combine de nombreux arbres de décisions dans une approche de type bagging[8]. Dans sa formule la plus classique, il effectue un apprentissage en parallèle sur de multiples arbres de décision construits aléatoirement et entraînés sur des sous-ensembles de données différents. Le nombre idéal d'arbres, qui peut aller jusqu'à plusieurs centaines voire plus, est un paramètre important : il est très variable et dépend du problème. Concrètement, chaque arbre de la forêt aléatoire est entraîné sur un sous ensemble aléatoire de données selon le principe du bagging, avec un sous ensemble aléatoire de features (caractéristiques variables des données) selon le principe des « projections

aléatoires ». Les prédictions sont ensuite moyennées lorsque les données sont quantitatives ou utilisés pour un vote pour des données qualitatives, dans le cas des arbres de classification. L'algorithme des forêts aléatoires est connu pour être un des classificateurs les plus efficaces « out-of-thebox » (c'est-à-dire nécessitant peu de prétraitement des données). Il a été utilisé dans de nombreuses applications, y compris grand public, comme pour la classification d'images de la caméra de console de jeu Kinect\* dans le but d'identifier des positions du corps [6]. Or sont une amélioration du bagging pour les arbres de décision CART dans le but de rendre les arbres utilisés plus indépendants (moins corrélés). Caractéristiques :

- Elles donnent de bons résultats surtout en grande dimension,
- Elles sont très simples à mettre en œuvre,
- Elles ont peu de paramètres.

**ExtraTreesClassifier** : est une méthode d'apprentissage d'un ensemble fondamentale basé sur des arbres de décision. ExtraTreesClassifier, comme RandomForest randomise certaines décisions et certains sous-ensembles de données pour minimiser le sur-apprentissage des données et le sur-apprentissage.

Examinons quelques méthodes d'ensemble classées de la variance la plus élevée à la plus faible, se terminant par ExtraTreesClassifier.

Arbre de décision (variance élevée) : Un arbre de décision unique est généralement trop adapté aux données dont il apprend parce qu'il apprend d'une seule voie de décisions. Les prédictions à partir d'un seul arbre de décision ne font généralement pas de prédictions précises sur les nouvelles données.

Random -forest (variance moyenne) : Les modèles forestiers aléatoires réduisent le risque de surajustement en introduisant le hasard en :

- Construisant plusieurs arbres (n\_estimators)
- Tirant des observations avec remplacement (c.-à-d. un échantillon amorcé)
- Fractionnant les nœuds sur la meilleure répartition parmi un sous-ensemble aléatoire des entités sélectionnées à chaque nœud

Extra trees (faible variance) : Extra Trees est comme Random Forest, en ce qu'il construit plusieurs arbres et divise les nœuds en utilisant des sous-ensembles aléatoires de fonctionnalités mais avec deux différences clés : il ne démarre pas les observations (ce qui signifie qu'il échantillonne sans remplacement), et que les nœuds sont divisés aléatoirement, pas les meilleures divisions. Donc, en résumé, ExtraTrees : construit plusieurs arbres avec bootstrap=Faux par défaut, ce qui signifie qu'il échantillonne sans remplacement

Les nœuds sont répartis sur la base de répartitions aléatoires parmi un sous-ensemble aléatoire des entités sélectionnées à chaque nœud

Dans Extra-Trees, l'aléatoire ne vient pas du bootstrapping des données, mais vient plutôt des divisions aléatoires de toutes les observations.

Extra-Trees est nommé pour (arbres extrêmement randomisés).

### 1. Une optimisation et un choix des paramètres pour les modèles choisis :

Avant d'entamer les étapes nous procéderons à la recherche d'hyper-paramètres pour « kNeighbors ». En variant k dans l'intervalle [2,4,6] nous appliquerons le modèle et nous afficherons les scores obtenus :

```
N_neighbors= 2 ;Mean train score 0.9318092588298941 ; Mean validate score 0.821029227705465
N_neighbors= 6 ;Mean train score 0.8197572629110598 ; Mean validate score 0.755824169926648
N_neighbors= 10 ;Mean train score 0.7491186252007781 ; Mean validate score 0.6998234722491093
```

FIGURE 22 – Les différents résultats d'entraînement de test de différents k

Nous avons conclu que k =2 est le paramètre optimal qui donne le meilleurs score.

En sélectionnant trois modèles qui gèrent bien la classification de nos séquences, nous pouvons augmenter l'occurrence du score pour chacun des modèles si nous prenons un bon choix de paramètre comme nous l'avons vu avec k-Neighbors.

Dans cette partie nous traitons le bon choix des paramètres pour « random-forest » et « extratree ». L'approche que nous allons suivre est « grid-search » sur deux paramètres « max-dept » et « n-estimators ».

« max-dept » présente la profondeur de chaque arbre dans cette forêt. Plus l'arbre est profond plus il

est devisé et plus il capture l'information sur les données. D'où l'avantage pour la grandeur de notre base de données.

« n-estimators » : présente le nombre d'arbres dans cette foret, car plus le nombre d'arbre est élevé, mieux c'est bon pour apprendre les données. Cependant l'ajout de nombreux arbres peut augmenter considérablement le temps d'exécution et ralentir le processus de formation. Ces deux paramètres seuls nous semblent très intéressant et suffisants pour notre grid-search.

Il existe d'autres paramètres mais nous les avons laissés par default puisqu'ils ne donnent pas un grand changement et ils peuvent contribuer à un « over-fitting ».

**min\_samples\_split** représente le nombre minimum d'échantillons requis pour diviser un noeud interne. Cela peut varier entre l'examen d'au moins un échantillon ou tous les échantillons à chaque noeud.

**max\_features** représente le nombre de fonctionnalités à considérer lors de la recherche de la meilleure répartition.

**min\_samples\_leaf** est le nombre minimum d'échantillons requis pour être au niveau d'un noeud feuille. Ce paramètre est similaire à **min\_samples\_splits**, cependant, il décrit le nombre minimum d'échantillons au niveau des feuilles.  
newline Tout d'abord, expliquons le concept de « grid-search »  
**gridSearch** : La méthode de recherche de grille est la plus facile à implémenter et à comprendre, mais malheureusement elle n'est pas efficace lorsque le nombre de paramètres est grand. Dans notre cas, on n'a que 2 paramètres à optimiser ce qui semble parfait [9].

2. **Une analyse des résultats des modèles :** Soit les paramètres spatiaux = (1, 2, ..., m) sur lesquels nous maximisons le score. Une manière simple de configurer une recherche de grille consiste à définir un vecteur de bornes inférieures  $a = (a_1, a_2, \dots, a_m)$  et un vecteur de bornes supérieures  $b = (b_1, b_2, \dots, b_m)$  pour chaque composante de .

La recherche de la grille implique de prendre n points également espacés dans chaque intervalle de la forme  $[a_i, b_i]$  y compris  $a_i$  et  $b_i$ . Cela crée un total de  $n^m$  points de grille possibles à vérifier. Enfin, une fois que chaque paire de points est calculée, le maximum de ces valeurs est choisi.

Le problème avec ce type de méthode c'est que le nombre d'évaluations augmente de façon exponentielle à mesure que n et m augmentent. Comme nous ne pouvons pas vraiment réduire m, la diminution de n est le seul moyen possible de garantir l'arrêt de la méthode dans un temps raisonnable. Cependant cela diminue la validité de la solution. Il existe une autre méthode de grille plus performante. Au lieu de prendre des valeurs exactes parmi ce que nous avons mis, elle prend des valeurs aléatoires dans un intervalle global que nous définissons. Cela donne plus de chance de tomber directement sur la bonne valeur. Or puisque notre cas donne de bons résultats, on s'appuiera seulement sur la recherche classique qu'on a citée.

Les paramètres qu'on a choisi sont max-dept : [50,100 ,150] et n-estimators : [50,200 ,700]. Pour le max-features. On a testé les paramètres dans une base de données de test et on a constaté que « log2 » est le meilleur pour notre cas. Donc pour réduire le temps d'exécution .nous avons pris uniquement deux paramètres (n-estimators , max-dept ) avec max-features= « log2 ». Puis nous appliquons grid-search avec un cross validation de 5 ,pour une meilleur généralisation .

```
param_grid = {
    'max_depth': [50,100,150],
    'n_estimators': [50,200, 700],
    'max_features': ['log2']#regardant le temps de calcul de ce gridsearch on a laisser que log2 apres une excution
                        #puisque c'est le meilleure parametre obtenu
}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X, y)
print(CV_rfc.best_params_)
```

FIGURE 23 – Les différents paramètres choisis

```

CV_etc = GridSearchCV(estimator=etc, param_grid=param_grid, cv= 5)
CV_etc.fit(X, y)
print(CV_etc.best_params_)

{'max_depth': 150, 'max_features': 'log2', 'n_estimators': 700}

]: resultat_scor_etc = CV_etc.best_score_
print(resultat_scor_etc)

0.9481094663725177

```

FIGURE 24 – Meilleure paramètres, score pour extra-tree

```

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X, y)
print(CV_rfc.best_params_)

'max_depth': 150, 'max_features': 'log2', 'n_estimators': 700

resultat_score_rfc = CV_rfc.best_score_
print(resultat_score_rfc)

.9537897882241999

```

FIGURE 25 – Meilleure paramètres, score pour random-forest

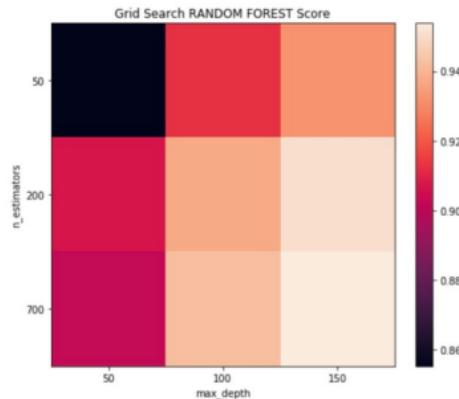


FIGURE 26 – Recherche de grille pour random-forest

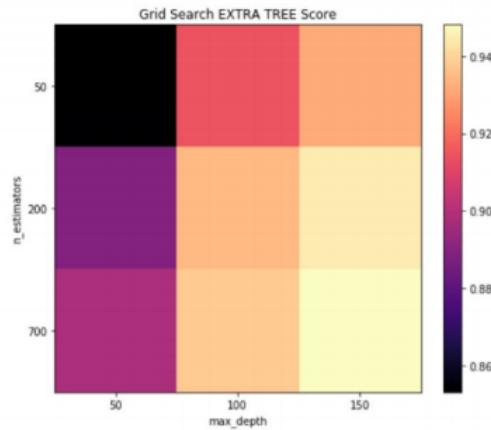


FIGURE 27 – Recherche de grille pour extra-tree

Les résultats pour les deux modèles semblent les mêmes :max-dept :150, N – estimators : 700. Le problème des fois avec une valeur élevé de n-estimators c'est qu'il peut contribuer à un sur-apprentissage. Nous allons gérer ce problème dans les étapes suivantes.

(1) —> premièrement, même avec l'utilisation de la grille cela ne semble pas parfait car on a un score très proche des deux modèles. Pour gérer ce problème, on a pensé à faire une cross validation et à calculer la variation du score. Puis on prend l'écart type et on visualise les résultats. En effet, le meilleur modèle, c'est celui qui possède avec un meilleur score (moyen score pour cross-val) et un minimum de variances pour les différents scores générés (le modèle stable). En appliquant cette approche voilà ce que nous trouvons :

model	cv_mean	cv_std
0 EXT	0.886628	0.012810
1 knn-2	0.834161	0.016714
2 RF	0.882909	0.010361

FIGURE 28 – Resultat des moyens score et leur écart-type

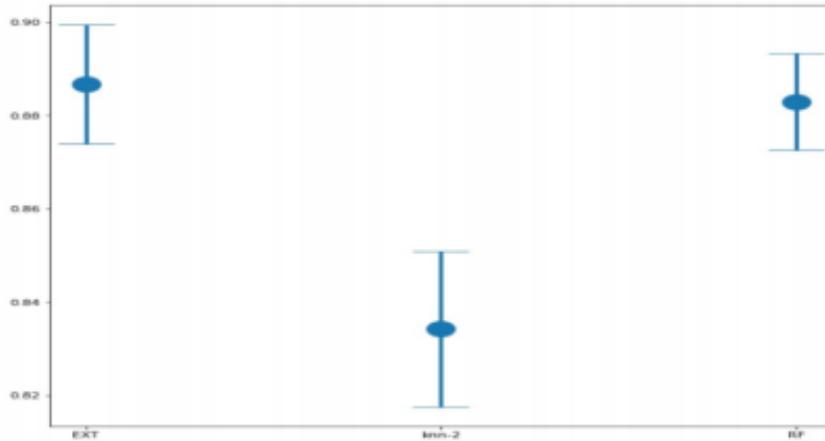


FIGURE 29 – Comparaison de la variance des scores obtenus par 5 cross validation

Nous constatons bel et bien que « extra-tree » est le meilleur modèle.

(2) —>deuxièmement une dernière confirmation de ce modèle. Vérifions ces modèles pour l'ensemble de donnée de test.

Nous prenons les meilleurs paramètres pour chaque modèle. Puis nous l'entraînons et nous calculons la prédiction pour chacun de ces modèles. En fin, nous affichons le score d'entraînement et de test :

```
]:
knc = KNeighborsClassifier(n_neighbors=2)
knc.fit(X_train,y_train)
pred_train = knc.predict(X_train)
pred_test = knc.predict(X_test)
train_score, test_score = accuracy_score(y_train,pred_train),accuracy_score(y_test,pred_test)
print('KNeighborsClassifier')
print('----->')
print("N_neighbors=",2,";Train score", train_score, "; Test score", test_score)
train_scores.append(train_score)
test_scores.append(test_score)

KNeighborsClassifier
----->
N_neighbors= 2 ;Train score 0.9410612884276484 ; Test score 0.8254910918227502
```

FIGURE 30 – Résultat d'entraînement et de test pour K-Neighbores

```
]:
ext = ExtraTreesClassifier(max_depth=150,n_estimators = 700, random_state = 10)
ext.fit(X_train,y_train)
pred_train = ext.predict(X_train)
pred_test = ext.predict(X_test)
train_score, test_score = accuracy_score(y_train,pred_train),accuracy_score(y_test,pred_test)
print('ExtraTreesClassifier')
print('----->')
print("max_depth=",150,";Train score", train_score, "; Test score", test_score)
train_scores.append(train_score)
test_scores.append(test_score)

ExtraTreesClassifier
----->
max_depth= 150 ;Train score 1.0 ; Test score 0.8830516217450891
```

FIGURE 31 – Résultat d'entraînement et de test pour extra-tree

```
]:
rdf = RandomForestClassifier(max_depth=150,n_estimators = 700, random_state = 10)
rdf.fit(X_train,y_train)
pred_train = rdf.predict(X_train)
pred_test = rdf.predict(X_test)
train_score, test_score = accuracy_score(y_train,pred_train),accuracy_score(y_test,pred_test)
print('RandomForestClassifier')
print('----->')
print("max_depth=",150,";Train score", train_score, "; Test score", test_score)
train_scores.append(train_score)
test_scores.append(test_score)

RandomForestClassifier
----->
max_depth= 150 ;Train score 0.9998041903270022 ; Test score 0.8734582000913659
```

FIGURE 32 – Résultat d'entraînement et de test pour random-forest

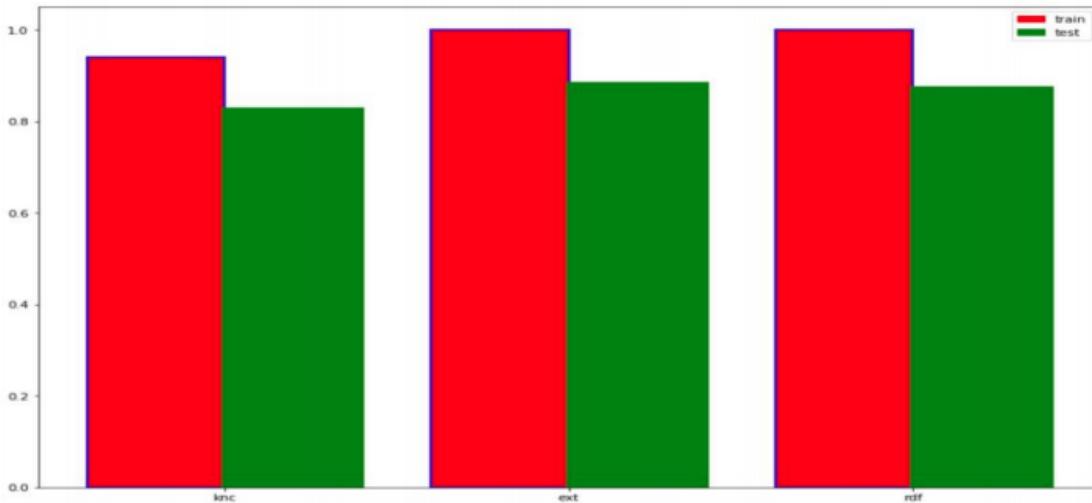


FIGURE 33 – Résultat des scores de test et d’entraînement pour les différents modèles en utilisant les meilleurs paramètres

On constate bien que les deux modèles « randome-forest » et « extra-tree » sont meilleurs que « k\_nearest\_neighbors ». Seulement, Ils ont presque le même score d’entraînement et presque le même score de test avec un petit avantage pour « extra-tree ».

Donc d’après (1) et (2) on conclut que « Extra-tree » est le meilleur modèle dans notre cas.

### 5.1.7 Évaluation des résultats

Passons maintenant à la phase de l’évaluation de notre modèle. Pour l’évaluation et étude de la qualité de notre classification, nous préférons choisir les trois mesures : rappel, précision et f-mesure.

Quelques définitions des mesures :

La précision : La précision est le rapport :  $\frac{tp}{tp+fp}$  où tp est le nombre de vrais positifs ainsi que fp est le nombre de faux positifs. La précision est intuitivement la capacité du classificateur à ne pas étiqueter comme positif un échantillon négatif.

Le rappel nous permet de répondre à la question suivante : Quelle proportion de résultats positifs réels a été identifiée correctement ?

Le rappel : Le rappel est le rapport  $\frac{tp}{tp+fn}$  où tp est le nombre de vrais positifs et fn le nombre de faux négatifs. Le rappel est intuitivement la capacité du classificateur à trouver tous les échantillons positifs.

La mesure F1 pris est :  $\frac{2}{recall^{-1}+precision^{-1}} = 2 \times \frac{precision \times recall}{precision + recall}$  ou en général F-bêta tel que :  $F_\beta = (1 + \beta^2) \times \frac{precision \times recall}{(\beta^2 \times precision) + recall}$  est nécessaire lorsque nous souhaitons rechercher un équilibre entre Précision et Rappel. L’exactitude peut être largement apportée par un grand nombre de vrais négatifs, Puisque dans notre cas les deux mesures semblent intéressantes. Donc le score F1 pourrait être une meilleure mesure à utiliser surtout si nous devons rechercher un équilibre entre la précision et le rappel ou bien lorsque la répartition des classes est inégale (grand nombre de négatifs réels).

Le score F-bêta ou le F-mesure : Le score F-bêta peut être interprété comme une moyenne harmonique pondérée de la précision et du rappel, où un score F-bêta atteint sa meilleure valeur à 1 et son pire score à 0. Le bêta est égal à 2 ici. Puisque les deux mesures comme nous l’avons déjà dit sont de la même importance.

	precision	recall	f1-score
297	0.8	0.67	0.73
298	1	1	1
299	0.88	1	0.93
300	0.89	0.89	0.89
accuracy	0.88	0.88	0.88
macro avg	0.89	0.89	0.87
weighted avg	0.91	0.88	0.88

FIGURE 34 – Résultats des scores rappel, précision et f-mesure

	0.83	0.83	0.85
288	1	0.12	0.22
289	0.88	1	0.93
290	0.83	1	0.91
291	0	0	0
292	1	1	1

FIGURE 35 – Scores pour les deux familles 292 et 289

Pour les scores selon chaque famille ; nous avons réglé le problème des imbalances. Donc on a belle et bien trouvé de bons scores pour la plupart des familles. Seules les deux familles 292 et 289 ont eu de mauvais résultats.

D'autres familles ont des scores imblancés entre la précision et le rappel exemple des familles 117 184 et 255. Je pense que cela est dû au manque de motifs caractérisant ces familles. Ou que la partie d'entraînement du modèle qui n'a pas eu assez de données pour ces familles.

Nous pouvons régler le problème en utilisant d'autres méthodes de cross\_validation mais puisque le résultat est assez bon dans l'ensemble, nous allons le garder.

### 5.1.8 Extraction des motifs

L'extraction des motifs importants pour la classification des familles est une étape cruciale dans notre projet. D'une part à travers cette extraction, nous pouvons générer des matrices de données avec un plus grand nombre de motifs et de familles. D'autre part elle va nous permettre d'exécuter nos modèles de manière rapide et ainsi tester la base de données globale avec 3016. Nous pouvons aller plus loin ; pour classifier les séquences selon les familles, ces motifs ont beaucoup de poids contrairement aux autres. Cela va permettre en utilisant un clustering d'identifier quelques groupes de motifs caractérisant chaque famille.

Pour faire l'extraction des motifs, nous allons tout d'abord afficher l'importance de chacun. Cette importance est obtenue par l'application du modèle extra-tree. Voilà Les résultats obtenus :

motifs_importance	
<b>CGCAAGCG</b>	8.371186e-07
<b>CGCCCGGA</b>	8.436999e-07
<b>GGACGACC</b>	1.003299e-06
<b>CACCAACACC</b>	1.025999e-06
<b>GCGCGGGCG</b>	1.036936e-06
...	...
<b>AACACAAAC</b>	2.023848e-03
<b>GACGCC</b>	2.103213e-03
<b>GGCAAAGC</b>	2.122471e-03
<b>GCACACCA</b>	2.350602e-03
<b>ACGACAGC</b>	2.369849e-03

FIGURE 36 – L'importance de chaque motifs pour cette classification

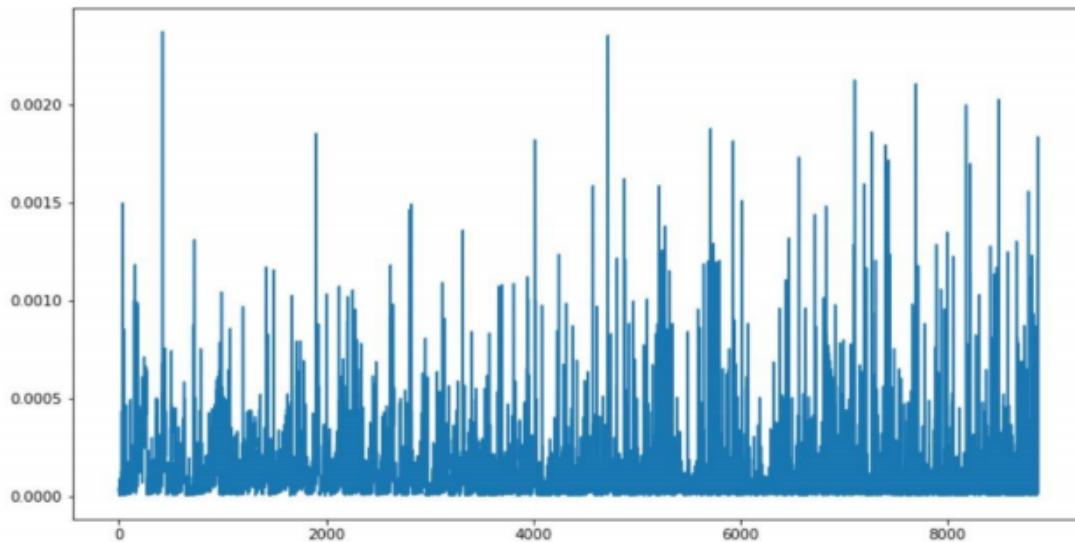


FIGURE 37 – L'importance de chaque motifs

Nous constatons bien que l'importance varie selon les motifs. La plupart d'entre eux ont une importance presque négligeable devant les autres. Pour régler ce problème, nous allons procéder à la sélection des motifs qui représente une importance élevée. Nous allons trier ces valeurs pour déterminer la valeur du seuil qu'on va prendre.

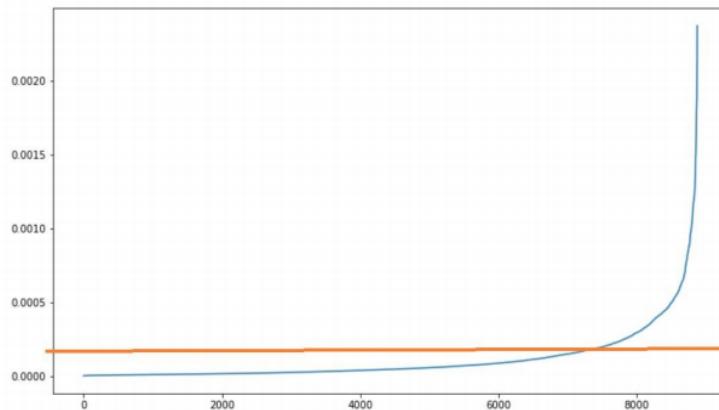


FIGURE 38 – Seuil d'importance

La valeur du seuil pris correspond au virage '0.0002'. C'est à dire là où la courbe commence à augmenter de façon exponentielle. Nous visualisons ces valeurs avec la loi normale et le box plot pour être sûr de notre seuil choisi.

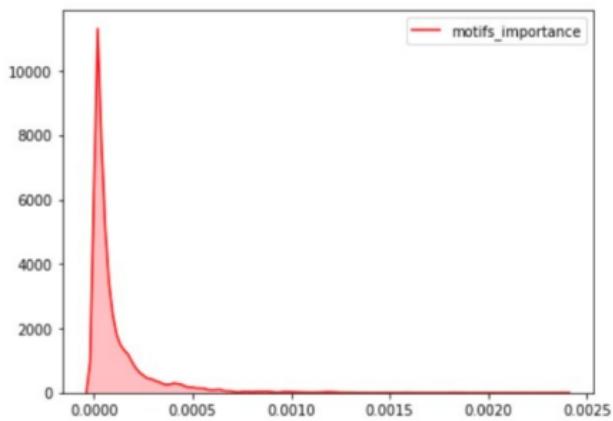


FIGURE 39 – Distribution normal des valeurs

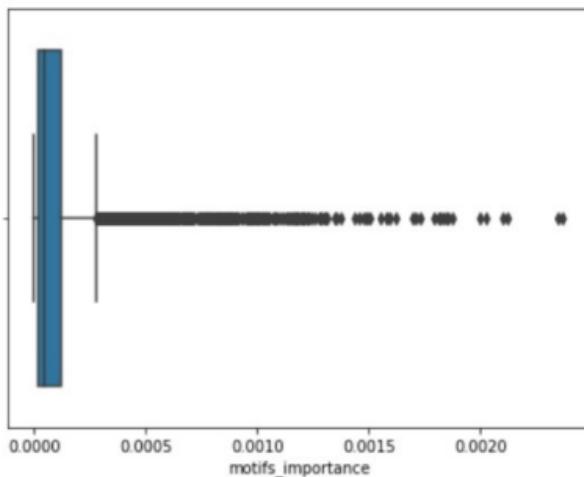


FIGURE 40 – Boxplot des valeurs obtenues

Avant d'extraire les motifs importants, nous essayerons de voir l'information au niveau de leurs longueurs. Nous essayerons de voir si les longueurs des chaines de caractères des motifs sont corrélées avec leurs importances.

	<b>motifs_importance</b>	<b>len_motif</b>
<b>ACGAAAGC</b>	0.000024	8
<b>ACGAGGCA</b>	0.000025	8
<b>CAAACCGA</b>	0.000044	8
<b>CAAGCCGG</b>	0.000015	8
<b>AGACCACG</b>	0.000014	8

FIGURE 41 – Longueur et importance des motifs

Nous utilisons les boîtes à moustaches pour visualiser la distribution d'importance suivant chaque catégorie de longueur.

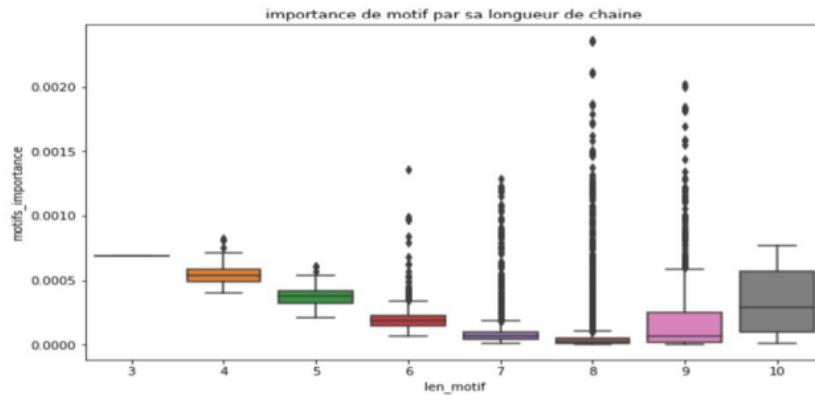


FIGURE 42 – Boxplot d’importance en fonction des longueurs des motifs

Nous constatons que les motifs de petites tailles Ainsi que les motifs de grandes tailles ont une grande importance. Cela semble évident. D'où, nous concluons que les motifs de grande taille sont spécifiques à chaque classe et c'est eux qui déterminent la classification. Pour ce qui est les motifs de petite taille, ils sont de fréquence unique pour chaque groupe de séquences caractérisant une famille. Donc ils peuvent contribuer à la classification. Nous procéderons à l'extraction des motifs importants selon le seuil déjà déterminé.

```
Best_motifs = motifs_importance[motifs_importance["motifs_importance"] > 0.0002]
index_best_features=Best_motifs.index
data_motif_reduced=data_motif[index_best_features]
```

FIGURE 43 – Motifs importants

Essayons de refaire l'entraînement ainsi que l'évaluation des modèles en utilisant ces motifs importants

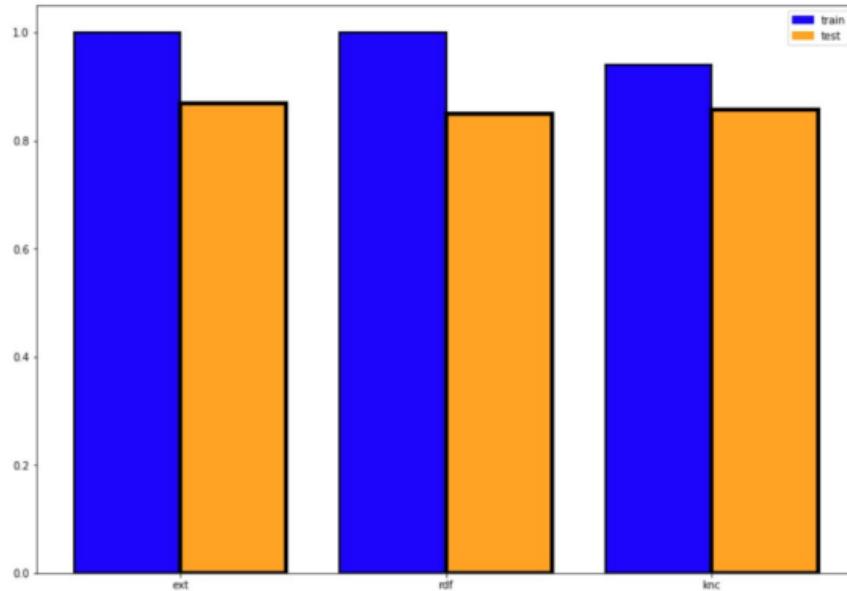


FIGURE 44 – Résultat ENTRAÎNEMENT / TEST

On constate que nous avons eu les mêmes résultats qu'avant. Malgré la réduction de presque 80% des motifs.

Évaluons maintenant notre modèle :

	precision	recall	f1-score
288	1	0.62	0.77
289	0	0	0
290	1	0.5	0.67
291	0.86	1	0.92
292	1	1	1
300	0.9	1	0.95
accuracy	0.86	0.86	0.86
macro avg	0.89	0.87	0.86
weighted avg	0.9	0.86	0.86

FIGURE 45 – Précision, rappel et F1-mesure

La seule classe avec 0 score c'est la classe numéro 289. Nous constatons qu'il sagit de la même classe trouvée avant. Nous avons des scores Im-balancés pour les autres familles mais moins que dans l'évaluation précédente. Le score moyen est presque du même niveau que celui qui précède. En outre cela aurait été mieux si on a pu utiliser d'autres mesures mais la contrainte du nombre énorme de famille nous en empêche. Le nombre de motifs restant est de l'ordre de 1300, visualisons maintenant l'importance de ces motifs.

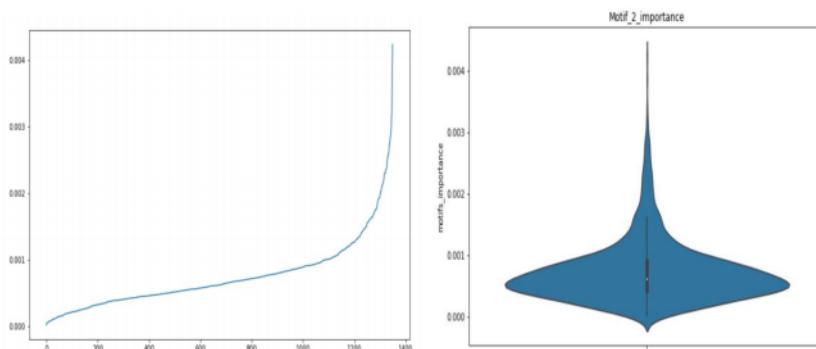


FIGURE 46 – Importance des motifs

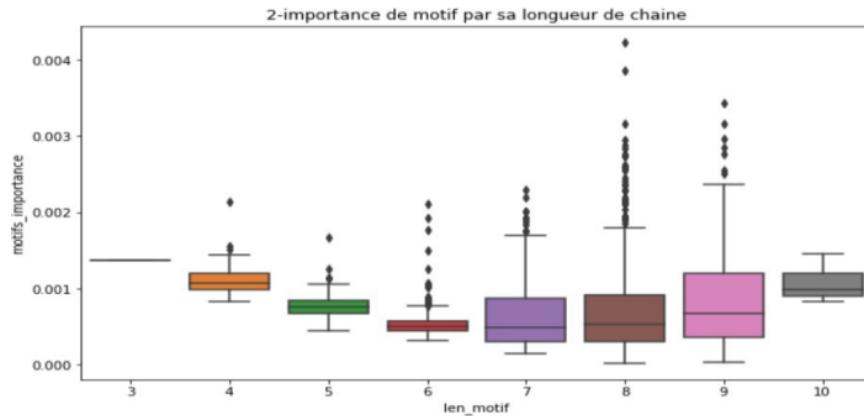


FIGURE 47 – Boxplot d’importance en fonction des longueurs des motifs

### 5.1.9 Visualisation

Pour pouvoir visualiser l’arbre d’extra-tree, nous procéderons à une deuxième extraction des motifs afin de n’en garder qu’un nombre minimal en utilisant la même méthode qu’avant.  
Voilà la nouvelle base de données qui en résulte :

	GACAAGAG	ACGGCGA	CCGGAAC	CGAAC	AAGAGC	CGCC	ACCC	CCCG	GGGC	GGGG	...
0	0	0	1	1	3	3	7	3	4	8	...
1	0	0	1	1	3	3	7	3	4	8	...
2	0	0	1	1	0	2	7	3	4	7	...
3	0	0	1	1	0	2	8	3	7	8	...
4	0	0	1	1	3	3	8	3	6	8	...
...	...	...	...	...	...	...	...	...	...	...	...
7291	0	0	0	0	0	1	1	3	4	2	...
7292	0	0	0	0	0	1	1	2	6	3	...
7293	0	0	0	0	0	1	1	2	6	3	...
7294	0	0	0	0	0	2	1	4	5	2	...
7295	0	1	0	0	0	0	1	3	5	1	...

FIGURE 48 – Nouvelle base de données

Nous appliquons notre meilleur modèle et nous visualisons l’arbre :

```
ExtraTreesClassifier
----->
max_depth= 150 ;Train score 0.9996083806540043 ; Test score 0.836455002284148
```

FIGURE 49 – classifieur de extra tree

Malgré les deux étapes d’extraction on a du mal à représenter l’arbre, à cause des paramètres choisis.

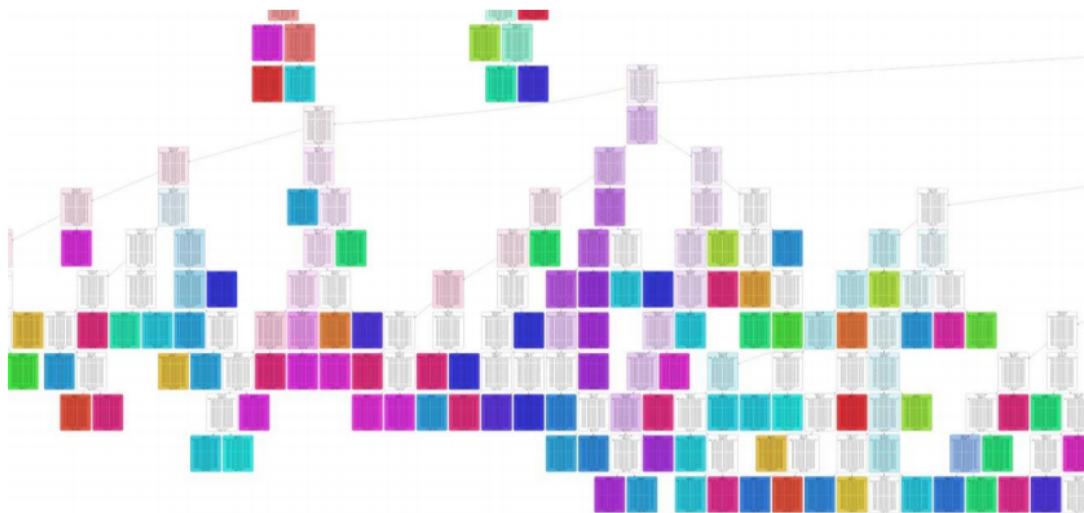


FIGURE 50 – Une partie de la forêt d'arbre générée

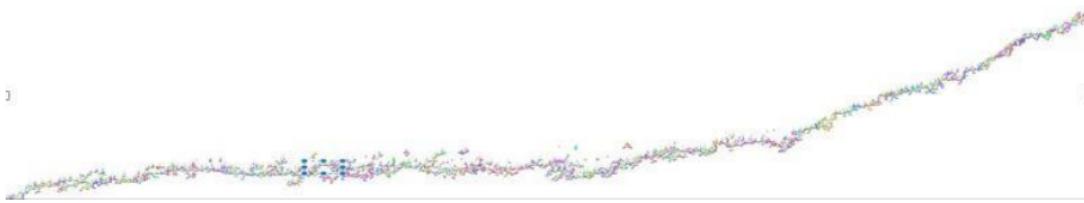


FIGURE 51 – La forêt d'arbre totale

Nous avons trouvé des difficultés lors de la visualisation de cette forêt vu le nombre de motifs et de familles. Ainsi le nombre d'arbres et de noeuds dus aux paramètres choisis. Nous essayerons d'appliquer le modèle d'arbre de décision seulement pour voir s'il y aurait une différence. Nous affichons l'arbre avec "décision tree", mais nous n'avons pas trouvé de meilleurs résultats

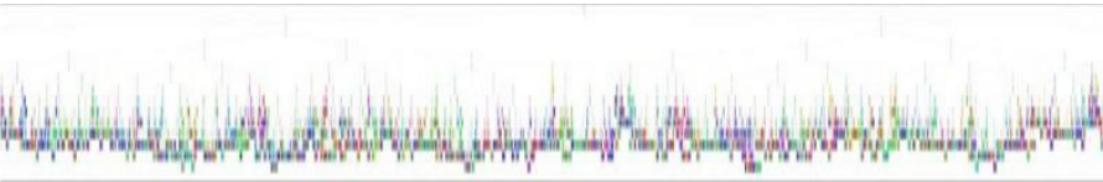


FIGURE 52 – Arbre de décision de cette classification

### 5.1.10 Application sur la base de données globale

Après avoir réussi à faire l'extraction les motifs importants nous avons eu la chance d'appliquer l'un des modèles sur la base de données globale de 3016 familles. La base de données en question à comme taille (49346, 361).

Vu le nombre énorme des familles et des séquences, nous ne sommes pas arrivés à appliquer notre modèle extra-tree avec ses paramètres. En effet l'exécution nous a prise plus de trois heures et sans aucun résultat.

C'est pourquoi nous avons appliqué K-NEIGHBORS puisqu'il est plus rapide et avec de bons résultats. Nous avons appliqué le modèle en utilisant le meilleur paramètre choisi et aussi qu'une validation croisée pour éviter le problème des données imbalancées. Les résultats obtenus alors semblent aussi presque parfaits :

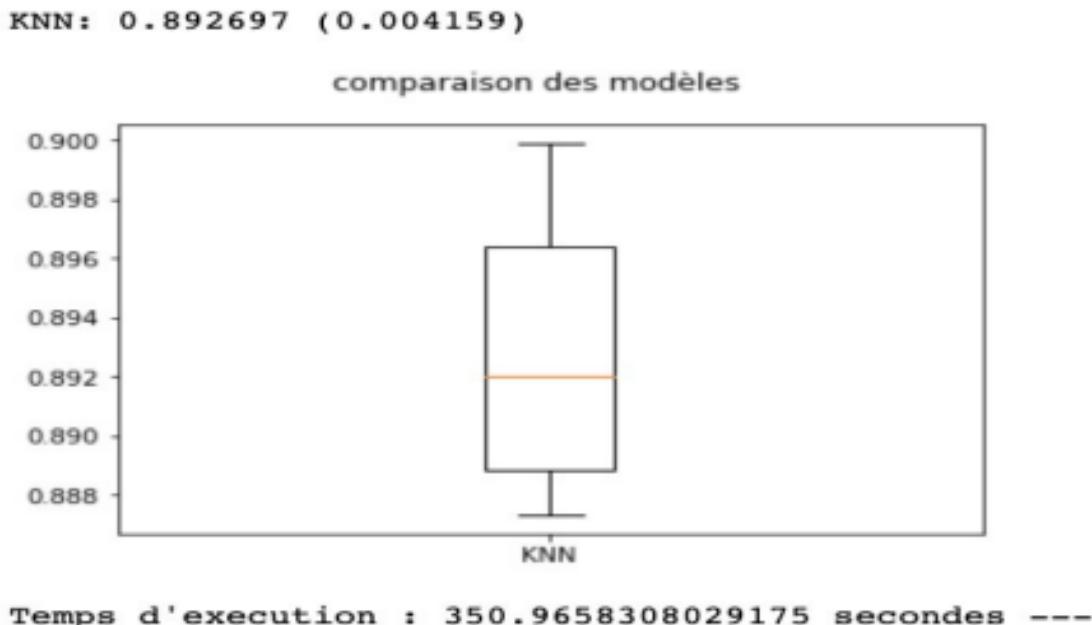


FIGURE 53 – Boxplot pour les scores de knn

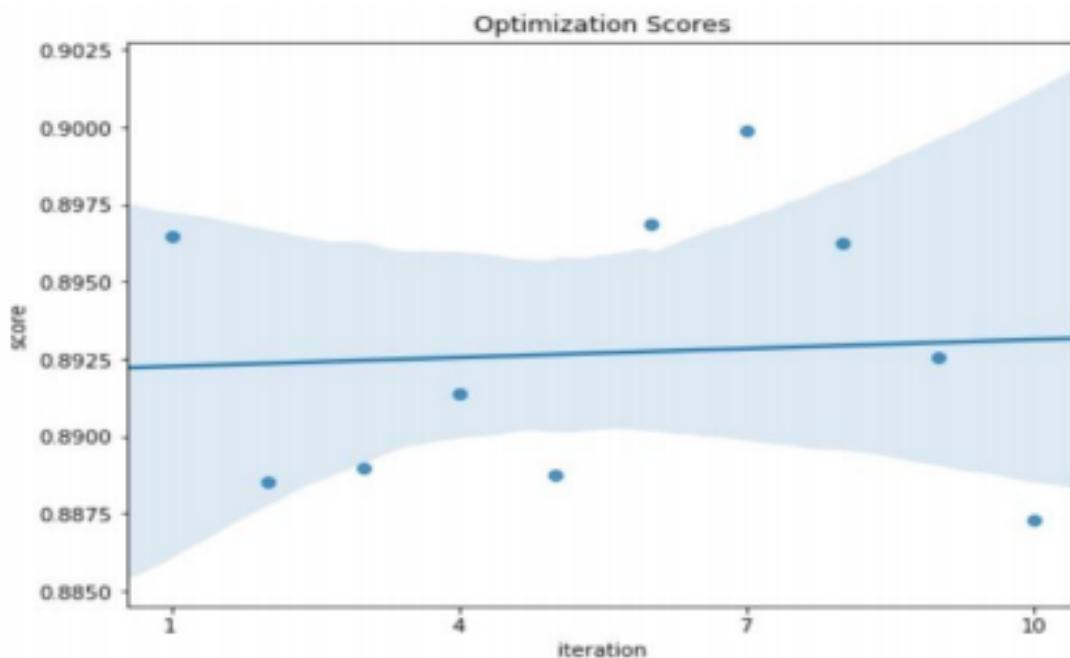


FIGURE 54 – Les différents scores de knn

Nous constatons qu'on a obtenu un bon résultat (résultat moyen 0.89). Évaluation les résultats des 3016 familles :

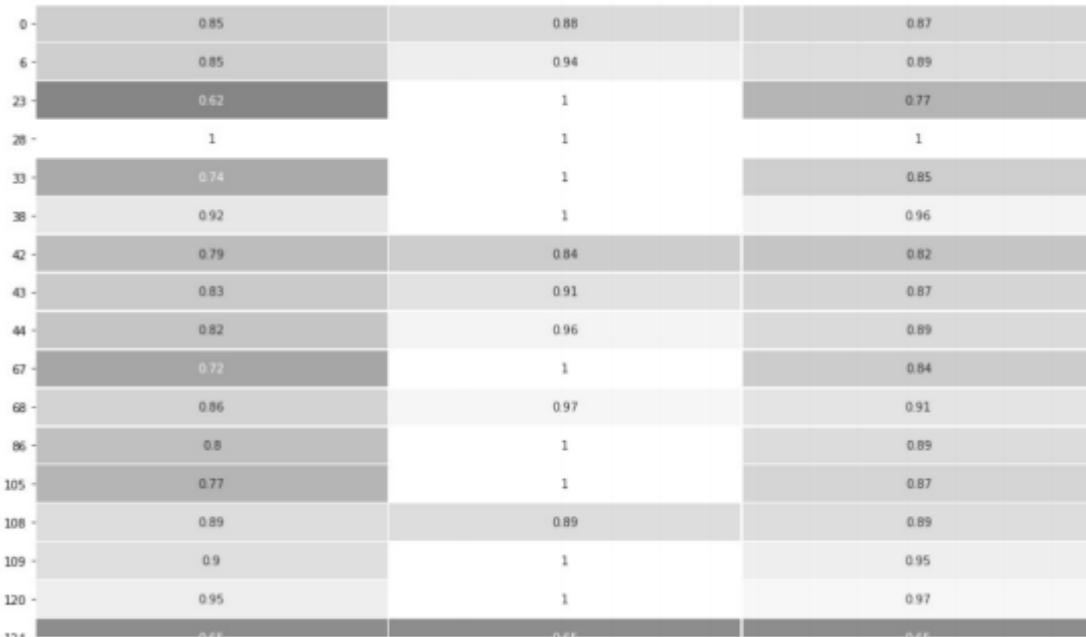


FIGURE 55 – Précision rappel et F1 mesure score

Les résultats sont presque parfaits pour l'ensemble de toutes les familles ainsi le score moyen de F-mesure est égale à 88

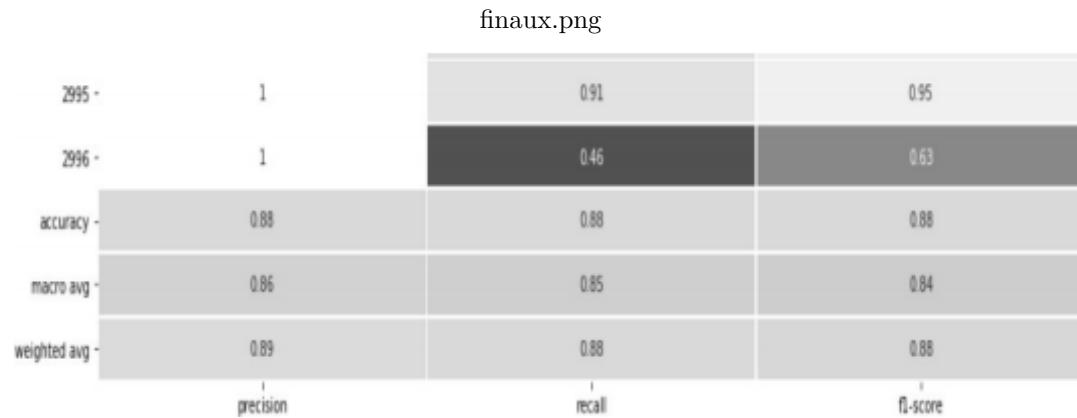


FIGURE 56 – résultats finaux

Nous aurions pu aboutir à de meilleurs résultats, seulement le manque de matériel ainsi que la liée au COVID-19 nous ont empêchés.

### 5.1.11 Continuité et évolution

Pendant la période du test nous avons essayé de faire un apprentissage non supervisé. Nous avons classifié les séquences juste en utilisant les mesures de distances et des modèles non supervisés à savoir k-means. D'après les résultats obtenus, nous constatons que le clustering lui aussi donne un bon résultat.

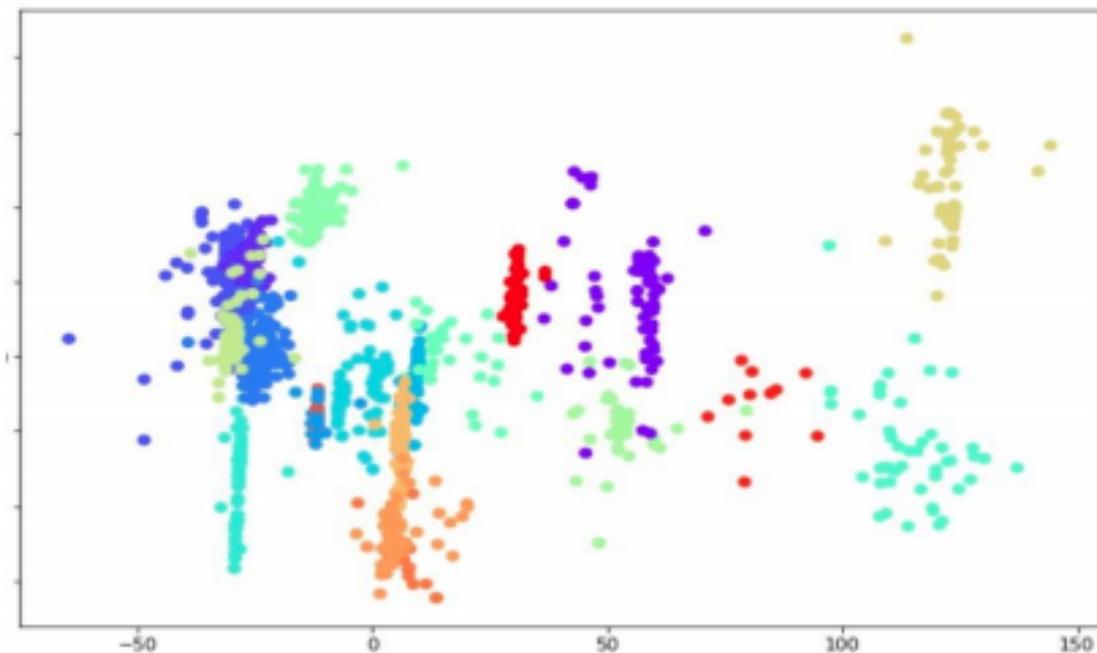


FIGURE 57 – Application de K-min pour la dataset de test

	0	1	2	3	4	5	6	7	8	9	...	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	15	2	0	24	0	22	0
1	0	0	0	0	0	0	0	0	15	0	0	...	25	0	0	0	0	0	0	0	0
2	0	11	0	0	0	4	0	0	0	0	...	0	1	0	0	0	4	0	0	0	2
3	0	0	0	12	0	0	0	2	19	26	...	0	0	0	0	0	0	0	2	1	0
4	0	9	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	25	0	0	0	0	0
6	0	0	0	13	0	0	0	0	1	0	...	0	0	0	7	18	0	0	0	0	0
7	0	0	0	0	0	7	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	14	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	23	0	0	0	0	0	...	0	0	0	0	0	0	0	2	0	0
10	0	0	0	0	0	0	0	0	0	0	...	0	24	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	23
12	15	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
14	0	0	26	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	8	0	0	0	...	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	3	0	0	...	0	0	0	0	2	0	0	24	0	0
17	7	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	...	0	0	28	0	0	0	0	0	0	0
19	5	0	0	0	0	2	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	13	0	0	0	...	0	0	0	0	0	0	0	0	0	0

FIGURE 58 – Matrice de contingence

Alors puisque nos données sont étiquetées, nous n'avons pas besoin du clustering pour classifier ces séquences. Cependant nous pouvons l'exploiter dans la sélection des motifs caractérisant chaque famille. Puisque nous avons généré les motifs importants pour classifier les séquences, il est possible que chaque famille ait un groupe de motifs qui la caractérise. Donc avec le clustering à haute dimension cela est possible. Cette détection de motifs ce fait en se basant sur les régions denses de la base de données

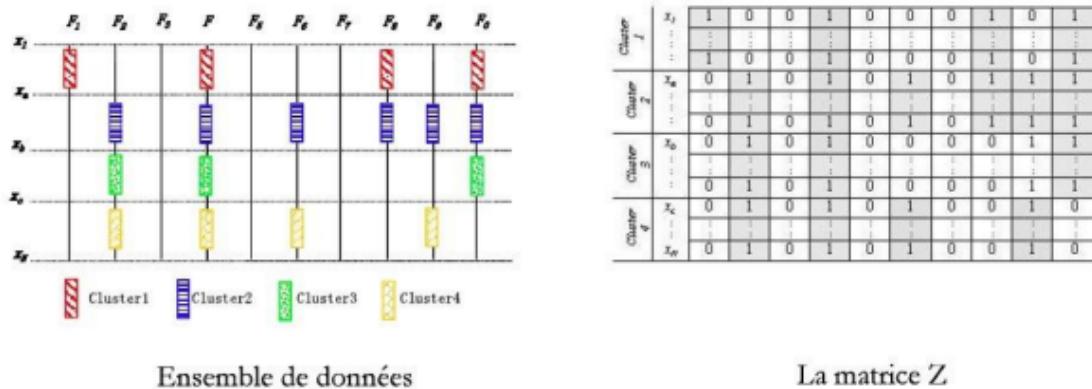


FIGURE 59 – Régions denses

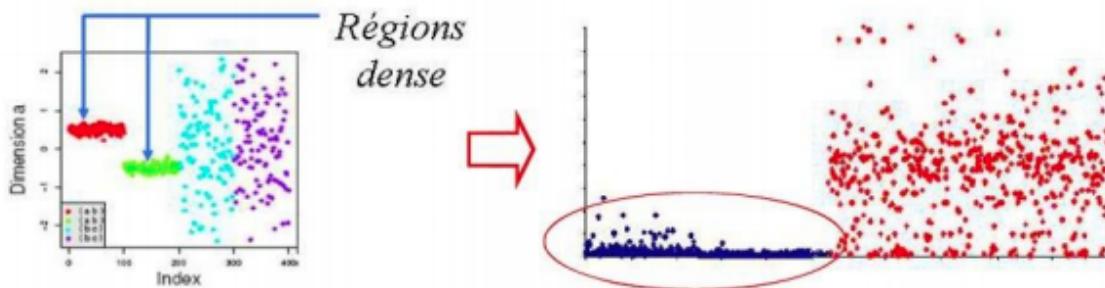


FIGURE 60 – Régions denses 2

Le modèle nous permet aussi de détecter les séquences rares c'est à dire celles qui ne contribuent pas à la formation d'une famille (cluster).

FIGURE 61 – Détection des valeurs aberrantes

Cette approche peut être réalisée en utilisant seulement les motifs de grandes tailles (de 4 caractères ou plus) comme elle nous donne la valeur 1 pour chaque apparition des motifs dans une séquence (non la fréquence).

Cette méthode est appelé pcka. Puisqu'elle ne fait pas l'objet de notre projet on n'en parlera pas.

## 5.2 Méthode2 : Classification basée sur LSTM

### 5.2.1 Un peu sur LSTM

LSTM pour Long Short-Term Memory, fait partie des architectures de réseaux de neurones récurrents [2]. La particularité de ce modèle est qu'il comporte des connections neuronales résiduelles, contrairement à un réseau de neurones classique. Ce type de réseaux est approprié pour la classification et la prédiction de données séquentielles et chronologiques de grande taille. L'un des problèmes des réseaux de neurones classiques est la disparition du gradient lors de la rétropropagation durant la phase de l'entraînement du réseau, chose qui rend l'apprentissage des données séquentielles très lent et des fois impossibles. Un autre problème des réseaux classiques, et qui est le plus important pour nous, est l'incapacité à apprendre des dépendances entre les données au long terme, comme par exemple l'utilisation des images précédentes pour pouvoir comprendre l'image présente dans une vidéo. Ci-dessous une figure montrant l'architecture générale d'un réseau LSTM

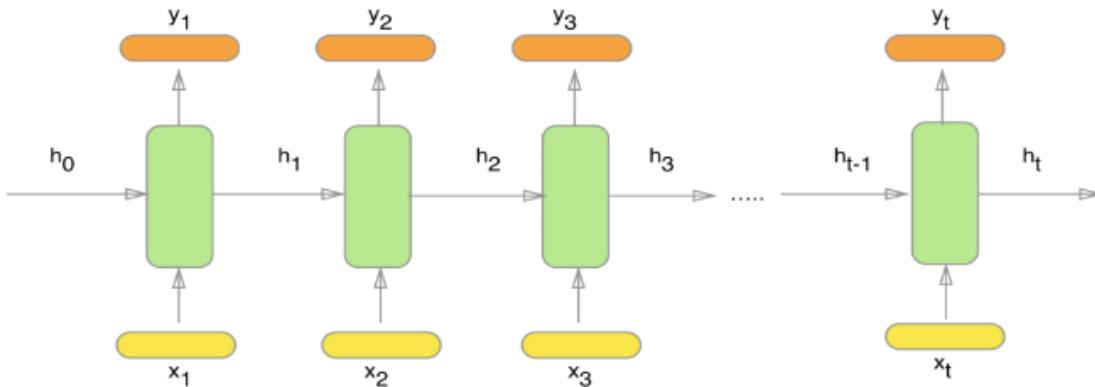


FIGURE 62 – Fonctionnement d'un LSTM

Dans cette figure, les boîtes en vert est-ce que l'on appelle les cellule LSTM. Ces cellules se composent d'un réseau de neurones classique qui prend en entrée un vecteur  $x_i$  et un autre  $h_{i-1}$  et remet en sortie deux vecteurs  $y_i$  et  $h_i$ . Les vecteurs  $x_i$  ici représentent la séquence de données en entrée, les vecteurs  $y_i$  sont les prédictions de chaque cellule, et les vecteurs  $h_i$  est-ce que l'on appelle les états cachés. Ces états cachés est ce qui fait la force du modèle LSTM, car ils permettent le transfert de l'information entre les cellules. Ainsi, lorsqu'on est rendu à la nième cellule pour faire le calcule avec la donnée  $x_n$  on a une information sur toutes les données qui précédent dans la séquence. Il faut noter que les paramètres utilisés dans chaque cellule sont les mêmes. Donc lorsqu'on fait la rétro-propagation, on affecte les mêmes paramètres dans chaque cellule. Ce qui fait qu'on peut dessiner cette architecture sous forme d'une boucle de la manière suivante :

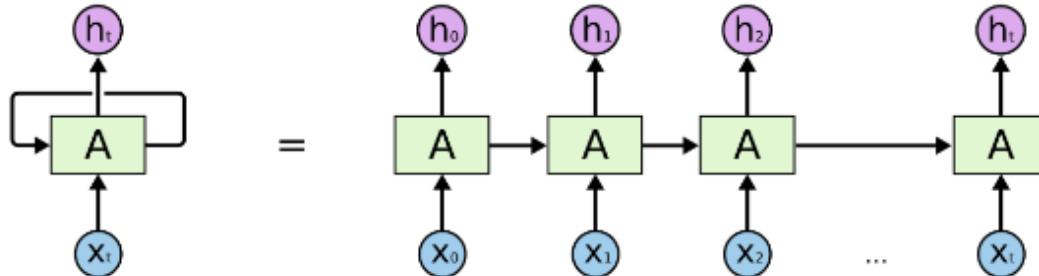


FIGURE 63 – Architectures Équivalentes LSTM

### 5.2.2 Concept de la méthode

Dans cette méthode, on va utiliser l'architecture LSTM pour faire la classification des séquences d'ARN. On a pris ce chemin car nos données sont justement des données séquentielles, et que la prédiction de la famille d'une séquence se fait en regardant la totalité des caractères de la séquence ainsi que les positions de ces caractères. Ainsi, les modèles implémentant cette méthode seront des boîtes basées sur l'architecture LSTM auxquelles on va donner en entrée une séquence ARN, et on aura en sortie la prédiction de la famille d'une telle séquence.

Pour cela, on va utiliser les séquences originales de la base de données. Dès lors, notre base de données sera sous forme d'un vecteur "X" qui regroupe les séquences ARN, donc  $X[i]$  est une chaîne de caractère, et un autre vecteur  $y$  qui regroupe les familles de ces séquences. Toutefois, on doit faire une manipulation sur les données du vecteur X en les rendant sous l'écriture one-hot pour qu'elles soient accessibles à nos modèles. Ainsi, notre manipulation est comme suit :

- On prend une séquence ARN (une chaîne de caractères) soit  $X[i] = 'AAGAU...C'$ .
- Ensuite, on transforme chaque caractère de la séquence sous sa forme one-hot en faisant la correspondance

'A' -> [1, 0, 0, 0]  
 'C' -> [0, 1, 0, 0]  
 'G' -> [0, 0, 1, 0]  
 'U' -> [0, 0, 0, 1]

- Pour que finalement  $X[i]$  se transforme en un vecteur comme suit :

$X[i] = [[1, 0, 0, 0], [1, 0, 0, 0], [0, 0, 1, 0], [1, 0, 0, 0], [0, 0, 0, 1], \dots, [0, 1, 0, 0]]$

En outre les familles des séquences ARN seront encodées aussi en encodage one-hot, donc on aura un vecteur d'étiquettes "y" de 3016 dimensions représentant la vérité terrain où la seule case de ce vecteur qui aura 1 comme valeur est celle correspondante à la famille de la séquence ARN donnée.

Après cette transformation, les données sont prêtes pour l'apprentissage et la prédiction par les modèles.

Les modèles de cette méthode seront implémentés à l'aide de la librairie Keras [3], et ce, parce que cette dernière permet une abstraction des détails non utiles et se focalise plutôt sur le fonctionnement et l'architecture des modèles, et donc une implémentation plus aisée que les autres librairies.

### 5.2.3 Modèles implémentés

#### 1. LSTM simple :

(a) Description du modèle : L'architecture de ce modèle est représentée comme suit :

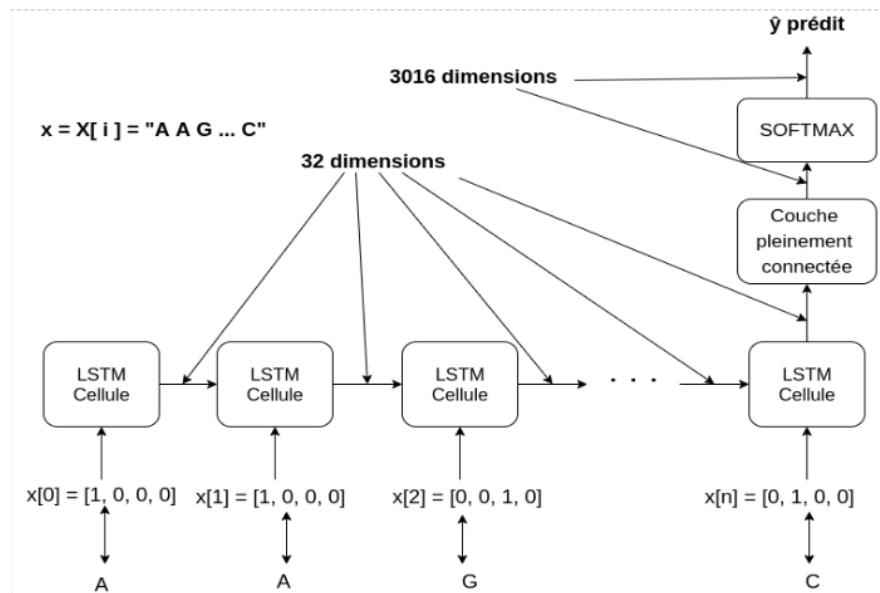


FIGURE 64 – architecture du modèle LSTM simple

Ce modèle se compose d'un modèle LSTM qui prend la séquence ARN en entrée transformée en encodage one-hot. Les états internes sont des vecteurs de 32 dimensions.

Il faut noter que dans ce modèle on ne prend pas en considération les sorties  $y_i$  des cellules sauf la dernière, et ce, parce qu'on veut savoir si rien qu'avec l'information transmise par le biais des états internes on va arriver à bien représenter le problème et donc avoir un bon modèle.

À la fin du bloc LSTM, on prend la sortie de la dernière cellule et on la fait entrée à une couche pleinement connectée qui prend un vecteur de taille 32 dimensions et retourne en sortie un vecteur de taille 3016 dimension sur lequel on va appliquer une SOFTMAX pour donner en sortie un vecteur correspondant à la prédiction des probabilités de la famille de la séquence ARN en entrée.

Ci-dessous le paramétrage de ce modèle sous la librairie Keras.

```

model = Sequential()

model.add(LSTM(32, return_sequences=False, input_shape=(None, 4)))
model.add(Dense(3016, name='out_layer', activation='softmax'))

```

```

↳ Model: "sequential_2"

```

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 32)	4736
out_layer (Dense)	(None, 3016)	99528

```

Total params: 104,264
Trainable params: 104,264
Non-trainable params: 0

```

---

FIGURE 65 – Paramétrage du modèle LSTM simple avec Keras

(b) Entraînement du modèle :

L’entraînement de ce modèle est basé sur l’optimisation de la fonction de perte qui est l’entropie croisée entre les prédictions  $\hat{y}$  faites par le modèle et les étiquettes du vecteur  $y$  représentant les vérités terrain [4].

De plus, ce modèle s’entraîne à base de l’algorithme de descente stochastique du gradient. En effet, l’entraînement se fait en prenant chaque séquence ARN de la base de données de l’entraînement, on calcule le gradient par rapport à cette donnée, puis on met à jour les paramètres du modèle par une rétro-propagation. Cette méthodologie était choisie pour ce modèle parce qu’on veut donner à chaque séquence plus de poids durant l’entraînement de notre modèle, contrairement à ce qu’on fait dans un entraînement par mini-batch où on prend un sous ensemble de données d’entraînement et on calcule le gradient par rapport à chacune de ces données, pour qu’après faire une seule mise à jour des paramètres du modèle par rapport à la somme des gradients des données de cet sous ensemble.

2. **LSTM avec bloc d’attention :**(a) Description du modèle :

L’architecture de ce modèle est la suivante :

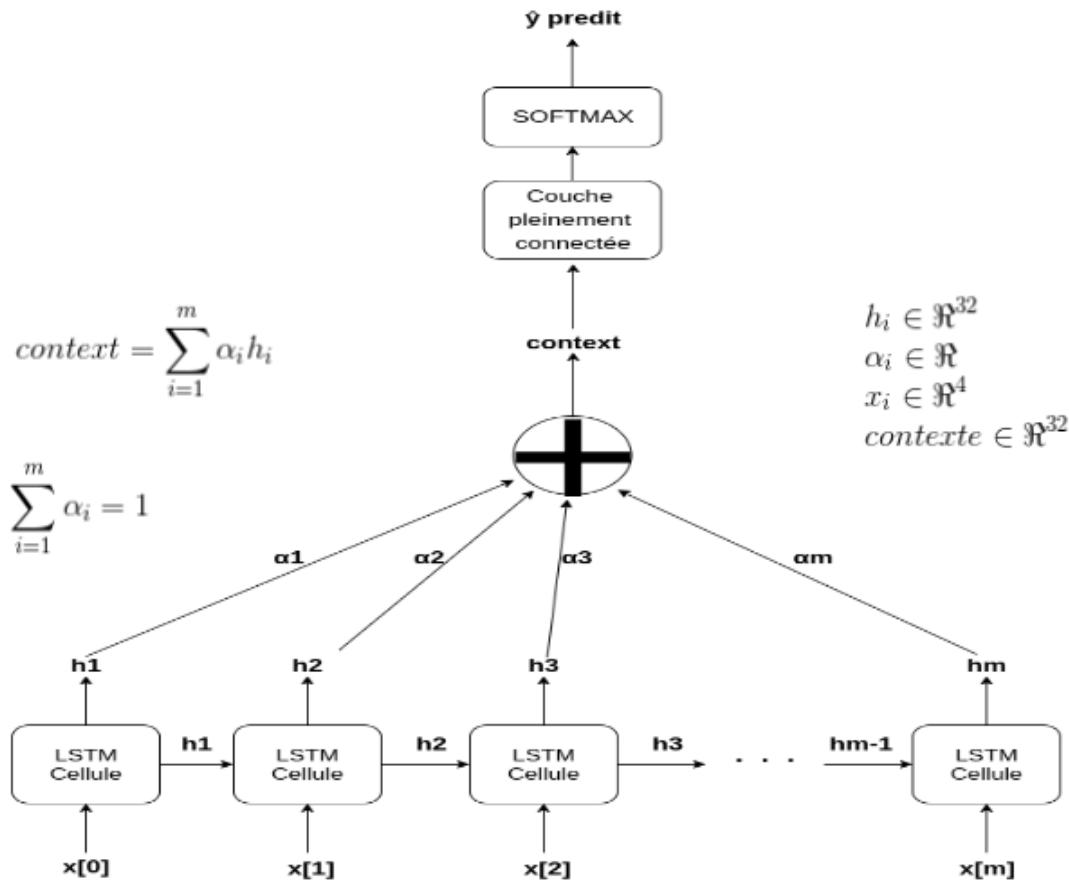


FIGURE 66 – Architecture du modèle LSTM avec bloc d'attention

Dans ce modèle on implémente ce que l'on appelle un bloc d'attention. Contrairement à ce qu'on a vu dans le modèle précédent, ici on n'ignore pas les sorties de cellules, donc chaque cellule contribue par deux façons à la prédiction de la famille d'une séquence en entrée :

- la première c'est qu'on fait propager l'information par le biais des états internes
- la deuxième c'est qu'on utilise directement les sorties des cellules dans la prédiction

Il faut noter que pour ce modèle les sorties des cellules sont les même que les états interne ( $h_i = y_i$ ).

La prédiction d'une famille se fait en multipliant chacune des sorties des cellules  $h_i$  par un terme  $i$ , qui désigne l'influence que cette sortie a sur la prédiction, pour calculer le vecteur de contexte sur lequel on se base pour faire la prédiction de la famille de la séquence ARN en entrée.

Ainsi, le travail du bloc attentionnel est de bien déterminer les  $i$  pour optimiser notre fonction de perte d'une façon plus robuste [5]. Ceci est fait par le biais d'une autre couche de neurones pleinement connectée comme suit :

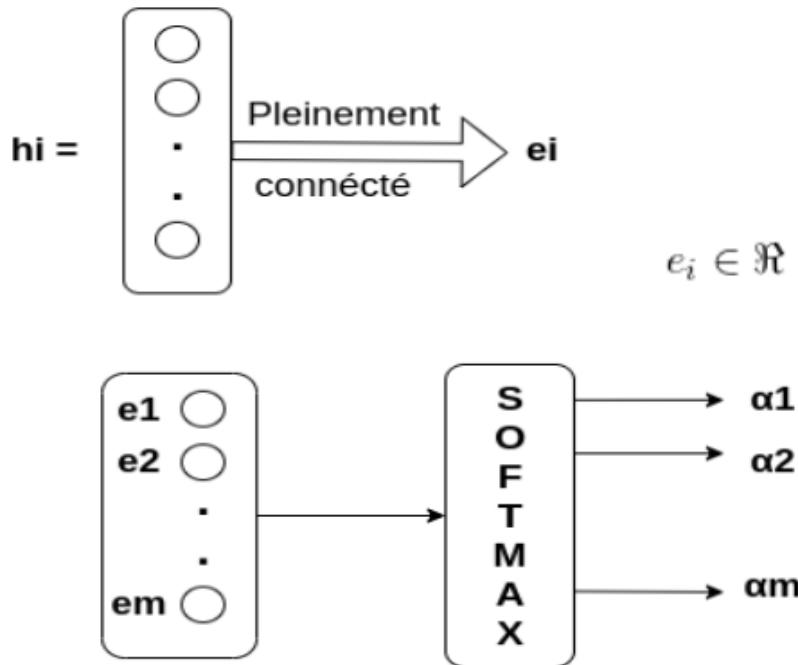


FIGURE 67 – bloc attentionnel

La figure en haut, montre la façon avec laquelle les  $i$  sont calculés. On prend toutes les sorties  $h_i$  des cellules, pour chacune de ces sorties on calcule un terme  $e_i$  à l'aide d'une couche pleinement connectée qui prend en entrée un vecteur de dimensions  $m$  ( $h_i$ ) et retourne en sortie un scalaire ( $e_i$ ), pour que par la suite on prend le vecteur formé par les termes  $e_i$  résultants et appliquer une softmax sur ce vecteur pour avoir un vecteur normalisé qui comporte les termes  $i$  recherchées. Avec cette, manière les  $i$  seront optimisés à chaque rétro-propagation. On espère donc qu'à la fin de l'entraînement, les paramètres de la couche pleinement connectée serons optimisés pour bien générer des  $i$  étant les sorties des cellules  $h_i$ , et ce, d'une façon qui rend la prédiction plus robuste et efficace.

Ci-dessous le paramétrage de ce modèle sous la librairie Keras :

```

inputs = Input(shape=(max_sequence, 4), name='input_layer')
encoder_out = LSTM(32, return_sequences=True, name='LSTM_layer')(inputs)

alpha = Dense(1, name='Attention_layer')(encoder_out)
alpha = Reshape(target_shape=(1, max_sequence))(alpha)
alpha = Activation('softmax', name='Activation_layer')(alpha)
alpha = Reshape(target_shape=(max_sequence, 1))(alpha)

decoder_in = Multiply(name='Multiply_layer')([encoder_out, alpha])
decoder_in = Lambda(lambda x: K.sum(x, axis=1), name='Sum_layer')(decoder_in) #Weighted sum

decoder_out = Dense(3016, activation='softmax', name='output_layer')(decoder_in)

model = Model(inputs=inputs, outputs=decoder_out)

```

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
input_layer (InputLayer)	(None, 4065, 4)	0	
LSTM_layer (LSTM)	(None, 4065, 32)	4736	input_layer[0][0]
Attention_layer (Dense)	(None, 4065, 1)	33	LSTM_layer[0][0]
reshape_1 (Reshape)	(None, 1, 4065)	0	Attention_layer[0][0]
Activation_layer (Activation)	(None, 1, 4065)	0	reshape_1[0][0]
reshape_2 (Reshape)	(None, 4065, 1)	0	Activation_layer[0][0]
Multiply_layer (Multiply)	(None, 4065, 32)	0	LSTM_layer[0][0] reshape_2[0][0]
Sum_layer (Lambda)	(None, 32)	0	Multiply_layer[0][0]
output_layer (Dense)	(None, 3016)	99528	Sum_layer[0][0]
<hr/>			
Total params:	104,297		
Trainable params:	104,297		
Non-trainable params:	0		

FIGURE 68 – paramétrage du modèle de LSTM avec bloc d'attention

(b) Entraînement du modèle :

Pour ce modèle aussi, l'entraînement est basé sur l'optimisation de l'entropie croisée entre les prédictions  $\hat{y}$  faites par le modèle et les étiquettes du vecteur  $y$ .

Contrairement au premier modèle, ce modèle s'entraîne avec une descente de gradient par mini-batch. Pour ce faire, on est obligé à ce que toutes les séquences de notre base de données X soient de même taille. Par conséquent, on cherche la taille de la plus longue séquence de la base de données m, pour que par la suite appliquer le zero padding sur les données ayant une taille inférieure à la taille maximale m.

Ainsi, l'entraînement sera fait par des sous-ensembles de données de l'entraînement en calculant le gradient par rapport à chacune de ces données, pour qu'après faire une seule mise à jour des paramètres du modèle par rapport à la somme des gradients des données de cet sous ensemble. Cette méthode optimise le temps d'entraînement et donne une courbe d'apprentissage plus robuste.

3. **Résultats expérimentaux :** Ci dessous les résultats des deux modèles décrits ci-haut :

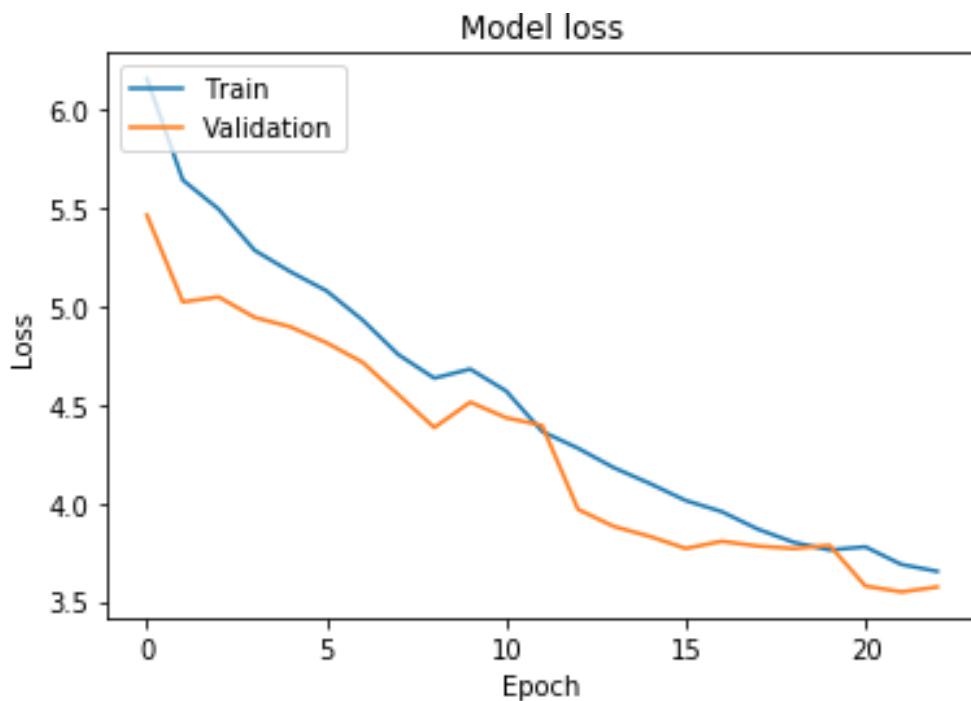


FIGURE 69 – Erreur d’entraînement et de test du modèle LSTM simple

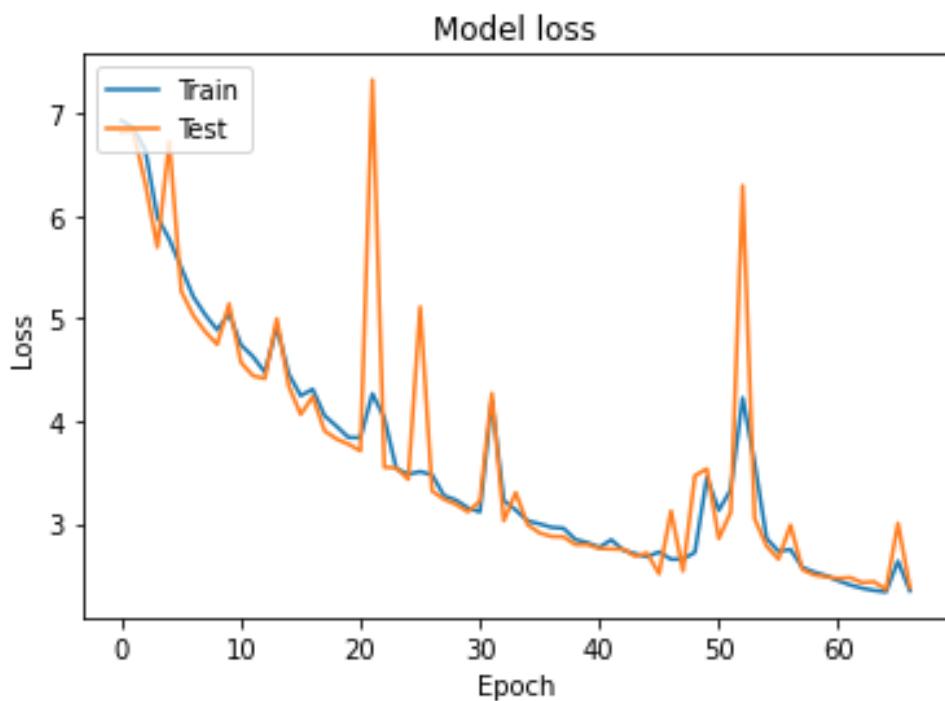


FIGURE 70 – Erreur d’entraînement et de test du modèle LSTM avec bloc d’attention

On remarque que l’erreur du modèle avec attention est inférieur significativement par rapport au modèle LSTM simple (2.3 contre 3.5). C’est d’ailleurs ce qu’on s’attendait vu la force de l’architecture du modèle avec attention. Il faut noter que ce dernier modèle pourrait faire mieux si on lui a entraîner

sur un nombre d'épochs de plus (au moins 100 épochs) ce qui n'est pas fait ici faute de temps.

## 6 Conclusion

Dans ce projet nous avons abordé l'application des méthodes d'apprentissage supervisé ainsi que l'apprentissage profond pour faire la classification des séquences d'ARN.

Les deux méthodes ont été bien abordées et nous avons eu de bons résultats.

Pour la première méthode le but principale était de classifier les séquences d'ARN par comparaison et choix de meilleur modèle et ses paramètres. Or, nous avons pu aller plus loin et faire l'extraction des motifs important pour notre classification.

On a vu que la deuxième méthode ne fait aucune hypothèse sur l'ensemble de données, et donc s'entraîne sur n'importe quelle séquences d'ARN à l'aide du fameux modèle de LSTM.

Finalement, nombreux sont les améliorations qui peuvent avoir lieu pour les deux méthodes, chose qui fera l'objet des futures travaux.

## 7 Références

- [1] Griffiths-Jones S, Bateman A, Marshall M, Khanna A, Eddy SR. "Rfam : an RNA family database". Nucleic Acids Res. 31 (1) : 439–41. doi :10.1093/nar/gkg006. PMC 165453. PMID 12520045, 2003
- [2] Sepp Hochreiter and Jürgen Schmidhuber, “Long Short-Term Memory”, Neural Computation 9(8) :17351780, 1997
- [3] Charles, P.W.D. ”Keras”. Github Repository, 2013
- [4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, “Deep Learning“, MIT Press, 2016
- [5] Abhishek Singh, “Brief Introduction to Attention Models”, Towards Data Science, 2019
- [6] Renu Khandelwal, “K-Nearest Neighbors(KNN)”, Medium, 2018
- [7] Chirag Sehra, “Decision Trees Explained Easily”, Medium, 2018
- [8] Breiman, Leo, “Statistical Modeling : The Two Cultures“, Statistical Science, 2001
- [9] Rohan Joseph, “Grid Search for model tuning“, Towards Data Science, 2018