1 Subject

How to include a misclassification cost during the supervised learning process and the evaluation of the models. Comparing Tanagra, R and Weka.

Everyone agrees that taking into consideration the misclassification costs is an important aspect of the practice of Data Mining. For instance, diagnosing disease for a healthy person does not produce the same consequences as to predict health for an ill person. Yet despite its importance, the topic is seldom addressed, both from a theoretical point of view i.e. how to integrate cost during the evaluation of models (easy) and their construction (a little less easy); and from the practical point of view i.e. how to implement the approach in software.

A misclassification costs matrix is a matrix c (i, j) where the row is the observed values of the class attribute, the column is the predicted values. Usually, we have c (i, j) ≥ 0 if i ≠ j, misclassifying is costly; and c (i, i) = 0, a correct classification does not cost anything. But this is a bit restrictive. In fact, correct classification induces a gain or a negative cost, rather we will write c (i, i) ≤ 0. Quantifying the consequences of a good or bad classification is a task of the domain expert. We do not have to do so. However, we must take into account these consequences during the knowledge extraction process.

Using the misclassification cost during the classifier evaluation is easy. We make a cross-product between the misclassification cost matrix and the confusion matrix. We obtain an "expected misclassification cost" (or an expected gain if we multiply the result by -1). Its interpretation is not very easy. It is mainly used for the comparison of models.

Handling costs during the learning process is less usual. Several approaches are possible. The first, simplest, is to learn the classifier regardless of costs. It is unlikely that we have good results. But this configuration can serve as reference for the evaluation of other strategies.

The second is also very simple, yet effective. We learn the classifier without considering the costs, but we use a classification rule that minimizes the expected loss. We use the conditional probabilities provided by the model to calculate the loss associated with each decision. It then chooses the decision which minimizes expected loss. It is a generalization of the classical approach which minimizes the misclassification error rate. The main advantage of this approach is that we can use, without specific modifications, the results provided by the software.

Other approaches are often described in the literature. Many of them rely on an aggregation scheme using a more or less adaptive resampling approach (bagging or boosting). Although they are mostly successful, they have a major drawback: because we combine many classifiers, the interpretation of the results becomes difficult or impossible.

We note however a very interesting approach that tries to combine the advantages of aggregating classifiers and the final production of a single model. We use the resampling approach in order to re-label the class attribute. Then we build the final model on this new variable. This is the idea of the MetaCost (Domingos, 1999). We have coded a variant called MultiCost in Tanagra. In contrast to MetaCost, it takes into account the costs during construction of each classifier.

If the techniques exist, what is available in free software? We note that the tools that integrate in a natural way are very few. A large majority ignores the problem. Some tools incorporate a fragmented way. RapidMiner, for instance, deals only with binary class attribute with c(i,i) = 0. Thus, it seems that Weka is one of the few programs which provide easy to use tools for integration of costs. This led us to program new components to the cost sensitive supervised learning approaches for the version 1.4.29 of Tanagra.

4 kind of components are available: (1) a correction of the classification rule using the misclassification cost matrix; (2) an aggregation approach combining corrected models (Bagging); (3) MultiCost, a variant of MetaCost; (4) and various decision tree induction methods which include the misclassification cost, such as a variant of C4.5 (Chauchat & Rakotomalala, 2001; this method is also implemented into Sipina, http://data-mining-tutorials.blogspot.com/2008/11/cost-sensitive-decision-trees.html) or CART(Breiman and al., 1984).

In this tutorial, we show how to use these components of Tanagra on a real (realistic) dataset. We also programmed the same procedures in the R software (http://www.r-project.org/) to give a better visibility on what is implemented. We compare our results with those of Weka. The algorithm underlying our analysis is a decision tree. According to the software, we use C4.5, CART or J48.

2 Dataset

The problem is to allocate wisely discount coupons to customers. There are 3 types of coupons (A - coupon discount, B - Special coupon, N - no coupon). The company has a profit when a customer uses really a coupon. The cost of a good or bad assignment of a coupon can be summarized in the following matrix:

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

In other words, if we use the following notation for a confusion matrix

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

The goal of the process is to maximize the profit or gain (or to minimize the COST = -1 * PROFIT)

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

This dataset was used during the DATA MINING CUP 2007 competition (http://www.data-mining-cup.com/2007//Wettbewerb/Aufgabe/1230970673/). Participants had a file of labeled 50,000 examples. They must classify 50,000 other examples for which the descriptors were provided but not the class attribute. They should complete this column and return it to the organizers who could then compare the prediction with the true label. Winner was the one that maximized the profit on the unlabeled dataset.

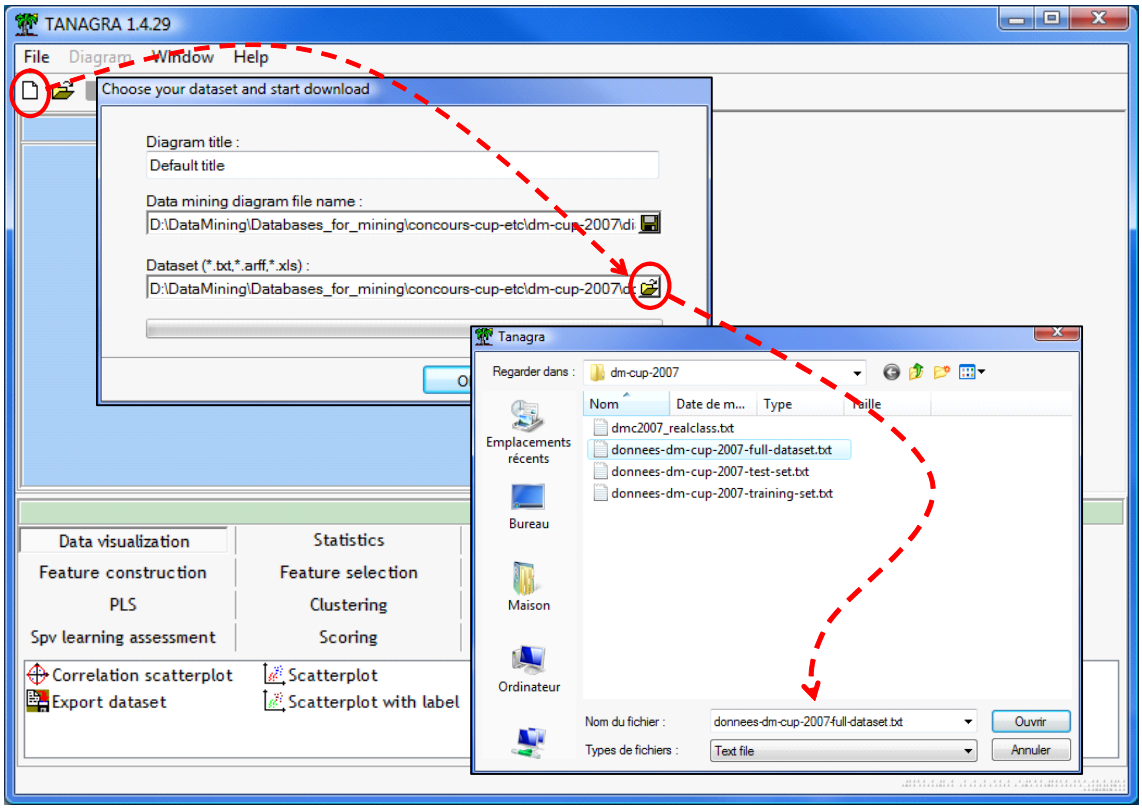
We are in a different configuration in this tutorial. Our data are divided into 2 parts, except that the second set - which is now labeled because the contest is finished - will be used as a test set. In our data file with 100,000 observations, we inserted an additional ID column with 2 values "TRAIN" and "TEST". In order to avoid biased comparison resulting to the optimization of the methods on the test set, we use, in the majority of cases, the default settings of the approaches.

The whole dataset (text file format) and the R source code program are available on line http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/dataset-dm-cup-2007.zip.

3 Tanagra

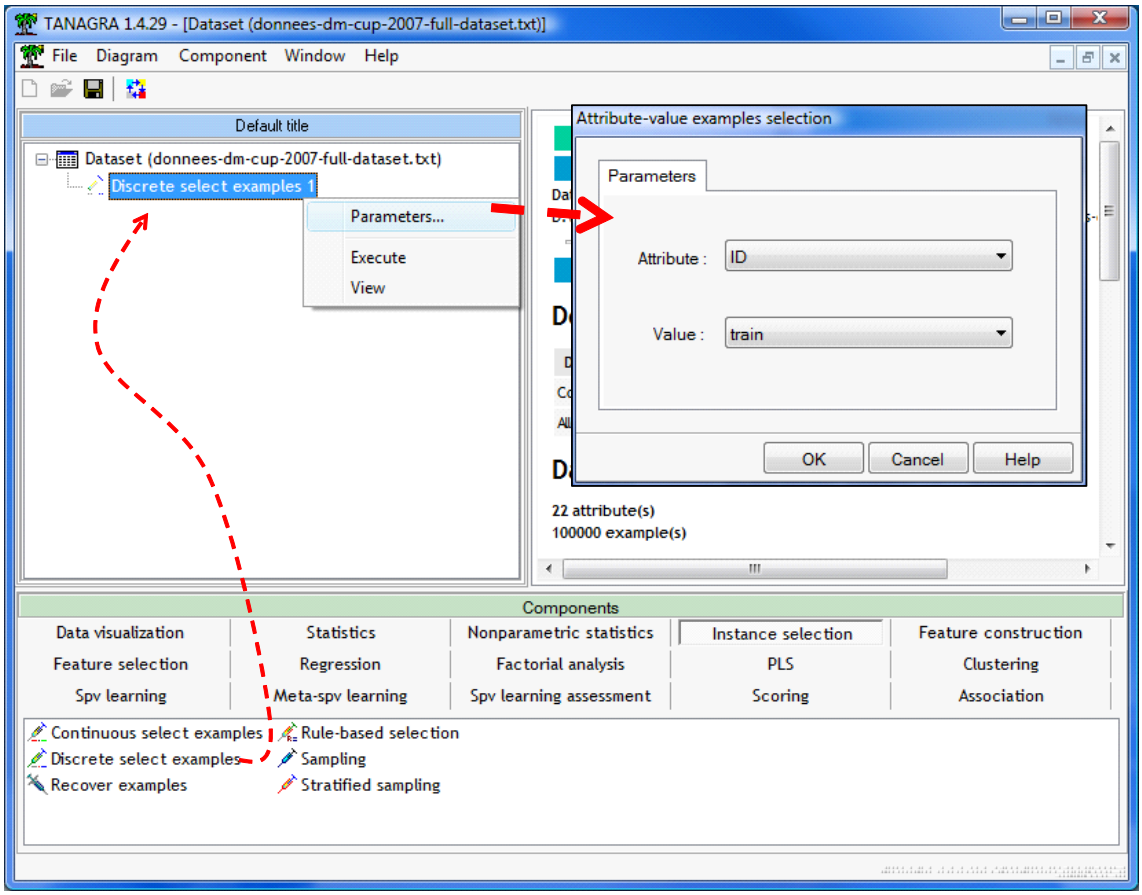
3.1 Creating a diagram and importing the dataset

We want to create a diagram and load the dataset. We activate the menu FILE / NEW, we select the data file « donnees-dm-cup-2007-full-dataset.txt ».

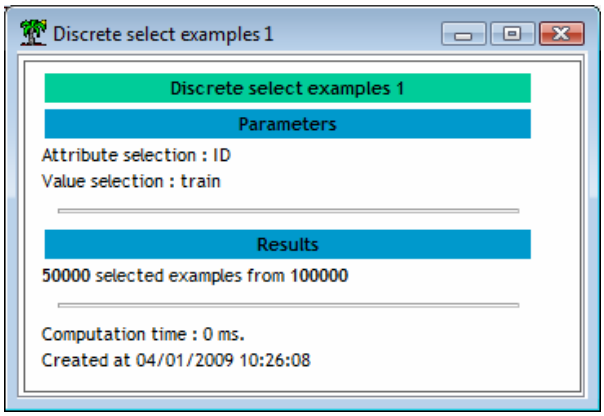


3.2 Partitioning the dataset into training and testing sets

We use the DISCRETE SELECT EXAMPLES component (INSTANCE SELECTION tab) in order to partition the dataset. We set the following parameters.

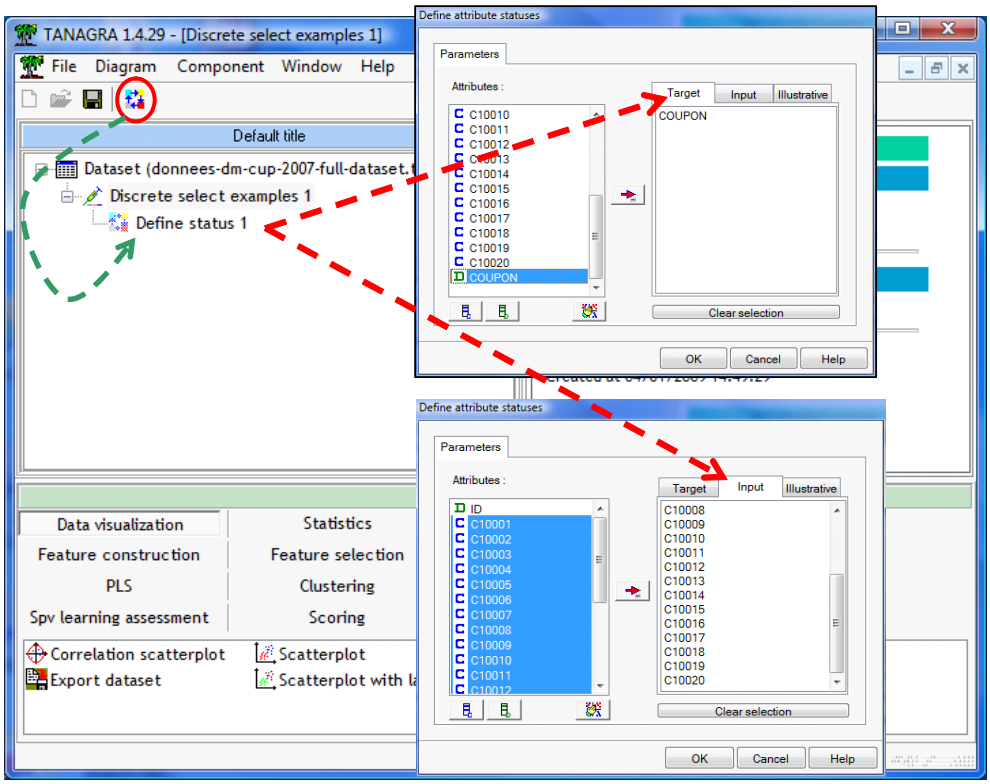


The learning set corresponds to ID = TRAIN. We click on VIEW menu. We have 50,000 examples for the construction of the classifiers.



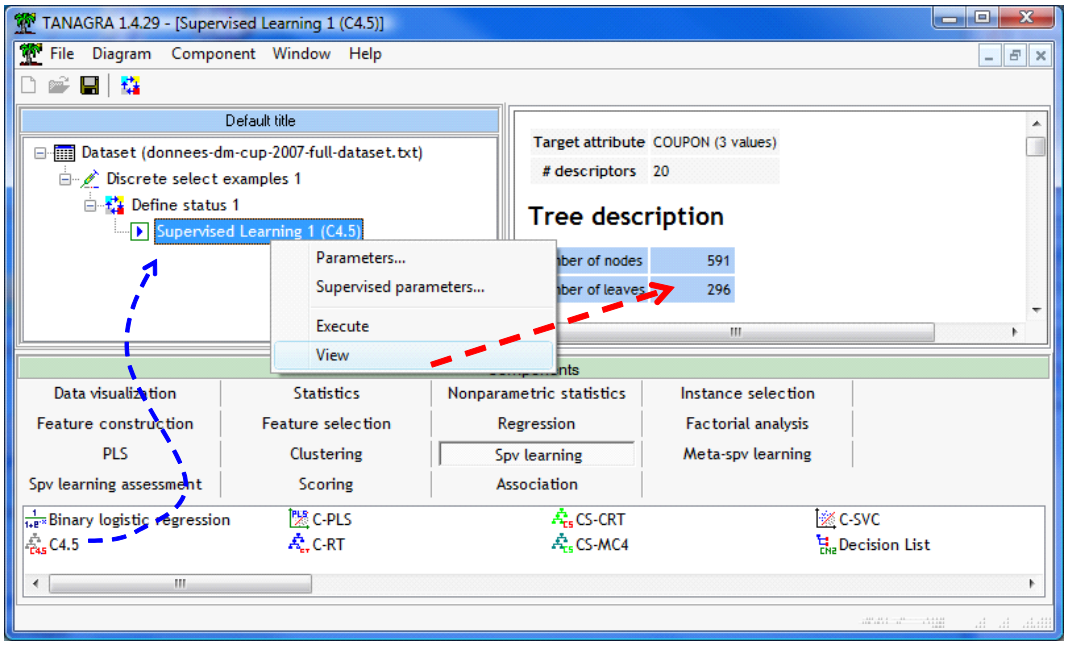
3.3 Type of variables

In the next step, we must set the TARGET (COUPON) and the INPUT variables (C10001 to C10020). We insert the DEFINE STATUS into the diagram, using the shortcut into the tool bar. The column ID is not used here.

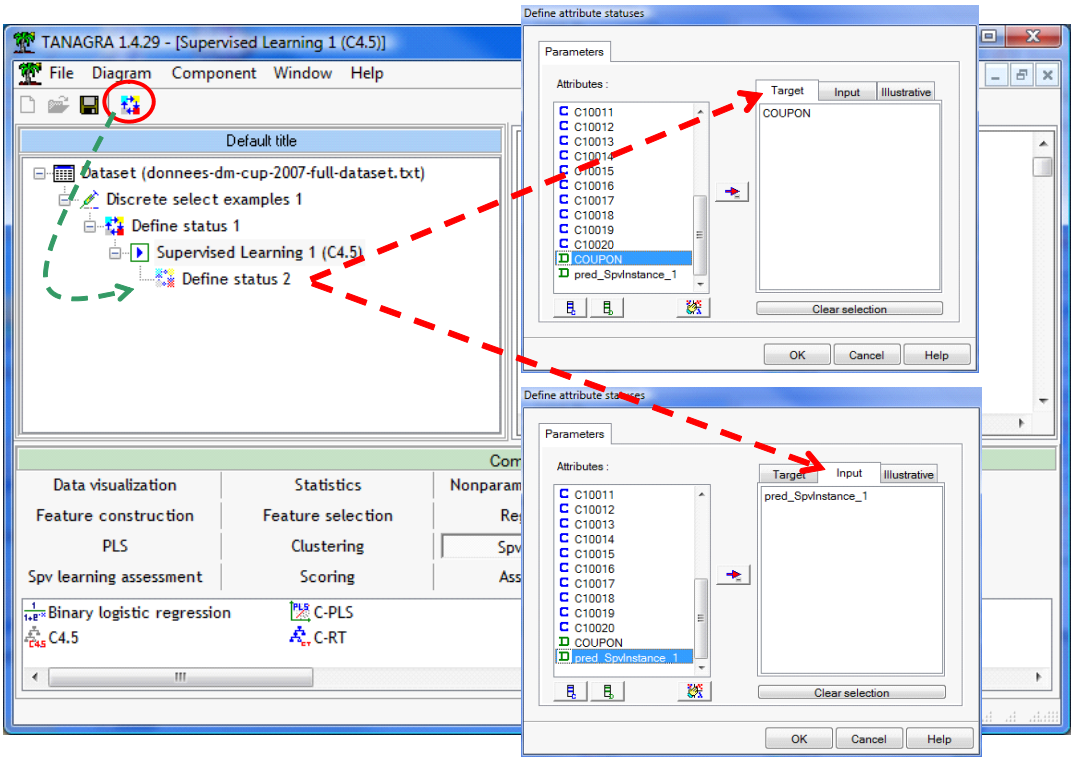


3.4 The standard C4.5 method

We want to evaluate the standard C4.5 approach, without accounting the misclassification cost matrix. This result will be used as reference for the evaluation of other techniques that will be presented thereafter. We insert the C4.5 component (SPV LEARNING tab) into the diagram. We click on the VIEW menu in order to obtain the results.



We obtain a tree with 296 leaves. We want now to evaluate this tree on the test set. We insert again the DEFINE STATUS component. We set the observed values COUPON as TARGET, the predicted values PRED_SPVINSTANCE_1 as INPUT.

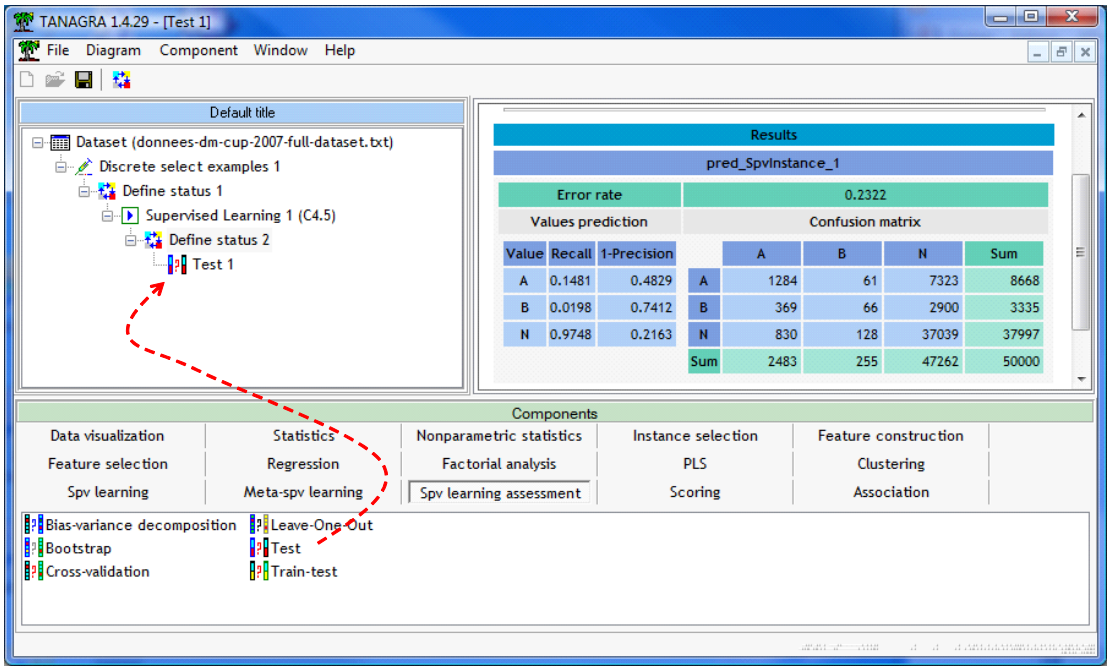


The TEST component (SPV LEARNING ASSESSMENT tab) allows comparing the observed and the predicted values. It uses automatically the unselected examples i.e. the test set. We click on the VIEW menu.

The error rate is 0.2322. But it is not really a relevant indicator in our context. We compute the PROFIT function, we obtain:

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

In the worst case, when we do not take into account the misclassification cost matrix, we obtain this profit. Let us observe the behavior of the other approaches which incorporate the costs during the learning phase.

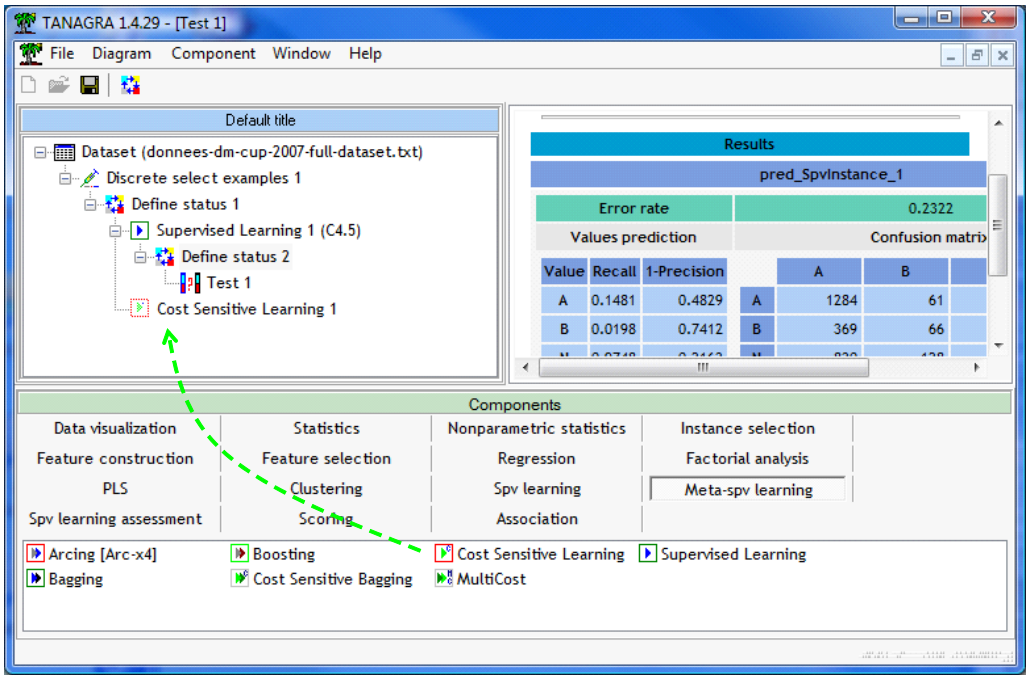


Value	Recall	1-Precision
A	0.1481	0.4829
B	0.0198	0.7412
N	0.9748	0.2163

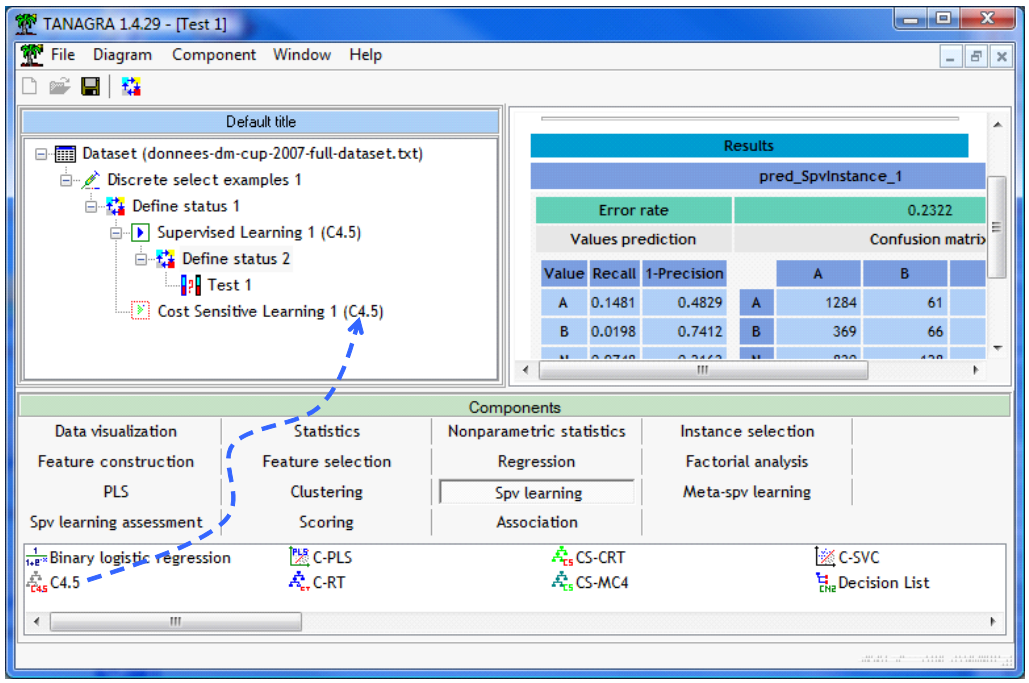
	A	B	N	Sum
A	1284	61	7323	8668
B	369	66	2900	3335
N	830	128	37039	37997
Sum	2483	255	47262	50000

3.5 Cost Sensitive Learning component + C4.5 decision tree algorithm

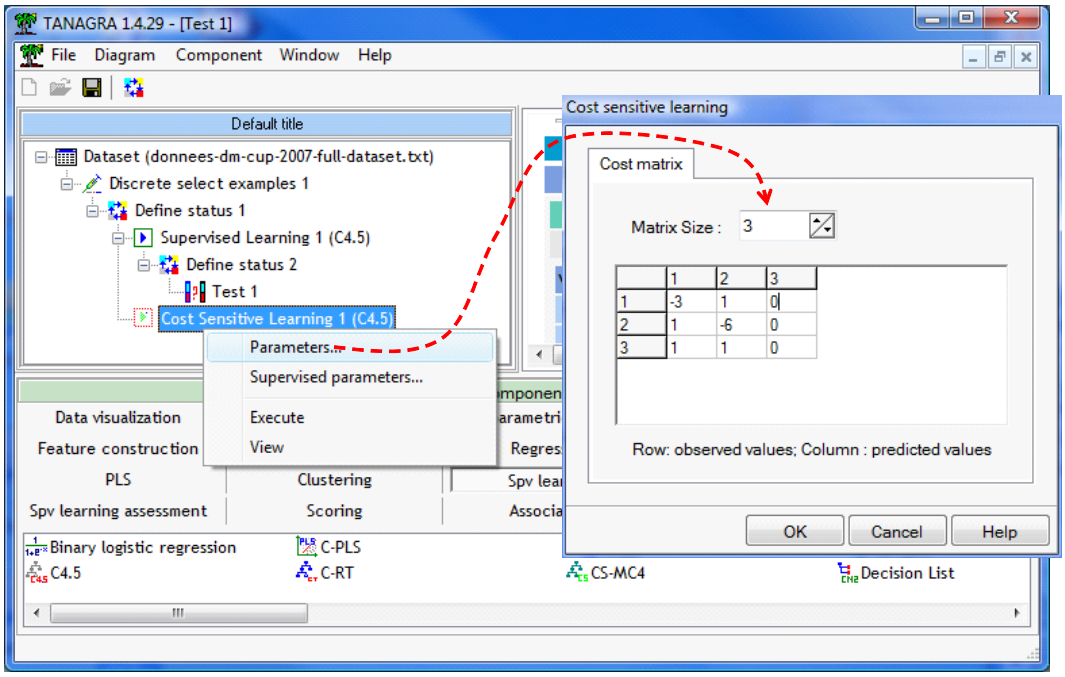
The second method that we implement is very simple. The learning algorithm is not modified. However, it is encapsulated in a framework which, when it classifies a new example, assigns the conclusion that minimizes the misclassification cost. This method uses the posterior probability estimation supplied by the classifier in order to compute the expected cost associated to each potential conclusion. The selected prediction is that which minimizes the cost.



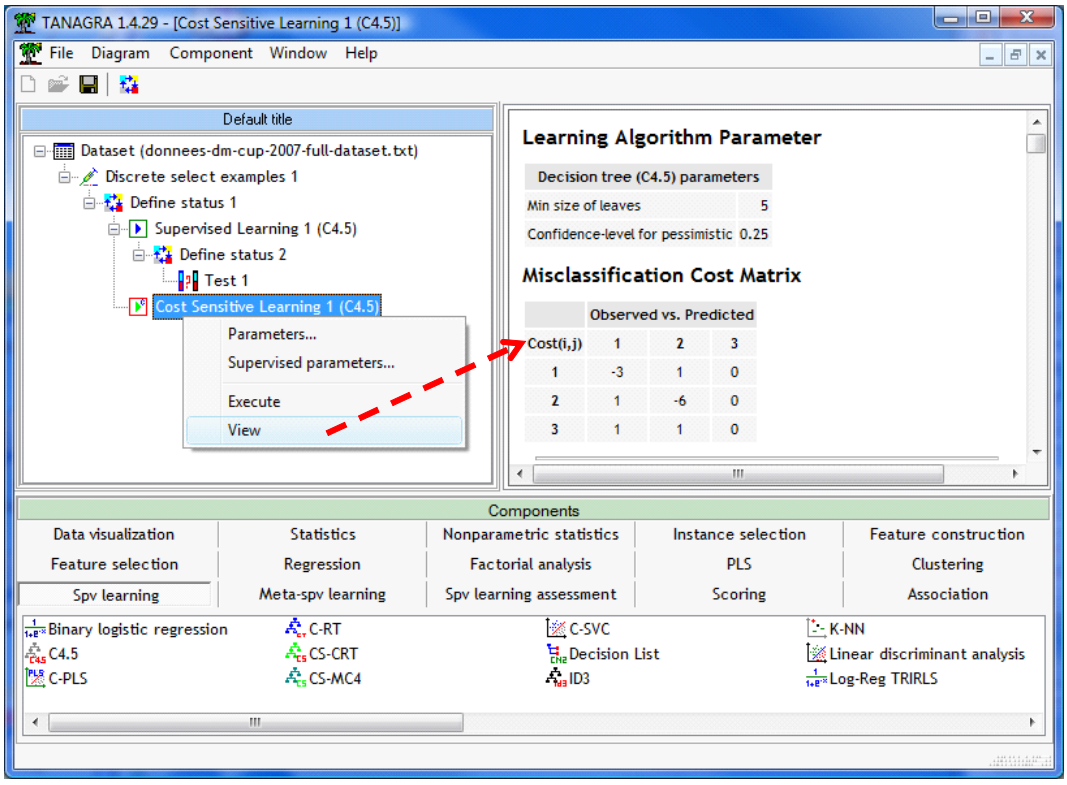
The method is implemented in two steps. First, we insert the generic component COST SENSITIVE LEARNING (META-SPV LEARNING tab) (see the screenshot above). Second, we insert the learning algorithm (SPV LEARNING tab). We use the C4.5 approach here, but actually we can use any supervised learning method.



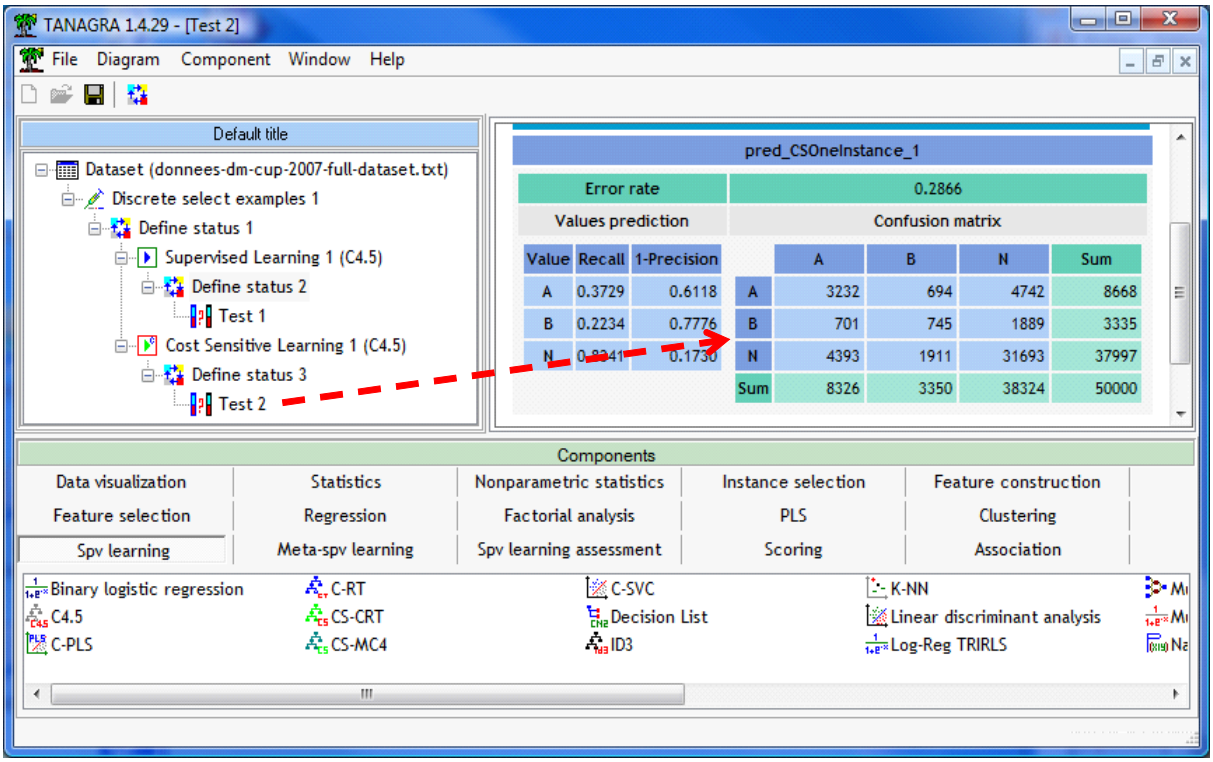
We must set now the misclassification cost matrix. We click on the PARAMETERS menu (*Note: the SUPERVISED PARAMETERS menu allows handling the settings of the encapsulated learning algorithm e.g. the C4.5 algorithm in this tutorial*).



We still have to activate the VIEW menu to access the result. The tree is completely identical to the previous one, unlike a cost matrix is now used during the classification phase, especially when we classify the examples of the test set.



Let us to evaluate the performance of the approach on the test set. We use again the TEST component. Into the DEFINE STATUS component, we set COUPON as TARGET, and the prediction of the learning method (PRED_CSONEINSTANCE_1) as INPUT.



We obtain the confusion matrix. We can compute the profit

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

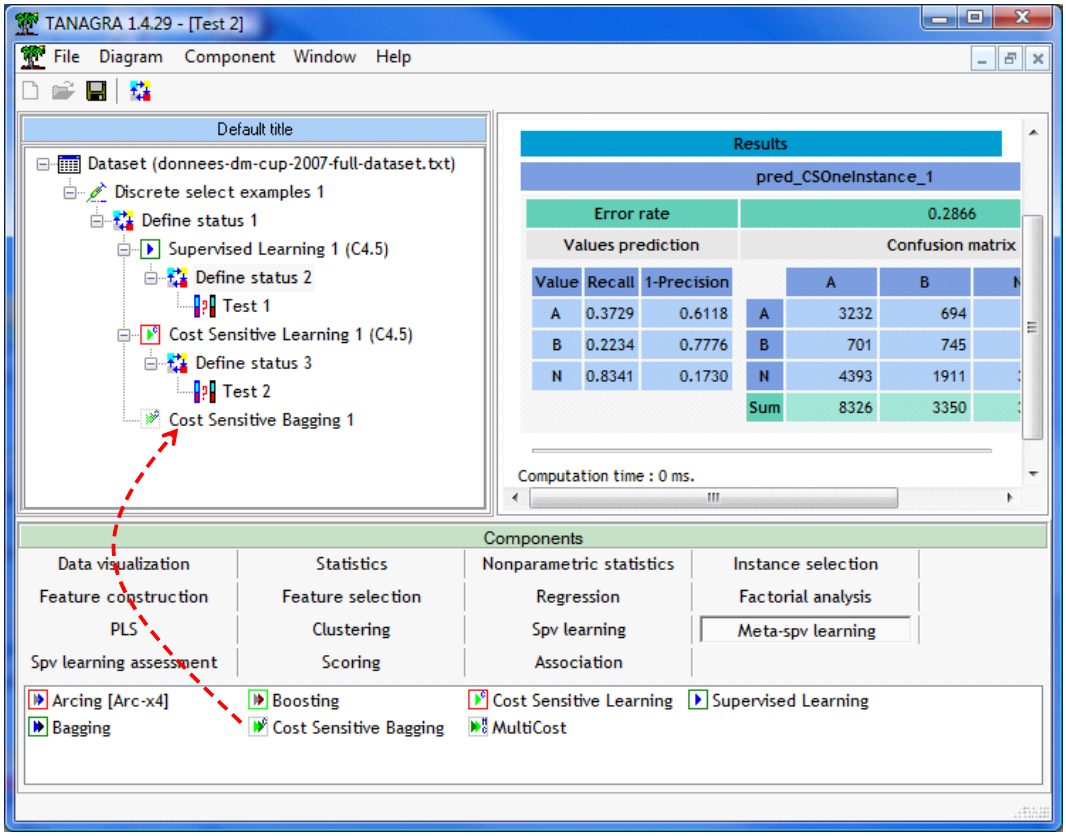
In comparison of the previous approach (ignoring completely the misclassification cost matrix), the performance is dramatically increased. Handling the costs during the classification phase, without modification of the learning algorithm is already an excellent way to implement a cost sensitive classifier.

3.6 Cost Sensitive Bagging + C4.5

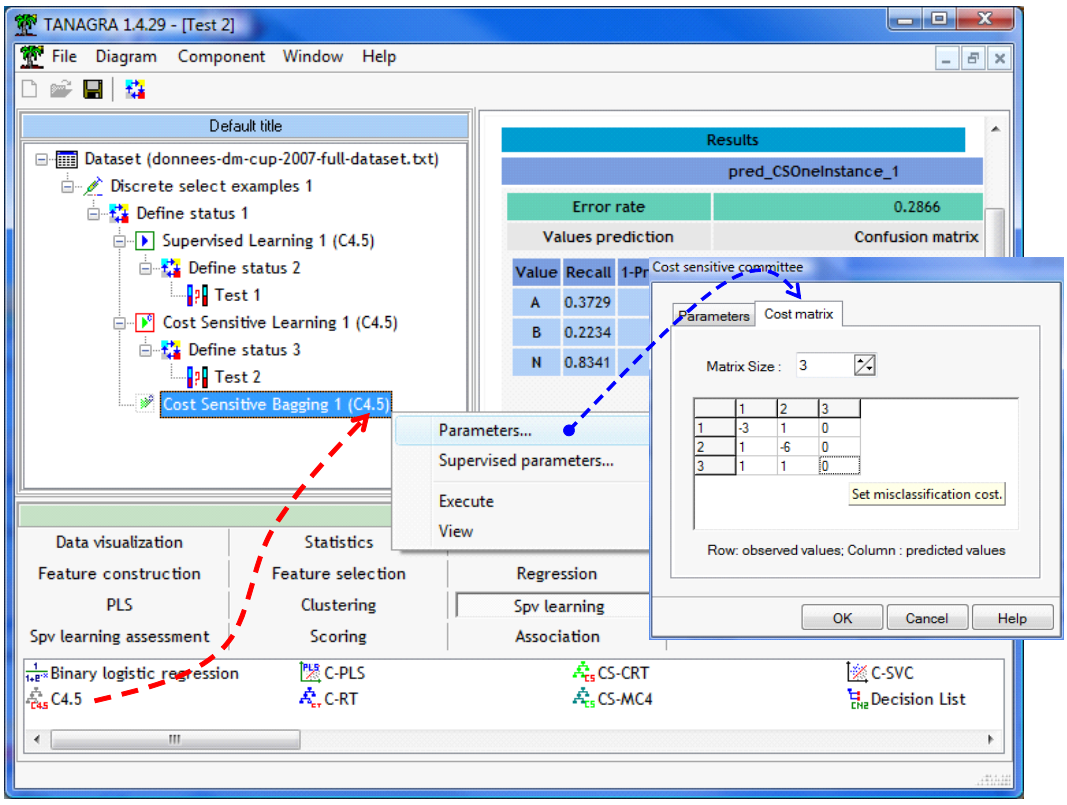
Aggregating approach is well known way to improve the performance of classifiers. Tanagra puts forward the COST SENSITIVE BAGGING component (META-SPV LEARNING tab).

It operates by two steps. First, using a resampling approach, it learns many individual cost-sensitive classifiers according the previous approach. Second, it uses the available classifiers for a better estimation of the posterior probabilities according to a voting scheme. A cost sensitive classification (minimizing the cost) is then performed when it classifies a new instance.

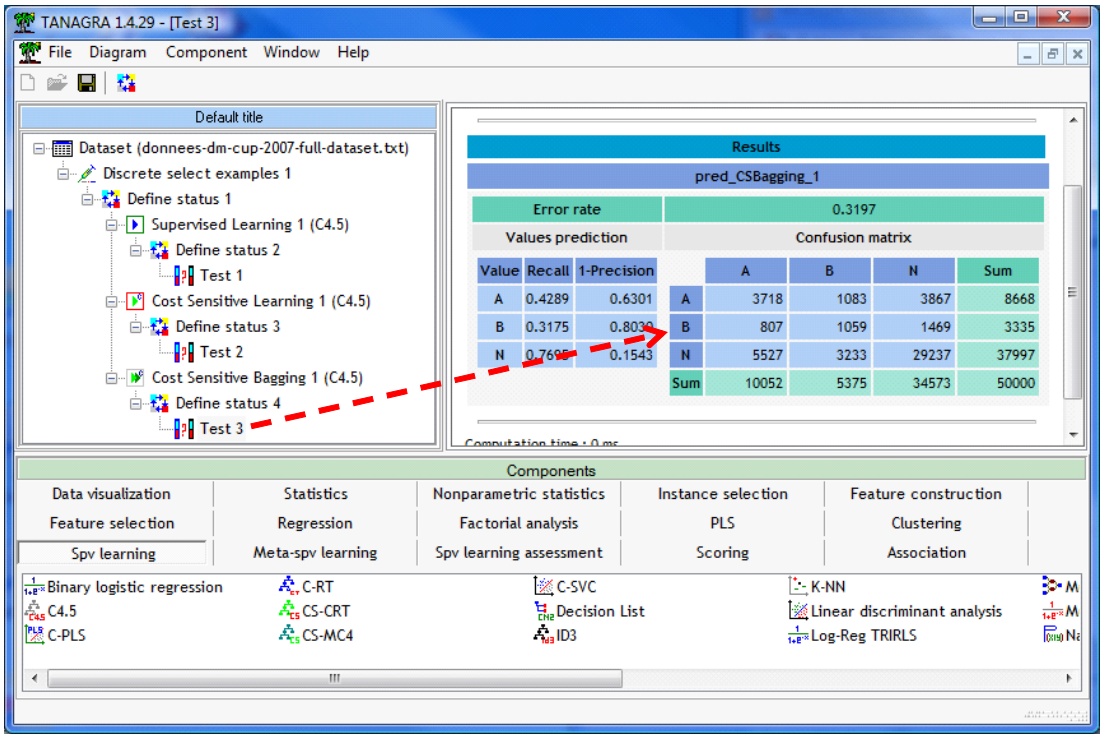
We insert the COST SENSITIVE BAGGING into the diagram.



Then, we embed the learning method (C4.5). We can use any learning method here. We set the parameters by clicking on the PARAMETERS menu. We set the misclassification cost matrix. The default number of replications is 25.



We click on the VIEW menu. In addition to the overall confusion matrix, we obtain the cost (says "Error rate" into the output) for each classifier, computed on the train set. We use again the TEST component in order to evaluate the performance of the approach. We compare COUPON to PRED_CSBAGGING_1.



The gain is

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

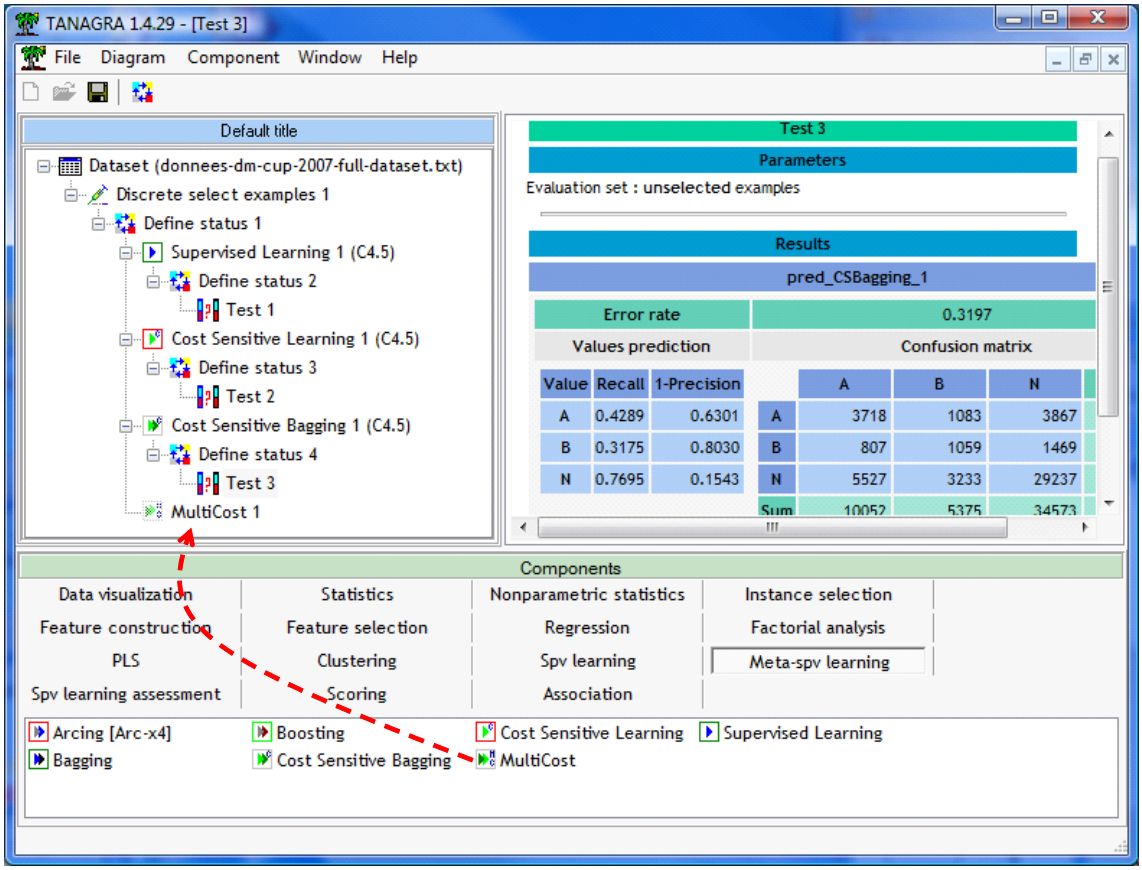
The improvement is less dramatic compared to the previous method (6858 vs. 6467 = 391). But it seems still significant.

3.7 MultiCost + C4.5

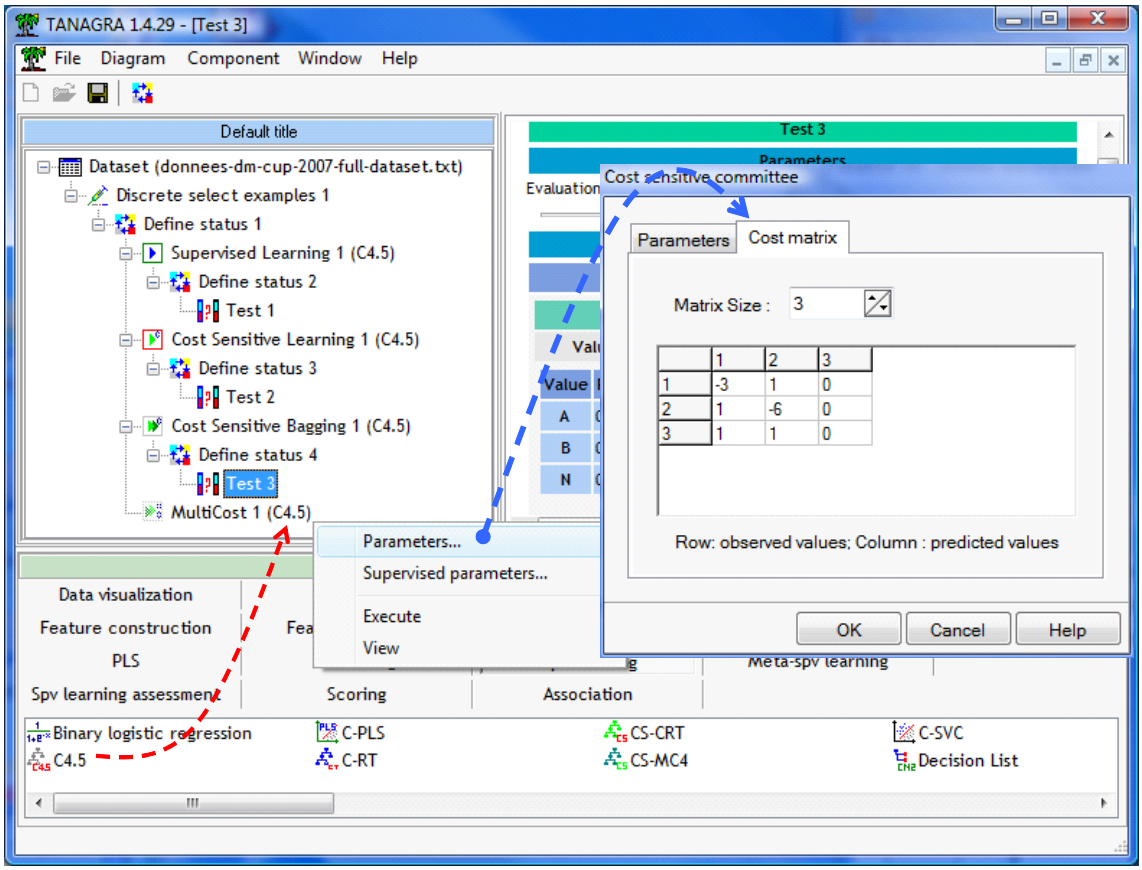
The disadvantage of the previous approach is that we have a set of models. The classification process for an individual is not readable; the interpretation of the results is difficult. Multicost is an attempt to alleviate this drawback.

MultiCost is a variant of MetaCost (Domingos, 1999). There are 3 steps in the process: (1) we create a set of models using the previous Bagging approach; (2) we create a new variable with predicted values according to the set of models (the classification process takes into account the misclassification cost); (3) we use this new column as a class attribute in a learning process, thus we obtain the final classifier. In contrast to MetaCost, the costs are already included in individual models (before the vote) in MultiCost.

We insert the MULTICOST component (META-SPV LEARNING tab) into the diagram.



Then we embed the C4.5 method. We click on the PARAMETERS menu to set the costs.



We click on the VIEW menu. In relation to the previous approach, we obtain a single model that we can interpret. We insert again the TEST component. We compare COUPON to PRED_MULTICOST_1 (Figure 1). We obtain

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

The gain is close to the BAGGING approach. But the main advantage here is that we manipulate a single classifier for the generalization process.

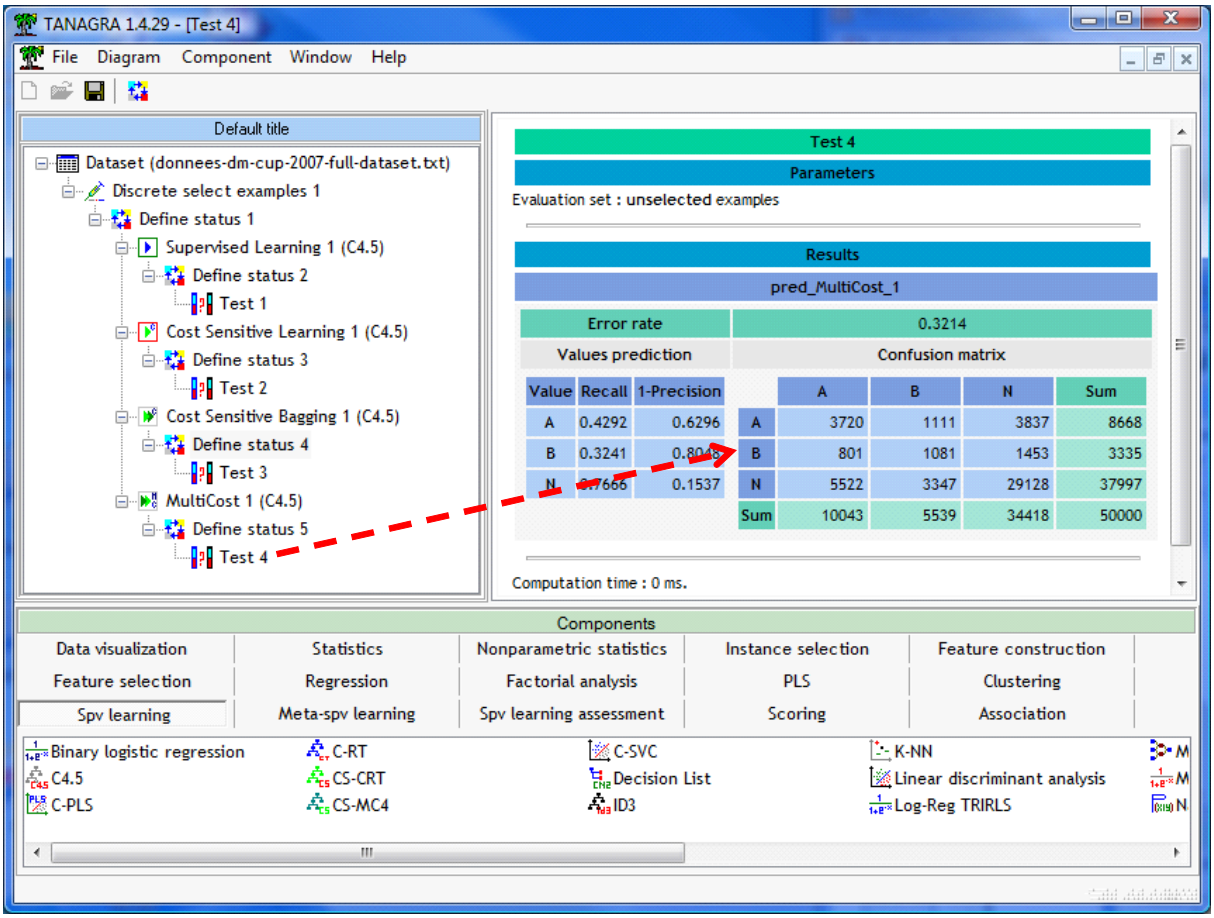
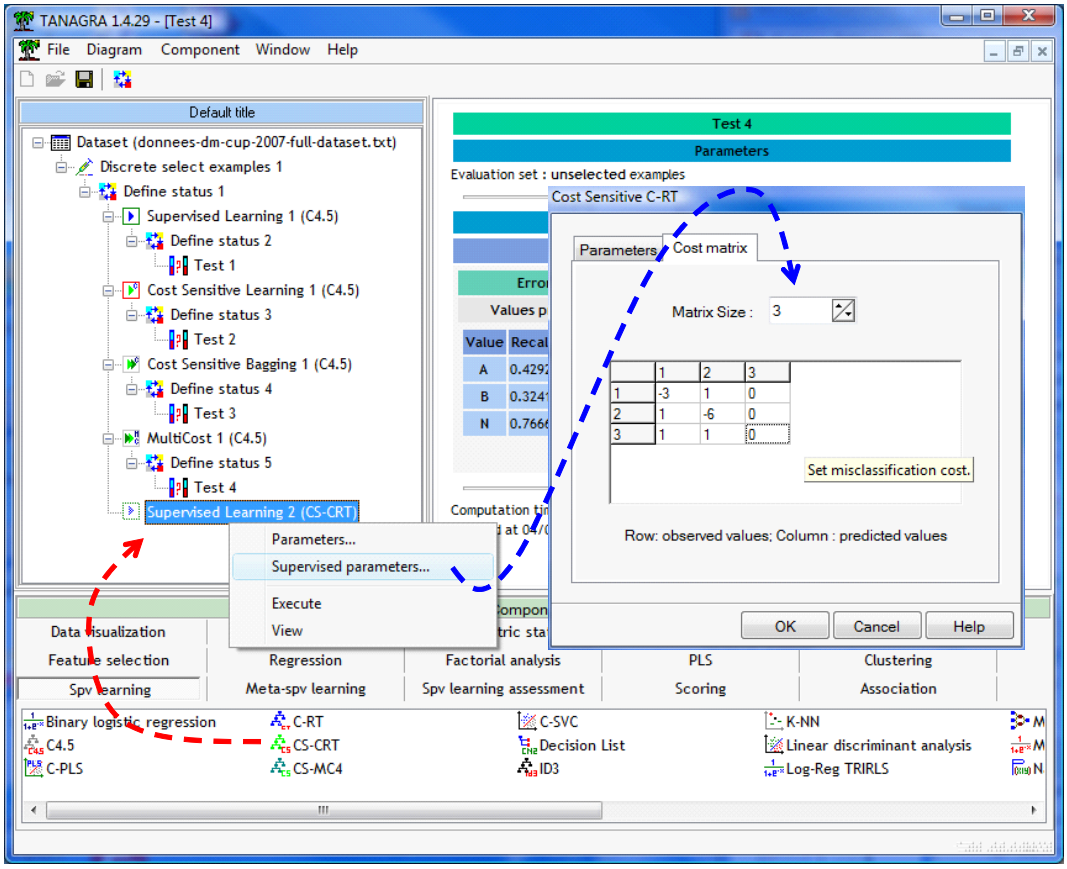


Figure 1 - Confusion matrix on the test set - MultiCost + C4.5

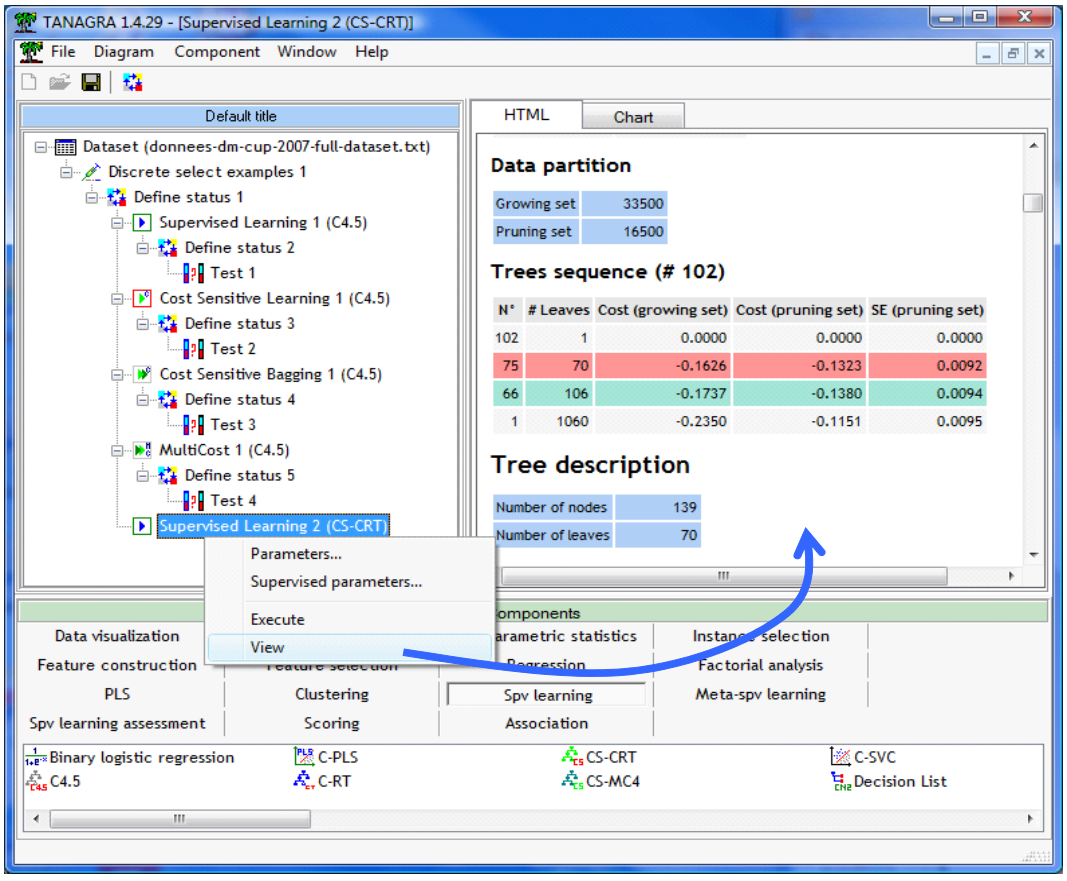
3.8 CART – Decision tree induction approach

Until now, the components act as a wrapper which modifies the results of a learning method. This last one is used as is, without modification. From now, we study methods which incorporate the misclassification cost matrix into the learning process, during the construction of the classifier. This is the case of the CART decision tree induction method. It can use the misclassification cost matrix during the post-pruning phase (Breiman and al., 1984; see section 11.4, pages 303 - 306).

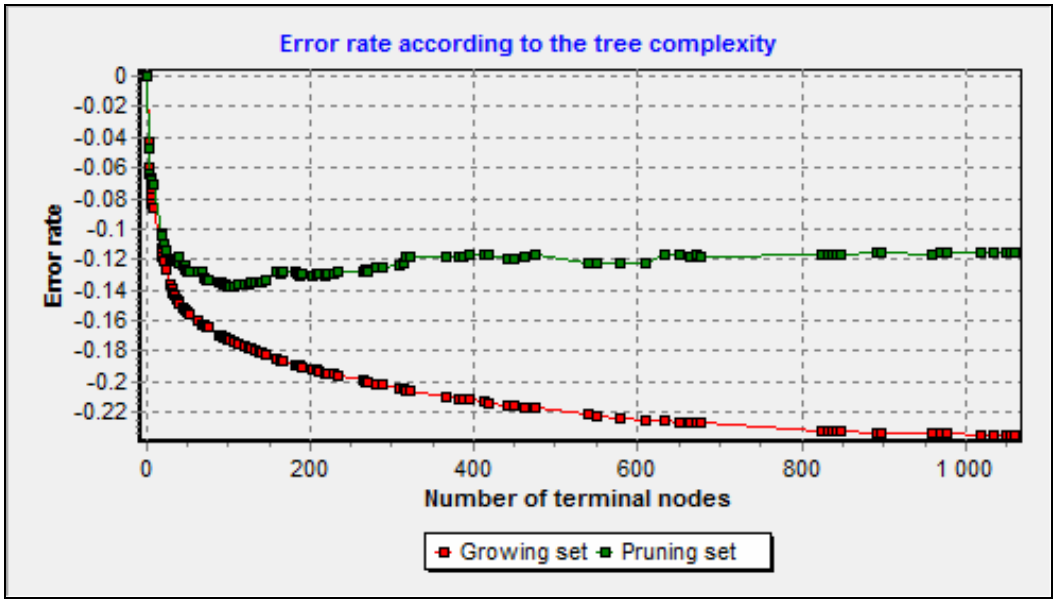
We insert the CS-CRT component (SPV LEARNING tab) into the diagram. We set the misclassification cost matrix using the SUPERVISED PARAMETERS menu. We use this menu because the handling of the costs is now embedded into the learning method.



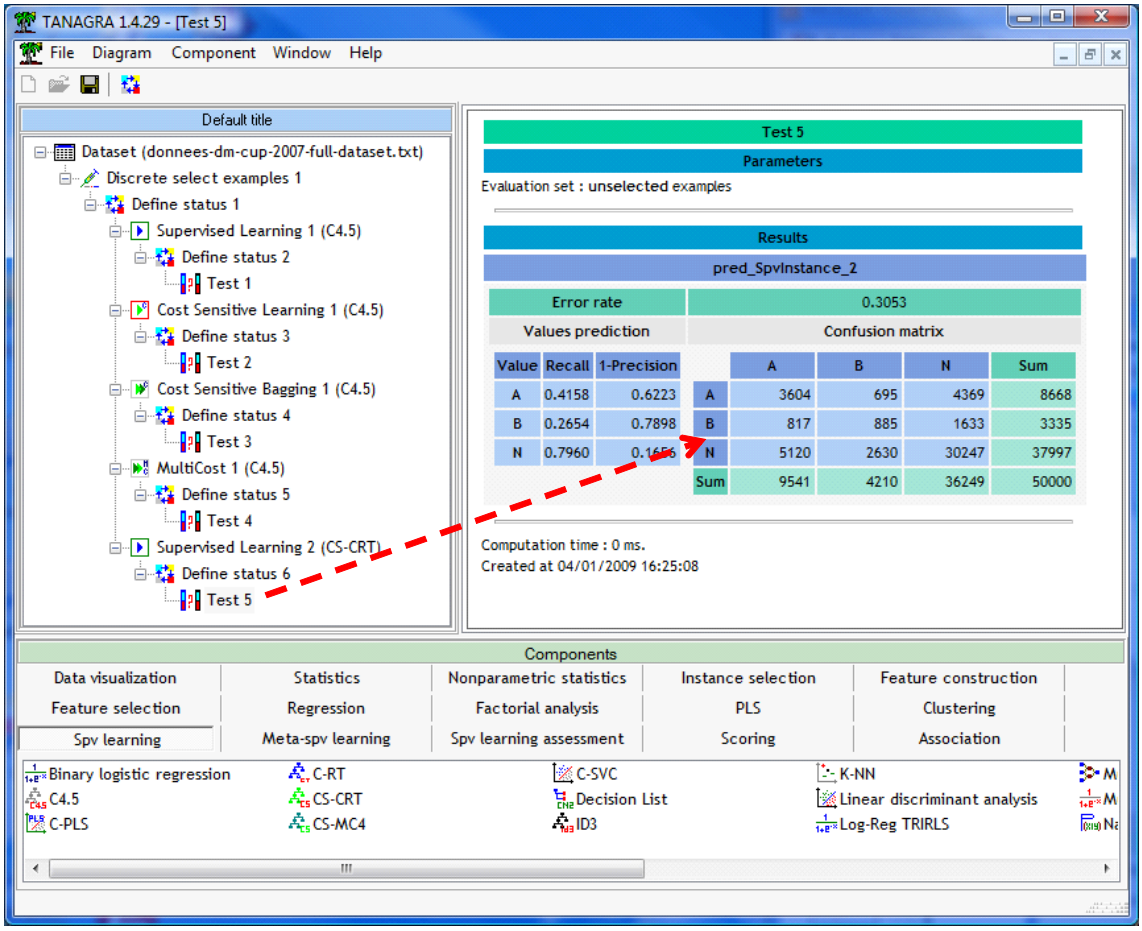
We obtain a tree with 70 leaves.



Into the CHART tab of the visualization window, we see that the pruning process cuts down efficiently the tree (the green curve, computed on the pruning set; the red curve is computed on the growing set; growing + pruning = learning set).



Let us see the performance on the test set (comparison of COUPON and PRED_SPVINSTANCE_2)



The gain is

Gain = 3 * 3604 + 6 * 885 – 1 * (817 + 5120 + 695 + 2630) = **6860**

It is very similar to MultiCost, even if the structure of the error is not the same.

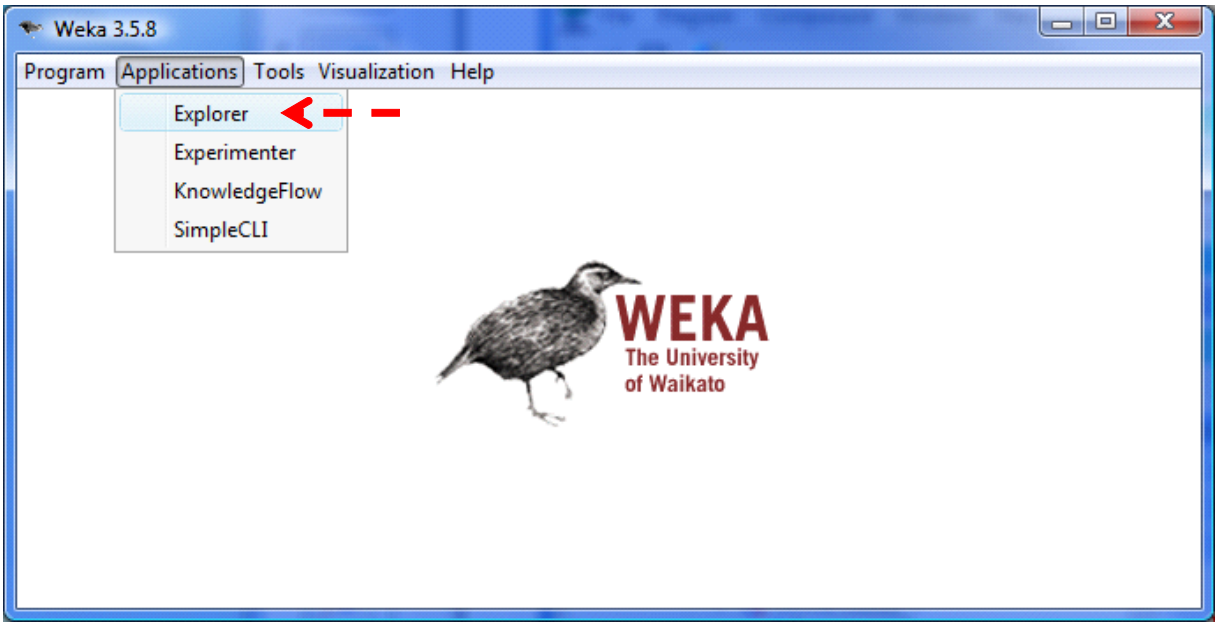
Note: Tanagra supplies another cost sensitive decision tree algorithm. It is the **MS-MC4** component. It is very similar to « *Cost Sensitive C4.5 (Chauchat & Rakotomalala, 2001)* » available into SIPINA (http://eric.univ-lyon2.fr/~ricco/sipina.html). It is described in some tutorials e.g. http://data-mining-tutorials.blogspot.com/2008/11/cost-sensitive-decision-trees.html. On our dataset, this method reaches **Gain = 6990**. The tree comprises 46 leaves.

4 Weka

Weka is one of very few programs to offer a relatively easy to use system to incorporate the misclassification costs. It can also handle negative costs (gains).

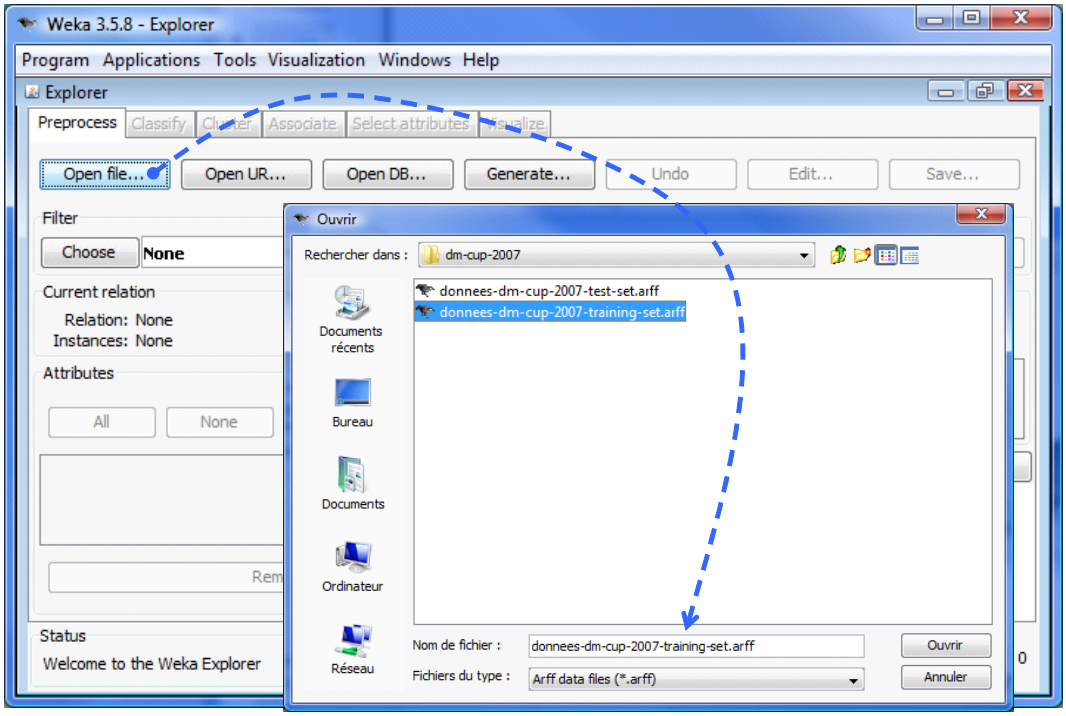
Weka cannot subdivide a data file from an ID column. We must split the data file manually: « donnees-dm-cup-2007-training-set.arff » and « donnees-dm-cup-2007-test-set.arff ». ARFF is the Weka file format.

There are various ways to use Weka. We use the EXPLORER module here. We click on the APPLICATIONS / EXPLORER menu.



4.1 Importation of the training set

We click on the OPEN FILE button in order to load the training set. We select « donnees-dm-cup-2007-training-set.arff ».

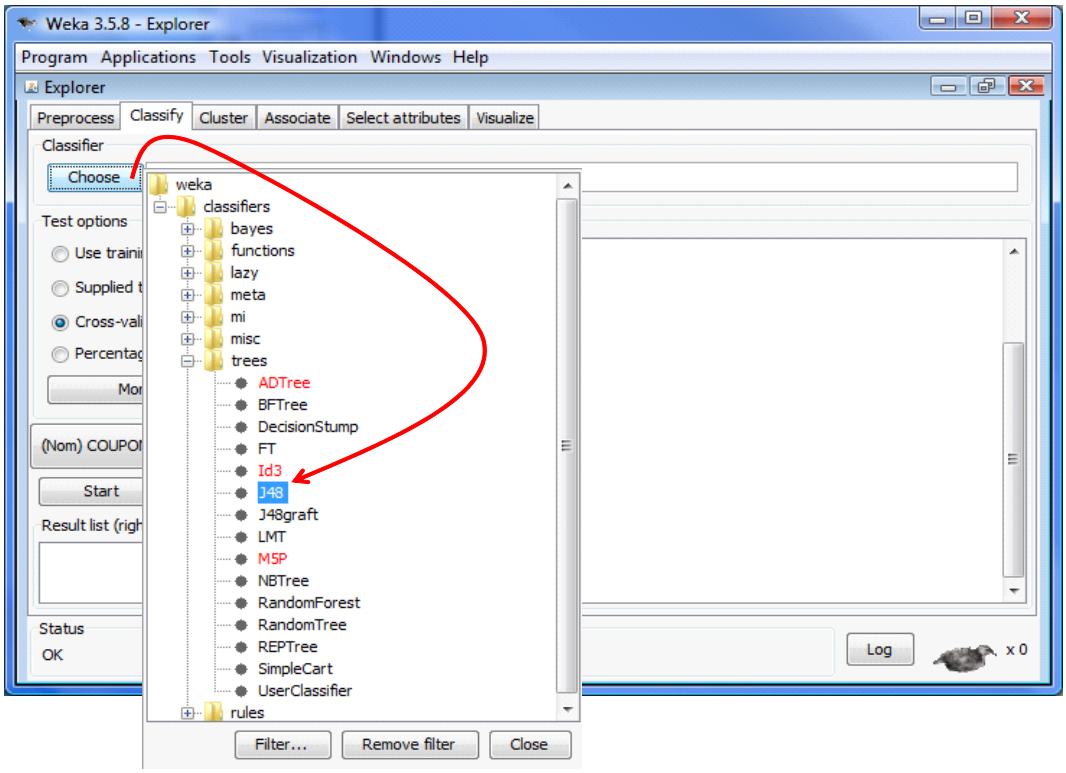


By default, the last column is the class attribute for Weka. We have the correct configuration.

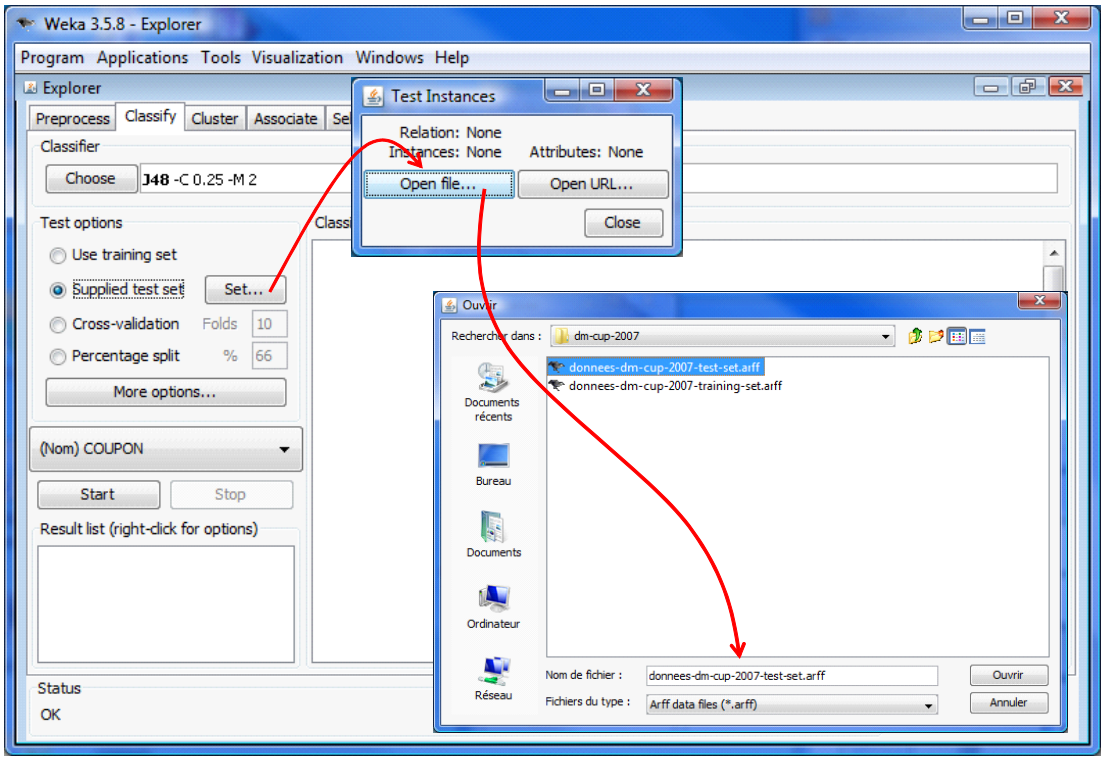
4.2 J48 without cost handling

J48 is a variant of C4.5. We implement this approach.

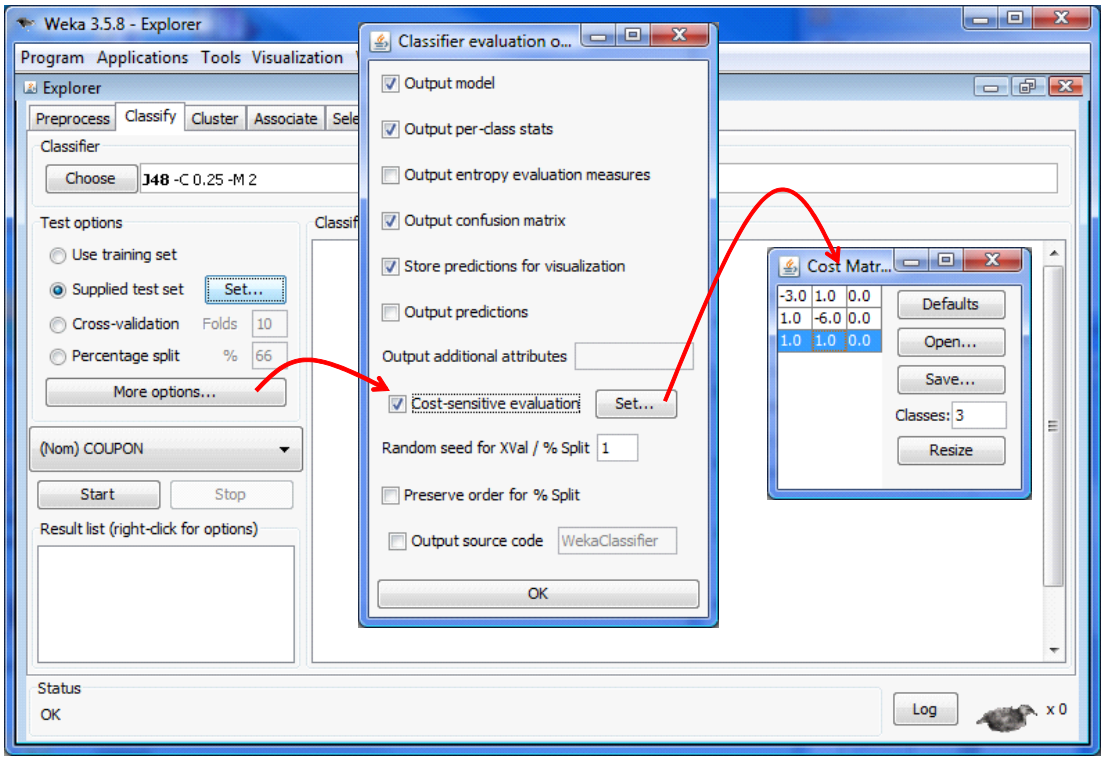
Choosing the method. In order to select the learning method, we select the CLASSIFY tab. Then, we click on the TREES node and the J48 method. We use the default settings.



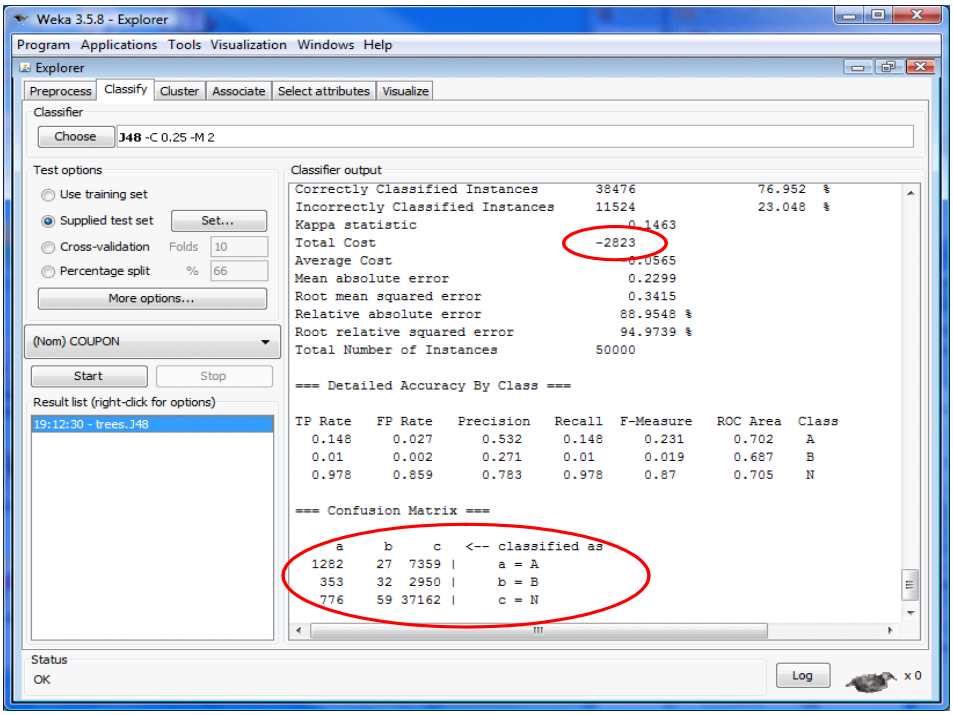
Selecting the test set. We click on the SUPPLIED TEST SET button in order to define the test set. We select the « donnees-dm-cup-2007-test-set.arff » data file.



Using the costs during the evaluation of the classifier. Weka can compute the cost function if we supply the misclassification cost matrix. We click on the MORE OPTION button. In the dialog box, we select the COST-SENSITIVE EVALUATION option. By clicking on the SET button, we insert the costs.

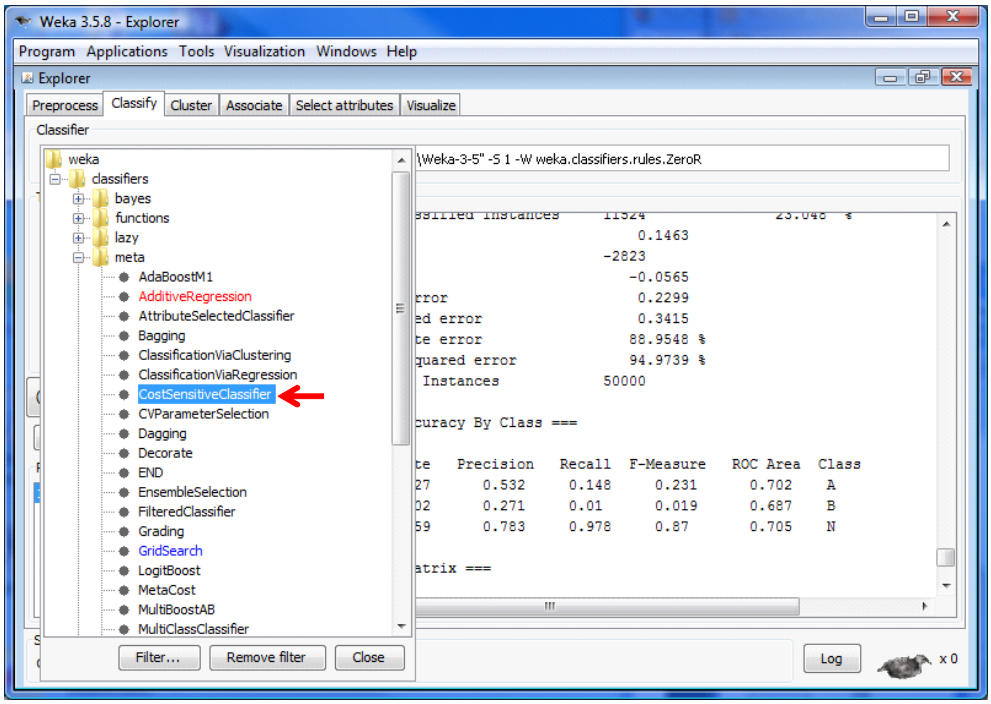


Train and test. We click on the START button in order to create the classifier. Using the J48 approach, Weka creates a tree with 213 leaves. The computed cost is -2823 i.e. we obtain **Gain = 2823**, very similar to C4.5 implemented with Tanagra (2860).

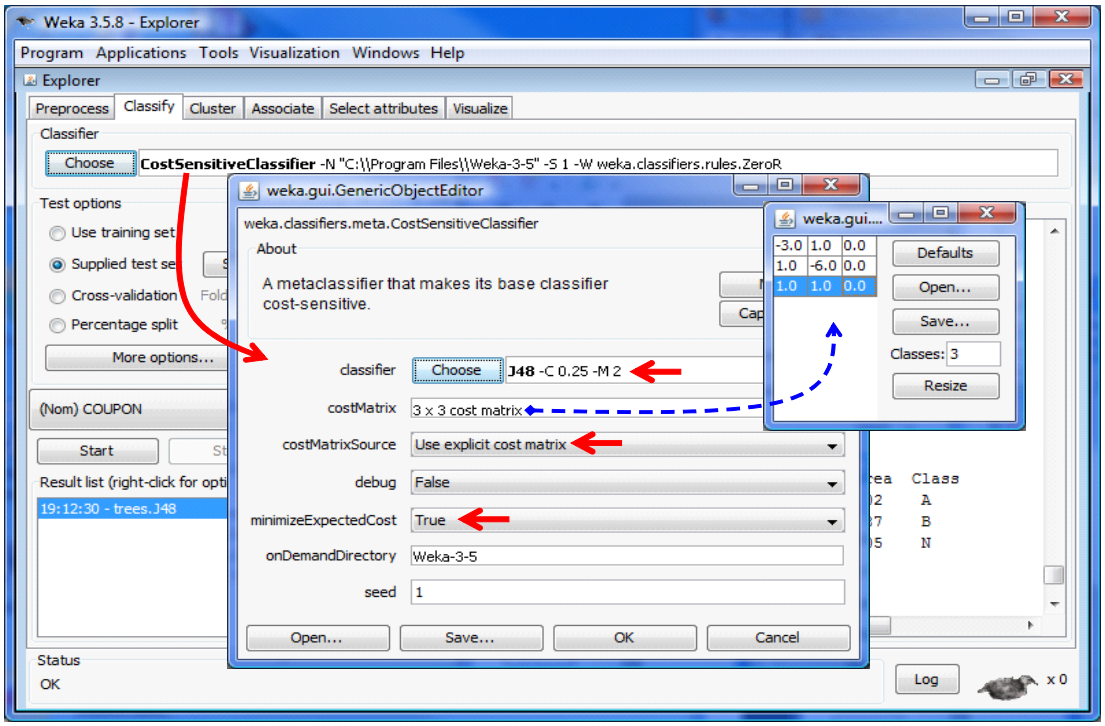


4.3 Cost Sensitive Learning + J48

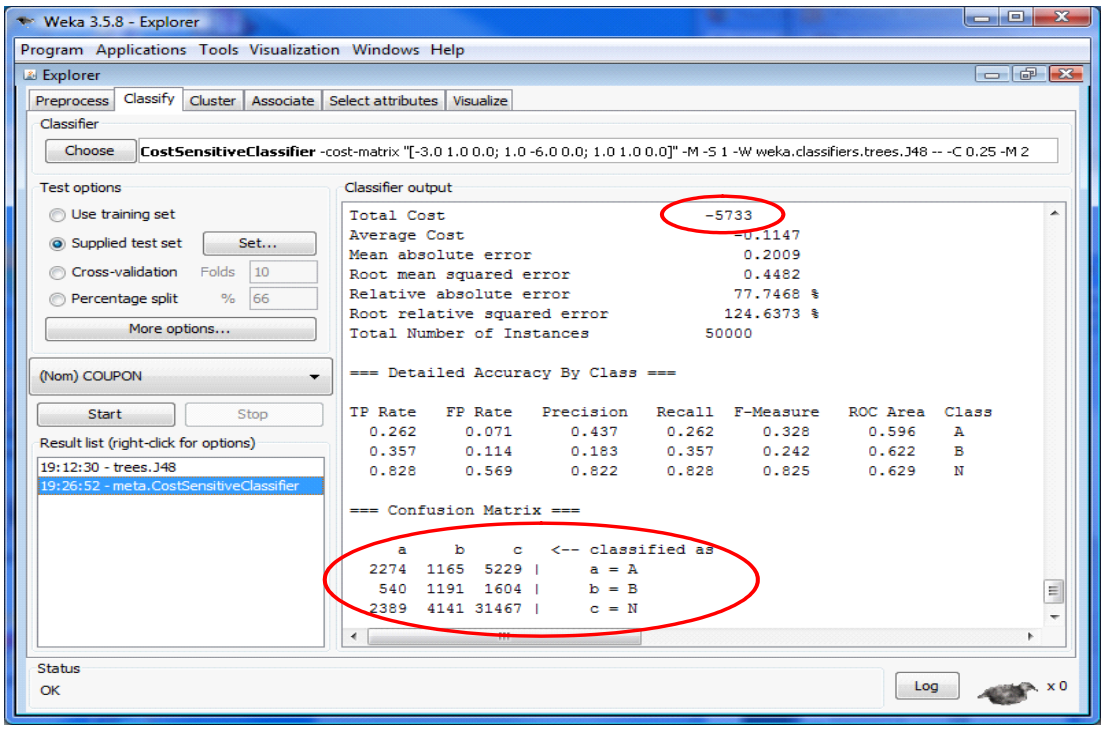
We can embed J48 in a cost sensitive approach. The approach is similar to Tanagra (section 3.5). The classification uses the posterior probabilities in order to compute the expected costs associated to each conclusion. It then selects the conclusion which minimizes the costs. We click on the CHOOSE button. Into the META node, we select the COST SENSITIVE CLASSIFIER approach.



We select the J48 method. We set also the misclassification costs matrix. We make sure that the approach uses the cost matrix during the process: COST MATRIX SOURCE must be « Use explicit cost matrix ». Finally, we must minimize the expected cost i.e. MINIMIZE EXPECTED COST = TRUE.



We click on the START button.

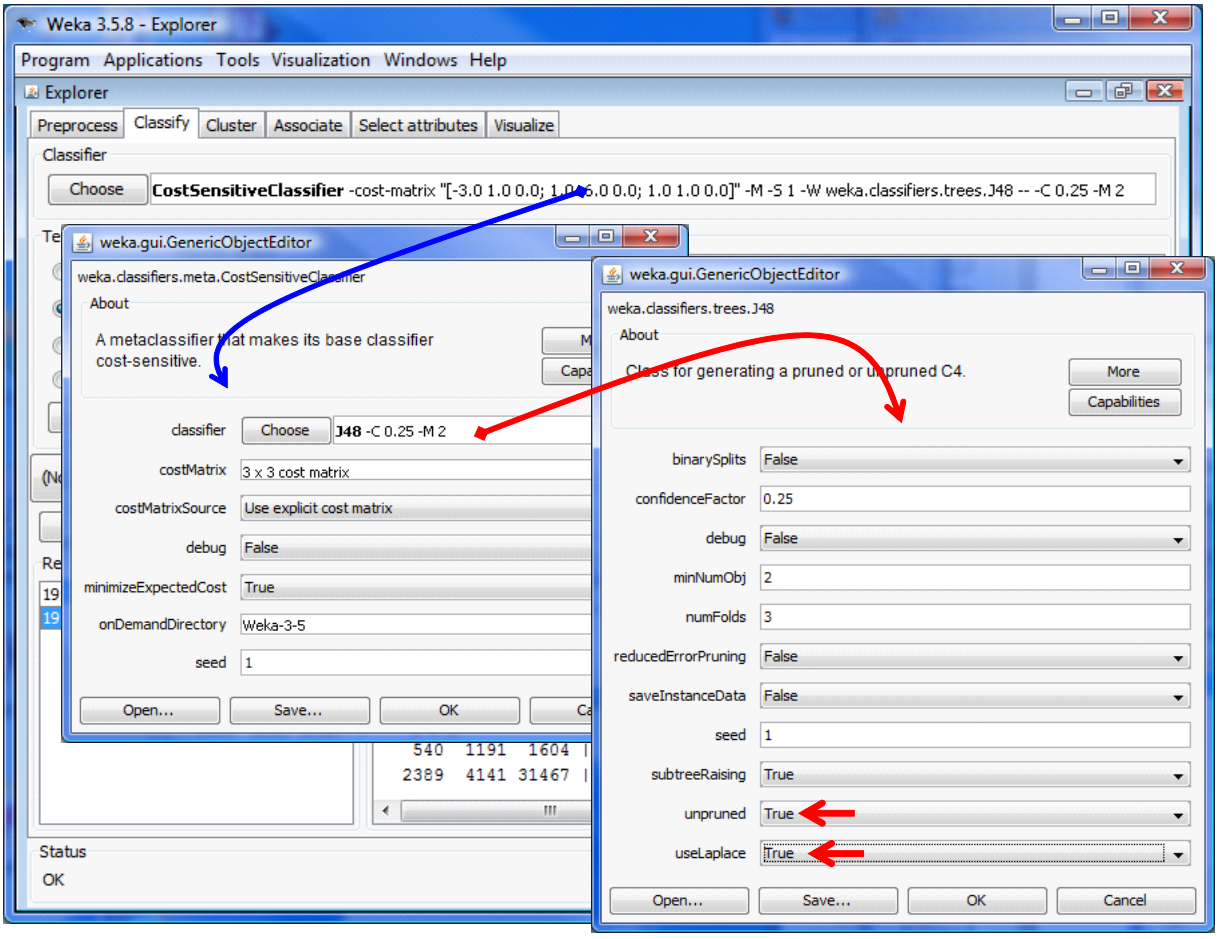


With a cost-sensitive approach, we obtain a profit = 5733. It is better than the standard J48, but it seems rather disappointing if we consider the performance of Tanagra with the same process (Cost Sensitive Learning + C4.5 → Gain = 6467).

The difference rests on the settings of the J48 method. We noted earlier that it was heavily pruned; more in all cases that C4.5. But we also know that trees which are too pruned give a bad estimation of the posterior probabilities (see for instance Elkan, « The foundations of cost-sensitive learning », IJCAI-2001; http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.514).

Then, we must modify the settings of J48 in order to: (1) prevent post pruning; (2) smoothing the posteriori probabilities estimation using the laplacian estimator.

To do that, we click on the description of the selected method. We set UNPRUNED = TRUE and USELAPLACE = TRUE.



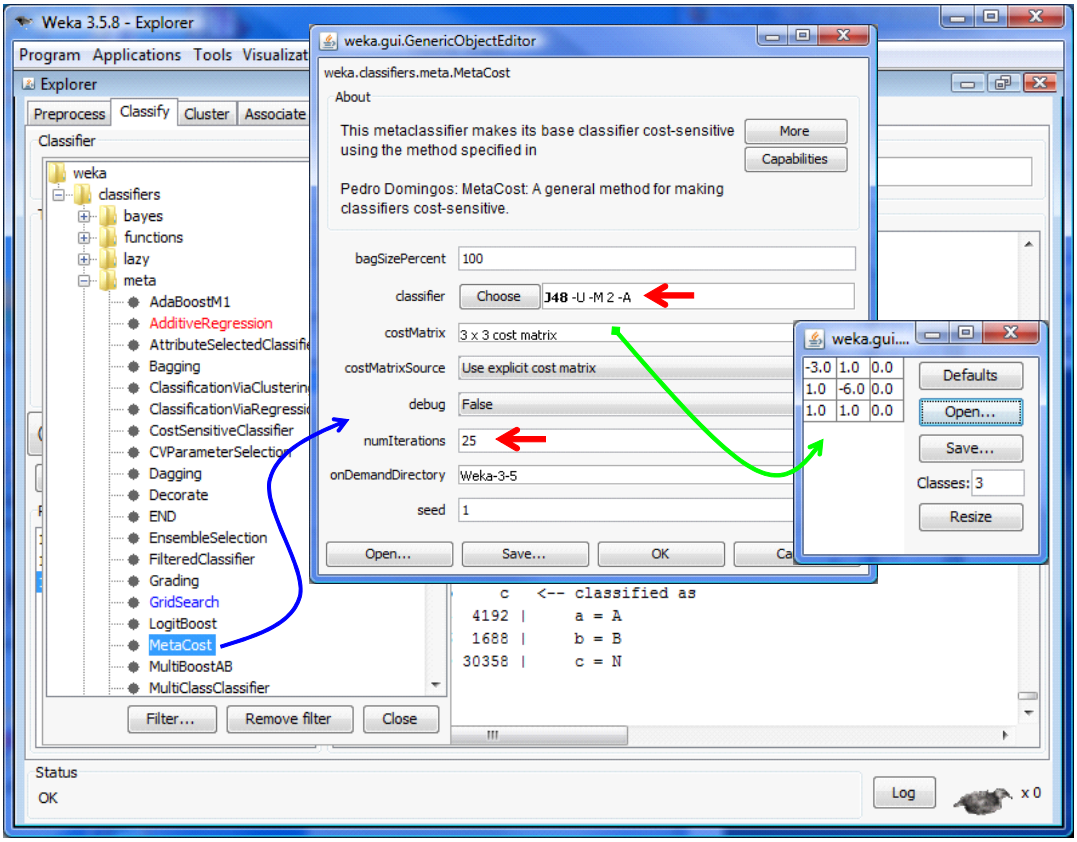
We click on START, the obtained profit (**Gain = 6462**) is similar to that obtained with Tanagra.

Note: In this case, we relied on known results of the literature to refer correctly settings. Unfortunately, this knowledge is not always available with an acuity that allows them to function effectively. We often use trial and error to detect the best parameters. We must not use the test set in this process.

4.4 MetaCost + J48

MetaCost (Domingos, 1999) is available under Weka. We click on the CHOOSE button, we select the node META. We set the parameters, especially the misclassification cost matrix. We ask 25 replications in order to obtain comparable results to Tanagra.

For the J48 algorithm, we set the parameters as before (no post-pruning, laplacian estimator of probabilities).



Then, we click on START button. We obtain **Gain = 6720**, we can compare this with the gain of MultiCost under Tanagra. There is a little difference. But it is negligible.

5 R software

Tanagra and Weka offer turnkey solutions. In the vast majority of cases, they are adequate. But when we want to insert some modifications or variants, it is more convenient to dispose of a programming language.

R software is a good solution in this case. With a few simple instructions, we can quickly obtain efficient solutions. In the following, we describe some implementations of the solutions given above (cost sensitive classification rule from posterior probabilities, cost sensitive bagging). The underlying learning algorithm is a decision tree from the RPART package.

Note: The comments are in French in the screenshot below, but in the on line archive file, an English version of the source code is available.

The RPART package. First, we load the RPART package

```
library(rpart)
```

Specifying the misclassification cost matrix. Second, we define the misclassification cost matrix

```
#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)
```

Partitioning the dataset. Then, we subdivide the dataset into training and testing sets.

```
#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)]
```

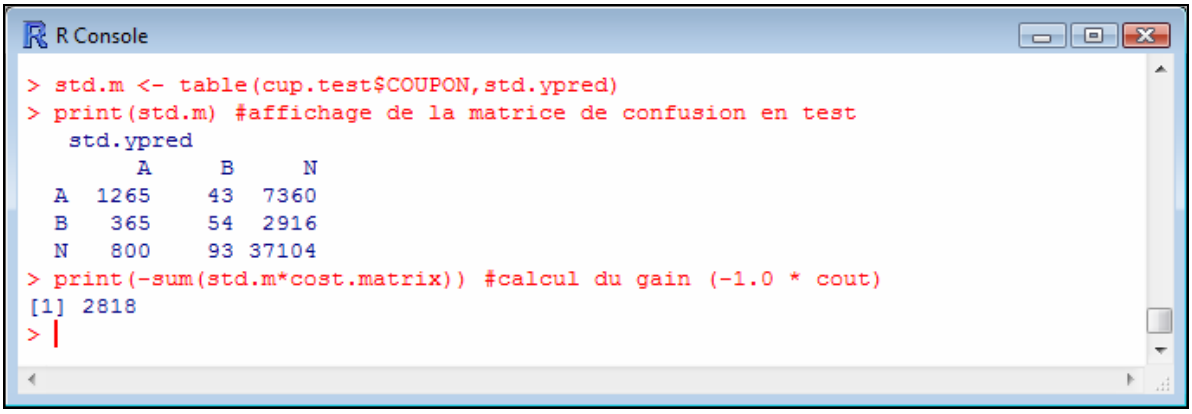
Cost insensitive approach. We can now evaluate the cost insensitive approach. We create a tree with the RPART procedure. We apply the predict(.) procedure of the RPART object on the test set. We set the settings in order to favor a large tree.

```
#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)
```

In the screenshot above: **rpart.control(.)** allows to set the parameters of the decision tree algorithm; **rpart(.)** creates the model**; predict(.)** is applied on the test set; then we compare the true values with the predicted values using **table(.)**; finally, we compute the profit function, we obtain **Gain = 2818**.



Cost sensitive classification process. With the same tree, we want to assign class using the misclassification costs matrix. The global procedure is the following

```
#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))
```

The obtained profit is **Gain = 6450**.

```
R Console
> 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)
   cost.std.ypred
        1     2     3
  A  3412   660  4596
  B   750   732  1853
  N  4721  2047 31229
> print(-sum(cost.std.m*cost.matrix))
[1] 6450
>
```

Of course, the **affectation(.)** is very important, let us see the source code:

```
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)
}
```

In the **predict(.)** procedure, we use the (type = « prob ») option in order to obtain the posterior probabilities; **produit(.)** computes the expected cost associated to each class prediction; **prediction(.)** reaches the minimum.

« Cost sensitive » Bagging – Learning phase. In the following source code, we create the trees without talking account the costs, but we use a cost sensitive classification process for the individual model.

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 – Classification phase. In the classification process, we use the set of trees in order to obtain the posterior probabilities. Then, we select the prediction which minimizes the expected loss.

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)

}
```

The utilization of these procedures…

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

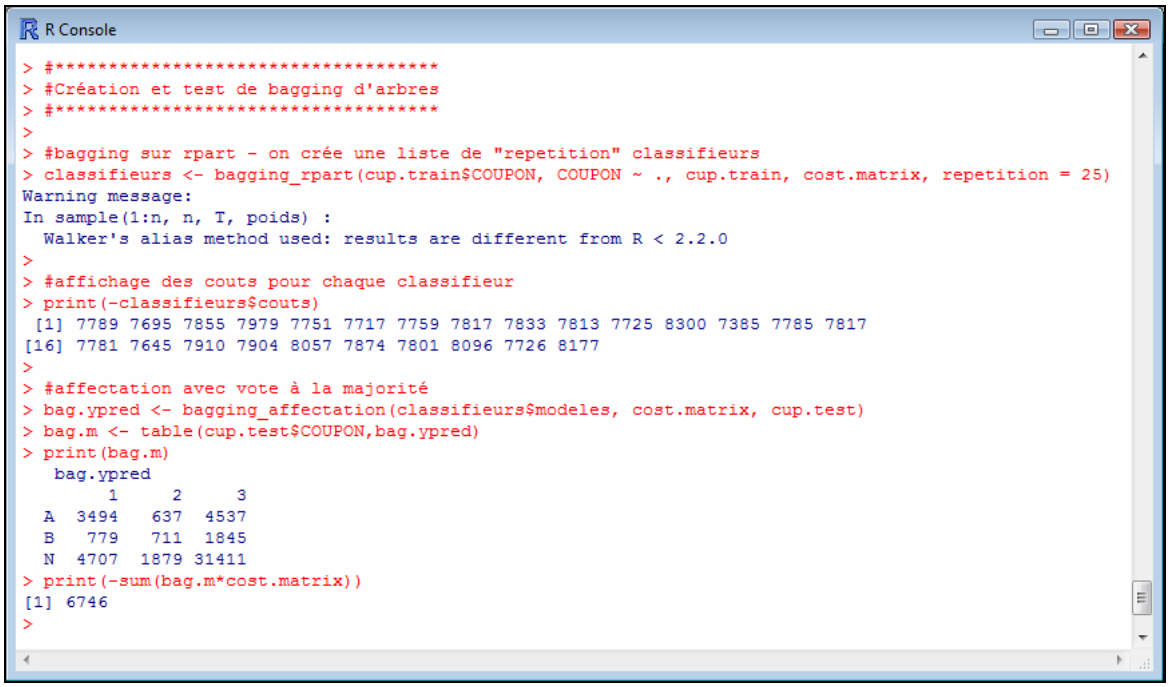
```
#************************************
#Création et test de bagging d'arbres
#************************************

#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))
```

… gives the following results:



The profit is **Gain = 6746**. This is comparable, within the same kind of approaches, to the performance of Tanagra and Weka.

6 Conclusion

Whatever the software, we can see above all the strong consistency of results. The methods are clearly distinguished when we summarize the obtained gains.

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

We are far from this, however, if one refers to the results reported on the competition site "Data Mining Cup – 2007". The winner obtained a Gain = 7907. This is well above our results (http://www.data-mining-cup.com/2007//Wettbewerb/Preise/1230988147/). It seems it is used a combination of large number of classifiers using a resampling approach (up than 2000 - http://www-i6.informatik.rwth-aachen.de/web/Teaching/LabCourses/DMC/dmclab.php). But unfortunately, the program is not available. We cannot verify this result.

The tools used in this tutorial, in the standard version, cannot handle this kind of strategies. We must develop a specific version if we want to implement a massive combination of classifiers, especially to better use the main memory in order to store differently the intermediate results.