Université de Mons

Datawarehousing and datamining

Travaux pratiques avec Weka

Auteur :
Maxime De Wolf

6 mai 2018

Table des matières

1 Weka : Tutoriel

1.1 Questions 17.1.9 et 17.1.10

Ces questions portent sur l'arbre de décision crée à partir du fichier *iris.arff*. Voici donc l'arbre de décision obtenu :

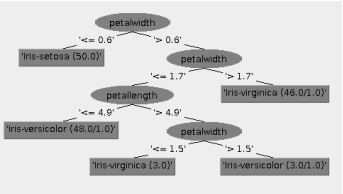


FIGURE 1 – Arbre de décision du *dataset iris.arff*

Question 17.1.9

Cette question consiste à évaluer la qualité de cet arbre (Figure 1) grâce à différentes options de tests. Ici, on effectuera ces tests une première fois avec le *dataset* complet et la 2^e fois avec la technique *10-fold cross-validation*. Nous comparons ensuite les résultats obtenus sur base des 2 *confusion matrix* :

TABLE 1 – *Confusion matrix* obtenues grâce à deux méthodes de test différentes

a	b	c	
50	0	0	a = Iris-setosa
0	49	1	b = Iris-versicolor
0	2	48	c = Iris-virginica

(a) *Dataset* complet

a	b	c	
49	1	0	a = Iris-setosa
0	47	3	b = Iris-versicolor
0	2	48	c = Iris-virginica

(b) *10-fold cross-validation*

Nous remarquons que le test sur le *dataset* complet classifie correctement 98% des instances tandis que ce chiffre descend à 96% avec le test *10-fold cross-validation*. Tester le modèle avec le *dataset* complet est une mauvaise idée car il donne une estimation optimiste de la qualité du modèle. En revanche, *10-fold cross-validation* permet de se faire une bonne idée de la généralisation du modèle et offre donc une meilleure mesure de qualité.

Question 17.1.10

En observant la localisation de ces erreurs, nous remarquons que certaines instances de classe *Iris-Verginica* ont des valeurs d'attributs équivalentes à celles d'instance de classe *Iris-Versicolor*. Le modèle n'a donc aucune chance de les différencier si nous voulons éviter l'*overfitting*. D'autre part, nous remarquons que l'instance de classe *Iris-Setosa* qui a été mal identifier aurait dû être correctement classé selon l'arbre de décision final obtenu.

1.2 Questions 17.2.4 à 17.2.11

Question 17.2.4

Le but de cette question est d'étudier la précision du classificateur *5-nearest neighbor* en fonction des attributs utilisés lors de cette classification. Ici, nous exécutons cette algorithme sur le *dataset glass.arff* et nous le test grâce à la technique *10-fold cross-validation*. Les résultats ainsi obtenus sont résumés dans la table suivante :

TABLE 2 – Précision obtenue en utilisant *IBk* pour différents sous-ensemble d'attributs

Nombre d'attributs	Attribut retiré	Précision de la classification
9	$\emptyset$	67.757
8	Si	71.4953
7	Fe	73.3645
6	Al	73.3645
5	Na	74.2991
4	Ba	74.7664
3	K	72.4299
2	Ca	71.9626
1	Mg	52.8037
0	RI	35.514

Grâce à ce tableau, nous remarquons donc que la précision de *IBk* sur le *dataset* complet est de 67.757% alors que nous obtenons une précision de 74.7664% une fois que nous retirons les attributs *Si, Fe, Al, Na, Ba* du *dataset*. Ce qui nous donne un gain d'environ 7% de précision.

Question 17.2.5

Cette question demande de critiquer la pertinence de la précision maximum obtenue dans la question précédente. Ou, en d'autres mots, cette estimation est-elle biaisée ou non ? Etant donné que le test du modèle est effectué sur le *dataset* d'entraînement, cette estimation est effectivement biaisée.

Question 17.2.6

Cette question nous demande de constater l'effet du bruit sur un modèle construit grâce à *IBk*. Pour ce faire, nous allons faire varier le pourcentage de bruits ainsi que la taille du voisinage dans les paramètres d'*IBk*. Ainsi, une estimation de la précision sera calculé avec *10-fold cross-validation*. Il est important de noter que ce test se fera sans les ajouts de bruits. Il n'y a donc présence de bruit que lors de la phase d'entraînement mais pas lors de la phase de test. La table suivante résume les résultat obtenus :

TABLE 3 – Effet du bruit sur la précision d'*IBk*, en fonction de différentes tailles de voisinage

Pourcentage de bruit	k = 1	k = 3	k = 5
0%	70.6%	72.0%	67.8%
10%	62.6%	69.6%	64.5%
20%	50.5%	63.1%	61.7%
30%	47.2%	58.4%	59.8%
40%	41.1%	54.7%	55.1%
50%	33.2%	44.4%	45.3%
60%	27.1%	35.5%	35.5%
70%	20.1%	28.5%	29.0%
80%	14.0%	21.0%	21.0%
90%	7.9%	13.6%	9.3%
100%	4.7%	7.9%	7.5%

Question 17.2.7

Cette question nous demande de critiquer les résultats obtenues lors de la question précédente. Plus particulièrement, on nous demande l'effet qu'a une augmentation du bruit au niveau de la classe. La Table 3 nous permet de constater que cette augmentation réduit la précision du classificateur *k-nearest neighbor* et ce peu importe la valeur du k.

Question 17.2.8

Cette question s'intéresse plutôt aux effets qu'a la modification de la valeur de k pour le classificateur *k-nearest neighbor*. Dans la Table 3 nous remarquons qu'augmenter la valeur de k rend le modèle obtenu plus robuste au bruit.

En effet, nous remarquons que bien qu'il n'y ait pas de différence significative entre les modèles obtenus pour k = 3 et k = 5, ils sont plus résistants au bruit que ceux obtenus pour k = 1.

Question 17.2.9

Pour cette question, nous devons comparer les classificateurs *IBk* et *J48* en fonction du pourcentage de l'ensemble d'apprentissage utilisé. Ces résultats sont encodés dans la table suivante :

TABLE 4 – Variation de la précision du modèle en fonction de la taille de l'ensemble d'apprentissage pour *IBk* et *J48*

Pourcentage de l'ensemble d'apprentissage	*IBk*	*J48*
10%	52.8%	45.3%
20%	63.6%	53.3%
30%	60.3%	59.3%
40%	63.6%	65.0%
50%	62.6%	63.1%
60%	64.5%	69.2%
70%	65.9%	67.8%
80%	67.8%	70.1%
90%	67.3%	69.6%
100%	66.8%	68.2%

Question 17.2.10

Cette question s'intéresse à l'effet de l'augmentation de nombre d'éléments de l'ensemble d'entraînement. Nous pouvons répondre à cette question grâce à la Table 4. Nous observons ainsi que plus l'ensemble d'entraînement est grand, plus la précision du modèle augmente jusqu'à un certain seuil où elle devient constante. Nous constatons que ce seuil se situe aux environs de 67% pour *IBk* et 69% pour *J48*.

Question 17.2.11

On nous demande ici d'identifier laquelle des deux techniques -*IBk* et *J48*- est la plus affectée par l'augmentation de la taille de l'ensemble d'entraînement. En nous référant à la Table 4, nous observons que *IBk* a une précision initiale de 52.8% pour arriver finalement à une précision de 66.8%. La précision du modèle *IBk* subit donc une variation de 14%. Pour *J48*, la précision du modèle varie de 45.3% à 68.2% ce qui donne une variation de 22.9%. Comme le modèle *J48* à une plus grande variation que *IBk*, nous concluons que *J48* est la technique la plus sensible à la taille de l'ensemble d'apprentissage.

1.3 Questions 17.3.1 à 17.3.11

Question 17.3.1

Pour résoudre cette question, nous utilisons le *boundary visualizer* de *Weka*. Cet outil sert à visualiser graphiquement les prédictions d'un modèle, dans notre cas, ce modèle est produit par le classificateur *1R*. La Figure 2 montre le résultat obtenu. Nous y voyons que la prédiction du modèle ne se fait uniquement qu'en fonction de la largeur des pétales (axe Y).

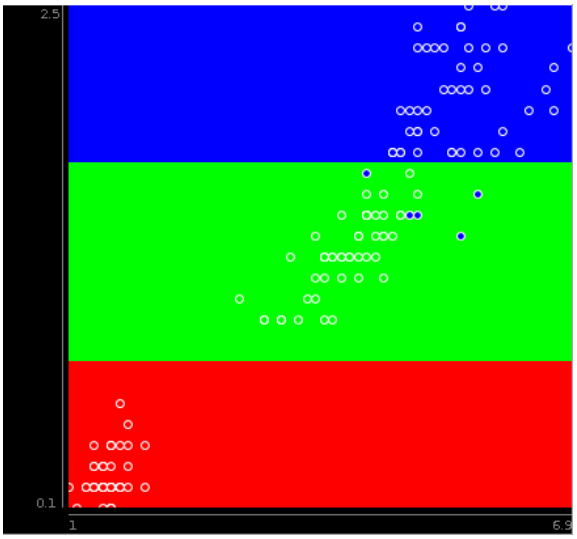


FIGURE 2 – Prédiction du type d'iris en fonction de la longueur des pétales (axe X) et de la largeur des pétales (axe Y) grâce au modèle 1R

Question 17.3.2

Pour résoudre cette question, nous avons du jouer avec le paramètre *minBucketSize* du classificateur *1R*. Après avoir testé plusieurs valeurs, nous pouvons conclure que ce paramètre ne change pas les résultats obtenus lors de la question précédente (voir Figure 2).

Nous pensons que c'est parce les valeurs que peuvent prendre la largeur des pétales (axe Y) varient entre 0.1 et 2.5. Il ne faut donc que 3 *buckets* pour discrétiser ces valeurs. Etant donné le nombre d'instances de ce jeu de données, la valeur de *minBucketSize* n'est utilisée que si elle n'est supérieure ou égale à 50. 50 est la valeur frontière car il y a 50 instances associés à chaque label.

Question 17.3.3

On doit répondre à la question "Pourquoi n'y a-t-il pas plus de régions lorsque le *bucket* a une petite valeur minimale ?". Cela s'explique car il s'agit d'une valeur minimale et donc, comme sur ce jeu de données on a relativement beaucoup d'instances par *bucket*, la valeur de ce paramètre n'est pas prise en compte.

Question 17.3.4

Le nombre minimum de région est 1 et la valeur minimum du paramètre *minBucketSize* pour l'obtenir est 51. Cela peut s'expliquer simplement par le fait que chaque classe ne définit chacune que 50 instances.

Question 17.3.5

En utilisant le classificateur *IBK* avec k=1, nous obtenons la prédiction visible à la Figure 3. Comme expliqué dans l'énoncé, chaque point ne possède qu'une des trois couleurs (rouge, vert ou bleu) car k vaut 1 et donc, le modèle ne prédit pas un point pouvant appartenir à plusieurs classe.

Nous remarquons néanmoins une zone du graphique qui fait exception à cela car sa couleur est une nuance de bleu (et pas totalement bleu). Nous pouvons justifier cela car il existe plusieurs instances qui ont les même longueur et largeur de pétale mais qui sont associés à des classes différentes. La zone possédant une couleur nuancée est en fait la zone qui contient ces instances.

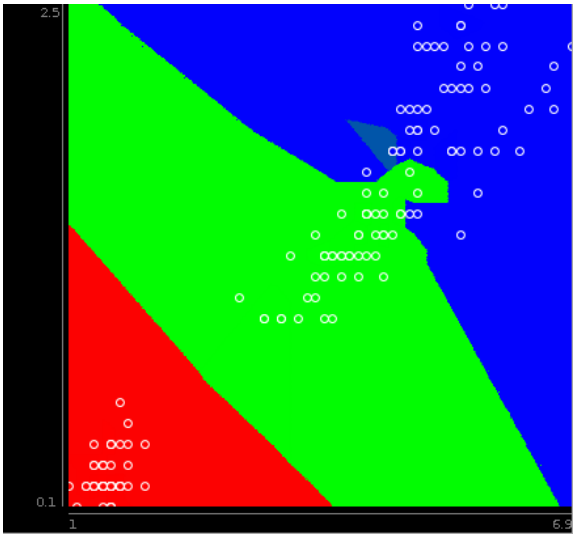


FIGURE 3 – Prédiction du type d'iris en fonction de la longueur des pétales (axe X) et de la largeur des pétales (axe Y) grâce au modèle IBK (avec k=1)

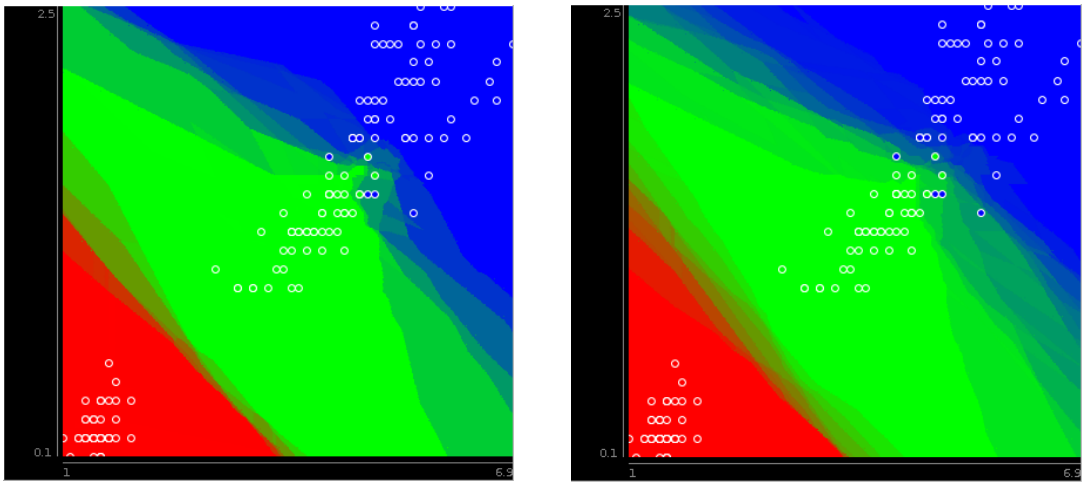
Question 17.3.6

En augmentant les valeurs du paramètre k, nous remarquons que les zones de la prédiction sont de plus en plus nuancées comme nous le montre la Figure 4.

Question 17.3.7

Effectivement, comme le montre la Figure 5, la prédiction donnée par le modèle *NaïveBayes* a un *pattern* très différent des prédictions précédemment obtenues.

Nous pouvons expliquer ce *pattern* par le fait que *NaïveBayes* base sa prédiction par rapport à la probabilité d'appartenir à une classe selon les valeurs X et Y d'une instance. Cela donne donc ce motif quadrillé.



(a) k=5 (b) k=10

FIGURE 4 – Prédiction du type d'iris en fonction de la longueur des pétales (axe X) et de la largeur des pétales (axe Y) grâce au modèle IBK

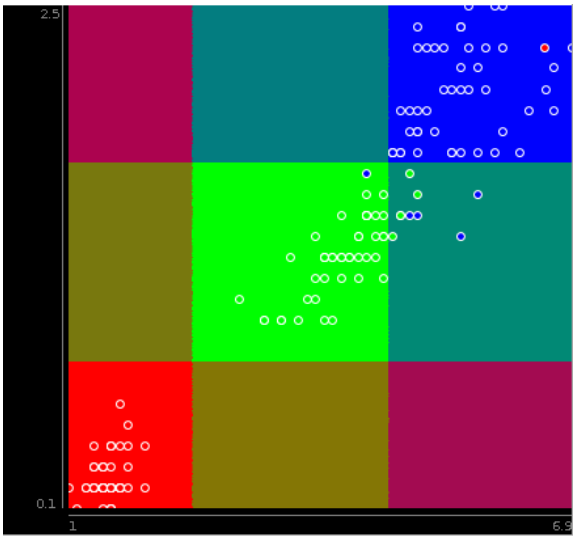


FIGURE 5 – Prédiction du type d'iris en fonction de la longueur des pétales (axe X) et de la largeur des pétales (axe Y) grâce au modèle NaïveBayes

La couleur de chaque zone est ensuite calculée en fonction de la "couleur" des instances qu'elle contient ainsi qu'aux autres zones "proche". Comme les instances de notre jeu de données se situent surtout sur la diagonale montante, nous obtenons un motif symétrique.

Question 17.3.8

Le modèle *JRip* nous donne la prédiction visible à la Figure 6. Nous pouvons y voir que le modèle fait ses prédictions en fonctions de la longueur des pétales (axe X) sauf pour les classes *iris-versicolor* où la largeur des pétales intervient également.

Un petit coup d'œil dans l'*explorer* de *Weka* nous permet de savoir quelles sont les règles de prédiction utilisées par *JRip*. Ces règles sont les suivantes :

— (petallength ≤ 1.9) $\implies$ class=Iris-setosa
— (petalwidth ≤ 1.6) and (petallength ≤ 4.9) $\implies$ class=Iris-versicolor
— otherwise class=Iris-virginica

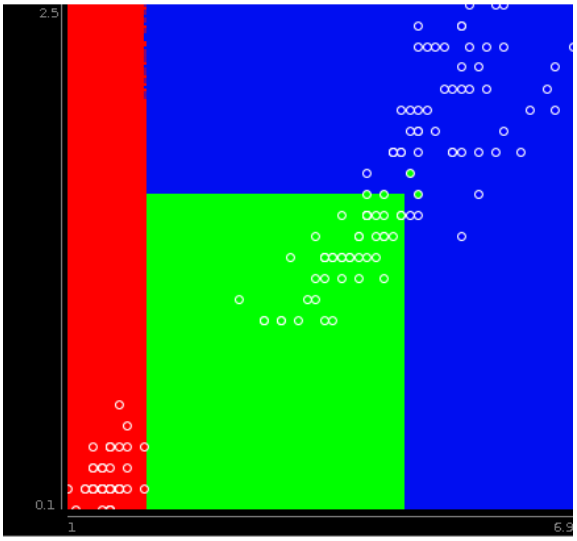


FIGURE 6 – Prédiction du type d'iris en fonction de la longueur des pétales (axe X) et de la largeur des pétales (axe Y) grâce au modèle JRip

Question 17.3.9

Les règles utilisées par *JRip* lors de la question précédente donnent de bonnes prédictions uniquement si elles sont exécutées dans le bon ordre. Voici un ensemble de règles équivalentes dont le résultat ne dépend pas de l'ordre dans lesquelles on les exécutent :

— (petallength ≤ 1.9) $\implies$ class=Iris-setosa
— (petalwidth ≤ 1.6) and ($1.9 <$ petallength ≤ 4.9) $\implies$ class=Iris-versicolor
— ($1.9 <$ petallength) and ($1.6 <$ petalwidth) or ($4.9 <$ petallength) $\implies$ Iris-virginica

Question 17.3.10

La Figure 7 montre la prédiction suivant le modèle *J48*. Sans surprise, nous pouvons voir que ce modèle découpe des zones soit selon l'axe X, soit selon l'axe Y, c'est pour cela qu'aucune zone n'est

délimitée par un segment diagonal.

Nous pouvons y voir une zone dont la couleur est une nuance de bleu et de vert. Cela vient du fait que l'arbre du modèle *J48* a subi un élagage afin d'éviter un *overfit* des données. Cette zone contient donc des instances appartenant à des classes différentes.

La Figure 8 montre l'arbre généré par J48 qui est utilisé pour faire les prédictions.

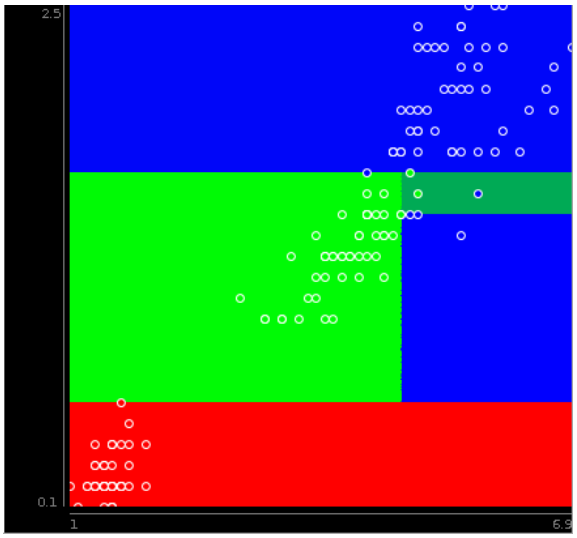


FIGURE 7 – Prédiction du type d'iris en fonction de la longueur des pétales (axe X) et de la largeur des pétales (axe Y) grâce au modèle J48

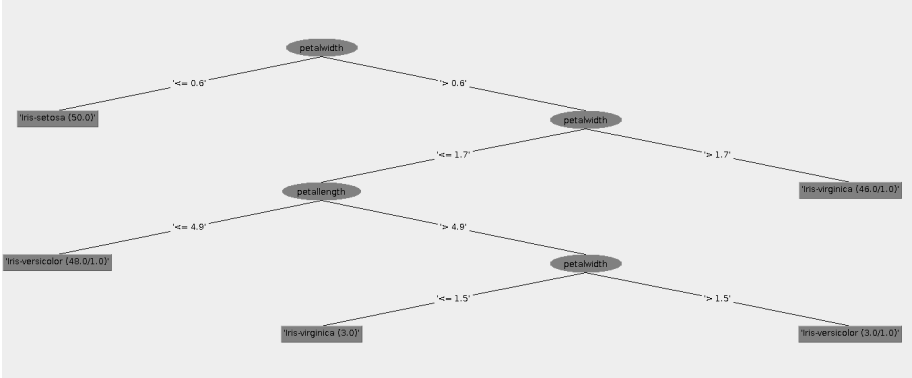
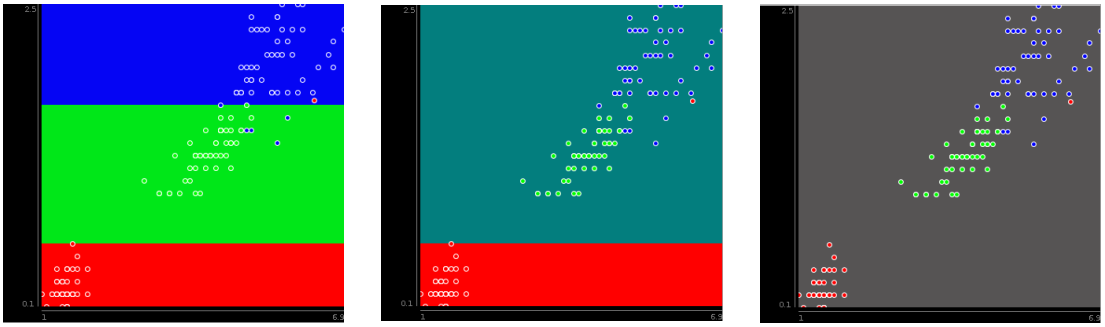


FIGURE 8 – Arbre généré par J48 afin de prédire le type d'iris

Question 17.3.11

Pour résoudre cette question, nous devons jouer avec les arguments du modèle *J48* afin d'obtenir 3, 2, puis une zone dans les prédictions. La Figure 9 résume les résultats obtenus.



(a) 7 à 49 instances par feuilles (b) 50 à 75 instances par feuilles (c) 76 à 150 instances par feuilles

FIGURE 9 – Variation du nombre de zones des prédictions du jeu de données *Iris.arff* en fonction du paramètre *minNumObj* de J48

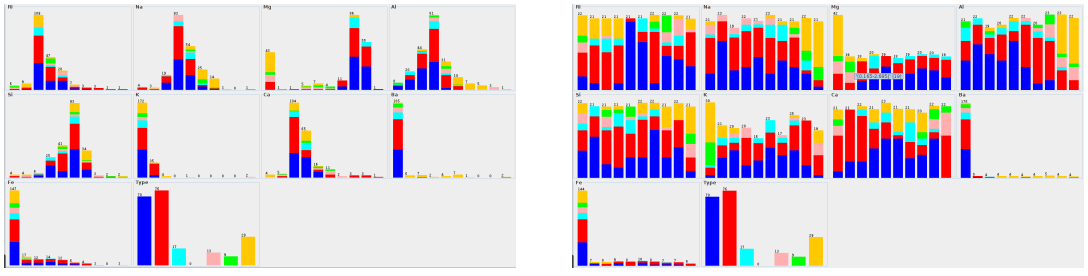
1.4 Questions 17.4.1 à 17.4.4 + question 17.4.8

Question 17.4.1

Cette question porte sur la discrétisation non-supervisée des données. Il nous est donc demandé de discrétiser les données du fichier *glass.arff* selon deux méthodes différentes : *equal-width* et *equal-frequency*. La Figure 10 montre les résultats ainsi obtenus.

La méthode *equal-width* discrétise les données en créant des intervalles de mêmes longueurs. Il est donc normal que nous voyons apparaître de gros écart de "population" entre les différents intervalles.

La méthode *equal-frequency* discrétise les données en créant des intervalles peuplés plus ou moins de la même façon. Il est donc normal que les écarts de "population" entre intervalles sont très petits. Nous voyons tout de même quelques intervalles ayant de gros écarts avec les autres. Cela s'explique car ces pics sont causés par un grand nombre d'instances ayant la même valeur d'attribut. Il est donc normal qu'il soit impossible de placer ces instances dans des intervalles différents.



(a) méthode *equal-width* (b) méthode *equal-frequency*

FIGURE 10 – Discrétisation du jeu de données *glass.arff* suivant 2 méthodes non-supervisées différentes

Question 17.4.2

La figure 11 montre le résultat de la discrétisation supervisée du fichier *iris.arff*. Cette méthode permet de mettre en évidence les attributs les plus appropriés afin d'effectuer une classification. Ici, nous remarquons que l'attribut *petalwidth* est un bon candidat pour tenter une classification. Cela conforte les résultats obtenus pour répondre aux questions du chapitre 17.3.

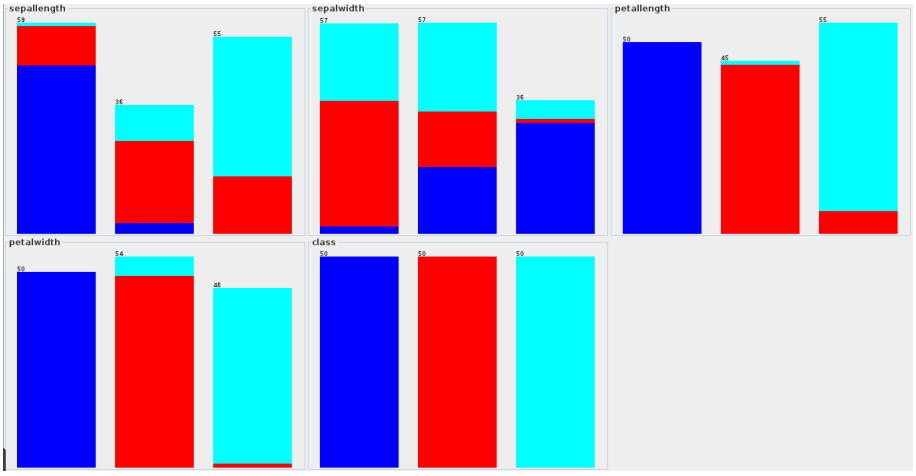


FIGURE 11 – Discrétisation supervisée du fichier *iris.arff*

Question 17.4.3

Cette question demande la même chose que la précédente sauf qu'on utilise cette fois le fichier *glass.arff*. Nous remarquons que certains attributs ne comptent qu'un seul intervalle. Ces attributs (*Fe* et *Si*) sont en fait de très mauvais candidats pour effectuer une classification comme nous l'avons déjà remarqué pendant la question 17.2.4.

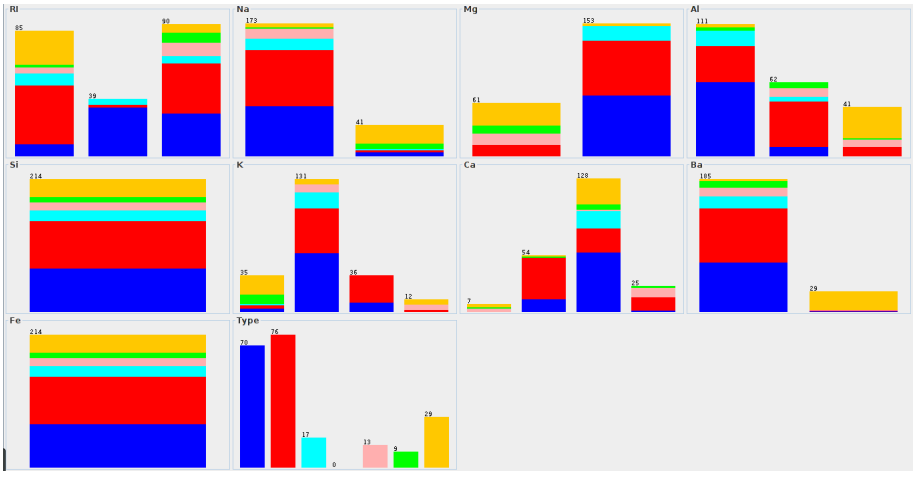


FIGURE 12 – Discrétisation supervisée du fichier *glass.arff*

Question 17.4.4

Pour cette question, nous effectuons la même manipulation qu'à la question précédente sauf que nous mettons le paramètre *makeBinary* du filtre à vrai. Nous remarquons alors que ce paramètre sert à ne créer que 2 intervalles maximum par attribut quitte à créer de nouveaux attributs pour stocker les intervalles excédants. Nous pouvons avoir un aperçu de cette manipulation grâce à la Figure 13.

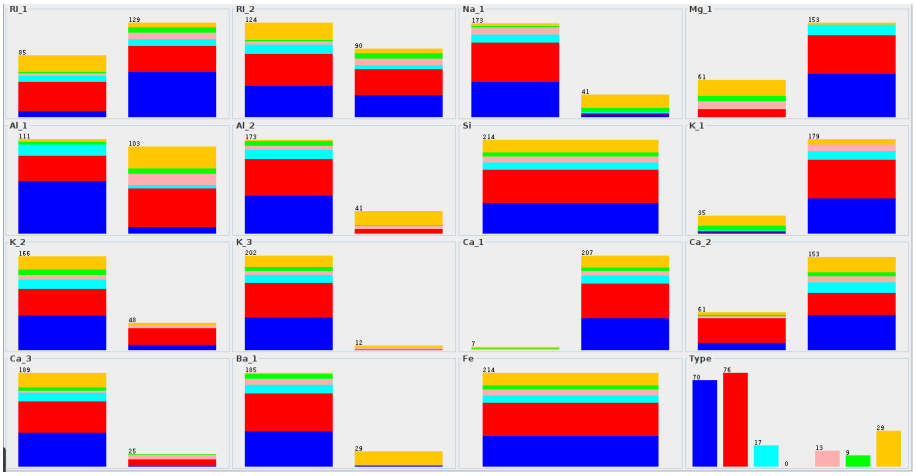


FIGURE 13 – Discrétisation supervisée du fichier *glass.arff* avec le paramètre *makeBinary* à vrai

Question 17.4.8

Pour cette question, nous devions trouver les 4 attributs les plus pertinents dans le but d'une classification. Nos résultats sont les suivants par ordre décroissant de pertinence :

1. *wage-increase-first-year*
2. *wage-increase-second-year*
3. *statutory-holidays*
4. *contribution-to-dental-plan*

1.5 Questions 17.5.1 à 17.5.4 + question 17.5.6

Question 17.5.1

Pour résoudre cette question, nous commençons par créer un document texte sur lequel nous appliquerons le filtre *StringToWordVector*. Le document texte que nous avons créer contient les mêmes données que la Table 5.

Ce filtre créer un jeu de données à partir de notre document texte. Il créer un attribut par mot, soit un total de 34 pour notre exemple. Ensuite, nous mettons le paramètre *minTermFreq* de ce

filtre à 2. Cela implique qu'il ne crée un nouvel attribut que pour un mot seulement si il apparaît au moins 2 fois dans le texte.

TABLE 5 – Document d'apprentissage

Document texte	**Classification**
The price of crude oil has increased significantly	yes
Demand for crude oil outstrips supply	yes
Some people do not like the flavor of olive oil	no
The food was very oily	no
Crude oil is in short supply	yes
Use a bit of cooking oil in the frying pan	no

Question 17.5.2

Nous utilisons ces données fraîchement récoltées afin de créer un arbre de décision grâce à *J48*. La Figure 14 nous montre cet arbre.

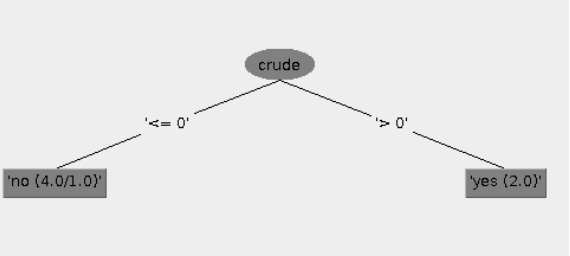


FIGURE 14 – Arbre de classification produit par *J48*

Question 17.5.3

Nous testons ensuite cet arbre sur un jeu de données extrait d'un autre fichier texte. Ce fichier contient les mêmes données que la Table 6. Ce modèle nous donne les prédictions suivantes par ordre d'apparition dans la Table 6 :

1. *yes*
2. *no*
3. *no*
4. *no*

Nous remarquons que ce modèle est cohérent même si sa troisième prédiction aurait du être *yes*. Cela est sûrement dû au fait que l'ensemble d'entraînement est très petit.

TABLE 6 – Document test

Document texte	**Classification**
Oil platforms extract crude oil	?
Canola oil is supposed to be healthy	?
Iraq has significant oil reserves	?
There are different types of cooking oil	?

Question 17.5.4

Pour cette question, nous devons construire deux modèles en utilisant *J48* et *NaiveBayesMultinomial* pour deux jeux de données textuelles. Le premier de ces deux jeu de données a pour but d'identifier les textes qui parlent de maïs (*ReutersCorn-train.arff*) tandis que le but du second est d'identifier les textes qui parlent de graines (*ReutersGrain-train.arff*).

La Table 7 résume les résultats obtenus pour les deux classificateurs sur le premier jeu de données. Nous voyons que *J48* est meilleur que *NaiveBayesMultinomial* sur tous les plans. Il est donc évident que pour ce jeu de données, *J48* est le classificateur que nous choisirions.

TABLE 7 – Résultats des deux modèles entraînés sur *ReutersCorn-train.arff* et testés sur *ReutersCorn-test.arff*

	Instances bien classifiées	Précision	*Recall*
J48	97.4%	97.3%	97.4%
NaiveBayesMultinomial	93.7%	96.4%	93.7%

La Table 8 montre les résultats des deux classificateurs sur le second jeu de données. Ici encore nous voyons que *J48* est le meilleur des deux classificateurs. Toutefois, nous remarquons que *NaiveBayesMultinomial* a une meilleure précision en moyenne. Cela veut dire que le modèle fait peu d'erreur lorsqu'il doit prédire la classe d'une instance. Cependant, il s'agit là d'une moyenne. En effet, en regardant les résultats de plus près, nous remarquons que ce modèle à en fait une très mauvaise précision (21.4%) pour ce qui est de repérer les textes qui parlent de graines. Cela conforte donc notre choix de *J48*.

TABLE 8 – Résultats des deux modèles entraînés sur *ReutersGrain-train.arff* et testés sur *ReutersGrain-test.arff*

	Instances bien classifiées	Précision	*Recall*
J48	94.5%	96.2%	94.5%
NaiveBayesMultinomial	86.3%	96.5%	86.3%

Question 17.5.6

La Table 9 montre les valeurs de *ROC* des deux classificateurs sur les deux jeux de données. Contrairement aux résultats précédemment obtenus, *NaiveBayesMultinomial* apparaît comme étant le meilleur classificateur pour ce problème.

TABLE 9 – Récapitulatif des valeurs *ROC* pour les 2 classificateurs et les 2 jeux de données

	Maïs	Graines
J48	69.4%	90.3%
NaiveBayesMultinomial	95.2%	96.5%

2 CoIL Challenge 2000

2.1 Description du problème

Le problème consiste à concevoir un modèle capable de détecter les personnes potentiellement intéressées par la souscription d'une assurance caravane. Le but de cette tâche est d'envoyer une publicité ciblée sur ces personnes afin de réduire les coûts de celle-ci. Nous nous limiterons donc à l'envoie de 800 de ces publicités.

2.2 Description des données

Pour concevoir ce modèle, nous possédons un fichier d'entraînement contenant les données "historiques" de 5822 personnes dont 348 (±6%) possèdent une assurance caravane.

Nous possédons également un fichier contenant les données de 4000 personnes parmi lesquelles nous aimerions envoyer les 800 courriers publicitaires. Nous nous attendons à ce que la même proportion de ces personnes soient intéressée par une assurance caravane que dans notre fichier "historique". Nous estimons donc que 6% de ces 4000 personnes (±240) soient intéressées.

2.3 Mesures importantes

Afin de repérer au mieux les personnes intéressées par l'assurance caravane, notre modèle doit maximiser la mesure *recall* afin de trouver un maximum de ces personnes. Nous devons également veiller à ce que la mesure *precision* de notre modèle ne soit pas trop mauvaise afin de ne pas dépasser les 800 courriers publicitaires à envoyer.

2.4 Pré-traitement des données

Extraction des données

Dans un premier temps, nous devons convertir les fichiers de données en format compatible pour le logiciel *Weka*. Pour ce faire, nous utilisions un petit programme *Python* écrit par un autre élève : Nico Salamone.

Sélection d'attributs

Afin de faire une sélection correcte des attributs du jeu de données, nous essayons plusieurs évaluateurs d'attributs afin d'être sûr de choisir la meilleure sélection d'attributs. Dans notre cas nous choisissons le modèle *NaiveBayes* pour évaluer les sélections. Nous évaluerons l'efficacité d'une sélection selon les valeurs *precision* et *recall* de la classification pour la classe qui nous intéresse. La Table 10 résume les résultats obtenus pour les différentes sélections effectuées.

Vous trouverez ci-dessus un détail sur les différents évaluateurs d'attributs testés.

Dans un premier temps nous effectuons la classification sur la totalité des attributs afin d'avoir un point de comparaison. Nous lançons donc le test sur les 85 attributs.

Ensuite, nous testons l'évaluateur *CfsSubsetEval* qui est l'évaluateur par défaut. Il sélectionne les attributs suivants : **25,42,43,44,47,59,61,64,68,82**. Comme le montre la Table 10, les résultats obtenus sont très médiocre.

L'évaluateur *ConsistencySubsetEval* est légèrement meilleur mais n'est toujours pas satisfaisant. Il sélectionne les attributs suivant : **1,3,4,6,16,21,23,24,29,30,32,37,40,44,47,54,59,61,64,68**.

Nous avons effectué plusieurs tests avec l'évaluateur *GainRatioAttributeEval*. D'abord, nous avons sélectionné les attributs qui un ratio de gain supérieur à 0.01. Cela nous a donné la sélection suivante : **82,61,47,85,64,68,86**. Malheureusement, une fois encore les résultats ne sont pas satisfaisant.
Nous avons ensuite sélectionné les attributs dont le ratio de gain est supérieur à 0. Cela nous a donc donné la sélection suivante : **82,61 ,47 ,85 ,64 ,68 ,25 ,19 ,59 ,42 ,37 ,65 ,43 ,44 ,30 ,31 ,18 ,54 ,75 ,10 ,34 ,39 ,12 ,80 ,5 ,16 ,32 ,1 ,29 ,28 ,21 ,40 ,22 ,24 ,23 ,35 ,36 ,13 ,2 ,6 ,3 ,4 ,86**. Cette sélection donne de bons résultats mais sont toujours inférieur aux résultats obtenus en sélectionnant la totalité des attributs.

Nous avons également testé d'autres évaluateurs mais nous ne les avons pas mentionné car ils sélectionnaient soit aucun soit tout les attributs.

TABLE 10 – Résultats des différentes sélections effectuées

	Precision	*Recall*
Pas de sélection	13.6%	48.6%
CfsSubsetEval	23.8%	10.9%
ConsistencySubsetEval	19.8%	28.2%
GainRatioAttributeEval > 0.01	24.3%	7.8%
GainRatioAttributeEval > 0	14.4%	46.0%

Au vu de ces résultats, nous pouvons dire que la sélection d'attributs n'améliore pas la classification de ce jeu de données. Nous notons toutefois que $GainRatioAttributeEval > 0$ constitue une alternative intéressante car bien qu'il a une valeur de *recall* légèrement plus petite, sa valeur de *precision* est légèrement plus grande. La sélection d'attribut qu'il propose pourrait donc nous permettre de ne pas dépasser le seuil des 800 courriers publicitaires à envoyer. Peut-être même que l'intersection des solutions de ces deux sélections nous permettrais d'avoir de meilleurs résultats.

2.5 Discussion sur trois classificateurs

Maintenant que nous avons décidé quelle sous-ensemble d'attributs nous utiliserons, nous devons choisir le classificateur qui maximise nos valeurs de mesures d'efficacité à savoir *precision* et *recall*. Afin de tester ces classificateurs, nous testons les modèles grâce à un test *10-folds cross-validation*. La Table 11 résume les résultats obtenus par les trois classificateurs que nous considérons. Vous trouverez également ci-dessous une discussion à propos de ceux-ci.

Le classificateur *NaiveBayes* a été retenu car c'est celui qui *a priori* paraissait être le meilleur. C'est pour cette raison que nous l'avions utilisé afin de choisir une sélection pertinente lors du précédent chapitre.

Nous avons également retenu *IBK* car nous l'avions utilisé à plusieurs reprises lors des exercices que nous avons fait dans le but de prendre en main le logiciel *Weka*. Nous l'avons donc testé pour différentes valeurs de *KNN*. Malheureusement, les résultats obtenus sont de plus en plus mauvais quand *KNN* augmente. Les résultats que nous avons retenus dans la Table 11 sont ceux pour *KNN* égal à 1 (qui sont les meilleurs).

Finalement, nous considérons *J48* car nous voulions essayer un classificateur basé sur les arbres et nous l'avions utilisé à plusieurs reprises pendant les exercices. L'arbre qu'il construit ne contient qu'une feuille qui classe toutes les instances dans "pas intéressé par une assurance caravane". Pour éviter cela, nous faisons en sorte que cet arbre ne soit pas élagué. Ce sont les résultats de cet arbre non-élagué qui se retrouvent dans la Table 11.

Table 11 – Résultats obtenus par les différents classificateurs

	Precision	*Recall*
NaiveBayes	13.6%	48.6%
IBK	10.9%	11.2%
J48	14.4%	15.2%

2.6 Choix du classificateur

A la vue des résultats obtenus dans la section précédente, nous pouvons aisément justifier notre choix de *NaiveBayes* en tant que classificateur. En effet, c'est lui qui a la plus grande valeur de *recall* ce qui veut dire que c'est lui qui détecte le plus de "personnes intéressées par une assurance caravane". Malheureusement, sa valeur de *precision* n'est pas très bonne, il est donc très probable que nous devrions sélectionner 800 instances parmi celles que notre modèle sélectionne. Nous n'aurons hélas aucun autre moyen de les sélectionner que de le faire aléatoirement.

2.7 Capacité descriptive du classificateur

Malheureusement, le modèle *NaiveBayes* ne nous indique pas comment il sélectionne les "personnes intéressée par une assurance caravane". Nous n'avons donc aucune idée des attributs utilisés pour déduire si une personne est intéressée ou non par une assurance caravane.

2.8 Méthode de sélection des instances

Afin de sélectionner les instances qui nous intéresse, nous allons lancer notre classificateur sur le jeu de données *ticeval2000.arff* en demandant à *Weka* d'écrire pour chaque instance la classe à laquelle il pense qu'elle appartient ainsi que la probabilité que cette classe soit effectivement correcte. Nous extrairons ensuite les 800 instances appartenant à la classe "est intéressé par une assurance caravane" qui ont la plus grande probabilité d'être vraie.

Nous devrons faire cette extraction nous-même car *Weka* ne permet pas d'exporter directement ses résultats. Nous écrirons donc un script *Python* pour cette tâche. L'étape suivante sera de comparer ces résultats avec le fichier de correction afin d'évaluer notre score.