1 Objectif

Intégration des coûts de mauvais classement dans la construction et l'évaluation des modèles en apprentissage supervisé. Comparaison de Tanagra, R et Weka.

Tout le monde s'accorde à dire que l'intégration des coûts de mauvais classement est un aspect incontournable de la pratique du Data Mining. Diagnostiquer une maladie chez un patient sain ne produit pas les mêmes conséquences que de prédire la bonne santé chez un individu malade. Dans le premier cas, le patient sera soigné à tort, ou peut être demandera-t-on des analyses supplémentaires superflues ; dans le second cas, il ne sera pas soigné, au risque de voir son état se détériorer de manière irrémédiable. Pourtant, malgré son importance, le sujet est peu abordé, tant du point de vue théorique c.-à-d. comment intégrer les coûts dans l'évaluation des modèles (facile) et dans leur construction (un peu moins facile), que du point de vue pratique c.-à-d. comment les mettre en œuvre dans les logiciels.

Une matrice de coûts de mauvais classement se présente sous la forme d'une matrice c(i,j) avec, en ligne les valeurs observées de la variable à prédire, en colonne les valeurs prédites par les modèles. Usuellement, nous avons $c(i,j) \geq 0$ si $i \neq j$, mal classer induit un coût ; et $c(i,i) = 0$, bien classer ne coûte rien. Mais cette première écriture est un peu restrictive. En réalité, bien classer entraîne souvent un gain, soit un coût négatif, nous écrirons plutôt $c(i,i) \leq 0$. Quantifier les conséquences d'un bon ou mauvais classement appartient aux experts du domaine. Il n'est pas question pour nous data miner de s'immiscer dans cette phase. En revanche, nous devons la prendre en compte lors du processus d'extraction de connaissances.

La prise en compte des coûts lors de l'évaluation ne pose pas de problèmes particuliers. Il s'agit de faire le produit terme à terme entre la matrice de coût et la matrice de confusion. Nous obtenons ainsi un « coût moyen de mauvais classement » (ou d'un gain moyen si nous multiplions le résultat par -1). Son interprétation n'est pas très aisée. Il vaut surtout pour comparer des modèles concurrents.

La prise en compte des coûts lors de l'élaboration du modèle de classement est moins connue. Plusieurs approches sont possibles. La première, la plus simple, consiste à faire comme si de rien n'était, c.-à-d. élaborer la fonction de classement sans tenir compte des coûts. Il est peu probable que l'on ait de bons résultats. Mais cette configuration peut servir de référence pour l'appréciation des autres stratégies. La seconde est aussi très simple, mais déjà efficace. Il s'agit toujours d'élaborer un modèle sans tenir compte des coûts, mais d'utiliser une règle d'affectation qui minimise le coût moyen lors du classement de nouveaux individus. Concrètement, on s'appuie sur les probabilités conditionnelles fournies par le modèle pour calculer la perte associée à chaque décision. On choisit alors la décision qui minimise la perte espérée. C'est une généralisation de la règle de classement classique qui consiste à minimiser l'erreur de classement. Le principal intérêt de

cette correction par les coûts est que nous pouvons exploiter, sans modifications spécifiques, les résultats fournis par les logiciels courants.

Après, d'autres techniques, plus ou moins sophistiquées, sont souvent décrites dans la littérature. Nombre d'entre eux reposent sur des schémas d'agrégation de modèles, basés sur des ré échantillonnages plus ou moins adaptatifs (bagging ou boosting). Même si elles sont pour la plupart performantes, elles présentent un inconvénient majeur : nous disposons d'une série de modèles, l'interprétation des résultats devient difficile, voire impossible.

Notons néanmoins une approche très intéressante qui cherche à combiner les avantages de l'agrégation de classifieurs et de la production finale d'un modèle unique. Nous utilisons le ré échantillonnage répété pour ré étiqueter la variable à prédire. Puis nous construisons le modèle final sur cette nouvelle variable. C'est l'idée de la méthode MetaCost de Domingos (1999). Nous avons intégré une variante appelée MultiCost dans Tanagra. A la différence de MetaCost, elle tient compte des coûts dès la construction des modèles individuels.

Si les techniques existent, qu'en est-il de leur implémentation dans les logiciels libres ? Après investigations, on se rend compte que les logiciels qui les intègrent de manière naturelle sont très peu nombreux. Une grande majorité ignore totalement le problème. Certains l'intègrent de manière parcellaire. RapidMiner, par exemple, ne traite que les problèmes à 2 classes avec c(i,i) = 0. Au final, il semble que Weka soit l'un des rares à proposer des outils faciles à manipuler pour l'intégration des coûts. Encore faut-il savoir les utiliser bien évidemment. Ce constat nous a amené à introduire de nouveaux composants destinés à la prise en compte des coûts en apprentissage supervisé dans la version 1.4.29 de Tanagra.

4 types d'outils sont disponibles : (1) une correction des règles d'affectation par la matrice de coût de mauvais classement ; (2) une méthode d'agrégation combinant des modèles corrigés (Bagging) ; (3) la méthode MultiCost, variante de MetaCost disponible dans Weka ; (4) et enfin, des techniques d'induction qui tiennent compte explicitement des coûts lors de l'élaboration du modèle de prédiction. Cette dernière piste est très intéressante. Il faut bien entendu que la méthode s'y prête. C'est le cas des arbres de décision lors de la phase de post élagage. Nous avons ainsi programmé deux nouveaux composants : la méthode CART sensible aux coûts telle qu'elle est décrite dans l'ouvrage de Breiman et al. (1984) ; et une technique dérivée de C4.5 que nous avions pu éprouver dans une étude réelle il fut un temps (Chauchat & Rakotomalala, 2001 ; voir http://sipina.over-blog.fr/article-18203843.html). Une variante de la méthode est par ailleurs disponible dans le logiciel SIPINA (voir http://tutoriels-data-mining.blogspot.com/2008/03/apprentissage-test-avec-sipina.html ou http://sipina.over-blog.fr/article-17592319.html pour le didacticiel associé).

Dans ce document, nous montrons la mise en œuvre de ces composants de Tanagra sur un problème réel (réaliste). Nous avons également programmé ces mêmes procédures dans le logiciel R (http://www.r-project.org/) pour donner une meilleure visibilité sur ce qui est implémenté. Nous

comparerons nos résultats avec ceux de Weka. L'algorithme sous-jacent à toutes nos analyses sera un arbre de décision. Selon les logiciels, nous utiliserons C4.5, CART ou J48.

2 Données

Le problème à résoudre consiste à attribuer à bon escient des coupons de ristourne à des clients d'une enseigne d'un magasin. Il y a 3 types de coupons (A – coupon de rabais, B – coupon spécial, N – pas de coupon). L'entreprise est gagnante lorsqu'elle attribue à un client un coupon qu'il utilise réellement. Le coût d'une bonne ou mauvaise attribution d'un coupon peut être résumé par la matrice suivante :

		Attribution		
		A	B	N
Observé (utilisation)	A	-3	1	0
	B	1	-6	0
	N	1	1	0

En d'autres termes, si l'on note la matrice de confusion comme suit

		Attribution		
		A	B	N
Observé (utilisation)	A	AA	AB	AN
	B	BA	BB	BN
	N	NA	NB	NN

L'objectif de l'apprentissage est de maximiser la fonction de profit ou de gain (ou de minimiser le COUT = -1 * PROFIT)

PROFIT = 3 * AA + 6 * BB – 1 * (NA + NB + BA + AB)

Ce problème a été présenté lors du concours DATA MINING CUP 2007 (http://www.data-mining-cup.com/2007//Wettbewerb/Aufgabe/1230970673/). Les participants disposaient d'un fichier étiqueté de 50.000 observations. Ils devaient classer 50.000 autres individus pour lesquels les descripteurs étaient fournis mais pas la variable à prédire. Ils devaient compléter cette dernière colonne et la renvoyer aux organisateurs qui pouvaient alors confronter la prédiction avec la vraie étiquette. Etait vainqueur celui qui maximisait la fonction de profit sur ce second fichier.

Nous sommes dans une configuration différente dans ce didacticiel. Nos données sont bien subdivisées en 2 parties, sauf que le second ensemble - qui est maintenant étiqueté puisque le concours est terminé - servira en réalité de fichier test.

Dans notre fichier global comportant 100.000 observations, nous avons inséré une colonne supplémentaire ID avec 2 valeurs « TRAIN » et « TEST ». Pour que nos résultats soient comparables avec ceux obtenus par les impétrants du concours, nous éviterons l'acharnement sur le fichier test.

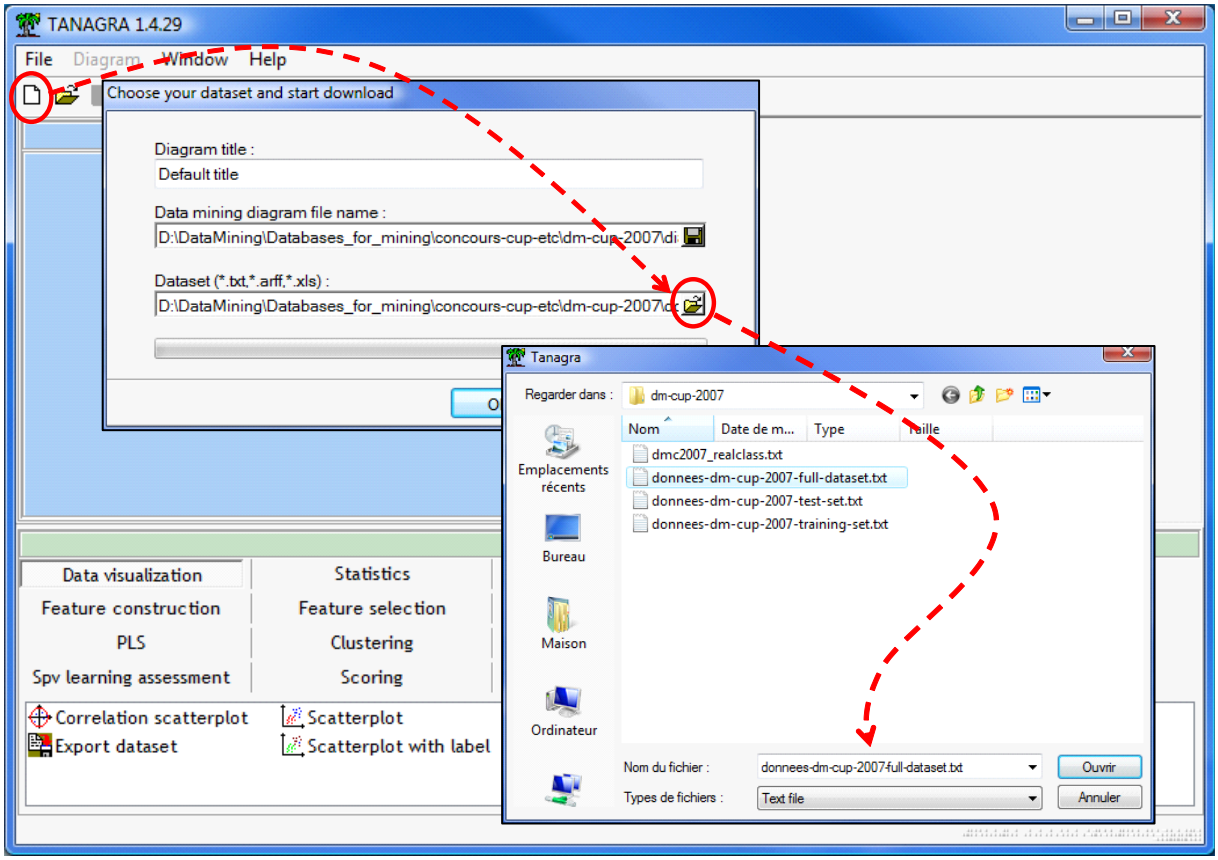
De fait, nous nous en tiendrons, dans la très grande majorité des cas et sauf justification théorique, aux paramètres par défaut des logiciels.

Les fichiers relatifs à ce didacticiel sont regroupés dans l'archive accessible en ligne : http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/dataset-dm-cup-2007.zip. Outre le fichier fusionné au format texte, nous y avons adjoint le fichier scindé en 2 parties au format WEKA (ARFF) et le code source du script pour R.

3 Analyse avec Tanagra

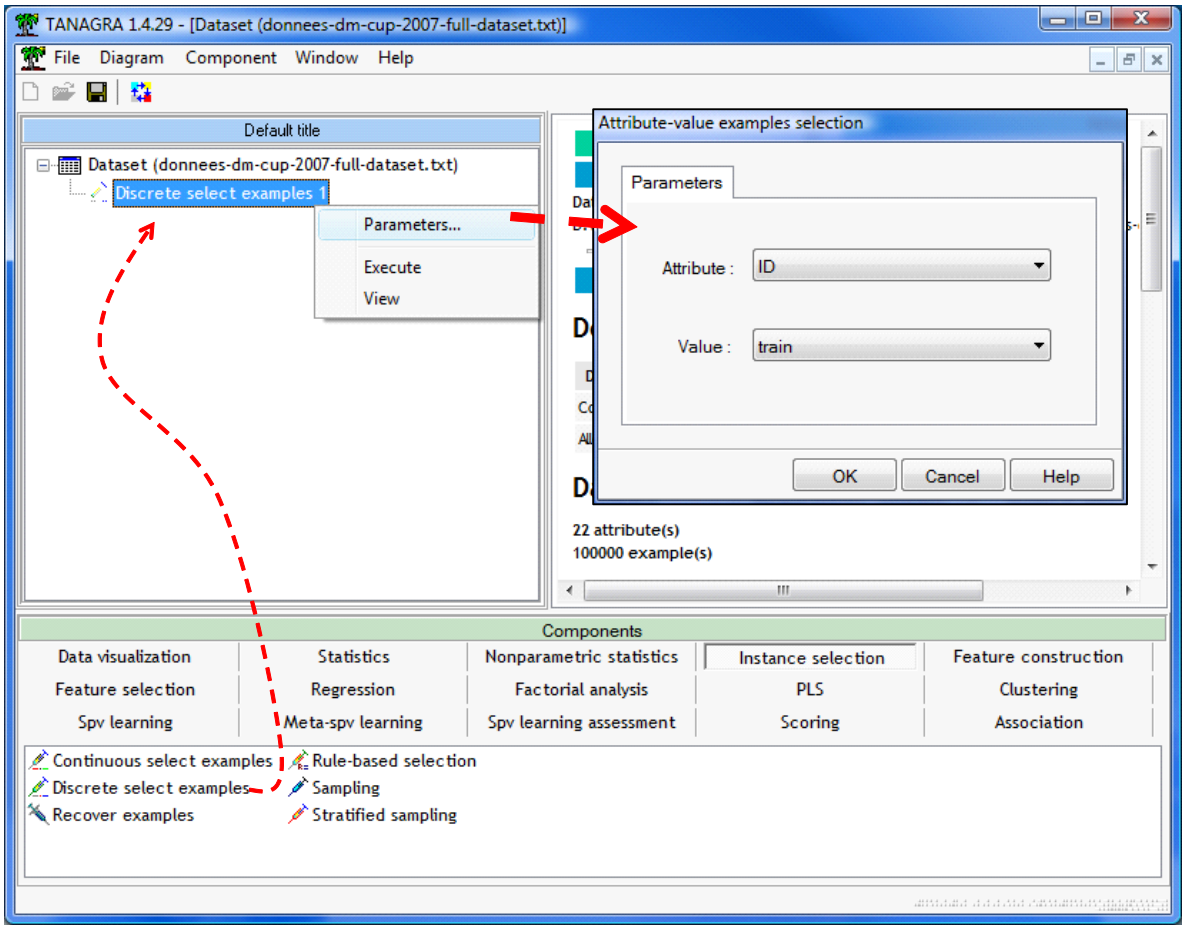
3.1 Importer les données et créer un diagramme

Nous souhaitons créer un diagramme et charger la totalité des données. Dans Tanagra, nous activons le menu FILE / NEW, nous sélectionnons le fichier « donnees-dm-cup-2007-full-dataset.txt ».

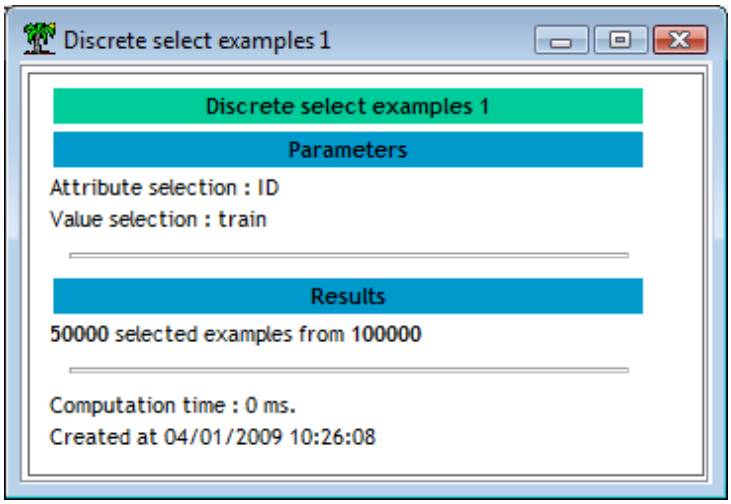


3.2 Partition en « données apprentissage » et « données test »

Nous exploitons la colonne ID pour partitionner les données. Nous insérons pour cela le composant DISCRETE SELECT EXAMPLES (onglet INSTANCE SELECTION). Nous le paramétrons de la manière suivante.

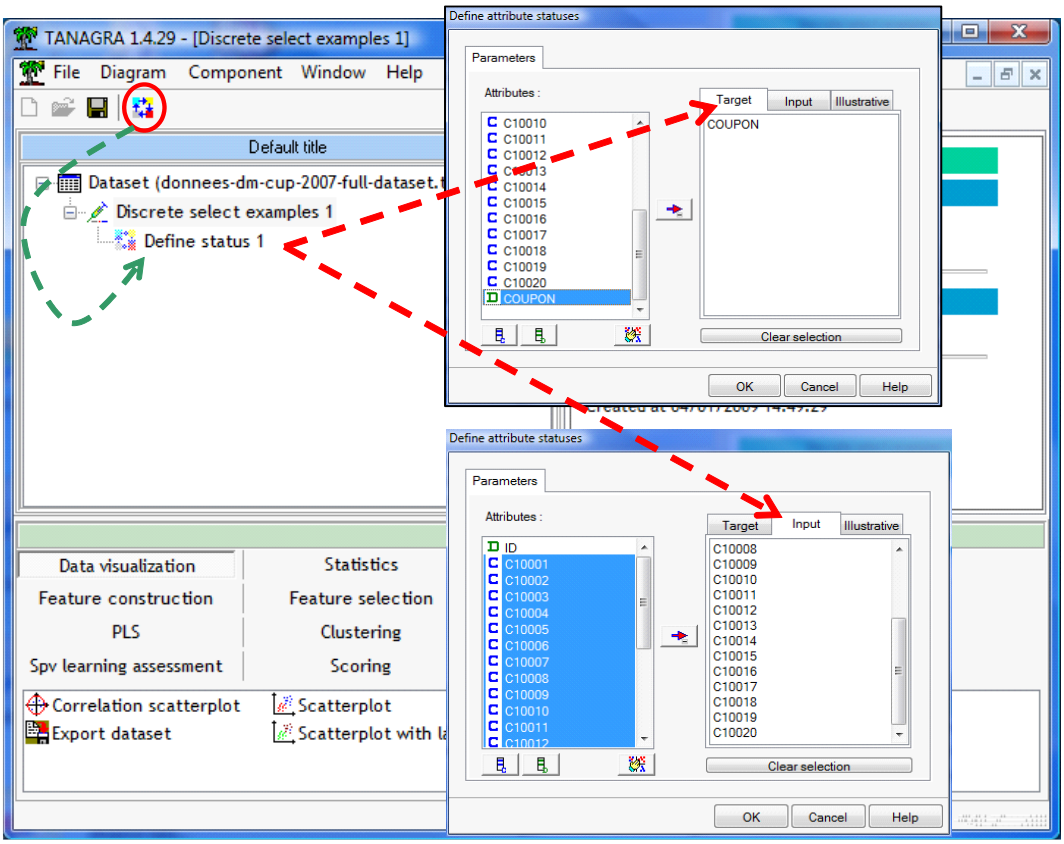


Le couple « ID = TRAIN » identifie les données actives, celles qui serviront pour la construction des modèles de prédiction. En cliquant sur VIEW, Tanagra annonce que 50.000 observations, sur les 100.000 composant le fichier, sont maintenant disponibles pour l'apprentissage.



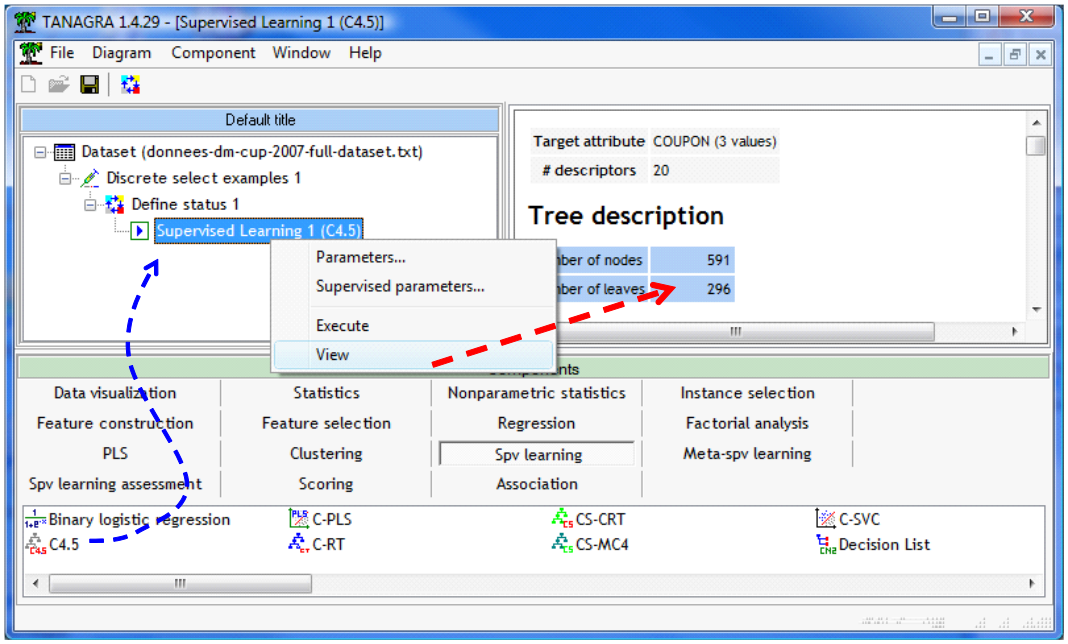
3.3 Statut des variables

L'étape suivante consiste à spécifier la variable à prédire et les variables prédictives. Nous insérons le composant DEFINE STATUS, accessible dans la barre d'outils. Nous plaçons COUPON en TARGET, les autres (C10001 à C10020) en INPUT. La colonne ID n'intervient plus à ce stade.



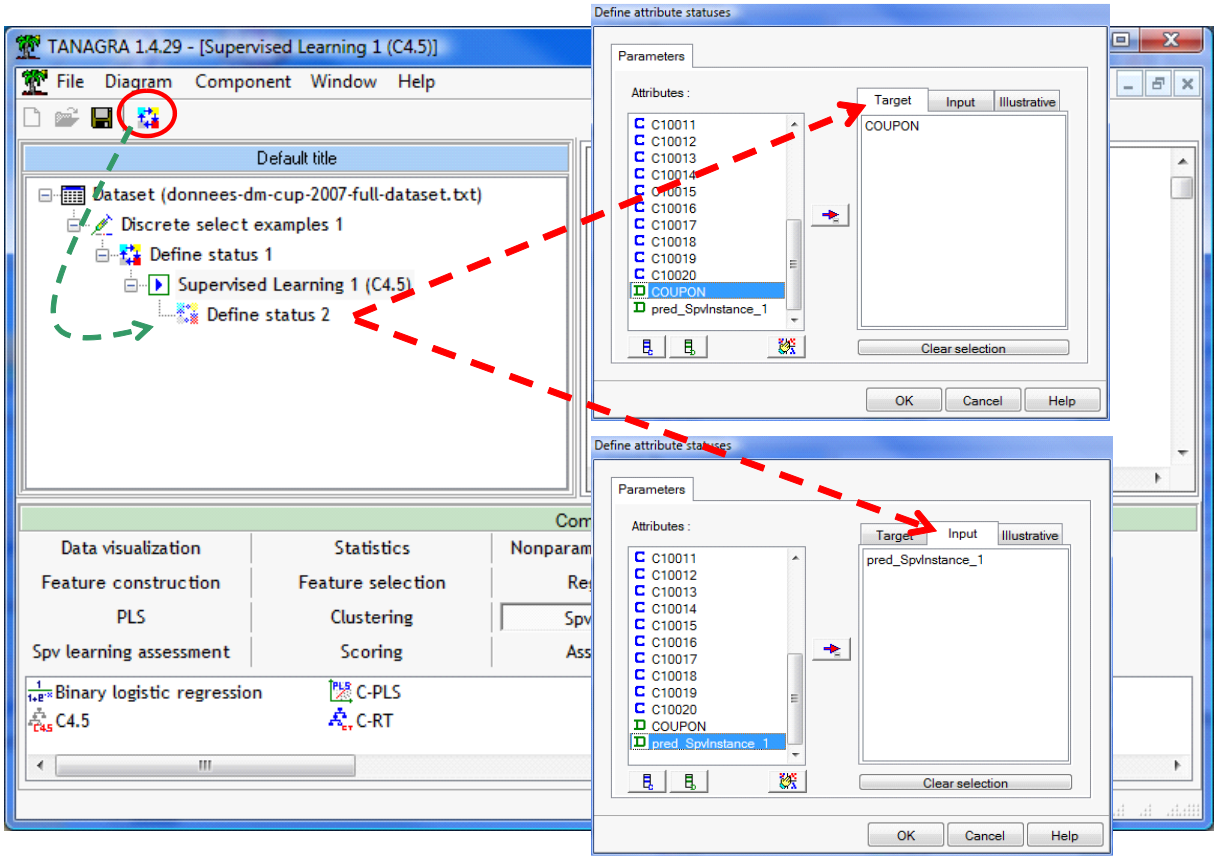
3.4 La méthode C4.5 sans prise en compte des coûts

Nous allons construire et évaluer un modèle de prédiction en utilisant la méthode C4.5. Les coûts sont totalement ignorés lors de l'élaboration du classifieur. Ce résultat servira de référence pour l'évaluation des autres techniques qui seront présentés par la suite. Nous insérons le composant C4.5 (onglet SPV LEARNING) dans le diagramme. Nous actionnons le menu VIEW pour accéder aux résultats.



La lecture de l'arbre de décision, comportant 296 feuilles, est d'un intérêt très limité dans notre contexte. Nous passons directement à l'évaluation.

Nous insérons un nouveau composant DEFINE STATUS, nous plaçons en TARGET la variable à prédire COUPON, en INPUT la variable PRED_SPVINSTANCE_1 produite par le composant d'apprentissage supervisé.

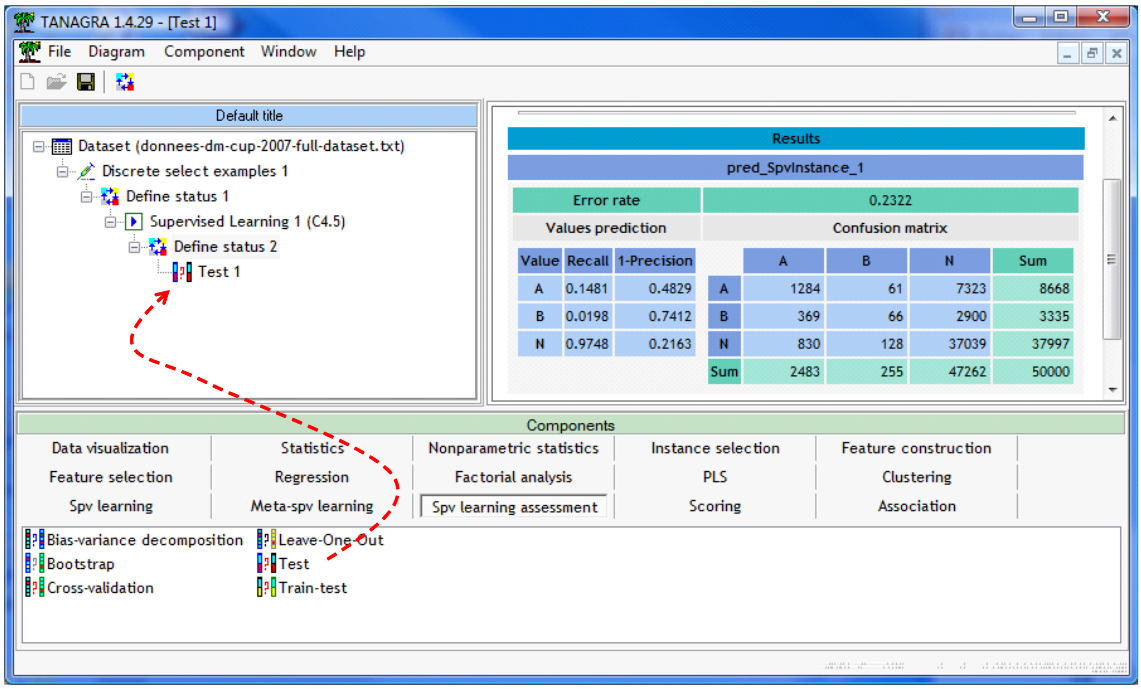


Il ne reste plus qu'à introduire le composant TEST (onglet SPV LEARNING ASSESSMENT) qui va confronter les valeurs observées et les valeurs prédites. Le composant est paramétré par défaut pour réaliser la comparaison sur les données non actives, c.-à-d. les observations qui nous avons initialement mise de côté. Cela nous convient, elles correspondent justement aux observations de l'échantillon test. Nous cliquons sur VIEW.

Le taux d'erreur de 0.2322 nous est très peu utile dans notre situation. Nous calculons directement la fonction de gain, nous obtenons

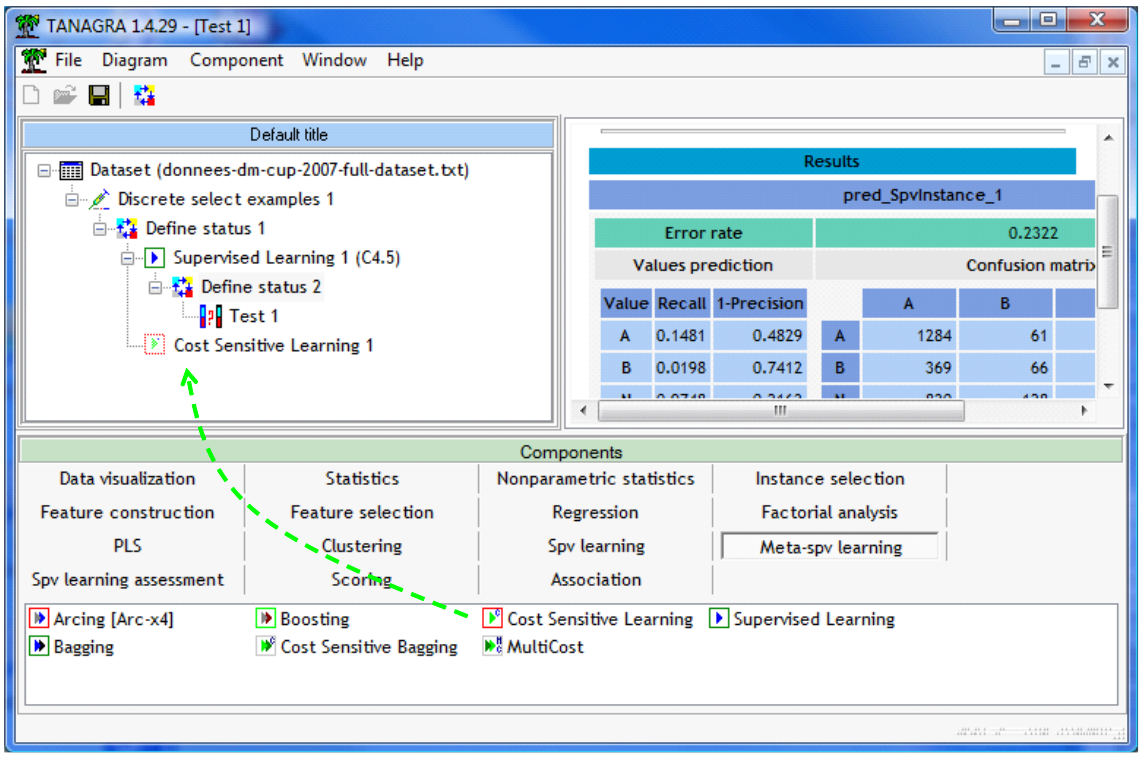
$$\text{Gain} = 3 * 1284 + 6 * 66 - 1 * (369 + 830 + 61 + 128) = \mathbf{2860}$$

Cette valeur nous servira de référence. On s'attend à ce que les méthodes qui intègrent le coût, soit dans le processus de classement, soit dans le processus d'apprentissage, seront de meilleure qualité en proposant un gain plus élevé.

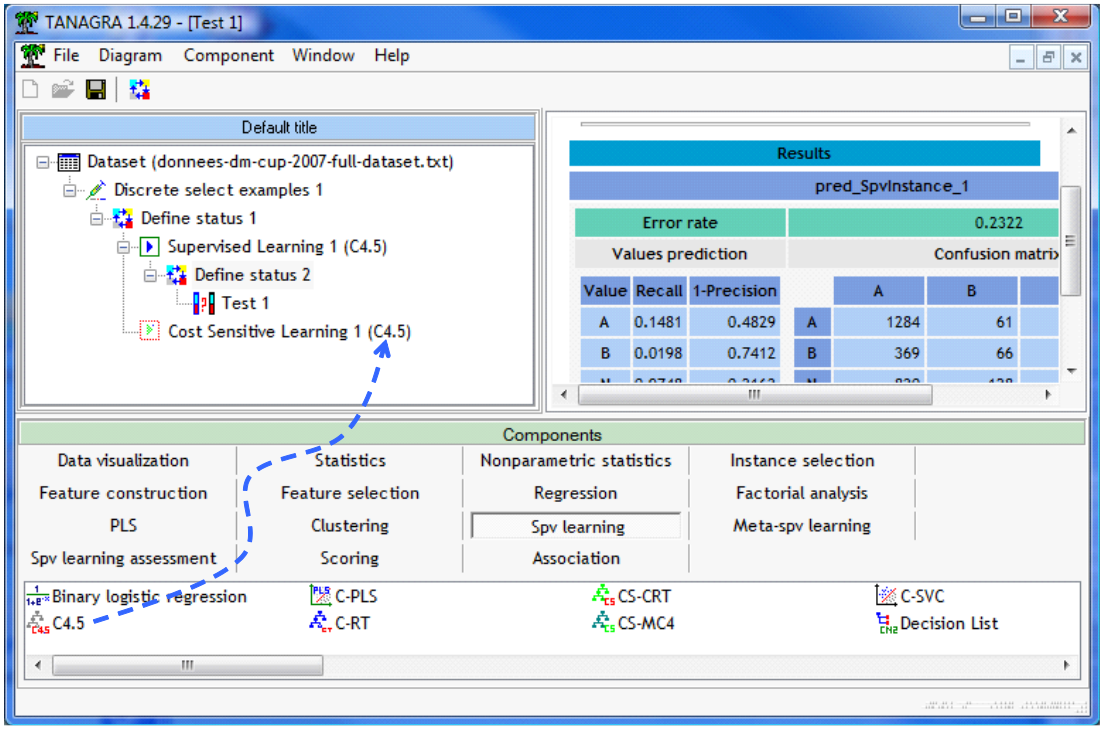


3.5 La méthode « Cost Sensitive Learning + C4.5 »

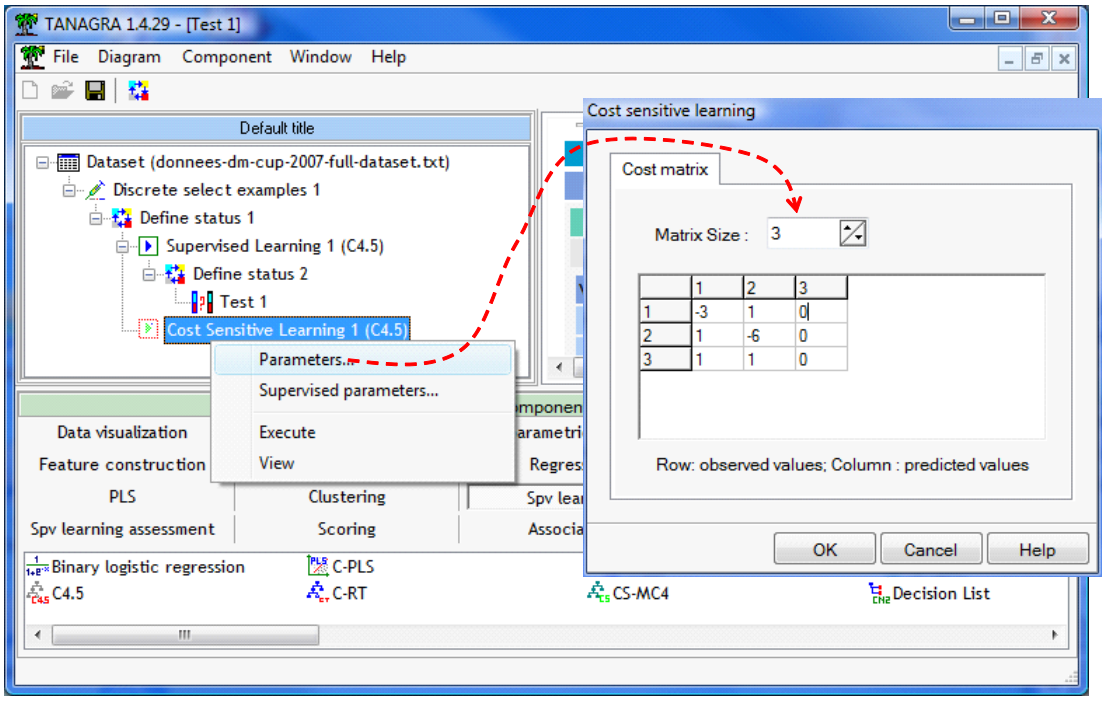
La seconde méthode que nous mettons à l'épreuve dans Tanagra est très simple. L'algorithme d'apprentissage n'est pas modifié. En revanche, il est encapsulé dans un dispositif qui, lorsqu'il doit classer un nouvel individu, applique la conclusion qui minimise le coût de mauvaise affectation. Schématiquement, le composant générique s'appuie sur les probabilités conditionnelles fournies par l'algorithme pour calculer le coût moyen de mauvaise affectation associée à chaque prédiction. La conclusion correspondra alors à celle qui minimise le coût.



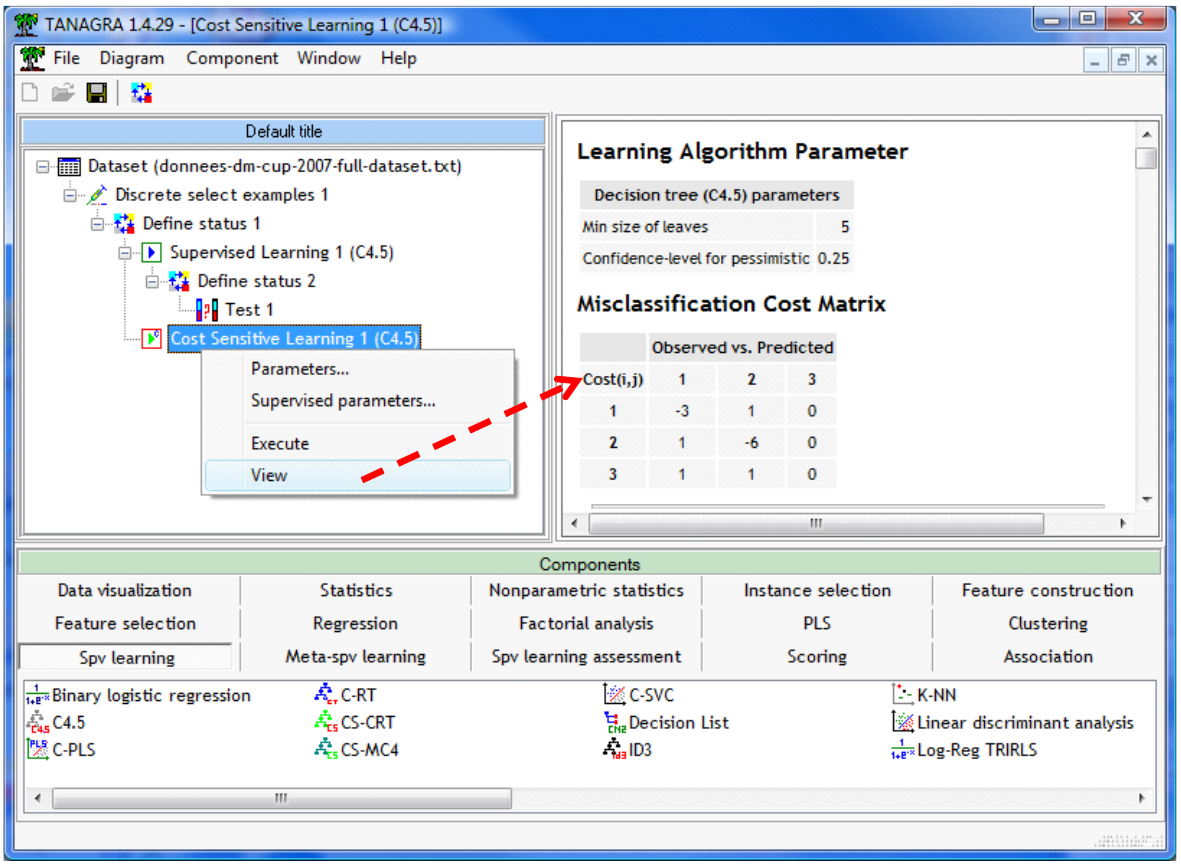
Dans Tanagra, la méthode est introduite en deux temps. Tout d'abord, nous introduisons le composant générique COST SENSITIVE LEARNING (onglet META-SPV LEARNING) (voir copie d'écran ci-dessus). Ensuite, nous y intégrons le composant d'apprentissage supervisé qui peut être n'importe quel outil de la palette SPV LEARNING. Dans notre cas, nous utilisons la méthode C4.5 comme précédemment.



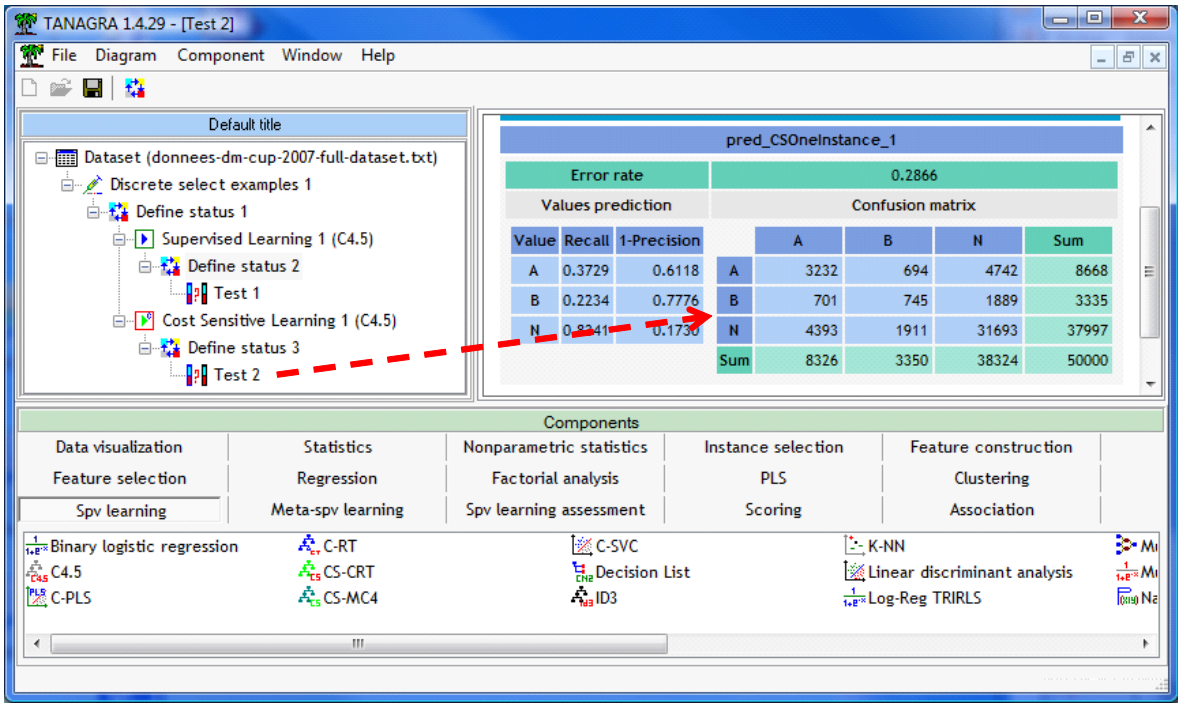
Nous devons maintenant paramétrer le composant en spécifiant la matrice de coût. Nous cliquons le menu PARAMETERS (*Remarque : Le menu contextuel SUPERVISED PARAMETERS, également accessible, sert à paramétrer l'algorithme d'apprentissage qui est encapsulé dans le composant*).



Il nous reste à actionner le menu VIEW pour accéder au résultat. L'arbre est complètement identique au précédent, à la différence qu'une matrice de coût est maintenant utilisée lors du classement des individus, notamment ceux de l'échantillon test.



Evaluons maintenant les performances de l'approche en réintroduisant le composant TEST.



Dans le DEFINE STATUS qui le précède, COUPON est toujours en TARGET, en INPUT nous mettons cette fois-ci la variable PRED_CSONEINSTANCE_1 nouvellement produite.

En appliquant la formule de gain, nous obtenons :

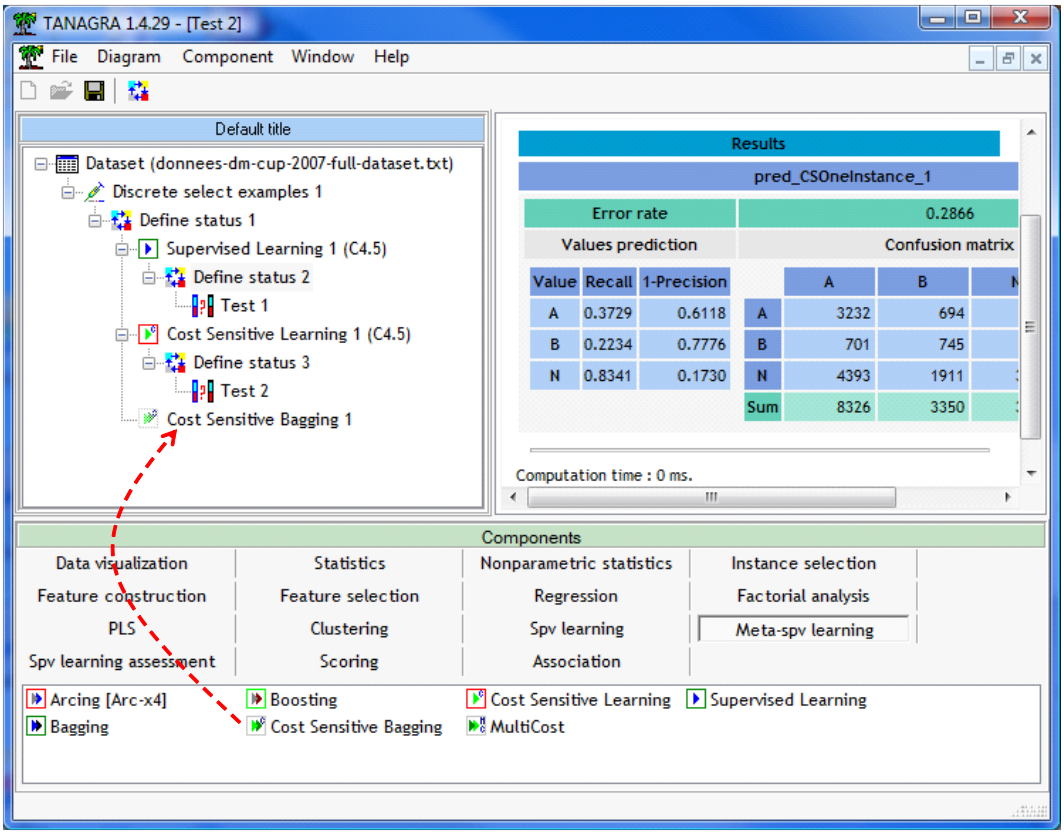
$$\text{Gain} = 3 * 3232 + 6 * 745 - 1 * (701 + 4393 + 694 + 1911) = \mathbf{6467}$$

L'écart avec la méthode précédente est plus que convaincant. Tenir compte des coûts, ne serait-ce que lors de la phase de classement, permet d'améliorer significativement les performances.

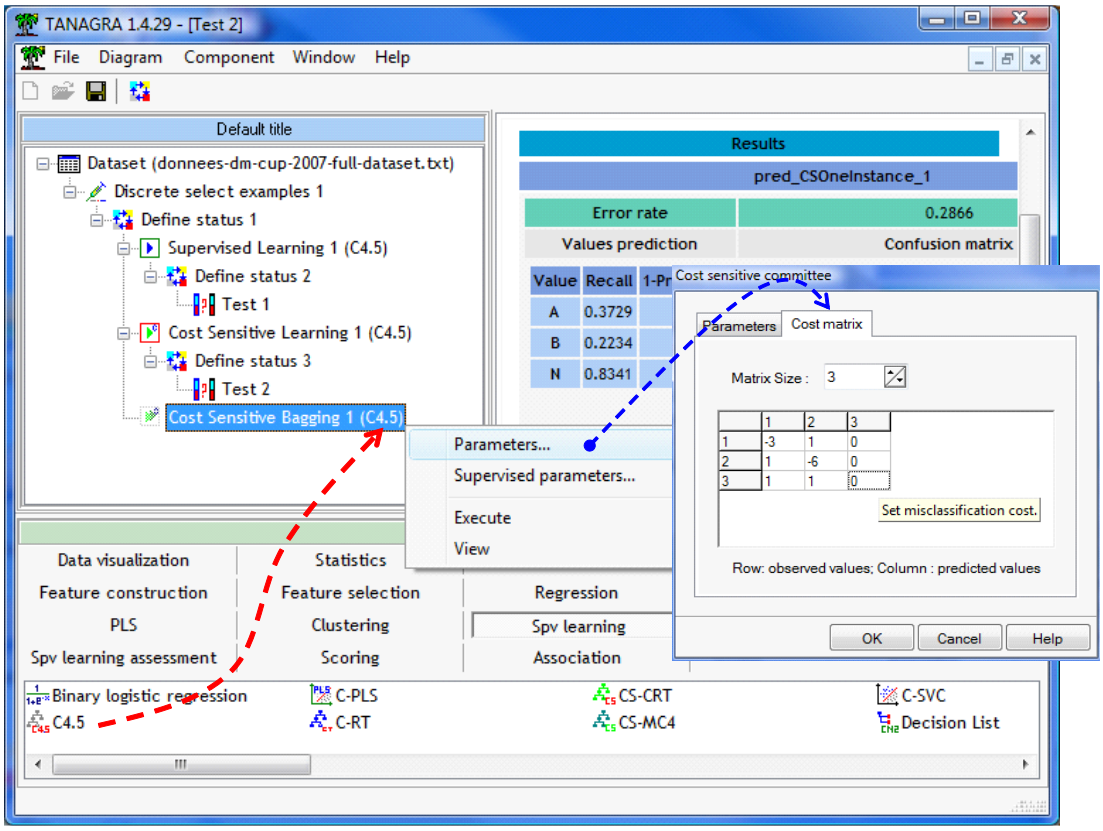
3.6 La méthode « Cost Sensitive Bagging + C4.5 »

Les méthodes d'agrégation par ré échantillonnage sont connues pour améliorer les performances des classifieurs individuels. Nous avons programmé la méthode BAGGING dans Tanagra. Il s'agit simplement de répéter la construction des modèles individuels sensibles au coût, de les faire voter, puis de s'appuyer de la distribution conditionnelle produite par ces votes pour produire une affectation sensible aux coûts.

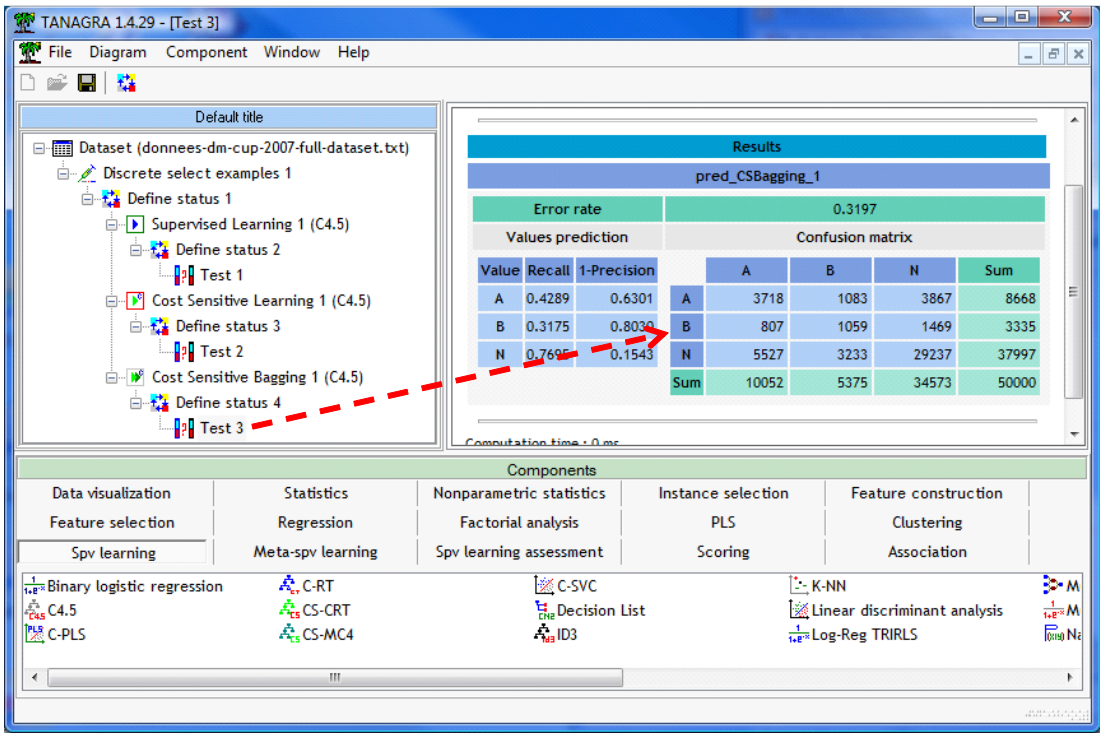
Dans notre cas, nous introduisons tout d'abord le composant COST SENSITIVE BAGGING (onglet META-SPV LEARNING)



Puis nous lui intégrons la méthode C4.5. Ici également, nous aurions pu utiliser tout autre algorithme d'apprentissage. Pour l'heure, le paramétrons le composant en actionnant le menu PARAMETERS. Nous saisissons la matrice de coût. Le nombre de réplications par défaut est de 25.



Nous actionnons le menu VIEW pour accéder aux résultats. Le calcul est un peu plus long bien évidemment. Outre la matrice de coût utilisée, nous visualisons également le coût associé à chaque modèle (colonne improprement appelé « Error Rate »). Etant calculée sur l'échantillon d'apprentissage, cette information est peu intéressante. Passons directement à l'évaluation sur la partie test en confrontant PRED_CSBAGGING_1 avec COUPON. Nous obtenons :



Nous calculons la fonction de Gain,

$$\text{Gain} = 3 * 3718 + 6 * 1059 - 1 * (807 + 5527 + 1083 + 3233) = \mathbf{6858}$$

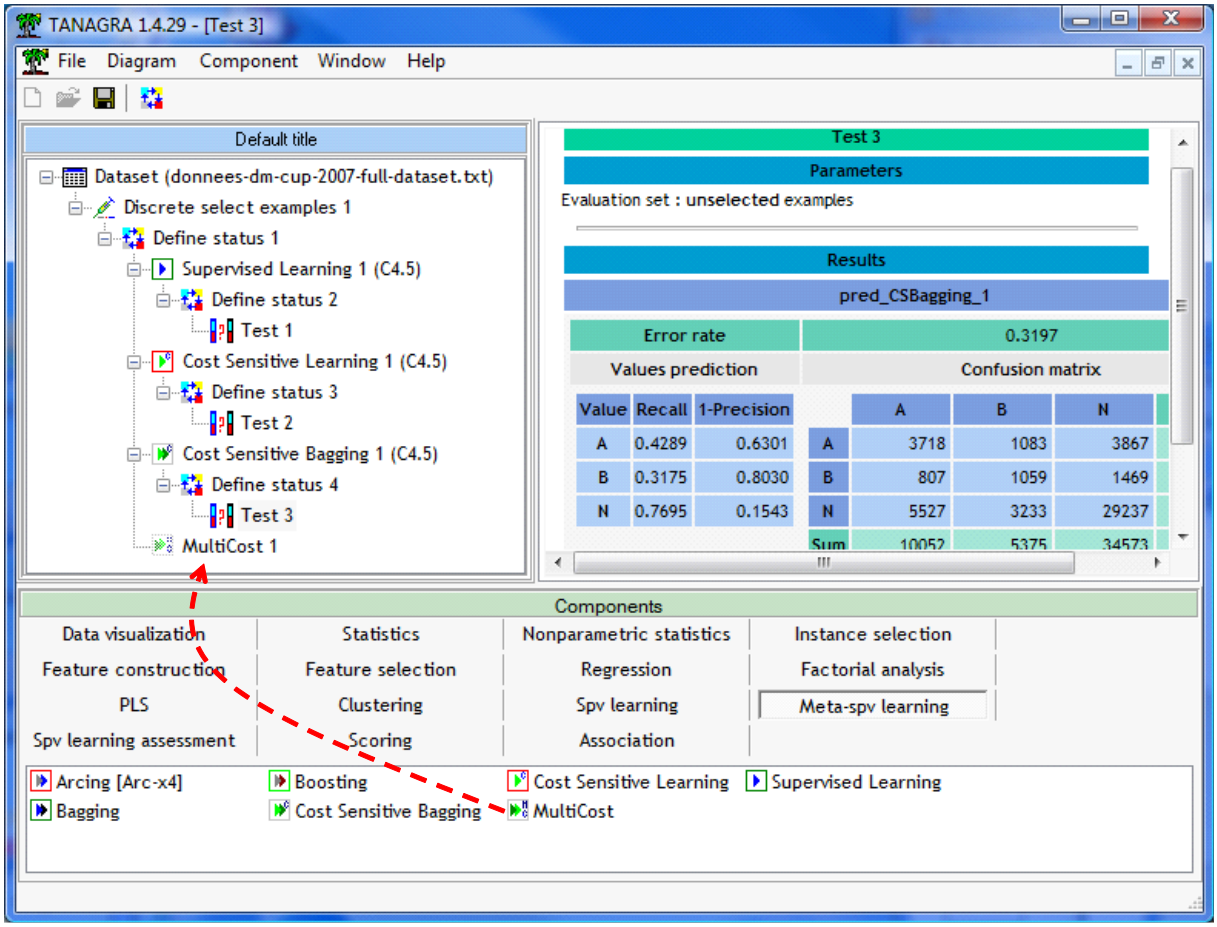
L'amélioration est moins spectaculaire par rapport à la configuration précédente (6858 vs. 6467 = 391). Elle semble quand même significative.

3.7 La méthode « MultiCost + C4.5 »

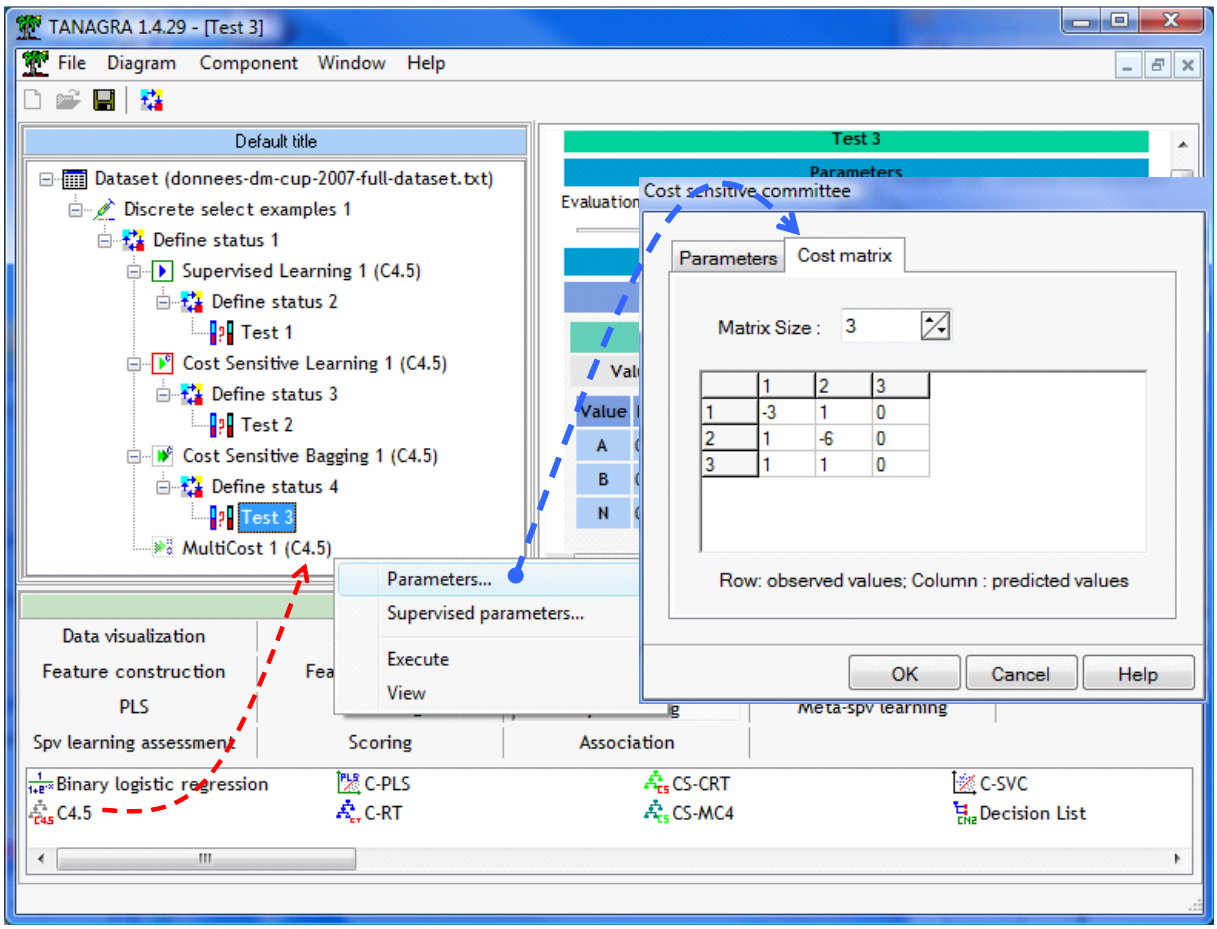
L'inconvénient de l'approche précédente est que nous disposons d'un ensemble de modèles. Le processus d'affectation pour un individu est peu lisible, fermant la porte à toute tentative d'interprétation des résultats.

La méthode MultiCost tente de combiner les deux approches. Elle est inspirée de la méthode MetaCost de Domingos (1999). Elle agit en trois temps : (1) nous créons un ensemble de modèles avec un processus Bagging ; (2) nous effectuons une prédiction sensible aux coûts en nous servant des distributions conditionnelles produites à l'aide du vote de l'ensemble des classifieurs ; (3) nous nous servons de cette nouvelle variable comme variable à prédire dans la construction d'un modèle unique. A la différence de MetaCost, les coûts sont déjà introduits au niveau des modèles individuels (avant le vote) dans MultiCost. Ils interviennent donc deux fois dans le processus.

Nous insérons le composant MULTICOST (onglet META-SPV LEARNING) dans le diagramme.



Puis nous lui intégrons l'algorithme C4.5. De nouveau nous paramétrons la matrice de coûts en actionnant le menu PARAMETERS.



Le menu VIEW produit le résultat. Par rapport à la situation précédente, nous avons un modèle unique que nous pouvons interpréter. Certes, dans notre cas, essayer d'interpréter un arbre à 317 feuilles est peut être un peu optimiste. Mais lorsque la situation s'y prête, cet aspect peut se révéler décisif dans la pratique du Data Mining.

Il ne nous reste plus qu'à mettre en place le dispositif d'évaluation sur l'échantillon test. Nous croisons cette fois-ci COUPON avec PRED_MULTICOST_1, nous obtenons (Figure 1)

$$\text{Gain} = 3 * 3720 + 6 * 1081 - 1 * (801 + 5522 + 1111 + 3347) = \mathbf{6865}$$

L'écart des performances est négligeable par rapport à la situation précédente. C'est néanmoins encourageant. En effet, pour un même niveau de performance, outre les aspects liés à l'interprétation évoqués précédemment, nous n'avons plus qu'une seule structure à manipuler lors du déploiement du modèle.

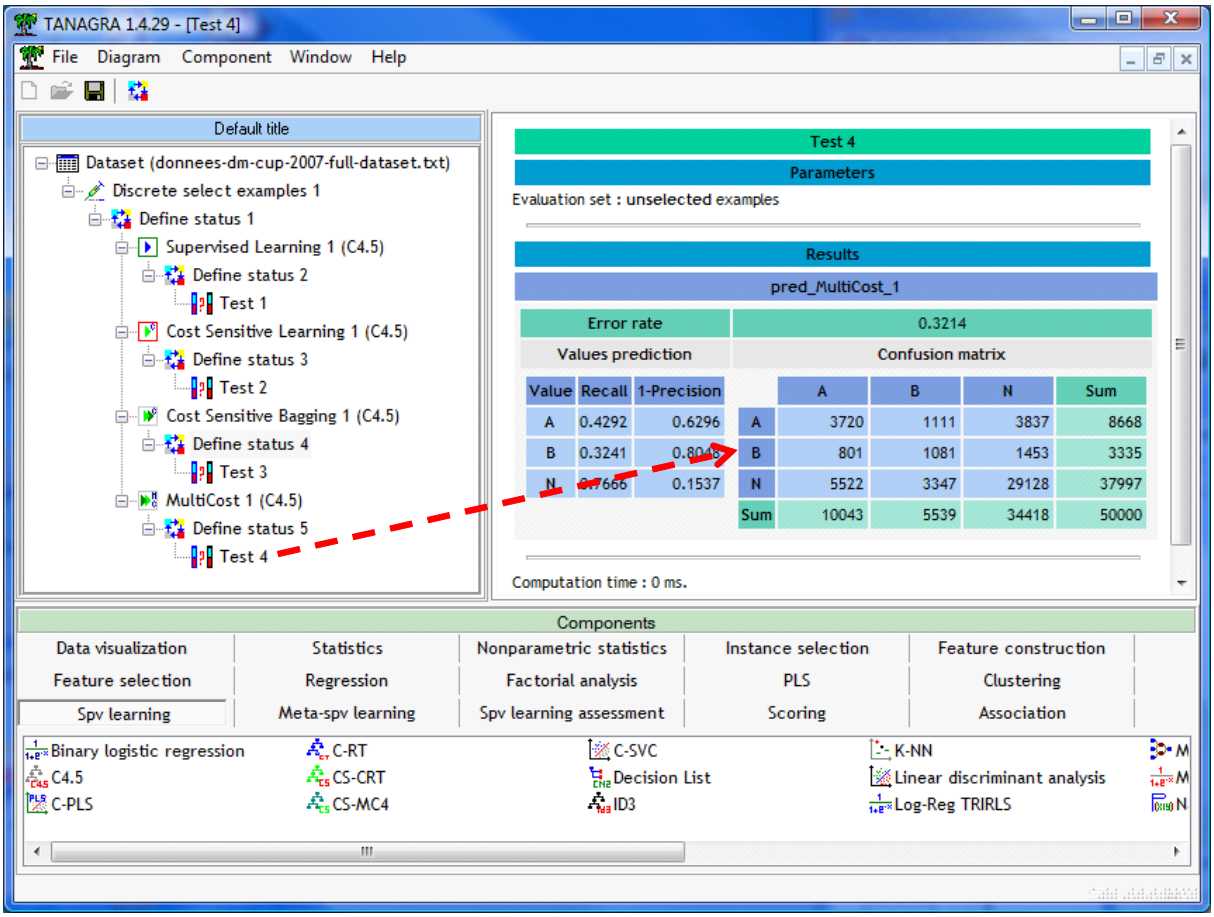
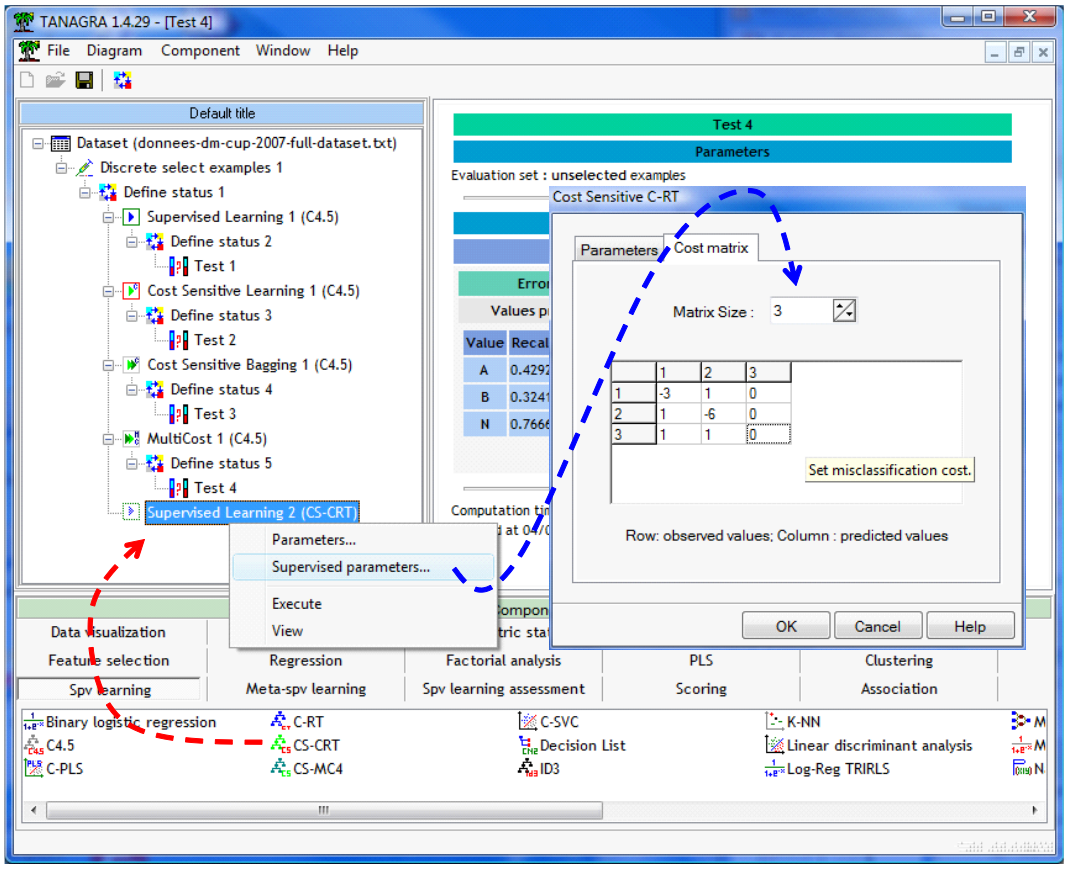


Figure 1 - Matrice de confusion en test - MultiCost + C4.5

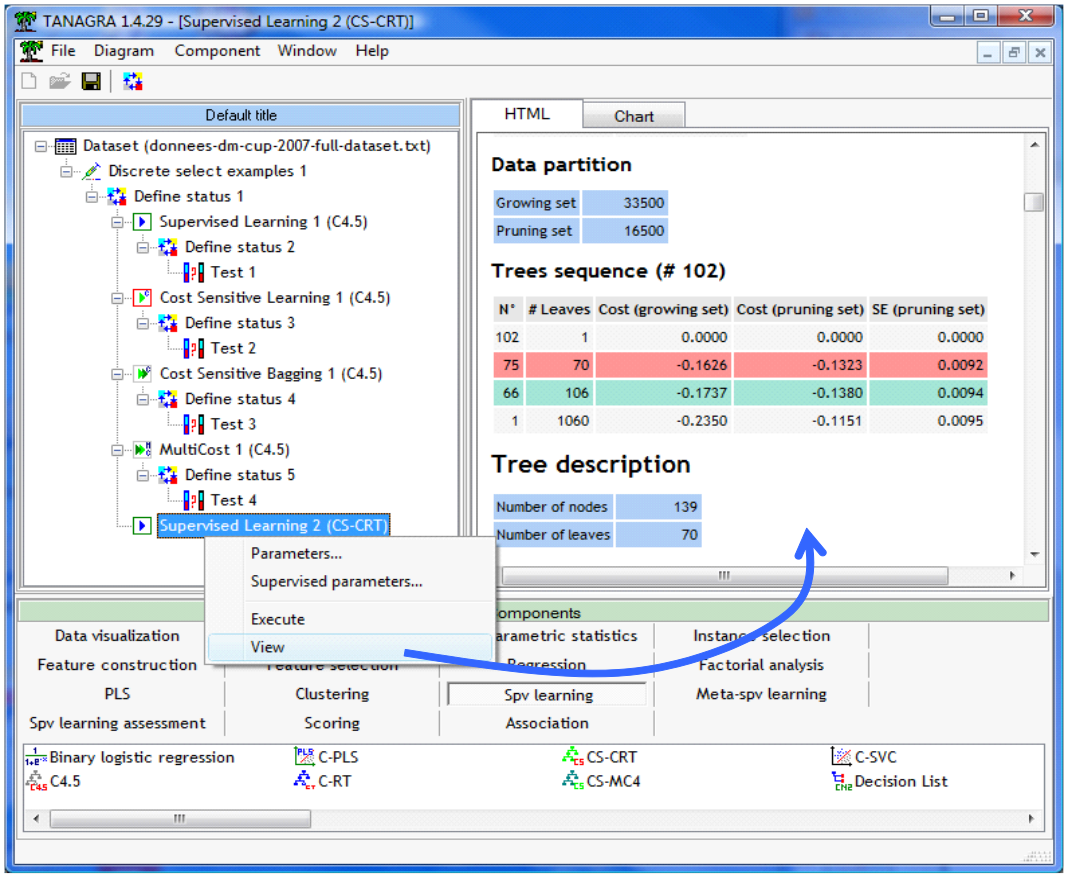
3.8 La méthode CART

Jusqu'à présent, les approches proposées agissaient comme des chapeaux externes qui intervenaient sur les résultats des modèles élaborés de manière classique. Dans cette section, nous étudions les méthodes qui intègrent les coûts au cœur même du dispositif d'apprentissage. C'est le cas de la méthode d'induction d'arbre CART. Il est possible d'optimiser le coût de mauvais classement lors de la phase de post-élagage (Breiman et al., 1984 ; voir section 11.4, pages 303 à 306).

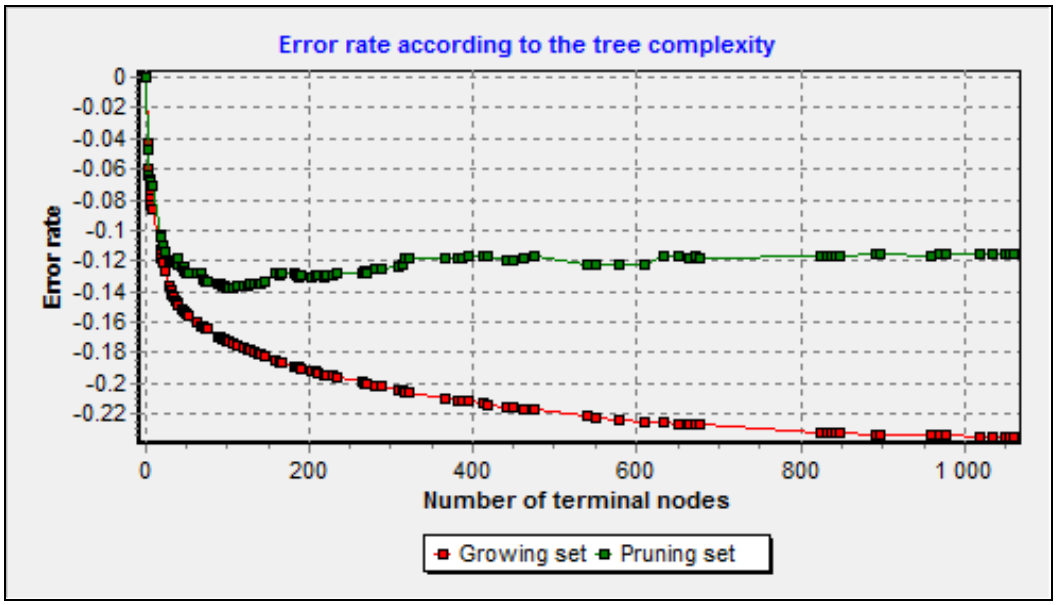
Nous introduisons le composant CS-CRT (onglet SPV LEARNING), la méthode CART sensible au coût, dans le diagramme. Nous paramétrons la matrice de coût via le menu SUPERVISED PARAMETERS maintenant puisque la prise en compte du coût est intégrée dans la méthode même.



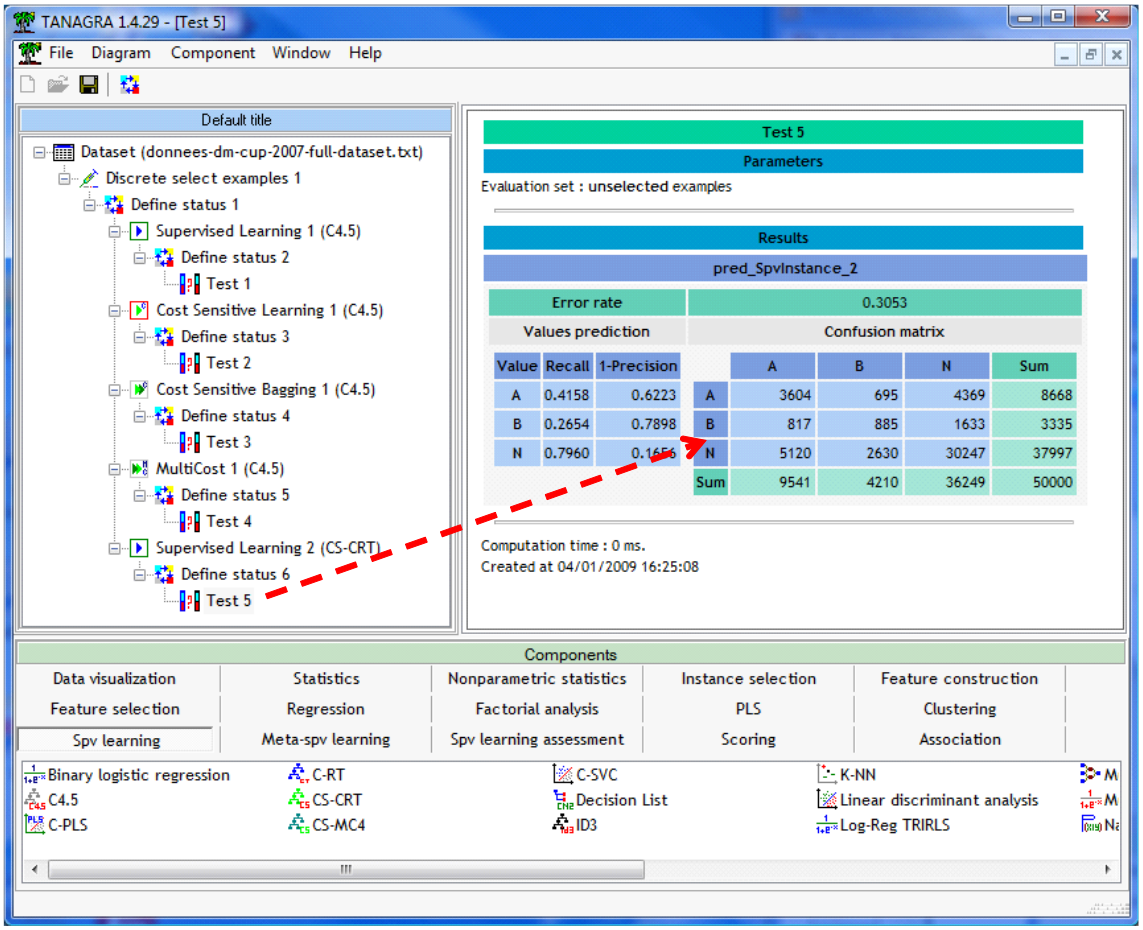
Nous actionnons le menu VIEW. Nous obtenons un arbre avec 70 feuilles.



Si l'on passe à l'onglet CHART de la fenêtre de visualisation. Nous constatons que l'élagage agit de manière appropriée, réduisant efficacement la taille de l'arbre sans dégrader les performances (courbe verte pour l'échantillon d'élagage prélevé dans l'échantillon d'apprentissage).



Voyons ce qu'il en est sur les 50.000 observations réservées pour le test (comparaison entre COUPON et PRED_SPVINSTANCE_2



Le gain est égal à

$$\text{Gain} = 3 * 3604 + 6 * 885 - 1 * (817 + 5120 + 695 + 2630) = \mathbf{6860}$$

Le gain final est très similaire à celui de la méthode MultiCost, même si la structure de l'erreur n'est pas la même.

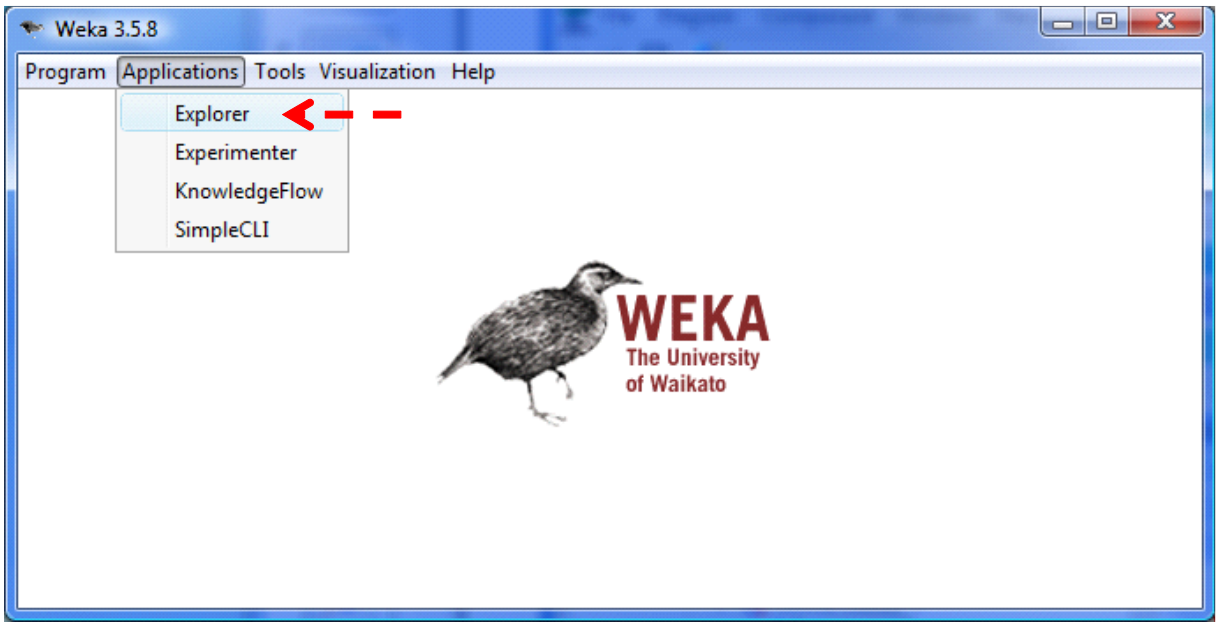
Remarque : Tanagra propose un second algorithme d'induction d'arbres sensible aux coûts. Il s'agit du composant **MS-MC4**. Une variante très similaire, appelée « *Cost Sensitive C4.5 (Chauchat & Rakotomalala, 2001)* », est implémentée dans SIPINA (http://eric.univ-lyon2.fr/~ricco/sipina.html). Elle a été mise à contribution dans un de nos précédents didacticiels accessible en ligne (voir http://tutoriels-data-mining.blogspot.com/2008/03/apprentissage-test-avec-sipina.html ou http://sipina.over-blog.fr/article-17592319.html). Utilisé sur nos données, la performance obtenue sur le fichier test est **Gain = 6990** avec un arbre à 46 feuilles.

4 Analyse avec Weka

Weka est un des très rares logiciels à proposer un dispositif assez facile d'accès pour intégrer les coûts de mauvais classement. Il sait également appréhender les coûts négatifs pour quantifier les gains. C'est suite à l'étude attentive du comportement de Weka d'ailleurs que nous avons défini le mode de gestion des coûts dans Tanagra.

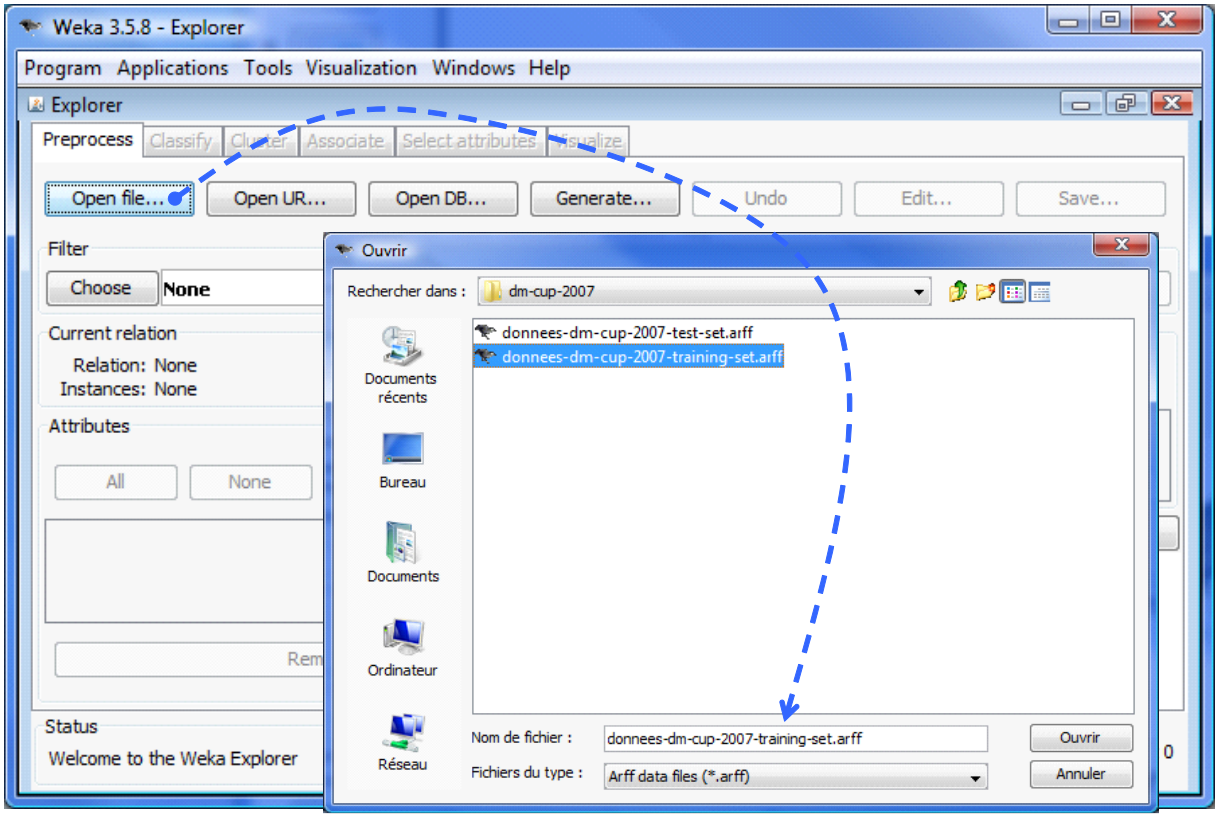
Weka ne sait pas subdiviser un fichier à partir d'une colonne des données. Nous avons donc scindé manuellement le fichier en apprentissage « donnees-dm-cup-2007-training-set.arff » et en test « donnees-dm-cup-2007-test-set.arff ». Nous utilisons le format ARFF spécifique à Weka.

Il y a plusieurs manières d'utiliser Weka. Dans ce didacticiel, le plus naturel est de passer par le module EXPLORER. Pour ce faire, au lancement de Weka, nous actionnons le menu APPLICATIONS / EXPLORER.



4.1 Importation du fichier « apprentissage »

Première étape toujours dans tout logiciel de Data Mining, nous devons importer les données. Le bouton OPEN de l'onglet PREPROCESS permet de le faire. Nous sélectionnons « donnees-dm-cup-2007-training-set.arff ».

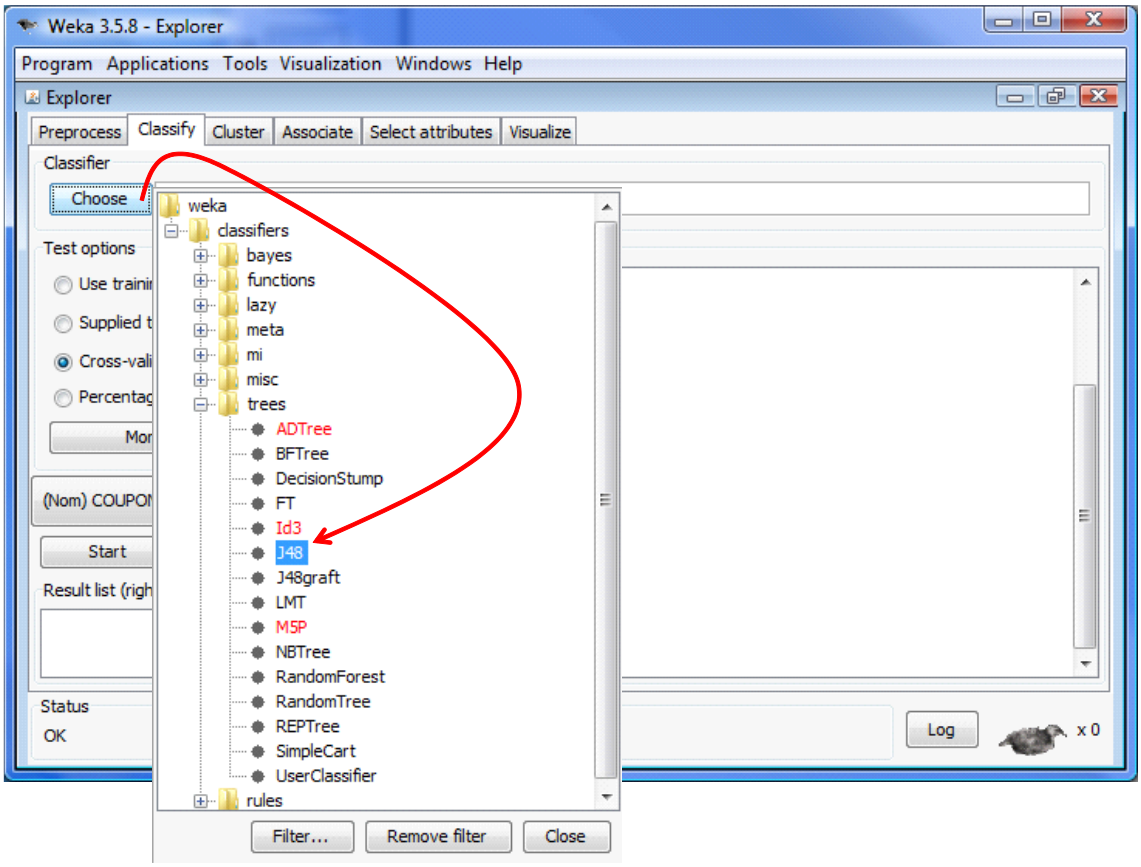


Par défaut, la dernière colonne correspond à la variable à prédire dans Weka. Notre fichier est correctement configuré.

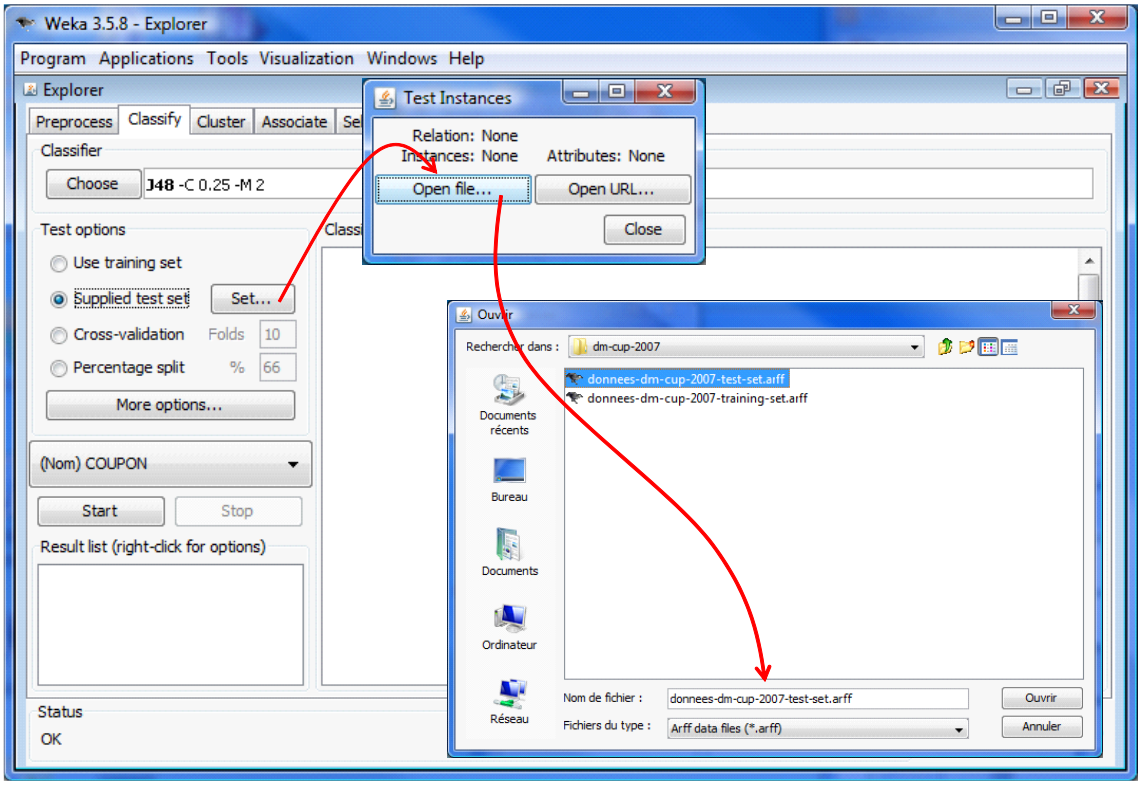
4.2 La méthode J48 sans prise en compte des coûts

Weka ne propose pas la méthode C4.5 de Quinlan (1993) mais plutôt la méthode J48 qui est très similaire. Nous l'utiliserons sans précautions particulières donc, les résultats devraient être très proches, sauf cas pathologiques que nous identifierons par la suite.

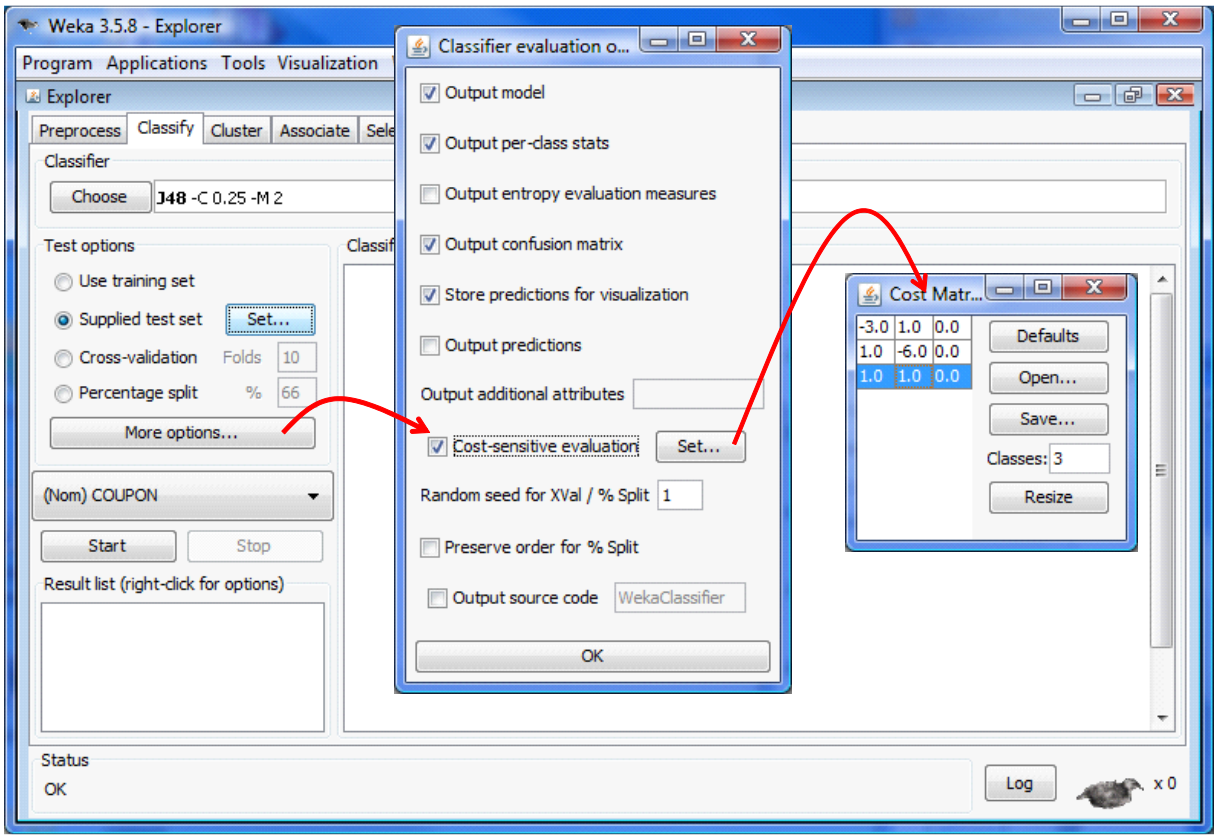
Choix de la méthode. Pour choisir la méthode d'apprentissage, nous accédons à l'onglet CLASSIFY. Nous choisissons dans le bloc TREES la méthode J48. Nous utilisons les paramètres par défaut.



Spécifier l'échantillon test. Nous sélectionnons l'option SUPPLIED TEST SET pour indiquer à Weka que nous utilisons un second fichier pour l'évaluation. En cliquant sur SET, nous pouvons désigner le fichier « donnees-dm-cup-2007-test-set.arff ».



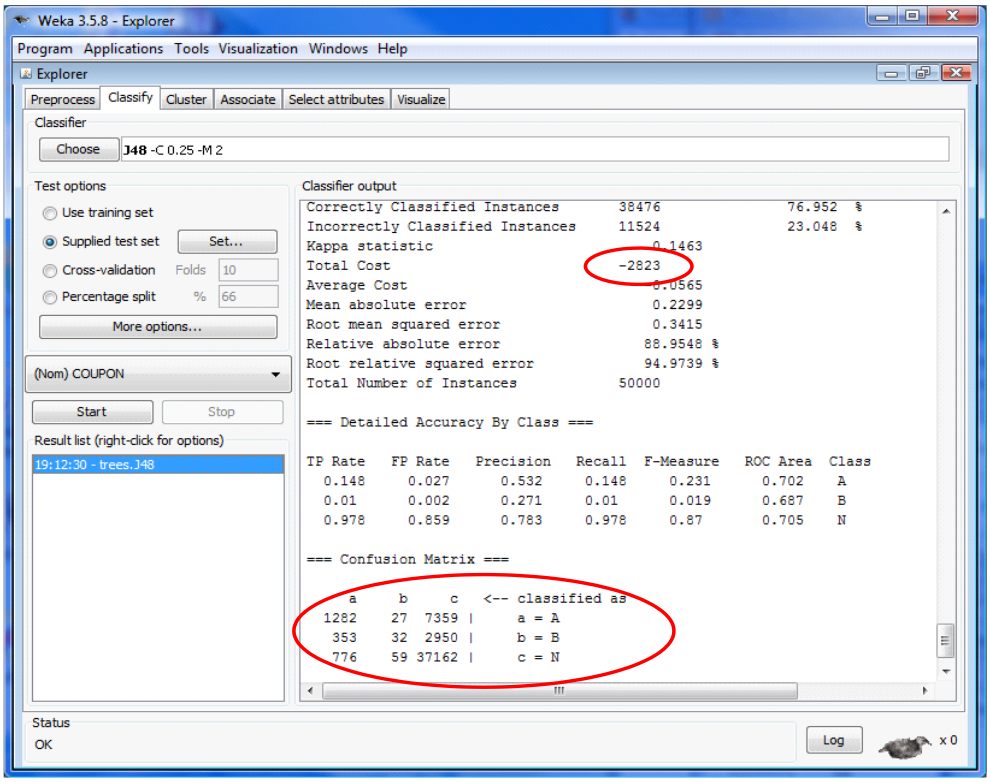
Prise en compte des coûts lors de l'évaluation. Weka propose une option très intéressante. Il sait bien sûr calculer le taux d'erreur mais, si nous lui fournissons la matrice de coût de mauvais classement, il sait aussi calculer directement la fonction de perte (-1.0 * fonction de gain). Nous accédons à l'option adéquate via le bouton MORE OPTIONS. Dans la boîte de dialogue qui apparaît nous cochons l'option COST-SENSITIVE EVALUATION. Avec le bouton SET, nous avons la possibilité de définir la matrice de coûts.



Apprentissage – test. Une fois tous ces préalables correctement définis, il ne nous reste plus qu'à cliquer sur le bouton START pour lancer l'analyse complète.

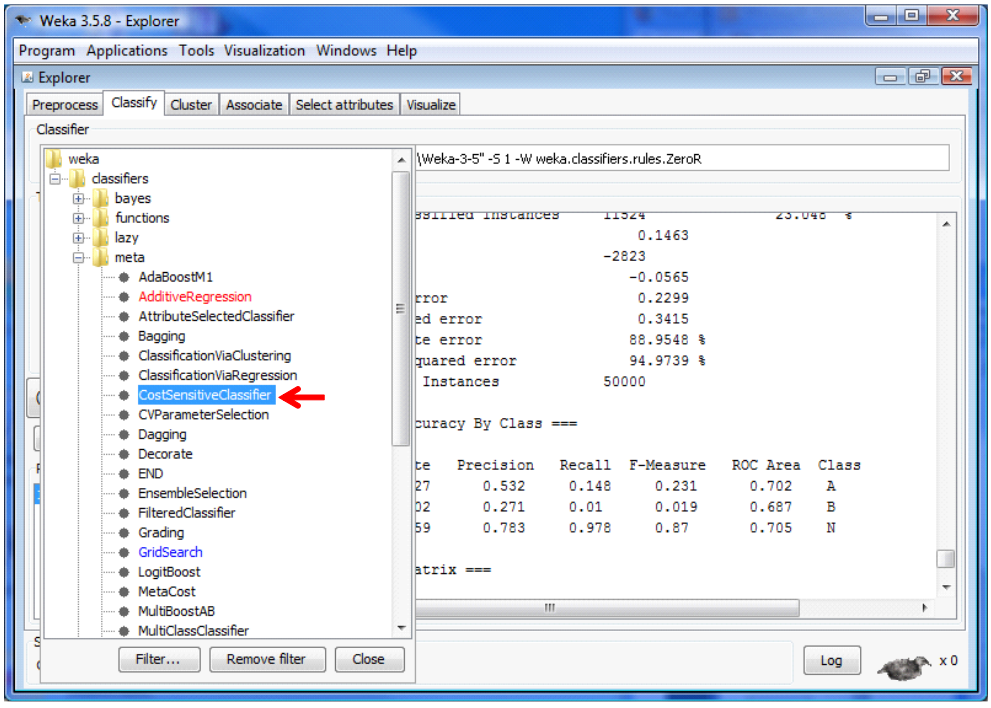
Weka via la méthode J48 a élaboré un arbre à 213 feuilles, moins que celui élaboré avec la méthode C4.5 de Tanagra, nous verrons plus loin que ce n'est pas anodin.

Le coût obtenu sur le fichier test est -2823, ce qui correspond à un **Gain = 2823**, finalement similaire à la méthode C4.5 (qui était de 2860 rappelons-le). Les résultats sont cohérents et c'est heureux ainsi.



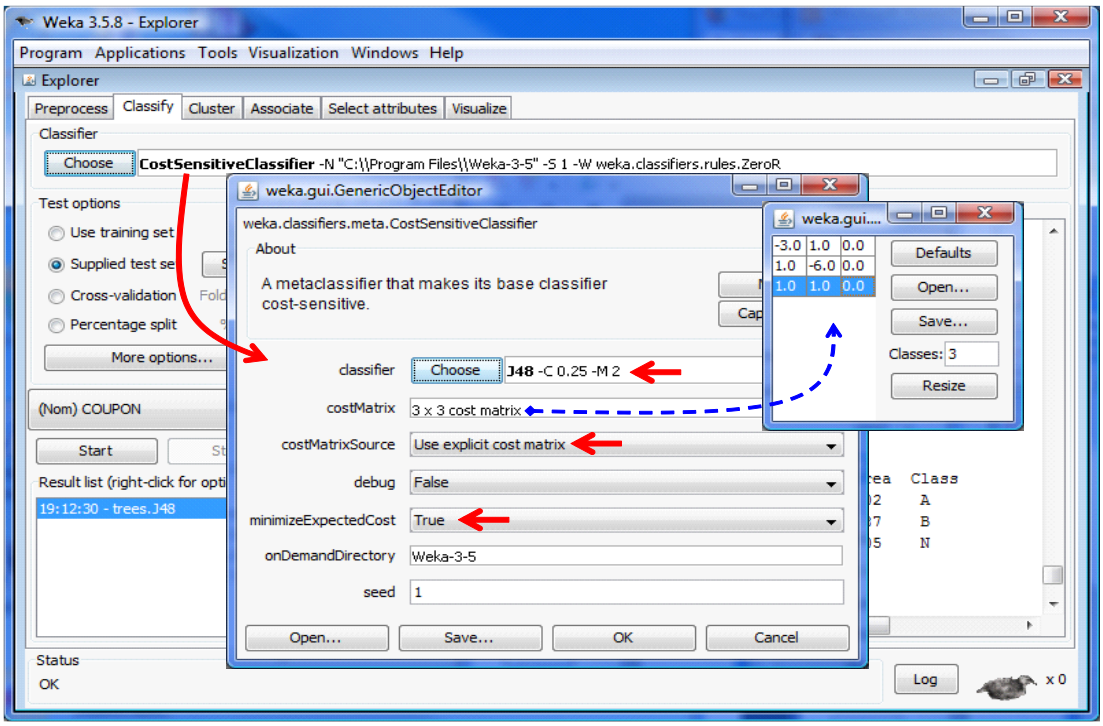
4.3 La méthode « Cost Sensitive Learning + J48 »

Il est possible d'encapsuler la méthode J48 dans une approche sensible au coût. Il s'agit aussi de corriger les affectations en minimisant le coût de mauvais classement. Nous cliquons sur le bouton CHOOSE et sélectionnons dans le bloc META la méthode COST SENSITIVE CLASSIFIER.

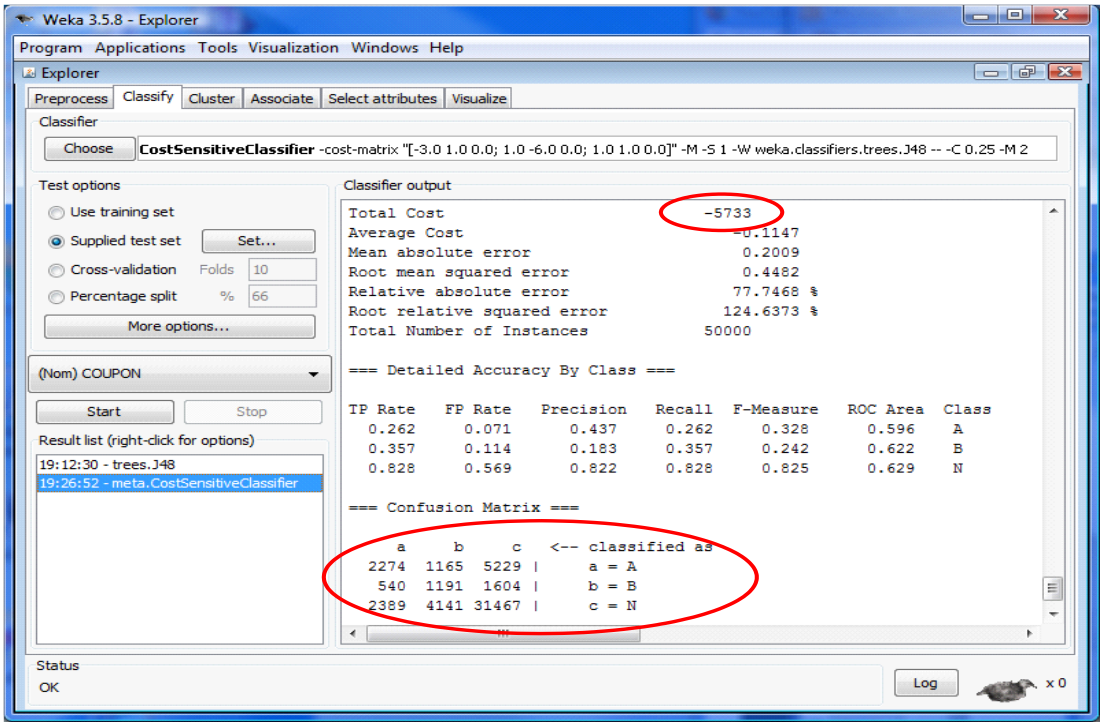


Nous le paramétrons en cliquant sur la barre de description de la méthode. Nous sélectionnons la méthode J48 en cliquant sur CLASSIFIER, sans modifier le paramétrage par défaut dans un premier

temps. Dans COST MATRIX, nous décrivons la matrice 3 x 3 avec les coûts adéquats. COST MATRIX SOURCE indique maintenant « Use explicit cost matrix ». Nous indiquons que l'on doit minimiser le coût de mauvaise affectation en précisant MINIMIZE EXPECTED COST = TRUE.



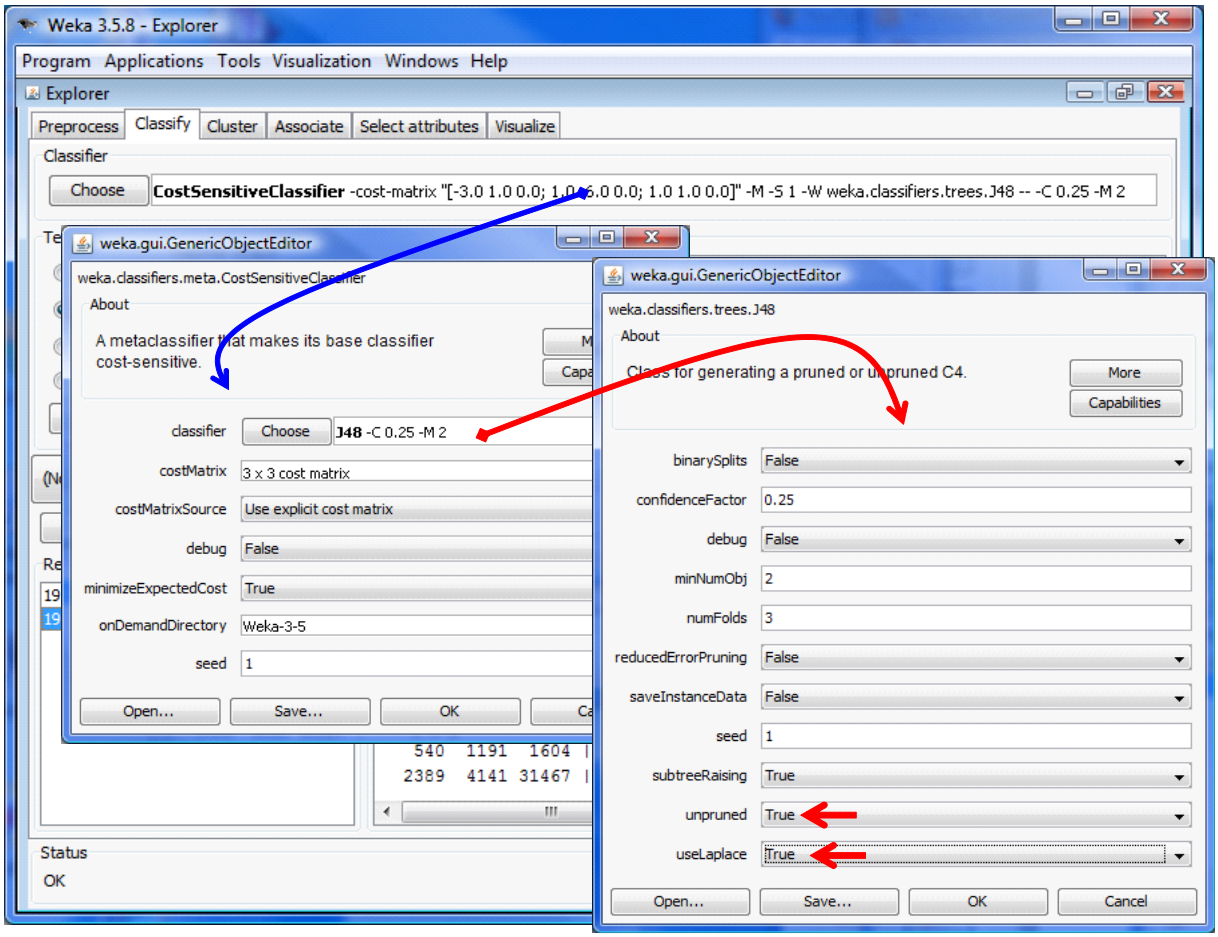
Il ne nous reste plus qu'à cliquer de nouveau sur START.



En utilisant une méthode sensible au coûts, nous obtenons un Gain = 5733. C'est mieux que pour la méthode ne tenant pas compte des coûts. Néanmoins, c'est plutôt décevant si l'on se réfère aux résultats fournis par Tanagra (Cost Sensitive Learning + C4.5 → Gain = 6467).

La réponse se trouve dans le paramétrage de la méthode J48. Nous avions remarqué précédemment qu'elle était fortement élaguée, plus en tous les cas que C4.5. Or, on sait par ailleurs que les arbres fortement élagués sont de mauvais estimateurs des probabilités conditionnelles d'affectation (voir par exemple Elkan, « The foundations of cost-sensitive learning », IJCAI-2001 ; http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.514). Nous devons donc paramétrer J48 de manière à : (1) empêcher le post-élagage, (2) à corriger les estimations des probabilités sur les petits effectifs en utilisant l'estimateur laplacien.

Pour ce faire, nous cliquons de nouveau dans les barres de description des méthodes. Nous indiquons UNPRUNED = TRUE et USELAPLACE = TRUE.

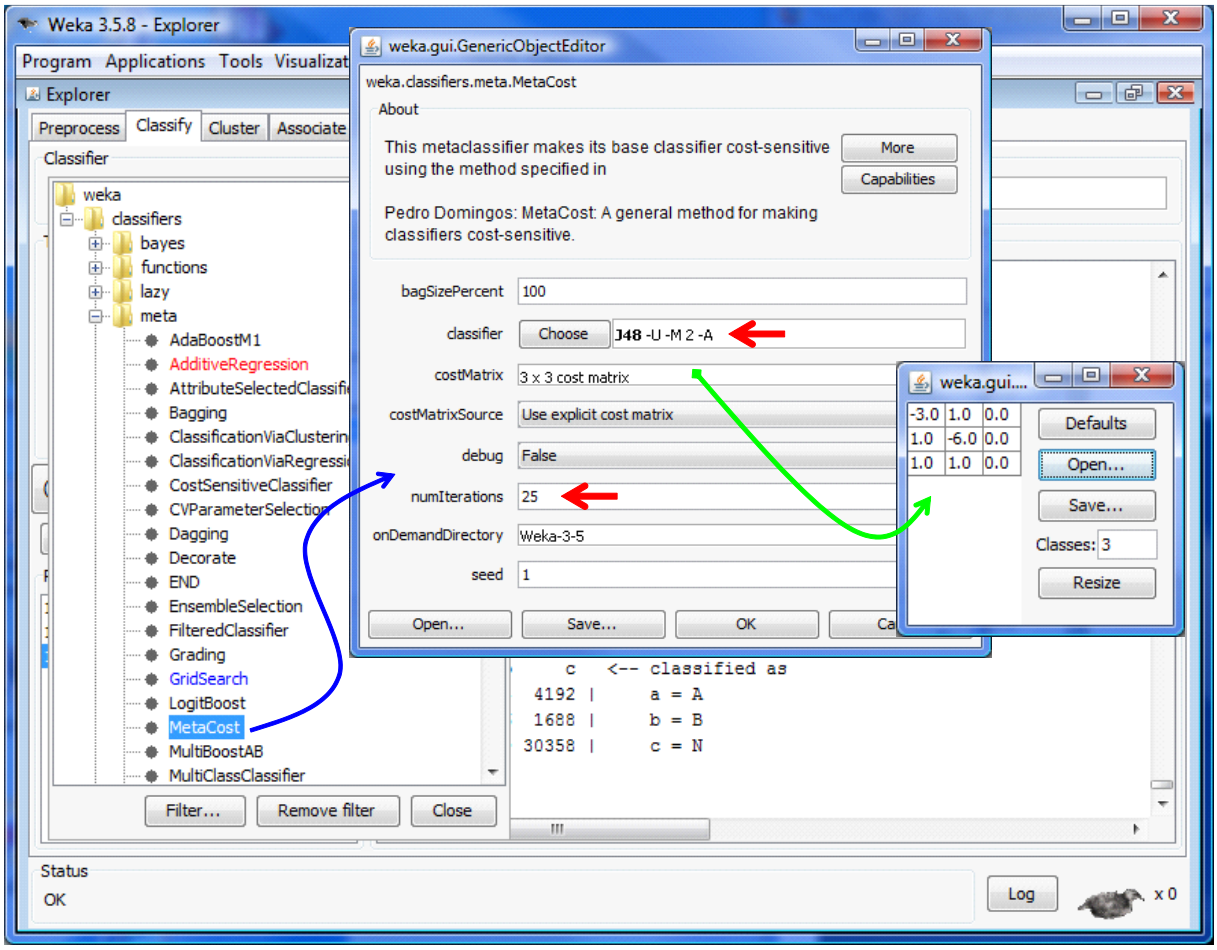


En cliquant sur START, nous constatons que nous nous rapprochons des résultats de Tanagra avec un **Gain = 6462**.

Remarque : Dans ce cas précis, nous nous sommes appuyés sur des résultats connus de la littérature pour aiguiller correctement le paramétrage. Malheureusement, ces connaissances ne sont pas toujours disponibles avec une acuité qui permet d'agir efficacement. Nous sommes obligés de tâtonner, nous exposant aux dangers du « sur apprentissage »... sur le fichier test puisque nous l'utilisons dès lors pour identifier le paramétrage « optimal ».

4.4 La méthode « MetaCost + J48 »

La méthode MetaCost (Domingos, 1999) est disponible dans Weka. Nous actionnons le bouton CHOOSE, nous le sélectionnons dans le bloc META. Nous le paramétrons en définissant la matrice de coûts. Nous demandons 25 itérations pour obtenir des résultats comparables à ceux de Tanagra. Enfin, nous choisissons l'algorithme d'apprentissage J48 en veillant à élaborer un arbre sans post élagage et en utilisant un estimateur laplacien des probabilités.



Il reste à cliquer sur START. Nous obtenons un **Gain = 6720**, à rapprocher avec la méthode « MultiCost + C4.5 » de Tanagra (Gain = 6865). Il y a certes un écart (145) que l'on ne peut pas négliger. Mais positionnées parmi l'ensemble des méthodes, elles sont relativement proches.

5 Analyse avec R

Tanagra et Weka offrent des solutions clés en main. Dans la très grande majorité des cas, nous pouvons nous en contenter. Les initiés, eux, ont accès aux implémentations puisque le code source est libre. Mais encore faut-il pouvoir s'y retrouver dans le dédale des classes.

R propose une solution intermédiaire qui se révèle très avantageuse dans de nombreuses configurations. Le langage interprété est très puissant, il nous permet de programmer nombre de fonctionnalités avec un minimum de lignes de code. Les spécialistes peuvent décrypter facilement le

code source et introduire des éventuelles améliorations sans passer par une recompilation toujours hasardeuse de l'application complète.

Dans ce qui suit, nous décrivons quelques implémentations des méthodes mis en avant ci-dessus (correction des affectations avec une matrice de coût, bagging sensible aux coûts). L'algorithme d'induction sous jacent est RPART accessible via le package du même nom.

Chargement du package RPART. La première instruction consiste à charger le package adéquat avec l'instruction

```
library(rpart)
```

Définir la matrice de coût. Nous définissons une fois pour toutes la matrice de coûts de mauvaise affectation.

```
#définir la matrice de coût
cost.matrix <- matrix(c(-3.0,1.0,1.0,1.0,-6.0,1.0,0.0,0.0,0.0),nrow=3,ncol=3)
print(cost.matrix)
```

Chargement et partition des données. Nous devons ensuite charger les données et créer les sous échantillons d'apprentissage et de test.

```
#chargement des données
setwd("D:/DataMining/Databases_for_mining/concours-cup-etc/dm-cup-2007")
cup <- read.table(file="donnees-dm-cup-2007-full-dataset.txt",header=T,sep="\t",dec=".")

#partition sur la base de la colonne ID
cup.train <- cup[cup$ID=="train",2:length(cup)]
cup.test <- cup[cup$ID=="test",2:length(cup)]
```

Nous pouvons maintenant lancer les séquences apprentissage et test pour évaluer les différentes solutions.

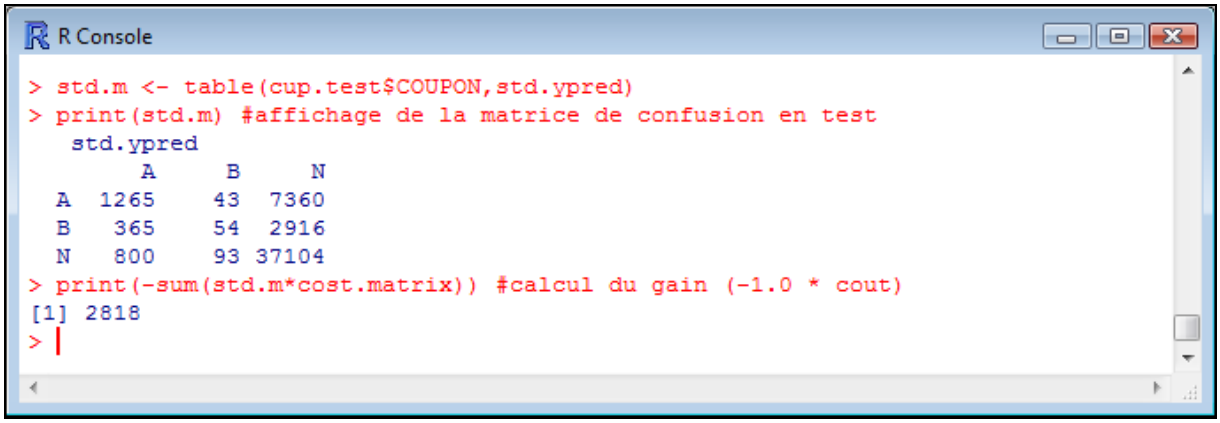
Apprentissage et test sans prise en compte des coûts. Dans un premier temps, nous construisons un arbre via la méthode RPART, nous l'appliquons sur la partie test en utilisant la méthode standard de prédiction.

```
#paramètres de rpart
#pas de validation croisée, et privilégier un arbre profond
parametres <- rpart.control(xval=0,cp=0,maxcompete=0,maxsurrogate=0)

#création de l'arbre de décision
std.modele <- rpart(COUPON ~ ., data = cup.train, method="class", control = parametres)

#prédiction sans prise en compte des coûts sur la partie test
std.ypred <- predict(std.modele,newdata = cup.test,type="class")
std.m <- table(cup.test$COUPON,std.ypred)
print(std.m) #affichage de la matrice de confusion en test
print(-sum(std.m*cost.matrix)) #calcul du gain (-1.0 * cout)
```

Ainsi, dans la copie d'écran ci-dessus : nous commençons par préciser les paramètres de l'algorithme avec **rpart.control(.)**, nous veillons à créer un arbre non élagué ; nous construisons le modèle de prédiction avec l'instruction **rpart(.)** ; nous créons la colonne des valeurs prédites avec **predict(.)** ; nous la croisons avec les valeurs observées avec **table(.)** pour obtenir la matrice de confusion ; reste à calculer le gain en faisant le produit entre cette dernière et la matrice de coûts. Nous obtenons **Gain = 2818**.

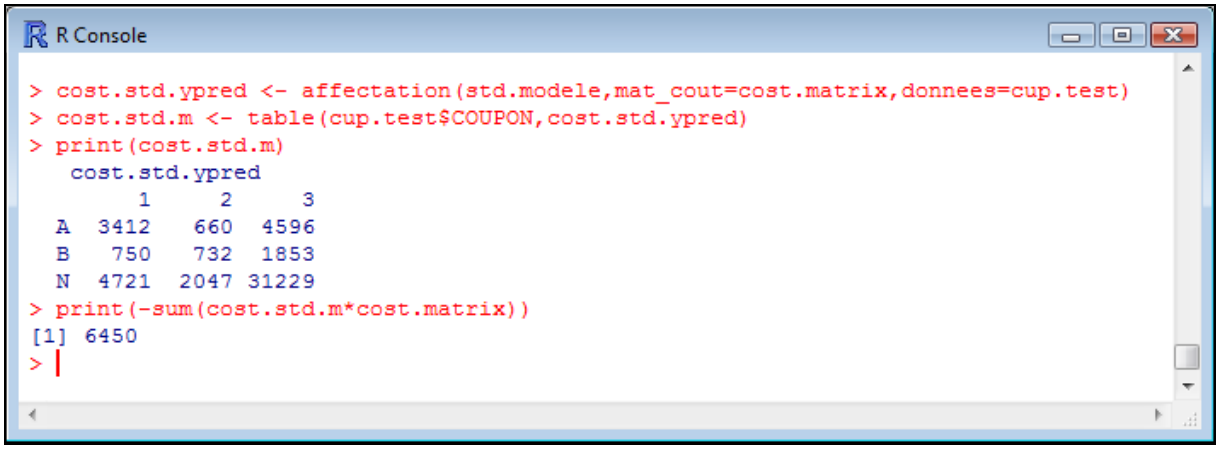


Affectation en tenant compte des coûts. Avec le même modèle, nous souhaitons maintenant produire une prédiction qui tient compte des coûts, en corrigeant le processus d'affectation pour chaque observation.

La procédure globale est la suivante.

```
#prédiction avec affectation tenant compte des couts
cost.std.ypred <- affectation(std.modele,mat_cout=cost.matrix,donnees=cup.test)
cost.std.m <- table(cup.test$COUPON,cost.std.ypred)
print(cost.std.m)
print(-sum(cost.std.m*cost.matrix))
```

Nous obtenons à la sortie un **Gain = 6450**.



Le rôle de la fonction **affectation(.)** est évidemment fondamental. Voyons en le détail :

D:\DataMining\Databases_for_mining\concours-cup-etc\dm-cup-2007\cost sensitive tree.r

```
#affectation basé sur les couts
#modele est l'arbre fourni par RPART
#mat_cout est la matrice de coûts
#donnees correspond à l'échantillon test
affectation <- function(modele,mat_cout,donnees){

  #produit entre le vecteur des probas
  #et une colonne de la matrice de la matrice de coût
  produit <- function(x,p){
    #somme du produit
    return(sum(p*x))
  }

  #prediction pour un individu
  #basé sur une matrice de coût
  #et les probas. d'affectation
  prediction <- function(p,mc){
    #recupérer dans un vecteur les couts d'affectation
    caff <- apply(mc,2,produit,p)
    #on renvoie l'indice de celui qui minimise les coûts
    return(which.min(caff))
  }

  #predict fournit les probabilités d'affectation pour chaque individu
  #avec ce "type" de paramétrage
  pred <- predict(modele,type="prob", newdata = donnees)

  #pour chaque individu, utiliser la prédiction basée sur les coûts
  ychapeau <- apply(pred,1,prediction,mc=mat_cout)

  #renvoyer le tout
  return(ychapeau)
}
```

Nous utilisons **predict(.)** sous cette forme (type = « prob ») afin de produire les probabilités conditionnelles. La fonction **prediction(.)** cherche la conclusion la moins coûteuse en calculant pour chaque modalité de la variable à prédire le **produit(.)** entre le vecteur de probabilité et la colonne adéquate de la matrice de coût.

« Cost sensitive » Bagging - Apprentissage. Nous nous proposons maintenant d'implémenter le bagging dans R. On ne tient jamais compte des coûts dans la phase de construction des modèles. Il s'agit simplement d'itérer l'apprentissage sur un échantillon du fichier d'apprentissage, tiré avec remise et avec une pondération uniforme.

D:\DataMining\Databases_for_mining\concours-cup-etc\dm-cup-2007\cost sensitive tree.r

```
#bagging sur les arbres
bagging_rpart <- function(y, formule, donnees, mat_cout, repetition){

  #pour avoir le même résultat à chaque exécution
  set.seed(15)

  #structure de liste qui va contenir tous les modèles générés
  tous <- NULL

  #vecteur de récupération des coûts
  cout <- NULL

  #effectif
  n <- nrow(donnees)

  #vecteur de poids
  poids <- rep(1/n,n)

  #paramètres de l'apprentissage rpart
  parametres <- rpart.control(xval=0,cp=0,maxcompete=0,maxsurrogate=0)

  #apprentissage
  for (k in 1:repetition){

    #échantillonnage avec remise, tous les individus ont le même poids
    echantillon <- sample(1:n,n,T,poids)
    #récupérer le sous-ensemble d'observations corresp.
    subsample <- donnees[echantillon,]
    #apprentissage
    modele <- rpart(formula = formule, data = subsample, method = "class", control = parametres)
    #collecte dans la liste
    tous[[k]] <- modele

    #prédiction du modèle sur les individus
    ychapeau <- affectation(modele,mat_cout,donnees)

    #calcul du cout
    mc <- table(y,ychapeau)
    cout[k] <- sum(mc*mat_cout)

  }
  #renvoyer les coûts associés à chaque itération
  return(list(couts=cout,modeles=tous))
}
```

« Cost sensitive » Bagging - Test. Dans la phase de déploiement, nous corrigeons les modèles individuels avec la matrice de coût pour définir l'affectation d'un modèle. Il nous reste alors à effectuer un vote simple de ces affectations pour réaliser la prédiction globale pour un individu.

D:\DataMining\Databases_for_mining\concours-cup-etc\dm-cup-2007\cost sensitive tree.r

```
#affectation par vote à la majorité simple
#à partir d'une liste de classifieurs
#pour un classifieur individuel, on utilise l'affectation basée sur les coûts
bagging_affectation <- function(liste,mat_cout,donnees){

  #calculer la fréquence des affectations pour un individu
  #et envoyer l'index de celui qui est majoritaire
  pred_on_vote <- function(x){
    #calculer la distribution
    frequence <- table(x)
    #renvoyer l'indice du max
    return(as.integer(names(which.max(frequence))))
  }

  #récupère une matrice d'affectation l'ensemble  des classifieurs
  allpred <- sapply(liste,affectation,mat_cout,donnees)

  #puis le prédiction par le vote à la majorité simple
  ychapeau <- apply(allpred,1,pred_on_vote)

  #renvoyer la prédiction
  return(ychapeau)

}
```

La mise en œuvre de ces procédures…

D:\DataMining\Databases_for_mining\concours-cup-etc\dm-cup-2007\cost sensitive tree.r

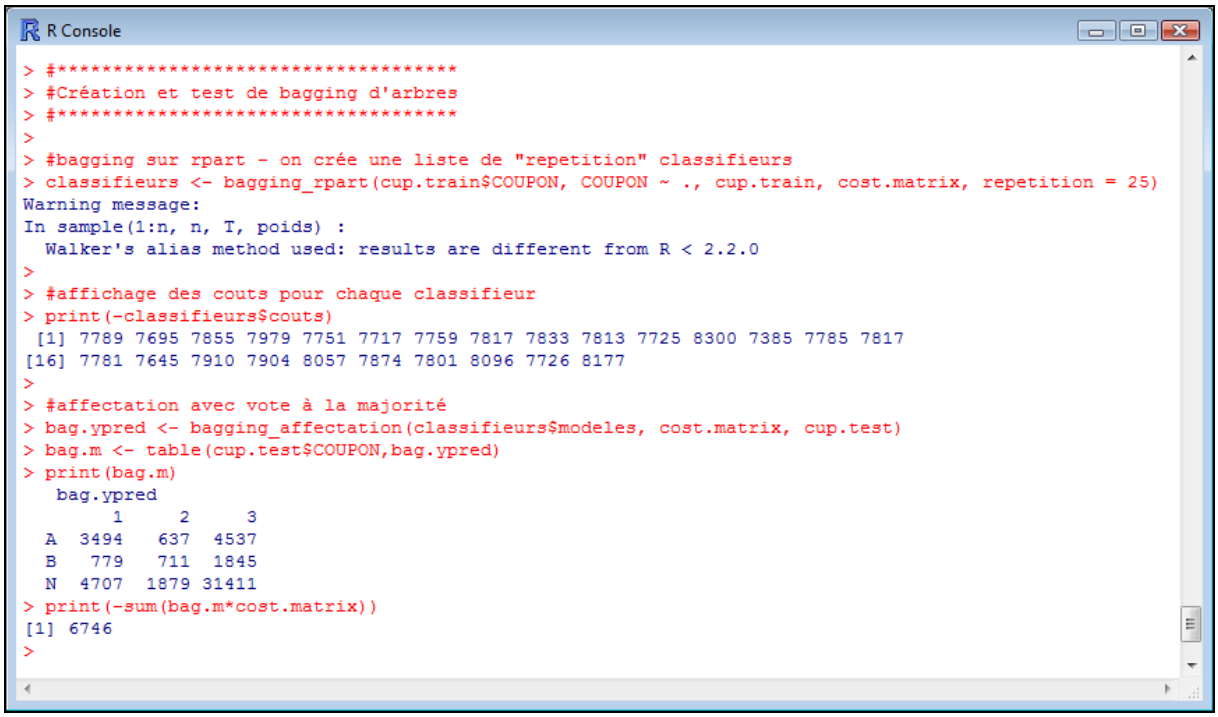
```
#************************************
#Création et test de bagging d'arbres
#************************************

#bagging sur rpart - on crée une liste de "repetition" classifieurs
classifieurs <- bagging_rpart(cup.train$COUPON, COUPON ~ ., cup.train, cost.matrix, repetition = 25)

#affichage des couts pour chaque classifieur
print(-classifieurs$couts)

#affectation avec vote à la majorité
bag.ypred <- bagging_affectation(classifieurs$modeles, cost.matrix, cup.test)
bag.m <- table(cup.test$COUPON,bag.ypred)
print(bag.m)
print(-sum(bag.m*cost.matrix))
```

… fournit les résultats suivants :



Nous obtenons **Gain = 6746**, tout à fait comparable, à stratégie égale, avec les résultats de Tanagra et de Weka.

6 Conclusion

Plutôt que de nous gausser sur les mérites de tel ou tel logiciel, d'autant plus que les paramétrages doivent certainement jouer un rôle non négligeable dès que l'on veut affiner les comportements, nous noterons surtout la remarquable cohérence des stratégies. Les groupes se détachent distinctement lorsque l'on récapitule les gains obtenus.

Method	Profit
CS-MC4	6990
MultiCost + C4.5	6865
CS-CART	6860
Bagging Cost Sensitive + C4.5	6856
Bagging Cost Sensitive + Rpart	6746
MetaCost (25) - J48 without pruning and with laplacian	6720
Cost Sensitive Learning + C4.5	6467
Cost Sensitive Learning + J48 without pruning and with laplacian	6462
Cost Sensitive Learning + Rpart	6450
C4.5	2860
J48	2823
Rpart	2818

Nous sommes loin du compte néanmoins si l'on se réfère aux résultats rapportés sur le site de la compétition « Data Mining Cup - 2007». Le meilleur des concurrents a, semble-t-il, obtenu un score de 7907, largement au-delà de nos résultats (http://www.data-mining-cup.com/2007//Wettbewerb/Preise/123098814 7/).

Nous n'avons pas accès au programme qui a été utilisé par ce concurrent. C'est un peu dommage car nous ne pouvons pas ainsi reproduire à l'identique, et par là vérifier, les performances annoncées. Nous avons eu accès néanmoins à un descriptif (en allemand – voir http://www-i6.informatik.rwth-aachen.de/web/Teaching/LabCourses/DMC/dmclab.php) des stratégies utilisées. Il apparaît que les meilleurs se sont surtout appuyés sur une combinaison massive de classifieurs (*jusqu'à 2000 si j'en crois l'article et ma traduction plus qu'approximative à l'aide de Google Traduction*). Il faudrait développer des applications spécifiques pour cela. En effet, on se heurte rapidement à des problèmes de gestion mémoire lorsque l'on augmente fortement le nombre de réplications avec les implémentations actuelles. J'imagine de plus que les expérimentations ont duré des journées entières, luxe que je ne peux plus m'offrir…