Datawarehousing et datamining
Projet 2016

UMONS
Faculté des Sciences
MaB1 Sciences - Informatiques
CHARLIER Maimilien

Année académique
2015 - 2016

Table des matières

1 Tutoriel

Tutoriel réalisé seul.

1.1 Introduction To The Explorer Interface

Ex. 17.1.9

> Load the iris data using the Preprocess panel. Evaluate C4.5 on this data using (a) the training set and (b) cross-validation. What is the estimated percentage of correct classifications for (a) and (b) ? Which estimate is more realistic ?

Le pourcentage de classifications correcte estimée pour le training set est de 98 % contre 96 % pour la cross validation. La classification avec le cross-validation est plus réaliste car comme vu en cours elle utilise la k-fold qui sépare les instances en 3 sous-ensemble distinct. Le training set donne de meilleur résultat ici car il utilise les mêmes donnée pour se générer les règles et pour se tester.Dans certain cas, on peut donc avoir des résultats complètement faussé.

Ex. 17.1.10

> Use the Visualize classifier errors function to find the wrongly classified test instances for the cross-validation performed in Exercise 17.1.9. What can you say about the location of the errors ?

En choisissant "x :petalength", on remarque que c'est la longueur des pétales qui définit la classe. on remarque que l'on a deux ensemble de point mal classé, un dans `iris-virgina` plutôt que dans `iris-versicolor` (le carré rouge dans les croix verte). On remarque la même chose dans l'autre sans `iris-versicolor` dans `iris-virgina`.

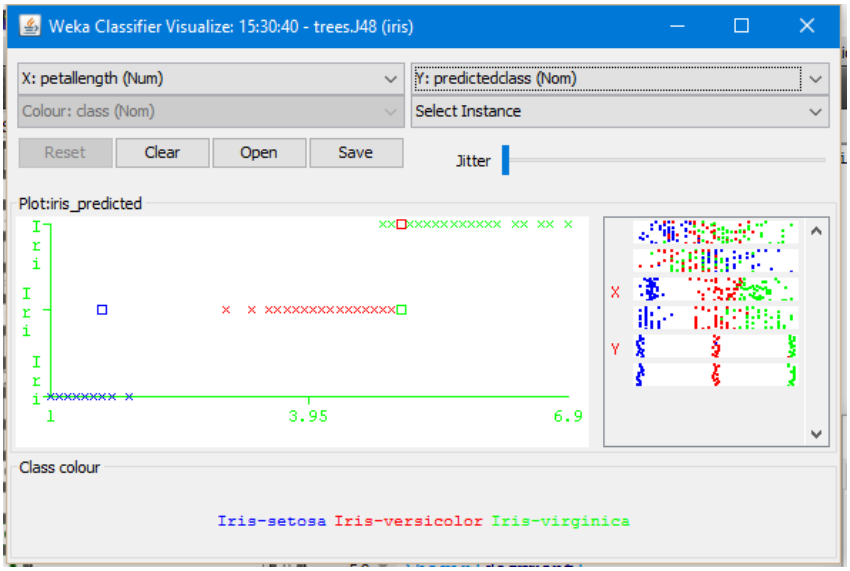


FIGURE 1 – Ex. 17.1.10 - Visualize classifier error

1.2 Nearest-neighbor Learning And Decision Trees

Ex. 17.2.4

> Record in Table 17.1 the best attribute set and the greatest accuracy obtained in each iteration. The best accuracy obtained in this process is quite a bit higher than the accuracy obtained on the full dataset.

Les résultats sont illustré dans le tableau 1, on remarque une croissance de l'exactitude en supprimant les attributs Fe, Si et Al.

Subset Size (No. of Attributes)	**Attributes in "Best"** Subset	**Classification** Accuracy
9	**RI**, **Na**, **Mg**, **Al**, **Si**, **K**, **Ca**, **Ba**, **Fe**	70.56%
8	**RI**, **Na**, **Mg**, **Al**, **Si**, **K**, **Ca**, **Ba**, ~~Fe~~	77.10%
7	**RI**, **Na**, **Mg**, **Al**, ~~Si~~, **K**, **Ca**, **Ba**, ~~Fe~~	77.57%
6	**RI**, **Na**, **Mg**, ~~Al~~, ~~Si~~, **K**, **Ca**, **Ba**, ~~Fe~~	78.97%
5	**RI**, **Na**, **Mg**, ~~Al~~, ~~Si~~, **K**, **Ca**, ~~Ba~~, ~~Fe~~	78.03%
4	**RI**, ~~Na~~, **Mg**, ~~Al~~, ~~Si~~, **K**, **Ca**, ~~Ba~~, ~~Fe~~	77.10%
3	**RI**, ~~Na~~, **Mg**, ~~Al~~, ~~Si~~, ~~K~~, **Ca**, ~~Ba~~, ~~Fe~~	73.83%
2	**RI**, ~~Na~~, **Mg**, ~~Al~~, ~~Si~~, ~~K~~, ~~Ca~~, ~~Ba~~, ~~Fe~~	65.88%
1	**RI**, ~~Na~~, ~~Mg~~, ~~Al~~, ~~Si~~, ~~K~~, ~~Ca~~, ~~Ba~~, ~~Fe~~	47, 19%
0	~~RI~~, ~~Na~~, ~~Mg~~, ~~Al~~, ~~Si~~, ~~K~~, ~~Ca~~, ~~Ba~~, ~~Fe~~	35.54%

TABLE 1 – Exactitude obtenue en utilisant IBk, sur différent sous-ensemble d'attribut.

Ex. 17.2.5

> Is this best accuracy an unbiased estimate of accuracy on future data ? Be sure to explain your answer.

On remarque que dans les données, la présence des attributs Al, Si et Fe entraine une augmentation de l'exactitude. Concernant les autres attributs, leur absences entrainent une chute de l'exactitude. On peut considérer les attributs Al, Si et Fe comme n'étant pas utile pour définir le type de verre.

Ex. 17.2.6 Voir tableau 2.

Percentage Noise	k = 1	k = 3	k = 5
0%	70.56%	71.96%	67.75%
10%	59.34%	63.08%	62.14%
20%	50%	57.94%	53.73%
30%	42.05%	41.58%	44.85%
40%	35.51%	41.12%	41.58%
50%	26.63%	29.90%	31.68%
60%	23.36%	24.29%	23.83%
70%	17.75%	21.02%	14.95%
80%	13.08%	14.48%	13.55%
90%	9.81%	14.95%	11.68%
100%	14.95%	18.22%	18.22%

TABLE 2 – Effet du bruit de classe sur IBk pour différentes tailles de voisinage.

Ex. 17.2.7

What is the effect of increasing the amount of class noise ?

L'ajout de bruit, réduit l'exactitude du tri des éléments du fichier. Par contre l'ajout de bruit espace plus les données ce qui permet en cas de classe très proche, de peut-être mieux pouvoir les discerner, cela si le niveau de bruit reste faible.

Ex. 17.2.8

What is the effect of altering the value of k ?

En présence de bruit de classe, l'utilisation des voisinages permet d'avoir une meilleurs exactitude. On peut donc dire que l'utilisation des voisinages peut améliorer les résultat au classement en présence de bruit.

Ex. 17.2.9

Record in Table 17.3 the data for learning curves for both the one-nearest-neighbor classifier (i.e., IBk with k = 1) and J48.

Voit tableau 3.

Percentage of Training Set	IBk	J48
10%	52.80%	45.32%
20%	63.55%	53.27%
30%	60.28%	59.34%
40%	63.55%	64.95%
50%	62.61%	63.08%
60%	64.48%	69.15%
70%	65.48%	67.75%
80%	67.75%	70.09%
90%	67.75%	69.15%
100%	66.82%	62.23%

TABLE 3 – Effet de la taille de l'ensemble d'entrainement sur IBk et J48.

Ex. 17.2.10

What is the effect of increasing the amount of training data ?

L'incrémentation de la taille de l'ensemble d'entrainement permet d'améliorer l'exactitude de la classification. Cela peut s'expliquer par le fait que l'on peut "poser" plus de question aux données et donc mieux les différencier. Pour J48, l'arbre passe du taille de 11 quand on est à 10% à une taille de 47 éléments quand on est à 100%. On remarque donc que labre s'est complexifié ce qui permet de mieux classifier les éléments.

Ex. 17.2.11

Is this effect more pronounced for IBk or J48 ?

Il est plus prononcé sur J48 l'exactitude augmente plus. Pour un pourcentage de Training Set de 80%, on remarque que l'on augmente l'exactitude sur J48 de $\sim 25\%$ en passant de $\sim 45\%$ à $\sim 70\%$ d'exactitude, alors que sur IBk on passe de $\sim 52\%$ à $\sim 66\%$, soit une augmentation de $\sim 14\%$.

1.3 Classification Boundaries

Ex. 17.3.1

Explain the plot based on what you know about 1R.

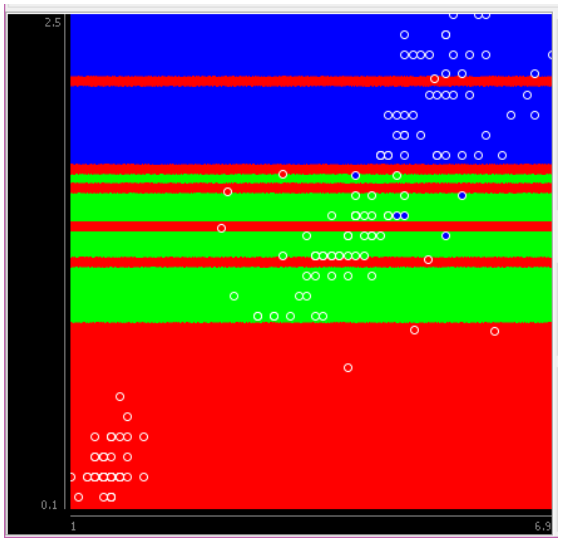
D'après le `classifier output`, les fleurs sont classées par largeur de pétale avec 1R :

```
petalwidth:
< 0.8 -> Iris-setosa
< 1.75 -> Iris-versicolor
>= 1.75 -> Iris-virginica
```

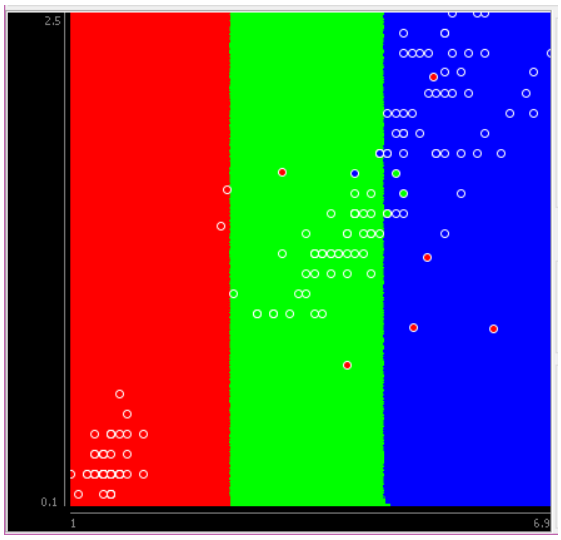
Ex. 17.3.2

Study the effect of the minBucketSize parameter on the classifier by regenerating the plot with values of 1, and then 20, and then some critical values in between. Describe what you see, and explain it.

Étudions l'effet du `minBucketSize` sur la classification, en modifiant les valeurs de ce paramètre on remarque que les résultats du classement sont toujours les mêmes et que les règles ne changent pas dans l'explorer (`classifier output`). Seul un effet visuel sur les zones en font se produit (Figure 2)



`minBucketSize` : 1



`minBucketSize` : 20

FIGURE 2 – Influence de `minBucketSize` sur 1R.

Ex. 17.3.3

You saw earlier that when visualizing 1R the plot always has three regions. But why aren't there more for small bucket sizes (e.g., 1) ? Use what you know about 1R to explain this apparent anomaly.

Avec un `minBucketSize` de 1, il n'y a que 3 grandes zones de couleurs dans le graphiques, cela s'explique du fait que la classification se fait en fonction des classes et il n'y en a que 3. On peut voir des lignes rouge couper la zone verte dû à la présence des plusieurs éléments de la classe dans la zone verte et que ceux si sont pris en compte par des tampon de taille 1.

Ex. 17.3.4

> Can you set minBucketSize to a value that results in less than three regions ? What is the smallest possible number of regions ? What is the smallest value for minBucketSize that gives this number of regions ? Explain the result based on what you know about the iris data.

On ne peut pas avoir moins de 3 régions peut importe la valeur de `minBucketSize` du au fait que l'on a 3 classes. La classe `iris-setona` défini la valeur minimal de `minBucketSize` car elle a des éléments unique dans d'autre classe. Avec une valeur de `minBucketSize` de 2, ont obtient bien 3 régions distincte. En effet, il n'y aura pas deux éléments successifs de la classe iris-setona dans les classes voisines.
Sans faire d'affichage, on peut choisir une valeur de 7 pour `minBucketSize` en se basant sur la matrice de confusion. Cela assurera d'avoir 3 régions distincte, car il y a au maximum 6 éléments "intru" dans les classes voisines. Dans notre cas, ses éléments ne sont pas successif donc la valeur de `minBucketSize` peut être inférieur à 6.

```
=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 44  6 |  b = Iris-versicolor
  0  6 44 |  c = Iris-virginica
```

Ex. 17.3.5

> With k = 1, which is the default value, it seems that the set of k-nearest neighbors could have only one member and therefore the color will always be pure red, green, or blue. Looking at the plot, this is indeed almost always the case : There is no mixing of colors because one class gets a probability of 1 and the others a probability of 0. Nevertheless, there is a small area in the plot where two colors are in fact mixed. Explain this.

L'affichage de Iris 2D avec IBk avec k = 1, nous donne 3 grand régions de couleurs unique et une région où le vert et le bleu sont superposé. Cela est du au fait que 2 éléments bleu et 1 élément vert s'y superpose comme on peut le voir à la figure 3 en utilisant l'outil de visualisation.

Ex. 17.3.6

> Experiment with different values of k, say 5 and 10. Describe what happens as k increases.

Dans IBk plus k sera grand, plus le nombre de voisin pris en compte sera grand. Cela aura donc comme effet de faire un dégradé sur les frontière de classe cas les éléments en frontière vont avoir des voisins d'une autre classe. Plus le k sera grand, plus la frontière sera lissée (figure 4).

Ex. 17.3.7

> Experiment with different values of k, say 5 and 10. Describe what happens as k increases.

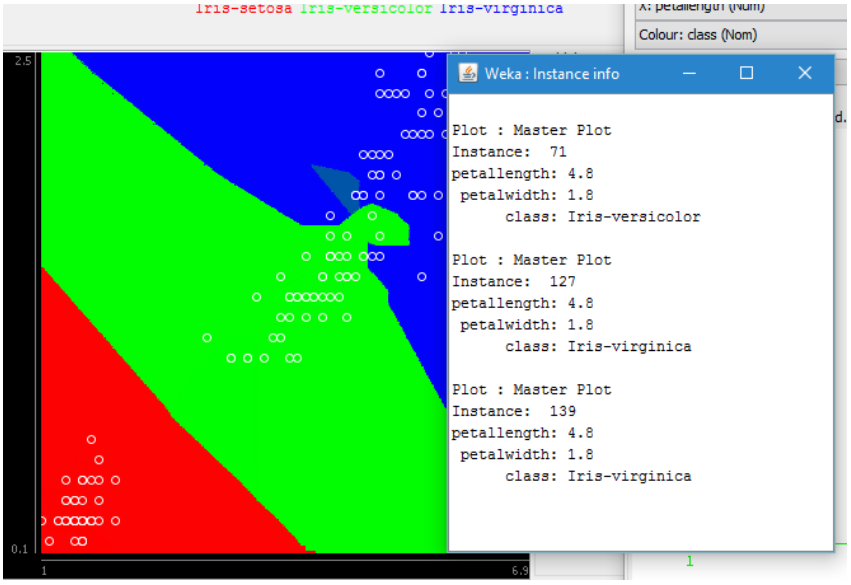


FIGURE 3 – Affichage de Iris 2D avec IBk.

La figure 5 illustre l'affichage de Iris 2D avec NaïveBayes. Comme on peut le voir les classes sont bien colorée avec leur couleur là où leurs éléments se situent. Pour ce qui est de la couleurs des autres rectangles, prenons par exemple celui en haut à gauche. Celui-ci est mauve, chaque case étant la multiplication des éléments (bleu dans ce cas) présent sur la ligne et des éléments (rouges dans ce cas) présent sur la colonnes.

Ex. 17.3.8

> What do you see ? Relate the plot to the output of the rules that you get by processing the data in the Explorer.

L'affichage des données de Iris 2D avec JRip (fig 6) donne 3 zones continue représentant les 3 classes.
D'après l'explorer, elle se base sur 3 conditions :

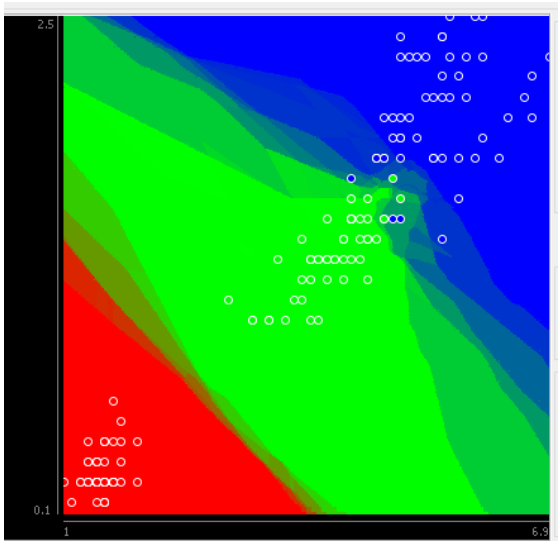
- (petallength <= 1.9) => class=Iris-setosa (50.0/0.0)
- (petalwidth <= 1.6) and (petallength <= 4.9) => class=Iris-versicolor (47.0/0.0)
- => class=Iris-virginica (53.0/3.0)

Ex. 17.3.9

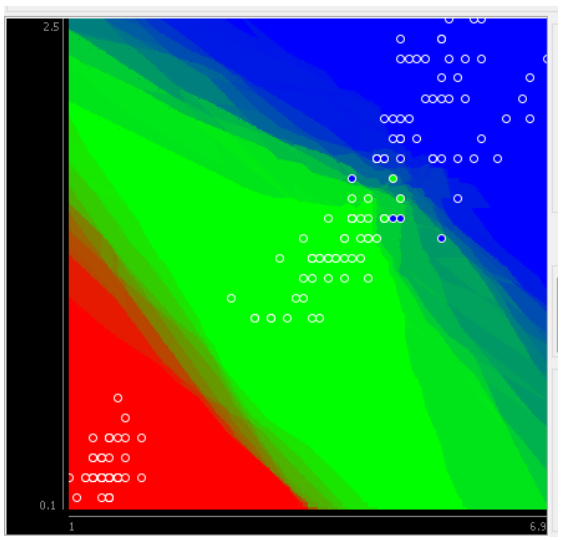
> The JRip output assumes that the rules will be executed in the correct sequence. Write down an equivalent set of rules that achieves the same effect regardless of the order in which they are executed. Generate a plot for J48, with default options.

Pour que les règles puissent être exécutée sans tenir compte de l'ordre il faudrait quelle soi comme suit :

- (petallength <= 1.9) => class=Iris-setosa
- (petalwidth <= 1.6) and (petallength <= 4.9) and (petallength > 1.9) => class=Iris-versicolor



k : 5



k : 10

FIGURE 4 – Influence de k sur IBk.

— =>(petalwidth > 1.6) and (petallength > 4.9) ==> class=Iris-virginica

Ex. 17.3.10

What do you see ? Again, relate the plot to the output that you get by processing the data in the Explorer interface. One way to control how much pruning J48 performs is to adjust the minimum number of instances required in a leaf, minNumObj.

L'affichage de Iris 2D avec J48 (fig. 7) donne plus de zone car l'arbre donne plus de condition que ce que fait Jrip. Voici le processus de décision utilisé dans l'arbre :

```
petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
|   petalwidth <= 1.7
|   |   petallength <= 4.9: Iris-versicolor (48.0/1.0)
|   |   petallength > 4.9
|   |   |   petalwidth <= 1.5: Iris-virginica (3.0)
|   |   |   petalwidth > 1.5: Iris-versicolor (3.0/1.0)
|   petalwidth > 1.7: Iris-virginica (46.0/1.0)

Number of Leaves  :  5

Size of the tree :  9
```

On a donc des feuilles avec 50, 49, 46, 4 et 3 éléments. Les deux feuilles avec 3 et 4 éléments pourraient être regroupée car elle n'affine pas le classement.

Ex. 17.3.11

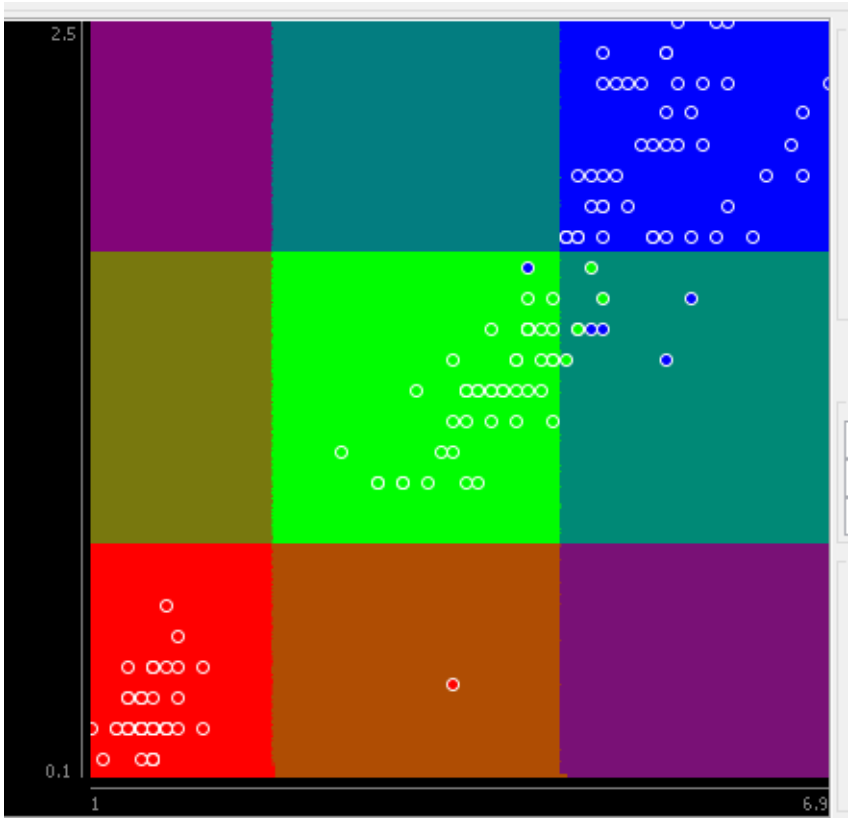


FIGURE 5 – Affichage de Iris 2D avec NaïveBayes.

> Suppose you want to generate trees with 3, 2, and 1 leaf node, respectively. What are the exact ranges of values for minNumObj that achieve this, given default values for the other parameters ?

Pour générer un arbre avec seulement 3, 2 ou une feuilles, les valeurs pour `minNumObj` serait de :

- 1 feuille uniquement : la valeur de `minNumObj` est de 76 à 150 le nombre totale d'instance.
- 2 feuilles uniquement : comme "`petalwidth <= 0.6`" regroupe déjà 50 cas, on peut utiliser une valeur de 50. On peut donc choisir un range pou `minNumObj` entre 50 et 75.
- 3 feuilles uniquement : `minNumObj` avec une valeur entre 7 et 49. Avec un valeur de 47, on a l'explication suivante : on a `petalwidth <= 0.6` qui regroupe 50 cas dans une branche, puis dans la seconde que l'on doit diviser en deux on veut à avoir la plus grande feuille qui contient 47 élément "`petalwidth > 1.7: Iris-virginica (46.0/1.0)`", le reste sera forcement regroupé dans une dernière feuille.

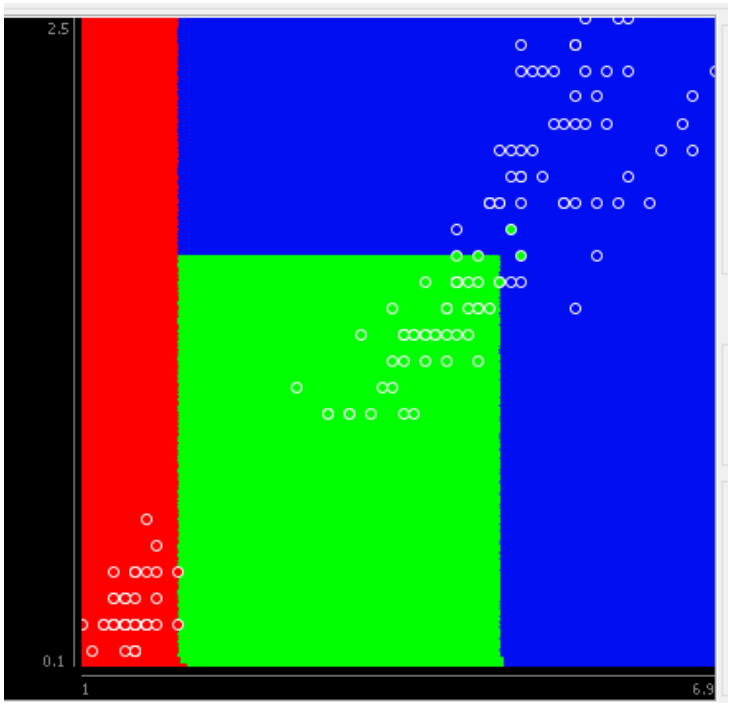


FIGURE 6 – Affichage de Iris 2D avec JRip.

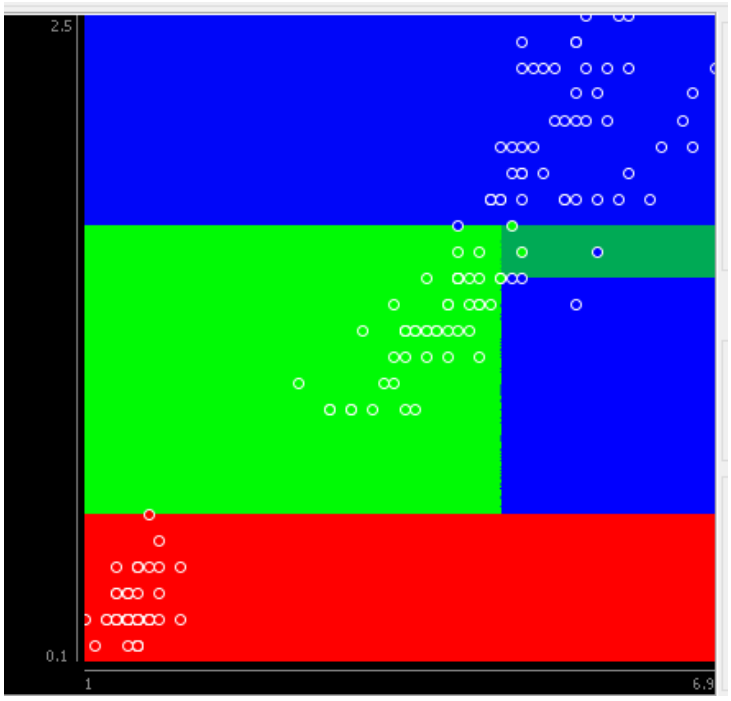


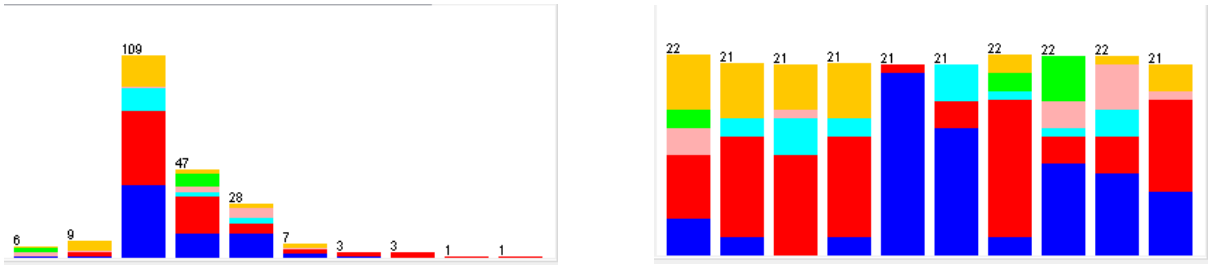
FIGURE 7 – Affichage de Iris 2D avec J48.

1.4 Preprocessing And Parameter Tuning

Ex. 17.4.1 - glass.arff

> What do you observe when you compare the histograms obtained ? The one for equal-frequency discretization is quite skewed for some attributes. Why ?

En appliquant la discrétisation selon deux méthodes (fig 8) à savoir en égalisant la taille des intervalles (*equal widht*) ou en égalisant les nombres de répétition (*equal fréquency*), ont obtient deux histogramme totalement différent. Dans le premier cas (*equal widht*), on aura une taille des bâtons proportionnel a la représentation des valeur d'attribut dans les intervalles. On pourra donc obtenir des bâtons plus grand si une intervalles contient plus d'instance qu'une autre intervalle. Dans le cas de l'*equal frequency*, tout les bâtons auront sensiblement la même taille car on aura égaliser l'histogramme en regroupant les éléments par répétition afin justement d'avoir un histogramme plat. Note : Cela ne s'applique localement qu'à un attribut, donc pour certain attribut nous n'auront pas d'histogramme plat. Exemple avec un attribut où toute les instances ont exactement la même valeur, on obtient un attribut sans importance car ne représente rien, l'histogramme n'aura qu'un bâton même si l'on applique *equal frequency*.



Equal width Equal frequency

FIGURE 8 – Discrétisation selon deux méthodes.

Ex. 17.4.2 - iris.arff

> Based on the histograms obtained, which of the discretized attributes would you consider to be most predictive ? Reload the glass data and apply supervised discretization to it.

L'affichage des histogrammes de iris après l'application du filtre *weka.filters.supervised.attribute.Discretize* en mode par défaut (fig 9) nous montre que l'attribut `petallength` et `petalwidth` sont les plus prédictible par rapport à la classe car chaque bâton des histogrammes sont presque complètement de la couleur de leur classe.

Ex. 17.4.3 - glass.arff

> For some attributes there is only a single bar in the histogram. What does that mean ?

Pour certain attribut de *glass.arff*, on constate qu'il n'y a plus qu'un seul bâton dans l'histogramme après avoir appliquer le filtre de discrétisation basé sur les classes. Pour l'expliquer, il faut regarder la répartition des instances dans l'attribut visé avant

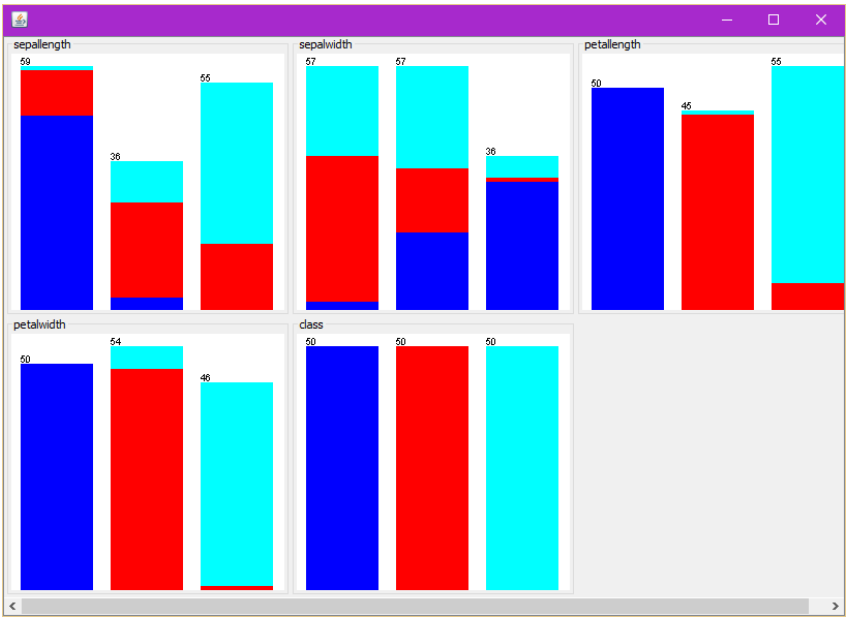
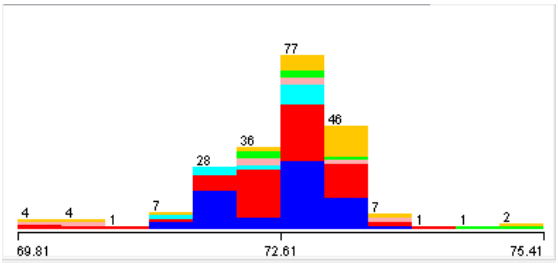
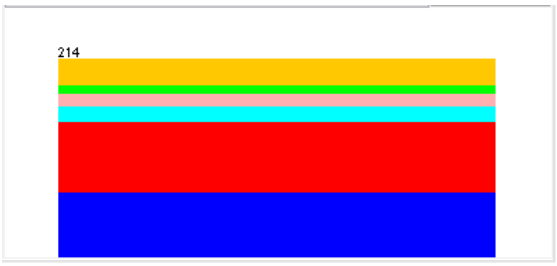


FIGURE 9 – Affichage des histogrammes Iris 2D après une discrétisation des attributs par *equal widht* .

l'application du filtre. Par exemple pour Si (fig 10), qui après avoir appliquer le filtre n'a plus qu'un seul bâton, on constante qu'avant pour chaque bâton de l'histogramme on a une répartition uniforme pour chaque classes (les couleurs sont repartie uniformément dans chaque bâton). Après l'application du filtre comme pour chaque intervalle dans l'attribut on avait autant d'instance de chaque classe, l'algorithme n'a eu d'autre choix que de les mettre tous dans le même intervalle. Autrement dit, les attributs ayant uniquement un seul bâton ne sont pas représentatif des classes. On peut donc les considérer comme du bruit.



(a) avant



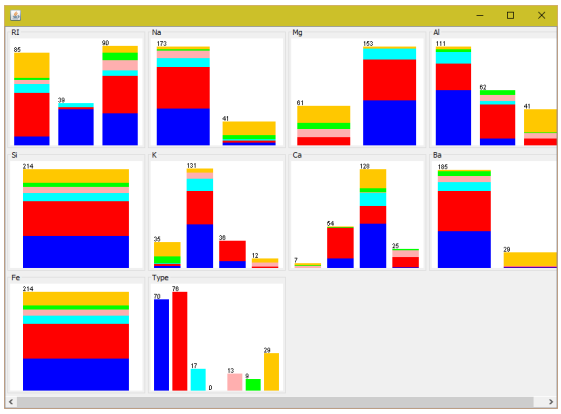
(b) après

FIGURE 10 – Histogramme de l'attribut Si de glass.arff avant et après avoir appliquer le filtre de discrétisation.

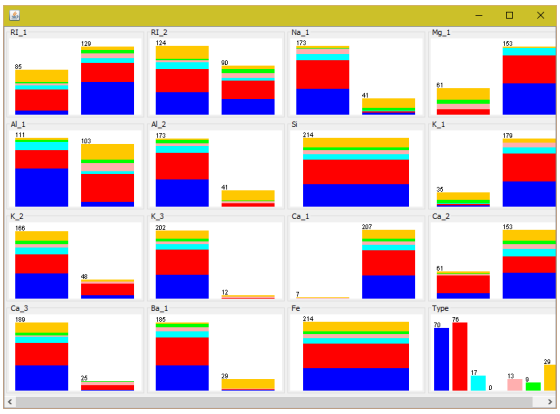
Ex. 17.4.4 - glass.arff

Choose one of the filters and use it to create binary attributes. Compare the result with the output generated when makeBinary is false. What do the binary attributes represent ?

En appliquant le filtre de discrétisation sur le fichier *glass.arf* en activant et désactivant l'option *makeBinary*, on constante une différence. Les attribut ayant plus de deux bâtons sont scindé en plusieurs histogramme avec deux bâtons quand l'option *makeBinary* est activé (fig 11). Par exemple, l'attribut **Al** qui avait 3 bâtons avec 3 intervalles différent a, b, c est scindé en deux histogrammes, le premier aura deux bâtons : a et b+c, le second aura aussi deux bâtons : a+b et c. On remarque donc qu'il ne s'agit que d'une façon de représenter les données mais qu'il n'y a pas de perte de donnée en faisant cela car on peut reconstruite l'histogramme à 3 bâtons à partir des deux histogrammes à 2 bâtons.



(a) false



(b) true

FIGURE 11 – Histogramme des attributs de glass.arff en appliquand le filtre de discrétisation avec ou sans l'option *makeBinary*.

Ex. 17.4.8 - labor.arff

> Apply the ranking technique to the labor negotiations data in labor.arff to determine the four most important attributes based on information gain.

En utilisant CfsSubsetEval sur le fichier *labor.arff*, on obtient une liste trier des attribut en fonction de leurs importances.
Les 4 premiers sont :

- `wage-increase-first-year`
- `wage-increase-second-year`
- `cost-of-living-adjustment`
- `statutory-holidays`

1.5 Document Classification

Ex. 17.5.1

> Make an ARFF file from the labeled mini-documents in Table 17.4 and run StringToWordVector with default options on this data. How many attributes are generated ? Now change the value of the option minTermFreq to 2. What attributes are generated now ?

Après avoir appliqué le filtre *StringToWordVector* en partant du table 17.4 du tutoriel on obtient 35 attributs en en ayant uniquement deux à l'origine.
En changeant la valeur de *minTermFreq* à 2, on obtient 6 attributs que voici :

- class
- crude
- oil
- supply
- of
- the

Ces attributs sont présent au moins deux fois dans les phrases originales, ceux présent uniquement une fois n'ont pas été retenu comme attribut.

Ex. 17.5.4

> Build classifiers for the two training sets by applying FilteredClassifier with StringToWordVector using (1) J48 and (2) NaiveBayesMultinomial, evaluating them on the corresponding test set in each case. What percentage of correct classifications is obtained in the four scenarios ? Based on the results, which classifier would you choose ?

Le tablea 4 illustre les pourcentages de classification correcte obtenu dans les 4 cas. On constate que J48 obtient de meilleurs résultat. Je préférerai donc choisir le classeur J48.

Collection	**J48**	**NaiveBayesMultinomial**
ReutersCorn	97.35%	93.70%
ReutersGrain	96.35%	90.72%

TABLE 4 – Pourcentage de classification correcte en fonction de deux algorithmes sur deux collections.

Ex. 17.5.6

> Which of the two classifiers used above produces the best AUC for the two Reuters datasets ? Compare this to the outcome for percent correct. What do the different outcomes mean ?

La table 4 nous donne le pourcentage de bonne classification et la table 5 nous donne l'AUC. On constate que J48 donne une meilleurs classification mais que sont AUC est

moins bon que celui de NaiveBayesMultinomial. D'après le cours, plus l'AUC est grand, plus le modèle est correcte. Celui-ci est donc plus important que le pourcentage de classifications correcte. NaiveBayesMultinomial est donc meilleurs dans ce cas-ci.

Collection	**J48**	**NaiveBayesMultinomial**
ReutersCorn	0.694	0.952
ReutersGrain	0.906	0.973

TABLE 5 – AUC en fonction de deux algorithmes sur deux collections.

2 Travail libre

Le travail libre consiste à utiliser les concept vu précédemment et permettant de maitriser les bases de Weka pour effectuer une classification de donnée sur un ensemble de donnée réel. Cette ensemble de données à été choisis par l'étudiant et valider par le professeur. Le but de la classification est de comparer différent algorithme et de déduire la quelle des classifications est la meilleur.

2.1 Présentation du jeu de donnée

Le jeu de donnée choisi est "SMSSpamCollection" ce fichier contient 5574 SMS classe par type *Spam* ou *légitime*. Ce jeu de donnée est issue du site de l'UCI SMS Spam Collection Data Set. Le fichier présent sur le site n'était pas formaté pour Weka, on peut voir le début de celui-ci en annexe Figure 19. Il a donc fallut formater le fichier pour être compatible avec Weka comme illustré en annexe figure 20.
On a donc deux attributs(illustré en figure 12) pour 5574 instances :

1. **sms-class** qui est la classe de l'ensenlble de donnée : Spam ou légitime (ham).
2. **sms-text** qui est le contenu des sms.

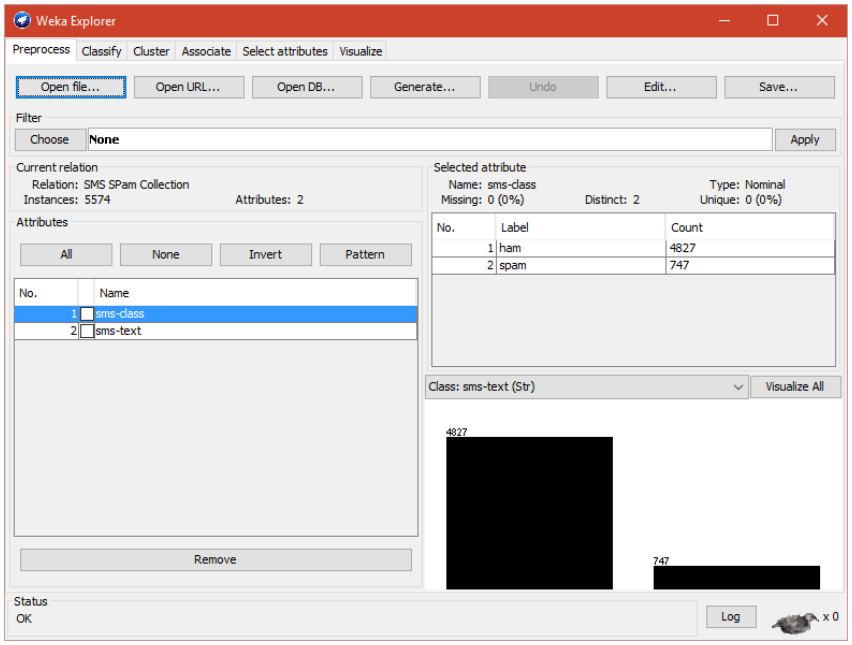


FIGURE 12 – "SMSSpamCollection" après import dans Weka.

2.2 Classification

Nous avons commencer par vouloir appliquer des algorithmes de classification comme iBK ou J48. Seulement ceux-ci ne s'appliquent pas sur des phrases. Il a donc fallut appliquer un filtre séparant les phrases en attributs d'un seul mot. Nous avons donc utilisé le filtre `weka.filters.unserpervised.attribut.StringToWordVector` avant

d'appliquer les algorithmes. En appliquant le filtre avec sa valeur par défaut on obtient 1028 attributs contre seulement deux à la base pour 5574 instances. En visualisant les attributs, on remarque que leurs répétitions est très faible (proche de 0) et que leur répartition n'est pas indicative de leur classe. En figure 13, on remarque que pour chaque attribut, on a une répartition uniforme de la classe. Aucun attribut ne dégage donc la classe directement.

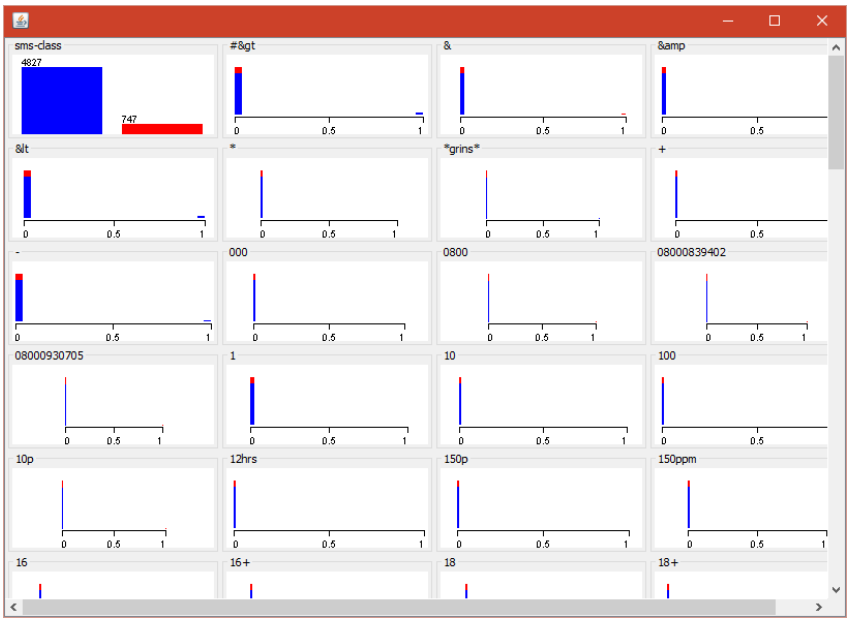


FIGURE 13 – Visualisation des attributs de "SMSSpamCollection" après application du filtre StringToWordVector dans Weka.

Nous avons ensuite appliqué quelques algorithmes de classification et allons illustrer dans un tableau. Ce tableau est illustré en Table 6, on remarque que `ZeroR` n'arrive pas à classifier les SMS. Quand à `J48` et `IBk` les résultats sont correctes d'un point de vue pourcentage, mais au niveau de la matrice de confusion ce n'est pas correcte du tout. Les matrices de confusion des deux algorithmes sont illustré en figure 14, on peut y voir que énormément d'élément sont mal classé, on retrouve notamment trop de SMS légitime classé comme SPAM.

Algorithme	**Instance correctement classifié** (%)	**Erreur relative absolue** (%)
OneR	88.66%	48.82%
J48	95.94%	25.90%
IBk	95.22%	24.89%

TABLE 6 – Première classification des SMS.

Nous avons donc choisi, d'essayer d'améliorer le classement. Premièrement nous avons changé le paramètre répétition des mots dans le filtre StringToWordVector. Avec une

```
=== Confusion Matrix ===

    a    b   <-- classified as
 4770   57 |    a = ham
  169  578 |    b = spam
```

(a) **J48**

```
=== Confusion Matrix ===

    a    b   <-- classified as
 4821    6 |    a = ham
  260  487 |    b = spam
```

(b) **IBk**

FIGURE 14 – Matrice de confusion obtenu après l'application des algorithmes.

valeur de 180 on obtient moins d'attributs, seulement 85 contre plus de 1000 plus tôt. Les valeurs de paramètres de répétition inférieur 180 n'ayant presque pas d'influence sur le nombres d'attributs crée. Le tableau 7 illustre les résultats, ceux-ci sont moins bon que précédemment, du à la perte de consistance des données en appliquant un filtre trop restrictif.

Algorithme	**Instance correctement classifié** (%)	**Erreur relative absolue** (%)
OneR	88.66%	48.82%
J48	93.70%	39.17%
IBk	94.79%	28.21%

TABLE 7 – Deuxième classification des SMS.

Les SMS ont peut-être des caractéristiques que l'on pourrait isoler en observant les données. Nous avons donc parcouru rapidement les messages contenues dans l'ensemble de donnée. Nous avons remarqué que les SPAMs contenaient des mots en majuscules, des numéros de téléphones et plus de ponctuation que la normal. On a donc décidé de passé rajouter des attributs illustrant ses caractéristiques. Pour ce faire, le jeu de donné a été mis en forme sous format `.csv` et traité dans python. Dans python nous avons parcours tout les SMS et pour chaque SMS nous avons compté certain caractères pour ensuite ajouté ses comptages comme attributs. Voici les différents attribut ajouté :

- **nb-maj** représentant le nombre de majuscule contenu dans le SMS correspondant.
- **nb-chiffre** représentant le nombre de digit contenu dans le SMS correspondant.
- **nb-quest** représentant le nombre de point d'interrogation "?" ou d'exclamation "!" contenu dans le SMS correspondant.
- **nb-point** représentant le nombre de point "." contenu dans le SMS correspondant.
- **nb-caract** représentant le nombre de caractère contenu dans le SMS correspondant.

Nous avons ensuite remis le fichier obtenu au format `.arff` comptable avec Weka et ouvert le fichier dans celui-ci. La figure 15 illustre les différent attribut en fonction de la classe. On remarque que l'attribut `nb-maj` est représentatif d'un partie des sms spam.

On le remarque du fait que la partie droite de l'attribut, représentant un nombre élevé de majuscules, n'est peuplé que des SMS spam. À l'inverse, un nombre de points élevé (attribut `nb-point`) illustre des SMS légitime. Pour les 3 autres attributs ajouté, la répartition est uniforme donc on ne peut rien en déduire.

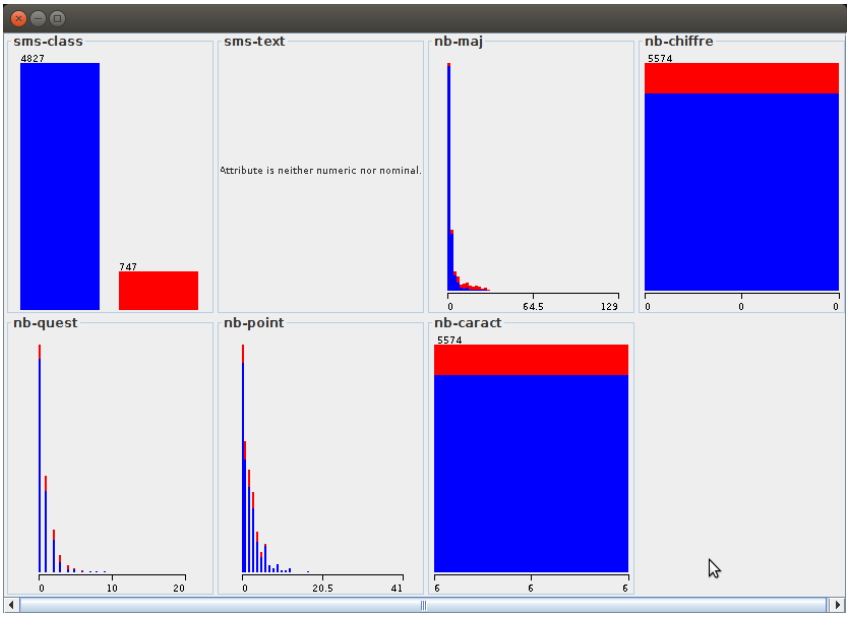


FIGURE 15 – "SMSSpamCollection" après import dans Weka.

Nous avons donc raffiné les données ce qui devrait permettre d'obtenir de meilleurs résultats. Comme précédemment, nous allons appliquer le filtre `weka.filters.unserpervised.attribut.StringToWordVector` sur la collection, pour scinder le contenu des SMS en attribut. Nous allons utiliser les paramètres par défaut afin de garder le plus d'information sur les données possible. On a vu plus haut que de mettre un paramètre plus grand faisait perdre des informations. Par contre, cela induit un temps de traitement beaucoup plus court ce qui en pratique pourrait être utile. Nous allons maintenant faire l'analyse complète avec 3 algorithmes : `rules maintenant faire s.OneR`, `tree.J48` et `lazy.IBk`. Les algoritmes vont être tutliser avec l'option "training set" plus tôt, on a vu que cette option était meilleurs que les autres. Nous allons travailler sur un ensemble contenant 5574 instances et 1838 attributs.

2.2.1 Algorithme J48

Il s'agit d'un algorithme basé utilisant un arbre de décision, celui-ci a été vu au cours et utilisé plus tôt dans le tutoriel. Nous avons laisser tout les parramètre par défaut car ceux-ci sont les plus adaptés comme on a pu le voir dans le tutoriel. L'algorithme nous retourne un arbre d'une très grand taille du au nombre d'attributs présent dans le jeu de donnée. Celui-ci comprend 83 feuilles et a une taille de 165. De ce fait, il n'est pas possible de l'illustré sur format papier. La figure 16 illustre le résumé des résultats obtenu dans Weka. On remarque que l'erreur relative est de 20% soit 5% qu'au départ. On remarque que l'on a très peu de SMS légitime classé en spam, mais environs 1/7 des

SPAM sont classé comme légitime. Le ROC est asses élevé, ce qui montre une fiabilité du modèle.

```
Correctly Classified Instances        5383               96.5734 %
Incorrectly Classified Instances       191                3.4266 %
Kappa statistic                          0.8475
Mean absolute error                      0.0459
Root mean squared error                  0.1769
Relative absolute error                 19.7789 %
Root relative squared error             51.9261 %
Total Number of Instances             5574

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
0.986     0.166      0.975     0.986     0.98       0.941    ham
0.834     0.014      0.903     0.834     0.867      0.941    spam
Weighted Avg.
0.966     0.146      0.965     0.966     0.965      0.941

=== Confusion Matrix ===

a    b   <-- classified as
4760   67 |    a = ham
124  623 |    b = spam
```

FIGURE 16 – Résumé de l'algorithme J48 sur le jeu de donnée final.

2.2.2 Algorithme IBk

L'algorithme IBk se basse sur ses voisins, le paramètre `k` de celui-ci permet de choisir le nombre de voisinage à prendre en compte pendant son exécution. Weka permet de sélectionner automatiquement le meilleurs nombre de `crossValidation` en activant l'option `crossValidate` à `True`. Nous avons activé cette option, puis nous avons fait varier le nombre de voisinage choisi en faisant varier `k` afin d'avoir les meilleurs résultats possible. Nous avons constaté que les résultats n'était pas influence par cette modification de visionnage. En figure 17, nous illustrons le résumé des résultats obtenu dans Weka. Le pourcentage d'instance correctement classifié est de 95.53% un résultat presque identique mais légèrement meilleurs que celui obtenu à la base. Comparé à J48 le pourcentage d'instance correctement classifié est plus faible. Le ROC est lui aussi plus faible, il est de 0.86 contre 0.941. Si l'on observe les matrices de confusion, on remarque que le nombre de faux positif est deux fois plus élevé pour les spams.

```
Correctly Classified Instances        5325               95.5328 %
Incorrectly Classified Instances       249                4.4672 %
Kappa statistic                          0.7812
Mean absolute error                      0.0445
Root mean squared error                  0.2099
Relative absolute error                 19.1633 %
Root relative squared error             61.624  %
Total Number of Instances             5574

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
0.996     0.309      0.954     0.996     0.975      0.865    ham
0.691     0.004      0.966     0.691     0.806      0.865    spam
Weighted Avg.
0.955     0.268      0.956     0.955     0.952      0.865

=== Confusion Matrix ===

a    b   <-- classified as
4809   18 |    a = ham
231  516 |    b = spam
```

FIGURE 17 – Résumé de l'algorithme J48 sur le jeu de donnée final.

2.2.3 Algorithme OneR

L'algorithme OneR se base sur un seul attribut pour classifier les données. On l'a vu plus tôt, il n'est pas possible de dégager un attribut unique permettant de classifier directement les SMS. Les résultats ne seront donc pas très bon. Après application de l'algorithme, on remarque que l'attribut choisi pour la classification est l'attribut "nb-maj". Cet attribut avait été mis en évidence plus tôt, car on pouvait dégager une tendance pour les spams. Ceux-ci contenant plus de majuscules que les messages légitime. On constate aussi que le travail de raffinage via python n'a était inutile car il est directement utilisé dans cette algorithme. En figure **??**, nous illustrons le résumé des résultats obtenu dans Weka. On constate que le nombre de faux négatif pour les SPAM est de presque 50%. Pour les autres valeurs, elles sont toute plus faible que pour les autres algorithmes.

2.2.4 Comparaison

Nous avons construit un tableau permettant de mette en avant certaines métriques des 3 algorithmes appliqué à nos données. On constate que l'AUC de J48 est beaucoup plus grand que pour les autres algorithmes. Comme les SPAM qui sont minoritaire et qui ne représente que $13,40\%$ du total des instances, nous avons choisi de mettre en avant le %

```
Correctly Classified Instances        5057               90.7248 %
Incorrectly Classified Instances       517                9.2752 %
Kappa statistic                          0.5752
Mean absolute error                      0.0928
Root mean squared error                  0.3046
Relative absolute error                 39.942  %
Root relative squared error             89.3984 %
Total Number of Instances             5574

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
0.957     0.416      0.937     0.957     0.947      0.77     ham
0.584     0.043      0.679     0.584     0.628      0.77     spam
Weighted Avg.
0.907     0.366      0.902     0.907     0.904      0.77

=== Confusion Matrix ===

a    b   <-- classified as
4621  206 |    a = ham
311  436 |    b = spam
```

FIGURE 18 – Résumé de l'algorithme J48 sur le jeu de donnée final.

de SMS correctement classé. On remarque que pour J48, on a plus de 80% de SMS SPAM correctement bloqué et 98% de SMS légitime correctement classé. Pour les autres algorithmes ce pourcentage chute fortement, et le nombre de faux positif au niveau des SPAMs est trop important.

Algorithme	**Instance correctement classifié** (%)	**AUC**	**Spam correctement classifié** (%)	**SMS légitime correctement classifié** (%)
J48	96.5734	0.941	83, 41	98, 62
IBk	95.5328	0.865	69, 08	99, 62
OneR	90.7248	0.77	58, 36	95, 73

TABLE 8 – Récapitulatif des 3 algorithmes.

2.3 Conclusion

Ce projet a permis d'apprendre à utiliser *Weka* et de renforcer nos connaissance et notre compréhension du cours de *Data Warehousing & Data Mining*. Un algorithme basé sur l'utilisation d'un arbre de décision permis d'optimiser l'utilisation de plusieurs attributs. Dans notre cas, étant donné qu'on en avait plus de 1000, cela permis de raffiner le processus de décision au maximum. Dans le as de la collection SMS SPAM, J48 permet un meilleur classement des SPAM avec un taux de 83% de SPAM correctement classifié. Par contre IBk classifie les SMS légitime de façon presque identique. Avec l'indicateur AUC, on peut dire avec certitude que le modèle généré avec J48 est plus fidèle que les autres modèles.

3 Annexes

```
ham Go until jurong point, crazy.. Available only in bugis n great world la e buffet.
ham Ok lar... Joking wif u oni...
spam Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 8
ham U dun say so early hor... U c already then say...
```

FIGURE 19 – Début de "SMS Spam Collection Data Set" issu de l'UCI.

```
@RELATION 'SMS SPam Collection'

@attribute sms-class {ham, spam}
@attribute sms-text string

@data

"ham", "Go until jurong point, crazy.. Available only in bugis n great world la e buf
"ham", "Ok lar... Joking wif u oni..."
"spam", "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA
"ham", "U dun say so early hor... U c already then say..."
```

FIGURE 20 – Début de "SMSSpamCollection" après modification pour être compatible avec Weka.