1. Objectif

Induction de règles prédictives – Comparaison de logiciels.

A l'origine, le Data Mining a souvent été assimilé aux techniques d'induction de règles prédictives, par différenciation avec les méthodes issues de la statistique ou de l'analyse de données. C'est bien entendu un peu réducteur. Le Data Mining est avant tout un processus d'extraction de connaissances à partir des données. Il s'inscrit dans le canevas de la « Business Intelligence », démarche qui vise à exploiter les données de l'entreprise à des fins décisionnelles (on parle d'Informatique Décisionnelle en français). A ce titre, il peut mettre en œuvre des techniques d'origines diverses.

Néanmoins, l'induction de règles tient une place privilégiée dans le Data Mining. En effet, elle fournit un modèle prédictif facilement interprétable, on sait lire sans connaissances statistiques préalables un modèle de prédiction de type « **Si** condition **Alors** Conclusion » (ex. **Si** Compte Client à découvert **Alors** Client défaillant pour remboursement des crédits ») ; les règles peuvent être facilement implémentées dans les systèmes d'information (ex. traduction d'une règle en requête SQL).

Nous nous plaçons dans le cadre de l'apprentissage supervisé dans ce didacticiel. Nous cherchons à prédire les valeurs d'une variable cible à partir d'une collection de descripteurs (ex. prédire la défaillance des clients lors d'un remboursement d'un crédit à partir de ses caractéristiques signalétiques – âge, sexe, revenu, etc.). Dans le cas des règles, cela veut dire que nous définissons à l'avance la variable que l'on doit obtenir dans la partie « Conclusion ».

Parmi les méthodes d'induction des règles prédictives, les approches « separate-and-conquer » ont monopolisé les conférences d'apprentissage automatique dans les années 90. Curieusement, le souffle semble un peu retombé aujourd'hui. Plus ennuyeux encore, elles sont peu implémentées, voire inexistantes, dans les logiciels commerciaux. Il faut se tourner vers les logiciels libres issues de l'apprentissage automatique (la communauté « machine learning ») pour les trouver. Pourtant, elles présentent plusieurs atouts par rapport aux autres techniques.

Par rapport aux arbres de décision (http://fr.wikipedia.org/wiki/Arbre_de_d%C3%A9cision) que l'on classe dans la catégorie des méthodes « divide-and-conquer », elles ne sont pas contraintes par la rigidité de la structure arborescente. Le système de représentation est plus puissant. Il faut parfois un arbre complexe pour reproduire logiquement une base de règles assez simple. Des séquences de segmentations se répètent à plusieurs reprises dans différentes parties de l'arbre. On parle de réplications des portions de l'arbre[1]. L'autre conséquence est que l'algorithme d'apprentissage explore différemment l'espace des solutions. Pour faire simple, nous dirons que les arbres recherchent la pureté moyenne sur les feuilles lors d'une opération de segmentation, alors que l'induction de règles recherche la pureté maximale sur une des feuilles. C'est pour cette raison d'ailleurs que Quinlan (1993) a intégré dans C4.5 un post-traitement des règles extraites de l'arbre, appelé « C4.5 rules », afin de rendre plus compact et robuste le modèle de prédiction.

[1] Voir R. Rakotomalala, « Graphes d'Induction », Thèse de Doctorat, Université Lyon 1, 1997 ; page 159.

Par rapport aux règles d'association prédictives (ex. CBA, http://www.comp.nus.edu.sg/~dm2/), elles ne souffrent pas du problème de redondance des règles générées. L'idée serait de produire le plus petit sous-ensemble de règles permettant de prédire avec efficacité les valeurs de la variable cible. Cela évite d'avoir à se lancer dans des procédés hasardeux de simplification de bases de règles ou, pire, d'avoir à gérer les phénomènes de collision (plusieurs règles, certaines étant contradictoires, sont déclenchées lors du classement d'un individu).

Dans ce didacticiel, une fois n'est pas coutume, nous décrivons dans un premier temps les techniques « separate-and-conquer » pour l'induction de règles. Je trouve en effet que ces méthodes sont peu connues des praticiens du Data Mining et, de ce fait, peu utilisées. Pourtant elles sont souvent performantes. Elles constituent une alternative tout à fait valable aux arbres de décision. Nous mettrons l'accent sur les approches par spécialisation, par opposition aux approches par généralisation, plus lentes et quasi-introuvables dans les logiciels.

Dans un second temps, nous montrons la mise en œuvre des différentes variantes implémentées dans les logiciels de Data Mining. Nous utiliserons **Tanagra 1.4.34**, **Sipina Recherche 3.3**, **Weka 3.6.0**, **R 2.9.2** avec le package RWeka, **RapidMiner 4.6**, et **Orange 2.0b**.

2. La stratégie « separate-and-conquer »

L'objectif est de construire des règles prédictives de la forme

Si prémisse **Alors** Conclusion

« Prémisse » est une conjonction de propositions de type « attribut – opérateur de comparaison – valeur ». Par exemple,

Age > 45 **et** Profession = ouvrier

Nous sommes dans un cadre supervisé, la conclusion est forcément une proposition relative à la variable à prédire. Une règle ne désigne qu'une modalité ; une modalité en revanche peut être désignée par plusieurs règles.

La paternité des approches « separate-and-conquer » est attribuée à la famille des algorithmes AQ de Michalski[2] (1969). Elles reposent sur la séquence d'opérations suivante[3] : construire une règle qui prédit au mieux sur une fraction des individus (conquête) ; retirer les individus couverts par la règle de l'ensemble d'apprentissage (séparer) ; réitérer jusqu'à ce qu'on ait épuisé les observations disponibles. Plusieurs pistes peuvent être explorées à partir de ce schéma générique. Dans la phase de séparation, nous pouvons retirer de la base toutes les observations couvertes par la règles, ou uniquement les observations qui sont correctement classées. Les caractéristiques de la base de connaissances obtenue sont différentes à la sortie : les règles peuvent être mutuellement exclusives ou indépendantes.

[2] Voir son impressionnante page de publications : http://www.mli.gmu.edu/michalski/. Il a fallu une vie pour écrire tout cela, il en faudra de même pour tout lire je pense.

[3] Notre texte doit beaucoup à l'excellent « survey » de J. Furnkranz, « Separate-and-Conquer Rule Learning », Artificial Intelligence Review, Volume 13, Issue 1, pages 3-54, 1999.

Dans la phase de construction de la règle (conquête), nous pouvons procéder par *généralisation* (bottom-up) c.-à-d. une observation dans la base est traduite en règle, puis nous essayons de retirer au fur et à mesure des conditions pour couvrir plus d'observations tout en veillant qu'il y ait un minimum de contre-exemples (ou en maximisant un critère qui tient compte du nombre de contre-exemples). Cette approche a fait ses preuves sur des petites bases très peu bruitées. Le temps de calcul devient prohibitif dès que la base devient importante et bruitée. A l'opposé, nous pouvons procéder par *spécialisation* (top-down) c.-à-d. nous cherchons la proposition qui explique le mieux une des modalités de la variable cible, nous rajoutons une seconde proposition qui va toujours dans le même sens, nous procédons ainsi jusqu'à ce qu'une règle d'arrêt soit déclenchée. Correctement programmée, cette technique est très rapide, comparable aux arbres de décision. On peut faire un rapprochement entre les deux techniques d'ailleurs, chaque étape de conquête d'une règle s'apparente à l'optimisation de la qualité d'une des branches d'un arbre de décision.

La phase de conquête peut se réduire à un problème d'optimisation d'un critère judicieusement choisi. Dans cette optique, on a vu fleurir des techniques plus ou moins sophistiquées d'exploration (ex. algorithme génétique, recherche tabou, etc. ; Furnkranz, 1999). Du point de vue théorique, il n'y a rien à dire. Du point de vue pratique, c'est un peu différent. N'oublions pas que nous souhaitons avant tout classer au mieux les individus de la population. En multipliant les hypothèses testées, nous nous exposons au sur apprentissage c.-à-d. les règles classent parfaitement les données d'apprentissage mais s'effondrent en généralisation dans la population. L'autre inconvénient bien sûr est le temps de calcul prohibitif dès que l'on traite des bases de taille un tant soit peu réalistes. Enfin, même si elles sont très présentes dans la littérature, ces approches n'ont cependant pas donné lieu à des outils, même des prototypes, librement accessibles. Pas à ma connaissance en tous cas. Et pourtant j'ai cherché. Même Weka qui possède une bibliothèque très riche d'algorithmes d'induction de règles n'en propose pas. Cela ne concourt pas à leur popularisation.

Dans ce qui suit, nous présentons deux approches pour l'induction de règles prédictives. Elles correspondent aux deux variantes[4] de l'algorithme CN2 implémentées dans Tanagra. Elles s'appliquent naturellement au cadre multi-classes (la variable cible comporte $K \geq 2$ modalités).

2.1. Les listes de décision

Les listes de décision sont dues à Rivest (1987). La méthode CN2 de Clark et Niblett (1989) les ont largement popularisées, notamment parce que leurs auteurs ont mis en ligne un prototype qui permettait à tout un chacun de tester la méthode sur son propre jeux de données.

Base de règles

La méthode produit une base de règles ordonnées et mutuellement exclusives. Lors du classement d'un individu, la première règle est évaluée. Si elle n'est pas déclenchée, on passe à la suivante, etc.

[4] P. Clark and T. Niblett, « The CN2 Induction Algorithm », Machine Learning, 3(4) :361 :283, 1989.
P. Clark and R. Boswell, « Rule Induction with CN2 : Some recent improvements », Machine Learning – EWSL-91, pages 151-163, Springer Verlag, 1991.
Voir aussi : http://www.cs.utexas.edu/users/pclark/software/

Si aucune règle n'est activée, une règle par défaut est utilisée. Le modèle prend donc la forme suivante :

Si Condition 1 **Alors** Conclusion 1
Sinon **Si** Condition 2 **Alors** Conclusion 2
Sinon …
Sinon (Règle par défaut) Conclusion M

L'énorme avantage de cette représentation est qu'il ne peut pas y avoir de collision entre les règles, une et une seule sera activée lors du classement d'un individu.

Prenons l'exemple du fichier « Vote au Congrès ». On cherche à prédire l'appartenance politique des parlementaires (« Républicain » vs. « Démocrate ») à partir de leurs votes[5]. Nous obtenons :

N°	Antecedent	Consequent	Distribution
1	IF physician-fee-freeze in [y] -- synfuels-corporation-cutb in [n]	Class in [republican]	(135; 3)
2	ELSE IF physician-fee-freeze in [n]	Class in [democrat]	(2; 245)
3	ELSE (DEFAULT RULE)	Class in [republican]	(31; 19)

Figure 1 - Liste de décision pour le fichier "Vote au Congrès"

L'**antécédent** correspond à la prémisse, le **conséquent** à la conclusion. La **distribution** montre les fréquences absolues des modalités de la variable à prédire pour les individus couverts par la règle. Plus pure sera la distribution c.-à-d. les observations sont concentrées sur une des modalités, plus précise sera la règle.

Voyons comment sera classé un parlementaire correspondant aux caractéristiques suivantes : *(Physician-fee-freeze = y ; Synfuels-corporation-cutb = y)*.

La règle n°1 n'est pas déclenchée, en effet l'individu ne répond pas simultanément aux deux propositions qui composent la condition (Physician-fee-freeze = y **ET** Synfuels-corporation-cutb = n) ; le système passe à la seconde règle qui n'est pas activée non plus ; finalement l'observation sera classée avec la règle par défaut n°3 : on lui affecte la modalité « républicain ».

Ce type de représentation n'est pas sans rappeler les graphes d'induction[6], une variante des arbres de décision où l'on autorise la fusion entre les nœuds. A la différence que le système est plus compact. L'équivalent « graphe » du modèle ci-dessus (Figure 1) est un peu plus complexe (Figure 2).

[5] http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records

[6] D. Zighed, R. Rakotomalala, « Graphes d'Induction : Apprentissage et Data Mining », Hermès, 2000.
Voir aussi R. Rakotomalala, « Graphes d'Induction », Thèse de Doctorat, Université Lyon 1, 1997 ; chapitre 7.

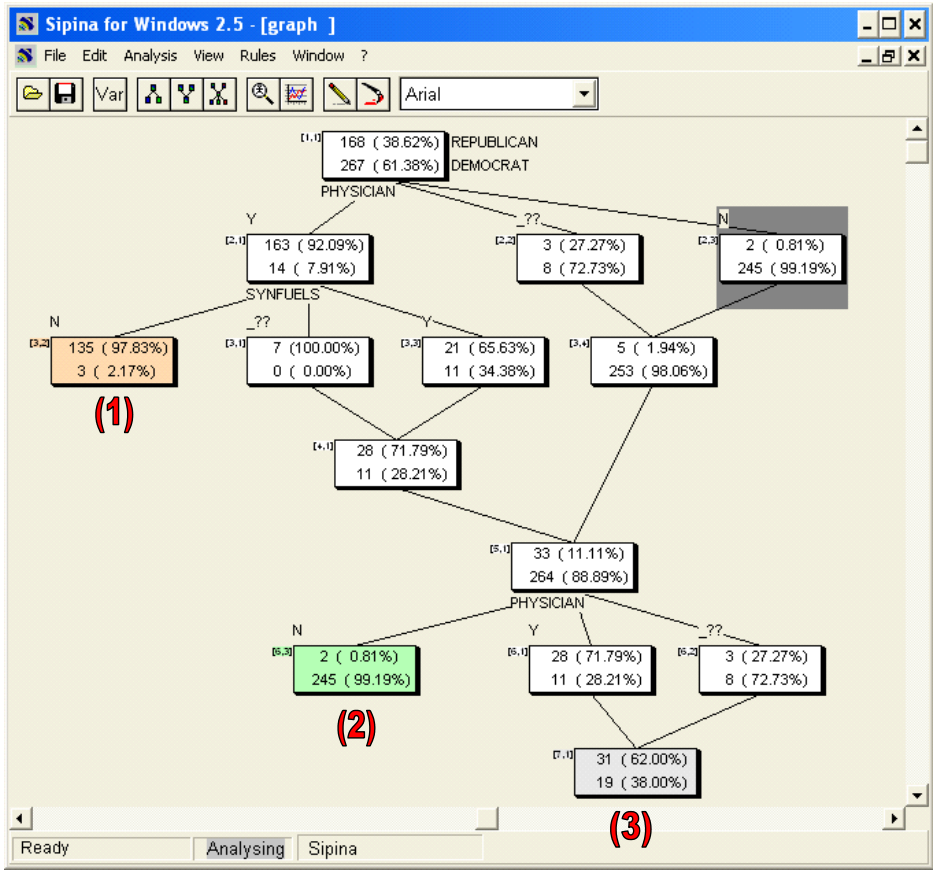


Figure 2 - Graphe de décision logiquement équivalent à la liste de décision (Fichier "Vote")

Les deux modèles sont logiquement équivalents. Les effectifs sur les feuilles sont cohérents. Mais on conviendra que la représentation sous forme de « listes de décision » est nettement plus lisible. C'est un argument de poids lorsque nous devons interpréter le processus de classement.

Algorithme d'apprentissage

L'algorithme d'apprentissage repose sur le principe « separate-and-conquer ». Il peut être résumé en deux étapes imbriquées. Une première procédure encapsule la recherche des règles.

```
DecisionList (var.cible, var.prédictives, exemples)

Base de règles = ∅
Répéter
      Règle = Spécialiser (var.cible, var.prédictives, exemples)
      Si (Règle != NULL) Alors
            Base de Règles = Base de Règles + {Règle}
            Exemples = Exemples – {Individus couverts par la Règle}
      Fin Si
Jusqu'à (Règle = NULL)
Base de Règles = Base de Règles + {Règle par défaut (exemples)}
Renvoyer (Base de règles)
```

A chaque étape, la méthode cherche par spécialisation la règle la plus intéressante pour les observations disponibles. Si on en trouve une, elle est ajoutée à la base de règles. Les observations

couvertes sont retirées de l'ensemble de données. Le processus s'arrête lorsqu'il n'est plus possible de produire une règle. Le système élabore alors la règle par défaut à partir des observations restantes. Il s'agit simplement de désigner la classe majoritaire.

La fonction « Spécialiser » tient un rôle capital dans ce processus. Elle cherche par adjonction successives de propositions une règle couvrant un sous-ensemble d'observations le plus pur possible du point de vue de la variable à prédire c.-à-d. les individus couverts par la règles sont prioritairement associés à une des modalités de la variable cible.

```
Spécialiser (var.cible, var.prédictives, exemples)
Règle = NULL
Max.Mesure = -∞
Tant Que (Règle.Arrêt(Règle) == FAUX)
  Ref.Mesure = -∞
  Pour Chaque Proposition Candidate
     Mesure = Evaluation (Règle, Proposition)
     Si (Mesure > Ref.Mesure)
     Alors
        Ref.Mesure = Mesure
        Proposition* = Proposition
     Fin Si
  Fin Pour
  Si (Ref.Mesure > Max.Mesure)
     Règle = Règle x Proposition*
     Max.Mesure = Ref.Mesure
  Fin Si
Fin Tant Que
Renvoyer (Règle)
```

Il s'agit d'une optimisation gloutonne (hill-climbing) assez basique[7]. Le processus est stoppé lorsqu'il n'est plus possible d'améliorer la mesure d'évaluation des règles.

La fonction « Evaluation » intervient à deux niveaux dans la recherche des solutions :

1. Tester si la règle formée avec l'adjonction de la proposition candidate est acceptable. Elle l'est dans deux cas : elle apporte significativement de l'information au sens du test du KHI-2 d'un tableau de contingence (la p-value est plus petite que le paramètre « Signif.Level for pruning » ; la

[7] Notons que la méthode CN2 authentique de Clark et Niblett (1989) utilise une méthode « beam-search » plus sophistiquée, mais aussi plus gourmande en ressources de calcul. L'idée est de maintenir « k » solutions en parallèle lors de l'exploration des solutions. Cette option est implémentée dans le logiciel Orange par exemple. Lorsque « k=1 », nous retrouvons une optimisation « hill-climbing ». Franchement, je doute fort que démultiplier l'espace de recherche en augmentant la valeur de « k » permette d'aboutir à un classifieur plus performant en généralisation, surtout sur des bases réelles souvent fortement bruitées.

valeur par défaut est 0.10 dans Tanagra) ; le nombre d'observations couverts est suffisamment grand (supérieur ou égal à « Min. Support of Rule »[8] ; 10 dans Tanagra).

Prenons l'exemple de l'adjonction de la dernière proposition (Synfuels = n) dans la première règle de notre exemple « Vote au Congrès » (Figure 1). Nous utilisons les effectifs affichés dans le graphe équivalent (Figure 2).

	Root		PHYSICIAN = Y		SYNFUELS = N
Republican	168	==>	**163**	**??? ==> ???**	**135**
Democrat	267		**14**		**3**

Avec la nouvelle proposition, la règle couvre $n_a = 135 + 3 = 138$ observations, supérieur au seuil minimum de 10. La première condition est remplie. Quant au test du KHI-2, nous formons le tableau de contingence suivant

	+Synfuels = n	Uncovered	Physician
Republican	**135**	28	163
Democrat	**3**	11	14
Total	**138**	39	177

Nous en déduisons un $\chi^2 = 28.29$, avec une p-value = 1.05×10^{-7} (< 0.10). L'ajout de la nouvelle proposition dans la prémisse est entériné.

2. La fonction sert ensuite à quantifier la pertinence de la règle. Deux mesures sont disponibles dans Tanagra : l'entropie de Shannon, elle caractérise la pureté des groupes ; la J-Mesure qui caractérise la distorsion entre la distribution actuelle et la distribution initiale des classes. Au-delà des formules et des interprétations, nous retiendrons surtout que la première produit des règles fortement spécialisées, et donc des bases de connaissances plus complexes. Il faut la privilégier lorsque nous travaillons sur des petites bases faiblement bruitées. La seconde en revanche privilégie les solutions plus simples, avec comparativement moins de règles, chacune d'elles couvrant une proportion plus élevée d'observations. Nous l'utiliserons de préférence lorsque nous traitons de grandes bases, fortement bruitées.

Soit p_k la probabilité P(Y=k) à la racine de la structure, la distribution initiale des classes dans l'échantillon d'apprentissage ; $p_{k/a}$ la probabilité P(Y=k/a) pour une règle avec l'antécédent « a » ; p_a la proportion des observations couvertes par la règle.

Reprenons notre première règle R de notre modèle, les distributions sont les suivantes

[8] En apprentissage automatique, le support d'une règle désigne le nombre d'observations couvertes par la règle. Le sens du terme a été un peu modifié avec l'avènement des règles d'association. Il correspondrait plutôt aux nombres d'observations positives couvertes par la règle. Pour éviter la confusion, certains logiciels adoptent le terme « couverture » (covering). Pourquoi pas. En ce qui nous concerne, nous nous en tenons au terme « support » en faisant référence à la définition originelle.

	Root	SYNFUELS = N
Republican	168	135
Democrat	267	3
Republican	0.386	0.978
Democrat	0.614	0.022

L'entropie de Shannon ne tient pas compte de la distribution initiale, elle s'écrit

$$S(a) = \sum_k p_{k/a} \log p_{k/a}$$

Tous les logarithmes sont à base 2 dans notre texte. Pour notre règle, S(a) est égale à

$$S(a) = \left[0.978 \times (-0.031) + 0.022 \times (-0.120)\right] = -0.151$$

La J-Mesure tient compte de la distribution a priori des classes, nous devons la privilégier lorsque cette dernière est initialement déséquilibrée

$$J(a) = p_a \times \sum_k \frac{p_{k/a}}{p_k} \log \frac{p_{k/a}}{p_k}$$

Toujours pour le même exemple, nous obtenons

$$J(a) = \frac{138}{435} \times \left[2.533 \times (3.396) + 0.035 \times (-0.171)\right] = 1.023$$

Tous deux mettent l'accent sur la pureté des groupes. Mais la première ignore délibérément le poids du groupe. Considérons une règle concurrente R' avec l'antécédent a', les distributions sont les suivantes :

	Root	a'
Republican	168	50
Democrat	267	1
Republican	0.386	0.980
Democrat	0.614	0.020

Nous obtenons respectivement : S (a') = -0.139 > S (a) = -0.151 ; J (a') = 0.381 < J (a) = 1.023. La mesure S(.) préfèrera la règle R', plus pure même si elle couvre moins d'individus ; alors que la mesure J(.) choisira la règle R, de poids plus élevé, même si elle est (très légèrement) moins pure. Ces dissimilitudes de comportement expliquent les différences entre les caractéristiques des bases de règles obtenues à l'issue du processus d'apprentissage. Les règles obtenus avec la mesure S(.) sont plus spécialisées, elles sont aussi plus nombreuses.

Traitement des descripteurs continus

L'analogie avec les arbres de décision est assez parlante disions-nous plus haut. Mais nous ne spécialisons qu'une branche à chaque étape. Cela induit une différence notable au niveau de l'implémentation. Le traitement des descripteurs continus n'est pas aussi facile. Le plus simple alors est de les discrétiser dans une phase de prétraitement, en utilisant de préférence des techniques contextuelles, tenant compte des valeurs de la variable à prédire. Pour ma part, j'utilise quasi-

systématiquement la méthode MDLPC de Fayyad et Irani (1993). Elle a largement fait ses preuves tant en robustesse qu'en performances[9]. Bien entendu, on peut s'en tenir également à des techniques plus frustes, non contextuelles, telles que la discrétisation en intervalles de largeur égales, ou en intervalles de fréquences égales. Les nombreuses expérimentations ont montré cependant qu'elles sont moins pertinentes dans un cadre supervisé.

2.2. Les règles indépendantes

Base de règles

Peu de temps après leur premier article, les auteurs de CN2 en ont publié un autre où ils préconisent finalement la production de règles indépendantes. Leur argumentation repose sur la difficulté à interpréter les listes de décision. En effet, lorsqu'elles sont ordonnées, pour lire correctement la règle n°i, nous devons tenir compte des (i-1) règles précédentes. L'interprétation n'est pas facile dès que le classifieur contient un nombre important de règles.

Le classifieur s'écrit dorénavant de la manière suivante

Si Condition 1 **Alors** Conclusion 1
Si Condition 2 **Alors** Conclusion 2
…
(Règle par défaut) Conclusion M

Lors du classement d'un nouvel individu, nous devons toutes les tester. Si aucune d'entre elles n'est activée, nous utilisons la règle par défaut. L'ennui est que plusieurs règles, avec parfois des conclusions contradictoires, peuvent être déclenchées. Nous devons adopter une stratégie pour gérer les collisions. Les auteurs proposent une solution que nous illustrons à l'aide d'un exemple.

L'algorithme d'apprentissage a produit la base de règles suivante pour le fichier « Vote au Congrès ».

N°	Antecedent	Consequent	Distribution
1	IF physician-fee-freeze in [y] -- adoption-of-the-budget-re in [n] -- el-salvador-aid in [y]	Class in [republican]	(137; 4)
2	IF physician-fee-freeze in [y] -- anti-satellite-test-ban in [y] -- adoption-of-the-budget-re in [y]	Class in [republican]	(17; 0)
3	IF physician-fee-freeze in [y] -- synfuels-corporation-cutb in [n]	Class in [republican]	(9; 3)
4	(DEFAULT RULE)	Class in [democrat]	(5; 261)

Figure 3 - Base de règles indépendantes - Fichier "Vote au Congrès"

Pour classer un individu portant les caractéristiques (physician-fee-freeze = y ; adoption-of-the-budget-re = n ; el-savlador-aid = y ; synfuels-corporation-cutb = n), les règles n°1 et n°3 seront activées. Nous effectuons l'addition des fréquences observées pour chaque règle, soit (republican = 137 + 9 = 146 ; democrat = 4 + 3 =7), et nous affectons à l'individu la classe majoritaire, soit « republican » dans notre exemple (137 > 7).

Cette stratégie fonctionne plutôt bien en pratique. Nous aurions pu en imaginer d'autres, plus simples : par exemple trier les règles par ordre de confiance (ou tout autre critère de précision) décroissant et classer à l'aide de la première règle déclenchée.

[9] http://tutoriels-data-mining.blogspot.com/2008/03/discrtisation-contextuelle-la-mthode.html

Algorithme d'apprentissage

La construction de règles indépendantes induit des modifications au niveau de l'algorithme d'apprentissage. L'idée est de prendre tour à tour chaque classe de la variable à prédire, en commençant par celle qui est la plus rare dans l'ensemble d'apprentissage. On ne traite pas la dernière modalité la plus fréquente, elle correspond simplement à la conclusion par défaut.

L'algorithme encapsulant la recherche des règles s'écrit maintenant

```
Induction Règles (var.cible, var.prédictives, exemples)

Base de règles = ∅
Trier les classes par ordre de fréquence croissante
Pour chaque classe « c » sauf la dernière
 Sample = Exemples
 Tant que Règle != NULL
  Règle = Spécialiser (c, var.cible, var.prédictives, sample)
  Si (Règle != NULL) Alors
   Base de Règles = Base de Règles + {Règle}
   Sample = Sample – {Individus « positifs » couverts par la Règle}
  Fin Si
 Fin Tant Que
Fin Pour
Exemples = Exemples – {Individus couverts par au moins une règle}
Base de Règles = Base de Règles + {Règle par défaut (exemples)}
Renvoyer (Base de règles)
```

L'approche s'applique naturellement au cadre multi-classes. La procédure « Spécialiser » tient toujours une place prépondérante.

```
Spécialiser (classe, var.cible, var.prédictives, exemples)
Règle = NULL
Max.Mesure = -∞
Répéter
   Ref.Mesure = -∞
   Proposition* = NULL
   Pour Chaque Proposition Candidate
       Mesure = Evaluation (classe, Règle, Proposition)
       Si (Mesure > Ref.Mesure)
       Alors
          Ref.Mesure = Mesure
          Proposition* = Proposition
       Fin Si
   Fin Pour
   Si (Ref.Mesure > Max.Mesure)
       Règle = Règle x Proposition*
       Max.Mesure = Ref.Mesure
   Fin Si
Jusqu'à (Proposition* == NULL)
Si Significative(Règle)==VRAI Alors Renvoyer (Règle) Sinon Renvoyer(NULL)
```

Ici également, « Spécialiser » correspond à une optimisation gloutonne relativement simple. Mais nous ciblons nommément une « classe » de la variable à prédire durant le processus. Nous ne retiendrons

donc que les règles ayant pour conclusion « Variable à prédire = classe » à chaque étape, chacune des classes étant analysées tour à tour via la procédure globale « Induction Règles ».

Comme précédemment, la fonction « Evaluation » joue un rôle central. Il sert dans un premier temps à vérifier si la règle couvre suffisamment d'observations en comparant le support de la règle avec le seuil « Support Min » (10 par défaut) demandé par l'utilisateur. Puis il sert dans un second temps à quantifier la pertinence de la règle[10] en confrontant la distribution courante de la classe avec la distribution observée à la racine du processus d'optimisation. Attention, dans le cadre d'induction de règles indépendantes, la racine est calculée localement, au début du processus de spécialisation. La configuration peut être illustrée à l'aide d'un tableau de contingence :

	Antécédent	Non (Antécédent)	Effectif
Conséquent [Obs. ∈ classe]	n_{ac}		n_c
Non (Conséquent) [Obs. ∉ classe]			
Effectif	n_a		n

Avec :

- n est le nombre d' « exemples » envoyée à la procédure « spécialiser »;
- n_a est le nombre d'observations couvertes par l'antécédent (prémisse) de la règle ;
- n_c est le nombre d'observations positives (appartenant à « classe ») dans les exemples envoyées à la procédure ;
- n_{ac} est le nombre d'observations positives couvertes par la règle.

Tanagra utilise trois types de mesures pour évaluer les règles. Elles sont ordonnées par pureté croissante des règles induites, la dernière étant paramétrable :

- La Statistique de Confiance

$$ZC(c,a) = \frac{n_{ac} - n_a n_c / n}{\sqrt{\frac{(n_a(n-n_a)/n)(n_c(n-n_c)/n)}{n-1}}}$$

- La Statistique du contre-exemple, qui n'est autre que l'indice d'implication de Lerman, Gras et Rostam (1981 -- http://msh.revues.org/document2213.html)

$$ZI(c,a) = -\frac{(n_a - n_{ac}) - (n-n_c)n_a/n}{\sqrt{\frac{(n-n_c)n_a}{n}}}$$

- Un indice original, la J-Mesure asymétrique,

[10] Voir « Mesures d'intérêt des règles dans A PRIORI MR » -- http://tutoriels-data-mining.blogspot.com/2009/02/mesures-dinteret-des-regles-dans-priori.html pour une étude approfondie des mesures d'évaluation des règles.

$$J_\beta(c,a) = \left(\frac{n_a}{n}\right)^{\frac{1}{\beta}} \times \left[\frac{n_{ac}}{n_a}\log\frac{n_{ac}}{n_a} - \frac{n_c - n_{ac}}{n - n_a}\log\frac{n_c - n_{ac}}{n - n_a}\right]$$

Par rapport à la J-Mesure usuelle, $J_\beta(c,a)$ indique bien la distorsion entre les distributions. Mais elle est asymétrique car elle donne une place prépondérante à la classe « c ». La mesure prend une valeur élevée lorsque cette dernière est surreprésentée. C'est ce que l'on souhaite dans le processus de recherche de la règle. La mesure est de plus paramétrée par une pondération β (« Weight »). Elle permet de réaliser un arbitrage entre l'importance du groupe (taille du groupe par rapport à l'effectif total) et la surreprésentation de « c » dans le groupe. Plus elle sera élevée, plus nous donnerons de l'importance à la pureté du groupe, plus nous obtiendrons des règles fortement spécialisées. La valeur par défaut est $\beta = 4$; elle réalise un bon compromis entre la spécialisation des règles et la proportion d'observations couvertes. Mais, à l'évidence, il serait plus judicieux de l'adapter aux caractéristiques de la base et des objectifs de l'analyse. S'il y a danger de sur-apprentissage (ex. bases fortement bruitées, rapport entre nombre d'attributs et d'exemples défavorable, etc.), nous avons intérêt à le diminuer ; dans le cas contraire (ex. bases faiblement bruitées, nombre d'observations très élevé au regard du nombre de descripteurs, etc.), nous l'augmenterons plutôt.

Les deux premières mesures (ZC et ZI) correspondent à des statistiques de test. Il est possible d'évaluer la significativité de l'écart entre la distribution des classes à la racine, et celle observée sur les individus couvertes par la règle. C'est le rôle de la procédure « Significative » dans l'algorithme ci-dessus. La règle est acceptée si les distributions diffèrent significativement c.-à-d. la *p-value* du test est plus petit que le seuil (Signif. Level = 0.05 par défaut) que l'on s'est choisi. Il faut le prendre comme une procédure de pré-élagage et non pas comme un véritable test au sens statistique du terme. Si nous abaissons la valeur du seuil, nous obtiendrons moins de règles au final.

Un exemple numérique

Voyons comment calculer ces différentes mesures sur la première règle (n°1) produite sur le fichier « Vote au Congrès » (Figure 3). Nous pouvons le dire maintenant, nous avions utilisé la J-mesure asymétrique avec $\beta = 20$ pour obtenir un modèle volontairement complexe à des fins pédagogiques (avec les paramètres usuels, une seule règle aurait été produite) :

Le tableau de contingence associé s'écrit

	antecedent	uncovered	Root
republican (c)	137	31	168
democrat (not c)	4	263	267
Total	141	294	435

ZC (c,a)	17.3472	p-value	0

ZI (c,a)	8.8730	p-value	0

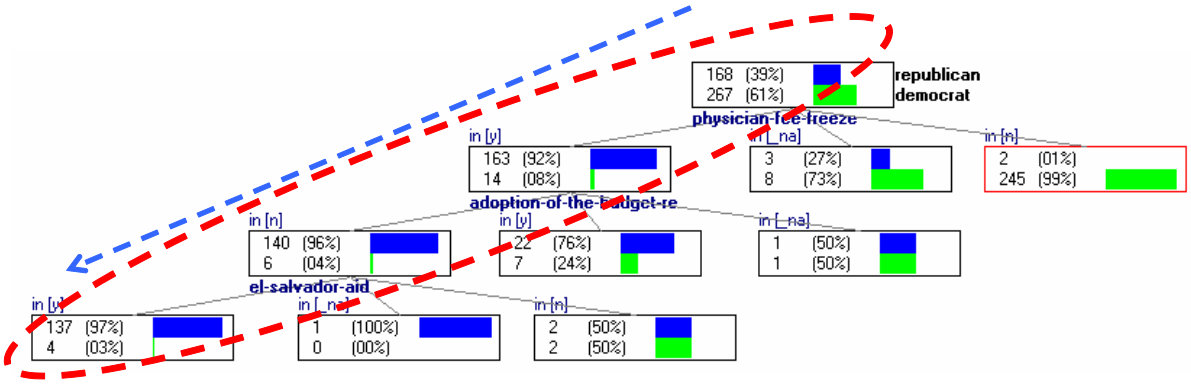
J20(c,a)	0.2853

Comparer les valeurs des mesures sur les mêmes règles n'a pas trop de sens. Nous nous bornerons à dire que les deux statistiques de test concluent à la significativité de la règle (ou plus exactement à un écart significatif entre la distribution des classes sur la racine et sur le groupe couvert par la règle).

Détaillons les calculs pour la J-Mesure asymétrique :

$$J_\beta(c,a) = \left(\frac{n_a}{n}\right)^{\frac{1}{\beta}} \times \left[\frac{n_{ac}}{n_a}\log\frac{n_{ac}}{n_a} - \frac{n_c - n_{ac}}{n - n_a}\log\frac{n_c - n_{ac}}{n - n_a}\right]$$
$$= \left(\frac{141}{435}\right)^{\frac{1}{20}} \times \left[\frac{137}{141}\log\frac{137}{141} - \frac{31}{294}\log\frac{31}{294}\right]$$
$$= 0.2853$$

Plus intéressante est la comparaison des valeurs d'un indice sur des configurations imbriquées. La règle est issue d'un processus d'optimisation. Nous disposons de la distribution des classes sur les situations intermédiaires, les nœuds de l'arbre qui mènent jusqu'à une feuille.



Voyons dans quelle mesure la règle finale (la feuille) est justifiée par rapport à la règle intermédiaire ne contenant qu'une seule proposition (physician-fee-freeze = y) située sur la branche :

R1' : Si Physician-Fee-Freeze = y Alors Class = Republican (163 ; 14)

La règle est clairement moins pure (92% de républicains vs. 97%), mais elle couvre plus d'observations (177 individus vs. 141).

Le tableau de contingence s'écrit ici

	antecedent'	uncovered	Root
republican (c)	163	5	168
democrat (not c)	14	253	267
Total	177	258	435

ZC (c,a')	18.9500	p-value	0
ZI (c,a')	9.0799	p-value	0
J20(c,a')	0.0007		

Manifestement, les deux premières mesures auraient préféré la règle plus simple R1' [ZC(a',c) = 18.95 > ZC(a,c) = 17.35 ; ZI(a',c) = 9.08 > ZI(a,c) = 8.87]; alors que la J-Measure, parce que nous avons donné une importance exagérée à la pureté des règles, préfère la solution plus complexe R1 [J20(a',c) = 0.0007 < J20(a,c) = 0.2853].

Honnêtement, nous ne pouvons pas dire si R1' est meilleur que R1 (ou inversement). Et n'étant pas expert politique, je me garderai bien de le faire. En revanche, cet exemple illustre bien l'impact des paramètres de la méthode sur l'orientation de l'algorithme d'apprentissage vers les solutions souhaitées. C'est ce qui importe en apprentissage automatique.

2.3. Les autres algorithmes d'apprentissage

Nous avons présenté deux approches très simples, implémentées dans Tanagra, pour produire des règles dans cette section. Bien évidemment, il en existe d'autres. Plutôt deux fois qu'une d'ailleurs. Dans son « survey », Furnkranz (1999) en recense pas moins de 40. Et encore, il doit être en dessous de la vérité, il s'est principalement concentré sur les publications en langue anglaise au sein de la communauté « machine learning ».

Pour résumer, si l'on reste toujours dans le cadre des propositions de type « attribut – valeur », la bataille se concentre principalement au niveau du choix de la mesure d'évaluation des règles et des stratégies de pré et (surtout) de post élagage pour optimiser les règles et bases de règles. Je reste assez sceptique quant à l'efficacité des usines à gaz qui en résultent. S'il faut distinguer une approche, je choisirai la méthode RIPPER de Cohen (1995 - http://www.cs.cmu.edu/~wcohen/) principalement à cause de sa notoriété. C'est un peu la méthode de référence s'agissant des techniques d'induction de règles prédictives. Elle a beaucoup été mise à contribution, notamment dans le « text mining ». Elle est disponible sous l'appellation JRIP dans le logiciel Weka. Nous en reparlerons plus loin.

3. Données

Un établissement bancaire souhaite placer un nouveau produit, une assurance vie, auprès de ses clients. Pour identifier les caractéristiques des clients qui répondent positivement à l'offre, une étude prospective a été lancée sur une région. Il en résulte un fichier[11] recensant la description des clients (âge, ancienneté dans l'établissement, la possession d'une carte de crédit, etc.) et le fait qu'ils aient répondu positivement à l'offre ou non (target = positif ou négatif).

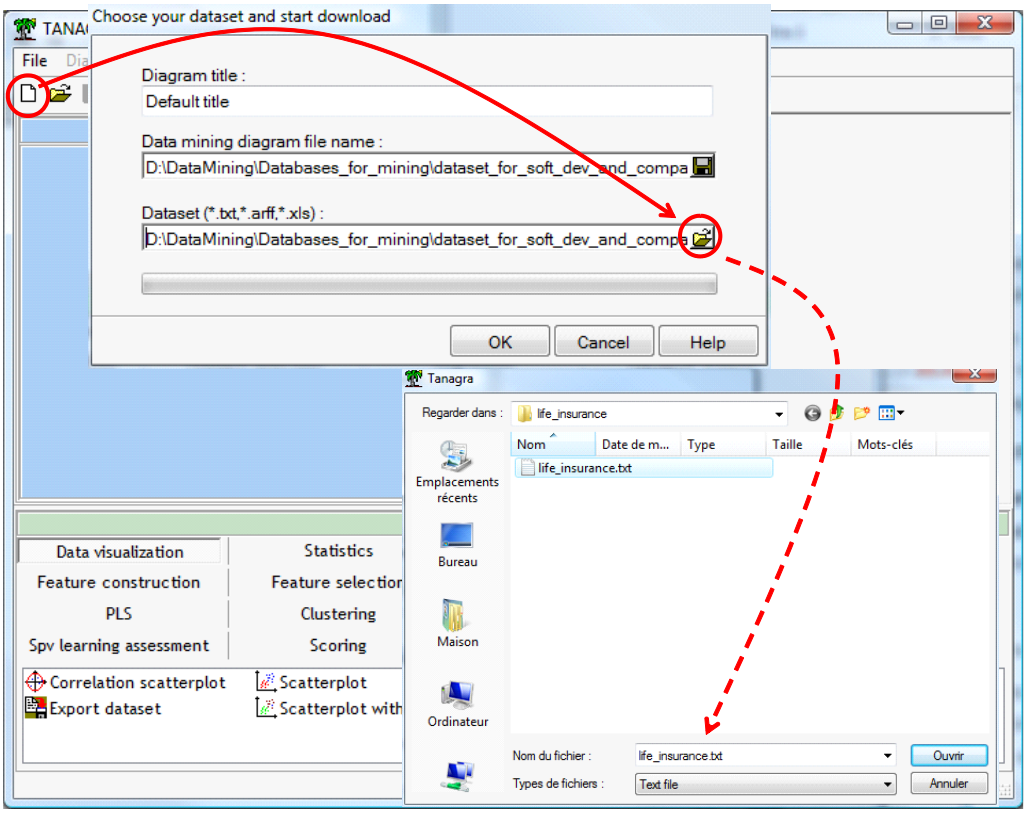
Les données (79.838 observations) ont été volontairement équilibrées (39.874 positifs vs. 39.964 négatifs). On veut réserver une moitié des individus pour construire les règles prédictives (l'échantillon d'apprentissage) ; l'autre, l'échantillon test, servira à l'évaluation des performances.

[11] http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/life_insurance.zip ; l'archive contient le même ensemble de données au format TXT (texte avec séparateur tabulation) et ARFF (pour Weka et RapidMiner).

4. Traitements avec Tanagra

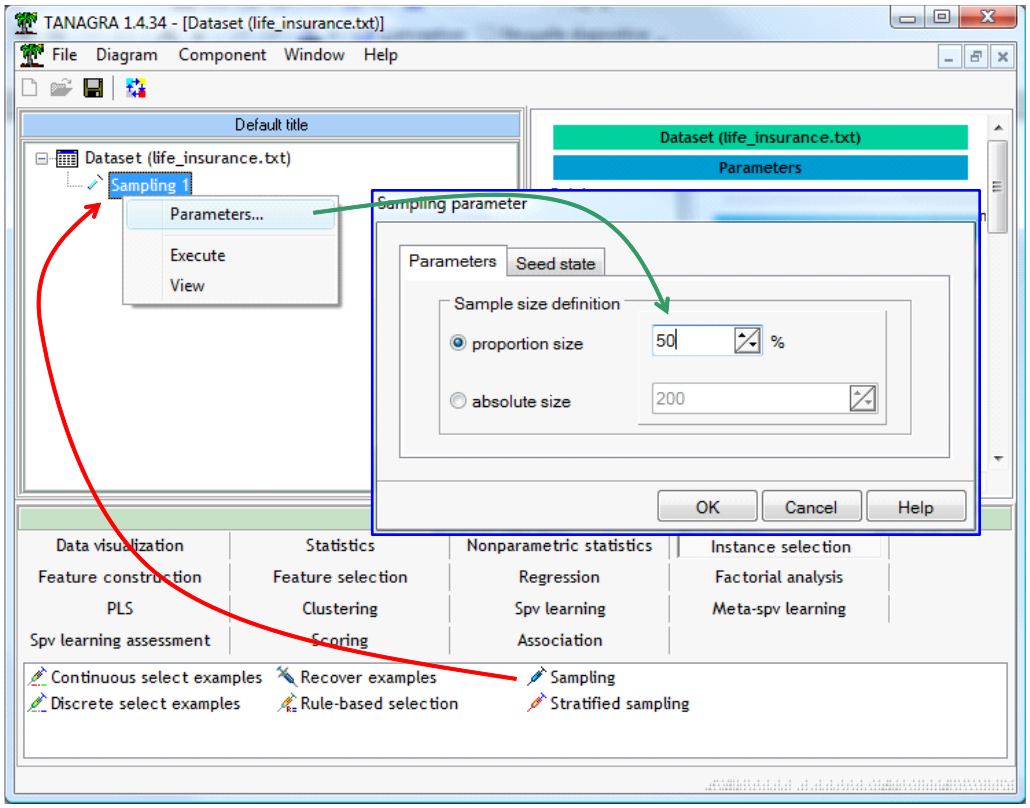
4.1. Importation des données

Après avoir démarré Tanagra, nous créons un nouveau diagramme avec FILE / NEW. Nous importons le fichier de données LIFE_INSURANCE.TXT (fichier texte avec séparateur tabulation).

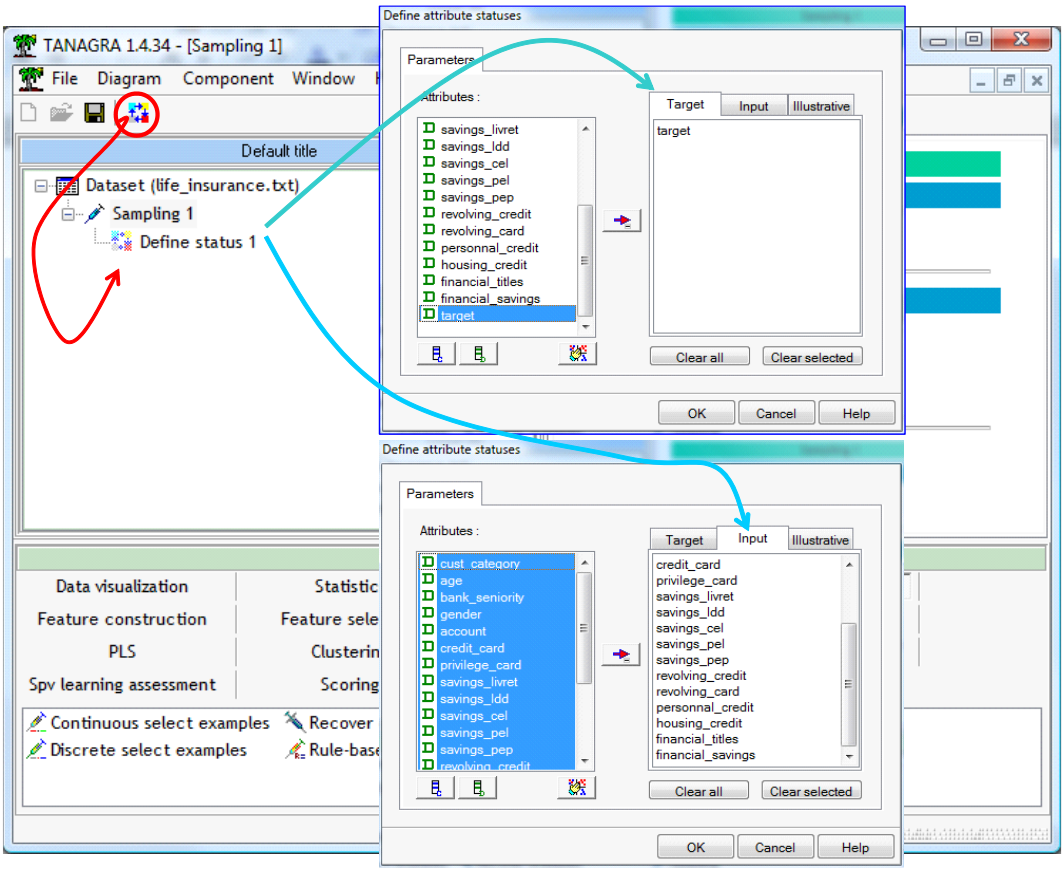


Nous vérifions que les comportent bien 79.838 exemples et 19 attributs, tous discrets.

Nous souhaitons partitionner les données en échantillons d'apprentissage et de test. Nous utilisons le composant SAMPLING (onglet INSTANCE SELECTION), nous le paramétrons pour que 50% des observations soient sélectionnées pour les calculs.

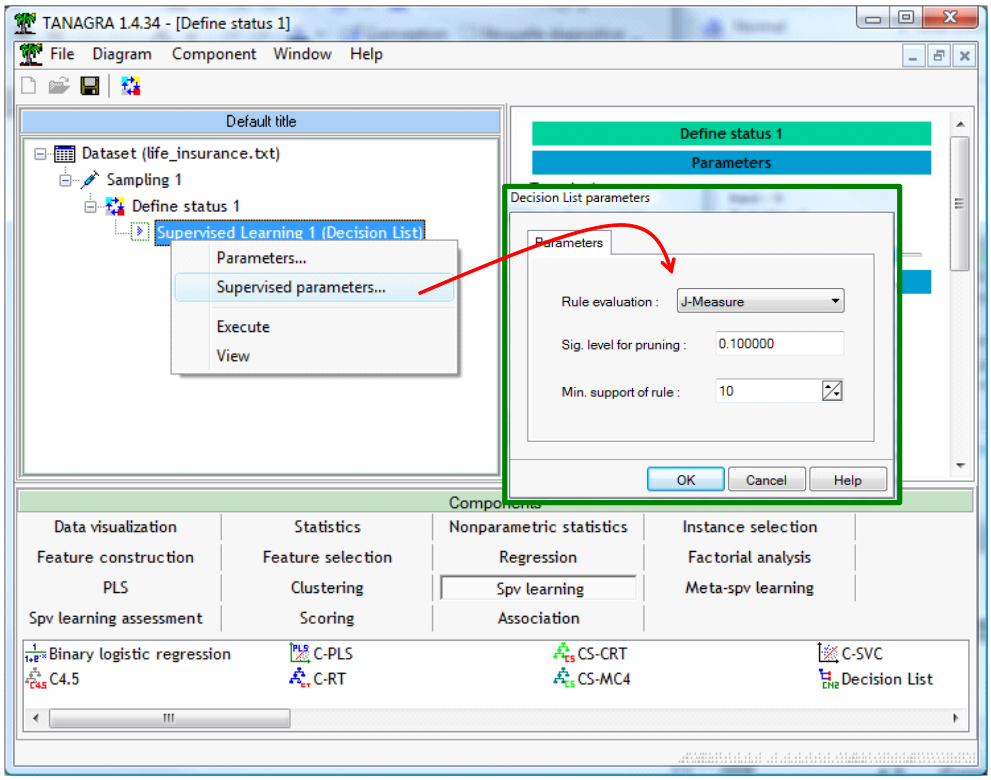


Nous définissons maintenant le rôle des variables avec le composant DEFINE STATUS. Nous plaçons « target » en TARGET, les autres en INPUT.



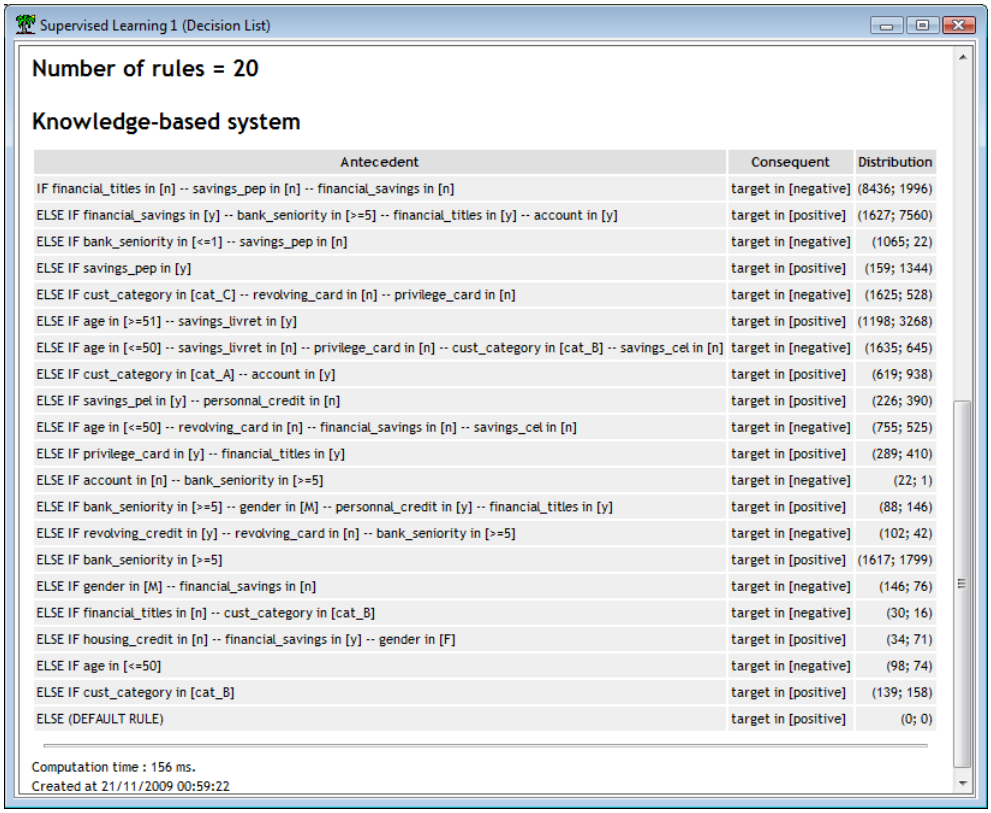
4.2. Construction et évaluation des Listes de Décision

Nous sommes prêts maintenant pour l'élaboration des modèles de prédiction. Nous souhaitons construire les Listes de décision. Nous insérons le composant DECISION LIST dans le diagramme[12]. Nous activons le menu SUPERVISED PARAMETERS pour consulter les paramètres par défaut, nous constatons que la J-MEASURE a été automatiquement sélectionnée. C'est une bonne stratégie pour une première approche, nous obtenons peu de règles et le temps de calcul est réduit.

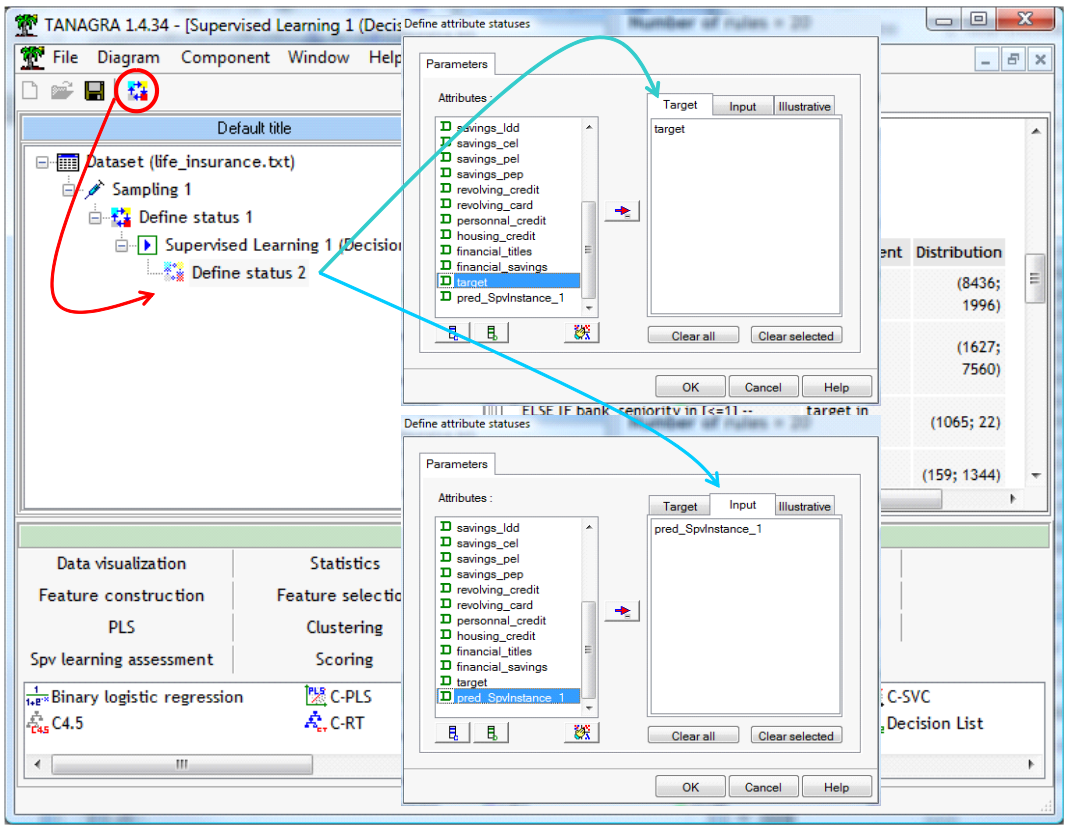


Nous validons le paramétrage, puis nous cliquons sur VIEW pour construire les règles. Le système a produit 20 règles en 156 ms.

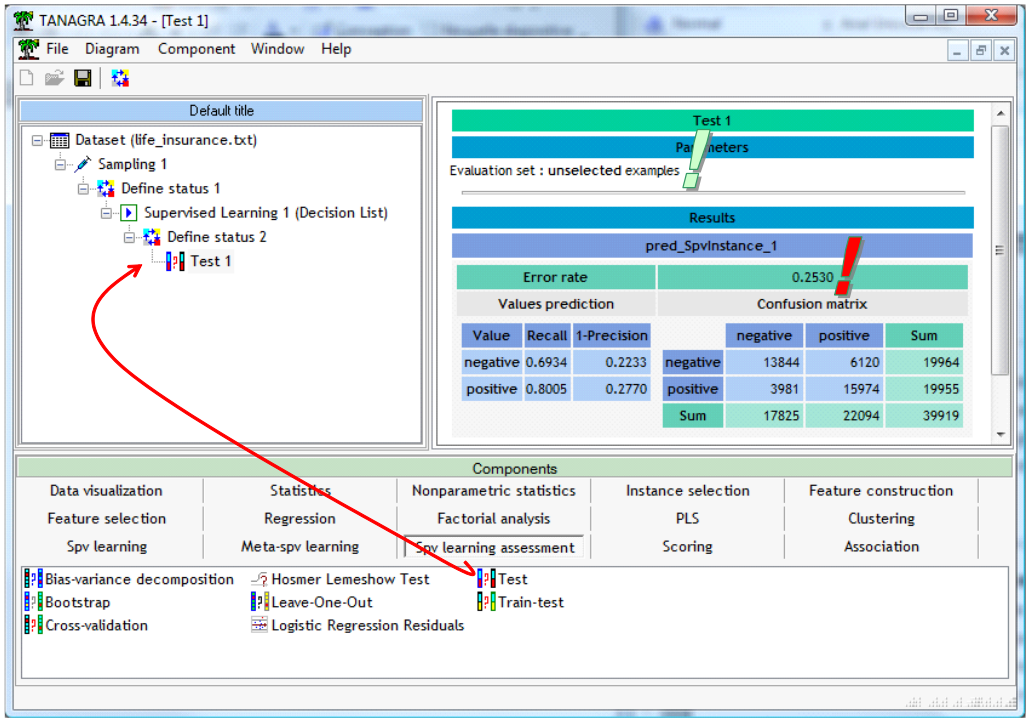
[12] Attention, si vos descripteurs sont continus, vous devrez les discrétiser au préalable. Voir : http://tutoriels-data-mining.blogspot.com/2008/03/listes-de-dcision-vs-arbres-de-dcision.html



Reste à évaluer les performances en généralisation. Nous introduisons de nouveau le composant DEFINE STATUS, nous plaçons toujours « target » en TARGET, nous mettons en INPUT la variable nouvellement créée par le modèle (PRED_SPVINSTANCE_1).

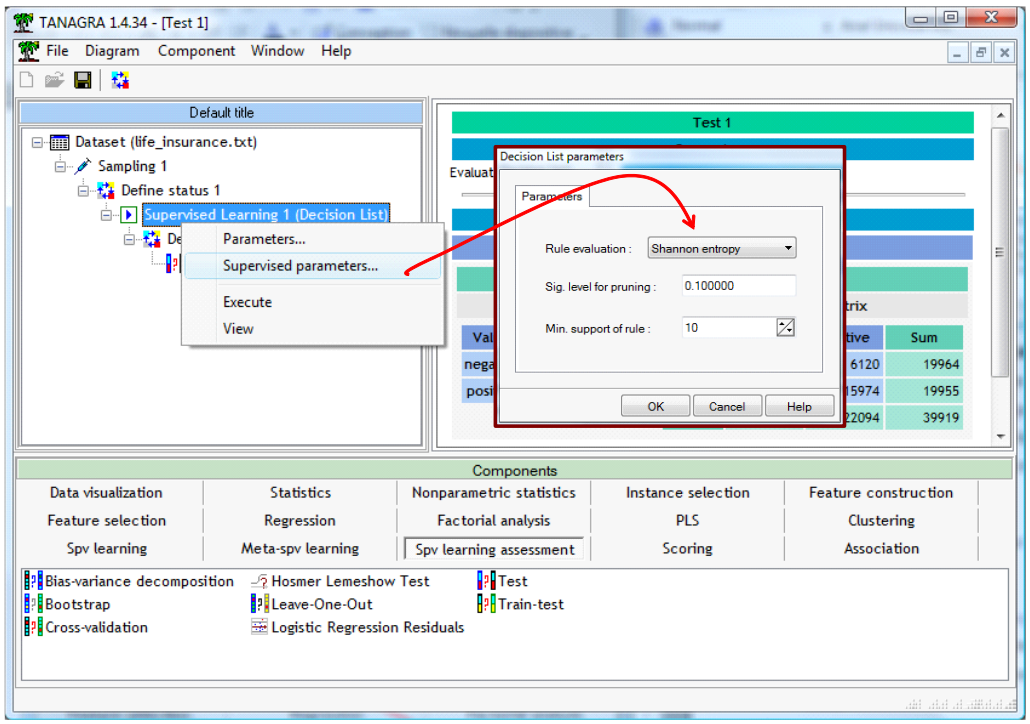


PRED_SPVINSTANCE_1 contient les prédictions du modèle sur la totalité des données. Nous insérons enfin le composant TEST (onglet SPV LEARNING ASSESSMENT). Il est automatiquement paramétré pour calculer la matrice de confusion sur l'échantillon test. Nous cliquons directement sur VIEW. Le taux d'erreur en test est de 25.3%.

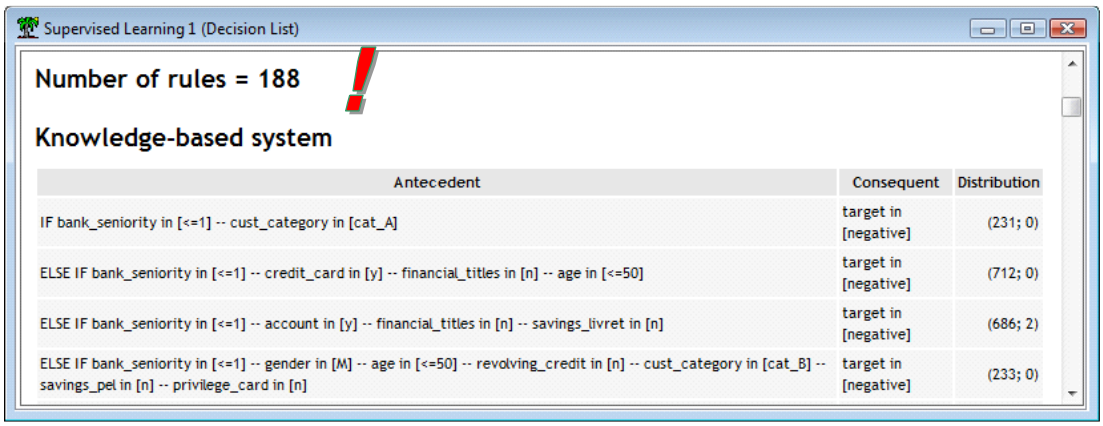


4.3. Modification des paramètres des Listes de Décision

Voyons comment se comporte le système si nous modifions la mesure d'évaluation des règles. Nous souhaitons substituer l'entropie de Shannon à la J-Measure.



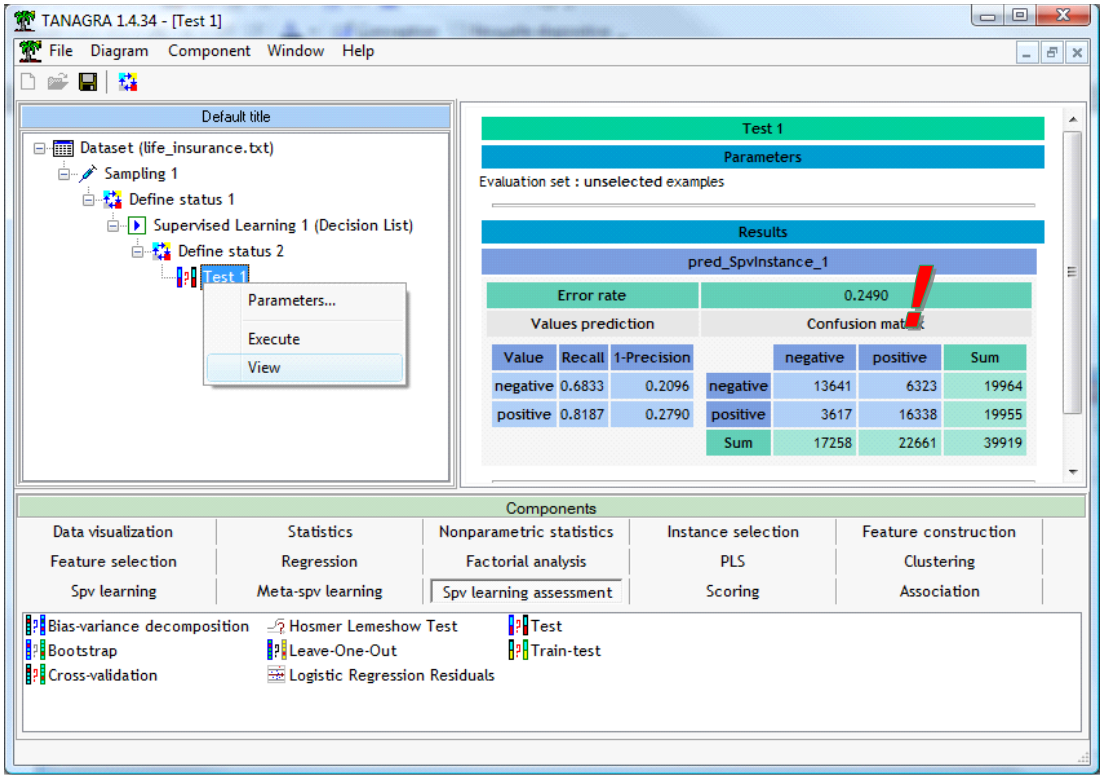
Pour ce faire, nous revenons sur le composant SUPERVISED LEARNING 1 (DECISION LIST) et nous actionnons le menu SUPERVISED PARAMETERS. Nous modifions « Rule Evaluation » en la passant à « Shannon Entropy ». Il ne reste plus qu'à cliquer sur le menu contextuel VIEW du composant.



Antecedent	Consequent	Distribution
IF bank_seniority in [<=1] -- cust_category in [cat_A]	target in [negative]	(231; 0)
ELSE IF bank_seniority in [<=1] -- credit_card in [y] -- financial_titles in [n] -- age in [<=50]	target in [negative]	(712; 0)
ELSE IF bank_seniority in [<=1] -- account in [y] -- financial_titles in [n] -- savings_livret in [n]	target in [negative]	(686; 2)
ELSE IF bank_seniority in [<=1] -- gender in [M] -- age in [<=50] -- revolving_credit in [n] -- cust_category in [cat_B] -- savings_pel in [n] -- privilege_card in [n]	target in [negative]	(233; 0)

Le temps de calcul est un peu plus long. C'est tout à fait normal, 188 règles ont été générées. Si nous observons les premières règles, nous remarquerons qu'elles sont nettement plus spécialisées, avec peu de contre-exemples. La contrepartie est que leur support est en moyenne plus faible.

Est-ce que cette précision accrue des règles se traduit par des meilleures performances du classifieur en prédiction ? Nous actionnons le menu VIEW du composant TEST. Nous constatons que le taux d'erreur en test est de 24.9%.



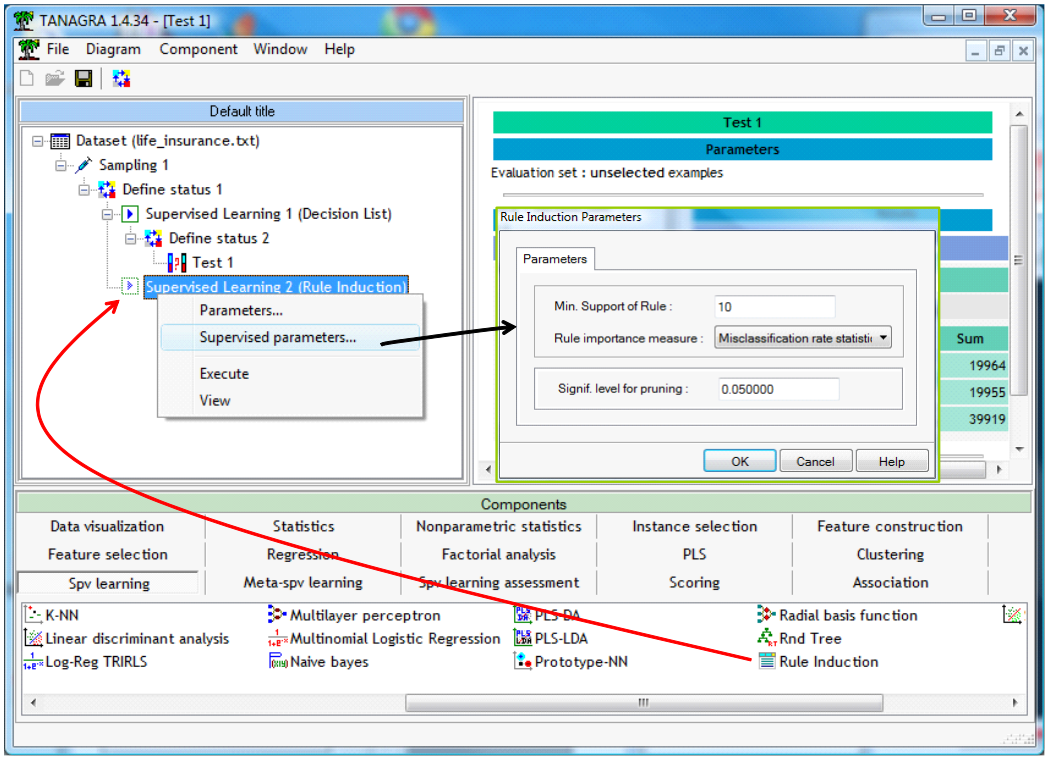
Value	Recall	1-Precision
negative	0.6833	0.2096
positive	0.8187	0.2790

	negative	positive	Sum
negative	13641	6323	19964
positive	3617	16338	19955
Sum	17258	22661	39919

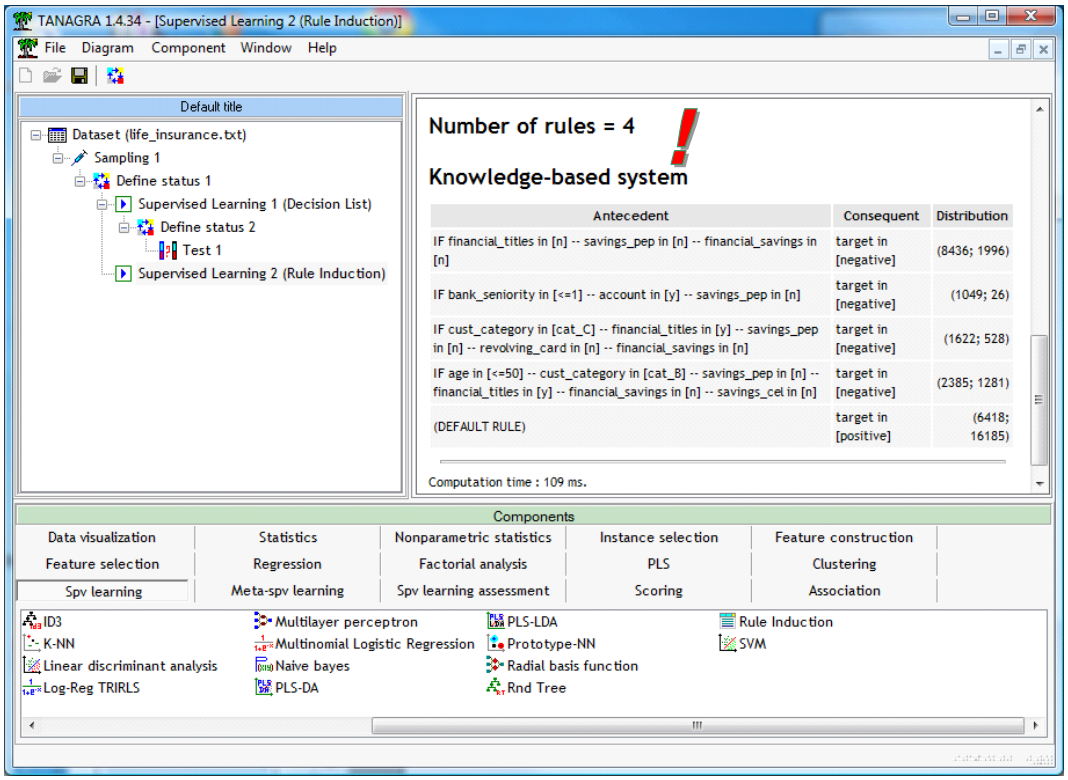
Est-ce que ces 0.4% d'écarts justifient les 168 règles supplémentaires ? Non bien évidemment. Dans notre contexte, la liste de décision avec la J-Measure semble être la plus appropriée.

4.4. Construction et évaluation des Règles prédictives indépendantes

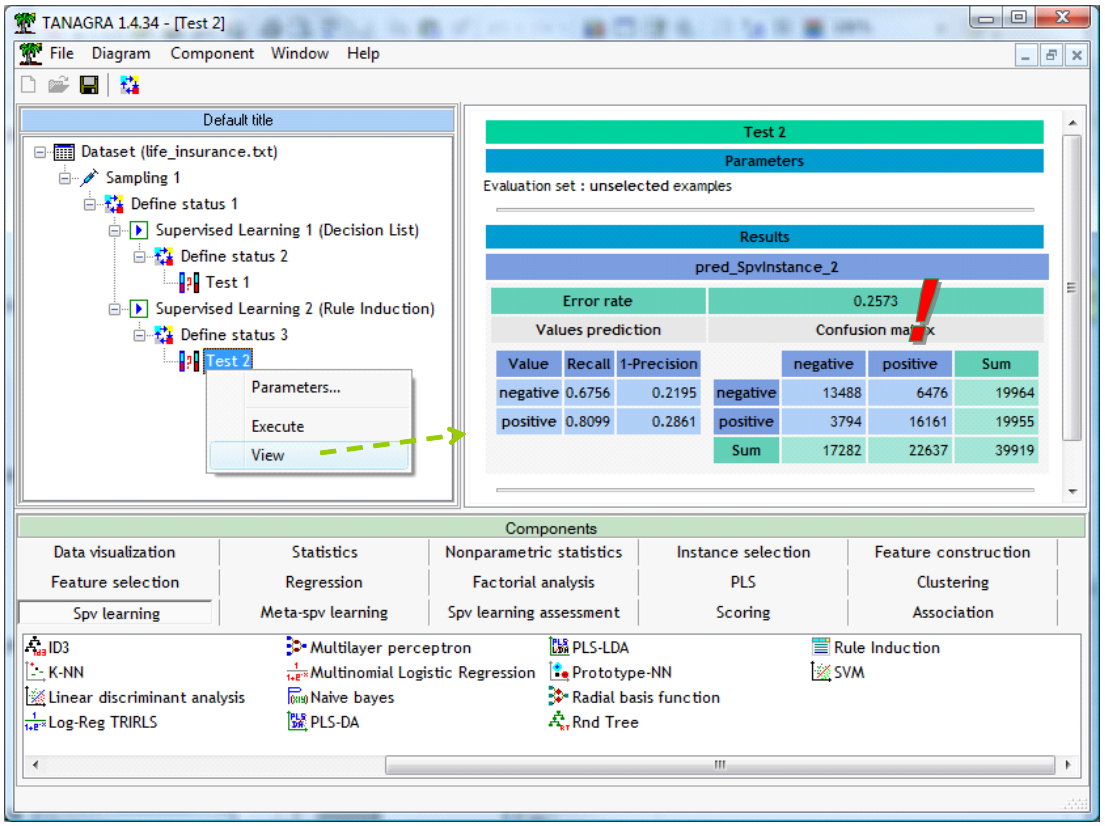
Voyons ce qu'il en est concernant l'induction des règles indépendantes. Nous insérons le composant RULE INDUCTION à la suite de DEFINE STATUS 1. Nous actionnons le menu contextuel SUPERVISED PARAMETERS pour inspecter les paramètres par défaut.



Nous n'effectuons aucune modification. Nous actionnons directement le menu VIEW.



4 règles (seulement !) sont produites (en un temps record, 109 ms.). Est-ce préjudiciable en généralisation ? Pour le savoir, nous l'évaluons sur l'échantillon test en réitérant la même séquence de composants (nous confrontons « target » et PRED_SPVINSTANCE_2 dans DEFINE STATUS 3).



Le taux d'erreur en test est de 25.7%. Il n'y avait que 4 règles, mais elles s'avèrent particulièrement pertinentes !

Remarque : Bien entendu, ceux qui le souhaitent peuvent modifier les paramètres de la méthode pour en observer les conséquences sur le nombre de règles et les performances en classement.

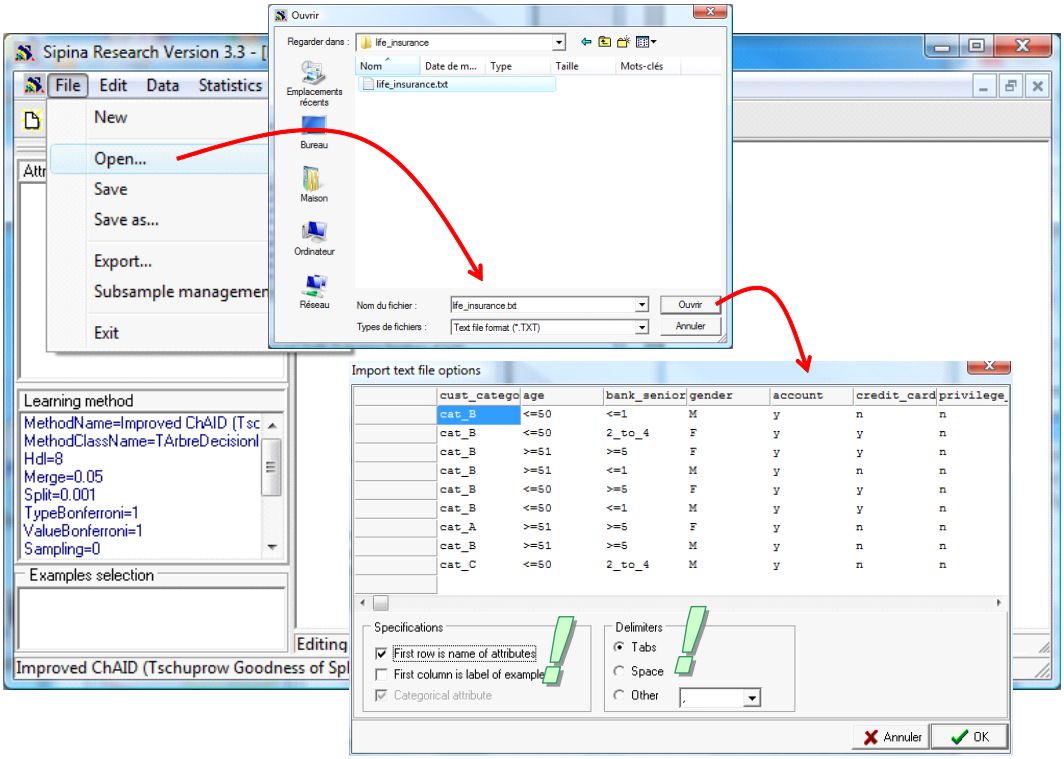
5. Traitements avec Sipina

SIPINA comporte plusieurs techniques d'induction de règles. Il a été mon terrain de jeux pendant longtemps s'agissant des implémentations plus ou moins loufoques en vue de publications. Nous y trouvons entre autres des méthodes fidèles à la transcription des auteurs de CN2 (les articles de 1989 et 1991). Je n'ai programmé que l'optimisation « hill-climbing » en revanche.

Assurez vous d'utiliser la **version 3.3** pour reproduire ce tutoriel, le numéro de version est visible dans la barre de titre de SIPINA.

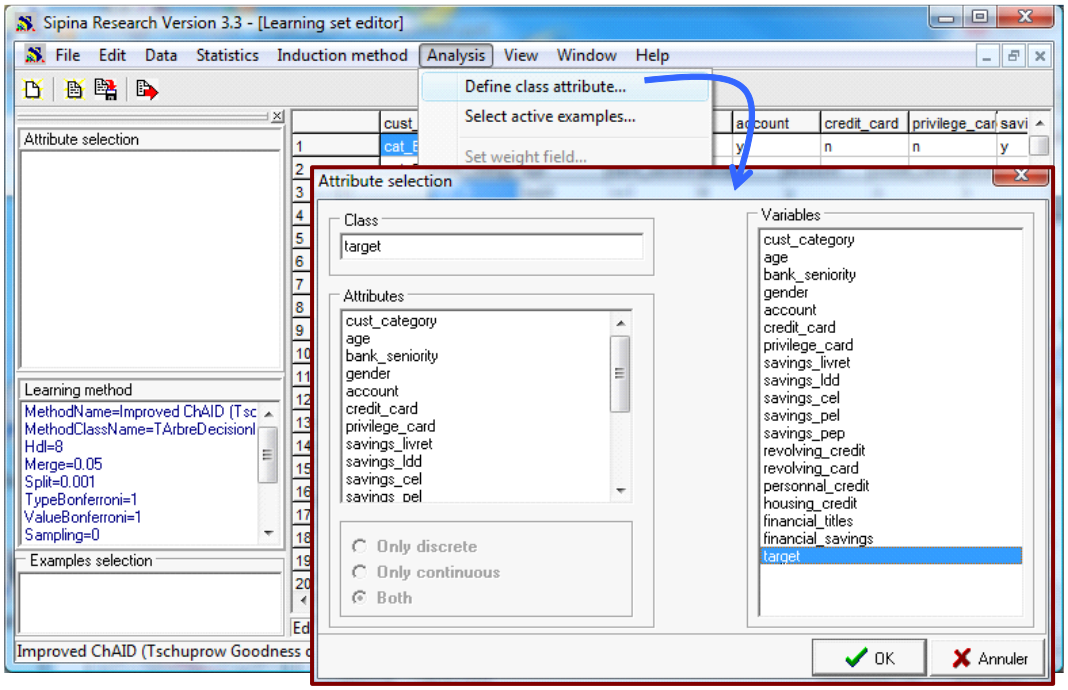
5.1. Importation des données

Après avoir démarré SIPINA, nous actionnons le menu FILE / OPEN pour ouvrir le fichier LIFE_INSURANCE.TXT. Une boîte de dialogue nous permet de paramétrer le format du fichier à importer : nous indiquons que la première ligne correspond aux noms de variables et que le séparateur est le caractère « tabulation ».

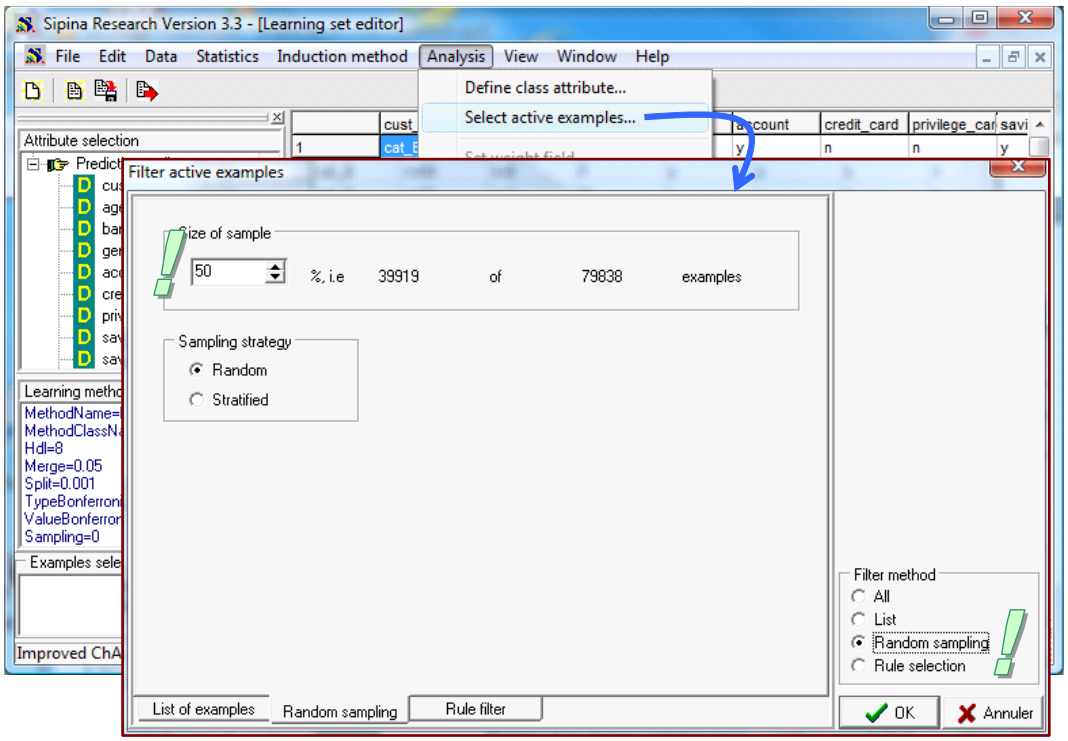


5.2. Construction et évaluation des Règles prédictives indépendantes

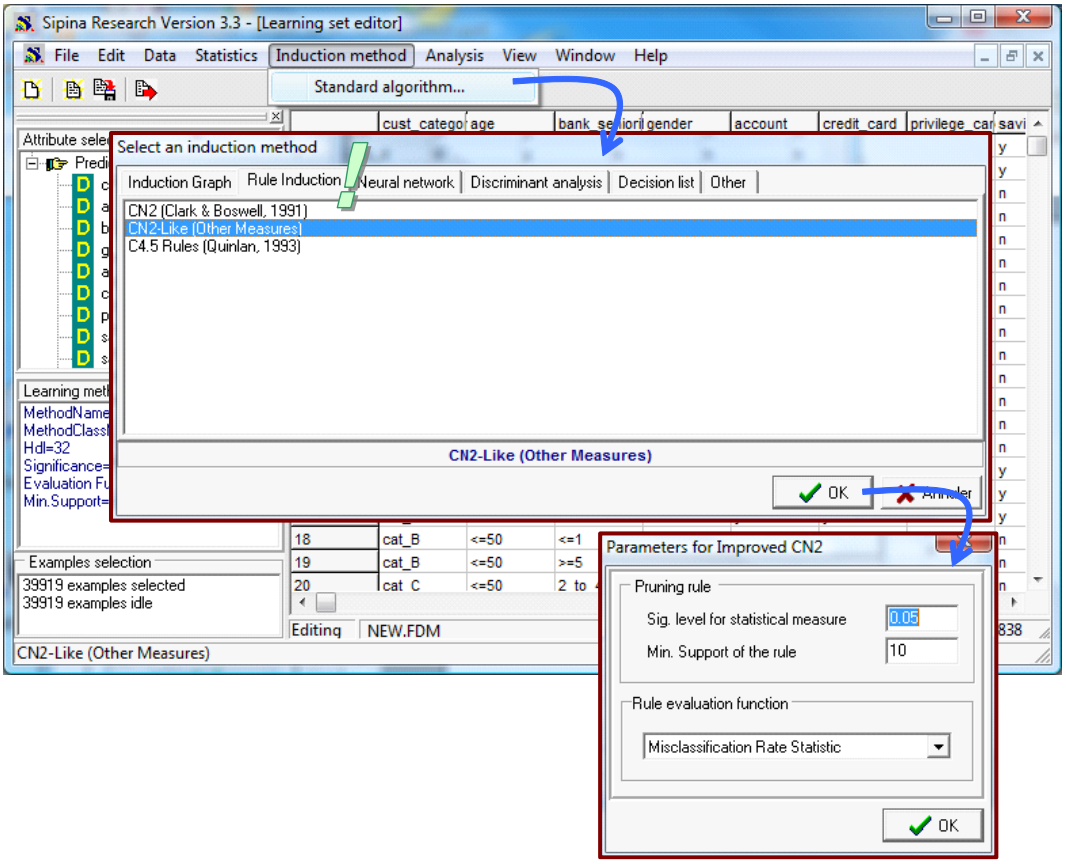
Nous devons spécifier le problème à traiter dans un premier temps c.-à-d. indiquer au logiciel la variable cible et les variables prédictives. Nous actionnons le menu ANALYSIS / DEFINE CLASS ATTRIBUTE pour cela. Nous réalisons la sélection par glisser – déposer dans les emplacements adéquats.



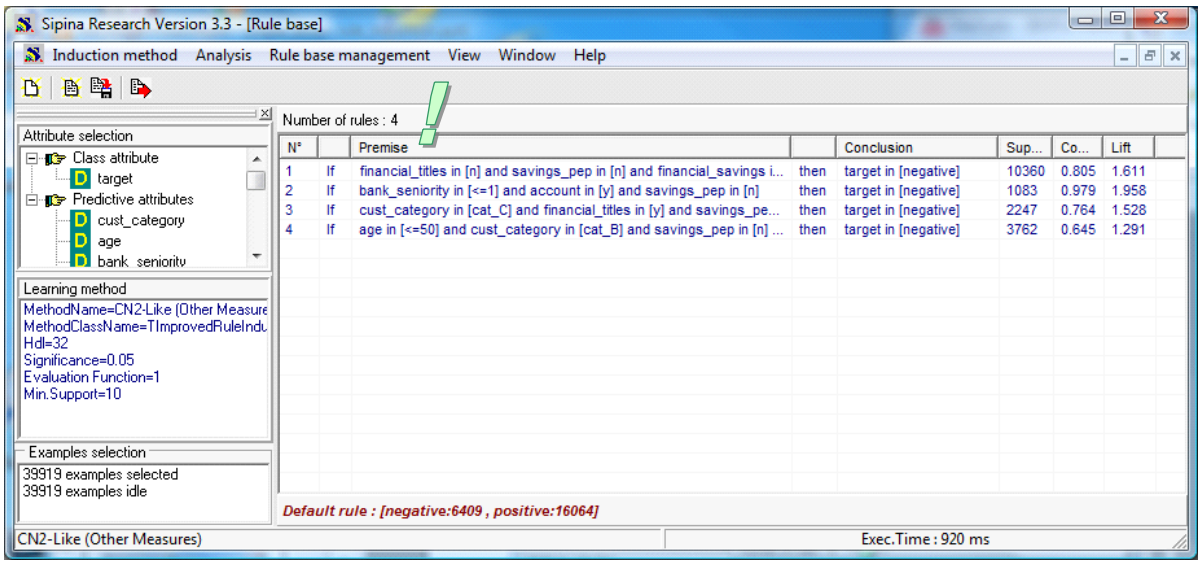
Puis nous devons partitionner les données en échantillons d'apprentissage et de test. Nous actionnons le menu ANALYSIS / SELECT ACTIVE EXAMPLES. Nous choisissons l'option RANDOM SAMPLING avec une sélection de 50% des individus disponibles.



L'étape suivante consiste à sélectionner la méthode d'apprentissage. Nous actionnons le menu INDUCTION METHOD / STANDARD ALGORITHM. Dans la boîte de dialogue qui apparaît, nous activons l'onglet RULE INDUCTION. Nous choisissons CN2 LIKE (Other measures). Une autre boîte de dialogue nous donne la possibilité de paramétrer la méthode. Nous validons la proposition.



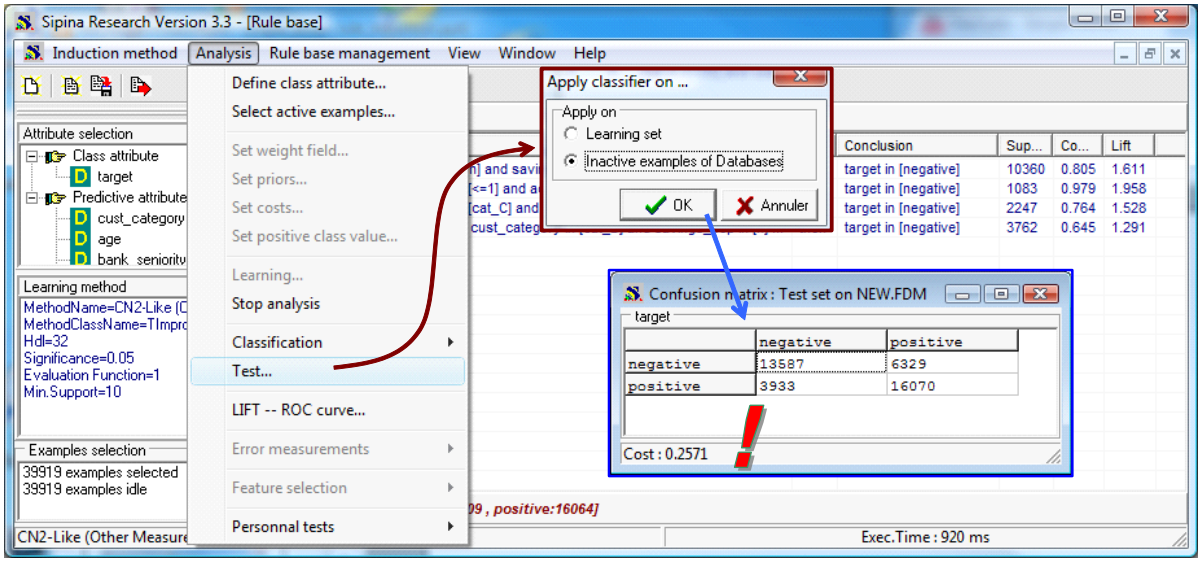
Nous pouvons lancer la construction des règles en cliquant sur ANALYSIS / LEARNING.



Nous obtenons 4 règles prédictives, les mêmes (à très peu de choses près) que celles de Tanagra[13]. Pour chaque règle, SIPINA indique le support (la couverture) absolu, la confiance et le lift. Il est possible de trier les règles selon ces critères en cliquant sur les en-têtes de colonnes.

[13] Ce n'est pas étonnant. C'est cette méthode que nous avons entièrement reprogrammée dans Tanagra avec de très légères modifications (tri des modalités de la variable cible par fréquence croissante), en élargissant l'éventail des mesures disponibles (introduction de la J-measure asymétrique), et surtout en réalisant un gros travail d'optimisation des temps de traitements (7 fois plus rapide en moyenne). Autre aspect qui influe sur les

Reste à évaluer ces règles. Nous cliquons sur le menu ANALYSIS / TEST, et nous choisissons les individus non utilisés lors de la phase d'induction.

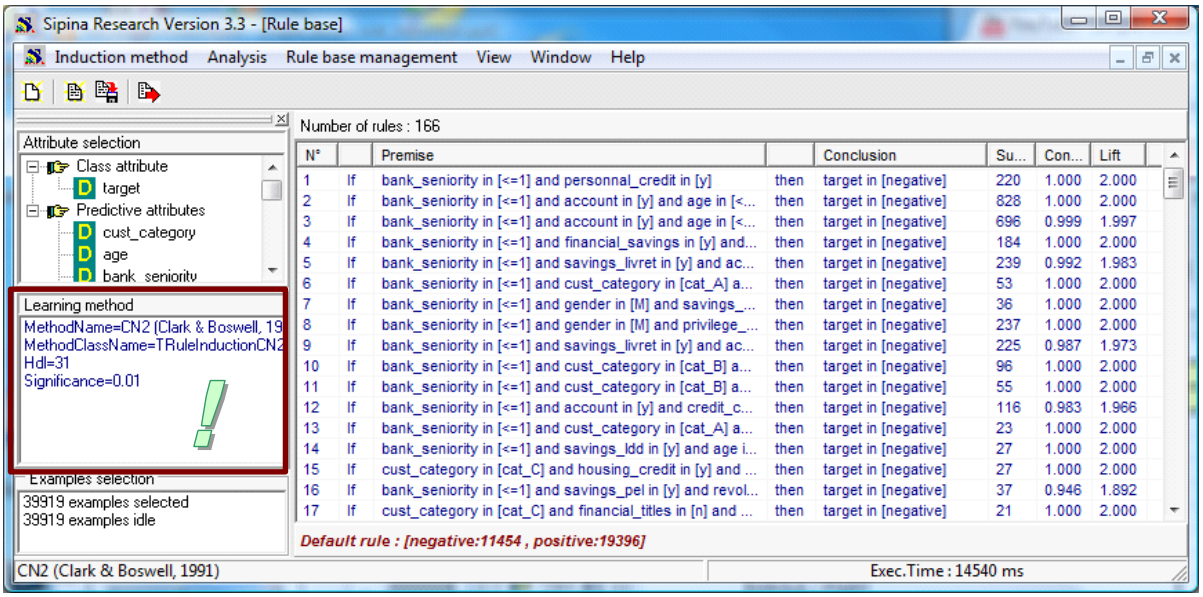


Le taux d'erreur en test est de 25.71%.

5.3. Les autres techniques d'induction de règles disponibles dans Sipina

SIPINA intègre d'autres techniques d'induction de règles. Nous pouvons par exemple étudier le comportement de l'algorithme CN2 de Clark et Boswell (1991). Après avoir clôturé l'analyse courante en cliquant sur ANALYSIS / STOP LEARNING. Nous choisissons la méthode adéquate dans la boîte de sélection (INDUCTION METHOD / STANDARD ALGORITHM → RULE INDUCTION).

Nous lançons les traitements avec ANALYSIS / LEARNING, nous obtenons 166 règles. Elles sont plus nombreuses et plus fortement spécialisées (la confiance des règles est souvent proche de 1).

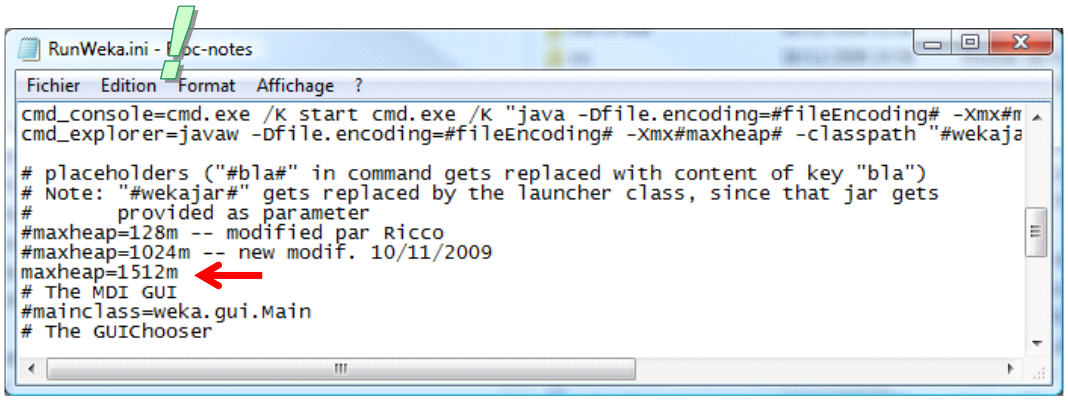


résultats, la subdivision des observations en apprentissage et test n'est pas réalisée de la même manière dans les deux logiciels.

Plus de règles, même fortement spécialisées, ne veut pas dire meilleures performances en classement du modèle global. Nous l'avions constaté précédemment. Il peut y avoir un phénomène sur-apprentissage avec une dégradation des performances en généralisation. Nous en avons l'illustration ici. Le taux d'erreur en test passe à 30.9% (ANALYSIS / TEST → INACTIVE EXAMPLES OF DATABASES).

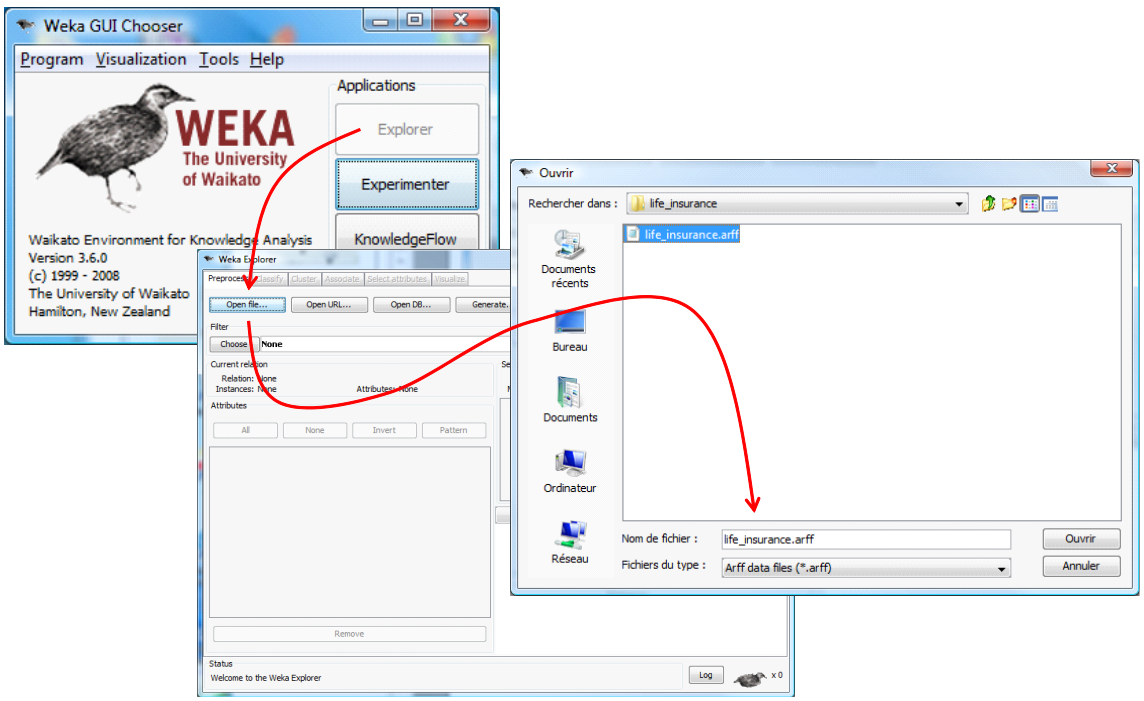
6. Traitements avec Weka

Nous utilisons la version 3.6.0 de Weka dans ce tutoriel. Nous traitons les données en mode « Explorer ». **Attention**, nous devons modifier le fichier de configuration RUNWEKA.INI pour pouvoir traiter notre fichier. Nous avons passé le paramètre « maxheap » (taille du tax) à « 1512m ».

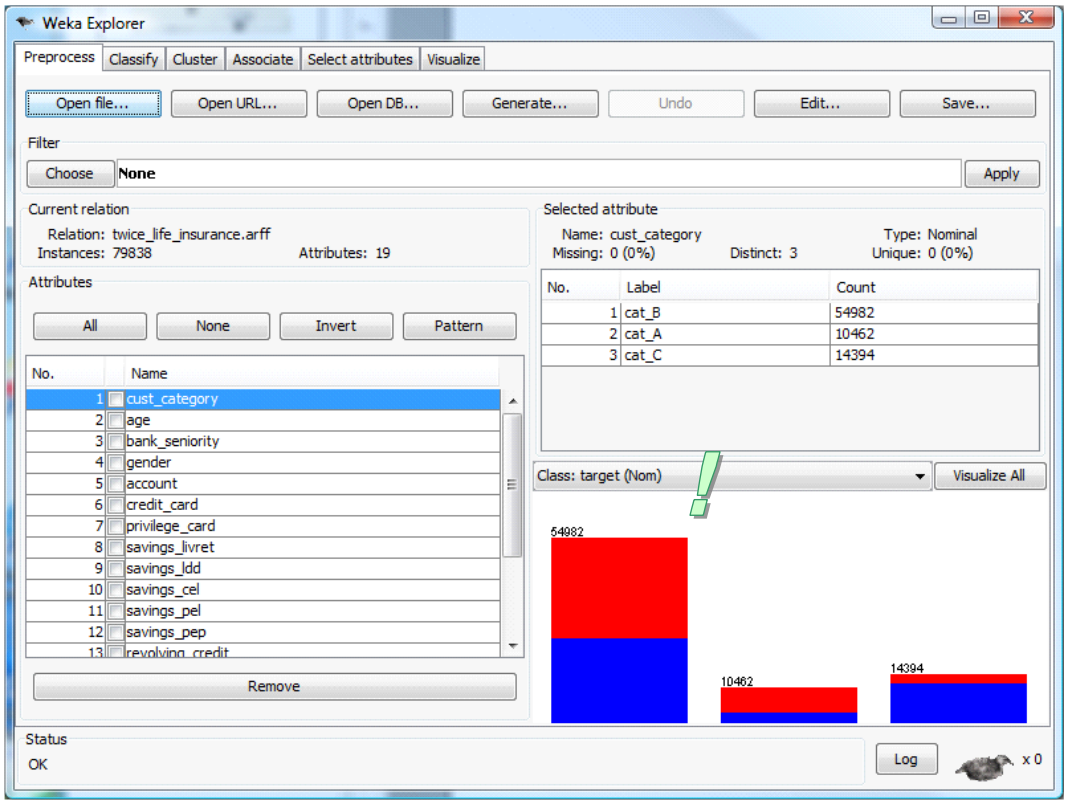


6.1. Importation des données

Weka sait traiter le format texte, mais il est bien plus simple d'utiliser le format natif ARFF. Après avoir démarré Weka et choisi le mode EXPLORER, nous chargeons le fichier en cliquant sur le bouton OPEN FILE. Nous sélectionnons le fichier LIFE_INSURANCE.ARFF



La variable cible (CLASS) est par défaut la dernière colonne. Cette configuration nous convient.



6.2. Construction et évaluation de JRIP

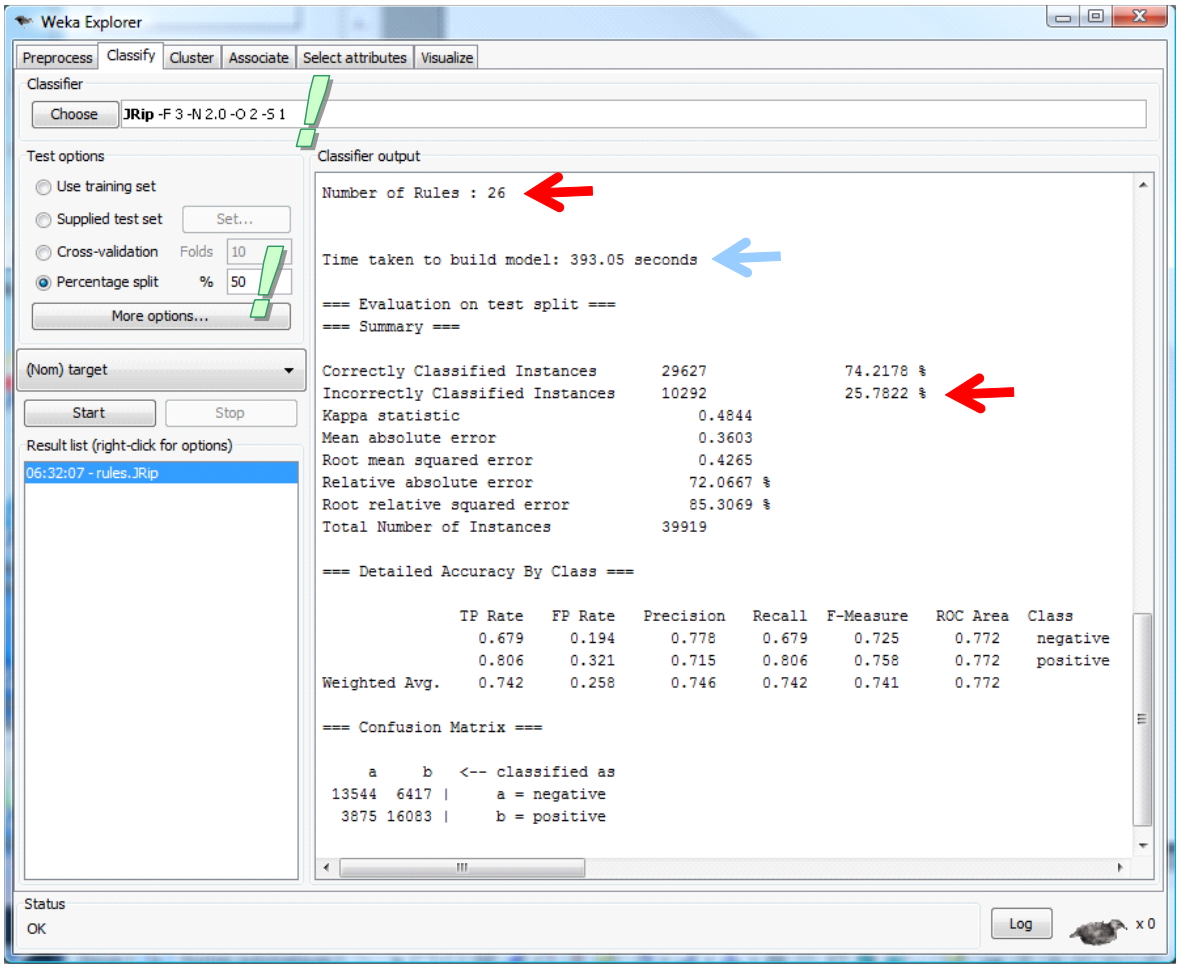
Nous passons à l'onglet CLASSIFY pour l'apprentissage supervisé. Nous indiquons que la moitié des observations sera utilisée pour l'évaluation (Test Options → Percentage Split = 50%).

Puis nous sélectionnons la méthode. Nous cliquons sur le bouton CLASSIFIER / CHOOSE, nous choisissons dans le groupe RULES la méthode JRIP qui correspond à RIPPER de Cohen (1995)[14]. Nous laissons les paramètres par défaut.

JRIP produit des règles indépendantes. Par rapport au schéma générique que nous avons décrit, il intègre une première procédure de post-élagage basée sur la description minimale des messages pour raccourcir les règles en retirant les propositions inutiles, et une seconde procédure pour réduire le nombre de règles dans la base. Il en résulte souvent un classifieur plus compact par rapport aux autres méthodes. En revanche, le temps de calcul pâtit de ces différentes optimisations, surtout lorsque l'on traite de grandes bases de données.

Nous cliquons sur le bouton START pour lancer les calculs.

[14] http://www.cs.cmu.edu/~wcohen/ ; l'auteur reconnaît du bout des lèvres l'équivalence entre son implémentation de RIPPER et JRIP, mais il ne la dément pas en tous les cas.



La méthode produit 26 règles en 393.5 secondes, le taux d'erreur en est de 25.8%

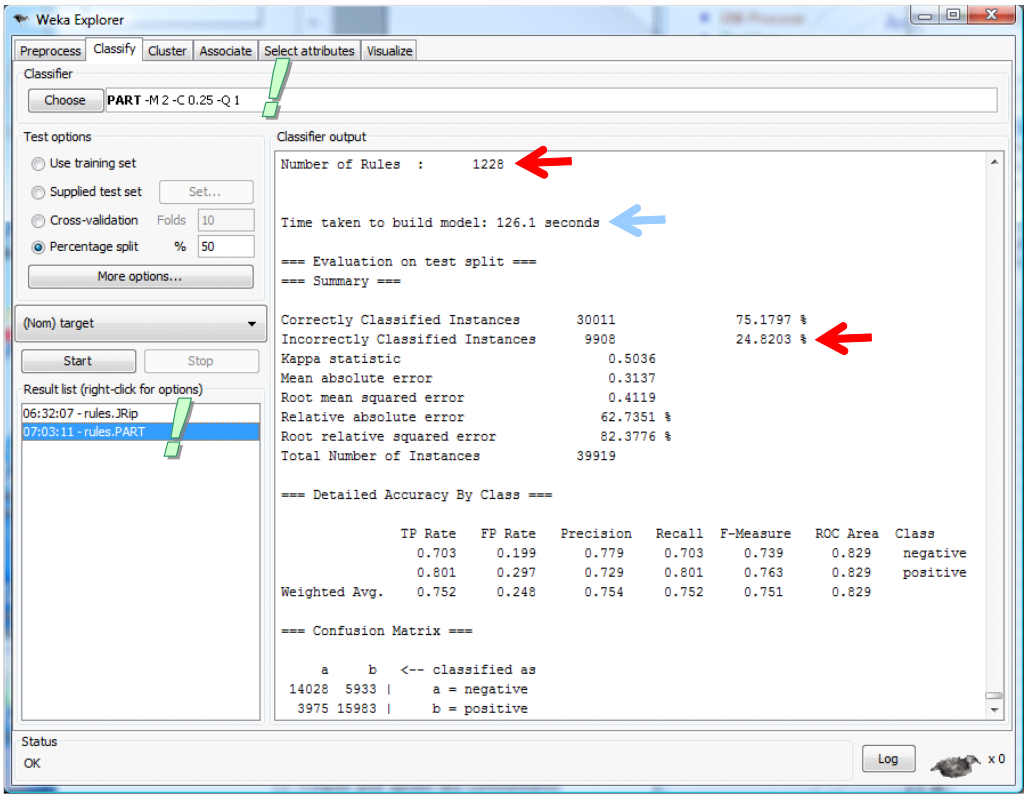
6.3. Les autres techniques d'induction de règles

La richesse de sa bibliothèque de calcul est la grande force de Weka. Il intègre toute une panoplie de méthodes introuvables par ailleurs. De plus, chacune d'entre elles correspondent à une référence publiée, et donc reconnue scientifiquement.

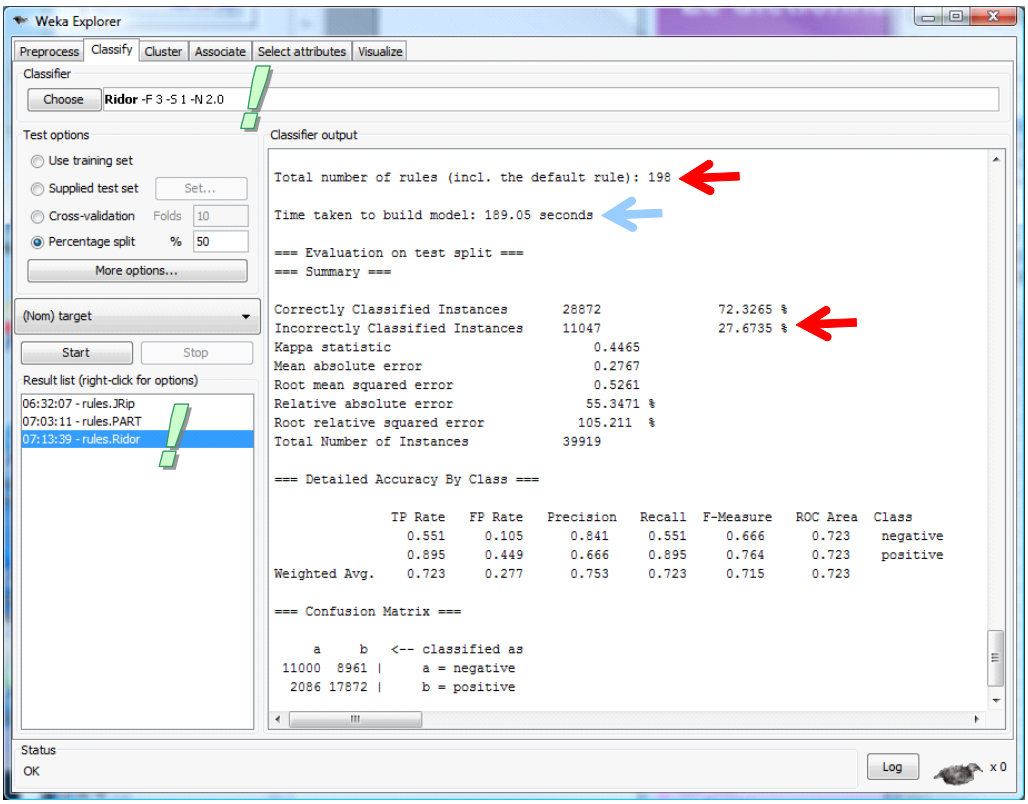
PART (Frank et Witten, « Generating Accurate Rule Sets without Global Optimization », in 15th ICML, 144-151, 1998) génère des listes de décision. Apparemment, la méthode crée un arbre de décision à chaque étape, sélectionne la branche la plus intéressante, retire les observations associées, et réitère le processus jusqu'à épuisement de la base. Nous la sélectionnons et nous cliquons de nouveau sur START.

La méthode a généré 1228 règles (!)[15], avec un taux d'erreur de 24.8% en test.

[15] L'interprétation des règles est impossible dans ce type de configuration.



RIDOR (Gaines et Compton, « Induction of Ripple Down Rules Applied to Modeling Large Databases », in Journal of Intelligent Information Systems, 5(3), 211-228, page 1995. Si je me réfère à la documentation de Weka, la méthode démarre avec la règle par défaut, puis il essaie de produire un système qui en explique les exceptions. Il poursuit jusqu'à obtention de feuilles pures.

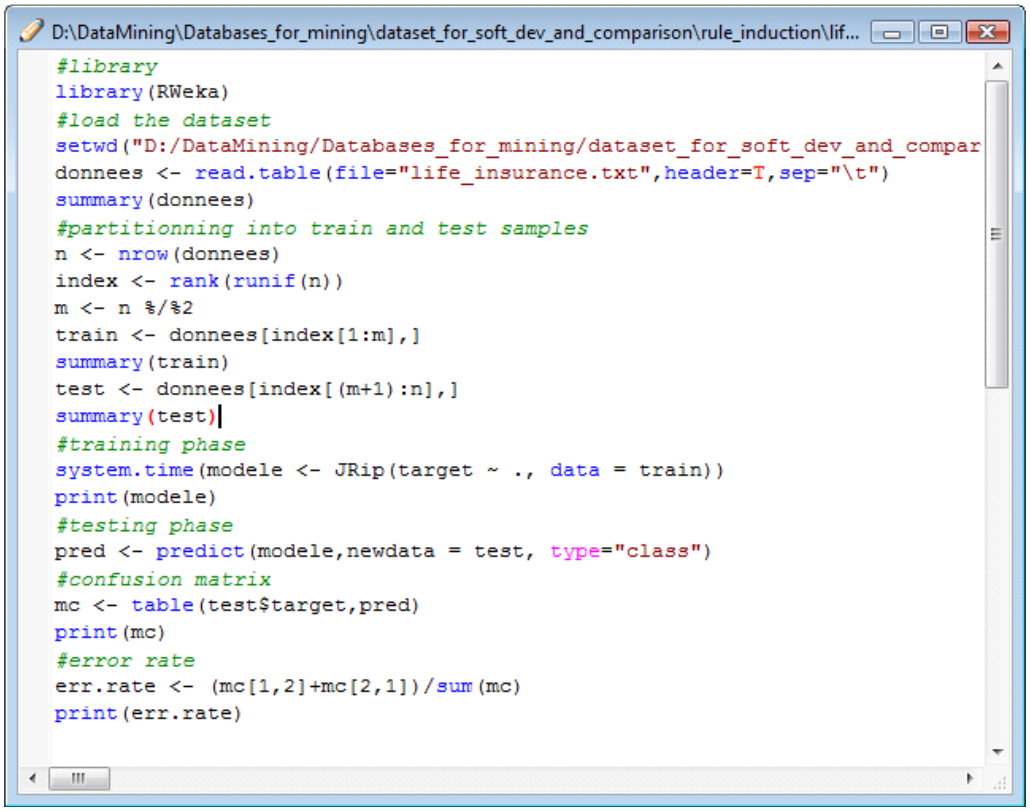


La méthode a généré 198 règles, avec un taux d'erreur de 27.7%

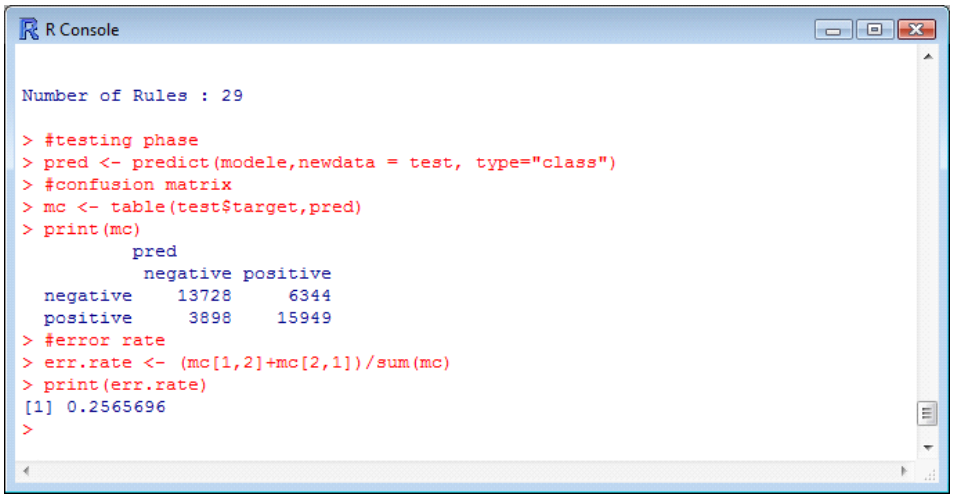
7. Traitements avec R (package RWeka)

Certains algorithmes de traitements de Weka sont accessibles dans R via le package RWeka (version 0.3-23). L'intérêt est de pouvoir bénéficier de l'environnement de R (programmation, manipulation des données, graphiques, etc.) tout en réalisant des traitements qui ne sont disponibles que dans Weka.

Après avoir démarré R, nous activons le package RWeka. Nous chargeons les données, puis nous les partitionnons en échantillons d'apprentissage et de test. Sur la partie apprentissage, nous construisons le modèle de prédiction avec JRip, nous évaluons les performances en élaborant la matrice de confusion sur la partie test.



Nous obtenons les résultats suivants.

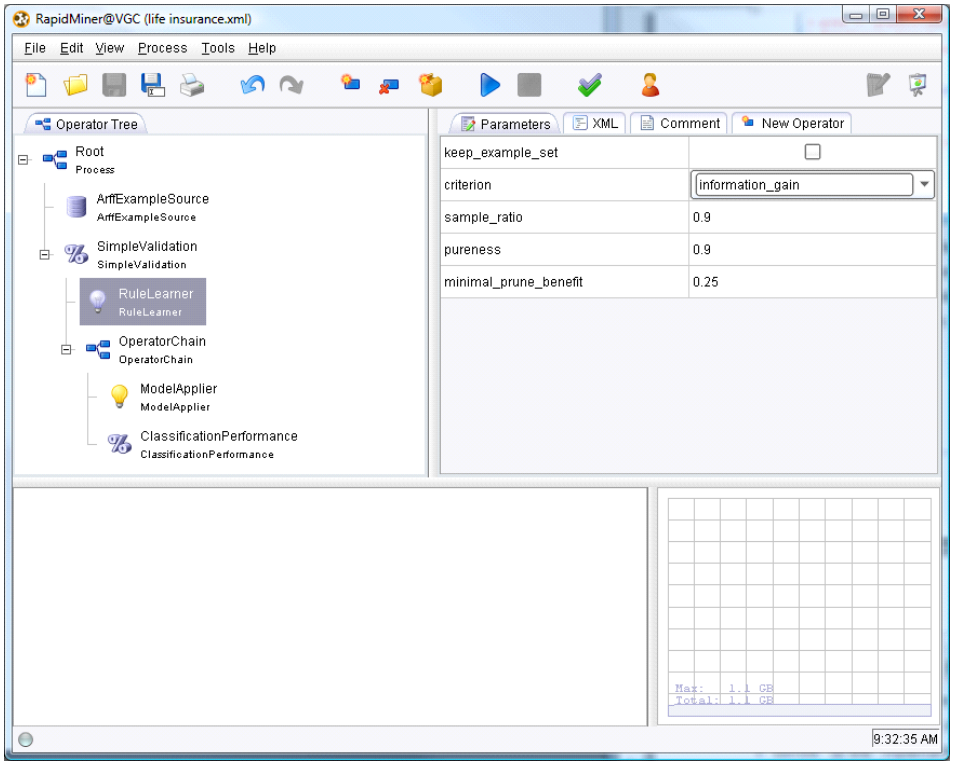


Nous avons 29 règles et un taux d'erreur en test de 25.7%. Comme le partitionnement est réalisé au hasard, il est normal que nous n'ayons pas exactement les mêmes résultats que JRip de Weka. Les différences sont négligeables. Plus étonnant en revanche, le calcul est 4 fois plus rapide dans R (89.7 secondes vs. 393.5 sec. Dans Weka). Je ne vois pas trop où est l'explication.

8. Traitements avec RapidMiner

RapidMiner propose plusieurs méthodes d'induction de règles prédictives. Il peut aussi intégrer les méthodes en provenance de Weka. Nous choisirons la méthode RULE LEARNER dans ce didacticiel, il semble qu'elle implémente un algorithme similaire à RIPPER de Cohen (1995) (dixit la documentation contextuelle).

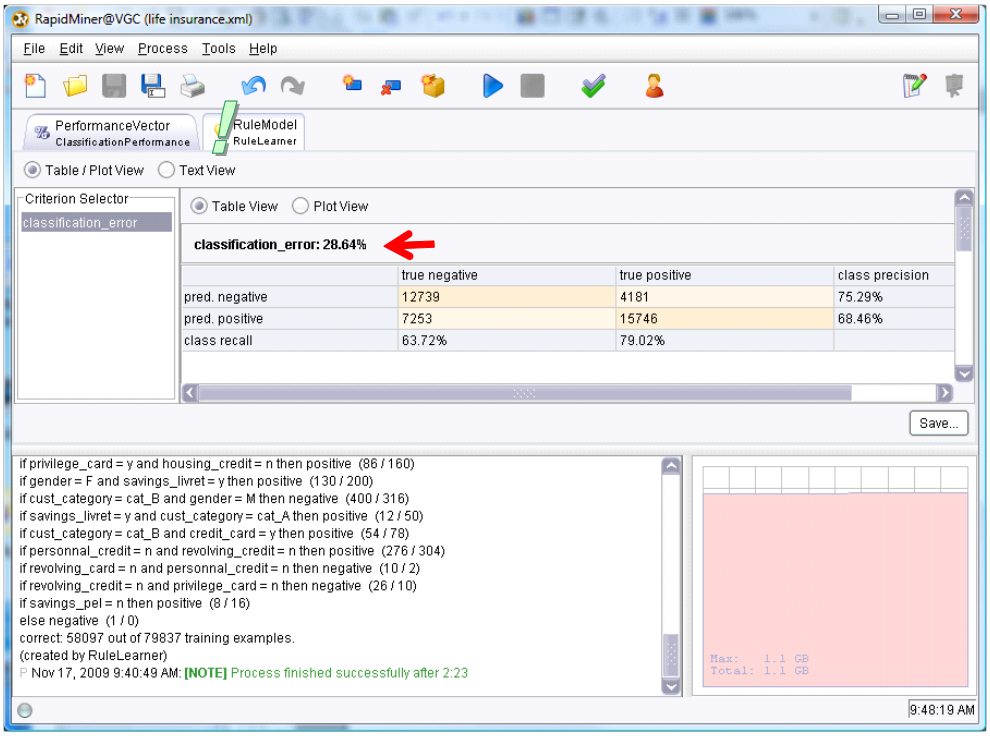
Le diagramme se présente comme suit.



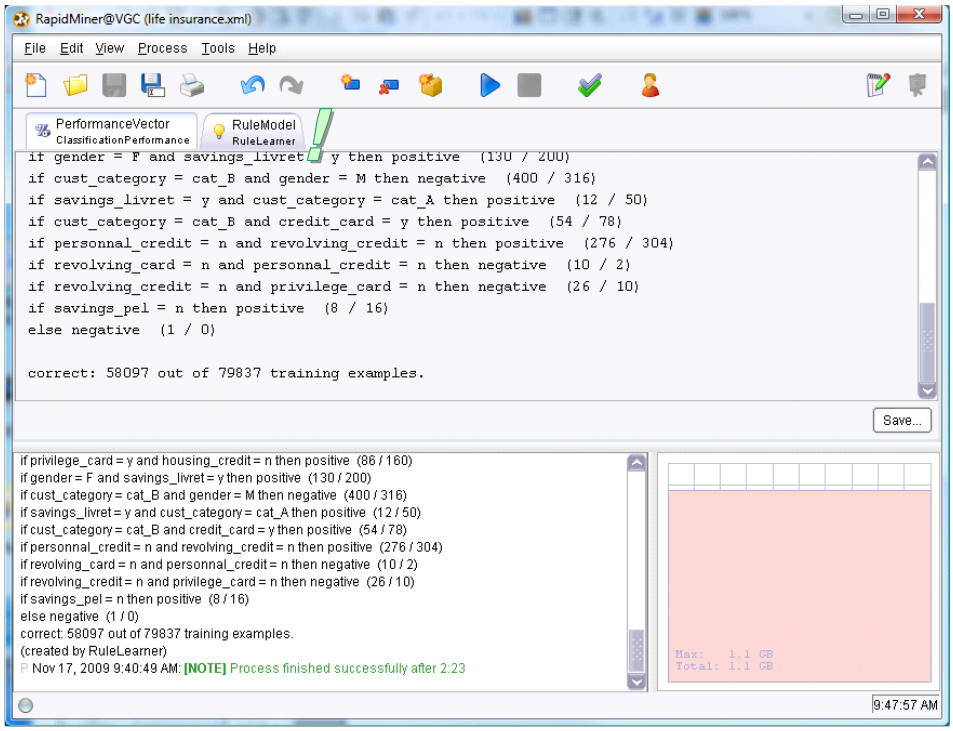
Nous décrivons les paramètres importants pour certains composants :

- ARFF EXAMPLE SOURCE charge le fichier LIFE_INSURANCE.ARFF. Nous lui indiquons la variable cible (LABEL_ATTRIBUTE = target)
- SIMPLE VALIDATION va partitionner les données, nous voulons obtenir une subdivision en 2 parts égales (SPLIT_RATIO = 0.5). Nous demandons à ce que le composant affiche le modèle de prédiction (CREATE COMPLETE MODEL est coché). Dans ce cas, il construira les règles sur la totalité des données disponibles (apprentissage et test confondus – je n'ai pas su comment faire pour faire afficher le modèle élaboré sur uniquement la partie « apprentissage »).
- RULE LEARNER est la méthode d'apprentissage, nous laissons les paramètres par défaut.
- CLASSIFICATION PERFORMANCE va construire la matrice de confusion, nous demandons l'affichage du taux d'erreur (CLASSIFICATION ERROR est coché).

Nous cliquons sur le bouton PLAY pour lancer les calculs. Nous obtenons 2 onglets. La première propose la matrice de confusion en test, avec un taux d'erreur de 28.64%.



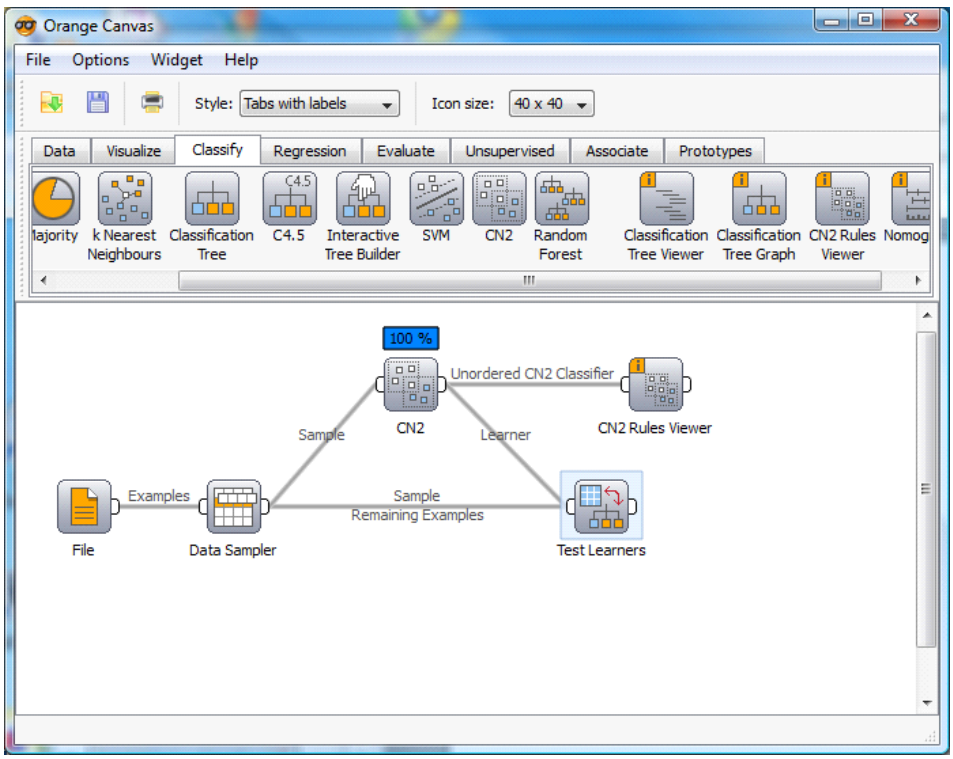
La seconde affiche la base de règles calculée sur la totalité des données. Nous en comptons 23.



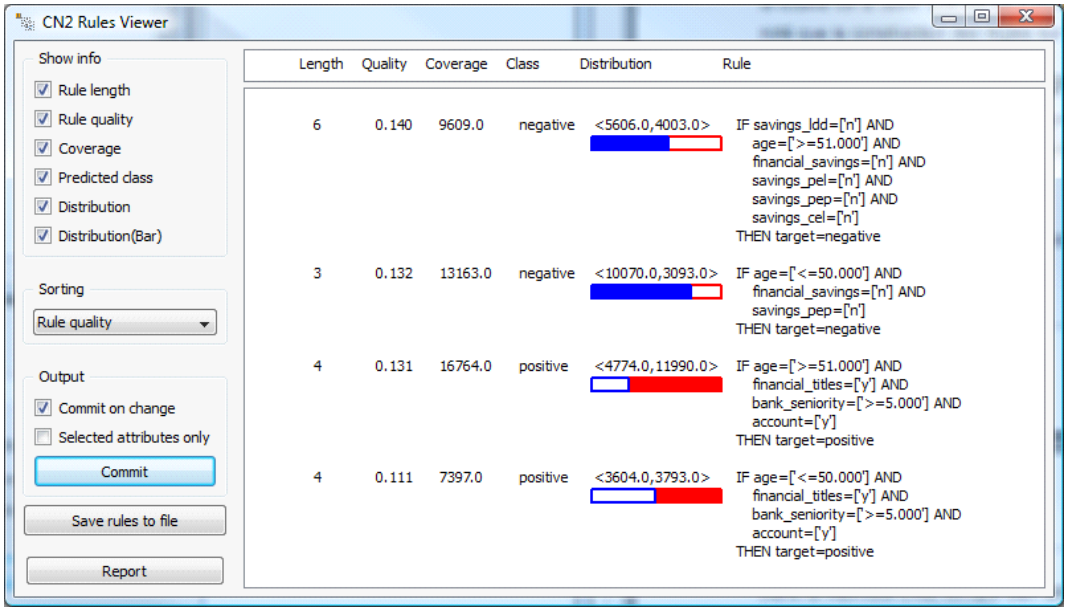
Quant au temps de calcul, RapidMiner n'affiche que la durée globale, incluant la construction des règles sur la totalité de la base. Elle est de 2.23 minutes. En suivant le déroulement des opérations, nous avons noté que la construction des règles sur la partie apprentissage est d'environ 55 secondes.

9. Traitements avec Orange

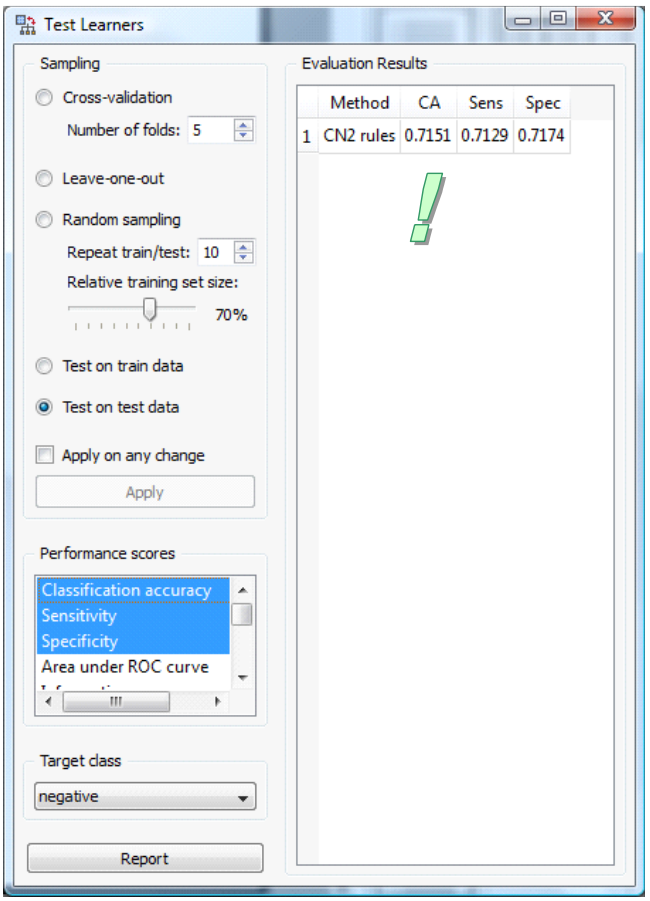
Dernier logiciel de ce comparatif, Orange implémente la méthode CN2 en la citant nommément, tout en introduisant des variantes néanmoins. Le schéma global est le suivant.



Dans DATA SAMPLER, nous demandons une partition en 2 parties égales (RANDOM SAMPLING – SAMPLE SIZE = 50%). Dans la méthode CN2, on fera bien attention de spécifier la mesure WRACC, sans cela le nombre de règles généré sera trop important. Nous pouvons les visualiser avec le composant CN2 RULES VIEWER. Le temps de calcul est de l'ordre de 30 secondes. Nous l'avons chronométré manuellement, Orange ne donne aucune indication à ce sujet.



Le taux d'erreur en test est de 28.5%.



10. Récapitulatif : Comparaison des performances

Nous récapitulons les principaux résultats dans un tableau (organisé selon les logiciels)

	Caractéristiques des règles		
Logiciel / Méthode	Nombre de règles	Taux d'erreur en test (%)	Temps de calcul (sec.)
ORANGE / CN2 (WRACC measure)	4	28.5	30.0
R / JRIP (Package Rweka)	29	25.7	89.7
RAPID MINER / RULE LEARNER	23	28.6	55.0
SIPINA / CN2 (Clark & Boswell, 1991 ; hill-climbing)	166	30.1	14.5
SIPINA / CN2 Like (Mis.Rate Stat)	4	25.7	0.9
TANAGRA / DECISION LIST (J-Measure)	20	25.3	0.2
TANAGRA / DECISION LIST (Shannon)	188	24.9	2.3
TANAGRA / RULE INDUCTION (Mis.Rate Stat)	4	25.7	0.1
WEKA / JRIP	26	25.8	393.5
WEKA / PART	1228	24.8	126.1
WEKA / RIDOR	198	27.7	189.1

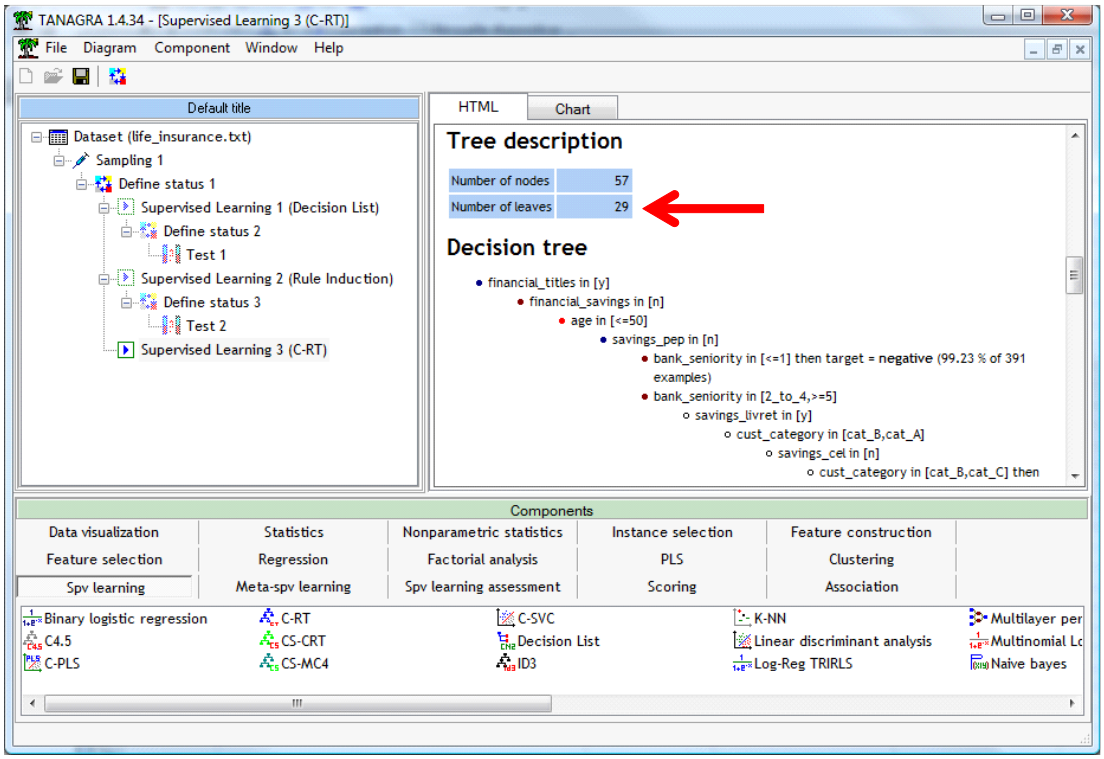
Voici quelques commentaires.

- C'est une lapalissade, mais disons-le quand même : moins on produit de règles, plus rapide seront les calculs.
- Les performances en prédiction sont du même ordre quelle que soit la méthode (autour de 27% de taux d'erreur).
- Ce n'est pas parce qu'on produit moins de règles que le classifieur sera moins bon en prédiction, au contraire. C'est un résultat que l'on retrouve souvent sur les bases de données réelles.

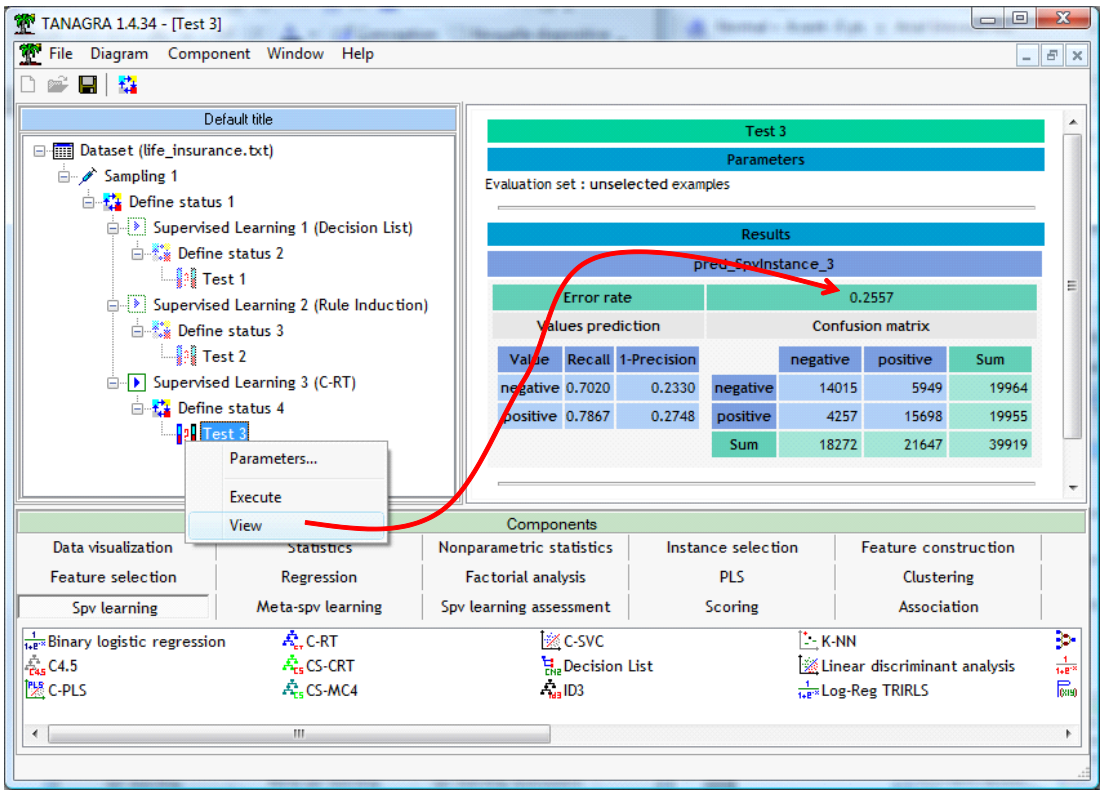
Nous avons systématiquement utilisé les valeurs par défaut des paramètres pour la totalité des algorithmes (mis à part la modification de la mesure dans certains cas). Il est évident que nous pouvons modifier grandement les performances en classement en les affinant. Après, il faut savoir dans quel sens les manipuler et avec quelle amplitude. Le problème est complexe, il faut comprendre le rôle de chaque paramètre. Pour ma part, je m'appuie sur une vision assez simple : est-ce que mes modifications vont emmener à produire des règles plus précises (confiance plus élevé), ou bien vont-elles emmener des règles plus fiables (support plus élevé). Il en résulte des modèles plus ou moins complexes, prompts à plus ou moins « coller » aux données. Le bon arbitrage peut s'appuyer sur les caractéristiques des données, les objectifs de l'étude, etc. Mais bien souvent, on se remet au tâtonnement, en réitérant les essais et évaluations.

11. Comparaison avec les arbres de décision

Bien évidemment, nous avons souhaité comparer les résultats des algorithmes d'induction de règles avec les arbres de décision. Nous avons utilisé la méthode C-RT de Tanagra. Nous obtenons l'arbre suivant au bout de 0.5 secondes.



Nous obtenons 29 règles puisque l'arbre comporte 29 feuilles, avec un taux d'erreur en test de 25.6%, assez similaire à ceux des méthodes analysées dans ce document. Ce n'est guère étonnant, les très nombreuses expérimentations menées par les chercheurs ont montré que ces deux familles d'approches proposaient des performances équivalentes dans la majorité des cas[16].



[16] Voir aussi http://tutoriels-data-mining.blogspot.com/2008/03/listes-de-dcision-vs-arbres-de-dcision.html

12. Conclusion

Dans ce didacticiel, nous avons voulu mettre en avant les techniques d'induction de règles prédictives. Peu connues du grand public, peu visibles dans les logiciels commerciaux, elles sont heureusement bien représentées dans les logiciels orientés « recherche » de la communauté « machine learning ». Nous constatons qu'elles constituent une alternative tout à fait crédible aux arbres de décision et aux règles d'association prédictives, tant en termes de performances en classement, qu'en termes de temps de traitement.