Extraction de Connaissances B.E. 2

Manipulation des méthodes d'apprentissage avec WEKA

Règles de Classification & d'Association,
Méthodes Bayésiennes (suite)
Régression Logistique, Ridge,
Random Forest, SVM

Expérimentation et comparaison statistique de méthodes

ECL - Option Informatique - 2024/2025

Nov-Déc 2024

Table des matières

☞ **A noter :**

Ce BE est composé de plusieurs parties : 3 travaux à rendre & plusieurs Bonus

1- Complément d'arbres de décision : arbres de régression
2- Règles de Classification et d'association
3- La méthode Bayésienne (appliquée aux Spams)
4- Plusieurs Bonus :
- méthodes "Random Forest" et "Ridge",
- Méthodes "Régression Logistique" et "SVM".

5- Travail Optionnel :
- Comparaison de méthodes (à voir aussi en BE3)

1 Classification : obtention d'arbre de régression

Les méthodes ci-après font suite au BE1 où nous n'avons pas étudié les *arbres de régression* (cas où la classe est numérique). Ces règles peuvent également être utilisées pour participer à une évaluation / comparaison de résultats de classification.

☞ **Il est important d'avoir à l'esprit** que la forme souhaitée des résultats d'un apprentissage est un facteur important dans la tache de fouille de données. Il y a des méthodes qui donnent de très bons résultats mais ne les expliquent pas (par une règle / une table / ...). Pour respectables qu'elles soient, si nous avons besoin d'une explication d'expert, ces méthodes ne nous intéresseront pas !

1.1 Arbres de régression

Rappel : la méthode *simpleCart* de WEKA ne construit pas d'arbre de régression ; elle produit un arbre binaire avec une classe binaire NON numérique et n'accepte pas une BD avec une classe numérique.

Nous allons utiliser ici la méthode **RepTree** de WEKA : *Regression Pruned Tree*. Elle s'applique aux classes numériques ou discrètes (catégorielles, énumérées).
Pour une explication de la méthode, voir la section 1.3 en page 4.

1.2 Comprendre et appliquer la méthode REPTree

Cette méthode utilise l'algorithme de création d'un arbre **binaire** de classification ou de régression vu en cours. Elle crée de multiples arbres dans différentes itérations avant d'en choisir le "meilleur". La mesure utilisée pour l'élagage est la moyenne de moindre carrée (**MSE**) des prédictions.

Il s'agit d'un concept général de construction d'arbre élagué par réduction d'erreur (*REP : Reduced Error Pruning*).
Rappelons que l'élagage est déjà utilisé dans des méthodes telles que C4.5 basée sur l'entropie pour la classification discrète.
De même, la méthode CART (vue en cours, à ne pas confondre avec *SimpleCart* proposée par Weka) basée sur la réduction de la variance (pour le cas d'une régression : classe numérique, voir cours) utilise l'élagage.

☞ Notons que dans REPTree, les valeurs manquantes sont gérées comme pour la méthode C4.5.
☞ Pour une classe discrète (cf. classification), REPTree utilise le gain d'information (à maximiser localement).
☞ Pour une classe numérique (cf. régression), elle utilise la variance (dont le totale globale est minimisé).

1.3 Les principes de la méthode RepTree

La méthode REPTree (Reduces Error Pruning) est une méthode rapide de classification / régression qui crée de multiples arbres en suivant la logique des arbres de régression **binaires** dans ses différentes itérations et choisit le **meilleur**. Le taux d'erreur calculé est l'erreur **moindre carrée**.

Pour les données non numériques, un arbre de décision est crée (on utilise l'entropie et *le gain d'information*) et la technique appliquée réduit l'erreur en maximisant l'entropie.
Pour les données numériques, c'est la variance qui est réduite (comme dans CART).

RepTree ordonne d'abord et une seule fois les attributs numériques. Elle génère plusieurs arbres (de régression) pendant ses itérations pour en choisir ensuite le meilleur. Pour construire "ses arbres", RepTree scinde les instances en différentes parties. La méthode exige donc un espace mémoire important mais en contre partie, l'arbre obtenu est plus petit (concis) et plus "juste" (performances).

RepTree traite les informations manquantes de la même manière que C4.5.

La procédure Reduced-Error Pruning
On traverse les noeuds internes de l'arbre du bas vers le haut et on vérifie pour chaque noeud interne si son remplacement par la classe majoritaire (pas forcément la plus majoritaire mais celle qui) ne réduit pas la justesse de l'arbre.
Pour estimer la justesse, un "ensemble d'élagage" est utilisé. Il a été démontré que cette procédure construit l'arbre le plus petit étant donné l'*"ensemble d'élagage"* utilisé.

Plus précisément, pour chaque sous arbre (non feuille) S de l'arbre T, on examine la modification de l'erreur de classification sur l'ensemble de test si S est remplacé par la meilleure feuille (= la classe parmi les classes majoritaires qui n'augmente pas l'erreur).
Si le nouvel arbre donne une erreur égale ou inférieure à l'erreur de S et si S ne contient pas de sous-arbre avec la même propriété, alors S est remplacé par la feuille. La procédure continue tant que ces élagages sont possibles sans augmenter l'erreur sur l'ensemble de test.
Dans la pratique, les noeuds sont élagués dans un unique parcours bottom-up et post-ordre de l'arbre de décision/régression.

1.4 Autres paramètres de RepTree

Extrait de la documentation Weka :

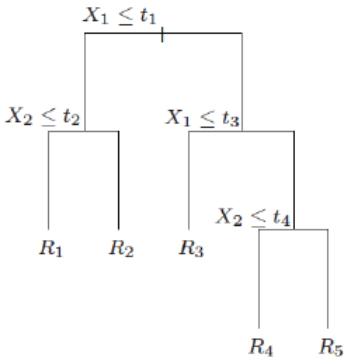
- **maxDepth** : It determines the maximum depth of your decision tree. By default, it is -1 which means the algorithm will automatically control the depth. But you can manually tweak this value to get the best results on your data
- **noPruning** : Pruning means to automatically cut back on a leaf node that does not contain much information. This keeps the decision tree simple and easy to interpret
- **numFolds** : The specified number of folds of data will be used for pruning the decision tree. The rest will be used for growing the rules
- **minNum** : Minimum number of instances per leaf. If not mentioned, the tree will keep splitting till all leaf nodes have only one class associated with it

1.5 Le principe d'élagage utilisé

Soit $\hat{p_k}(R)$ la fraction des instances descendues dans un noeud R.
Soit $|T|$ le nombre de feuilles de ce sous-arbre. La **mesure utilisée pour l'élagage** sera (on cherchera à minimiser cette valeur) :

$$C_\alpha(T_i) = \sum_{j=1}^{|T|}[1 - \hat{p_i}(R_j)] + \alpha\ |T|$$

Où α est un paramètre d'élagage ; un α grand produira un plus petit arbre (davantage élagué). Comme la méthode CART, REPTree choisit le meilleur α par une méthode 5-X-V ou 10-X-V.



REPTree (comme CART) cherche à produire, dans l'ensemble de k arbres $T = T_1, .., T_i, ...T_k$ construits itérativement, celui (T_i) qui **minimisera** $C_\alpha(T_i)$

2 Appliquer REPTree et Comprendre les résultats

2.1 Exemple de classification avec une classe discrète

Nous appliquons cette méthode à la BD "iris.arff" (fournie avec Weka dans le sous-répertoire *Data*, fourni également avec les data-sets du BE1). Elle contient 150 données sur la **classification** (donc de classe discrète) de 3 variétés d'Iris selon 4 attributs numériques différents.

Sous Weka, vous devez :
- Charger la BD. "iris.arff",
- Choisir la méthode dans *Classify / Trees / REPTree*,
- Conserver les paramètres par défaut et appliquer la méthode.

Exemple de résultats sous Weka (résultats complets sont donnés en section 2.3 ci-dessous) :

```
petallength < 2.5 : Iris-setosa (33/0) [17/0]
petallength >= 2.5
|   petalwidth < 1.75 : Iris-versicolor (36/3) [18/2]
|   petalwidth >= 1.75 : Iris-virginica (31/1) [15/0]
```

2.1.1 Interprétation des résultats de la classification

Dans le cas d'une classe discrète (comme ici) :
Prenons la 3e ligne | **petalwidth <1.75 : Iris-versicolor (36/3) [18/2]**

- A gauche du " : " on a les conditions appliquées par la règle (comme d'habitude !), puis
- Le premier terme est la valeur de la classe (ici la variété "Iris-versicolor", une valeur discrète)
- Entre les parenthèses, à gauche de "/", on a le nombre d'instances concernées par cette règle
- A droite de "/" : le nombre (total des poids) des instances mal classées.
  Si poids uniforme =1, ce nombre représente le nombre d'instances mal classées.

➛ Donc, avec **(36/3)** : sur les 36 instances concernées par cette règle, 3 seront mal classées.

- Entre les crochets : on a les statistiques calculées sur l'ensemble retenu (dit *hold-out*) utilisé pour l'élagage :
  Entre ces [], le premier nombre est le nombre d'instances bien classées dans l'ensemble d'élagage considéré.
  Ici, les 18 instances viennent s'ajouter au 36 (du **(36/3)**). Cette règle concerne donc $36 + 18 = 54$ instances.
  Le second nombre entre les crochets est le nombre d'instances mal classées dans ce même ensemble.

  Habituellement, le pourcentage d'erreur sur l'ensemble "pruning" est supérieure à celle de l'ensemble "growing".

  Les 4 nombres (des 2 côtés des '/) sont calculés avant que Weka utilise l'ensemble "pruning" pour améliorer les prédictions sur la totalité de la BD.

  Rappelons que Weka donne les résultats de l'évaluation du modèle (accuracy, etc..) après fusion des ensembles growing et pruning sur la totalité de la BD.

De ce fait :
si on considère la première ligne de ces résultats :

```
petallength < 2.5 : Iris-setosa (33/0) [17/0]
```

Manifestement, 17 instances ont été classées dans l'ensemble d'élagage avec la même classe `Iris-setosa` que les 33 instances. Le nombre (0) d'erreur pour les deux (après les deux '/') peut étonner.

On se demande pourquoi élaguer sans gagner en erreur ?

Il faut avoir à l'esprit que le but est d'obtenir un arbre qui satisfait le compromis taille-erreur.
De plus, ici, 17 instance n'avaient pas satisfait le test `petallength < 2.5` (par exemple, elles avaient leur `petallength` à une autre frontière que `2.5` mais inférieure à 2.5) mais qui respectaient le test et *in fine* étaient de la classe `Iris-setosa`.

Par exemple, ces 17 instances avaient `petallength < 2.45` avec la classe `Iris-setosa`. Weka les a tous regroupé, a réduit la taille de l'arbre et l'erreur n'a pas été modifiée [1]

Remarques sur les résultats : dans le cas discret, la somme de tous les nombres à gauche d'un "/" (entre parenthèses ou crochets) donne le total d'instances utilisées dans cette branche.
De même, le 2e nombre (celui après "/") donne le nombre d'instances mal classées. La somme de tous ces nombres donne l'erreur de classification de la méthode.

N.B. : pour faire des tests numériques et décider des branches, *RepTree* ordonne les instances selon les valeurs des attributs numériques.

2.2 Exemple de Régression avec une classe numérique

Dans le cas d'une classe numérique, REPTree trie les instances (la classe comprise) selon la valeur de la classe.
L'interprétation des résultats de REPTree est quelque peu différente dans le cas d'une classe numérique.

Il nous faut donc une BD avec une classe numérique. Appliquons RepTree à la BD "cpu-avec-vendeur.arff" où la classe est un réel : il s'agit des performances des ordinateurs selon leur configuration (RAM, Cache, ...).

1. Pour le voir, lancer cette même méthode mais demander à NE PAS élaguer ➛ on aura des [0/0] (voir section 2.3)

- Choisir la méthode dans *Classify / Trees / REPTree* à appliquer à la BD "cpu-avec-vendeur.arff".
- Conserver les paramètres par défaut et appliquer la méthode.

Exemple de résultats (partiels) sous Weka :

```
MMAX < 48000
|   MMAX < 22485
|   |   CACH < 20
|   |   |   MMAX < 10000
|   |   |   |   MMAX < 6100
|   |   |   |   |   MMAX < 3500 : 20.03 (21/5.14) [13/5]
|   |   |   |   |   MMAX >= 3500 : 28.43 (32/8.41) [12/23.49]
|   |   |   |   MMAX >= 6100 : 40.76 (19/16.14) [14/41.38]
|   |   |   MMAX >= 10000
|   |   |   |   vendor = adviser : 71.04 (0/0) [0/0]
|   |   |   |   vendor = amdahl : 71.04 (0/0) [0/0]
|   |   |   |   vendor = apollo : 71.04 (0/0) [0/0] <<=== et plein d'autres lignes "vendeur=..."
```

2.2.1 Interprétation des résultats de la régression

Dans le cas d'une classe numérique : soit la ligne `MMAX < 3500 : 20.03 (21/5.14) [13/5]`

Après le ' :'

- Le premier nombre est **la moyenne** des valeurs-de-classe de ce noeud (ici la classe "performances")
- Entre les parenthèses, à gauche de "/", on a le nombre d'instances concernées par cette règle
- A droite de "/" : **moyenne** de l'erreur moindre carrée de cette feuille relative aux seules instances de l'ensemble "growing" descendues dans cette feuille.
- Entre les crochets : le premier nombre est le nombre d'instances bien classées dans l'ensemble "growing" considéré et le second nombre est la **moyenne** de l'erreur moindre carrée relative aux instances de l'ensemble "pruning" (d'élagage) descendues dans cette feuille.

☞ **Dans le cas numérique**, le coefficient de **corrélation** (voir la fenêtre des résultats affichés par Weka) est un indicateur des performances du modèle obtenu (comme l'indice R^2 dans le cas d'une méthode de régression)

☞ L'élagage est un paramètre intrinsèque de cette méthode ; il est par défaut appliqué. Parfois, on obtient de meilleur résultats sans élagage mais l'arbre sera plus grand (cf. relancer la BD. IRIS sans élagage).

2.3 REPTree : effet des plis et élagage

REPTree est une méthode appréciée pour sa vitesse et la qualité du modèle produit. On voit ci-dessous les détails de cette méthode appliquée à la BD. *Iris.arff* (fourni avec Weka).

Dans une tâche de classification (arbre de décision), il convient de comparer les résultats de cette méthode aux méthodes telles que Id3, C4.5, SimpleCART, etc.

Rappelons que pour obtenir un arbre de régression (classe numérique), les méthodes Id3, C4.5, SimpleCART, etc. ne s'appliquent pas.

☞ Modifier *numFolds* (par défaut = 3) pour faire varier la taille (profondeur) de l'arbre (élagué).

Ce paramètre détermine la quantité de données utilisée pour l'élagage. Les données sont divisées en *numFolds* parties ; il y aura *un pli* (1 fold) utilisé pour l'élagage / test et le reste pour créer les arbres / règles.

Il est possible de dégager une relation (tendance) entre la *taille* de l'arbre et *numFolds* : plus numFolds est élevé, plus l'arbre devrait agrandir. Ce qui n'empêchera pas que pour certaines valeurs de *numFolds* (et pour un même BD.), la tendance soit contredite. Cela dépendra des tirages aléatoires, de la représentativité des données & autres paramètres.

• Pour *numFolds=3*, on a (paramètre "-N 3" de la méthode REPTree choisie) :

```
Scheme:       weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0

petallength < 2.5 : Iris-setosa (33/0) [17/0]
petallength >= 2.5
|   petalwidth < 1.75 : Iris-versicolor (36/3) [18/2]
|   petalwidth >= 1.75 : Iris-virginica (31/1) [15/0]

Correctly Classified Instances         141               94      %
Incorrectly Classified Instances         9                6      %
```

• Pour *numFolds=4*, on a (paramètre "-N 4") :

```
Scheme:       weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 4 -S 1 -L -1 -I 0.0

petallength < 2.45 : Iris-setosa (38/0) [12/0]
petallength >= 2.45
|   petallength < 4.75 : Iris-versicolor (34/0) [11/1]
|   petallength >= 4.75 : Iris-virginica (40/3) [15/3]

Correctly Classified Instances         141               94      %
Incorrectly Classified Instances         9                6      %
```

• Et avec *numFolds=5*, on a (paramètre "-N 5") :

```
Scheme:       weka.classifiers.trees.REPTree -M 2 -V 0.001 -N 5 -S 1 -L -1 -I 0.0

petallength < 2.45 : Iris-setosa (40/0) [10/0]
petallength >= 2.45
|   petallength < 4.95
|   |   petalwidth < 1.65 : Iris-versicolor (38/0) [9/0]
|   |   petalwidth >= 1.65 : Iris-virginica (5/1) [2/0]
|   petallength >= 4.95 : Iris-virginica (37/1) [9/1]

Correctly Classified Instances         141               94      %
Incorrectly Classified Instances         9                6      %
```

Pour cette même base de données ("iris.arff"), le taux d'erreur ne change pas mais on peut constater les différences dans les arbres.
Remarquez bien comment ces règles sont (quoique légèrement) différentes, et parfois un même attribut est testé contre des valeurs légèrement différentes.

Remarque : si *numFolds=2*, RepTree utilisera la moitié des données pour chaque arbre (Growing et Pruning). Dans ce cas et lors d'une CLASSIFICATION, il arrive que les feuilles du modèle final donnent des **verdicts identiques** mais dans deux feuilles apparemment **séparées**.
Par exemple, si on utilise *numFolds=2* avec la BD "vote.arff", on obtient (les autres paramètres inchangés) :

```
phisisian-fee-freese = n : democrate ....
phisisian-fee-freese = y :
       synfuel-corporation-cutback = n : republican (75.76/3.35)  [70.12/0.7]
       synfuel-corporation-cutback = y : republican (17.24/7.79)  [18.59/5.59]
```

On a le même conclusion dans les 2 dernières feuilles !
Pour ces 2 feuilles, on constate dans la 1ère que le nombre d'instances de Growing est supérieur à celui de Pruning et la classe "republican" en est déduite. Mais dans la dernière feuille, c'est l'inverse : c'est l'arbre Pruning qui impose son verdict avec un nombre d'instances de Pruning supérieur à celui de Growing.

Suite de l'exemple : ...

Et si on élague pas :

☞ Ne pas s'étonner que "sans élagage", des valeurs nulles entre '[]" (telles que [0/0]) soient pléthores !

• Si vous appliquez RepTree sans élagage à la BD *iris.arff*, vous aurez un début de règle :

```
petallength < 2.45 : Iris-setosa (50/0) [0/0]
```

D'où vient la valeur *2.45* **de** *petallength* **?**

Pour trouver une réponse, sauvegardez la BD au format ".csv" puis sous un tableur, trier les données selon la colonne *petallength* ET PUIS la colonne "classe" (2 critères de tri). Vous aurez :

No instance	sepallength	sepalwidth	petallength	petalwidth	class
48	...				
49	4,8	3,4	1,9	0,2	Iris-setosa
50	5,1	3,8	1,9	0,4	Iris-setosa
51	5,1	2,5	3	1,1	Iris-versicolor
52	4,9	2,4	3,3	1	Iris-versicolor
53	...				

On remarque qu'entre les lignes 50 et 51, la classe change à cette frontière. Il y a une partition (au sens discrétisation) : La valeur pour le test de *petallength* sera $(1,9+3)/2 = 2.45$ ➙ Il y a bien 50 instance où `petallength < 2.45`

Si on a utilisé un tableur comme indiqué, et si on trie selon la colonne *petalwidth*, la frontière $(1,8+1,7)/2 = 1,75$ entre les instances numéros (après tri) 104 et 105 s'établit avec une erreur pour l'instance numéro 105 (classe *Iris-versicolor*) parmi toutes celles de la classe *Iris-virginica*.

☞ On a constaté plus haut qu'avec élagage et *"-N 3"*, ce test devient : `petallength < 2.5 : Iris-setosa (33/0) [17/0]`
➙ Pourquoi `2.5` ? La réponse est dans les plis (folds) utilisés pour l'élagage. Les ensembles ayant servis à la construction et à l'élagage n'étant pas les mêmes, la frontière de la partition à cet endroit vaut 2.5 !

☞ Rappel : REPTree réordonne (une seule fois par attribut – selon les docs.–) les attributs numériques de la BD.

3 Travail à rendre (1)

Penser à demander à Weka toutes les mesures d'évaluation (cf. BE1, demander à l 'enseignant !).

☞ Rappelons que "sans élagage", des valeurs nulles entre '[]" (telles que [0/0]) seront légions !

Pour ce travail, on utilisera les BDs suivantes :

- La BD. "Cars.arff" (déjà fournie en BE1) avec 1728 observations et 7 attributs dont la classe *verdict*. Voir section 13 page 29 pour une description de la BD.
- La BD. "vote.arff" (fournie dans "data" de Weka"). Cette BD contient les votes pour le congréé aux USA à partir de 17 caractéristiques dont la dernière est la classe : le vote *"democrat"* ou *"republican"* (voir les détails sur le site *https ://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records*)
  Pour un commentaire des résultats de Weka, voir la section 3.1 en page 10.

1. Modifier les paramètres et choisir "sans élagage" (d'arbres) puis appliquer la même méthode à ces BDs et expliquer les résultats.
2. Appliquer la même méthode à la BD *wine* (rouge ou blanc). Visualiser la fenêtre "classifier error" avec les axes "qualité" vs. "qualité prédite" et commenter.
   • Indications : on peut voir les erreurs sur les deux axes demandés.
   ➛ Si accord (total), on verrait (presque toutes) les instances sur une droite $y = x$.
   De plus, Ici, le coefficient de corrélation est faible.
   ➛ Les petites croix = les plus justes.
   On aimerait voir une droite y=x (avec des petites croix). Or, on remarque plutôt plusieurs paquetages qui donnent une idée pour une utilisation du culstering (voir BE3) : 5-6 classes de prédiction de ces valeurs.
   Même en normalisant / standardisant les données (voir BE1 et les filtres sur les données), il faut regarder chaque paquet de données (groupes visibles dans "visualize error") et vérifier si on a $y = x$!

3.1 A propos des résultats de RepTree sur la BD. "vote.arff"

Vous avez remarqué que dans les résultats sur la BD. "vote.arff", selon le paramètre "numFolds", on a des sorties contenant des valeurs réelles où l'on attend un nombre d'instances (un entier).
Appliquons **RepTree** à cette BD. avec les paramètres par défaut, c-à-d. avec numFolds=3 avec la méthode habituelle validation croisée 10-XV.
☞ Mettre "debug" à vrai pour avoir des détails et demandez également une sortie "plain Text" dans "More options" (cf. BE1),

Après l'exécution de la méthode, on obtient :

```
    physician-fee-freeze = n : democrat (169.08/3.17) [84.33/0.58]
    physician-fee-freeze = y : republican (120.92/12.08) [60.67/5.25]
```

Matrice de confusion (accuracy = 95.4 % avec un indice *kappa* élevé de 0.90) :

```
  a    b <-- classified as
  256 11 | a = democrat
  9   159 | b = republican
```

Concernant (169.08/3.17) [84.33/0.58] ci-dessus (pour les 2 ensembles 'growing' et 'pruning', voir le principe de *RepTree*), la somme des valeurs à gauche des '/' est 169.08+84.33= 253.41.
Rappelons cette somme représente le total des poids des instance de ce noeud (dont la classe = 'democrat').
Puisque les poids sont identiques et égaux (à 1) pour toutes les instances (aucune n'est plus informative / importante qu'une autre) [2], on devrait avoir un entier à la place de 169.08 ou 84.33.
En outre, les valeurs après les "/" donnent l'erreur moindre carrée des instances mal classées arrivées à ce noeud [3].

D'où viennent ces réels ?
• La base de données "vote.arff" contient des valeurs manquantes. Si vous éditez (sous un éditeur de texte) ce fichier, vous y verrez des informations sur les attributs dont des valeurs manquent (signalés avec " ?") [4]

```
  Attribute: #Missing Values:
  1: 0
  2: 0
  3: 12
  4: 48
  5: 11 <<---- l'attribut "physician-fee-freeze" a 11 valeurs manquantes
  6: 11
```

Notons que l'entête au début de la BD. (fichier vote.arff) présentent les attributs en commençant d'abord par la classe. C'est pourquoi "physician-fee-freeze" est le 5e dans cet ordre de présentation.

Et un peu plus bas (l'attribut "physician-fee-freeze" est bien cette fois à la colonne 4 de la BD.) :

```
% Class predictiveness and predictability:Pr(C|A=V) and Pr(A=V|C)
  Attribute 1:
  ....
  Attribute 4: (A = physician-fee-freeze)
  0.08; 0.05 (C=democrat; V=y)
  0.92; 0.50 (C=republican; V=y)
  0.99; 0.92 (C=democrat; V=n)
  0.01; 0.01 (C=republican; V=n)
  0.73; 0.03 (C=democrat; V=?) <<---- Important
  0.27; 0.01 (C=republican; V=?)
  Attribute 5:
  .....
```

On note qu'il manque 11 valeurs à l'attribut *physician-fee-freeze* (notées " ?") et on compte dans la BD. que parmi ces 11 cas, 8 sont associées à la classe 'democrat' et 3 associées à 'republican'.

De ce fait, la ligne :

2. On dit en général que cette valeur représente "nombre d'instances".
3. Additionnez les 4 valeurs à gauche d'une '/' pour retrouver les 435 instances de la BD.
4. Notons que ces informations sont calculables à partir de la BD.

`0.73; 0.03 (C=democrat; V=?) <<---- Important`

nous apprend que lorsque physician-fee-freeze= ?, la classe serait 'democrat' dans 73% des cas [5] ($\frac{8}{11} \simeq 0.73$).

Sans davantage rentrer dans les détails, on peut affirmer ceci (on reprend la sortie) :

`physician-fee-freeze = n : democrat (169.08/3.17) [84.33/0.58]`

◦ n ($n \leq 169$) instances sont descendues dans la branche *physician-fee-freeze=n* de l'arbre "growing".

◦ Proportionnellement à $\frac{11}{435}$ (11 valeurs ' ?' pour l'attribut sur le total de 435) et au prorata de la probabilité 0.73 ci-dessus, k autres instances (avec *physician-fee-freeze=' ?'*) pourraient descendre dans la même branche dont la classe est 'democrat' ; ce qui explique les parties décimales,

◦ La somme arithmétique $n+k = 169.08+84.33 = 253.41$ donne le nombre total probable de ces 2 groupes.

◦ La matrice de confusion prédit *in fine* que sur le total de 435 instances, 256 ont été affectées à cette même branche après avoir fait la moyenne des résultats dans les arbres obtenus pendant 10-XV. Naturellement, aucun des 10 arbres que Weka affiche (si vous avez activé "Debug") ne donne à lui seul ces résultats finaux, Weka calcule une moyenne des résultats individuels.

Pour avoir une idée des valeurs n, k, on supprime les 11 instances où *physician-fee-freeze=' ?'* puis on applique laRepTree avec les mêmes paramètres pour obtenir la sortie :

```
physician-fee-freeze = n : democrat (164/2) [83/0]
physician-fee-freeze = y : republican (118/10) [59/4]
```

Et la matrice de confusion (accuracy = 96.2 % avec un indice *kappa* élevé de 0.92):

```
   a    b <-- classified as
   245 14 | a = democrat
   2   163 | b = republican
```

On constate que les valeurs manquantes baissaient (de peu) les performances du modèle.

5. De plus, si la classe = 'democrat', il y a %3 de chance que physician-fee-freeze= ?.

4 Règles de Classification

Pour une base de données contenant des observations de la forme (x, y) où $x = x_1, x_2, ..., x_k$ est l'ensemble des variables *explicatives* et y la variable *cible* (la décision, la classe), une règle de classification est de la forme $LHS \implies RHS$ avec $LHS \subseteq x$ et $RHS = y$.

Par exemple, un médecin peut avoir en tête des règles lui permettant de déduire une maladie à partir de symptômes. Si x représente l'ensemble des symptômes et y une maladie :

$$toux \ \& \ mal\_de\_gorge \implies rhume.$$

☞ Même si la connaissance (avérée) d'une maladie permettrait de déduire des symptômes (à travers des règles dites d'*association*, voir ci-dessous), nous ne sommes intéressés ici que par les règles dont la partie droite est la *classe* seule.

Nous avons vu (cours, chapitre 4, Part 2) l'algorithme de construction des règles de classification (ou de couverture, *covering*) appelé **PRISM**. Une méthode similaire mais plus élaborée appelée **Ripper** (*Repeated Incremental Pruning to Produce Error Reduction*) a également été évoquée.

4.1 Ripper

Ripper est une méthode industrielle et performante pour obtenir des règles de classification. Elle s'appuie sur une méthode de génération de règles de classification par élagage pour la réduction incrémentale d'erreurs. Voir section 14.1 en page 32 pour les détails de la méthode Ripper.

4.2 Exercice (à consigner dans le rapport)

Pour vous entraîner, appliquer la méthode Ripper (*JRip* sous Weka) à la BD "météo" (weather-nominal.arff").
☞ Cette méthode ne s'applique pas si la BD contient des attributs numériques.

Consigner les résultats dans votre rapport en introduction à cette partie.

☞ Vous pouvez être étonnés que malgré la "justesse" apparente des règles, il y a encore 5 erreurs !
Nous avions déjà observé ce phénomène dans le BE1 où ID3/J48 donnaient des arbres sans erreur alors que leur matrice de confusion affichait des erreurs.
La raison : Weka récupère le "meilleur" arbre mais rend compte qu'il y a eu des "incidents" (erreurs) pendant la construction de ces arbres/règles.
Parfois, on arrive à destination sans anicroche sur la route (on a alors pas de raison d'être prudent), mais parfois si ! Pourtant, dans les deux cas, on est arrivé !

5 Travail à rendre (2)

Ce travail s'applique aux 2 BDs citées ci-dessous.
Si une méthode ne figure pas dans la liste des méthodes, installez le package concerné et redémarrer Weka.

Les méthodes utilisées seront : *Ripper* (ci-dessus), *Decision Table* (voir l'aide de WEKA sur la méthode), DTNB (voir ci-dessous), PRISM, etc.

Pour ce travail, on utilisera les mêmes (deux) BDs (l'une puis l'autre) :

1. La BD. "Cars.arff" (fournie avec ce sujet) avec 1728 observations et 7 attributs dont la classe *verdict*. Voir section 13 page 29 pour une description de la BD.
2. La BD. "vote.arff" (fournie dans "data" de Weka"). Cette BD contient les votes pour le congréé aux USA à partir de 17 caractéristiques dont la dernière est la classe : le vote *"democrat"* ou *"republican"* (voir les détails sur le site *https ://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records*)

On appliquera différentes méthodes d'extraction des règles de classification sur ces deux BDs (à tour de rôle). Les détails de ce travail sont donnés ci-dessous.

5.0.1 Détails travail à rendre

La méthode *PRISM* sous Weka est donnée dans l'onglet "classify / rules" puis PRISM.

De même, la méthode *Ripper* (voir section 14.1 page 32) sous Weka est appelée **JRip**.

1. Choisissez une des 2 BDs. Si vous n'avez pas d'idée, commencer par "cars.arff". La 2e BD subira le même traitement.

   ☞ Au besoin, demander "Display Rules" dans les options de la méthode pour visualiser les règles.
2. Appliquer *Ripper* à cette BD pour obtenir des règles de classification.
3. Si vous travaillez avec la BD "vote.arff", trouver et expliquer pourquoi on ne peut pas y appliquer PRISM. Comment faire si on veut absolument appliquer PRISM à cette BD [6].
4. Appliquer la méthode *Decision Table* pour obtenir des règles de classification (dans les paramètres de la méthode, activer "Display rules").

   Penser bien à demander l'affichage des règles dans les options de la commande. Une fois la méthode sélectionnée, cliquer dans "More Options" et demander "Output Predictions" sous forme "Plain text" pour obtenir plus d'informations.
5. Appliquer la méthode DTNB (*Decision Table Naive Bayes*, voir ci-dessous pour les détails) pour obtenir des règles de classification. Cette méthode est plus complexe et plus lente à donner ses résultats.

   Comme ci-dessus, penser à demander l'affichage des règles + "Output Predictions" sous forme "Plain text" pour obtenir plus d'informations.

   ➛ Lire ci-dessous à propos de DTNB.
6. Établir une table de comparaison de ces méthodes (cf. BE1, voir également à la fin de ce sujet).

   Cette table contiendra les mesures telles que *Correctly Classified Instances, Incorrectly Classified, TP Rate, Precision, F-Measure, ROC Area, Kappa Statistics, etc.*

   Idem pour les mesures *Coverage of cases et Mean rel. region size* et / ou différentes mesures d'erreur.

   ☞ Contentez-vous des mesures proposées par les sorties de Weka.
7. Par un clique droit sur la méthode appliquée dans le cadre "Result list", visualiser "Threshold Curve" (ROC). Il faudra reproduire et commenter ces courbes dans votre rapport. La courbe ROC est abordé en cours *Chap4- Part 3*. Vous aurez peut-être besoin d'en consulter les pages sur ce sujet :

Pour la courbe ROC, consulter le support du cours *5-cours-chap4-part-3-xxx.pdf* qui évoque les courbes ROC.
8. {Question optionnelle} : Quand la méthode (ses sorties) le permet, pouvez vous extraire quelques règles de classification (2 ou 3) répétées dans toutes ces méthodes ? N'oubliez pas de demander l'affichage des règles pour les méthodes (e.g. DTNB) qui ne les affichent pas d'ordinaire.
9. Désigner la meilleure méthode.
10. Consigner vos résultats puis appliquer les mêmes étapes à la 2e BD.
11. Comparez et désigner la "gagnante" pour cette BD.
12. Peut-on dire que la même méthode l'emporte, quelque soit la BD ?

6. Indice : cette méthode ne s'applique pas si la BD contient des attributs numériques ou a des **valeurs manquantes**.

5.1 A propos de la méthode DTNB

La méthode DTNB est complexe et lente proportionnellement au nombre d'attributs. Si vous voulez appliquer DTNB à une *grosse* BD (par exemple "supermarket.arff" fournie avec Weka), procéder à un tirage pour ne conserver que x% des données d'abord (2% de "supermarket" donne env. 100 observations).

Appliquer ensuite cette méthode au résultat. L'échantillonnage se fait dans l'onglet "Preprocessing / Filter/ Resample". Voir BE1 également.

DTNB construit des règles de classification en utilisant une méthode hybride de *table de décision* (DT) et *Bayes naïve* (NB).
A chaque étape, l'algorithme estime la valeur de la division de l'ensemble (S) des attributs en 2 parties : une partie pour DT (S_{dt}) et l'autre pour NB (S_{nb}).

La probabilité de la classe est estimée comme suit :

Soit S_{dt} l'ensemble d'attributs pour DT et S_{bn} pour Bayes.

La probabilité de la classe est calculée par

$$P(classe|S) = \frac{\alpha.P_{dt}(classe|S_{dt}).P_{nb}(classe|S_{nb})}{P(classe)}$$

où $P_{dt}(classe|S_{dt})$ et $P_{nb}(classe|S_{nb})$ sont les probabilités des classes respectives dans DT et NB.
La constante α est une constante de normalisation (cf. Bayes Naïve vue en cours) et $P(classe)$ est la probabilité *a priori* de la *classe* (sa proportion dans la BD).

Toutes les probabilités sont calculées après un **lissage de Laplace** (cf. cours et BE1).

6 Règles d'association

L'exposé des méthodes d'apprentissage des règles d'association en cours a souligné leur importance en particulier en commerce (de tous genres !).

6.1 Application d'une méthode d'obtention de règles d'association

Nous avons testé plusieurs méthodes de *classification*. Pour utiliser une méthode produisant des règles d'*association* (RA), procédez comme la section suivante. On rappelle que les règles d'association permettent de découvrir les liens entre différents attributs (pas seulement la classe finale). On veut par exemple découvrir les liens entre la température, l'humidité et le vent dans ce site.

Commencer par charger la BD "météo" (Golf/Play).
Choisissez l'onglet ***Associate*** (rangée en haut) ; la méthode ***Apriori -N 10 ....*** s'affiche dans la zone ***choose***.

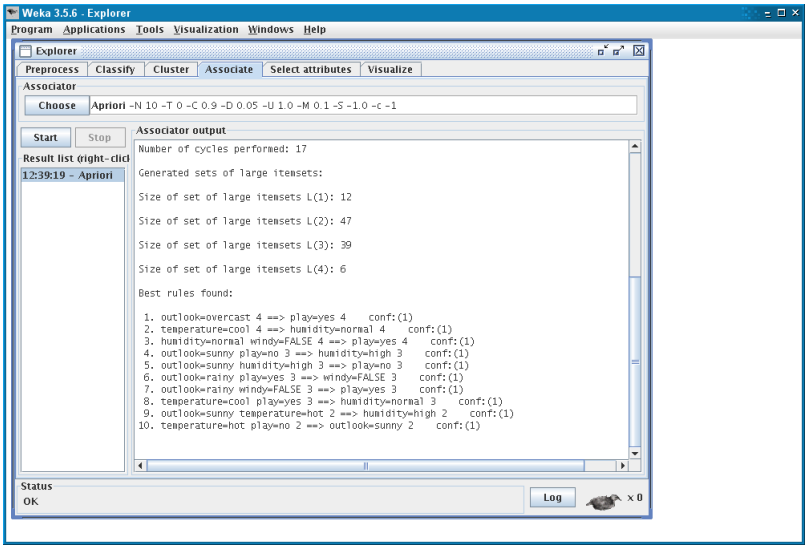
Paramètres de la méthode A Priori : Cliquez dans la zone *Apriori -N 10 ....* pour visualiser et modifier les paramètres de la méthode.

Dans la boîte de dialogue qui s'affiche, vous pouvez spécifier les seuils de **Confiance** (*MetricType Confidence* et *minMetric*) et de **Support** (*upperBoundMinSupport et lowerBoundMinSupport*).

Pour cette expérience, laissez le *MetricType* sur *Confidence*, le *upperBoundMinSupport* à 1.0, le *removeAllMissingCols* à *False*, le *significanceLevel* à -1.0 et le *delta* à 0.05.

Le paramètre *numRules* permet de fixer le nombre maximal de règles à extraire. Elles seront présentées par valeur de métrique décroissante (la confiance dans notre cas). Vous pouvez aussi demander à obtenir les itemsets fréquents.

Appliquez la méthode et étudiez les règles d'association qui s'affichent dans la fenêtre des résultats.



☞ Notez que l'extraction de règles d'association avec Weka ne peut pas être réalisée si un des attributs est de type *string*.

6.2 Règles de classification par la méthode A PRIORI

Il est possible, sous WEKA, d'obtenir (seulement) des règles de classification via la méthode A PRIORI. Pour cela, sous Weka, dans la méthode A PRIORI, choisir l'option "CAR" qui permet d'obtenir seulement des règles de classification (CAR : class associate rules).

Charger la BD. "vote.arff" (BD. fournie par Weka dans le répertoire "data"). Utiliser *A PRIORI* pour générer les règles d'association (RA) mais pensez bien cochez le paramètre "car" (*classification association rules*).

☞ Faire attention à ce que "metricType" soit bien "Confidence" (fig. ci-dessous à gauche). Vous devriez trouver une sortie de la forme de la fig. ci-dessous à droite.


```
Generated sets of large itemsets:

Size of set of large itemsets L(1): 6

Size of set of large itemsets L(2): 5

Size of set of large itemsets L(3): 1

Best rules found:

 1. adoption-of-the-budget-resolution=y physician-fee-freeze=n 219 ==> Class=democrat 219    conf:(1)
 2. adoption-of-the-budget-resolution=y physician-fee-freeze=n aid-to-nicaraguan-contras=y 198 ==> Class=democrat 198
 3. physician-fee-freeze=n aid-to-nicaraguan-contras=y 211 ==> Class=democrat 210    conf:(1)
 4. physician-fee-freeze=n education-spending=n 202 ==> Class=democrat 201    conf:(1)
 5. physician-fee-freeze=n 247 ==> Class=democrat 245    conf:(0.99)
 6. el-salvador-aid=n aid-to-nicaraguan-contras=y 204 ==> Class=democrat 197    conf:(0.97)
 7. el-salvador-aid=n 208 ==> Class=democrat 200    conf:(0.96)
 8. adoption-of-the-budget-resolution=y aid-to-nicaraguan-contras=y 215 ==> Class=democrat 203    conf:(0.94)
 9. education-spending=n 233 ==> Class=democrat 213    conf:(0.91)
10. adoption-of-the-budget-resolution=y 253 ==> Class=democrat 231    conf:(0.91)
```

← Remarquer à gauche que le nombre de règles demandées = 10. Augmenter (pour une confiance = 100%) si vous le souhaitez. ☞ Bien noter qu'ici, le seuil de fréquence est de 10% (0,1). ce qui est (un peu bas). Modifier cette valeur pour obtenir différents ensemble de règles.

☞ Bonus : un bonus est accordé aux compte-rendus qui intègrent et comparent ces règles (de classification issues des RA) aux règles de classification obtenues dans la section précédente.
Ne cherchez pas à les intégrer dans la table de comparaisons : il manque les statistiques ! Cherchez simplement à voir si ces règles sont trouvées par les autres méthodes.
Y a-t-il des règles en commun ? Les experts (qui exploitent vos modèles) "aiment bien" les règles qui reviennent dans différentes méthodes !

6.3 Obtention des itemsets fréquents

Sous Weka, on peut obtenir les itemsets fréquents (suivant le seuil de fréquence). Dans les paramètres de la méthode *A priori*, vous pouvez cocher "outputItemSets" pour obtenir les itemsets fréquents.

Notez bien la manière de Weka d'afficher ces itemsets : Weka affiche les fréquents en excluant la classe. Chaque itemset est donné suivi de son nombre d'occurrence. Sous chaque itemset fréquent, une information concernant la classe. Par exemple, pour la BD. "météo", si on a :

```
windy=FALSE 8
0 6                ← ce couple est de la forme :      classe   nb_occurrences
```

On peut alors comprendre que par 6 fois, la classe 0 (ici *Play=Yes*) est présente quand *Windy=False*. Le 0 qui précède est donc le rang de la classe *Yes* (sans autre signification).
Évidemment, plus le seuil de fréquence est faible, plus on a d'itemsets fréquents.

On peut récupérer ces itemsets pour d'autres calculs (voir le BE suivant).

6.4 Autre méthode de règles d'association

Il existe plusieurs méthodes pour obtenir des règles d'association sous Weka.

Par exemple : dans l'onglet "associate", on a la méthode "FilteredAssoc" qui prend en paramètre la méthode "A PRIORI" et donne des résultats intéressants.

☞ Comme A PRIORI, la méthode "FilteredAssoc" avec "A priori" ne s'applique qu'aux données catégorielles.

☞ Rappel : Weka ne permet pas d'obtenir des règles d'association **avec des données numériques** (ou mixtes). Pour cela, on peut discrétiser les données (on peut l'appliquer sur tous les attributs, la discrétisation ne touche pas aux non numériques).

7 Travail à rendre (3)

1- Traiter totalement la classification de l'exemple des **champignons** ("mushrooms.arff", voir répertoire de ce BE, voir aussi la section 9.1, voir aussi https ://archive.ics.uci.edu/ml/datasets/mushroom pour une description) avec WEKA, en y appliquant la méthode qui vous semble la mieux adaptée et qui réduit l'erreur.

Appliquez aussi la méthode **Apriori** à cette même BD et discutez simplement des règles obtenues.

2- Appliquer la méthode **Apriori** à l'exemple **Zoo** et présenter les résultats.

3- Obtenez des règles d'association de l'exemple de credits **Credit-g** et présenter les résultats.
☞ Sur cette BD, une discrétisation peut être nécessaire.

8 Travail en bonus (1)

1- Traiter totalement la classification de l'exemple des **champignons** ("mushrooms.arff", voir répertoire de ce BE, voir aussi la section 9.1, voir aussi https ://archive.ics.uci.edu/ml/datasets/mushroom pour une description) avec WEKA, en y appliquant la méthode qui vous semble la mieux adaptée et qui réduit l'erreur.

Appliquez aussi la méthode **Apriori** à cette même BD et discutez simplement des règles obtenues.

9 Travail optionnel (2), non trivial : poste traitement des itemsets et règles

☞ Cette section est optionnelle. Mais elle vous montre comment dans la pratique procède-t-on manuellement à certaines amélioration des règles d'association obtenues.

Lorsque le nombre d'itemsets est élevé donnant lieu à un grand nombre de règles, on procède à un post-traitement des itemsets pour en diminuer le nombre et conserver les plus intéressants / compactes. Ce post-traitement définit et utilise souvent une *distance* entre les itemsets.

9.0.1 Distance entre des itemsets

Il s'agit d'une autre méthode de recherche des itemsets intéressants.
La notion de distance permet de regrouper les itemsets *similaires*[7]

On prendra la moyenne des cardinaux normalisés de la différence symétrique sur l'ensemble des items et de celle sur l'ensemble des itemsets. Par exemple, si l'on considère deux itemsets X et Y , alors :

$$d(X,Y) = \frac{1}{2}\left(\frac{|X \Delta Y|}{|X \cup Y|} + \frac{|t_X \Delta t_Y|}{|t_X \cup t_Y|}\right)$$

où Δ est l'opérateur de différence symétrique et où t_X (resp. t_Y) dénote l'ensemble des tuples contenant X (resp. Y) dans la BD d'origine (pas seulement dans les *fréquents*). $|A|$ = taille de l'ensemble A.

Différence symétrique : pour deux ensembles A et B, la différence symétrique (Δ) est :

$$A \Delta B = (A \cap B^c) \cup (B \cap A^c) = (A - B) \cup (B - A) = (A \cup B) - (A \cap B)$$

☞ Voir aussi ci-dessous pour d'autres expression de distance (moins complexes !).

Remarque : l'expérience montre que la seule première fraction de la distance $d(X,Y)$ ci-dessus donne de bons résultats. Bien que les distances ainsi calculées soient moins "bonnes", le coût de cette solution (beaucoup moins de calculs) permet de l'envisager.

Rappelez-vous : en matière de l'EC, les meilleurs résultats (ici les règles) s'obtiennent par étapes et par tâtonnement. Si un expert (vous) en champignons devait rejeter, pour cause de manque d'intérêt, les règles obtenues avec une distance simplifiée, on reprendrait la fonctions complète de distance $d(X,Y)$!

Avec la BD. *mushrooms* (voir ci-après), si vous optez pour la solution de sélection des itemsets fréquents les plus proches à l'aide de la distance $d(X,Y)$, vous pouvez considérer la distance entre **deux itemsets dans lesquels la classe finale (valeurs 118 ou 119 pour les champignons) apparaît**. En effet, nous voulons surtout savoir si un champignon est comestible ou pas.

Cette contrainte permet de fixer un seuil de fréquence dans l'intervalle $[0.3, 0.5]$ (0.3 est encore acceptable) et d'obtenir des itemsets parmi lesquels la classe finale figure environ 50 fois.

- Un *modus operandi* possible :
  1. Tester avec les seuils 0.3, 0.35, 0.4, 0.45 et 0.5 puis choisissez ;
  2. Enlever les itemsets sans l'une des classes finales (118 ou 119) ;
     ➞ Pour cela exécuter la commande Linux `grep " 11[89]" fichier_regles > fichier_sortie_118_119`
  3. Calculer la distance entre les itemsets deux à deux. Conserver les distances dans un tableau (matrice diagonale) ;
  4. Faire des paquets d'itemsets "proches". Vous pouvez appliquer *k-means* (voir l'algorithme ci-dessous). K-means consulte sans cesse les distances. Le calcul de ces paquets "proches" peut aussi se faire intuitivement. Dans les 2 cas, il faut avoir une idée du nombre de paquets de règles.
  5. Choisissez un représentant par paquet. Vous pouvez décider d'avoir plus d'un itemset par paquet de règles. Le nombre d'itemsets finaux (et donc de règles) dépend du nombre de paquets et du nombre de représentants par paquet.

7. Voir avec l'enseignant pour un éventuel code pour calculer la distance.

6. Générer les règles de classification correspondantes.

Pourquoi créer les règles vous-même ? : sachant que l'on veut les règles qui concluent sur la classe finale, vous pouvez vous-même générer les règles : la tâche est grandement simplifiée car un itemset retenu contenant la classe finale donne lieu à au plus une seule règle de classification !

☞ Notons que vous pouvez appliquer la recherche des Clos / Maxi aux résultats de l'application de la distance pour réduire encore plus le nombre de règles.

9.0.2 D'autres mesures de distance

Il existe de multiples expressions de distance (ou symétriquement **similarité**) entre deux itemsets.
On peut citer :

1. Une mesure de similarité (ou *affinité*) pour deux itemsets X et Y est :

$$sim(X,Y) = \frac{Supp(\{X,Y\})}{Supp(X) + Supp(Y) - Supp(\{X,Y\})}$$

où $Supp(X)$ est la fréquence de X dans la BD et $Supp(\{X,Y\})$ est la fréquence des transactions contenant à la fois X et Y dans la BD.

➙ Obtenir la $distance = 1 - affinite$.

2. Une mesure de distance est la **divergence** (distance) de Kullback-Leibler :

$$KL(X||Y) = Supp(X)\ log\left(\frac{Supp(X)}{Supp(Y)}\right) + Supp(Y)\ log\left(\frac{Supp(Y)}{Supp(X)}\right)$$

où $Supp(X)$ est la fréquence de X dans la BD

9.0.3 Regroupement par clustering des itemsets

• Les algorithmes de clustering (on utilisera *K-means*) permettent de regrouper des objets "proches/similaires",

➥ Les points les plus proches d'un espace à n dimensions où un cluster = une "hypersphère" (n-sphère) de dimension n

• Le clustering est un problème n-p complet, donc "difficile" de trouver une solution exacte.

➥ on peut trouver une solution approchée avec un optimum local

On utilisera K-means en BE3. Elle est cependant très simple (dans sa version de base).

Principe de l'algorithme K-means :

• Données en entrée + fonction de similarité / distance.

But : - minimiser la distance intra-classe (somme distance centre-points)
- maximiser distance inter-classe

• Étapes : On commence en fixant un nombre de clusters (par exemple : 5) et on place aléatoirement les 5 barycentres (= centres de cluster)

1) on détermine pour chaque point P de l'espace le barycentre le plus proche (parmi les 5 actuels) ; le cluster de P sera celui du barycentre le plus proche
2) et pour chaque cluster, recalculer le barycentre des points regroupés ensemble. Le nouveau barycentre peut être différent de celui qui avait fédéré le groupe ;
3) on réitère en considérant les barycentres ainsi modifiés et on refait les groupes ;
4) on s'arrête quand les clusters ne changent plus (point fixe) ou très peu ($< \varepsilon$) ;
ou au bout de 10000 itérations
ou au bout de "somme des distances inférieure à un seuil fixé".

9.0.4 Utilisation des règles de classification pour la prédiction

Réfléchissez à une procédure permettant d'identifier les règles caractérisant **au mieux** le fait qu'un champignon (BD. "Mushrooms.arff", voir la section 9.1) soit comestible ou vénéneux (e.g. notion d'inclusion / couverture des itemsets). Vous serez attentif au fait que chaque champignon vérifie au moins une règle (ne

pas exclure un champignon=danger !).

Bonus : coder votre solution. Une fois votre algorithme mis au point, vous pourrez tester son efficacité en l'appliquant aux données d'entrée et vous pourrez ainsi vérifier sa précision.

Validation croisée : Afin d'éviter certains problèmes d'*overfitting*, une technique classique consiste à diviser le jeu de données initial (`mushrooms.data`) en deux jeux :
- l'un composé d'une certaine proportion (par exemple 90 %) du jeu de données d'entrée qui sert à l'apprentissage.
- l'autre composé du reste des tuples et qui sert à évaluer votre classifieur (ensemble de test).

Fin de la section optionnelle.

9.1 Règles association et BD Champignons

• Voir https ://archive.ics.uci.edu/ml/datasets/mushroom pour une description e ce Dataset.

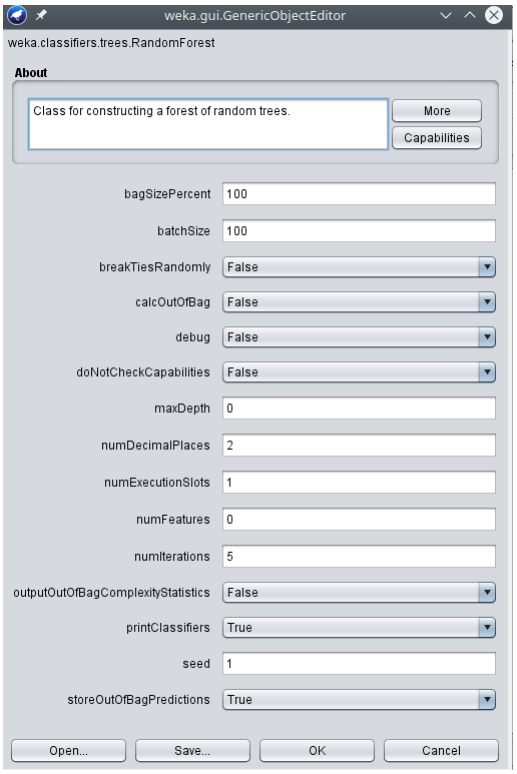
A titre indicatif : sur les 8124 données, les règles idéales (liste de décision = ordre d'application de la première à la dernière) que l'on peut obtenir par d'autres méthodes plus complexes sont données ci-dessous.
☞ Les nombres entre parenthèses donnent le nombre de cas concernés = support.

Appliquées dans l'ordre, la confiance de l'ensemble est de 100%. On notera que le support descend à 8 (sur 8124) pour couvrir tous les cas.

— (odor = f) ⇒ class=p (2160.0)
— (gill-size = n) and (gill-color = b) ⇒ class=p (1152.0)
— (gill-size = n) and (odor = p) ⇒ class=p (256.0)
— (odor = c) ⇒ class=p (192.0)
— (spore-print-color = r) ⇒ class=p (72.0)
— (stalk-surface-above-ring = k) and (gill-spacing = c) ⇒ class=p (68.0)
— (habitat = l) and (cap-color = w) ⇒ class=p (8.0)
— (stalk-color-above-ring = y) ⇒ class=p (8.0)
— True ⇒ class=e (4208.0) règle par défaut si aucune autre ne s'applique.

10 Travail en Bonus (3)

10.1 Random Forests



Appliquer la méthode *Random Forest* (sous Weka : classify / Trees / RandomForest) à la BD "météo".
➛ Il faudra peut-être installer les packages "ensembleLibrary" et "rotationForest" pour voir cette méthode dans la liste des méthodes de la rubrique "Trees".
N'hésitez pas à l'appliquer aux mêmes bases de données.

A l'aide de la figure ci-contre, fixer les paramètres. En particulier, pensez bien à réduire le nombre d'arbres de décision obtenu à 5 ou 10 (au lieu de 100 par défaut) et demandez ces arbres en sortie Weka.

Cette méthode donne souvent des résultats intéressants. Malheureusement, il est délicat d'obtenir un ensemble de règles puisque les arbres retenus donnent un résultat collégial.

☞ Chercher à savoir ce qu'est cette méthode et donner une présentation de 5 lignes dans votre rapport.
☞ Consigner les résultats pour une comparaison.

☞ S'applique également aux données numériques.

10.2 Régression Logistique Ridge

Appliquer la méta-méthode *Régression Logistique d'arête / de crête* (on dit aussi : régression avec pénalité). Sous Weka, choisir : *classify / functions / Logistic* et l'appliquer aux mêmes bases de données en conservant les valeurs par défaut des paramètres.

☞ Chercher à savoir ce qu'est cette méthode et donner une présentation de 5 lignes dans votre rapport. Consigner les résultats pour un comparaison.

• Établir une table de comparaison pour ces deux méthode puis désigner la "gagnant".

☞ Remarques : dans le rapport à remettre, donner quelques lignes (5 par exemple) d'explication / présentation sur les méthodes employées, en particulier, *Random Forest* et *Logistic*. Vous apprendrez à vous informer des avis et expériences de la communauté Data Mining (/ Weka).

11 Travail à rendre (4)

Il s'agit ici d'appliquer les méthodes "Régression Logistique" et "SVM"

11.1 Régression Logistique

• Charger la BD Spam "spambase_binary.arff". Cette BD. est issue de la binarisation de la BD "spambase.arff" (voir section **??**, page **??**).

Pour nous éviter un travail long sous Weka (lorsque Weka travaille sur un grand ensemble de données), nous allons d'abord sous-échantillonner les données pour réduire la taille de l'ensemble d'apprentissage.

Nous devons d'abord réorganiser les données dans un ordre aléatoire. Cette étape est importante si nous voulons un nombre proportionnellement égal de données de chaque classe, d'autant que par défaut, les instances positives sont placées en premier dans la BD.

• Dans l'onglet *Pre-Processing*, section "Filter", appuyer sur "Choose / filtres / unsupervised / instance / Randomize".
Notez le **seed** (la graine) aléatoire utilisée (qui doit être 42) pour randomiser l'échantillonnage des données et cliquez sur le bouton "Apply" tout à droit en face de "Choose".
➛ Astuce : Une fois que vous avez sélectionné le filtre "Randomize", vous pouvez cliquer sur la ligne de commande "Randomize -S 42" (à côté de "Choose") pour afficher le menu des options de ce filtre.

Nous allons maintenant sélectionner un sous-ensemble de données à l'aide du filtre "RemoveFolds".

Nous voulons garder 10% du jeu de données.
• Après avoir sélectionné le filtre "RemoveFolds" (sous "Randomize"), cliquez sur la ligne de commande à côté du bouton "Choose" pour afficher et modifier ses paramètres.
En fait, les options par défaut de création de 10 plis (folds) et la sélection du premier pli (option "fold" : 1) nous conviennent.

• Cliquez sur OK, puis sur "Apply". Vous verrez sous "Choose" quelques informations (nombre d'attributs, nombre d'instances, ...). Enregistrez maintenant cet ensemble de données sous le nom *spambase_binary_fold1.arff*. Il y a 461 instances.

On utilisera les données restantes comme ensemble de test (et validation).

• **Recharger** l'ensemble de données binarisé complet *spambase_binary.arff* et randomiser les à nouveau (en utilisant la même graine).
• Sélectionnez à nouveau le filtre "RemoveFolds" mais cette fois, dans les option de ce filtre (en cliquant sur la zone de commande "RemoveFolds -S O -N 10 -F 1"), placez "invertSelection" sur "True" dans le menu des options du filtre, puis cliquez sur "Apply".
Cela produit 4140 instances.

• Enregistrez cet ensemble de données sous le nom "spambase_binary_fold2-10.arff".

On va maintenant construire nos classifieurs par les méthodes **Régression Logistique** et **SVM**.
Pour ce faire, il faudra classer les e-mails comme spam ou non-spam puis évaluer les performances des modèles obtenus.

• **Recharger** le jeu de données "spambase_binary_fold1.arff". On utilisera cette BD das la suite.
• Allez dans l'onglet "Classify" et sélectionnez "Choose / functions / SimpleLogistic".
• Dans le cadre "test Options", sélectionnez "Supplied test set", cliquez sur le bouton "Set" puis "Open File" et chargez "spambase_binary_fold2-10.arff".

• Cliquez sur "Start" pour entraîner le modèle. Examiner le cadre à droite (la sortie du classifieur) pour afficher les informations du modèle.

Répondre aux questions suivantes :
(I) Quel est le pourcentage d'instances correctement classées (Accuracy) ?
(II) Comment les coefficients de régression de la classe 1 se rapportent-ils à ceux de la classe 0 ? Pouvez-vous déduire cela de la forme du modèle de Régression Logistique ?
➛ N.B. : remarquez les symétries dans les valeurs des 2 classes et pensez à la courbe de la Régression Logistique !

➛ Visiter les courbes proposées (clique droit sur la méthode puis "visualize ..."). En particulier, "thresold curve" vous renseigne sur la courbe ROC.

(III) Notez les coefficients pour la classe 1 pour les attributs *[word_freq_hp_binarized]* et [char_freq_$_binarise].

Généralement, on s'attendrait à ce qu'un mail contenant "$" apparaisse dans les spams. De même, la chaîne "hp" devrait être un "marqueur" des e-mails non-spam (aka ham), car les données ont été collectées auprès des labos de HP.
(IV) Est-ce que les coefficients de régression ont-ils un sens étant donné que **la classe 1 correspond au spam** ?

➛ Indice : considérez la fonction sigmoïde (rappel du cours : $Pr(...) = \frac{1}{1+e^{-x}}$) et comment elle transforme les valeurs en une probabilité entre 0 et 1.
Puisque nos attributs sont booléens, un coefficient positif ne peut qu'augmenter la somme totale alimentée par le sigmoïde et donc déplacer la sortie du sigmoïde vers 1.

(V) Que peut-il se passer si nous avons des attributs (var prédictives) à valeurs réelles ?

11.2 SVM

Nous allons maintenant entraîner un modèle SVM.
• Dans l'onglet "Classify", sélectionnez "Choose / functions / SMO" (SMO = *Sequential Minimal Optimization* qui est un algorithme utilisé pour les modèles SVM).
• Utilisez les paramètres par défaut et cliquez sur "Start". Cela construit un modèle SVM (assez similaire à la Régression Logistique).

Examinez la sortie du classifieur et **essayez de répondre aux questions suivantes** :
(I) Quel est le pourcentage d'instances correctement classées ? Comment se compare-t-il au résultat de Régression logistique ?
(II) Quels sont les coefficients pour les attributs [word_freq_hp_binarized] et [char_freq_$_binarisé] ? Comparez-les à ceux que vous avez trouvé avec la Régression Logistique ci-dessus.
(III) Quel est le lien entre une SVM linéaire et la Régression logistique ?

➛Conseil : tenez compte de la frontière de décision (hyperplan) de classification appris dans chaque modèle.

☞ **N.B.** : nous avons vu le SVM avec noyau au cours 6. La méthode SMO utilisée emploie le noyau "PolyKernel" par défaut.

11.3 Aller plus loin (IBK, KNN, feature selection)

Nous allons maintenant évaluer les performances d'un classifieur IBK/KNN.
Pour ce faire, nous allons introduire un nouveau jeu de données (*Splice*, données génétiques, BD. fournie).
Pour la classification, nous allons identifier les hyperplans de séparation des *introns* et des *exons* sur les séquences de gènes.

➛ Le fichier "splice.names" vous donnera plus d'explication sur les attributs.

Dans cette BD., l'attribut "class" (décision) peut prendre 3 valeurs : **N, IE , EI**.

• Charger les ensembles de données :
splice_train.arff : données d'entraînement et *splice_test.arff* : données de test

Nous utiliserons également un nouveau classifieur.

• Sous l'onglet "Classify", sélectionnez "lazy / IBk".
Il s'agit d'un classifieur "K-plus-proches-voisins" (KNN).
• Dans le panneau "Test Options", sélectionnez "Use training Set" et appuyez sur "Start".
• Observez la sortie du classifieur et **répondez aux questions suivantes** :
(I) Quelle est la précision de la classification ? Est-ce significatif ?
(II) Pourquoi tester sur les données d'apprentissage est-il une idée **particulièrement mauvaise** pour un classifieur 1-NN (1-plus-proche-voisin) ?
(III) Vous attendiez-vous à ce que les performances du classifieur sur un ensemble de test soient aussi bonnes ?

Évaluez maintenant le classifieur sur l'ensemble de test. Pour ce faire,
• Dans le panneau "Test option", sélectionnez "Supplied Test set" et chargez le fichier "splice_test.arff".
• Dans le panneau de la liste des résultats, cliquez avec le bouton droit sur le classifieur et sélectionnez "Re-evaluate the model" sur l'ensemble de test actuel.

• Observez la sortie et **répondez aux questions suivantes** :
(IV) Quelle serait la précision du classifieur si tous les points étaient étiquetés N ?
➛ Conseil : Affichez la distribution de l'attribut de classe des données de test. Pour ce faire, chargez les données de test dans l'onglet "Pre-Processing" et sélectionnez l'attribut de classe dans le panneau "Attributs".

• Explorez maintenant l'effet du paramètre k. Pour ce faire, entraînez le classifieur plusieurs fois, en définissant à chaque fois l'option KNN sur une valeur différente. Essayez 5, 10, 100, 1000 et 10000 et testez le classifieur sur l'ensemble de test.
➛ Astuce : pour modifier l'option KNN, vous devez afficher le panneau d'options du classifieur.

Questions :
(V) Comment le paramètre k affecte-t-il les résultats ?
➛ Astuce : Considérez à quel point le classifieur généralise des données inédites (inconnues) et comment il se compare à nouveau au taux de base.
Tracez (plot) les résultats (valeur k sur l'axe des x et PC sur l'axe des y).
(VI) Pouvez-vous déduire quelque chose en observant ce plot ?

12 (Optionnel) L'outil *Experimenter* en Extraction de Connaissances

Cet outil permet de comparer des modèles obtenus sous Weka. Toutes les méthodes disponibles en Weka ne sont pas concernées.

- BDs utilisés pour les comparaisons :
  - Bank-Data
  - Spam

En plus des outils d'Extraction de connaissances déjà manipulés, Weka propose deux autres outils intéressants :
- un outil d'analyse des résultats (*experimenter*) qui propose *t-statistique* (voir chapitre 5 du cours),
- un autre pour automatiser les étapes d'une expérience (*KnowledgeFlow*).

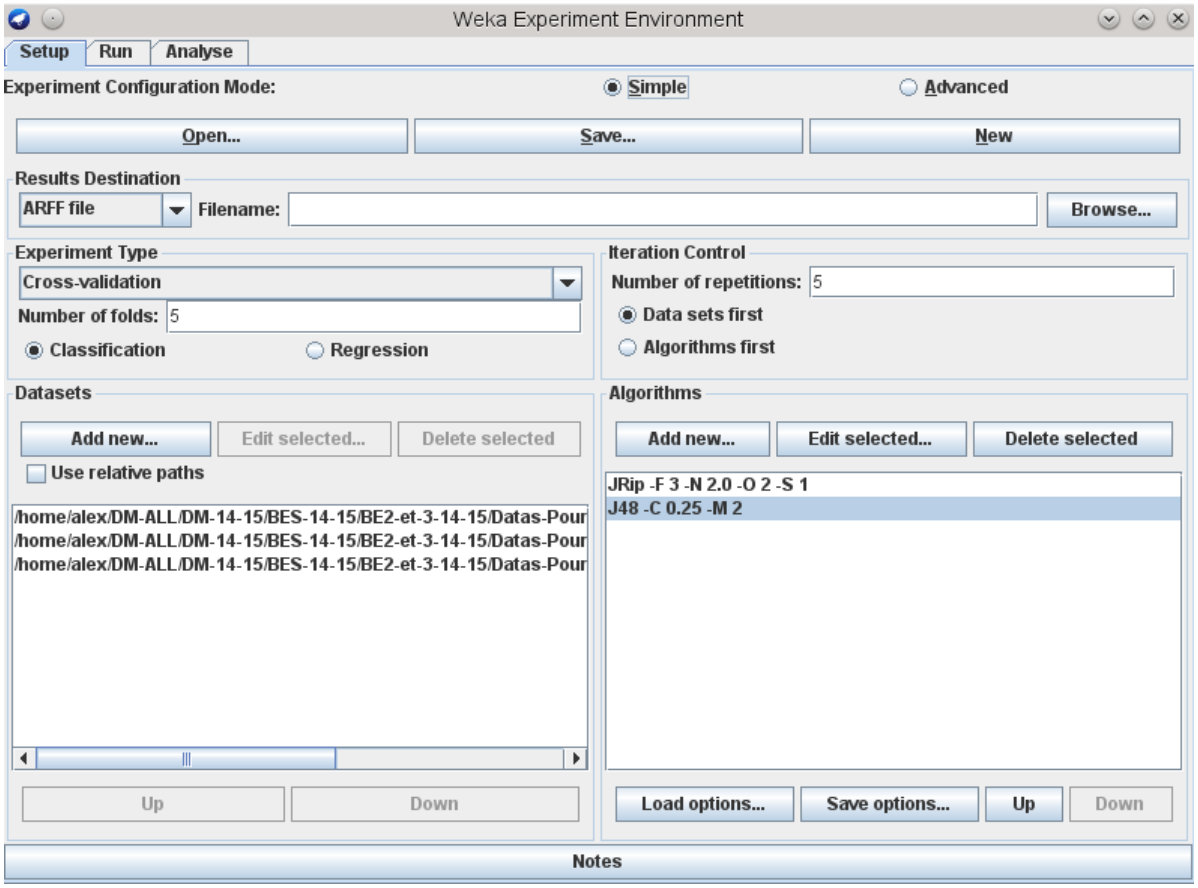
Cet outil permet de comparer plusieurs méthodes de classification [8] (deux ou plus) appliquées à une Base d'apprentissage (préférez deux ou plus BDs différentes pour de meilleurs résultats).
Les intéressés pourront avoir plus d'information sur ces outils en consultant les fichiers *.pdf* fournis dans le répertoire d'installation de Weka.

12.1 Un exemple

Supposons disposer de trois bases de données de *spam* (fournis) nommées *spambase_part1.csv*, *spambase_part2.csv* et *spambase_part3.csv* obtenues en scindant la BD originale des spams en 3 parties stratifiées (mêmes rapport $\frac{Spam}{NonSpam}$ dans les 3).

On décide de comparer les méthodes d'Arbre de Décision **J48** (*trees/J48* = version Weka de C4.5) et la méthode de classification et d'extraction de règles **JRip** (méthode Ripper implantée dans Weka par *rules/-JRip*) qui est une amélioration de la méthode de classification par règles vue au chapitre 4. [9].

Pour utiliser l'expérimentateur de Weka, on choisit *Experimenter* dans le menu *Application* de Weka. La fenêtre ci-dessous montre quelques détails (détails ci-dessous)



8. Seulement en Classification pour la version actuelle de Weka
9. Ripper : *repeated incremental pruning to produce error reduction* : une méthode d'apprentissage de règles de classification optimisée qui procède ensuite à un élagage des conditions de règles pour réduire les erreurs. Voir section 14.1

On choisit l'onglet *Setup* puis :
- **New** et on précise un fichier *.arff* qui contiendra nos résultats (Ex. : *spam-results.arff* dans le répertoire des données de spam) ;
- **Experiment Type** = Cross-Validation et **Number of folds** (plis) = 5
- **Data Sets : Add New** et donner les 3 fichiers .csv fournis (sélectionner les 3 en même temps)
- A droite : **Number of Repetitions** = 5, et **Data Sets first**
- **Algorithms** : choisissez JRip (*rules/JRip*) et J48 (*trees/J48*) avec leurs paramètres par défaut ;

Vous pouvez modifier les divers paramètres de ces méthodes (sélectionnez puis *Edit Selected*).
La figure précédente vous permet de vérifier vos choix.
On peut sauvegarder (par *Save*) la configuration de l'expérience que l'on vient de choisir, ou ouvrir (par *Open*) une précédente configuration.

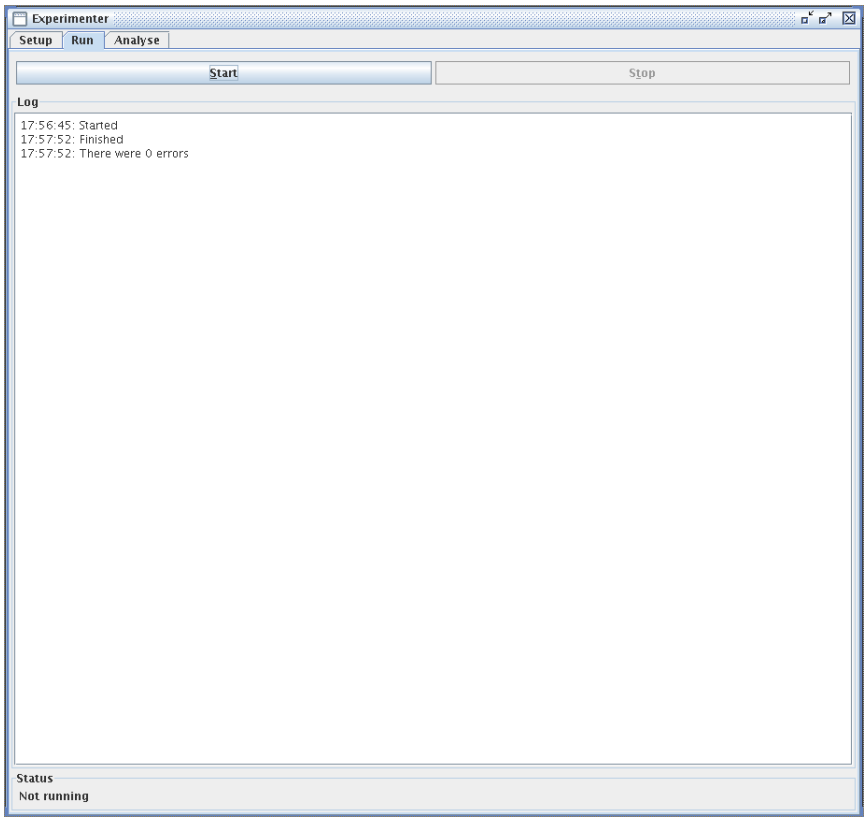
Passez au 2e onglet dans cette même fenêtre, en haut à gauche et choisir **Run** pour lancer les apprentissages (par les méthodes choisies)et démarrer l'exécution de l'expérience en appuyant sur *Start*.

Cette fenêtre pourrait contenir des résidus des expériences précédentes (pas le cas ici).

☞ IMPORTANT : Il est préférable que les méthodes d'apprentissage choisies soient **préalablement testées** (dans *Explorer* de Weka, comme vous l'avez fait dans les BEs précédents) pour éviter les erreurs telles que données inadaptées à la méthode, erreur dans les BDs, etc. dans *Experimenter*. Il serait frustrant de lancer une série de méthodes d'apprentissage dont la dernière pourrait mener à une erreur et invalider ainsi l'ensemble de l'expérience (perte du temps).

☞ Surveiller les fichiers *csv* avec un séparateur autre que la virgule.

Pendant les apprentissages, on peut voir dans cette fenêtre les différentes étapes et les erreurs éventuelles. En bas de cette fenêtre, vous pouvez vérifier l'état d'avancement de l'expérience. Attendre "finished" avec 0 erreurs ! (cf. figure suivante).



On constate également qu'il est possible d'arrêter le déroulement d'une expérience par l'onglet *Stop*.

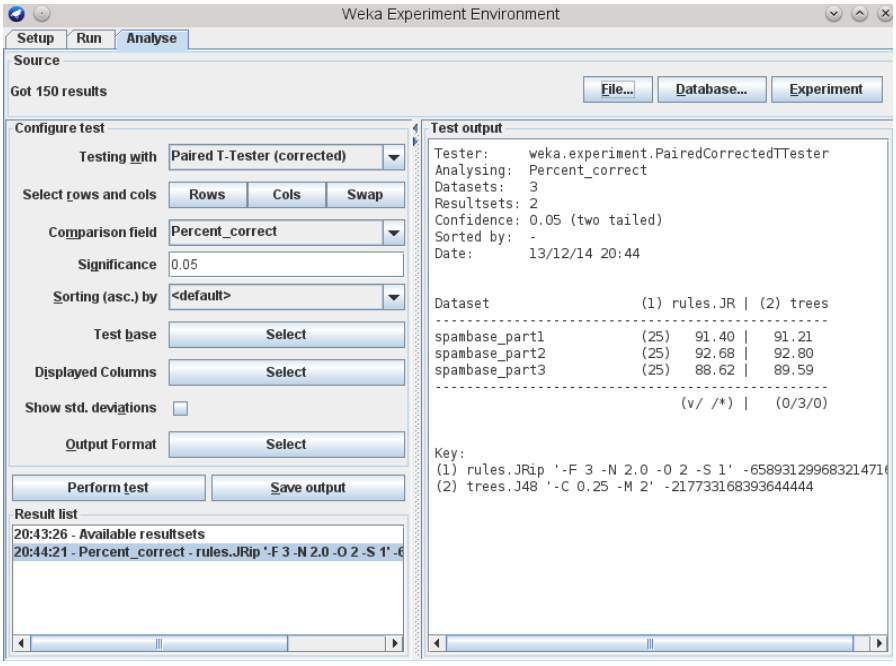
A la fin des apprentissages, le message *there was 0 errors* s'affiche et on peut passer à l'onglet *Analyse*.

On passe ensuite à l'onglet *Analyse.*
En phase d'analyse, on peut choisir (en haut à droite dans l'onglet Analyse, figure suivante) :
- l'onglet *File* pour visaualiser un résultat sauvegardé (P. Ex. *spam-results.arff*, nous n'avons pas encore !)
- une BD de résultats individuels des méthodes pré exécutées (*Database*) ou
- d'analyser les résultats de l'expérience courante (*Experiment*).

Cliquer sur "Experiment" pour sélectionner les valeurs par défaut.

Les tests proposés sont les *T-Test* apparié ou par défaut *T-Test* apparié-corrigé (voir cours capitre 5).
Une fois les autres paramètres dont le degré de signifiance (par défaut = 0,05 → 90% de confiance au verdicte) choisis, on appuie sur *Perform Test* (en bas, à gauche) pour visualiser les résultats dans la fenêtre droite en laissant toutes les autres valeurs par défaut.

✌ Si rien de semblable à la figure ci-dessous ne s'affiche, une des méthodes appliquées ne convient peut être pas à la BD choisie (ce n'est pas le cas ici).



Fenêtre Analyse : dans la matrice affichée (dans la zone de droite), les schémas utilisés sont donnés en colonne tandis que les BDs d'apprentissage sont données en lignes.
Le taux de succès de chaque méthode est donné dans chaque ligne de la BD concernée : 91,40% pour JRip et 91,21% pour J48, en face de la première BD (*spambase_part1*).

Sous la matrice, la notation **v** ou * dans la colonne *(1) rules.JR* (la méthode de référence) indique qu'un résultat spécifique est meilleur (v) ou pire (*) que la méthode de référence, pour le niveau de la signification des tests (ici 0.05 → 90% de confiance). La notation (v/ /*) sous la colonne de la méthode de référence (ici JRip) rappelle simplement la notation. La valeur importante suivante est (0/3/0).

En bas de chaque colonne suivante, on a les valeurs (xx/yy/zz) qui donnent le nombre de fois (xx) où le modèle (de la colonne) a été meilleur que la référence, yy fois la même que la référence, ou zz fois pire que la référence.

Dans cet exemple, J48 est 0 fois meilleure que JRip (le 0 dans 0/-/-), par 3 fois (car 3 BDs) identique à JRip (3 dans -/3/-) et 0 fois pire que la référence JRip (le 0 dans -/-/0). Tout ceci dans la limite de l'indice de confiance (du test de signifiance).

Les deux méthodes donnent donc des résultats sensiblement similaires sur les 3 BDs puisque J48 a obtenu par 3 fois (sur les 3 BDs) des résultats identiques à la référence JRip.

Toujours sous l'onglet *Analyse*, le bouton *Test Base* permet de choisir ou de changer la méthode de référence.

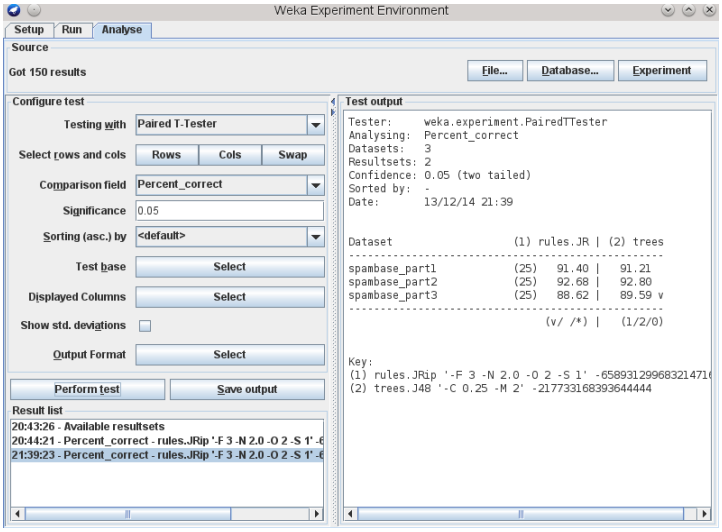
On peut par exemple choisir ici J48 comme référence, ré-appuyer sur *Perform Test* et constater que les résultats des comparaison resteront les mêmes.

Le champ *Comparaison Field* (ce qui a été comparé) a été le taux de succès (*Percent Correct*). On peut changer ce dernier en d'autres indicateurs d'évaluation (e.g. taux d'échec, indice Kappa, différents types d'erreurs du chapitre 5 du cours, etc) par le bouton *Comparaison field*.

On peut également demander le calcul et l'affichage de la variance pour ce choix en cochant *Show std. deviations* suivi de *Perform Test* pour voir les résultats complétés.

La valeur **(25)** en face de chaque BD représente le nombre d'estimations utilisés pour calculer la variance (ici 5 plis et 5 itérations : 5 x 5 = 25).

Si l'on change la méthode de test (*Testing with*) en T-Test apparié (donc non corrigé), on constatera que J48 sera 1 fois meilleure que JRip et par deux fois identique à celle-ci (par le triplet 1/2/0). Une lettre 'v' vient signaler la BD *qui y a contribuée*.



En fixant la valeur de l'indice de confiance à 0.05 (donc une confiance de 90% aux résultats de comparaison, voir chapitre 5), on demande en quelque sorte à ne pas trop (5%) *chipoter* (tergiverser) dans le calcul de la différénece des méthodes!!! C'est comme si dans une élection, il y a un taux de 90% de participation.
Par contre, si vous fixez par exemple cette valeur à 0.35 (vous y mettez une confiance de $100-2*35=30\%$), vous aurez la méthode J48 qui l'emporte mais c'est comme si le taux de participation à une élection n'est que de 30 % (vs. 90% précédents).

On peut sauvegarder les résultats de cette fenêtre dans plusieurs formats (CSV, GnuPlot, HTML, LaTex, Texte).

N.B. : En bas de la fenêtre, dans la zone *Result list (ou Key dans certaine versions de Weka)*, les méthodes utilisés suivies de leurs paramètre sont rappelées. Les valeurs qui suivent sont des identifiants internes à Weka.

12.2 Travail (en bonus) à rendre pour cette partie

• Dans le travail à rendre, comparez J48 et ID3 (lorsque les données ne sont pas numériques, ou bien discrétisez les d'abord) pour constater plus de différences dans les comparaisons.

• Parmi les BDs fournies, celles de la **reconnaissance des lettres** (*letter.arff* fournies dans l'archive sous différentes formes) constituent un bon support pour appliquer cet outil et bien le comprendre.
Pensez d'abord à tester individuellement les méthodes de classification choisies avant de vous lancer dans les comparaisons.

☞ Vous pouvez trouver des informations complémentaires dans le document pdf consacré aux expériences qui se trouve dans le répertoire d'installation de Weka.

13 Quelques commentaires sur les Bases de données (pour ce BE)

(I) vote vote.arff. Fournie avec ce BE.

(II) Supermarket ➙ Arff, fournie avec Weka dans le répertoire "data".

(III) BD Cars

Contient 1728 enregistrements sur des voitures (couvrant tous les cas). Les fichiers :
➙ **Cars.attr** : noms et codage des attributs ;
➙ **Cars.data** : les données (pour *freddie*) ;
➙ **Cars.xls** : fichier dont Cars.data est extrait (pour information).

Les **6 attributs** +1 pour la classe sont (voir le fichier *Cars.attr* pour les symboles utilisés) :

- Le prix d'achat (très élevé, élevé, moyen, bas),
- Le cout de la maintenance (très élevé, élevé, moyen, bas),
- Nombre de portières (2, 4, 4, 5 et plus)
- Nombre de personnes transportables (2, 4, plus)
- Taille du coffre (petit, moyen, grand)
- La sûreté et sécurité (faible, moyen, élevée)
- Le verdict (avis) du consommateur (non acceptable, acceptable, bon, très bon)

La distribution de classe :

Classe	nb instances	pourcentage
non acceptable	1210	70,023 %
acceptable	384	22,222 %
bon	69	3,993 %
très bon	65	3,762 %

(IV) PopularKids

(V) BD météo (**weather catégorielles et mixte**)

Fichier *.arff*.

(VI) BD sur les banques (**Bank-data**) Fichier *.csv, .arff*.

(VII) BD Haggis. Fichier *.arff*.

(VIII) BD de **Pourriels** (*Spam*).

Fichiers : *spambase.DOCUMENTATION.txt* = explication de la BD, *spambase.data* = les données, *spambase.names* = noms et types des attributs.

Indications sur la préparation des données pour des méthodes qui nécessitent une classe non numérique :
- Lire Annexes en section **??** pour le format .data/.names.
- Le fichier *spambase.data* contient toutes les données séparées par des virgules.
- la classe finale = is_spam : booléen (1=TRUE, 0=FALSE).

13.1 Autres DBs

(IX) BD **Titanic** Les données sont fournies sous format *.arff* et *.csv*).

(X) BD de Reconnaissance des lettre (**Letter-recognition**).

Fichiers de données et des noms de **20 000 instances**.

Important : différents fichiers et dossiers sont fournis :
- ➥ Dossier : En-2-Parts-classe-en-tete : la BD découpée en deux. La classe est en 1e colonne.
- ➥ Dossier : En-4-Parts-classe-en-tete : la BD découpée en 4. La classe est en 1e colonne.
- ➥ Dossier : En-2-Parts-classe-ala-fin : la BD découpée en deux. La classe est en dernière colonne.
- ➥ Dossier : En-4-Parts-classe-ala-fin : la BD découpée en 4. La classe est en dernière colonne.

  ➙✻ Utiliser plutôt ces deux derniers pour les comparaisons des méthodes.

(XI) BD classification des animaux (**Zoo**).

Fichier *.arff*.

Remarques : une version non numérique de cette BD est fournie (*zoo-nominal.arff*). Dans cette version, le nombre de pattes est donné par un type énuméré (0..9). Ce qui permet d'appliquer une méthode comme **id3**.
Si vous devez scinder ce fichier en deux (pour les comparaisons de méthodes), veillez à conserver le même rapport entre les classes dans les deux fichiers (stratification).

(XII) BD de Pocker (**Pocker Hand**).

Fichiers : *poker-hand.txt* = les noms des attributs, *poker-hand-training-true.data* (25000 instances)= données pour apprentissge, *poker-hand-testing.data* (1000 000 instances) = données pour tester le modèle obtenu.

Cette BD n' pas un intéret majeur. Elle constitue simplement un bon exemple pour tester des classifieurs. Un dossier contenant *poker-hand-training-true.data* découpé en deux est fourni.

☞ Rappel : le fichier .data ici ne porte pas les noms des attributs (que vous trouverez dans *poker-hand.txt*).

(XIII) BD de crédits (**Credit**).

Fichier *.arff* ('g' pour Germany).

(XIV) BD de recensement (**Recensement-US**).

Fichier *.arff*. 16200 instances environ.

☞ Des données manquantes dans cette BD (cela ne gène pas son utilisation).

(XV) BD sur les universités (**University**).

Fichier *.arff*.

(XVI) BD sur les vitres (**glass**).

Fichier *.csv, .data + .names, .arff*.

Certaines BDs nécessitent une conversion au format *.arff* (voir passgae *.csv* à *.arff* ci-dessus).

13.2 BD météo (ou Golf)

Sur le jeu de données "météo", lancer l'extraction des règles d'association (méthode A Priori). Essayer diverses méthodes de test.
La même base d'instance avec deux attributs numériques :

Num	Temps(S)	Temperature(T)	Humidité(H)	Vent(V)	Classe
1	ensoleillé	Elevée	Elevée	non	N
2	ensoleillé	Elevée	Elevée	oui	N
3	nuageux	Elevée	Elevée	non	P
4	pluvieux	Moyenne	Elevée	non	P
5	pluvieux	Faible	Normale	non	P
6	pluvieux	Faible	Normale	oui	N
7	nuageux	Faible	Normale	oui	P
8	ensoleillé	Moyenne	Elevée	non	N
9	ensoleillé	Faible	Normale	non	P
10	pluvieux	Moyenne	Normale	non	P
11	ensoleillé	Moyenne	Normale	oui	P
12	nuageux	Moyenne	Elevée	oui	P
13	nuageux	Elevée	Normale	non	P
14	pluvieux	Moyenne	Elevée	oui	N

Num	Temps(S)	Temperature(T)	Humidité(H)	Vent(V)	Classe
1	ensoleillé	81	78	non	N
2	ensoleillé	80	90	oui	N
3	nuageux	83	80	non	P
4	pluvieux	75	96	non	P
5	pluvieux	69	75	non	P
6	pluvieux	64	70	oui	N
7	nuageux	65	65	oui	P
8	ensoleillé	72	83	non	N
9	ensoleillé	68	72	non	P
10	pluvieux	71	74	non	P
11	ensoleillé	75	69	oui	P
12	nuageux	70	77	oui	P
13	nuageux	85	70	non	P
14	pluvieux	73	82	oui	N

13.3 BD Banque

Voir BE1.

14 Compléments

14.1 Algorithme de Ripper

• Le principe de base de Ripper est :

- Initiliser $E \leftarrow$ toutes les instances ;
- Sciender E en deux ensembles *Croissance* et *Elagage* avec un ratio 2 : 1
- Pour chaque classe C pour laquelle *Croissance* et *Elagage* contiennent une instance
  - Utiliser un algorithme basique de couverture (voir chapitre 4 du cours) pour créer la meilleure règle parfaite pour C
  - Calculer l'intérêt $W(R)$ pour la règle dans *Elagage* et $W(R-)$ pour la même règle privée de sa toute dernière condition ;
  - Tant que $W(R-) > W(R)$, supprimer la condition finale de la règle et répéter l'étape précédente
  - Pour les règles générées, retenir celle avec la plus grande valeur de $W(R)$
  - Enlever les instances de E couvertes par cette règle

• Dans l'algorithme Ripper, les classes sont examinées une par une dans l'ordre de leur taille (nombre d'instances de la classe).

◦ Un premier ensemble de règles est construit pour la classe en utilisant un algorithme comme ci-dessus.

◦ Une condition d'arrêt supplémentaire dépendante de la longueur de la description (DL) est introduite.

Le calcul du DL est assez complexe prenant en compte le nombre de bits nécessaires pour communiquer un ensemble d'instances et un ensemble de règles avec k conditions (ainsi que d'autres informations).

◦ Une fois produites les règles pour une classe, chaque règle est reconsidérée et deux variantes seront produites en utilisant encore un algorithme comme ci-dessus mais cette fois, les instances couvertes par les autres règles de la classe seront enlevées de l'ensemble *Elagage*.

◦ Un taux de succès calculé sur les instances restantes est utilisé comme critère d'élagage.

◦ Si l'une des variantes donne un meilleur DL, celle-ci remplace la règle.

◦ On passe ensuite à la construction des règles pour les instances non couvertes.

◦ Un test final a lieu pour s'assurer que chaque règle contribue à la réduction de DL avant de passer à la classe suivante.

Les mesures utilisées par Ripper :

$G = Pr[log(p/t) - log(P/T)]$ le Gain d'information

$W = \dfrac{p+1}{t+2}$ l'intérêt

$A = \dfrac{p+n'}{T}$ La justesse de la règle

p le nombre d'instances positivement couvertes par la règles (TP)

n le nombre d'instances négativement couvertes par la règles (FN)

$t = p + n$ le nombre total d'instances couvertes par la règles

P le nombre d'instances positives de la classe

N le nombre d'instances négatives de la classe

$T = P + N$ le nombre total d'instances de la classe