TP3 : K plus proches voisins et classification naïve bayésienne

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
12 avril 2022

Table des matières

Liste des tableaux

Liste des Algorithmes

Table des figures

1 Introduction

Dans ce TP3, nous allons implémenter deux techniques d'apprentissage machine :

- Les **K plus proches voisins** ; et
- La **classification naïve bayésienne**.

La difficulté ici sera de programmer entièrement ces deux méthodes sans passer par des bibliothèques faisant directement le travail. Une fois que nous aurons implémenter nos deux algorithmes, nous les validerons en comparant les résultats obtenus avec ceux obtenus en utilisant des bibliothèques comme `scikit-learn`.

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'entrainement et 30% pour la phase de test. Ainsi, les modèles sont établis sur le jeu d'entrainement puis évalués sur le jeu de test. Aussi, nous avons choisi de normaliser au préalable les données de chaque attribut entre 0 et 1, pour augmenter la précision de nos résultats.

3.1 K plus proches voisins (K-nearest neighbors)

3.1.1 Métrique de distance choisie

Pour mesurer la proximité entre les observations, nous avons implémenté une fonction de similarité. Cette fonction, qui mesure la distance entre deux points, estime l'affinité entre ses deux observations. Ainsi, plus deux points sont proches l'un de l'autre, plus ils sont similaires.

Concernant la métrique de distance, nous avons choisi intuitivement la distance euclidienne car cette dernière formalise la notion de distance entre deux points qui est une ligne droite. De plus, pour nos trois jeux de données, quasiment tous les attributs sont des variables quantitatives excepté l'attribut concernant le sexe pour le dataset des abalones. Ainsi, la distance euclidienne offre un bon compromis entre la gestion des valeurs faibles et celle des valeurs élevées.

3.1.2 Pseudo-code

Algorithme 1 : Algorithme des K plus proches voisins

Entrées : K : nombre de K plus proches voisins
m : nombre d'attributs
X^{train} : matrice de taille $n_1 \times m$ avec n_1 le nombre d'individus d'entraînement
X^{test} : matrice de taille $n_2 \times m$ avec n_2 le nombre d'individus test

for $i \leftarrow 1$ **to** n_2 **do**
  for $i \leftarrow 1$ **to** n_1 **do**
    Calculer la distance entre l'observation test X_i^{test} et l'observation d'entrainement X_j^{train}.
  end
  Trier les distances calculées par ordre croissant en fonction des valeurs de distance,
  Récupérer les K premières observations d'entraînement de la matrice ordonnée,
  Extraire la classe majoritaire de ces lignes et affecter à cette classe l'observation X_i^{test}.
end

Algorithme 2 : Choix de la meilleure valeur de K

Entrées : L : nombre de sous échantillons du jeu de données initial
K_{min}
K_{max}

Nous divisons les données d'entraînemet originales en L échantillons, afin d'effectuer une validation croisée.
for $K \leftarrow K_{min}$ **to** K_{max} **do**
  for $i \leftarrow 1$ **to** L **do**
    Le i-ieme échantillon est celui correspondant aux données test.
    Les $L-1$ autres échantillons sont les données d'entraînement.
    Nous appliquons l'algorithme KNN (algorithme 1) avec ces données en utilisant la valeur K.
    Nous calculons l'exactitude entre les vrais labels et les prédictions.
  end
  Nous calculons ensuite la moyenne des exactitudes, associée à une valeur de K.
end
Nous choisissons la valeur de K qui maximise la moyenne des exactitudes.

3.1.3 À la recherche de la meilleure valeur de K, évaluation et comparaison des performances de notre modèle VS la méthode de la bibliothèque `scikit-learn`

Nous avons trois jeu de données différents, ainsi nous avons recherché la meilleure valeure de K pour chacun d'eux. Pour cette quête, nous avons utilisé la validation croisée. Puisque notre validation croisée dépend du hasard, nous l'avons fixé à 1 pour nos trois dataset.

Pour chacun des jeux de données, la classification n'est pas binaire mais multiclasses. Nous utilisons donc la méthode de un contre tous pour évaluer notre méthode. Plus précisément, nous calculons

l'exactitude, la précision, le rappel, le score F1 et la matrice de confusion, en considérant chacune des classes comme étant la classe positive et le reste des classes consitituant la classe négative.

3.1.3.1 Iris dataset Pour la recherche du meilleur K, nous avons fait varier K entre 1 et 20 et nous avons effectué une validation croisée sur $L = 5$ échantillons. Pour ce jeu de données, **la meilleure valeur de K que nous avons trouvée avec notre modèle est : 8.**

(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)	15 (VP)	0 (FN)
	1 (FP)	29 (VN)	1 (FP)	29 (FN)
L'exactitude (Accuracy)	0.96		0.98	
La précision (Precision)	0.93		0.93	
Le rappel (Recall)	0.93		1	
Le score F1	0.93		0.97	

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

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	12 (VP)	1 (FN)	12 (VP)	1 (FN)
	1 (FP)	31 (VN)	0 (FP)	32 (FN)
L'exactitude (Accuracy)	0.96		0.98	
La précision (Precision)	0.92		1	
Le rappel (Recall)	0.92		0.92	
Le score F1	0.92		0.96	

TABLE 2 – Résumé des résultats obtenus par classe avec notre modèle (KNN) 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 cet espèce d'iris, que notre modèle est aussi performant qu'avec la bibliothèque `scikit-learn`.
Concernant les espèces "Iris-versicolour" et "Iris-virginica" (cf : tables (b) et (c)), nous observons que notre modèle obtient de très bons résultats. En effet, pour les deux espèces, nous avons 96% de bien classés avec un score F1, une précision et un rappel de 92.5% en moyenne. Ceci veut dire que ces espèces sont correctement gérées par notre modèle. De plus, pour ces deux spécimens d'iris, nous remarquons qu'avec la bibliothèque `scikit-learn` les résultats sont un peu plus performants. Le taux de bien classés passe de 96% à 98% et le score F1, qui était à 92.5% de moyenne, est passé à 96%.

De manière générale, nous pouvons dire que notre modèle est très efficace pour ce jeu de données, même si les performances avec la méthode de la bibliothèque `scikit-learn` sont légèrement plus élevées.

3.1.3.2 Vinho verde - vin blanc Pour la recherche du meilleur K, nous avons fait varier K entre 1 et 10 et nous avons effectué une validation croisée sur $L = 20$ échantillons. Pour ce jeu de données, **la meilleure valeur de K que nous avons trouvée avec notre modèle est : 1.**

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

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	386 (VP)	109 (FN)	392 (VP)	103 (FN)
	92 (FP)	224 (VN)	75 (FP)	241 (FN)
L'exactitude (Accuracy)	0.75		0.78	
La précision (Precision)	0.81		0.84	
Le rappel (Recall)	0.78		0.79	
Le score F1	0.79		0.81	

(b) En considérant la classe "1" (bon : qualité ≥ 5) comme la classe positive

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	224 (VP)	92 (FN)	241 (VP)	75 (FN)
	109 (FP)	386 (VN)	103 (FP)	392 (FN)
L'exactitude (Accuracy)	0.75		0.78	
La précision (Precision)	0.67		0.70	
Le rappel (Recall)	0.71		0.76	
Le score F1	0.69		0.73	

TABLE 3 – Résumé des résultats obtenus avec notre modèle (KNN) 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 (75%). Cependant, si nous nous attardons sur les autres 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 0.79 tandis que pour la classe "1", nous avons 0.69. 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 à peine supérieurs à ceux de notre modèle. Précisément, pour les deux classes, nous obtenons un taux de bien classés de 78% avec un score F1 de 0.77 en moyenne. Nous pouvons aussi émettre la même observation que nous avons faite sur notre modèle quant à la performance entre nos différentes classes.

De manière générale, nous pouvons dire que notre modèle est plutôt efficace pour ce jeu de données, même si les performances avec la méthode de la bibliothèque `scikit-learn` sont légèrement plus élevées.

3.1.3.3 Abalones Pour la recherche du meilleur K, nous avons fait varier K entre 1 et 30 et nous avons effectué une validation croisée sur $L = 10$ échantillons. Pour ce jeu de données, **la meilleure valeur de K que nous avons trouvée avec notre modèle est : 18.**

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

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	76 (VP)	57 (FN)	82 (VP)	51 (FN)
	28 (FP)	1093 (VN)	32 (FP)	1089 (FN)
L'exactitude (Accuracy)	0.93		0.93	
La précision (Precision)	0.73		0.72	
Le rappel (Recall)	0.57		0.62	
Le score F1	0.64		0.66	

(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	932 (VP)	38 (FN)	929 (VP)	41 (FN)
	188 (FP)	96 (VN)	186 (FP)	98 (FN)
L'exactitude (Accuracy)	0.82		0.82	
La précision (Precision)	0.83		0.83	
Le rappel (Recall)	0.961		0.958	
Le score F1	0.89		0.89	

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

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	20 (VP)	131 (FN)	16 (VP)	135 (FN)
	10 (FP)	1093 (VN)	9 (FP)	1094 (FN)
L'exactitude (Accuracy)	0.89		0.89	
La précision (Precision)	0.67		0.64	
Le rappel (Recall)	0.13		0.11	
Le score F1	0.22		0.18	

TABLE 4 – Résumé des résultats obtenus avec notre modèle (KNN) 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 93% 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 0.64. 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 (0.57) et une précision de 0.73, 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), nous remarquons de suite que notre modèle fait aussi bien que la méthode de la bibliothèque `scikit-learn`. En effet, nous obtenons les mêmes résultats. 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 89% 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 supérieure au seuil de 0.5 (0.67) et que nous pouvons considérer comme plutôt élevée - et sur le rappel - qui est très faible (0.13) -, 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. À présent, si nous considérons les résultats issus de la méthode avec la bibliothèque `scikit-learn`, nous pouvons dire qu'ils sont quasiment similaires à ceux obtenus avec notre modèle, mais nous pouvons noter tout de même que nos résultats sont légèrement meilleurs.

De manière générale, nous pouvons dire que notre modèle fait aussi bien qu'avec la méthode de la bibliothèque `scikit-learn`, même si les performances sont peu notables sur ce jeu de données. Par contre, nous pouvons mentionner, dans les deux cas, que les résultats sont plutôt bons 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.

3.2 Classification Naïve Bayésienne

3.2.1 Pseudo-code

Algorithme 3 : Algorithme de la classification naïve bayésienne

Entrées : m : nombre d'attributs
X_{train} : matrice de taille $n_1 \times m$ avec n_1 le nombre d'individus d'entraînement
X_{test} : matrice de taille $n_2 \times m$ avec n_2 le nombre d'individus test

Séparer le jeu de données d'entraînement par classe.
for $j \leftarrow 1$ **to** m **do**
  Calculer la moyenne μ_j^{classe} et l'écart-type σ_j^{classe} de chaque attribut j pour chaque classe.
end
for $j \leftarrow 1$ **to** n_2 **do**
  for *chaque classe* **do**
    Calculer la probabilité de f_i^{classe}. L'hypothèse d'indépendance des attributs nous permet de la calculer avec le produit des probabilités en utilisant la loi de densité gaussienne sur chaque attribut.

$$f_i^{classe} = \prod_{j=1}^{m} \frac{1}{\sigma_j^{classe}\sqrt{2\pi}} \, \mathrm{e}^{-\frac{1}{2}\left(\frac{i - \mu_j^{classe}}{\sigma_j^{classe}}\right)^2}$$

  end
  Retenir la classe avec la probabilité la plus grande.
end

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

3.2.2.1 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)	14 (VP)	1 (FN)
	3 (FP)	27 (VN)	3 (FP)	27 (FN)
L'exactitude (Accuracy)	0.91		0.91	
La précision (Precision)	0.82		0.82	
Le rappel (Recall)	0.93		0.93	
Le score F1	0.87		0.87	

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

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	10 (VP)	3 (FN)	10 (VP)	3 (FN)
	1 (FP)	31 (VN)	1 (FP)	31 (FN)
L'exactitude (Accuracy)	0.91		0.91	
La précision (Precision)	0.91		0.91	
Le rappel (Recall)	0.77		0.77	
Le score F1	0.83		0.83	

TABLE 5 – Résumé des résultats obtenus par classe avec notre modèle (Bayes Naïf) et avec la méthode de la bibliothèque `scikit-learn` 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. Nous pouvons aussi dire, pour cet espèce d'iris, que notre modèle est aussi performant qu'avec la bibliothèque `scikit-learn`.
Concernant les espèces "Iris-versicolour" et "Iris-virginica" (cf : tables (b) et (c)), nous observons que notre modèle obtient de très bons résultats. En effet, pour les deux espèces, nous avons 91% de bien classés avec un score F1, une précision et un rappel très élevés. Ceci veut dire que ces espèces

sont correctement gérées par notre modèle. De plus, pour ces deux spécimens d'iris, nous remarquons qu'avec la bibliothèque `scikit-learn` les résultats identiques aux nôtres.

De manière générale, nous pouvons dire que notre modèle est très performants pour ce jeu de données et est équivalent avec la méthode de la bibliothèque `scikit-learn`.

3.2.2.2 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	364 (VP)	131 (FN)	380 (VP)	115 (FN)
	39 (FP)	277 (VN)	43 (FP)	273 (FN)
L'exactitude (Accuracy)	0.79		0.81	
La précision (Precision)	0.90		0.90	
Le rappel (Recall)	0.74		0.77	
Le score F1	0.81		0.83	

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

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	277 (VP)	39 (FN)	273 (VP)	43 (FN)
	131 (FP)	364 (VN)	115 (FP)	380 (FN)
L'exactitude (Accuracy)	0.79		0.81	
La précision (Precision)	0.68		0.70	
Le rappel (Recall)	0.88		0.86	
Le score F1	0.77		0.78	

TABLE 6 – Résumé des résultats obtenus avec notre modèle (Bayes Naïf) et avec la méthode de la bibliothèque `scikit-learn` sur Vinho verde - vin blanc

Sur cette table 6, nous pouvons noter que le taux de bien classés, avec notre modèle, est identique pour les deux classes (79%). Cependant, si nous nous attardons sur les autres 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 0.81 tandis que pour la classe "1", nous avons 0.77. 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 à peine supérieurs à ceux de notre modèle. Précisément, pour les deux classes, nous obtenons un taux de bien classés de 81% contre 79% pour notre modèle et avec un score F1 de 0.805 en moyenne contre 0.79 en moyenne pour notre modèle. Nous pouvons aussi émettre la même observation que nous avons faite sur notre modèle quant à la performance entre nos différentes

classes.

De manière générale, nous pouvons dire que notre modèle est plutôt efficace pour ce jeu de données, même si les performances avec la méthode de la bibliothèque `scikit-learn` sont légèrement plus élevées.

3.2.2.3 Abalones

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

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	124 (VP)	9 (FN)	121 (VP)	12 (FN)
	188 (FP)	933 (VN)	164 (FP)	957 (FN)
L'exactitude (Accuracy)	0.84		0.86	
La précision (Precision)	0.40		0.42	
Le rappel (Recall)	0.93		0.91	
Le score F1	0.56		0.58	

(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	339 (VP)	631 (FN)	512 (VP)	458 (FN)
	53 (FP)	231 (VN)	84 (FP)	957 (FN)
L'exactitude (Accuracy)	0.45		0.57	
La précision (Precision)	0.86		0.86	
Le rappel (Recall)	0.35		0.53	
Le score F1	0.50		0.65	

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

	Notre modèle		Avec la bibliothèque `scikit-learn`	
Matrice de confusion	102 (VP)	49 (FN)	74 (VP)	77 (FN)
	448 (FP)	655 (VN)	299 (FP)	804 (FN)
L'exactitude (Accuracy)	0.60		0.70	
La précision (Precision)	0.19		0.20	
Le rappel (Recall)	0.68		0.49	
Le score F1	0.29		0.28	

TABLE 7 – Résumé des résultats obtenus avec notre modèle (Bayes Naïf) et avec la méthode de la bibliothèque `scikit-learn` sur les abalones

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 84% 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 0.56. 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 très élevé (0.93) et une précision légèrement inférieure au seuil de 0.5 (0.4), ce qui est plutôt faible. Au vu de ces deux résultats, nous pouvons dire que notre modèle extrait cette classe correctement. Toutefois, elle présente un faible taux de confiance, car elle est parfois extraite comme un autre type. 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 45%, ce qui est assez faible. De plus, nous avons un score F1 de 0.5, ceci signifie que pour une prédiction positive correcte, le modèle fait deux erreurs (Faux négatif ou Faux positif). Néanmoins, nous avons aussi un rappel faible (0.35) et une précision élevée (0.86). De ces grandeurs, nous pouvons dire que notre algorithme ne peut pas toujours extraire cette classe. Si elle est extraite, elle l'est avec un niveau de confiance élevé. Par contre, si nous étudions les performances de la méthode avec la bibliothèque `scikit-learn`, nous observons qu'il y a une différence significative. En effet, nous avons une exactitude de 57% contre 45%, un score F1 de 0.65 contre 0.5. De plus, contrairement à notre modèle, la précision et le rappel sont tous supérieurs au seuil de 0.5, nous pouvons les considérer comme élevés et ceci veut dire que cette classe est correctement gérée par la bibliothèque. Au vu de tous ses résultats, nous pouvons dire que la méthode avec la bibliothèque `scikit-learn` est plus performante que notre modèle pour cette classe.
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 60% 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 très adapté pour cette classe sur ce jeu de données. De plus, si nous nous attardons sur le rappel - qui est supérieur au seuil de 0.5 (0.68) et que nous pouvons considérer comme plutôt élevé - et sur la précision - qui est très faible (0.13)-, nous pouvons dire que notre modèle extrait cette classe correctement. Toutefois, elle présente un faible taux de confiance, car elle est parfois extraite comme un autre type. À présent, si nous considérons les résultats issus de la méthode avec la bibliothèque `scikit-learn`, nous voyons que le taux de bien classés est de 70% mais que le score F1 est très faible (0.28). De plus, nous constatons aussi que la précision et le rappel sont inférieurs au seuil 0.5. Nous pouvons donc dire qu'ils sont faibles. Ceci signifie que cette classe est mal gérée par la bibliothèque, car elle n'est généralement pas extraite. Lorsque c'est le cas, elle ne présente pas un niveau de confiance élevé. Ainsi, au vu de ces résultats, nous pouvons dire que notre modèle fait mieux que la méthode de la bibliothèque `scikit-learn` pour cette classe.

De manière générale, nous pouvons dire que notre modèle n'est pas très adapté pour ce jeu de données. Il en est de même avec la méthode de la bibliothèque `scikit-learn`.

4 Comparaison des performances des deux algorithmes

Jeu de données étudié	Iris		Vinho verde		Abalones	
Classifieur	KNN	Bayes naïf	KNN	Bayes naïf	KNN	Bayes naïf
Temps d'exécution (secondes)	0.0314	**0.0065**	16.06	**0.0779**	26.98	**0.1189**
Exactitude (Accuracy)	**0.97**	0.94	0.75	**0.79**	**0.88**	0.63
Précision (Precision)	**0.95**	0.91	0.74	**0.79**	**0.74**	0.48
Rappel (Recall)	**0.95**	0.90	0.74	**0.81**	**0.55**	0.65
Score F1	**0.95**	0.90	0.74	**0.80**	**0.58**	0.45

TABLE 8 – 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

De ce tableau 8, 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 la classification KNN, quel que soit le jeu de données étudié. Ceci s'explique car l'algorithme KNN est plus gourmant lors de la phase d'évaluation et ne réalise par réellement d'appprentissage. De plus, les temps affichés ici 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. Cependant, en ce qui concerne les métriques de performances, les résultats obtenus sont meilleurs pour la classification des Iris et la classification des Abalones avec la méthode des K plus proches voisins. Quant au jeu de données Vinho-verde, nous remarquons que la classification naïve bayésienne permet d'obtenir de meilleurs résultats. Or, avec le classifieur naïf bayésien, nous émettons l'hypothèse que tous les attributs sont indépendants. Il serait donc possible que les attributs du jeu de données Vinho-verde soient des variables quantitatives indépendantes, contrairement aux deux autres jeux de données qui incluraient des attributs dépendants.

5 Conclusion

Tout d'abord, à travers ce projet, nous avons pu constater que les résultats que nous obtenons avec les algorithmes que nous avons implémentés étaient très proches de ceux issus avec les méthodes de la bibliothèque `scikit-learn`.
Ensuite, si nous regardons nos deux algorithmes, nous avons noté que le temps d'exécution avec le classifieur Bayes naïf est nettement plus rapide qu'avec le classifieur KNN. Concernant nos problèmes de classification, nous avons remarqué que la méthode des K plus proches voisins était plus performante que la classification naïve bayésienne sur les jeux de données des fleurs (Iris dataset) et des ormeaux (Abalones). Nous devons aussi mentionner, pour le dernier jeu de données (Abalones), que même si les résultats sont meilleurs avec la méthode des KNN, il semble que notre modèle a tout de même des difficultés (score F1 légèrement au dessus du seuil de 0.5). Quant au jeu de données sur le vin, c'est avec la méthode Bayes naïf que les résultats sont les meilleurs.

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

6 Annexes

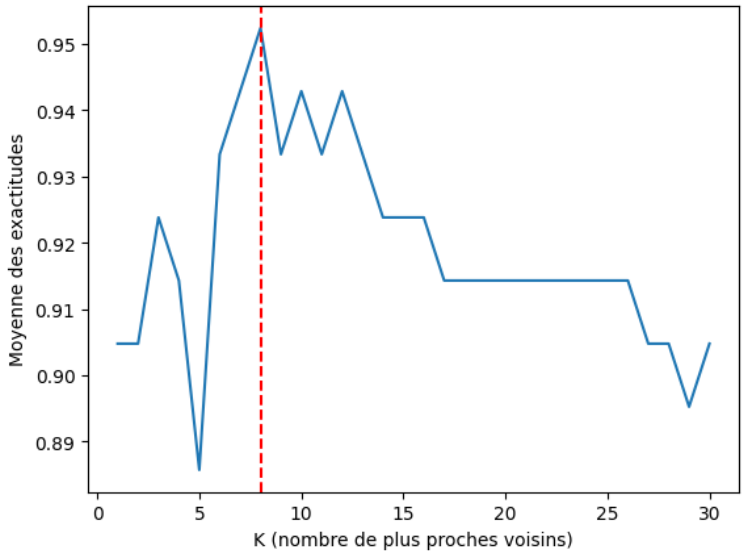

FIGURE 1 – Moyenne des exactitudes en fonction des valeurs de K pour le jeu de données iris

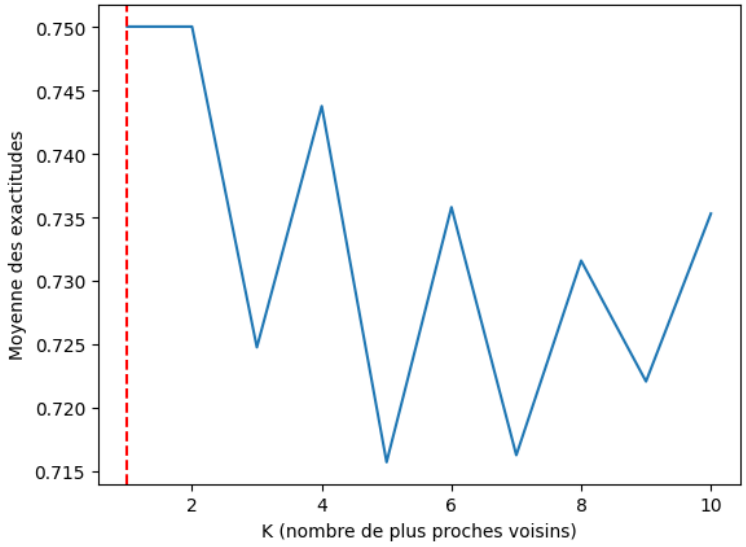

FIGURE 2 – Moyenne des exactitudes en fonction des valeurs de K pour le jeu de données vins

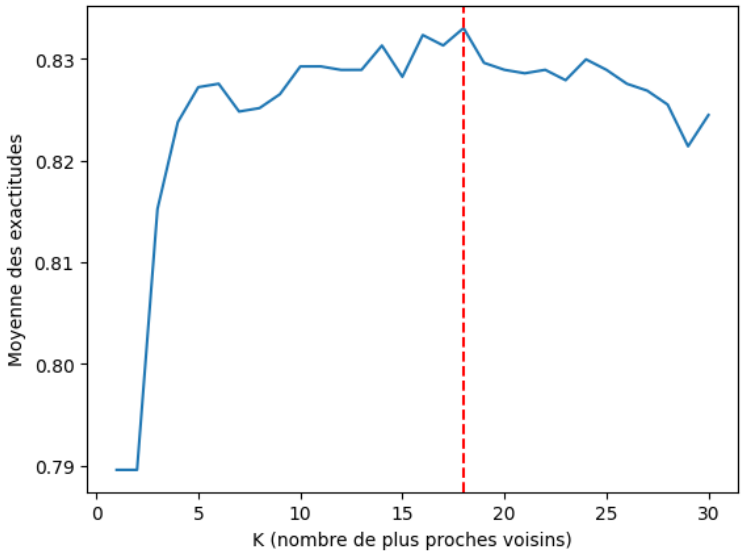

FIGURE 3 – Moyenne des exactitudes en fonction des valeurs de K pour le jeu de données abalone