TP4 : Arbres de décision et réseaux de neurones

Équipe 22
Chollet Lucas (536 799 258)
Deflesselle Camille (536 797 433)

Dans le cadre du cours
Techniques avancées en intelligence artificielle (IFT-7025)

Travail présenté à
M. Lemire Paquin

Département d'Informatique et de Génie Logiciel
Faculté des Sciences et de Génie
Université Laval

Date de remise du rapport
1er mai 2022

Table des matières

Liste des tableaux

Table des figures

1 Introduction

Ce TP4 est une suite de notre TP3. À l'instar du TP3, nous allons aussi implémenter deux techniques d'apprentissage machine :

- Les **arbres de décision** ; et
- Les **réseaux de neurones**.

La difficulté ici sera de programmer entièrement ces deux méthodes sans passer par des bibliothèques faisant directement le travail.

2 Jeux de données et métriques d'évaluation des performances de nos modèles

2.1 Les différents dataset

Nous évaluerons nos deux méthodes d'apprentissage sur trois jeux de données :

1. **Iris dataset**
   Ce jeu de données classe des fleurs en trois différentes classes d'iris en fonction de la taille des différentes caractéristiques. Les attributs utilisés pour la prédiction sont donc des attributs quantitatifs. Notre but sera alors de faire une classification classique. Nous devons prédire pour chaque instance testée, l'une des classes suivantes :
   (a) Iris-setosa (0 en numérique)
   (b) Iris-versicolor (1 en numérique)
   (c) Iris-virginica (2 en numérique)

   Au cours de ce TP, nous convertissons les classes à prédire en variables numériques, pour faciliter l'implémentation du code.
2. **Vinho verde - vin blanc**
   Pour notre TP, ce jeu de données a été réorganisé en deux classes : *bon* et *mauvais*. Ainsi, notre but sera prédire si un vin est plutôt bon ou mauvais étant donné sa composition chimique. Tous les attributs utilisés sont des attributs quantitatifs. Dans ce jeu de données, nous devons prédire la classe d'une instance, parmi les suivantes :
   (a) 0 : Mauvais (qualité ≤ 5)
   (b) 1 : Bon (qualité ≥ 7)
3. **Abalones**
   Pour notre TP, le dataset a été réorganisé de tel sorte à obtenir seulement trois classes qui correspondent à des intervalles du nombre d'anneaux. Nous savons que le nombre d'anneaux est corrélé avec l'âge des ormeaux (abalones), cependant déterminer l'âge de l'oiseau de cette manière est assez exténuant. Donc, notre but sera de prédire leur âge à partir de caractéristiques physiques plus facilement mesurables. Nous utiliserons des attributs quantitatifs et un attribut qualitatif (le sexe des ormeaux). Dans ce jeu de données, nous devons attribuer à une instance l'une des trois classes suivantes, concernant le nombre d'anneaux n_{rings} :
   (a) 0 : $n_{rings} < 8$
   (b) 1 : $n_{rings} \in [8, 14]$
   (c) 2 : $n_{rings} > 14$

2.2 Métriques d'évaluation

En apprentissage, il existe différentes métriques pour évaluer les performances d'un algorithme, tel que :

1. **La précision**
   Mathématiquement, nous avons cette relation concernant la précision :

$$precision = \frac{Vrai\ positif\ (VP)}{Vrai\ positif\ (VP) + Faux\ positif\ (FP)}$$

   Il s'agit alors du nombre de positifs bien prédits (Vrai positif) divisé par l'ensemble des positifs prédits (Vrai positif + Faux positif). Ainsi, la précision permet donc de connaître le nombre de prédictions positifs bien effectuées.
   Nous pouvons alors dire que plus la précision est élevée, plus notre classificateur minimise le nombre de faux positifs.

2. **Le rappel**
   Cette métrique est semblable à la précédente et son expression mathématique est la suivante :

$$rappel = \frac{Vrai\ positif\ (VP)}{Vrai\ positif\ (VP) + Faux\ negatif\ (FN)}$$

   Il s'agit alors du nombre de positifs bien prédit (Vrai positif) divisé par l'ensemble des positifs réels (Vrai positif + Faux positif). Ainsi, le rappel permet donc de savoir le pourcentage de positifs bien prédits par notre modèle.
   Nous pouvons alors dire que plus le rappel est élevé, plus notre classificateur maximise le nombre de vrais positifs et minimise le nombre de faux négatifs. Cependant, il faut être vigilant car si nous obtenons un rappel élevé, cela ne veut pas nécessairement dire que notre classificateur ne se trompe pas.

3. **Le score F1**
   Mathématiquement, nous avons cette relation concernant le score F1 :

$$F_1 = 2 * \frac{precision * rappel}{precision + rappel}$$

$$\Leftrightarrow$$

$$F_1 = \frac{2 * Vrai\ positif\ (VP)}{2 * Vrai\ positif\ (VP) + Faux\ positif\ (FP) + Faux\ negatif\ (FN)}$$

   De cette formule, nous pouvons donc dire que le score F1 compare les positifs bien prédits (Vrai positif) aux erreurs faites par le classificateur (Faux négatif + Faux positif). De plus, nous pouvons observer que cettre métrique combine à la fois la précision et le rappel. Il faut noter que le score F1 est plus utilisé lorsque les données sont déséquilibrées, concernant le nombre d'instances par classe. En effet, celui-ci sera dans ce cas plus intéressant que le taux de bien classés (Accuracy) car il ne prend pas en compte le taux de vrais négatifs. Aussi, ce score donne autant d'importance aux faux positifs et aux faux négatifs.

4. **La matrice de confusion**
   Cette matrice de confusion est une table résumant les résultats de prédiction pour un problème de classification. Elle compare les données réelles pour une classe cible à celles prédites par un modèle. Voici à quoi ressemble une matrice de confusion.

		Valeur Prédite		
		Positif	Négatif	Total
Valeur Réelle	Positif	Nombre de Vrai positif (VP)	Nombre de Faux négatif (FN)	VP + FN
	Négatif	Nombre de Faux positif (FP)	Nombre de Vrai négatif (VN)	FP + VN
	Total	VP + FP	FN + VN	N_{total}

TABLE 1 – Exemple de matrice de confusion.

De cette table 1, nous pouvons directement calculer les différentes métriques que nous avons citées ci-dessus, à l'aide des formules mathématiques présentées.

3 Techniques d'apprentissage automatique

Pour nos deux techniques d'apprentissage et quel que soit le jeu de données, nous avons utilisé 70% des données pour la phase d'entraînement et 30% pour la phase de test. Ainsi, les modèles sont établis sur le jeu d'entraînement puis évalués sur le jeu de test.

3.1 Arbres de décision

3.1.1 Différents résultats AVANT élagage

3.1.1.1 Entraînement sur nos dataset

Dans cette sous-section, pour chacun de nos dataset, nous avons entraîné notre modèle avec les données d'entraînement.
Afin de vérifier si notre algorithme est en train d'apprendre correctement à partir des exemples que nous lui présentons, nous avons tracé la courbe d'apprentissage pour nos trois jeux de données en suivant la méthode de notre manuel. Pour cela, nous avons pris 100 données (des données d'entraînement) générées aléatoirement que nous avons divisés en deux sous-ensembles : un ensemble d'entraînement et un ensemble de test. Nous apprenons une hypothèse h avec le jeu d'entraînement et mesurons sa précision avec le jeu de test. Nous faisons cela en commençant par un ensemble d'entraînement de taille 1 et en augmentant un à chaque fois jusqu'à la taille 99. Pour chaque taille, nous répétons le processus de fractionnement aléatoire 20 fois puis nous prenons la moyenne des résultats des 20 essais dans le but de réduire la variance.

3.1.1.1.1 Iris dataset

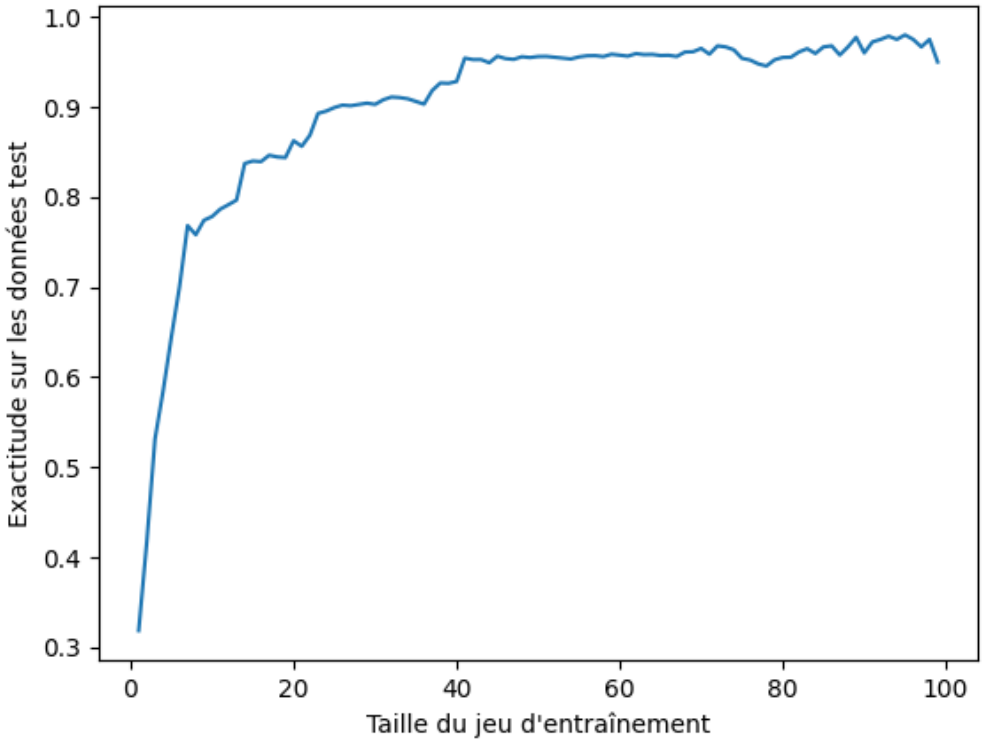

FIGURE 1 – Courbe d'apprentissage de notre arbre de décision sans élagage sur le jeu de données Iris. Exactitude sur le jeu de données test en fonction de la taille du jeu de données en entraînement. Chaque point de données est la moyenne de 20 essais.

Sur cette figure 3, nous pouvons clairement voir que plus la taille de l'ensemble d'entraînement augmente, plus la précision sur les données de test augmente. Dans ce graphique, nous atteignons un taux de bien classés de 95% et nous pouvons remarquer que l'exactitude sur les données de test semble converger avec l'augmentation de la taille du jeu d'entraînement. Nous pouvons donc dire que notre arbre de décision sans élagage apprend correctement sur ce dataset.

3.1.1.1.2 Vinho verde - vin blanc

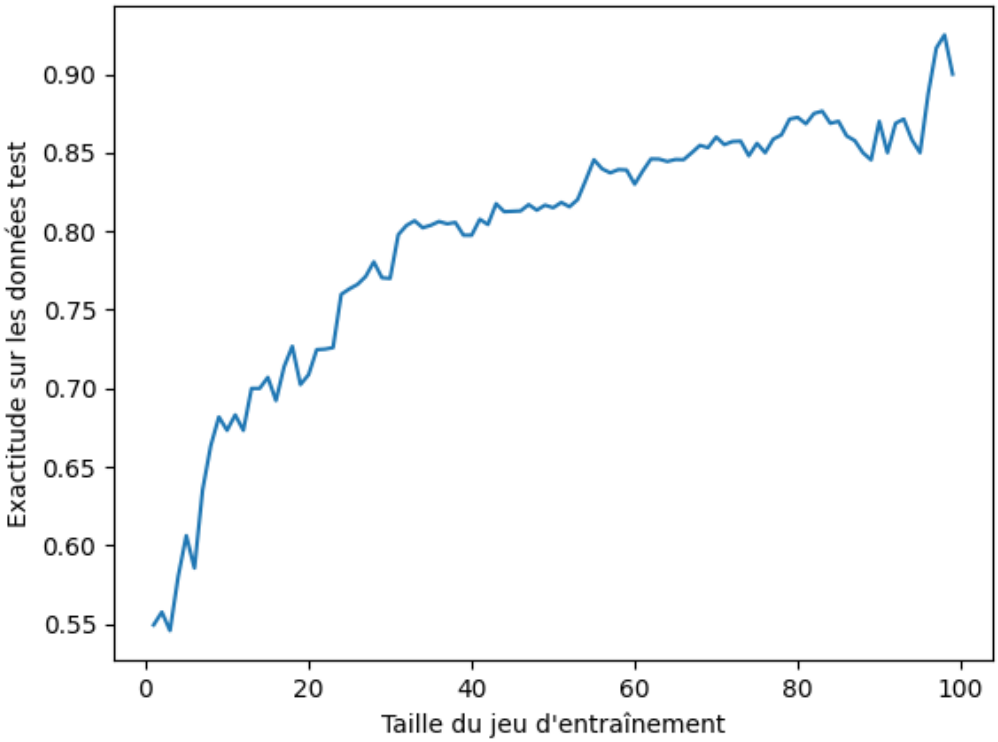

FIGURE 2 – Courbe d'apprentissage de notre arbre de décision sans élagage sur le jeu de données Vinho verde - vin blanc. Exactitude sur le jeu de données test en fonction de la taille du jeu de données en entraînement. Chaque point de données est la moyenne de 20 essais.

Sur cette figure 2, la courbe nous montre que plus la taille du jeu d'entraînement augmente, plus la précision sur les données de test augmente. Dans ce graphique, nous atteignons une précision de 90% et il semble que la courbe pourrait continuer à augmenter avec plus de données. Nous pouvons donc dire que notre arbre de décision sans élagage apprend correctement sur ce dataset.

3.1.1.1.3 Abalones

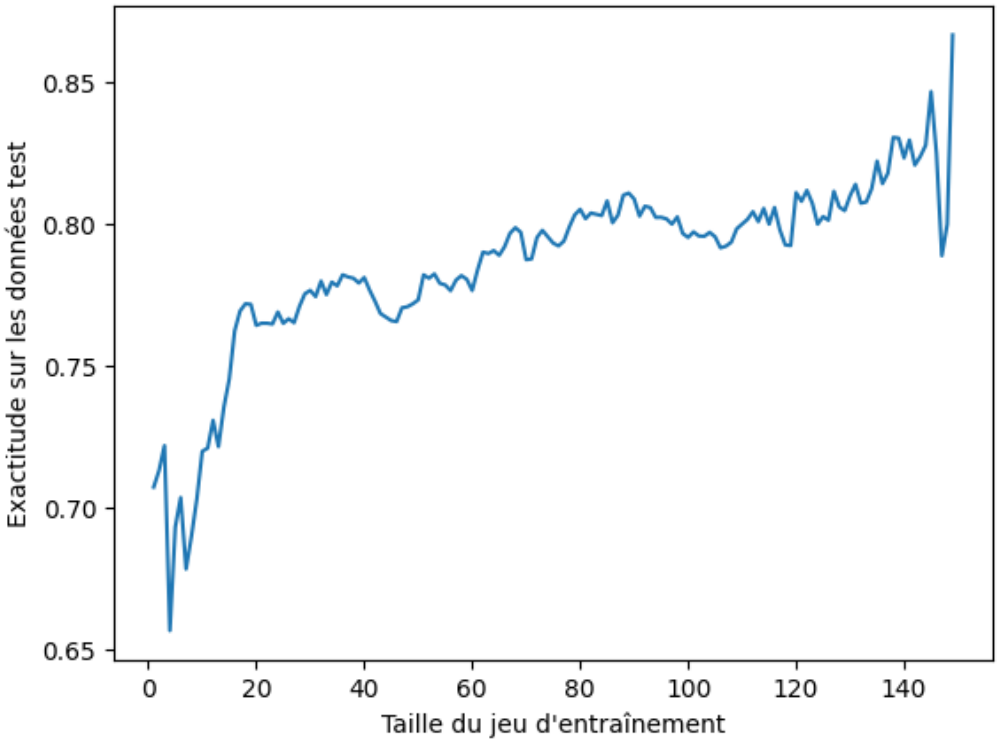

FIGURE 3 – Courbe d'apprentissage de notre arbre de décision sans élagage sur le jeu de données Abalones. Exactitude sur le jeu de données test en fonction de la taille du jeu de données en entraînement. Chaque point de données est la moyenne de 30 essais.

Contrairement aux deux autrès dataset, pour la construction de la courbe d'apprentissage du jeu de données Abalones, nous avons pris 150 données (des données d'entraînement) générées aléatoirement et répété 30 fois l'opération. Puis nous avons pris la moyenne des résultats de ces 30 essais. Nous avons procédé autrement pour pouvoir interpréter plus facilement le graphique.
Sur cette figure 3, nous pouvons clairement voir que plus la taille de l'ensemble d'entraînement augmente, plus la précision sur les données de test augmente. Dans ce graphique, nous atteignons un taux de bien classés de 87% et nous pouvons remarquer que l'exactitude sur les données de test pourrait continuer à augmenter avec plus de données. Nous pouvons donc dire que notre arbre de décision apprend correctement sur ce dataset.

3.1.1.2 Comparaison et évaluation des performances de notre modèle VS la méthode de la bibliothèque `scikit-learn`

3.1.1.2.1 Iris dataset

(a) En considérant la classe "Iris-setosa" comme la classe positive

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	17 (VP)	0 (FN)	17 (VP)	0 (FN)
	0 (FP)	28 (VN)	0 (FP)	28 (FN)
L'exactitude (Accuracy)	1		1	
La précision (Precision)	1		1	
Le rappel (Recall)	1		1	
Le score F1	1		1	

(b) En considérant la classe "Iris-versicolour" comme la classe positive

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	14 (VP)	1 (FN)	13 (VP)	2 (FN)
	4 (FP)	26 (VN)	4 (FP)	26 (FN)
L'exactitude (Accuracy)	0.89		0.87	
La précision (Precision)	0.78		0.76	
Le rappel (Recall)	0.93		0.87	
Le score F1	0.85		0.81	

(c) En considérant la classe "Iris-virginica" comme la classe positive

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	9 (VP)	4 (FN)	9 (VP)	4 (FN)
	1 (FP)	31 (VN)	2 (FP)	30 (FN)
L'exactitude (Accuracy)	0.89		0.87	
La précision (Precision)	0.9		0.82	
Le rappel (Recall)	0.69		0.69	
Le score F1	0.78		0.75	

TABLE 2 – Résumé des résultats obtenus par classe avec notre modèle (arbre de décision) et avec la méthode de la bibliothèque `scikit-learn` sur Iris dataset.

Sur cette table 2, nous pouvons noter que notre modèle classe parfaitement les fleurs "Iris-setosa" (cf : table (a)). En effet, nous avons un résultat de 100% quelle que soit la métrique. Nous pouvons aussi dire, pour cette espèce d'iris, que notre modèle est aussi performant qu'avec la bibliothèque `scikit-learn`.
Ensuite, si nous regardons la table (b), nous constatons que le taux de bien classés et le score F1 sont élevés, respectivement ils sont de 89% et 85%. Nous avons un rappel très élevé (0.93) et une précision élevé (0.78). Ceci veut dire que cette espèce est correctement gérée par notre modèle. De

plus, pour ce spécimen d'iris, nous remarquons qu'avec la bibliothèque `scikit-learn` les résultats sont un peu moins performants qu'avec notre modèle. En effet, le taux de bien classés passe de 89% à 87% et le score F1, qui était à 85%, est passé à 81%.
Pour finir, analysons la table (c). Pour notre modèle, nous constatons que le taux de bien classés est de 89% et que le score F1 associé est de 78%. Le rappel est assez élevé (0.69) et la précision est très élevé (0.9), nous pouvons dire que notre modèle gère correctement cette espèce d'iris. Comme pour l'espèce "Iris-versicolour", nous remarquons que les résultats obtenus avec notre modèle sont légèrement supérieurs à ceux issus de la bibliothèque `scikit-learn`.

De manière générale, nous pouvons dire que notre modèle est efficace pour ce jeu de données, et que celui-ci est légèrement supérieur à la bibliothèque `scikit-learn` en terme de performance.

3.1.1.2.2 Vinho verde - vin blanc

(a) En considérant la classe "0" (mauvais : qualité ≤ 5) comme la classe positive

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	428 (VP)	67 (FN)	427 (VP)	68 (FN)
	50 (FP)	266 (VN)	43 (FP)	273 (FN)
L'exactitude (Accuracy)	0.86		0.86	
La précision (Precision)	0.90		0.91	
Le rappel (Recall)	0.86		0.86	
Le score F1	0.88		0.88	

(b) En considérant la classe "1" (bon : qualité ≥ 7) comme la classe positive

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	266 (VP)	50 (FN)	273 (VP)	43 (FN)
	67 (FP)	428 (VN)	68 (FP)	427 (FN)
L'exactitude (Accuracy)	0.86		0.86	
La précision (Precision)	0.80		0.80	
Le rappel (Recall)	0.84		0.86	
Le score F1	0.82		0.83	

TABLE 3 – Résumé des résultats obtenus avec notre modèle (arbre de décision) et avec la méthode de la bibliothèque `scikit-learn` sur Vinho verde - vin blanc.

Sur cette table 3, nous pouvons noter que le taux de bien classés, avec notre modèle, est identique pour les deux classes (86%). Cependant, si nous nous attardons sur les autrès métriques, nous constatons que notre modèle est légèrement plus performant pour la classe "0" (qualité ≤ 5) que pour la classe "1" (qualité ≥ 7). En effet, par exemple pour le score F1, pour la classe "0", nous avons 0.88 tandis que pour la classe "1", nous avons 0.82. Pour ces deux classes, nous avons des précisions et

des rappels qui sont assez élevés. Ceci veut dire que notre modèle gère correctement ces deux classes. Maintenant, si nous regardons les résultats obtenus avec la méthode de la bibliothèque `scikit-learn`, nous notons que ceux-ci sont quasiment équivalents à ceux de notre modèle.

De manière générale, nous pouvons dire que notre modèle est efficace pour ce jeu de données, et qu'il fait aussi bien qu'avec la méthode de la bibliothèque `scikit-learn`.

3.1.1.2.3 Abalones

(a) En considérant la classe "0" ($n_{rings} < 8$) comme la classe positive

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	79 (VP)	54 (FN)	75 (VP)	58 (FN)
	55 (FP)	1066 (VN)	59 (FP)	1062 (FN)
L'exactitude (Accuracy)	0.91		0.91	
La précision (Precision)	0.59		0.56	
Le rappel (Recall)	0.59		0.56	
Le score F1	0.59		0.56	

(b) En considérant la classe "1" ($n_{rings} \in [8, 14]$) comme la classe positive

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	819 (VP)	151 (FN)	815 (VP)	155 (FN)
	137 (FP)	147 (VN)	151 (FP)	133 (FN)
L'exactitude (Accuracy)	0.77		0.76	
La précision (Precision)	0.86		0.84	
Le rappel (Recall)	0.84		0.84	
Le score F1	0.85		0.84	

(c) En considérant la classe "2" ($n_{rings} > 14$) comme la classe positive

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	66 (VP)	85 (FN)	57 (VP)	94 (FN)
	98 (FP)	1005 (VN)	97 (FP)	1006 (FN)
L'exactitude (Accuracy)	0.85		0.85	
La précision (Precision)	0.40		0.37	
Le rappel (Recall)	0.44		0.38	
Le score F1	0.42		0.37	

TABLE 4 – Résumé des résultats obtenus avec notre modèle (arbre de décision) et avec la méthode de la bibliothèque `scikit-learn` sur les abalones.

Sur cette table 4, lorsque nous regardons les résultats, nous remarquons que ceux-ci sont plutôt hétérogènes entre les différentes classes. Ainsi, il serait plus judicieux d'analyser tableau par tableau. Tout d'abord, commençons par la table (a). Nous avons 91% de bien classés avec notre modèle. Cependant, il y a une très grande différence entre cette mesure et le score F1 associé qui est de 0.58. Cette différence vient du fait que pour le score F1, le nombre de vrais négatifs n'est pas pris en compte et, ici, nous voyons que celui-ci est très élevé si nous considérons la classe "0" comme la classe positive. Même s'il est plus faible que l'exactitude, il est tout de même supérieur à 0.5 ce qui

reste raisonnable. Nous avons un rappel (0.58) et une précision (0.58) qui sont légèrement supérieur au seuil de 0.5, nous pouvons donc dire que ces deux grandeurs sont assez élevées, ce qui signifie que notre modèle gère assez bien cette classe où le nombre d'anneaux est inférieur à 8. Maintenant, si nous regardons les performances obtenues avec la bibliothèque `scikit-learn`, nous constatons que celles-ci sont quasiment équivalentes à celles de notre modèle.
Ensuite, si nous regardons la table (b) où nous avons considéré la classe "1" comme la classe positive, nous remarquons que nous avons une exactitude de 77% et un score F1 associé de 85%. Nous avons également une précision et un rappel qui sont élevés. Ces résultats nous montrent que notre algorithme gère correctement cette classe où le nombre d'anneaux est compris entre 8 et 14 inclus. Si nous regardons les résultats que nous avons obtenus avec la méthode avec la bibliothèque `scikit-learn`, nous observons qu'ils sont très proches de ceux issus de notre modèle. Cependant, nous pouvons quand même noter que notre modèle est très légèrement supérieur à cette méthode.
Pour finir, analysons la table (c) où nous avons considéré la classe "2" comme la classe positive. Pour notre modèle, nous constatons que le taux de bien classés est de 85% et que le score F1 associé est très différent. Contrairement à la table (a), ici, le score F1 est inférieur au seuil de 0.5. Ce qui nous montre que notre modèle n'est pas très adapté pour cette classe sur ce jeu de données. De plus, le rappel (0.44) et la précision (0.40) qui sont inférieurs au seuil de 0.5 et que nous pouvons considérer comme plutôt faibles, nous montre que cette classe est mal géré par notre modèle, car il n'est généralement pas extrait. Cependant, si elle est extraite, elle l'est avec un niveau de confiance faible. À présent, si nous considérons les résultats issus de la méthode avec la bibliothèque `scikit-learn`, nous voyons qu'ils sont quasiment identiques à ceux de notre modèle. Néanmoins, nous pouvons noter que nos résulats sont légèrement supérieurs à cette méthode.

De manière générale, nous pouvons dire que notre modèle est plutôt efficace pour ce jeu de données, et qu'il fait légèrement mieux qu'avec la méthode de la bibliothèque `scikit-learn`.

3.1.1.3 Exemple de visualisation d'un arbre décision sans élagage

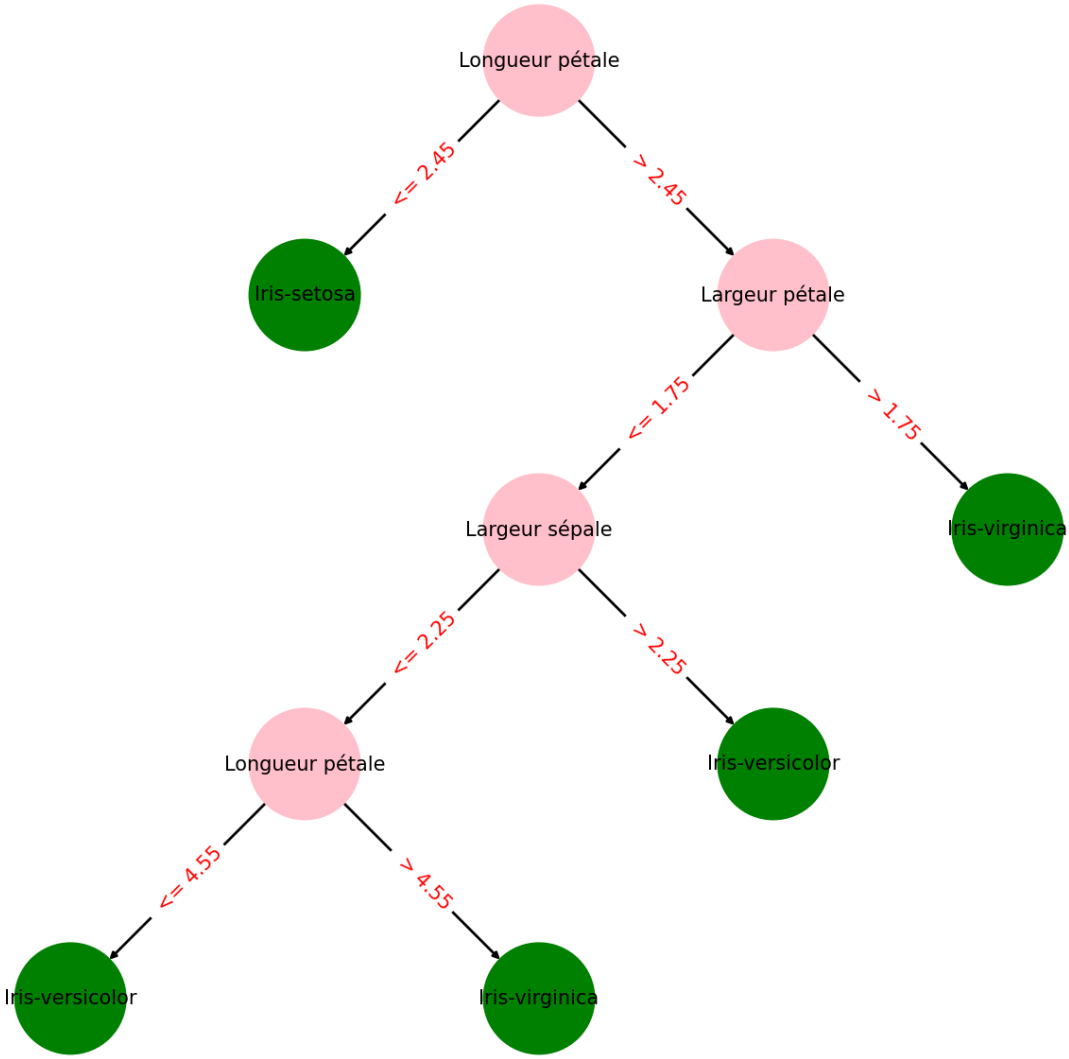

FIGURE 4 – Arbre de décision avec Iris dataset.

Sur cette figure 4, nous avons tracé l'arbre de décision que nous avons obtenu avec Iris dataset sans élagage. Ce que nous voyons en vert, ce sont les noeuds terminaux (les feuilles), ils indiquent un résultat final. Nous en dénombrons cinq. À l'inverse, les noeuds en rose sont des noeuds de décision, ils indiquent une décision à prendre (quatre au total). Enfin, nous avons, entre chaque noeud, les branches alternatives (les arêtes) qui signalent un résultat possible. Le choix des valeurs de ces branches s'est fait de tel sorte que l'échantillon soit séparé en deux en maximisant le gain d'information obtenu. Ainsi, la scission résultante nous amène à avoir beaucoup d'une classe et peu de l'autre. Idéalement, nous voudrions que nos noeuds n'aient pas d'entropie, c'est-à-dire que tous les exemples à ce noeud appartiennent définitivement à une classe.
Nous pouvons remarquer que la racine de l'arbre est l'attribut "Longueur du pétale" et que sa profondeur est de quatre.

3.1.2 Différents résultats APRÈS élagage

Précédemment, nos différents résultats étaient issus d'un arbre avant l'étape d'élagage. Maintenant, nous allons présenter les résultats que nous obtenons après élagage des différents arbres de décision qui ont été générés par notre algorithme.
L'étape d'élagage permet d'éviter le sur-apprentissage. Il existe deux types d'élagage :

1. Le **pré-élagage** : utiliser des critères d'arrêt de la division. Par exemmple : nombre minimum des échantillons dans un noeud, un taux d'homogénéité d'un sous-ensemble.
2. Le **post-élagage** : construire l'arbre, ensuite éliminer les branches qui n'améliorent pas la performance de l'arbre.

Dans notre cas, nous allons choisir d'élaguer nos arbres avec la seconde méthode. Plus précisément, nous utiliserons l'élagage par test de χ^2.
À savoir, nous partons des feuilles de notre arbre. Si un noeud parent a deux noeuds terminaux (des feuilles), nous calculons une valeur K determinée par la formule suivante :

$$K = \sum_{i,j} \frac{(N_{i,j} - N'_{i,j})^2}{N'_{i,j}}$$

avec $N_{i,j}$ le nombre d'entités de la classe i dans l'enfant j.
$N'_{i,j}$ le nombre d'entités de la classe i dans l'enfant j en supposant une sélection aléatoire que nous pouvons calculer comme suit : $N'_{i,j} = N_i \times P_j$ où N_i est le nombre de classe i dans le noeud de décision et P_j est la proportion des données allant dans le noeud enfant j.

Après le calcul de la valeur K, nous la comparons à un seuil t qui est mesuré à partir d'une distribution χ^2. Pour obtenir ce seuil, nous avons besoin du degrés de liberté que nous avons obtenu comme ceci : $dl = n - 1$ où n est le nombre d'instances au niveau du noeud et d'une confiance égale à $1 - \alpha$. Ici, nous avons pris une confiance à 95% donc un $\alpha = 0.05$. Ensuite, si K est inférieure au seuil t, nous éliminons tous les enfants (les feuilles) du noeud parent et ledit noeud devient un noeud terminal (une feuille) prenant comme valeur la classe i majoritaire des noeuds enfants j. Nous répétons ce processus jusqu'à ce qu'il n'y ait plus de noeuds enfants à éliminer.

À titre d'exemple, voici ce que nous obtenons lorsque nous élaguons l'arbre de décision que nous avons obtenu à la section précédente.

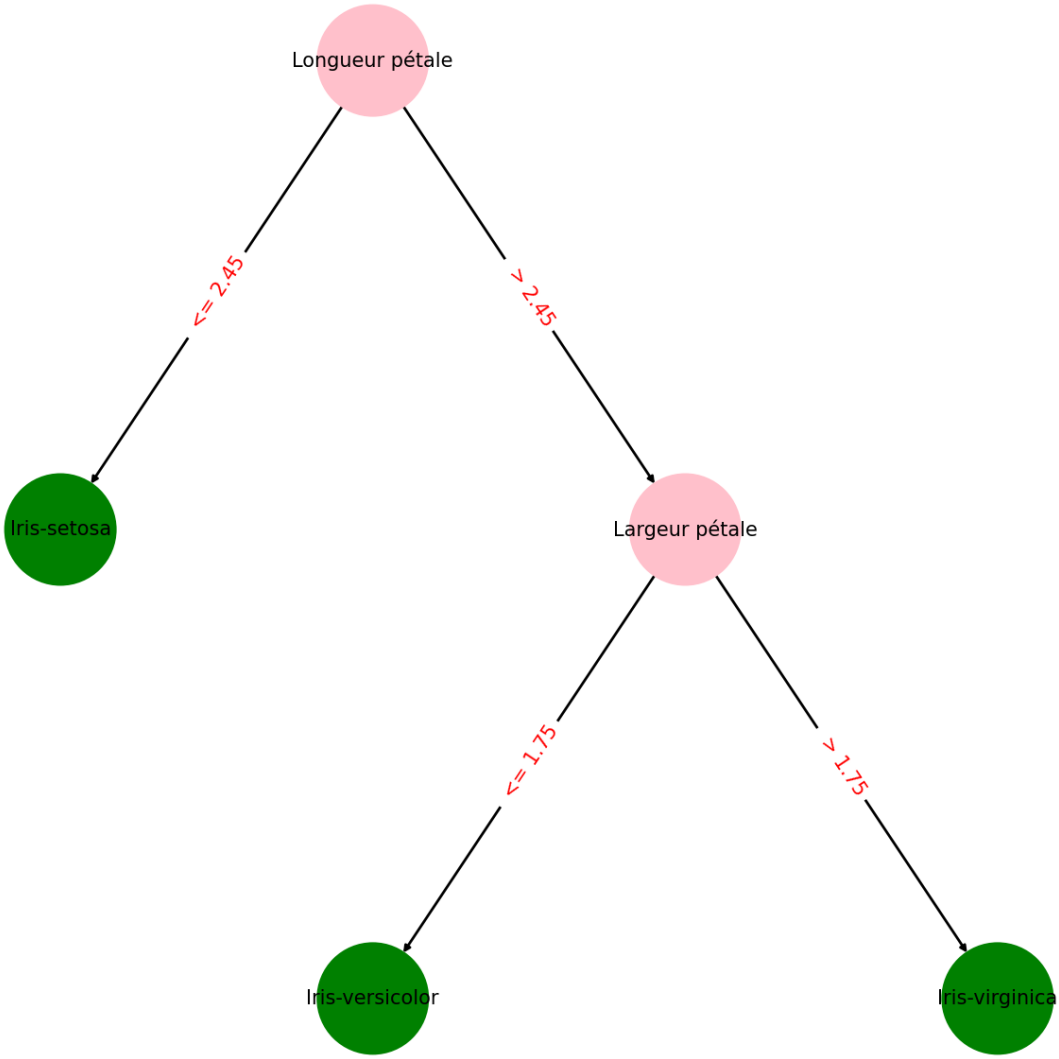

FIGURE 5 – Arbre de décision avec Iris dataset APRÈS élagage.

Sur cette figure 5, nous avons tracé l'arbre de décision que nous avons obtenu avec Iris dataset avec élagage. Si nous comparons cet arbre avec l'arbre de la figure 4, nous constatons de suite qu'il y a eu de gros changements. En effet, nous sommes passés de cinq noeuds terminaux (feuilles) à trois, de quatre noeuds de décision à deux et d'une profondeur de quatre à deux. Maintenant, la classification des iris peut se faire uniquement à l'aide des deux attributs : "Longueur du pétale" et "Largeur du pétale".

3.1.2.1 Entraînement sur nos dataset

Dans cette sous-section, pour chacun de nos dataset, nous avons entraîné notre modèle avec les données d'entraînement.
Afin de vérifier si notre algorithme est en train d'apprendre correctement à partir des exemples que nous lui présentons, nous avons tracé la courbe d'apprentissage pour nos trois jeux de données en suivant la méthode de notre manuel. Pour cela, nous avons pris 100 données (des données d'en-

traînement) générées aléatoirement que nous avons divisés en deux sous-ensembles : un ensemble d'entraînement et un ensemble de test. Nous apprenons une hypothèse h avec le jeu d'entraînement et mesurons sa précision avec le jeu de test. Nous faisons cela en commençant par un ensemble d'entraînement de taille 1 et en augmentant un à chaque fois jusqu'à la taille 99. Pour chaque taille, nous répétons le processus de fractionnement aléatoire 20 fois puis nous prenons la moyenne des résultats des 20 essais dans le but de réduire la variance.

3.1.2.1.1 Iris dataset

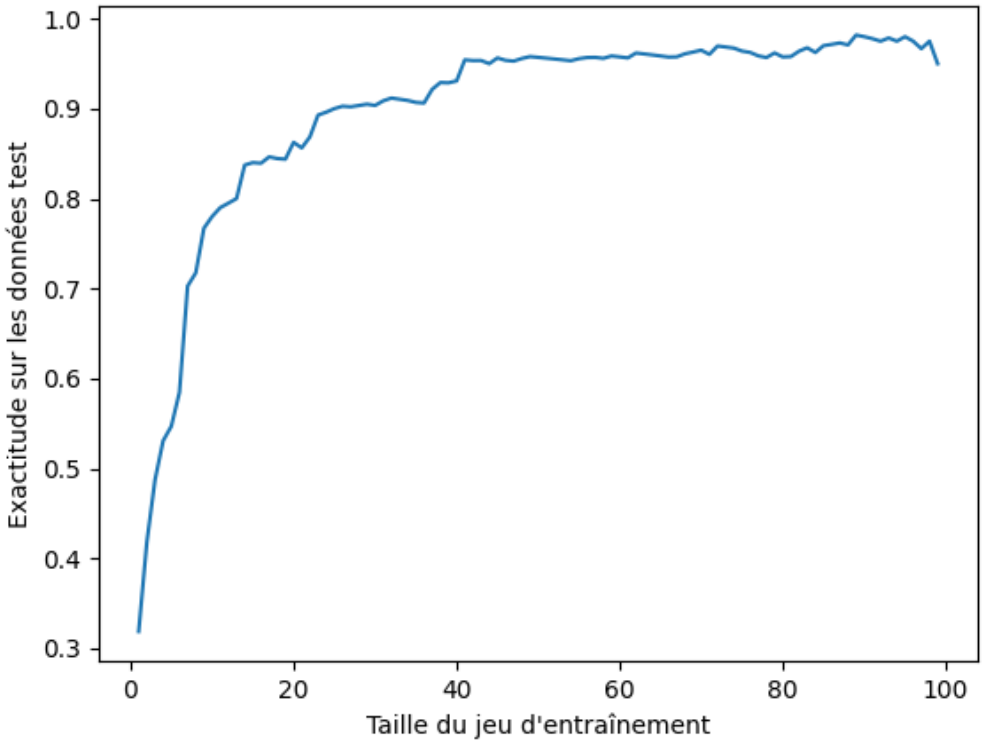

FIGURE 6 – Courbe d'apprentissage de notre arbre de décision avec élagage sur le jeu de données Iris. Exactitude sur le jeu de données test en fonction de la taille du jeu de données en entraînement. Chaque point de données est la moyenne de 20 essais.

Sur cette figure 6, la courbe nous montre que plus la taille du jeu d'entraînement augmente, plus la précision augmente. Dans ce graphique, nous atteignons une précision de 95% et nous pouvons remarquer que l'exactitude sur les données de test semble converger avec l'augmentation de la taille du jeu d'entraînement. Nous pouvons donc dire que notre arbre de décision avec élagage apprend correctement sur ce dataset.
Nous pouvons aussi remarquer que notre courbe d'apprentissage est très semblable à celle de la figure 1 où l'arbre de décision n'était pas élagué.

3.1.2.1.2 Vinho verde - vin blanc

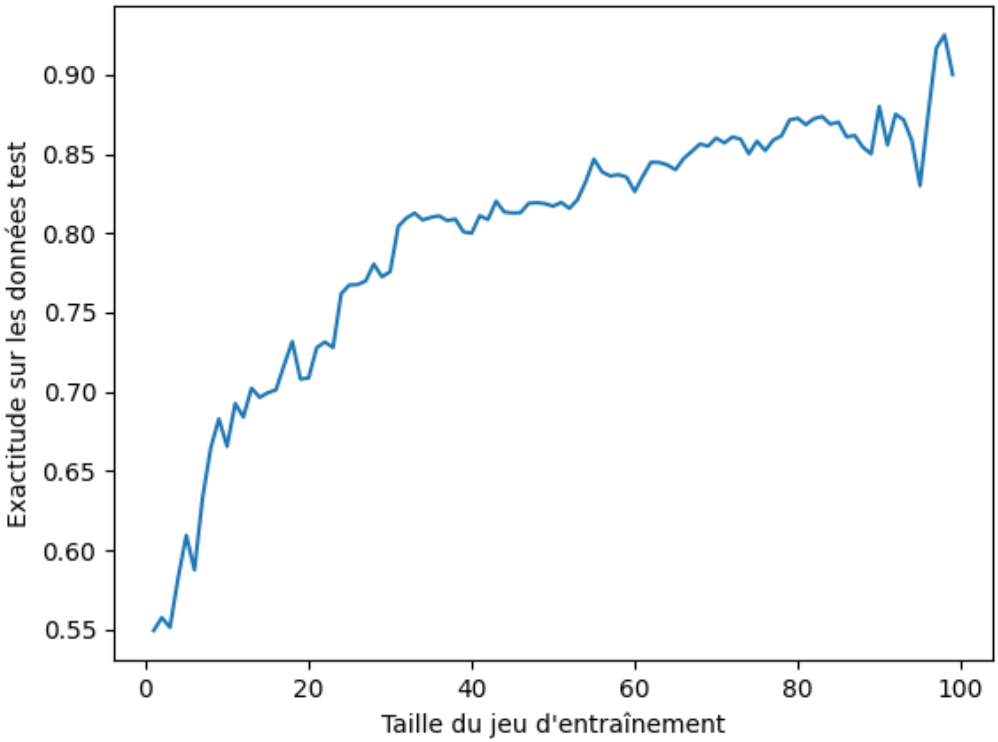

FIGURE 7 – Courbe d'apprentissage de notre arbre de décision avec élagage sur le jeu de données Vinho verde - vin blanc. Exactitude sur le jeu de données test en fonction de la taille du jeu de données en entraînement. Chaque point de données est la moyenne de 20 essais.

Sur cette figure 7, la courbe nous montre que plus la taille du jeu d'entraînement augmente, plus la précision augmente. Dans ce graphique, nous atteignons une précision de 90% et il semble que la courbe pourrait continuer à augmenter avec plus de données. Ici, notre courbe d'apprentissage est très semblable à celle de la figure 2 où l'arbre de décision n'était pas élagué. Nous pouvons donc dire que notre arbre de décision avec élagage apprend correctement sur ce dataset.

3.1.2.1.3 Abalones

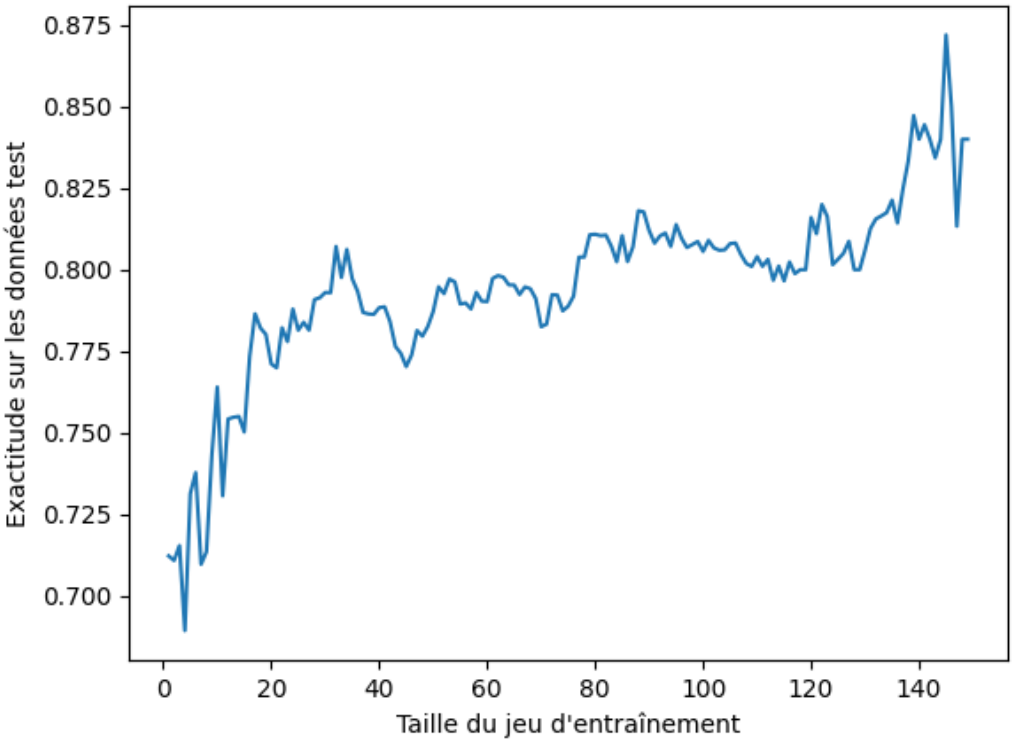

FIGURE 8 – Courbe d'apprentissage de notre arbre de décision avec élagage sur le jeu de données Abalones. Exactitude sur le jeu de données test en fonction de la taille du jeu de données en entraînement. Chaque point de données est la moyenne de 30 essais.

Contrairement aux deux autrès dataset, pour la construction de la courbe d'apprentissage du jeu de données Abalones, nous avons pris 150 données (des données d'entraînement) générées aléatoirement et répété 30 fois l'opération. Puis nous avons pris la moyenne des résultats de ces 30 essais. Nous avons procédé autrement pour pouvoir interpréter plus facilement le graphique.
Sur cette figure 8, la courbe nous montre que plus la taille du jeu d'entraînement augmente, plus la précision augmente. Dans ce graphique, nous atteignons une précision de 84% et il semble que la courbe pourrait continuer à augmenter avec plus de données. Nous pouvons donc dire que notre arbre de décision avec élagage apprend correctement sur ce dataset.
Nous pouvons aussi remarquer que notre courbe d'apprentissage est très semblable à celle de la figure 3 où l'arbre de décision n'était pas élagué.

3.1.2.2 Évaluation des performances de notre modèle

3.1.2.2.1 Iris dataset

(a) En considérant la classe "Iris-setosa" comme la classe positive

	Notre modèle	
Matrice de confusion	17 (VP)	0 (FN)
	0 (FP)	28 (VN)
L'exactitude (Accuracy)	1	
La précision (Precision)	1	
Le rappel (Recall)	1	
Le score F1	1	

(b) En considérant la classe "Iris-versicolour" comme la classe positive

	Notre modèle	
Matrice de confusion	14 (VP)	1 (FN)
	4 (FP)	26 (VN)
L'exactitude (Accuracy)	0.89	
La précision (Precision)	0.78	
Le rappel (Recall)	0.93	
Le score F1	0.85	

(c) En considérant la classe "Iris-virginica" comme la classe positive

	Notre modèle	
Matrice de confusion	9 (VP)	4 (FN)
	1 (FP)	31 (VN)
L'exactitude (Accuracy)	0.89	
La précision (Precision)	0.90	
Le rappel (Recall)	0.69	
Le score F1	0.78	

TABLE 5 – Résumé des résultats obtenus par classe avec notre modèle (arbre de décision avec élagage) sur Iris dataset.

Sur cette table 5, nous pouvons noter que notre modèle classe parfaitement les fleurs "Iris-setosa" (cf : table (a)). En effet, nous avons un résultat de 100% quelle que soit la métrique.
Ensuite, si nous regardons la table (b), nous constatons que le taux de bien classés et le score F1 sont élevés, respectivement ils sont de 89% et 85%. Nous avons un rappel très élevé (0.93) et une précision élevé (0.78). Ceci veut dire que cette espèce est correctement gérée par notre modèle.
Pour finir, analysons la table (c). Pour notre modèle, nous constatons que le taux de bien classés est de 89% et que le score F1 associé est de 78%. Le rappel est assez élevé (0.69) et la précision est très élevé (0.9), nous pouvons dire que notre modèle gère correctement cette espèce d'iris.

De manière générale, nous pouvons dire que notre modèle est efficace pour ce jeu de données.

3.1.2.2.2 Vinho verde - vin blanc

(a) En considérant la classe "0" (mauvais : qualité ≤ 5) comme la classe positive

	Notre modèle	
Matrice de confusion	427 (VP)	68 (FN)
	50 (FP)	266 (VN)
L'exactitude (Accuracy)	0.86	
La précision (Precision)	0.90	
Le rappel (Recall)	0.86	
Le score F1	0.88	

(b) En considérant la classe "1" (bon : qualité ≥ 7) comme la classe positive

	Notre modèle	
Matrice de confusion	266 (VP)	50 (FN)
	68 (FP)	427 (VN)
L'exactitude (Accuracy)	0.86	
La précision (Precision)	0.80	
Le rappel (Recall)	0.84	
Le score F1	0.82	

TABLE 6 – Résumé des résultats obtenus avec notre modèle (arbre de décision avec élagage).

Sur cette table 6, nous pouvons noter que le taux de bien classés est identique pour les deux classes (86%). Cependant, si nous nous attardons sur les autrès métriques, nous constatons que notre modèle est plus performant pour la classe "0" (qualité ≤ 5) que pour la classe "1" (qualité ≥ 7). En effet, par exemple pour le score F1, pour la classe "0", nous avons 0.88 tandis que pour la classe "1", nous avons 0.82. Pour ces deux classes, nous avons des précisions et des rappels qui sont assez élevés. Ceci veut dire que notre modèle gère correctement ces deux classes.

De manière générale, nous pouvons dire que notre modèle est efficace pour ce jeu de données.

3.1.2.2.3 Abalones

(a) En considérant la classe "0" ($n_{rings} < 8$) comme la classe positive

	Notre modèle	
Matrice de confusion	79 (VP)	54 (FN)
	57 (FP)	1064 (VN)
L'exactitude (Accuracy)	0.91	
La précision (Precision)	0.58	
Le rappel (Recall)	0.59	
Le score F1	0.59	

(b) En considérant la classe "1" ($n_{rings} \in [8, 14]$) comme la classe positive

	Notre modèle	
Matrice de confusion	817 (VP)	153 (FN)
	142 (FP)	142 (VN)
L'exactitude (Accuracy)	0.76	
La précision (Precision)	0.85	
Le rappel (Recall)	0.84	
Le score F1	0.85	

(c) En considérant la classe "2" ($n_{rings} > 14$) comme la classe positive

	Notre modèle	
Matrice de confusion	62 (VP)	89 (FN)
	97 (FP)	1006 (VN)
L'exactitude (Accuracy)	0.85	
La précision (Precision)	0.39	
Le rappel (Recall)	0.41	
Le score F1	0.40	

TABLE 7 – Résumé des résultats obtenus avec notre modèle (arbre de décision avec élagage).

Sur cette table 7, lorsque nous regardons les résultats, nous remarquons que ceux-ci sont plutôt hétérogènes entre les différentes classes. Ainsi, il serait plus judicieux d'analyser tableau par tableau. Tout d'abord, commençons par la table (a). Nous avons 91% de bien classés avec notre modèle. Cependant, il y a une très grande différence entre cette mesure et le score F1 associé qui est de 0.59. Cette différence vient du fait que pour le score F1, le nombre de vrais négatifs n'est pas pris en compte et, ici, nous voyons que celui-ci est très élevé si nous considérons la classe "0" comme la classe positive. Même s'il est plus faible que l'exactitude, il est tout de même supérieur à 0.5 ce qui reste raisonnable. Nous avons un rappel (0.59) et une précision (0.58) qui sont légèrement supérieurs au seuil 0.5 ce qui est assez élevé. Nous pouvons donc dire que notre modèle gère correctement cette classe.
Ensuite, si nous regardons la table (b) où nous avons considéré la classe "1" comme la classe positive, nous remarquons que nous avons une exactitude de 76% et un score F1 associé de 85%. Nous

avons également une précision et un rappel qui sont élevés. Ces résultats nous montrent que notre algorithme gère correctement cette classe où le nombre d'anneaux est compris entre 8 et 14 inclus. Pour finir, analysons la table (c) où nous avons considéré la classe "2" comme la classe positive. Nous constatons que le taux de bien classés est de 85% et le score F1 associé qui est de 0.40. Cette différence vient du fait que pour le score F1, le nombre de vrais négatifs n'est pas pris en compte et, ici, nous voyons que celui-ci est très élevé si nous considérons la classe "2" comme la classe positive. Ce qui nous montre que notre modèle n'est pas très adapté pour cette classe sur ce jeu de données. Cette observation est confirmé lorsque nous nous attardons sur la précision et le rappel. Ici, ces deux grandeurs sont inférieures au seuil de 0.5 ce qui signifie que notre modèle gère très mal cette classe car elle n'est généralement pas extraite et quand c'est le cas, le niveau de confiance est faible.

De manière générale, nous pouvons dire que notre arbre de décision avec élagage est plutôt efficace pour ce jeu de données.

3.1.3 Comparaison des résultats AVANT et APRÈS élégage

Jeu de données étudié	Iris		Vinho verde		Abalones	
Arbre de décision (élagage)	Avant	Après	Avant	Après	Avant	Après
Temps d'exécution (s)	0.033	**0.031**	4.018	**3.925**	8.399	**8.275**
Temps d'apprentissage (s)	**0.026**	0.031	**3.799**	3.959	**8.527**	8.814
Temps de prédiction d'un exemple (ms)	0.026	**0.025**	0.185	**0.118**	0.184	**0.170**
Exactitude (Accuracy)	**0.926**	**0.926**	**0.856**	0.855	**0.846**	0.843
Précision (Precision)	**0.893**	**0.893**	**0.847**	0.846	**0.616**	0.608
Rappel (Recall)	**0.875**	**0.875**	**0.854**	0.852	**0.625**	0.616
Score F1	**0.877**	**0.877**	**0.85**	0.849	**0.62**	0.611

TABLE 8 – Tableau récapitulatif des performances de l'algorithme (arbre de décision) avant et après élagage sur les trois jeux de données, où les métriques (exactitude, précision, rappel, score F1) correspondent à la moyenne de celles calculées par classe.

De ce tableau 8, nous remarquons que plus le nombre d'instances est élevé, plus l'algorithme est long à exécuter. Sur notre machine, de manière général, nous constatons que le temps d'apprentissage est plus long après élagage. Ceci s'explique par le fait que ce temps d'apprentissage comprend l'entraînement et l'élagae. Cependant, nous remarquons que ce temps perdu est gagné sur le temps d'exécution. Nous pouvons aussi noter que le temps de prédiction d'un exemple est plus rapide avec l'arbre de décision avec élagage. De plus, nous constatons qu'en terme de perfomance, les résultats obtenus avec l'arbre de décision sans élagage sont très proches de ceux issus de l'arbre avec élagage. Néanmoins, nous observons qu'ils sont très légèrement supérieurs.

3.2 Réseaux de neurones

3.2.1 Première implémentation

1. Nous utilisons 1 couche d'entrée dont le nombre de neurones est le nombre d'attributs du jeu de données ;

2. Nous utilisons une couche de sortie dont le nombre de neurones est le nombre de classes à prédire (2 ou 3). Pour utiliser le réseau de neurones, nous transformons les classes des instances en vecteurs encodés. Par exemple, pour le jeu de données iris, les transformations sont les suivantes :
$$0 \rightarrow [1,0,0]\ ;\ 1 \rightarrow [0,1,0]\ ;\ 2 \rightarrow [0,0,1]$$
3. Nous avons choisi d'utiliser la fonction d'activation sigmoid entre les différentes couches.
4. Puisque l'apprentissage des réseaux de neurones fait appel à un algorithme itératif au cours duquel les poids sont ajustés, il faut tout d'abord initialiser les poids avec des valeurs de départ raisonnables. Parce que la qualité de la solution et le temps nécessaire pour préparer le réseau (apprentissage) sont en jeu. Il est important d'initialiser les poids en utilisant de petites valeurs pour les poids de sorte que, au début de l'apprentissage, le réseau fonctionne en mode linéaire, puis qu'il augmente les valeurs de ses poids afin d'ajuster les données avec suffisamment de précision. Ainsi, nous initialiserons les poids de notre réseau de neurones avec une distribution gaussienne centrée sur 0 et avec un écart type de
$$\frac{1}{\sqrt{n}}$$
avec n le nombre de poids associés à chaque neurone, de manière à éviter une saturation des neurones.

3.2.1.1 Iris dataset

Sur ce jeu de données, nous avons quatre attributs et trois entrées. En suivant cette logique, nous prenons quatre neurones pour la couche d'entrée et trois neurones pour la couche de sortie.

Ici, nous avons choisi arbitrairement les hyperparamètres suivants :

1. Nombre de neurones dans la couche cachée : 3
2. Nombre de couches cachées : 1
3. Nombre d'époques : 1000
4. Nombre de la taille de batch : 16
5. Taux d'apprentissage : 0.5

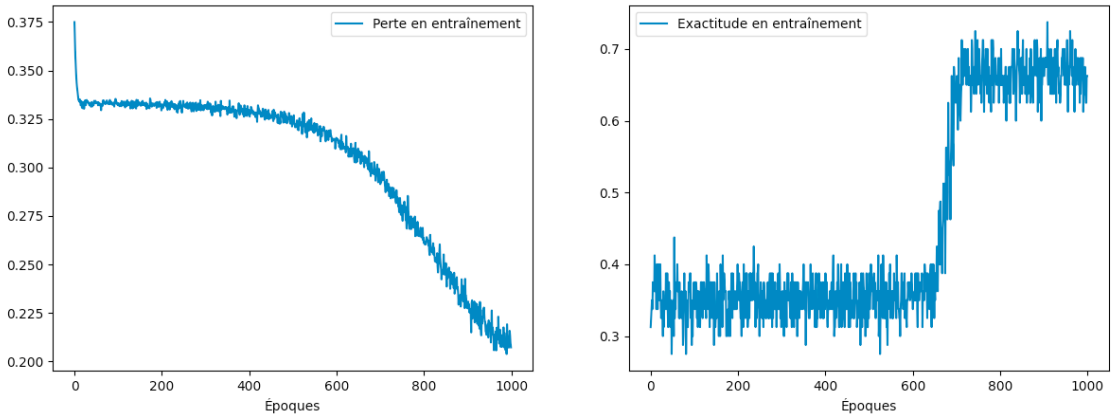

FIGURE 9 – Courbe d'apprentissage sur le jeu de données Iris, en initialisant les poids à 0. Perte et Exactitude sur le jeu de données test en fonction de l'époque.

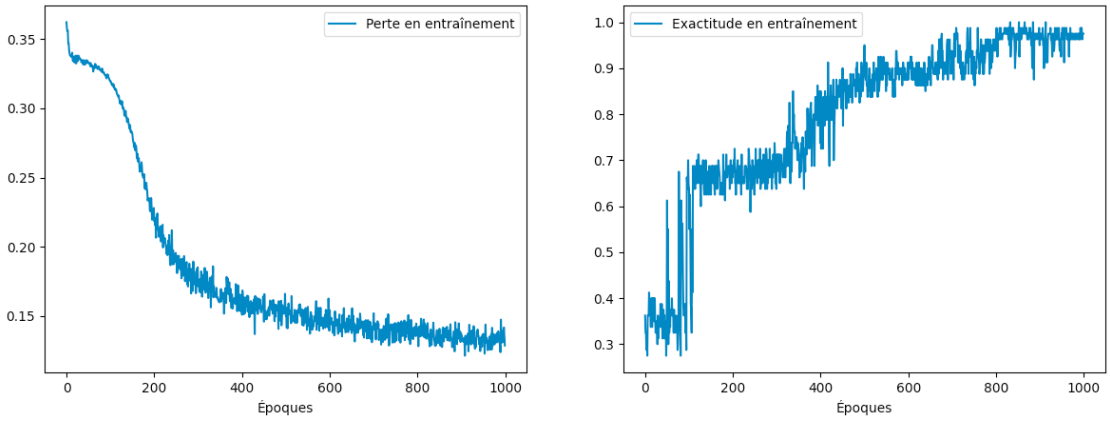

FIGURE 10 – Courbe d'apprentissage sur le jeu de données Iris, en initialisant les poids judicieusement. Perte et Exactitude sur le jeu de données test en fonction de l'époque.

Sur les figures 9, nous avons tracé les courbes d'appprentissage de notre réseau de neurones initialisé avec des poids de zeros. Tandis que sur la figure 10, nous avons la courbe d'apprentissage de notre réseau de neurone initialisé avec des poids obtenus avec notre technique.
Lorsque nous comparons l'exactitude obtenus sur ces deux figures, nous voyons que la précision est plus élevée quand nous initialisons notre réseau de neurones avec des poids différents de zéro. En effet, au bout de 1000 époques nous atteignons une précision de 95 (en initialisant les poids judicieusement) et nous pouvons remarquer que l'exactitude sur les données en entraînement semble converger avec l'augmentation du nombre d'époque. Alors qu'en initialisant les poids à zéro, nous constatons que l'exactitude en entraînement stagne aux alentours de 35% dans un premier temps puis stagne aux alentours de 68%. Même si ces deux courbes nous montre que le nombre d'époques augmente avec la précision, nous devons quand même noter que ces deux courbes n'ont pas la même allure. En effet, avec des poids de zéros, la courbe d'exactitude ressemble à une marche.
En ce qui concerne les courbes de perte en entraînement, nous remarquons que la perte décroit avec le nombre d'époques. Cependant, la perte la plus faible est obtenue en initialisant les poids judicieusement. Concernant la perte avec une initialisation des poids à zéro, nous voyons que la perte en entraînement décroit très très lentement jusqu'à un seuil de 600 époques environ. Puis celle-ci décroit fortement. Nous pouvons voir que ce seuil correspond à la marche que nous observons sur la courbe d'exactitude associée.
Nous pouvons donc dire que notre réseau de neurones apprend beaucoup mieux en initialisant les poids judicieusement.

Ci-dessous, nous évaluons notre modèle sur les données de test de notre Iris dataset.

(a) En considérant la classe "Iris-setosa" comme la classe positive

	Notre modèle	
Matrice de confusion	17 (VP)	0 (FN)
	0 (FP)	28 (VN)
L'exactitude (Accuracy)	1	
La précision (Precision)	1	
Le rappel (Recall)	1	
Le score F1	1	

(b) En considérant la classe "Iris-versicolor" comme la classe positive

	Notre modèle	
Matrice de confusion	12 (VP)	3 (FN)
	0 (FP)	30 (VN)
L'exactitude (Accuracy)	0.93	
La précision (Precision)	1	
Le rappel (Recall)	0.8	
Le score F1	0.89	

(c) En considérant la classe "Iris-virginica" comme la classe positive

	Notre modèle	
Matrice de confusion	13 (VP)	0 (FN)
	3 (FP)	29 (VN)
L'exactitude (Accuracy)	0.93	
La précision (Precision)	0.81	
Le rappel (Recall)	1	
Le score F1	0.90	

TABLE 9 – Résumé des résultats obtenus par classe avec notre modèle (réseau de neurones) sur Iris dataset.

Sur cette table 9, nous pouvons noter que notre modèle classe parfaitement les fleurs "Iris-setosa" (cf : table (a)). En effet, nous avons un résultat de 100% quelle que soit la métrique.
Ensuite, si nous regardons la table (b), nous constatons que le taux de bien classés et le score F1 sont élevés, respectivement ils sont de 93% et 89%. Nous avons un rappel élevé (0.8) et une précision maximale (1). Ceci veut dire que cette espèce est correctement gérée par notre modèle.
Pour finir, analysons la table (c). Pour notre modèle, nous constatons que le taux de bien classés est de 93% et que le score F1 associé est de 90%. Le rappel est maximal (1) et la précision est élevé (0.8), nous pouvons dire que notre modèle gère correctement cette espèce d'iris.

De manière générale, nous pouvons dire que notre modèle est très efficace pour ce jeu de données.

3.2.1.2 Vinho verde - vin blanc

Sur ce jeu de données, nous avons onze attributs et deux entrées. En suivant cette logique, nous prenons onze neurones pour la couche d'entrée et deux neurones pour la couche de sortie.

Ici, nous avons choisi arbitrairement les hyperparamètres suivants :

1. Nombre de neurones dans la couche cachée : 5
2. Nombre de couches cachées : 1
3. Nombre d'époques : 1000
4. Nombre de la taille de batch : 64
5. Taux d'apprentissage : 0.5

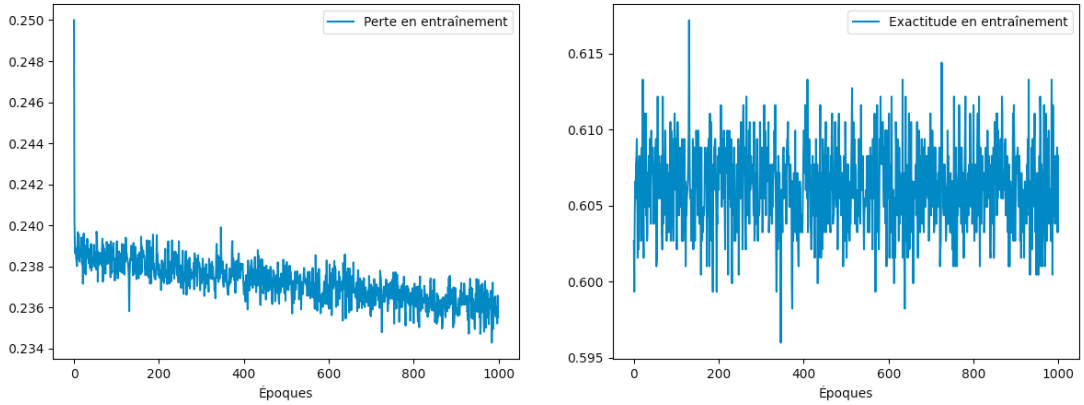

FIGURE 11 – Courbe d'apprentissage sur le jeu de données Vin, en initialisant les poids à 0. Perte et Exactitude sur le jeu de données test en fonction de l'époque.

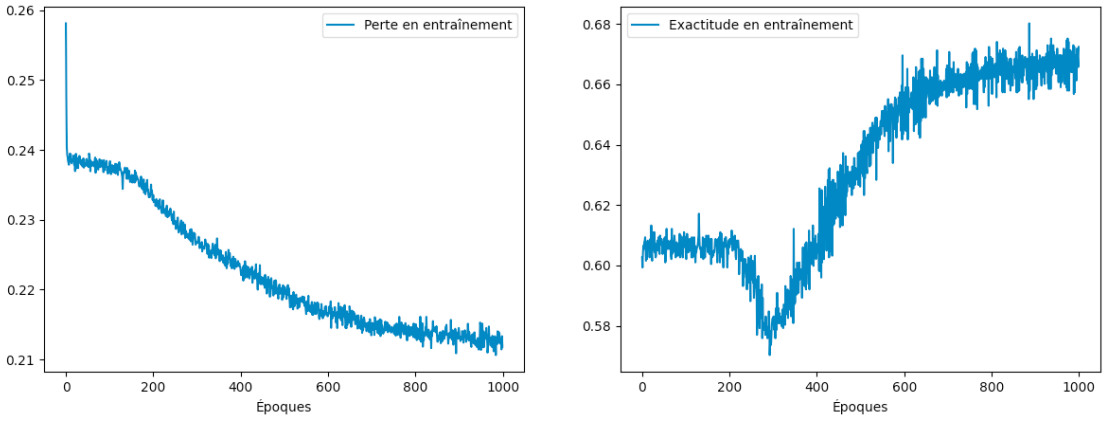

FIGURE 12 – Courbe d'apprentissage sur le jeu de données Vin, en initialisant les poids judicieusement. Perte et Exactitude sur le jeu de données test en fonction de l'époque.

Sur les figures 11, nous avons tracé les courbes d'appprentissage de notre réseau de neurones initialisé avec des poids de zeros. Tandis que sur la figure 12, nous avons la courbe d'apprentissage de notre réseau de neurone initialisé avec des poids obtenus avec notre technique.

Lorsque nous comparons l'exactitude obtenus sur ces deux figures, nous voyons que la précision est plus élevée quand nous initialisons notre réseau de neurones avec des poids différents de zéro. En effet, au bout de 1000 époques nous atteignons une précision de 67 (en initialisant les poids judicieusement) et nous pouvons remarquer que l'exactitude sur les données en entraînement pourrait continuer à augmenter avec l'augmentation du nombre d'époque. Alors qu'en initialisant les poids à zéro, nous constatons que l'exactitude en entraînement stagne et converge aux alentours de 60.7%.
En ce qui concerne les courbes de perte en entraînement, nous remarquons que la perte décroit avec le nombre d'époques. Cependant, la perte la plus faible est obtenue en initialisant les poids judicieusement.
Nous pouvons donc dire que notre réseau de neurones apprend beaucoup mieux en initialisant les poids judicieusement.

Ci-dessous, nous évaluons notre modèle sur les données de test de notre jeu de données Vinho verde - vin blanc.

(a) En considérant la classe "0" (mauvais : qualité ≤ 5) comme la classe positive

	Notre modèle	
Matrice de confusion	426 (VP)	69 (FN)
	179 (FP)	137 (VN)
L'exactitude (Accuracy)	0.69	
La précision (Precision)	0.70	
Le rappel (Recall)	0.86	
Le score F1	0.77	

(b) En considérant la classe "1" (bon : qualité ≥ 7) comme la classe positive

	Notre modèle	
Matrice de confusion	137 (VP)	179 (FN)
	69 (FP)	426 (VN)
L'exactitude (Accuracy)	0.69	
La précision (Precision)	0.68	
Le rappel (Recall)	0.65	
Le score F1	0.65	

TABLE 10 – Résumé des résultats obtenus avec notre modèle (Réseau de neurones) sur Vinho verde - vin blanc.

Sur cette table 10, nous pouvons noter que le taux de bien classés est identique pour les deux classes (69%). Cependant, si nous nous attardons sur les autrès métriques, nous constatons que notre modèle est plus performant pour la classe "0" (qualité ≤ 5) que pour la classe "1" (qualité ≥ 7). En effet, par exemple pour le score F1, pour la classe "0", nous avons 0.77 tandis que pour la classe "1", nous avons 0.65. Pour ces deux classes, nous avons des précisions et des rappels qui sont assez élevés. Ceci veut dire que notre modèle gère correctement ces deux classes.

De manière générale, nous pouvons dire que notre modèle est efficace pour ce jeu de données.

3.2.1.3 Abalones

Sur ce jeu de données, nous avons huit attributs et trois entrées. En suivant cette logique, nous prenons huit neurones pour la couche d'entrée et trois neurones pour la couche de sortie.

Ici, nous avons choisi arbitrairement les hyperparamètres suivants :

1. Nombre de neurones dans la couche cachée : 5
2. Nombre de couches cachées : 1
3. Nombre d'époques : 100
4. Nombre de la taille de batch : 64
5. Taux d'apprentissage : 0.5

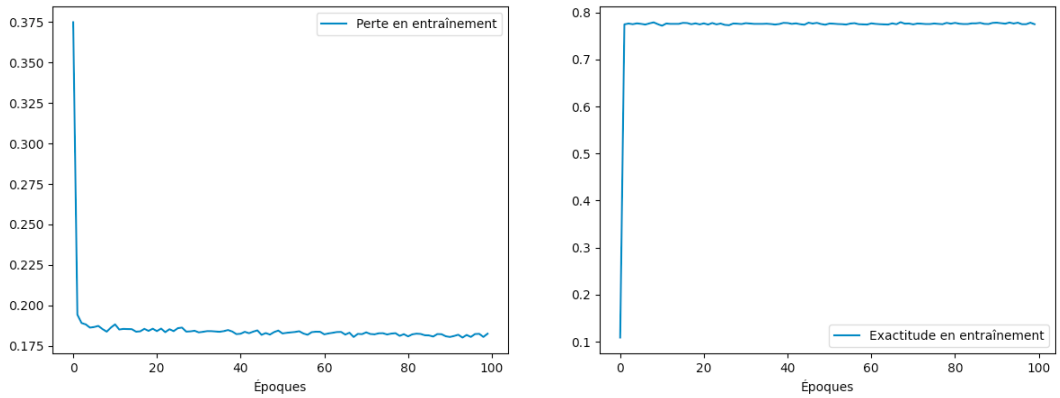

FIGURE 13 – Courbe d'apprentissage sur le jeu de données Abalones, en initialisant les poids à 0. Perte et Exactitude sur le jeu de données test en fonction de l'époque.

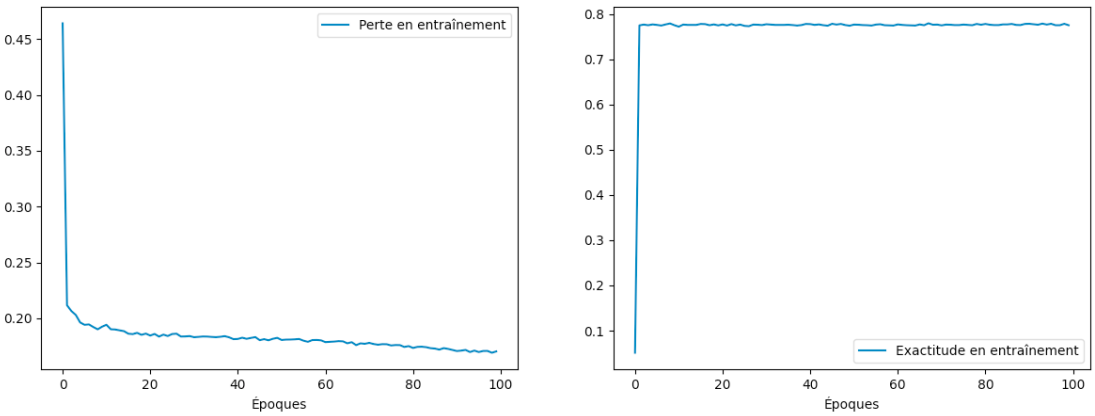

FIGURE 14 – Courbe d'apprentissage sur le jeu de données Abalones, en initialisant les poids judicieusement. Perte et Exactitude sur le jeu de données test en fonction de l'époque.

Sur les figures 13, nous avons tracé les courbes d'appprentissage de notre réseau de neurones initialisé avec des poids de zeros. Tandis que sur la figure 14, nous avons la courbe d'apprentissage de notre

réseau de neurone initialisé avec des poids obtenus avec notre technique.
Nous remarquons de suite que dans les deux cas, nous obtenons les mêmes courbes concernant l'exactitude et la perte en entraînement. Concernant la courbe d'exactitude, nous voyons que celle-ci augmente fortement au bout d'une époque pour atteindre les 78%. Puis cette valeur stagne quelque soit le nombre d'époques. Nous pouvons donc dire que notre modèle n'apprend pas correctement avec ces choix d'hyperparamètres quelque soit le type d'initialisation des poids.

Ci-dessous, nous évaluons notre modèle sur les données de test de notre jeu de données Abalones.

(a) En considérant la classe "0" ($n_{rings} < 8$) comme la classe positive

	Notre modèle	
Matrice de confusion	0 (VP)	133 (FN)
	0 (FP)	1121 (VN)
L'exactitude (Accuracy)	0.89	
La précision (Precision)	Nan	
Le rappel (Recall)	0	
Le score F1	Nan	

(b) En considérant la classe "1" ($n_{rings} \in [8, 14]$) comme la classe positive

	Notre modèle	
Matrice de confusion	970 (VP)	0 (FN)
	284 (FP)	0 (VN)
L'exactitude (Accuracy)	0.77	
La précision (Precision)	0.77	
Le rappel (Recall)	1	
Le score F1	0.88	

(c) En considérant la classe "2" ($n_{rings} > 14$) comme la classe positive

	Notre modèle	
Matrice de confusion	0 (VP)	151 (FN)
	0 (FP)	1103 (VN)
L'exactitude (Accuracy)	0.88	
La précision (Precision)	Nan	
Le rappel (Recall)	0	
Le score F1	Nan	

TABLE 11 – Résumé des résultats obtenus avec notre modèle (Réseau de neurones) sur les abalones

Sur cette table 11, nous voyons que les tables (a) et (c) sont très semblables. Nous avons des exactitudes quasiment identiques (89% pour la table (a) et 88% pour la table (c)). Nous avons des précisions, rappels et scores F1 identiques qui sont respectivement "NaN", 0 et "NaN". Ces métriques

nous montre que notre modèles gère très mal ces deux classes où le nombre d'anneaux est inférieur à 8 et où le nombre d'anneaux est supérieur à 14.
Ensuite, si nous regardons la table (b) où nous avons considéré la classe "1" comme la classe positive, nous remarquons que nous avons une exactitude de 77% et un score F1 associé de 88%. Nous avons également une précision et un rappel qui sont élevés. Ces résultats nous montrent que notre algorithme gère correctement cette classe où le nombre d'anneaux est compris entre 8 et 14 inclus.

De manière générale, avec ce paramétrage, nous pouvons dire que notre réseau de neurones n'est pas très efficace sur ce jeu de données. En effet, il apprend à prédire la classe majoritaire, soit la classe "1", dans tous les cas.

3.2.2 Recherche d'hyperparamètres

Pour ce TP, nous utilisons la validation croisée afin de choisir le nombre de neurones et le nombre de couches cachées pour chaque dataset, dans le but d'améliorer les performances des modèles sur nos jeux de données. Nous réalisons donc une "recherche en grille" en faisant une validation croisée sur le jeu de données d'entraînement, avec différentes combinaisons d'hyperparamètres. Nous divisons le jeu de données étudié en k sets de données et entraînons/évaluons le modèle k fois en utilisant l'un des k-fold pour l'évaluation et le reste pour l'entraînement. Nous retenons les paramètres qui ont permis de maximiser la moyenne des exactitudes en validation obtenues à la fin de l'apprentissage.

3.2.2.1 Choix du nombre de neurones dans la couche cachée

Il existe de nombreuses méthodes empiriques pour déterminer le nombre correct de neurones à utiliser dans les couches cachées, telles que :

1. Le nombre de neurones cachés doit être compris entre la taille de la couche d'entrée et la taille de la couche de sortie ;
2. Le nombre de neurones cachés doit être 2/3 de la taille de la couche d'entrée, plus la taille de la couche de sortie ;
3. Le nombre de neurones cachés doit être inférieur à deux fois la taille de la couche d'entrée.

Dans notre cas, nous le déterminerons en fixant les autres hyperparamètres qui ont permis de maximiser la moyenne des exactitudes en validation obtenues à la fin de l'apprentissage et en faisant varier un nombre de neurones entre 2 et 50.

3.2.2.1.1 Iris dataset

Pour trouver la meilleure valeur concernant le nombre de neurones dans la couche cachée, nous avons fait une recherche avec les valeurs suivantes :

1. K-fold : 5.
2. Nombre de neurones dans la couche cachée : entre 2 et 50.
3. Nombre de couche cachée : 1.
4. Nombre d'époques : 1000.
5. Nombre de la taille de batch : 16.
6. Taux d'apprentissage : 0.5

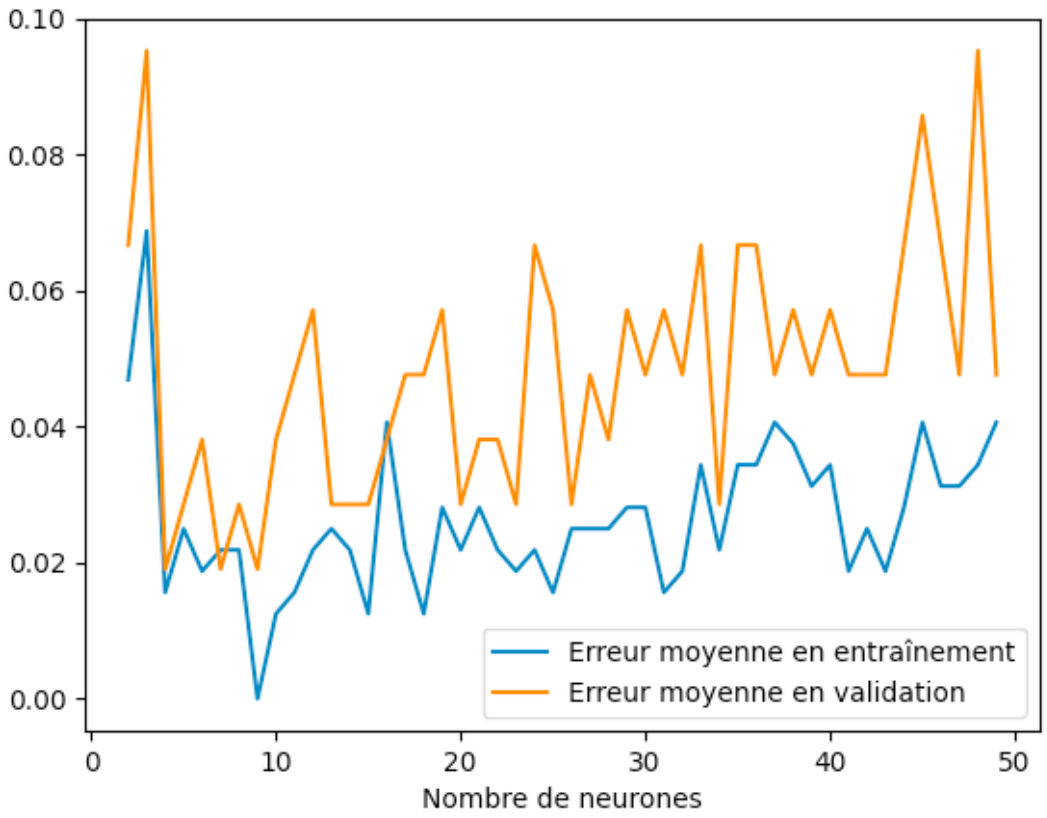

FIGURE 15 – Erreur moyenne en fonction du nombre de neurones dans la couche cachée.

Sur cette figure 15, nous avons tracé la courbe de l'erreur moyenne en fonction du nombre de neurones dans la couche cachée. Nous constatons que l'allure de cette erreur moyenne en validation et en entraînement sont semblables. L'erreur minimale obtenue en moyenne en validation est celle correspondant à un nombre de neurones égal à 4. C'est donc cette valeur que nous utiliserons par la suite pour Iris dataset.

3.2.2.1.2 Vinho verde - vin blanc

Pour trouver la meilleure valeur concernant le nombre de neurones dans la couche cachée, nous avons fait une recherche avec les valeurs suivantes :

1. K-fold : 10.
2. Nombre de neurones dans la couche cachée : entre 2 et 50.
3. Nombre de couche cachée : 1.
4. Nombre d'époques : 1000.
5. Nombre de la taille de batch : 64.
6. Taux d'apprentissage : 0.5

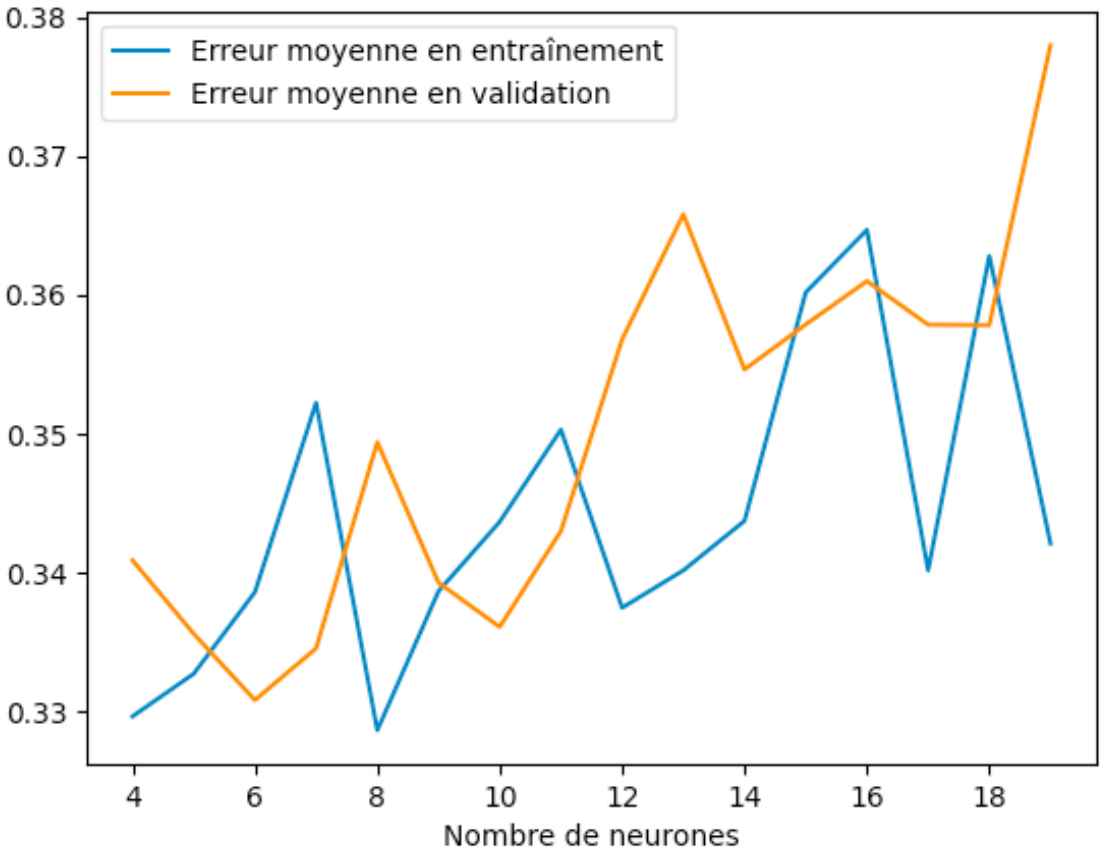

FIGURE 16 – Erreur moyenne en fonction du nombre de neurones dans la couche cachée.

Sur cette figure 16, nous avons tracé la courbe de l'erreur moyenne en fonction du nombre de neurones dans la couche cachée. Nous constatons que l'allure de cette erreur moyenne en validation et en entraînement sont semblables. L'erreur minimale obtenue en moyenne en validation est celle correspondant à un nombre de neurones égal à 6. C'est donc cette valeur que nous utiliserons par la suite pour le jeu de données Vinho verde - vin blanc.

3.2.2.1.3 Abalones

Pour trouver la meilleure valeur concernant le nombre de neurones dans la couche cachée, nous avons fait une recherche avec les valeurs suivantes :

1. Nombre de neurones dans la couche cachée : entre 2 et 50.
2. Nombre de couche cachée : 1.
3. Nombre d'époques : 1000.
4. Nombre de la taille de batch : 16.
5. Taux d'apprentissage : 0.5

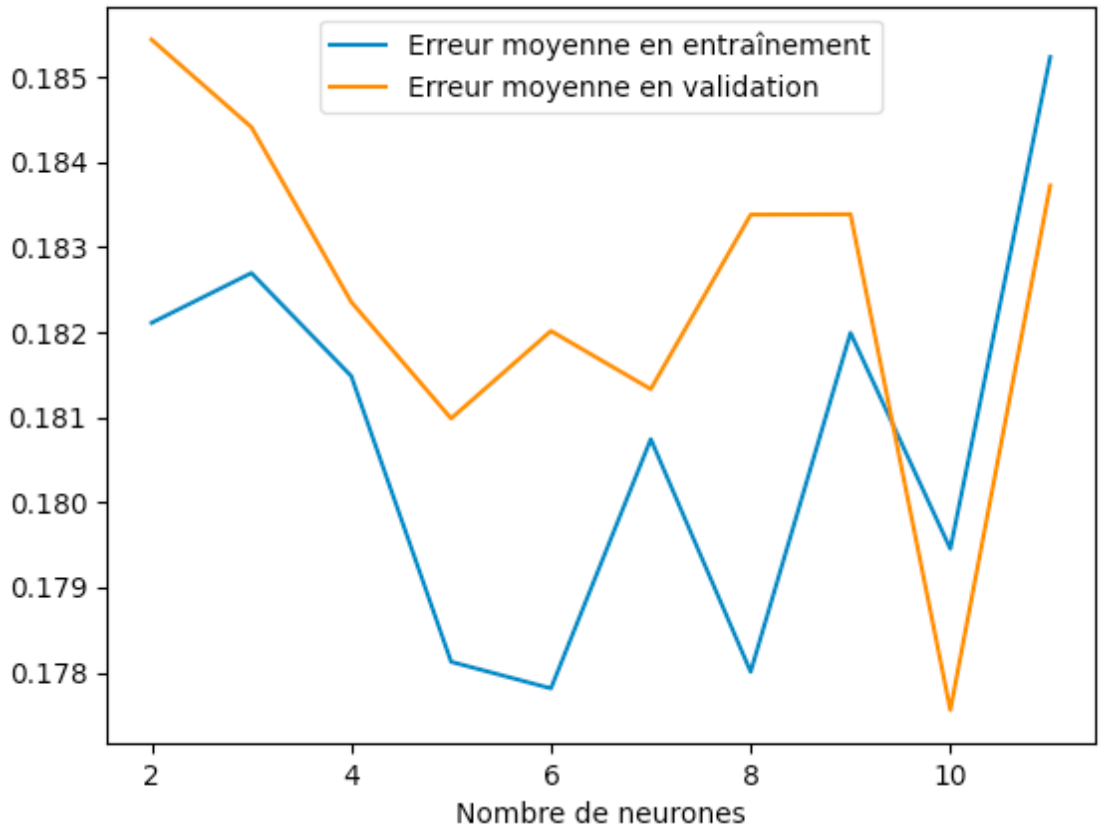

FIGURE 17 – Erreur moyenne en fonction du nombre de neurones dans la couche cachée pour le jeu Abalones.

Sur cette figure 17, nous avons tracé la courbe de l'erreur moyenne en fonction du nombre de neurones dans la couche cachée. Nous constatons que l'allure de cette erreur moyenne en validation et en entraînement sont semblables. L'erreur minimale obtenue en moyenne en validation est celle correspondant à un nombre de neurones égal à 10. C'est donc cette valeur que nous utiliserons par la suite pour le jeu de données Abalones.

3.2.2.2 Choix du nombre de couches cachées

En utilisant le nombre de neurones trouvés à l'étape précédente, nous faisaons à nouveau une validation croisée pour comparer cinq architectures de réseaux de neurones : un avec 1 couche cachée soit `RN-1C`, un avec 2 couches cachées soit `RN-2C`, et ainsi de suite jusqu'à `RN-5C`.

Les fonctions sigmoïdes sont fréquemment utilisées dans les réseaux de neurones pour activer les neurones. C'est une fonction logarithmique avec une forme caractéristique en S.
Pour les noeuds avec des fonctions d'activation sigmoïdes, nous savons que la dérivée partielle de la fonction sigmoïde atteint une valeur maximale de 0.25. Lorsqu'il y a plus de couches dans le réseau, la valeur du produit de la dérivée diminue jusqu'à ce qu'à un moment où la dérivée partielle de la fonction de perte s'approche d'une valeur proche de zéro, et la dérivée partielle disparaît. C'est ce que nous appelons phénomène de *Vanishing Gradient*.

Pour palier ce problème, le plus simple consiste à remplacer la fonction d'activation du réseau. Au lieu de sigmoïde, nous pouvons par exemple utiliser une fonction d'activation telle que ReLU.

3.2.2.2.1 Iris dataset

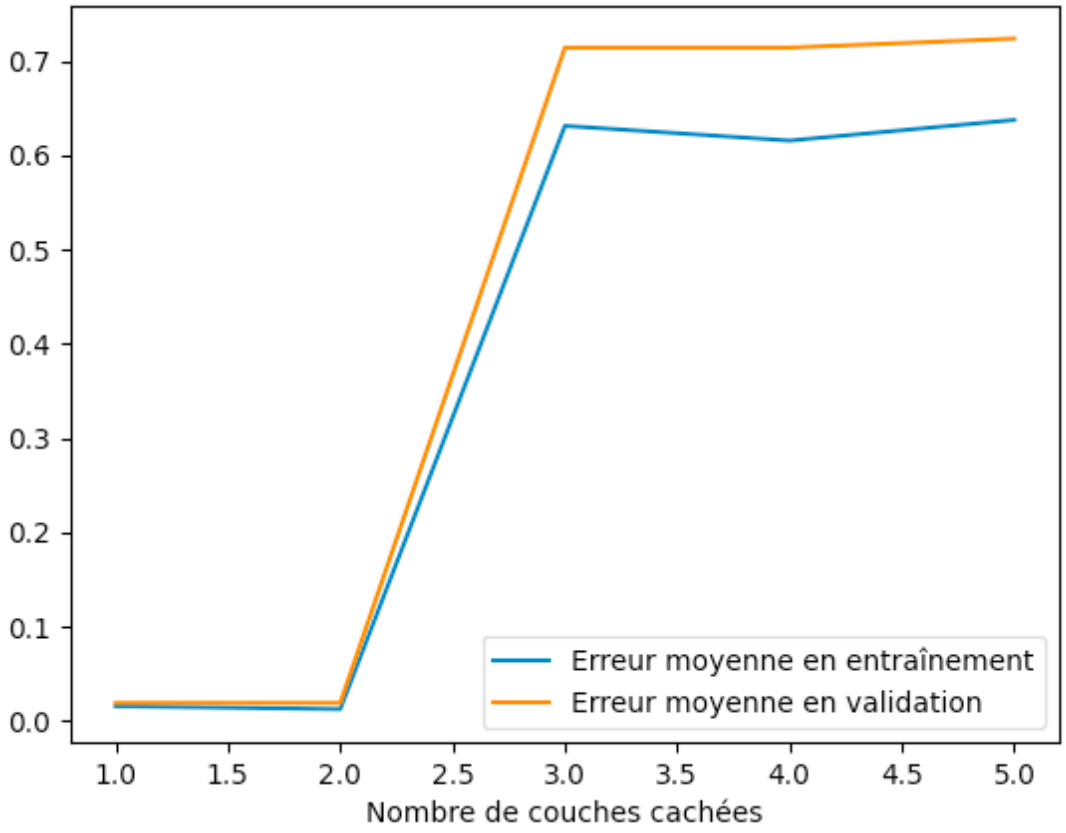

FIGURE 18 – Erreur moyenne en fonction du nombre de couches cachées.

Sur cette figure 18, nous avons tracé la courbe de l'erreur moyenne en fonction du nombre de couches cachhées. Nous constatons que l'allure de cette erreur moyenne en validation et en entraînement sont semblables. L'erreur minimale obtenue en moyenne en validation est celle correspondant à un nombre de couches égal à 2. C'est donc cette valeur que nous utiliserons par la suite pour Iris dataset.
De plus, nous pouvons noter que nous observons ici le pnénomène de *Vanishing Gradient* (courbe en forme de S).

3.2.2.2.2 Vinho verde - vin blanc

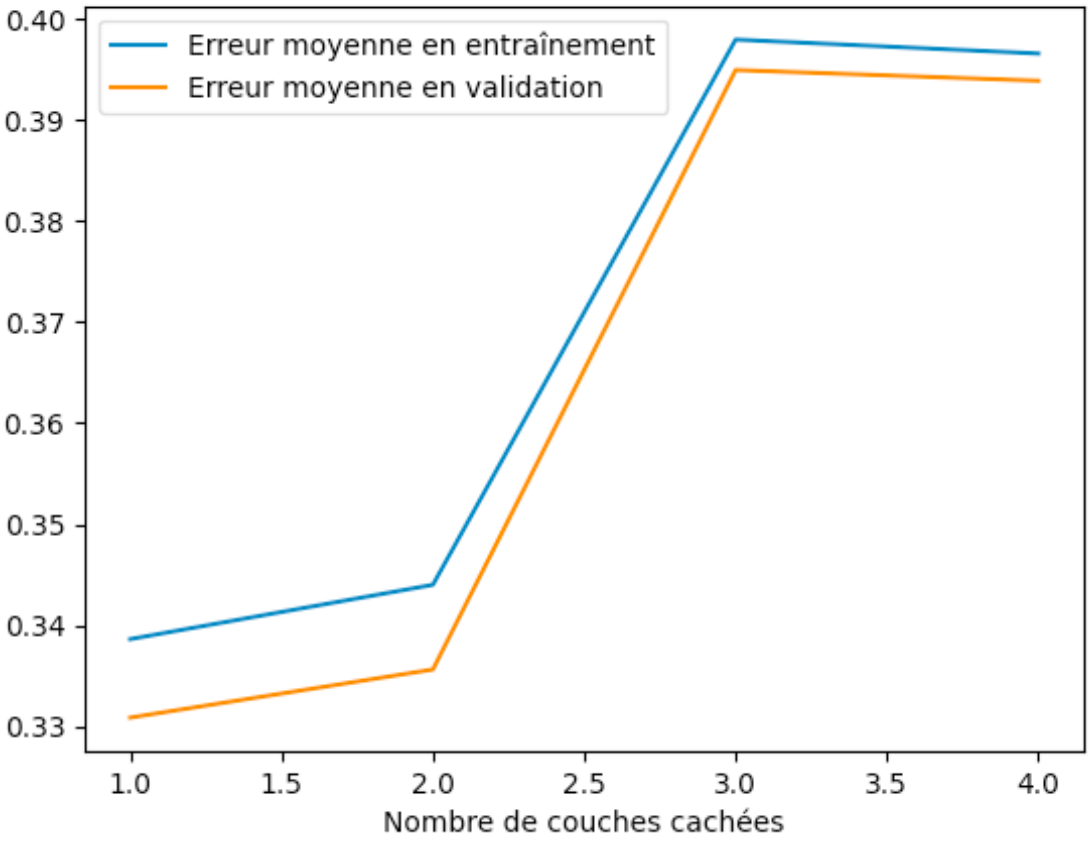

FIGURE 19 – Erreur moyenne en fonction du nombre de couches cachées.

Sur cette figure 19, nous avons tracé la courbe de l'erreur moyenne en fonction du nombre de couches cachhées. Nous constatons que l'allure de cette erreur moyenne en validation et en entraînement sont semblables. L'erreur minimale obtenue en moyenne en validation est celle correspondant à un nombre de couches égal à 1. C'est donc cette valeur que nous utiliserons par la suite pour le jeu de données Vinho verde - vin blanc.
De plus, nous pouvons noter que nous observons ici le pnénomène de *Vanishing Gradient* (courbe en forme de S).

3.2.2.2.3 Abalones

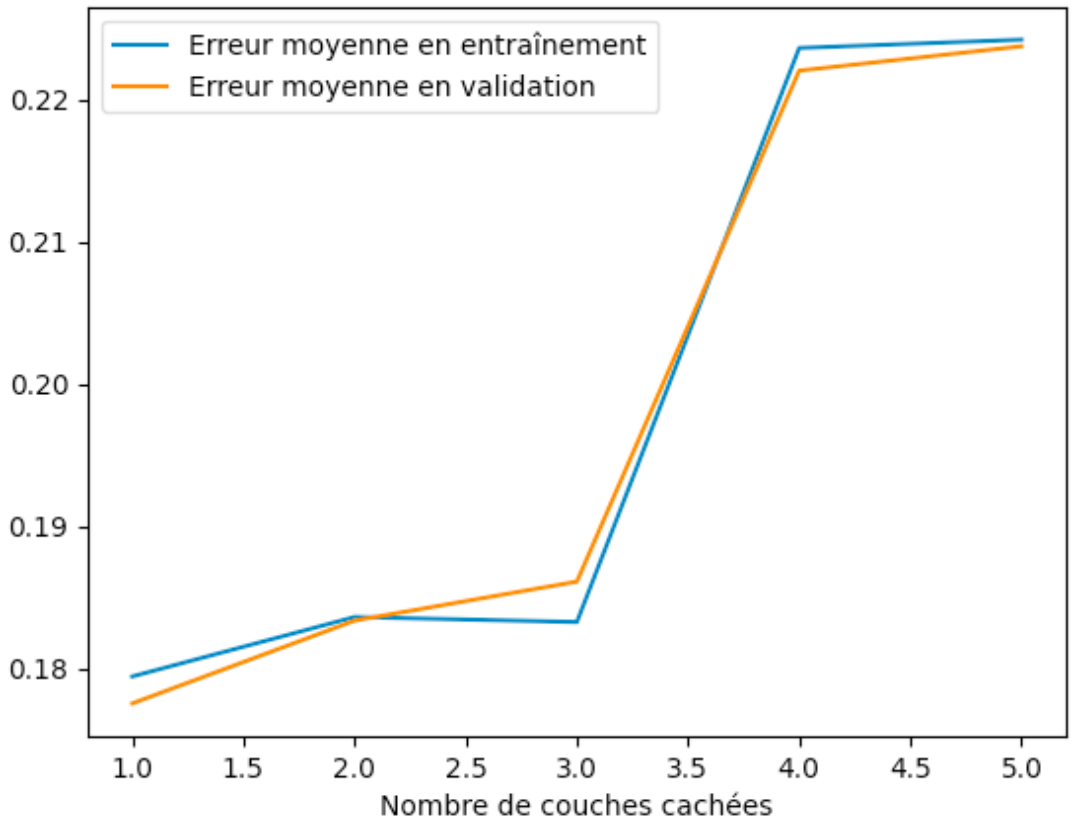

FIGURE 20 – Erreur moyenne en fonction du nombre de couches cachées.

Sur cette figure 20, nous avons tracé la courbe de l'erreur moyenne en fonction du nombre de couches cachhées. Nous constatons que l'allure de cette erreur moyenne en validation et en entraînement sont semblables. L'erreur minimale obtenue en moyenne en validation est celle correspondant à un nombre de couches égal à 1. C'est donc cette valeur que nous utiliserons par la suite pour le jeu de données Abalones.
De plus, nous pouvons noter que nous observons ici le pnénomène de *Vanishing Gradient* (courbe en forme de S).

3.2.2.3 Évaluation de la meilleure architecture

3.2.2.3.1 Iris dataset

Nous avons retenu les valeurs suivantes pour l'évaluation de notre modèle sur ce jeu de données.

1. Nombre de neurones dans la couche cachée : 4
2. Nombre de couches cachées : 2
3. Nombre d'époques : 1000
4. Nombre de la taille de batch : 16
5. Taux d'apprentissage : 0.5

(a) En considérant la classe "Iris-setosa" comme la classe positive

	Notre modèle	
Matrice de confusion	17 (VP)	0 (FN)
	0 (FP)	28 (VN)
L'exactitude (Accuracy)	1	
La précision (Precision)	1	
Le rappel (Recall)	1	
Le score F1	1	

(b) En considérant la classe "Iris-versicolour" comme la classe positive

	Notre modèle	
Matrice de confusion	14 (VP)	1 (FN)
	0 (FP)	30 (VN)
L'exactitude (Accuracy)	0.98	
La précision (Precision)	1	
Le rappel (Recall)	0.93	
Le score F1	0.97	

(c) En considérant la classe "Iris-virginica" comme la classe positive

	Notre modèle	
Matrice de confusion	13 (VP)	0 (FN)
	1 (FP)	31 (VN)
L'exactitude (Accuracy)	0.98	
La précision (Precision)	0.93	
Le rappel (Recall)	1	
Le score F1	0.96	

TABLE 12 – Résumé des résultats obtenus par classe avec notre modèle (réseau de neurones) sur Iris dataset.

Sur cette table 12, nous pouvons noter que notre modèle classe parfaitement les fleurs "Iris-setosa" (cf : table (a)). En effet, nous avons un résultat de 100% quelle que soit la métrique.
Ensuite, si nous regardons la table (b), nous constatons que le taux de bien classés et le score F1 sont élevés, respectivement ils sont de 98% et 97%. Nous avons un rappel très élevé (0.93) et une précision maximale (1). Ceci veut dire que cette espèce est correctement gérée par notre modèle.
Pour finir, analysons la table (c). Pour notre modèle, nous constatons que le taux de bien classés est de 98% et que le score F1 associé est de 96%. Le rappel est maximal (1) et la précision est très élevé (0.93), nous pouvons dire que notre modèle gère correctement cette espèce d'iris.

De manière générale, nous pouvons dire que notre modèle est très efficace pour ce jeu de données.

3.2.2.3.2 Vinho verde - vin blanc

Nous avons retenu les valeurs suivantes pour l'évaluation de notre modèle sur ce jeu de données.

1. Nombre de neurones dans la couche cachée : 6
2. Nombre de couches cachées : 1
3. Nombre d'époques : 20000
4. Nombre de la taille de batch : 64
5. Taux d'apprentissage : 0.5

(a) En considérant la classe "0" (mauvais : qualité ≤ 5) comme la classe positive

	Notre modèle	
Matrice de confusion	417 (VP)	78 (FN)
	54 (FP)	262 (VN)
L'exactitude (Accuracy)	0.84	
La précision (Precision)	0.89	
Le rappel (Recall)	0.84	
Le score F1	0.86	

(b) En considérant la classe "1" (bon : qualité ≥ 7) comme la classe positive

	Notre modèle	
Matrice de confusion	262 (VP)	54 (FN)
	78 (FP)	417 (VN)
L'exactitude (Accuracy)	0.84	
La précision (Precision)	0.77	
Le rappel (Recall)	0.83	
Le score F1	0.80	

TABLE 13 – Résumé des résultats obtenus avec notre modèle (Réseau de neurones) sur Vinho verde - vin blanc.

Sur cette table 13, nous pouvons noter que le taux de bien classés est identique pour les deux classes (84%). Cependant, si nous nous attardons sur les autrès métriques, nous constatons que notre modèle est plus performant pour la classe "0" (qualité ≤ 5) que pour la classe "1" (qualité ≥ 7). En effet, par exemple pour le score F1, pour la classe "0", nous avons 0.86 tandis que pour la classe "1", nous avons 0.80. Pour ces deux classes, nous avons des précisions et des rappels qui sont élevés. Ceci veut dire que notre modèle gère correctement ces deux classes.

De manière générale, nous pouvons dire que notre modèle est efficace pour ce jeu de données.

3.2.2.3.3 Abalones

Nous avons retenu les valeurs suivantes pour l'évaluation de notre modèle sur ce jeu de données.

1. Nombre de neurones dans la couche cachée : 10
2. Nombre de couches cachées : 1
3. Nombre d'époques : 1000
4. Nombre de la taille de batch : 64
5. Taux d'apprentissage : 0.5

(a) En considérant la classe "0" ($n_{rings} < 8$) comme la classe positive

	Notre modèle	
Matrice de confusion	74 (VP)	58 (FN)
	21 (FP)	1100 (VN)
L'exactitude (Accuracy)	0.94	
La précision (Precision)	0.78	
Le rappel (Recall)	0.56	
Le score F1	0.66	

(b) En considérant la classe "1" ($n_{rings} \in [8, 14]$) comme la classe positive

	Notre modèle	
Matrice de confusion	949 (VP)	21 (FN)
	204 (FP)	80 (VN)
L'exactitude (Accuracy)	0.82	
La précision (Precision)	0.82	
Le rappel (Recall)	0.98	
Le score F1	0.89	

(c) En considérant la classe "2" ($n_{rings} > 14$) comme la classe positive

	Notre modèle	
Matrice de confusion	5 (VP)	146 (FN)
	0 (FP)	1103 (VN)
L'exactitude (Accuracy)	0.88	
La précision (Precision)	1	
Le rappel (Recall)	0.03	
Le score F1	0.06	

TABLE 14 – Résumé des résultats obtenus avec notre modèle (Réseau de neurones) sur les abalones

Sur cette table 14, lorsque nous regardons les résultats, nous remarquons que ceux-ci sont plutôt hétérogènes entre les différentes classes. Ainsi, il serait plus judicieux d'analyser tableau par tableau. Tout d'abord, commençons par la table (a). Nous avons 94% de bien classés avec notre modèle. Cependant, il y a une très grande différence entre cette mesure et le score F1 associé qui est de 0.66. Cette différence vient du fait que pour le score F1, le nombre de vrais négatifs n'est pas pris

en compte et, ici, nous voyons que celui-ci est très élevé si nous considérons la classe "0" comme la classe positive. Même s'il est plus faible que l'exactitude, il est tout de même supérieur à 0.5 ce qui reste raisonnable. Nous avons un rappel qui est légèrement au dessus du seuil de 0.5 (0.56) et une précision de 0.78, nous pouvons donc dire que ces deux grandeurs sont assez élevées, ce qui signifie que notre modèle gère assez bien cette classe où le nombre d'anneaux est inférieur à 8.
Ensuite, si nous regardons la table (b), nous avons une exactitude de 82%, un score F1 de 0.89, une précision et un rappel élevés. Ces résultats nous montrent que notre algorithme gère correctement cette classe où le nombre d'anneaux est compris entre 8 et 14 inclus.
Pour finir, analysons la table (c). Pour notre modèle, nous constatons que le taux de bien classés est de 88% et que le score F1 associé est très différent. Contrairement à la table (a), ici, le score F1 est très inférieur au seuil de 0.5. Ce qui nous montre que notre modèle n'est pas bien adapté à cette classe sur ce jeu de données. De plus, si nous nous attardons sur la précision - qui est maximale - et sur le rappel - qui est très faible (0.03) -, nous pouvons dire que notre modèle ne peut pas toujours extraire cette classe. Cependant, s'il le fait, c'est avec un niveau de confiance élevé.

Au vu des ces résultats, nous pouvons mentionner que notre modèle est bon pour détecter la classe où le nombre d'anneaux est compris entre 8 et 14 inclus, qui correspond à la classe majoritaire. Nous avons un cas d'instances de classes déséquilibrées, ce qui pourrait expliquer ces résultats inégaux.

4 Comparaison entre les algorithmes expérimentés dans le cours

Jeu de données étudié	Iris			
Classifieur	KNN	NaifBayes	DecisionTree	RN
Temps d'exécution (s)	0.0314	**0.0065**	0.031	2.13
Temps d'apprentissage (s)	**0**	**0.0011**	0.031	2.13
Temps de prédiction d'un exemple (ms)	1.30	**0.006**	0.025	0.81
Exactitude (Accuracy)	0.97	0.94	0.93	**0.99**
Précision (Precision)	0.95	0.91	0.89	**0.98**
Rappel (Recall)	0.95	0.90	0.88	**0.98**
Score F1	0.95	0.90	0.88	**0.98**

Jeu de données étudié	Vinho verde			
Classifieur	KNN	NaifBayes	DecisionTree	RN
Temps d'exécution (s)	16.06	**0.0779**	3.925	212
Temps d'apprentissage (s)	**0**	**0.033**	3.959	212
Temps de prédiction d'un exemple (ms)	40.4	0.147	**0.118**	0.12
Exactitude (Accuracy)	0.75	0.79	**0.86**	0.84
Précision (Precision)	0.74	0.79	**0.85**	0.83
Rappel (Recall)	0.74	0.81	**0.85**	0.84
Score F1	0.74	0.80	**0.85**	0.83

Jeu de données étudié	Abalones			
Classifieur	KNN	NaifBayes	DecisionTree	RN
Temps d'exécution (s)	26.98	**0.1189**	8.275	16.5
Temps d'apprentissage (s)	**0**	**0.022**	8.814	16.5
Temps de prédiction d'un exemple (ms)	52.3	4.71	**0.17**	0.95
Exactitude (Accuracy)	**0.88**	0.63	0.83	**0.88**
Précision (Precision)	0.74	0.48	0.61	**0.87**
Rappel (Recall)	0.55	**0.65**	0.62	0.53
Score F1	0.58	0.45	**0.61**	0.54

TABLE 15 – Tableau récapitulatif des performances des algorithmes sur les trois jeux de données, où les métriques (exactitude, précision, rappel, score F1) correspondent à la moyenne de celles calculées par classe.

Pour remplir ce tableau 15, nous avons choisi de mettre les résultats de l'arbre de décision avec élagage car nous voulions mettre en avant le temps d'exécution et le temps de prédiction d'un exemple qui étaient lus courts (les métriques de performances étant très proches). De plus, les temps affichés ici pour la classification KNN ne prennent pas en compte la recherche de la meilleure valeur de K, qui dépend du choix des hyperparamètres (nombre L de validation effectuée lors de la boucle de validation croisée et valeurs de K_{min} et K_{max}). Ce sont les temps d'exécution de l'algorithme en utilisant une valeur de K déjà calculée.

De cette table, nous remarquons que plus le nombre d'instances est élevé, plus l'algorithme est long à exécuter. Nous déduisons aussi que l'algorithme naïf bayésien est bien plus rapide que les autres classificateurs, quel que soit le jeu de données étudié. Concernant le temps d'apprentissage, nous voyons que le plus rapide est aussi l'algorithme naïf bayésien sans compter la classification KNN car elle ne réalise pas vraiment d'apprentissage. À propos du temps de prédiction d'un exemple, nous observons qu'avec l'arbre de décision avec élagage le temps est plus court sur les jeux de données Abalones et Vinho verde - vin blanc. Pour Iris dataset, nous voyons que le temps de prédiction d'un exemple est le plus court l'algorithme naïf bayésien.
Cependant, en ce qui concerne les métriques de performances, les résultats obtenus sont meilleurs pour la classification des Iris avec la méthode des réseaux de neurones. Quant au jeu de données Vinho-verde, nous remarquons que la classification avec l'arbre de décision élagué permet d'obtenir de meilleurs résultats. Par contre, pour le jeu de données Abalones, nous voyons que les performances optimales sont dispachées entre les quatre classificateurs. Néanmoins, nous pouvons dire que l'algorithme naïf bayésien est le moins performant pour ce jeu de données. En effet sont score F1 est inférieur au seuil 0.5.

5 Conclusion

Tout d'abord, à travers ce projet, nous avons pu constater que les résultats que nous obtenons avec notre arbre de décision que nous avons implémenté étaient très proches de ceux issus avec la méthode de la bibliothèque `scikit-learn` voire légèrement meilleurs.
Ensuite, si nous regardons nos deux algorithmes et les deux autres que nous avons implémenté dans le TP précédent, nous avons noté que le temps d'exécution avec le classifieur Bayes naïf est nettement plus rapide qu'avec les autres. Concernant nos problèmes de classification, nous avons remarqué que la méthode des réseaux de neurones était plus performante que les autres classificateurs sur les jeux de données des fleurs (Iris dataset). Il est important de noter que pour ce jeu de données, qui ne contient que 150 instances, les résultats sont très bons et rapides à obtenir quel que soit l'algorithme, sachant que le jeu de données est plutôt équilibré. Pour Vinho verde, c'est avec l'arbre de décision que nous obtenons les meilleurs résultats. Nous devons aussi mentionner, pour le dernier jeu de données (Abalones), qu'il n'y pas de méthodes qui sortent vraiment du lot. En effet, en moyenne, nos méthodes étaient peu efficace sur ce jeu de données.

Globalement, pour ce travail tout s'est bien déroulé. Nos réflexions se sont essentiellement tournées vers le choix des hyperparamètres pour l'implémentation de l'algorithme NeuralNetwork et sur la création de l'arbre de décision et plus particulièrement son élagage.
Finalement, nous pensons nous être bien approprié ces deux algorithmes d'apprentissage et avoir bien compris leur fonctionnement.

Finalement, nous pensons nous être bien approprié ces deux algorithmes d'apprentissage et avoir bien compris leur fonctionnement.