Projet partiel

Travail remis à M. Brahim Chaib-draa
dans le cadre du cours
IFT-4102 - *Techniques avancées en intelligence artificielle*

par

Jérémie Gince (111 224 046)
jeremie.gince.1@ulaval.ca

Richard Paré (111 158 243)
richard.pare.3@ulaval.ca

Département d'informatique et de génie logiciel
Faculté des sciences et de génie, Université Laval
30 mars 2020

1 Introduction

Dans ce rapport, il est question de comparer deux techniques de classification: *K plus proches voisins* et la *Classification Naïve Bayésienne*. Les deux algorithmes seront testés sur cinq ensembles de données : *Iris*, *Congressional*, *Monks (1)*, *Monks (2)* et *Monks (3)*. Afin de comparer les deux techniques d'apprentissage machine, les tests de performance suivants ont été utilisés: la matrice de confusion, la précision, le rappel et l'exactitude.

2 Ensembles de données

Les ensembles de données *Iris*, *Congressional*, *Monks (1)*, *Monks (2)* et *Monks (3)* sont caractérisés comme suit. L'ensemble de données *Iris* est constitué de 150 vecteurs de caractéristiques de fleurs iris considéré comme tout de même simple à solutionner. L'ensemble de données *Congressional* est constitué de 435 vecteurs de caractéristiques qui représentent des votes politiques. Pour ce qui est des ensembles de données *Monks (1)*, *Monks (2)* et *Monks (3)*, ce sont des ensembles constitués de 556, 601 et 554 vecteurs de caractéristiques respectivement construites à des fins pédagogiques.

Afin de savoir qu'elle est le balancement de chacun ensemble de donnée, l'entropie de Shannon a été utilisée sur ceux-ci. Elle est définie comme suit pour un ensemble de données contenant n instances, k classes de grandeur c_i.

$$H = -\sum_{i}^{k} \frac{c_i}{n} \cdot \log\left(\frac{c_i}{n}\right)$$

Dans le cas où $H \to 0$, cela signifie que les classes sont mal balancées. En effet, afin d'avoir un balancement maximum, il faut que l'entropie soit maximale. L'entropie doit être maximale puisque cela signifiera qu'il est maximalement difficile d'extraire de l'information sur le balancement des classes. En d'autres mots, si l'entropie est élevée, il est difficile de faire des prédictions aléatoires qui sont justes.

On peut donc définir un facteur de balancement nommé β comme suit sachant que l'entropie de Shannon est maximale à $\log(k)$ dans le cas présent:

$$\beta = \frac{H}{\log(k)}$$
$$\implies \beta = -\frac{1}{\log(k)} \sum_{i}^{k} \frac{c_i}{n} \cdot \log\left(\frac{c_i}{n}\right)$$

Avec cette représentation on peut facilement quantifier le balancement des classes. En effet, quand $\beta = 1$ les données sont maximalement balancées et quand $\beta = 0$, les données ne sont aucunement balancées.

En appliquant le calcul de β sur chacun des ensembles de données, nous obtenons les résultats suivants:

Dataset	β
Iris	1.000
Congressional	0.962
Monks (1)	1.000
Monks (2)	0.927
Monks (3)	0.999

Table 1: Facteur β pour chacun des ensembles de données

Avec les résultats du tableau 1, on peut rapidement voir les ensembles de données utilisés sont relativement bien balancés. Toutefois, le facteur β de *Monks (2)* est de 0.927 ce qui le facteur minimal de tous. En effet, ce facteur est relativement bas par rapport aux autres et le nombre d'instances pour chaque classe de cet ensemble est [395, 206], ce qui signifie que cet ensemble de données n'est pas très bien balancé. En effet, il y a deux fois plus d'instances d'une classe que dans l'autre. On peut donc s'attendre à de moins bons résultats lors de la classification puisqu'on n'effectue aucun rebalancement des données dans ce travail.

3 Définition des tests d'évaluations de performances

3.1 Confusion matrix

En classification la matrice de confusion est très utilisée afin de s'assurer que nos modèles supervisés ne confondent pas certaines classes ensemble. La matrice de confusion est essentiellement un outil de visualisation de la confusion de notre modèle comme son nom le suggère.

Elle se définit comme suit:

	Classes actuelles				
Classes prédites	Classe 0	Classe 1	Classe 2	...	Classe n
Classe 0	k_0	$g_{0,1}$	$g_{0,2}$		$g_{0,n}$
Classe 1	$g_{1,0}$	k_1	$g_{1,2}$		$g_{1,n}$
Classe 2	$g_{2,0}$	$g_{2,1}$	k_2		$g_{2,n}$
$\vdots$				$\ddots$	
Classe n	$g_{n,0}$	$g_{n,1}$	$g_{n,2}$		k_n

Table 2: Matrice de confusion pour n classes

Dans cette dernière les k_i sont les vrais positifs de la classe i et les $g_{i,j}$ sont les fausses attributions de la classe i à la classe j. Avec une telle matrice, nous pouvons voir que plus cette matrice est diagonale, plus le modèle est précis et dans le cas contraire, alors plus le modèle est confus.

Référence: Confusion Matrix link

3.2 Precision

La précision est utile en classification afin de connaître le ratio entre le nombre de prédictions correctement alloué à une certaine classe et le nombre de prédictions alloué à cette même classe.

$$precision_i = \frac{k_i}{k_i + \sum_{j=0}^{n} g_{i,j}\delta_{i,j}}$$

où $\delta_{i,j}$ est le delta de Kronecker pour les indices i et j.

Dans le cas où nous avons une classification multiclasses nous pouvons nous intéresser à la précision moyenne que l'on peut calculer ainsi:

$$\langle precision \rangle = \frac{\sum_i^n precision_i}{n}$$

3.3 Recall

Le rappel est utile en classification afin de connaître la sensibilité de prédiction du modèle en calculant le ratio entre le nombre de prédictions correctement alloué à une certaine classe et le nombre d'étiquettes alloué à cette même classe.

$$recall_i = \frac{k_i}{k_i + \sum_{j=0}^{n} g_{j,i}\delta_{i,j}}$$

où $\delta_{i,j}$ est le delta de Kronecker pour les indices i et j.

Dans le cas où nous avons une classification multiclasses nous pouvons nous intéresser à le rappel moyenne que l'on peut calculer ainsi:

$$\langle recall \rangle = \frac{\sum_i^n recall_i}{n}$$

4 Techniques d'apprentissage automatique à développer

4.1 Classification Knn - K plus proches voisins

4.1.1 Implémentation

L'algorithme de classification des K plus proches voisins est simple, mais efficace. En effet, sa force ne vient pas de sa complexité, mais bien de son choix de métrique afin de calculer les distances entre les divers points de l'ensemble donnés. Voici un court résumé de l'approche.

Dans cette méthode, nous prenons un ensemble de données disons T qui est constitué d'un uplet de longueur deux sous la forme: (vecteur de caractéristiques, étiquette). Le vecteur de caractéristique est un uplet de longueur n constitué de n caractéristiques représentant une certaine donnée. Nous allons nommé ce uplet v_i pour la donnée i. Maintenant, l'algorithme des K plus proches voisins aura pour plus de classifier un exemple de données nommées μ étant lui-même un vecteur de caractéristiques. Il faudra donc simplement calculer toutes les distances D entre tous les vecteurs v_i de l'ensemble T et μ. Par la suite, on prend les k vecteurs v_i (l'ensemble V_n constitue les k étiquettes accordées à ces k vecteurs) les plus proches de μ selon la métrique D et ainsi considérer que l'étiquette accordée à μ est l'étiquette se retrouvant le plus dans l'ensemble V_n.

Voici le pseudo-code de l'algorithme Knn.

```
def predict(exemple, label):
    let train_data be a map in form: {sample_i, label_i}
    let neighbors be a list

    WHILE train_data NOT empty DO
        sample_i, label_i = train_data.popitem()
        distance = D[example, sample_i]
        neighbors.ADD((sample_i, distance, label_i))
    END WHILE

    let nearest_labels be a list of the K nearest labels of neighbors
    let nearest_labels_count be a map in form: {label_i: count of the label_i in nearest_labels}

    let prediction_label be the label with the maximum value of nearest_labels_count
    return prediction_label, prediction_label == label
```

Dans le présent projet, la métrique D utilisée est la distance euclidienne.

$$euclidean\_distance = \sqrt{\sum_i (x_i - x'_i)^2}$$

Ce choix a été considéré comme le plus approprié puisque cette métrique donnait de meilleurs résultats que la distance de Manhattan

$$manhattan\_distance = \sum_i |x_i - x'_i|$$

et la distance de Chebychev.

$$chebychev\_distance = max(|x_i - x'_i|)$$

Par contre, le choix de prendre la distance euclidienne comme métrique n'est certainement pas le plus optimal. En effet, le plus optimal serait de prendre une distance de Minkowski pondéré.

$$weighted\_minkowski\_distance = \left(\sum_i w_i (x_i - x'_i)^p \right)^{1/p}$$

Nous aurions pu trouver les paramètres p et w_i les plus optimaux en entraînant notre Knn en pré processus et ainsi augmenter notre précision lors de nos classifications. La recherche des paramètres optimaux aurait pu se faire à l'aide d'une descente de gradients lors de l'apprentissage avec l'ensemble des données d'entraînement. Dans le contexte actuel, la contrainte de temps c'est avérée fatale pour cette partie d'optimisation du projet. Ces optimisations sont donc retenues pour d'éventuels changements dans la prochaine version du projet.

De plus, l'utilisation de la validation croisée a été effectuée afin de trouver le paramètre K optimal pour chacun des ensembles de données. La démarche va comme suit. Considérons un ensemble de données T que l'on sépare en cv sections (par défaut $cv = 5$). Par la suite nous copions ces sections cv fois ce qui donne un tenseur représenté par la matrice ci-bas.

$s(0,0)$	$s(0,1)$	$s(0,2)$	$\dots$	$s(0,cv)$
$s(1,1)$	$s(1,1)$	$s(1,2)$	$\dots$	$s(1,cv)$
$s(2,1)$	$s(2,1)$	$s(2,2)$	$\dots$	$s(2,cv)$
$\vdots$			$\ddots$	$\vdots$
$s(cv,0)$	$s(cv,1)$	$s(cv,2)$	$\dots$	$s(cv,cv)$

Table 3: Représentation visuelle de la fragmentation en sections $s(i,j)$ de l'ensemble de données T

Par la suite, on pose les sections $s(i,j)$ où $i = j$ comme étant les sections de test et les sections $s(i,j)$ où $i \neq j$ comme étant les sections d'entraînement. Par la suite, on fait les cv entraînements pour un K donné et on garde la moyenne des *exactitudes* en mémoire associée à ce K. Une fois fait, on reproduit cette démarche pour plusieurs K, généralement de K_{min} à K_{max} et on prend le K ayant le maximum d'*exactitude* comme étant notre K optimal et ainsi se termine la validation croisée.

4.1.2 Résultats

Lors de la classification des ensembles de données *Iris*, *Congressional*, *Monks (1)*, *Monks (2)* et *Monks (3)*, les résultats obtenus avec l'algorithme des *K plus proches voisins* sont les suivants:

Accuracy (%)				
Dataset				
Iris	Congressional	Monks (1)	Monks (2)	Monks (3)
100	100	75.93	69.68	87.27

Table 4: Exactitude pour le Knn avec un train ratio de 0.9

Iris			Congressional		Monks (1)		Monks (2)		Monks (3)	
Classe 0	Classe 1	Classe 2	Classe 0	Classe 1	Classe 0	Classe 1	Classe 0	Classe 1	Classe 0	Classe 1
100.0	100	100.0	100	100	74.14	78.00	79.34	53.42	80.16	96.76

Table 5: Precision pour le Knn avec un train ratio de 0.9

Ici, on peut voir que les résultats de classification semblent bien élevés à l'exception de la classification de *Monks (1)* et *Monks (2)*. Ces résultats sont élevés à cause de la robustesse de l'algorithme présent toutefois, les résultats moins élevés peuvent être dus à plusieurs facteurs. En effet, si les données tests sont différentes des données d'entraînement, il est alors difficile de récolter suffisamment d'information pour la classification. En effet, comme

Iris			Congressional		Monks (1)		Monks (2)		Monks (3)	
Classe 0	Classe 1	Classe 2	Classe 0	Classe 1	Classe 0	Classe 1	Classe 0	Classe 1	Classe 0	Classe 1
100.0	100	100.0	100	100	79.63	72.22	74.14	60.56	97.06	78.51

Table 6: Rappel pour le Knn avec un train ratio de 0.9

Dataset	Matrice de confusion
Iris	$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 6 \end{bmatrix}$
Congressional	$\begin{bmatrix} 19 & 0 \\ 0 & 25 \end{bmatrix}$
Monks (1)	$\begin{bmatrix} 172 & 60 \\ 44 & 156 \end{bmatrix}$
Monks (2)	$\begin{bmatrix} 215 & 56 \\ 75 & 86 \end{bmatrix}$
Monks (3)	$\begin{bmatrix} 198 & 49 \\ 6 & 179 \end{bmatrix}$

Table 7: Matrice de confusion pour l'ensemble test pour tous les ensembles de données avec un train ratio de 0.9

Dataset	K
Iris	10
Congressional	7
Monks (1)	14
Monks (2)	3
Monks (3)	6

Table 8: K optimal trouvé selon la validation croisée avec un train ratio de 0.9

on peut le voir avec les matrices de confusions du tableau 7 pour les ensembles de données *Monks (1)* et *Monks (2)*, il semble que le modèle confond les deux classes ensemble relativement souvent puisque les éléments (0,1) et (1,0) sont approximativement la moitié des valeurs des éléments sur la diagonale.

Pour ce qui est de l'optimisation du paramètre K, il semble que les ensembles de données *Monks (1)* et *Monks (2)* possèdent les K aux extremums des K présents comme on peut le constater au tableau 8. Cela veut dire que la densité des des points des ensembles de données sont soit très dense ou très clairsemé (opposé de dense). L'hypothèse faite ici est que le K de l'ensemble *Monks (1)* est très clairsemé et alors, il est très difficile de générer une bonne prédiction avec un K petit puisque beaucoup de données sont à des positions relatives très petites. c'est à ce moment qu'il serait nécessaire d'utiliser une métrique de Minkowski pondéré comme présenté précédemment, afin que le modèle soit capable d'apprendre à générer un hyperespace possédant des sections plus corrélées avec les étiquettes des données et ainsi réduire le paramètre K avec la validation croisée et finalement augmenter notre précision lors des prédictions. Idem pour ce qui en est de l'ensemble de données *Monks (2)* qui est à l'opposé trop dense, mais génère le même problème de distances relatives trop petites et requière alors une fonction de distance non linéaire afin d'être capable de mieux séparer l'hyperespace.

Comme mentionné implicitement l'absence de bons résultats pour nos deux ensembles de données problématiques est dû à la linéarité de notre fonction de distance. En effet, selon nos données la classification requière une fonction de plus en plus non linéaire plus la fonction $f(v_i) = label_i$ est considérée comme non linéaire. Bref, plus notre fonction de distance aura un pouvoir d'expression élevé, plus nous pouvons espérer avoir de bons résultats face à la classification de nos ensembles de données. Dans le cas actuel, le pouvoir d'expression de la distance euclidienne semble suffisant pour les ensembles de données *Iris*, *Congressional* et *Monks (3)* mais semble en manquer pour les deux autres.

Pour ce qui est du choix du ratio de données d'entraînement, celui-ci a été initialisé à 0.9. Un tel ratio a été nécessaire afin de générer les meilleurs résultats possible. En effet, nous pouvons voir à l'aide su tableau 17 que ce ratio génère des meilleurs résultats que l' autre. Une explication assez simple est derrière cette affirmation; l'algorithme du Knn a besoin du maximum de donnée d'entraînement afin d'avoir suffisamment d'information pour ses prédictions. En effet, une des forces du Knn est sa robustesse qui vient essentiellement de son nombre élevé de données d'entraînement et de sa diversité de données qui peut venir seulement avec un grand ensemble de données.

5 Classification Naïve Bayésienne

5.1 Implémentation

L'algorithme de classification naïve bayésienne est basé sur le théorème de Bayes, un résultat important en probabilité.

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{1}$$

La technique est basé sur la probabilité jointe de chacune des classes C_i et des caractéristiques F_i. Cette probabilité s'exprime comme suit

$$P(C_i|F_1, ..., F_n) \times \alpha \times P(C_i, F_1, ...F_n) = P(C_i) \times \prod_{i=1}^{n} P(F_i|C_i) \tag{2}$$

Dans la phase d'apprentissage, l'algorithme va créer 2 structures de données de stockage d'information. L'une d'entre elle est la distribution probabiliste de chacune des classes qui composent le problème de classification. Soit

$$P(C_i) \tag{3}$$

La deuxième structure de donnée contiendra les probabilités conditionnelles. Soit

$$P(F_i|C_i) \tag{4}$$

Ou F_i est la ième caractéristique.
Ensuite, pour faire une prédiction nous chercherons la classe qui maximise la probabilité jointe.

Voici le pseudocode de la classification bayésienne.

```
//Parameters
//Will contain the distribution of each class in the training dataset
probability_of_each_class = {}
//Will contain the conditional probability for each feature and each class
probability_conditional = []

def train(self, train, train_labels):
        FOR each class
                //add the count of each unique class in train_labels to probability_of_each_class
        END LOOP

        FOR each feature
                //add the conditional probability of each feature in probability_conditional
        END LOOP

def predict(self, exemple, label):
        calculated_conditional_probability = [[]]
        total_calculated_probability_for_each_class = []
        FOR each feature
                FOR each class
                        calculated_conditional_probability[class][feature] =
            probability_conditional[class][feature]
```

```
        END LOOP
END LOOP

FOR each class
        total_calculated_probability_for_each_class[class] = probability_of_each_class[class]*
product(calculated_conditional_probability[class])
END LOOP

predicted_label = max(total_calculated_probability_for_each_class)

return predicted_label
```

5.2 Résultats

Lors de la classification des ensembles de données *Iris*, *Congressional*, *Monks (1)*, *Monks (2)* et *Monks (3)*, les résultats obtenus avec l'algorithme de *classification naïve bayésienne* sont les suivants:

Iris			Congressional		Monks (1)		Monks (2)		Monks (3)	
Classe 0	Classe 1	Classe 2	Classe 0	Classe 1	Classe 0	Classe 1	Classe 0	Classe 1	Classe 0	Classe 1
100.0	95.0	100.0	91.89	92.0	76.39	66.2	84.14	15.49	100.0	94.74

Table 9: Résultat du rappel de chacun des datasets à 60% de ratio d'entraînement

Iris			Congressional		Monks (1)		Monks (2)		Monks (3)	
Classe 0	Classe 1	Classe 2	Classe 0	Classe 1	Classe 0	Classe 1	Classe 0	Classe 1	Classe 0	Classe 1
100.0	100.0	94.74	89.47	93.88	69.33	73.71	67.03	32.35	94.44	100.0

Table 10: Résultat de la précision de chacun des datasets à 60% de ratio d'entraînement

Accuracy (%)				
Dataset				
Iris	Congressional	Monks (1)	Monks (2)	Monks (3)
98.33	91.95	71.30	61.57	97.22

Table 11: Résultat de l'exactitude de chacun des datasets à 60% de ratio d'entraînement

Voici les matrices de confusion pour chacun des jeux de données.

	Classe 0	Classe 1	Classe 2
Classe 0	22	0	0
Classe 1	0	19	0
Classe 2	0	1	18

Table 12: Matrice de confusion pour le dataset Iris

	Classe 0	Classe 1
Classe 0	68	8
Classe 1	6	92

Table 13: Matrice de confusion pour le dataset Congressional

	Classe 0	Classe 1
Classe 0	165	73
Classe 1	51	143

Table 14: Matrice de confusion pour le dataset Monks 1

	Classe 0	Classe 1
Classe 0	244	120
Classe 1	46	22

Table 15: Matrice de confusion pour le dataset Monks 2

	Classe 0	Classe 1
Classe 0	204	12
Classe 1	0	216

Table 16: Matrice de confusion pour le dataset Monks 3

Cette méthode obtient une moyenne d'accuracy de 95.8% sur les datasets Iris, Congressional et Monks(3). Cette moyenne est de 66.4% sur les datasets Monks(1) et Monks(2). Il ne faut pas oublier que la méthode de classification naïve bayésienne suppose que les distributions probabilistes des caractéristiques sont indépendantes. Nous pourrions donc poser l'hypothèse que dans ces datasets, la distribution d'une caractéristique en influence une autre.

6 Comparaison

En observant le tableau des résultats (17), nous observons que le KNN est généralement meilleur en termes d'exactitude que le NBC. Par contre, le KNN peut prendre 10 à 100 fois plus de temps à entraîner. Les 2 méthodes semblent toutefois moins bien performer sur les ensembles de données Monks (1) et Monks (2). Le KNN semble mieux réussir sur ceux-ci, respectivement 4.63 et 8.11% meilleur en termes d'accuracy. Par contre, pour Monks (3), le NBC a une accuracy 9.95% supérieure.

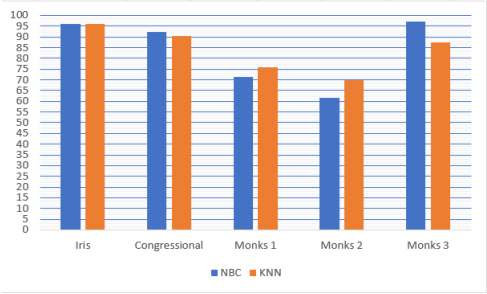

Figure 1: Résultat d'exactitudes pour les 2 algorithmes chacun à 50% de taux d'entraînement

De plus, le KNN a obtenu 7.65% plus d'accuracy sur le dataset congressional avec un taux d'entraînement de 10%. C'est à 50% de taux d'entraînement que le NBC est plus performante que le KNN.

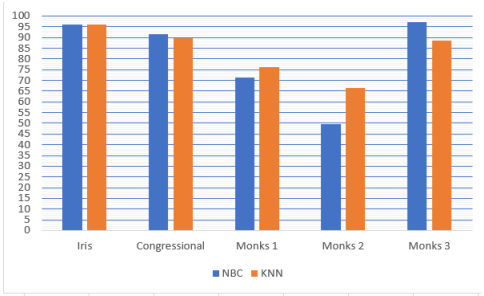

Figure 2: Résultat de précision pour les 2 algorithmes chacun à 50% de taux d'entraînement

Nous observons aussi que les courbes d'exactitude, précision et rappel ont toutes des allures similaires pour chacun des algorithmes.

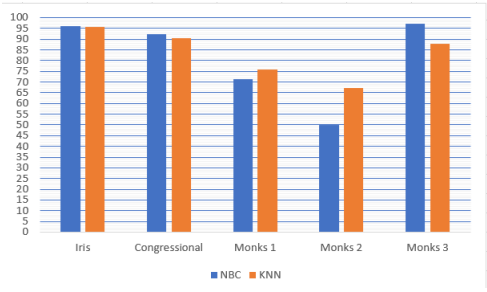

Figure 3: Résultat de rappel pour les 2 algorithmes chacun à 50% de taux d'entraînement

Afin de bien comparer les deux algorithmes, voici une courbe ROC des performances de ceux-ci avec leurs paramètres optimaux. Chacun des points de cette courbe correspond à une classe pour tous les ensembles de données.

À l'aide de la courbe ROC illustrée à la figure 4, on peut facilement voir que Knn et Bayes naïf performent de façon similaire. Toutefois, on peut remarquer que Bayes naïf classe deux classes de façon aléatoire, tandis que Knn

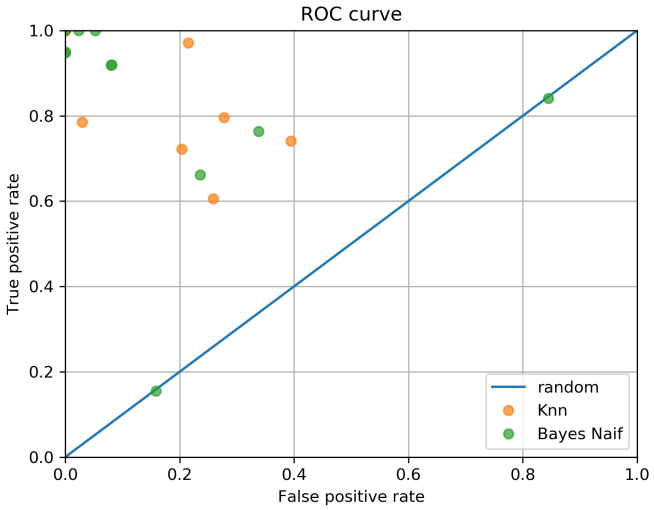

Figure 4: Courbe ROC comparant Knn et Bayes naïf

classe toutes ses classes avec un degré de précision tout de même haut à chaque fois. Il est donc particulièrement évident avec cette représentation que la classification Knn est plus robuste que la classification Bayes naïve.

7 Conclusion

Afin de conclure l'analyse et la comparaison, chaque méthode de classification comporte des avantages et des inconvénients. Dans notre étude, le KNN prenait plus temps à prédire, mais donnait généralement de meilleurs résultats à des taux d'entraînements plus faibles que la classification naïve bayésienne. Il serait intéressant d'étendre notre analyse à d'autres méthodes de classifications telles que les réseaux de neurones ou les machines à support de vecteurs.

L'implémentation de ces deux algorithmes n'a pas été de tout repos. En effet, quelques difficultés ont été rencontrées durant ce travail.

Pour le NBC, nous avons eu de la difficulté a établir les structures de données qui contiennent les résultats de l'entrainement. Nous avons opté pour des structures de bases en python (des listes et des dictionnaires). Il y a cependant place à l'amélioration. Celles-ci pourraient être cachées sous des classes qui exposeraient des fonctionnalités. Le code serait ainsi plus simple et la facilité de lecture du code serait améliorée.

Pour ce qui est du Knn, nous avons eu beaucoup de difficulté à trouver un moyen afin de préentraîner le modèle dans l'optique d'optimiser la prédiction du Knn. Malheureusement, après beaucoup de recherche sur internet, aucune méthode satisfaisant nos exigences n'a été trouvée dans un délai raisonnable. Une telle méthode nous aurait permis d'effectuer un préentraînement sur les données et ainsi accélérer la prédiction de notre Knn qui est plutôt lent à ce niveau.

La sortie du fichier *entrainer_et_tester.py* est disponible en annexe.

8 Annexe

Dataset	Train ratio (%)	Accuracy (%)		$\langle Prcision \rangle$ (%)		$\langle Rappel \rangle$ (%)		Temps d'exécution (ms)	
[redacted]		Méthode de classification							
		NBC	KNN	NBC	KNN	NBC	KNN	NBC	KNN
Iris	10	92.59	93.33	92.82	93.71	92.74	93.03	40	365.61
	50	96.00	96.00	96.00	96.03	96.30	95.74	20	2242.45
	90	100.00	100.00	100.00	100.00	100.00	100.00	10	5483.33
Congressional	10	80.36	88.01	80.25	87.33	81.64	88.63	110	2051.96
	50	92.20	90.37	91.55	89.61	92.34	90.62	70	16434.38
	90	95.45	100.00	95.37	100.00	95.37	100.00	20	46080.66
Monks 1	[redacted]	71.30	75.93	71.52	76.07	71.3	75.93	90	9539.15
Monks 2		61.57	69.68	49.70	66.38	49.82	67.35	120	12040.94
Monks 3		97.22	87.27	97.22	88.46	97.37	87.78	100	7987.68

Table 17: Résumé des résultats avec les tests de performance

File - entrainer_tester

```
C:\Users\ginco\.conda\envs\SN-Project-1.2-GPU\python.exe "C:/Users/ginco/Documents/GitHub/Ulaval_projects/IFT-4102_AI_avancee/IFT-4102-H20_Projet Partiel/Code/entrainer_tester.py"
Knn Train ratio: 0.9
findBestKWithCrossValidation: True


------------------------------------------------------------------------------------------------------------------------
Iris dataset classification:


 Train results:
Data set size: 135
Chosen K: 10

Confusion Matrix:
 [[45  0  0]
 [ 0 45  2]
 [ 0  1 42]]
Accuracy: 97.78 %
Precision [%]: [100.    95.74  97.67]
Mean Precision: 97.81 %
Recall [%]: [100.    97.83  95.45]
Mean Recall: 97.76 %

 Test results:
Data set size: 15

Confusion Matrix:
 [[5 0 0]
 [0 4 0]
 [0 0 6]]
Accuracy: 100.00 %
Precision [%]: [100. 100. 100.]
Mean Precision: 100.00 %
Recall [%]: [100. 100. 100.]
Mean Recall: 100.00 %

 --- Elapse time: 5693.82 ms ---

------------------------------------------------------------------------------------------------------------------------
Congressional dataset classification:


 Train results:
Data set size: 391
Chosen K: 7

Confusion Matrix:
 [[141  14]
 [  8 228]]
Accuracy: 94.37 %
Precision [%]: [90.97 96.61]
Mean Precision: 93.79 %
Recall [%]: [94.63 94.21]
Mean Recall: 94.42 %

 Test results:
Data set size: 44

Confusion Matrix:
 [[19  0]
 [ 0 25]]
Accuracy: 100.00 %
Precision [%]: [100. 100.]
Mean Precision: 100.00 %
Recall [%]: [100. 100.]
Mean Recall: 100.00 %

 --- Elapse time: 43071.73 ms ---

------------------------------------------------------------------------------------------------------------------------
Monks(1) dataset classification:


 Train results:
Data set size: 124
Chosen K: 14

Confusion Matrix:
 [[57 15]
 [ 5 47]]
Accuracy: 83.87 %
Precision [%]: [79.17 90.38]
Mean Precision: 84.78 %
Recall [%]: [91.94 75.81]
Mean Recall: 83.87 %

 Test results:
Data set size: 432

Confusion Matrix:
 [[172  60]
 [ 44 156]]
Accuracy: 75.93 %
Precision [%]: [74.14 78.  ]
```

```
Mean Precision: 76.07 %
Recall [%]: [79.63 72.22]
Mean Recall: 75.93 %

 --- Elapse time: 8261.07 ms ---

------------------------------------------------------------------------------------------------------------------------
Monks(2) dataset classification:


 Train results:
Data set size: 169
Chosen K: 3

Confusion Matrix:
 [[89 15]
 [16 49]]
Accuracy: 81.66 %
Precision [%]: [85.58 75.38]
Mean Precision: 80.48 %
Recall [%]: [84.76 76.56]
Mean Recall: 80.66 %

 Test results:
Data set size: 432

Confusion Matrix:
 [[215  56]
 [ 75  86]]
Accuracy: 69.68 %
Precision [%]: [79.34 53.42]
Mean Precision: 66.38 %
Recall [%]: [74.14 60.56]
Mean Recall: 67.35 %

 --- Elapse time: 12191.01 ms ---

------------------------------------------------------------------------------------------------------------------------
Monks(3) dataset classification:


 Train results:
Data set size: 122
Chosen K: 6

Confusion Matrix:
 [[60  7]
 [ 2 53]]
Accuracy: 92.62 %
Precision [%]: [89.55 96.36]
Mean Precision: 92.96 %
Recall [%]: [96.77 88.33]
Mean Recall: 92.55 %

 Test results:
Data set size: 432

Confusion Matrix:
 [[198  49]
 [  6 179]]
Accuracy: 87.27 %
Precision [%]: [80.16 96.76]
Mean Precision: 88.46 %
Recall [%]: [97.06 78.51]
Mean Recall: 87.78 %

 --- Elapse time: 7097.20 ms ---

------------------------------------------------------------------------------------------------------------------------
Bayes Naif Train ratio: 0.6


------------------------------------------------------------------------------------------------------------------------
Iris dataset classification:

Probability of each class
P(0) = 0.311
P(1) = 0.333
P(2) = 0.356

Probability of each feature
Probabilities knowing 0
P(0) = Gaussian distribution with average 5.025 and variance 0.145
P(1) = Gaussian distribution with average 3.425 and variance 0.162
P(2) = Gaussian distribution with average 1.421 and variance 0.027
P(3) = Gaussian distribution with average 0.239 and variance 0.010
Probabilities knowing 1
P(0) = Gaussian distribution with average 5.867 and variance 0.324
P(1) = Gaussian distribution with average 2.797 and variance 0.097
P(2) = Gaussian distribution with average 4.223 and variance 0.261
P(3) = Gaussian distribution with average 1.320 and variance 0.036
Probabilities knowing 2
P(0) = Gaussian distribution with average 6.616 and variance 0.461
P(1) = Gaussian distribution with average 2.934 and variance 0.097
```

```
189 P(2) = Gaussian distribution with average 5.606 and variance 0.327
190 P(3) = Gaussian distribution with average 1.978 and variance 0.072
191
192  Test results:
193 Data set size: 60
194
195 Confusion Matrix:
196  [[22  0  0]
197  [ 0 19  0]
198  [ 0  1 18]]
199 Accuracy: 98.33 %
200 Precision [%]: [100.   100.    94.74]
201 Mean Precision: 98.25 %
202 Recall [%]: [100.  95. 100.]
203 Mean Recall: 98.33 %
204
205  --- Elapse time: 10.11 ms ---
206
207 ------------------------------------------------------------------------------------------------------------------------
    ------------------------------------------------------
208 Congressional dataset classification:
209
210 Probability of each class
211 P(0) = 0.360
212 P(1) = 0.640
213
214 Probability of each feature
215 Probabilities knowing 0
216 Probabilities for feature number 0
217 P(0) = 0.755
218 P(1) = 0.245
219 Probabilities for feature number 1
220 P(0) = 0.404
221 P(1) = 0.457
222 P(2) = 0.138
223 Probabilities for feature number 2
224 P(0) = 0.819
225 P(1) = 0.170
226 P(2) = 0.011
227 Probabilities for feature number 3
228 P(0) = 0.021
229 P(1) = 0.979
230 Probabilities for feature number 4
231 P(0) = 0.043
232 P(1) = 0.947
233 P(2) = 0.011
234 Probabilities for feature number 5
235 P(0) = 0.106
236 P(1) = 0.894
237 Probabilities for feature number 6
238 P(0) = 0.723
239 P(1) = 0.255
240 P(2) = 0.021
241 Probabilities for feature number 7
242 P(0) = 0.809
243 P(1) = 0.160
244 P(2) = 0.032
245 Probabilities for feature number 8
246 P(0) = 0.894
247 P(1) = 0.106
248 Probabilities for feature number 9
249 P(0) = 0.372
250 P(1) = 0.628
251 Probabilities for feature number 10
252 P(0) = 0.872
253 P(1) = 0.064
254 P(2) = 0.064
255 Probabilities for feature number 11
256 P(0) = 0.149
257 P(1) = 0.755
258 P(2) = 0.096
259 Probabilities for feature number 12
260 P(0) = 0.117
261 P(1) = 0.840
262 P(2) = 0.043
263 Probabilities for feature number 13
264 P(0) = 0.011
265 P(1) = 0.968
266 P(2) = 0.021
267 Probabilities for feature number 14
268 P(0) = 0.862
269 P(1) = 0.074
270 P(2) = 0.064
271 Probabilities for feature number 15
272 P(0) = 0.319
273 P(1) = 0.553
274 P(2) = 0.128
275 Probabilities knowing 1
276 Probabilities for feature number 0
277 P(0) = 0.401
278 P(1) = 0.569
279 P(2) = 0.030
280 Probabilities for feature number 1
281 P(0) = 0.431
282 P(1) = 0.485
283 P(2) = 0.084
284 Probabilities for feature number 2
285 P(0) = 0.126
```

```
P(1) = 0.844
P(2) = 0.030
Probabilities for feature number 3
P(0) = 0.910
P(1) = 0.054
P(2) = 0.036
Probabilities for feature number 4
P(0) = 0.760
P(1) = 0.204
P(2) = 0.036
Probabilities for feature number 5
P(0) = 0.479
P(1) = 0.479
P(2) = 0.042
Probabilities for feature number 6
P(0) = 0.210
P(1) = 0.760
P(2) = 0.030
Probabilities for feature number 7
P(0) = 0.156
P(1) = 0.838
P(2) = 0.006
Probabilities for feature number 8
P(0) = 0.240
P(1) = 0.695
P(2) = 0.066
Probabilities for feature number 9
P(0) = 0.503
P(1) = 0.491
P(2) = 0.006
Probabilities for feature number 10
P(0) = 0.473
P(1) = 0.479
P(2) = 0.048
Probabilities for feature number 11
P(0) = 0.784
P(1) = 0.156
P(2) = 0.060
Probabilities for feature number 12
P(0) = 0.635
P(1) = 0.305
P(2) = 0.060
Probabilities for feature number 13
P(0) = 0.623
P(1) = 0.341
P(2) = 0.036
Probabilities for feature number 14
P(0) = 0.347
P(1) = 0.605
P(2) = 0.048
Probabilities for feature number 15
P(0) = 0.048
P(1) = 0.635
P(2) = 0.317

 Test results:
Data set size: 174

Confusion Matrix:
 [[68  8]
 [ 6 92]]
Accuracy: 91.95 %
Precision [%]: [89.47 93.88]
Mean Precision: 91.68 %
Recall [%]: [91.89 92.  ]
Mean Recall: 91.95 %

 --- Elapse time: 33.71 ms ---

------------------------------------------------------------------------------------------------------------------------
    ------------------------------------------------------
Monks(1) dataset classification:

Probability of each class
P(0) = 0.500
P(1) = 0.500

Probability of each feature
Probabilities knowing 0
Probabilities for feature number 0
P(1) = 0.500
P(2) = 0.323
P(3) = 0.177
Probabilities for feature number 1
P(1) = 0.323
P(2) = 0.323
P(3) = 0.355
Probabilities for feature number 2
P(1) = 0.484
P(2) = 0.516
Probabilities for feature number 3
P(1) = 0.258
P(2) = 0.387
P(3) = 0.355
Probabilities for feature number 4
P(2) = 0.323
P(3) = 0.306
P(4) = 0.371
```

```
383 Probabilities for feature number 5
384 P(1) = 0.435
385 P(2) = 0.565
386 Probabilities knowing 1
387 Probabilities for feature number 0
388 P(1) = 0.226
389 P(2) = 0.355
390 P(3) = 0.419
391 Probabilities for feature number 1
392 P(1) = 0.242
393 P(2) = 0.355
394 P(3) = 0.403
395 Probabilities for feature number 2
396 P(1) = 0.565
397 P(2) = 0.435
398 Probabilities for feature number 3
399 P(1) = 0.419
400 P(2) = 0.242
401 P(3) = 0.339
402 Probabilities for feature number 4
403 P(1) = 0.468
404 P(2) = 0.177
405 P(3) = 0.177
406 P(4) = 0.177
407 Probabilities for feature number 5
408 P(1) = 0.468
409 P(2) = 0.532
410
411  Test results:
412 Data set size: 432
413
414 Confusion Matrix:
415  [[165  73]
416  [ 51 143]]
417 Accuracy: 71.30 %
418 Precision [%]: [69.33 73.71]
419 Mean Precision: 71.52 %
420 Recall [%]: [76.39 66.2 ]
421 Mean Recall: 71.30 %
422
423  --- Elapse time: 46.88 ms ---
424
425 ------------------------------------------------------------------------------------------------------------------------------------------------------
    ------------------------------------------------------
426 Monks(2) dataset classification:
427
428 Probability of each class
429 P(0) = 0.621
430 P(1) = 0.379
431
432 Probability of each feature
433 Probabilities knowing 0
434 Probabilities for feature number 0
435 P(1) = 0.362
436 P(2) = 0.333
437 P(3) = 0.305
438 Probabilities for feature number 1
439 P(1) = 0.343
440 P(2) = 0.352
441 P(3) = 0.305
442 Probabilities for feature number 2
443 P(1) = 0.476
444 P(2) = 0.524
445 Probabilities for feature number 3
446 P(1) = 0.371
447 P(2) = 0.305
448 P(3) = 0.324
449 Probabilities for feature number 4
450 P(1) = 0.276
451 P(2) = 0.190
452 P(3) = 0.286
453 P(4) = 0.248
454 Probabilities for feature number 5
455 P(1) = 0.533
456 P(2) = 0.467
457 Probabilities knowing 1
458 Probabilities for feature number 0
459 P(1) = 0.297
460 P(2) = 0.344
461 P(3) = 0.359
462 Probabilities for feature number 1
463 P(1) = 0.297
464 P(2) = 0.406
465 P(3) = 0.297
466 Probabilities for feature number 2
467 P(1) = 0.516
468 P(2) = 0.484
469 Probabilities for feature number 3
470 P(1) = 0.234
471 P(2) = 0.344
472 P(3) = 0.422
473 Probabilities for feature number 4
474 P(1) = 0.219
475 P(2) = 0.312
476 P(3) = 0.297
477 P(4) = 0.172
478 Probabilities for feature number 5
479 P(1) = 0.438
```

File - entrainer_tester

```
P(2) = 0.562

 Test results:
Data set size: 432

Confusion Matrix:
 [[244 120]
 [ 46  22]]
Accuracy: 61.57 %
Precision [%]: [67.03 32.35]
Mean Precision: 49.69 %
Recall [%]: [84.14 15.49]
Mean Recall: 49.82 %

 --- Elapse time: 39.85 ms ---

------------------------------------------------------------------------------------------------------------------------
Monks(3) dataset classification:

Probability of each class
P(0) = 0.508
P(1) = 0.492

Probability of each feature
Probabilities knowing 0
Probabilities for feature number 0
P(1) = 0.355
P(2) = 0.371
P(3) = 0.274
Probabilities for feature number 1
P(1) = 0.210
P(2) = 0.177
P(3) = 0.613
Probabilities for feature number 2
P(1) = 0.516
P(2) = 0.484
Probabilities for feature number 3
P(1) = 0.306
P(2) = 0.339
P(3) = 0.355
Probabilities for feature number 4
P(1) = 0.161
P(2) = 0.161
P(3) = 0.194
P(4) = 0.484
Probabilities for feature number 5
P(1) = 0.532
P(2) = 0.468
Probabilities knowing 1
Probabilities for feature number 0
P(1) = 0.433
P(2) = 0.283
P(3) = 0.283
Probabilities for feature number 1
P(1) = 0.433
P(2) = 0.517
P(3) = 0.050
Probabilities for feature number 2
P(1) = 0.550
P(2) = 0.450
Probabilities for feature number 3
P(1) = 0.350
P(2) = 0.283
P(3) = 0.367
Probabilities for feature number 4
P(1) = 0.367
P(2) = 0.350
P(3) = 0.267
P(4) = 0.017
Probabilities for feature number 5
P(1) = 0.433
P(2) = 0.567

 Test results:
Data set size: 432

Confusion Matrix:
 [[204  12]
 [  0 216]]
Accuracy: 97.22 %
Precision [%]: [ 94.44 100.  ]
Mean Precision: 97.22 %
Recall [%]: [100.    94.74]
Mean Recall: 97.37 %

 --- Elapse time: 50.64 ms ---

------------------------------------------------------------------------------------------------------------------------

Process finished with exit code 0
```