Objectif

Comparer TANAGRA, SIPINA et WEKA lors de l'apprentissage d'un réseau de neurones.

S'agissant de l'apprentissage d'un réseau de neurones, un perceptron multicouches dans notre cas (MULTILAYER PERCEPTRON en anglais), quel que soit le logiciel utilisé, nous devons impérativement passer par les étapes suivantes :

- Importer les données dans le logiciel ;
- Définir le problème à résoudre, c.-à-d. sélectionner les descripteurs et la variable à prédire ;
- Subdiviser les données en ensemble d'apprentissage et de test ;
- Sélectionner la méthode et éventuellement la paramétrer ;
- Lancer l'induction sur les données d'apprentissage ;
- Evaluer le réseau sur l'ensemble test.

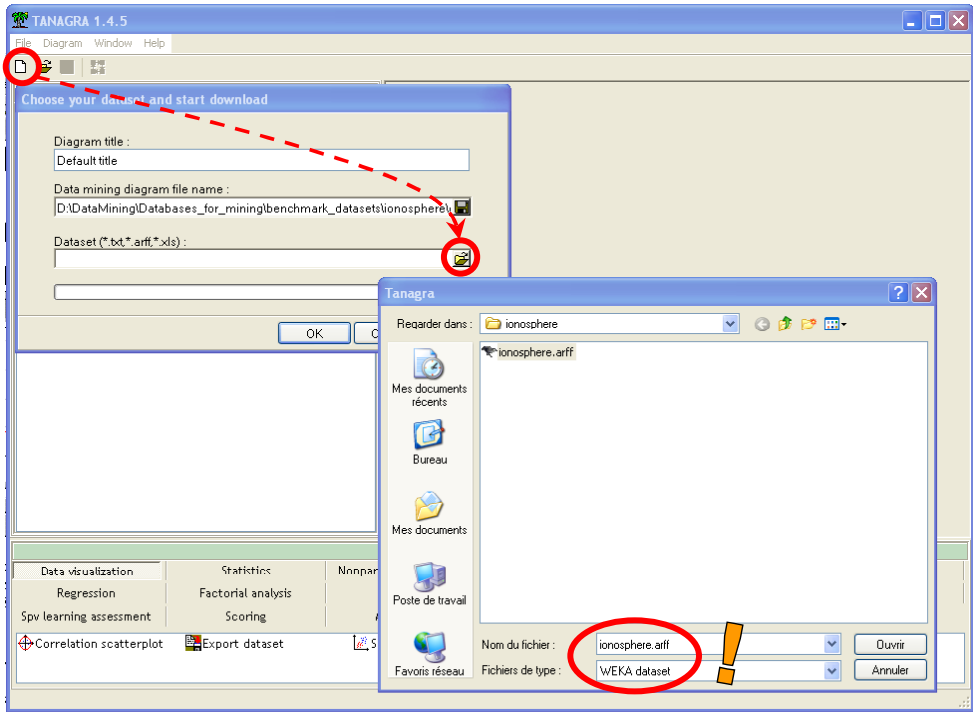
Fichier

Nous utilisons le fichier IONOSPHERE.ARFF (source UCI IRVINE – format de données WEKA). Les données ont été d'emblée centrées et réduites pour nous affranchir des différences entre les techniques de pré-traitement mis en œuvre par les logiciels. Le fichier comporte 351 observations, 33 descripteurs continus ; l'attribut classe est binaire.

Construire un perceptron multicouches avec TANAGRA

Charger les données

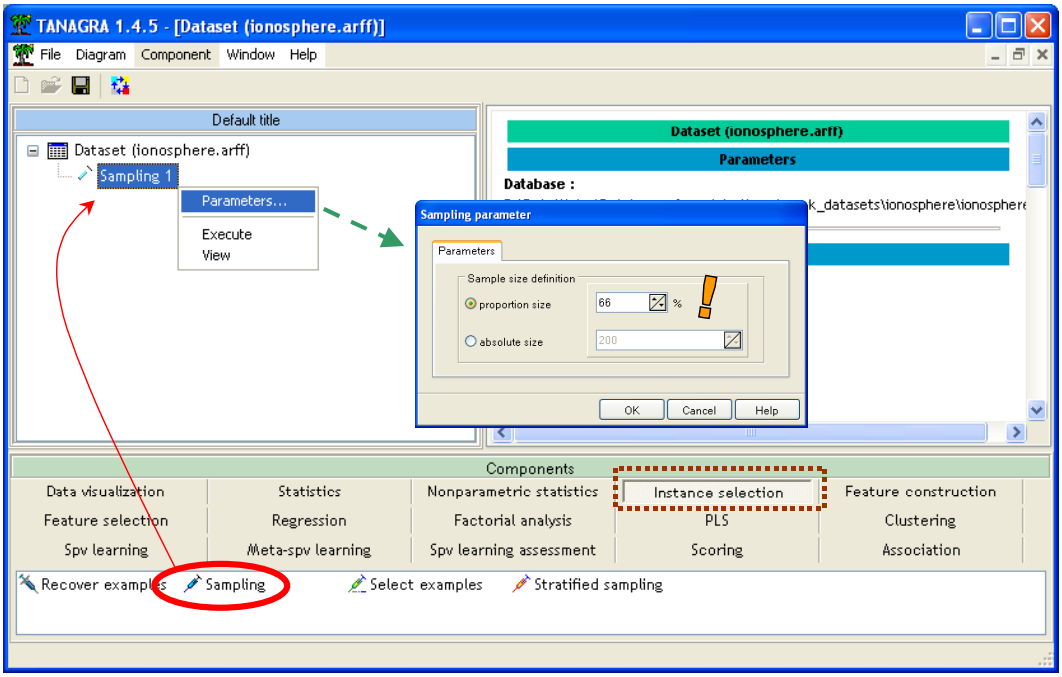
Première étape toujours, après le lancement du logiciel, nous devons charger les données. Nous activons le menu FILE/NEW pour créer un nouveau diagramme et nous sélectionnons le fichier IONOSPHERE.ARFF.



Subdiviser les données en ensemble d'apprentissage et de test

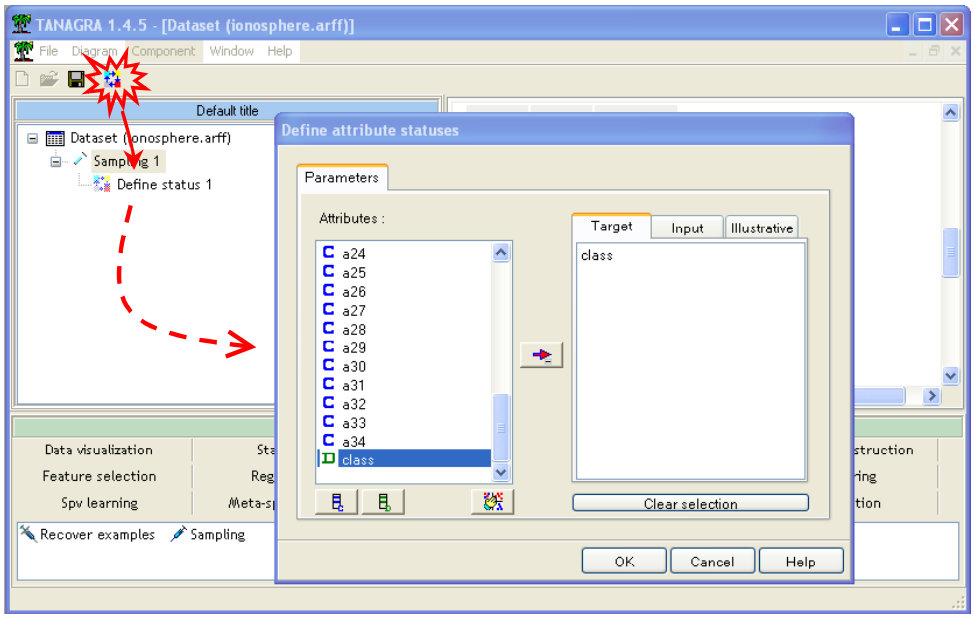
L'étape suivante consiste à subdiviser les données en 2 parties : la première servira à construire le modèle de prédiction, la seconde sera utilisée pour en évaluer les performances en prédiction.

Pour ce faire, nous insérons le composant SAMPLING dans le diagramme, 66% des observations seront utilisés pour l'apprentissage.



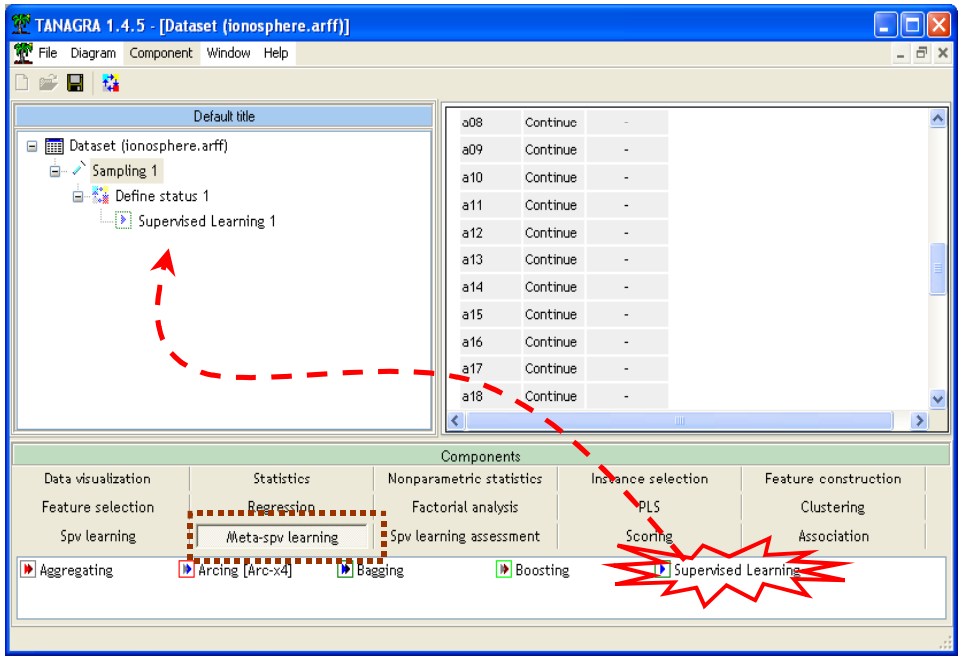
Sélectionner les attributs

Pour définir les variables d'entrées et la variable à prédire, nous plaçons le composant DEFINE STATUS dans le diagramme. Il y a un raccourci dans la barre d'outil. Nous plaçons en INPUT toutes les variables continues, en TARGET la variable CLASS.

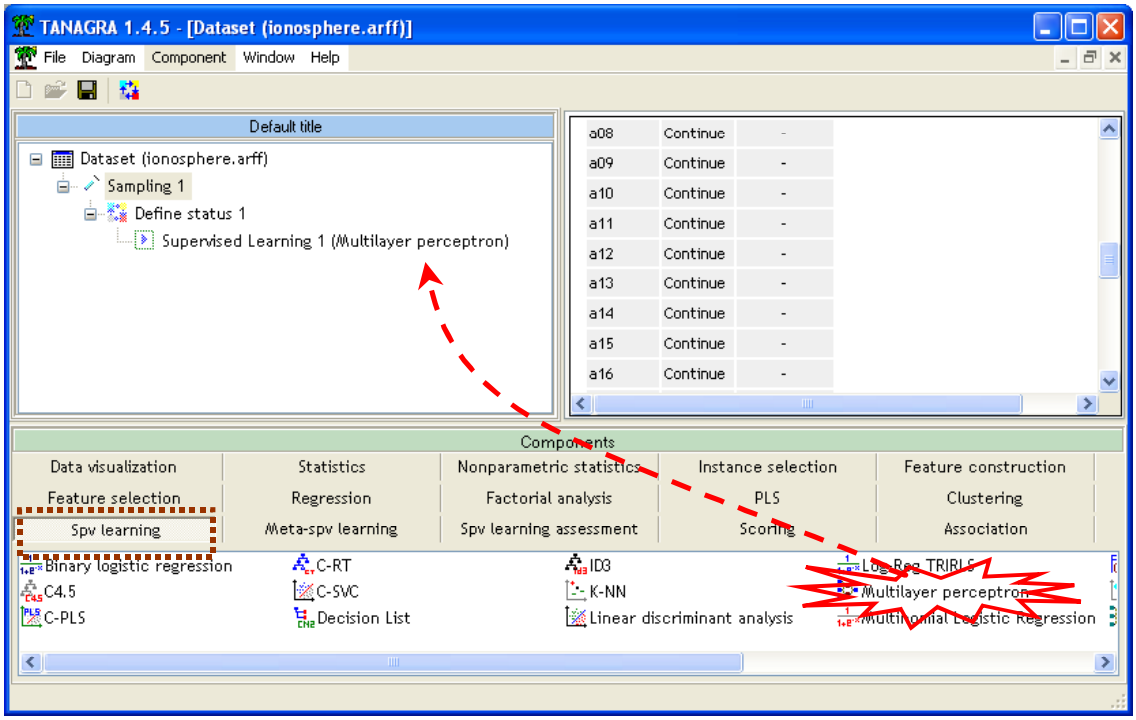


Méthode d'apprentissage

Il nous reste alors à placer le perceptron multicouches dans le diagramme. Comme pour toutes les méthodes supervisées, nous devons procéder en deux étapes : placer tout d'abord le « méta-apprentisseur » qui instancie la méthode, puis lui attribuer l'algorithme d'apprentissage proprement dit. Nous insérons donc tout d'abord le composant SUPERVISED LEARNING qui se trouve dans la palette META-SPV LEARNING.

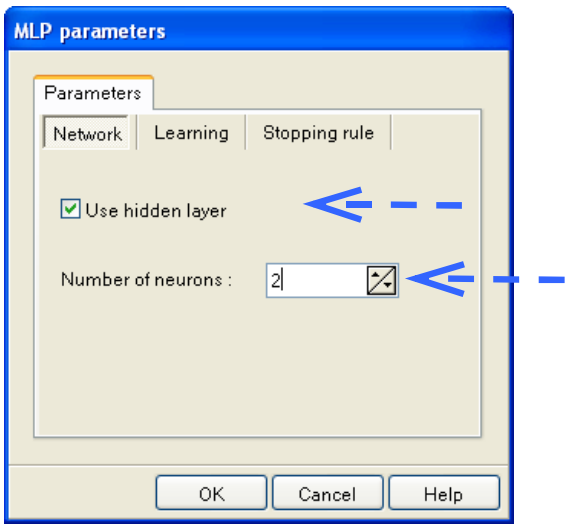


Puis nous lui attribuons la méthode MULTILAYER PERCEPTRON situé dans la palette SPV LEARNING.



Paramétrage

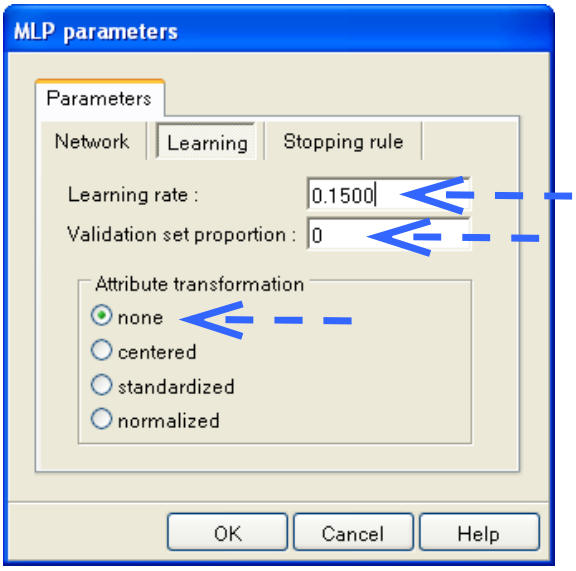
Regardons en détail maintenant les paramètres de la méthode en cliquant sur le menu SUPERVISED PARAMETERS. Le premier onglet **NETWORK** permet de définir l'architecture du réseau, nous optons pour un réseau avec une couche cachée à 2 neurones.



Le second onglet **LEARNING** permet de définir les paramètres de l'apprentissage. Nous fixons la constante d'apprentissage « LEARNING RATE » à 0.15.

Il est d'usage avec les réseaux de neurones d'utiliser une fraction des données d'apprentissage, appelé « ensemble de validation » pour suivre l'évolution de l'erreur, cela permet par exemple de stopper l'induction lorsque nous constatons que le taux d'erreur sur cet ensemble de validation ne diminue plus lors des itérations[1]. Cette option serait surtout intéressante si nous avions la possibilité de suivre cette évolution, et éventuellement, d'arrêter de manière interactive le processus. Cela n'est pas possible avec TANAGRA, ça le sera en revanche avec SIPINA. Nous décidons donc de dédier toutes les observations en apprentissage pour la mise à jour des poids synaptiques -- VALIDATION SET PROPORTION = 0.

Enfin, puisque les données ont déjà été pré-traitées en amont, nous décidons de ne procéder à aucune transformation des données – ATTRIBUTE TRANSFORMATION = NONE.

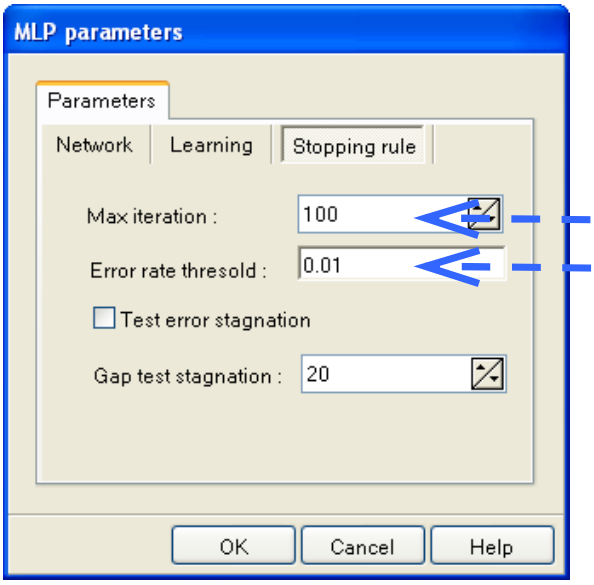


Le troisième onglet **STOPPING RULE** enfin permet de définir les règles d'arrêt du processus d'apprentissage. Plusieurs possibilités sont offertes. Le plus simple finalement consiste à fixer le nombre d'itérations (MAX ITERATION = 100) et le taux d'erreur seuil qui permettrait de penser que l'apprentissage est suffisamment performant (ERROR RATE THRESOLD = 0.01).

Une autre possibilité serait de vérifier que le taux d'erreur calculé ne décroît plus TEST ERROR STAGNATION au bout de GAP TEST STAGNATION itérations. Il faut néanmoins être très prudent avec cette option, en effet dans certains cas, le taux d'erreur peut stagner pendant un certain temps avant de décroître à nouveau lorsque de nouvelles zones de

[1] Le terme « validation » peut prêter à confusion. C'est néanmoins le terme utilisé dans la majorité des logiciels.

l'espace de représentation sont explorées. Dans la plupart des cas, il est plus approprié de ne pas activer cette option.



Lecture des résultats

Il reste maintenant à lancer l'apprentissage et visualiser le résultat en activant le menu VIEW du composant. La fenêtre de résultats s'affiche.

Supervised Learning 1 (Multilayer perceptron)

Parameters

MLP architecture	
Use hidden layer	yes
Neurons in the hidden layer	2
Learning parameters	
Validation set proportion	0.00
Learning rate	0.15
Attribute transformation	none
Stopping rule	
Max iteration	100
Error rate thresold	0.0100
Verifiy error stagnation	no

Results

Classifier performances

Error rate	0.0260

Values prediction

Value	Recall	1-Precision
good	1.0000	0.0403
bad	0.9318	0.0000

Confusion matrix

	good	bad	Sum
good	143	0	143
bad	6	82	88
Sum	149	82	231

La première partie de la fenêtre résume les paramètres du modèle et présente la matrice de confusion calculée sur les données en apprentissage. Le taux d'erreur en resubstitution est égal à 0.026. Nous savons que ce taux est très souvent optimiste, il l'est d'autant plus avec les

réseaux de neurones. En effet, il est courant d'atteindre un taux d'erreur égal à zéro en apprentissage lorsque nous mettons beaucoup de neurones dans la couche cachée.

Dans la suite de la fenêtre de résultats, nous pouvons voir les poids synaptiques « WEIGHTS ». Il est possible de récupérer ces poids dans un tableur pour réaliser un classement sur de nouvelles observations par exemple. Attention, il est nécessaire dans cette optique de récupérer, le cas échéant, les paramètres calculés lors de la transformation des données.

La partie ATTRIBUTE CONTRUBTION calcule le taux d'erreur du modèle dans lequel on aurait désactivé un des descripteurs. Il compare alors ce taux avec celui du modèle complet. Cette procédure permet ainsi d'évaluer, de manière un peu empirique quand même, la contribution individuelle de chaque descripteur dans les performances du réseau.

Attribute contribution

Excluded attribute	Error rate	Difference	Statistics
none	0.0260	-	-
a01	0.1169	0.0909	8.6868
a03	0.0433	0.0173	1.6546
a04	0.0519	0.0260	2.4819
a05	0.0563	0.0303	2.8956
a06	0.0563	0.0303	2.8956
a07	0.0260	0.0000	0.0000

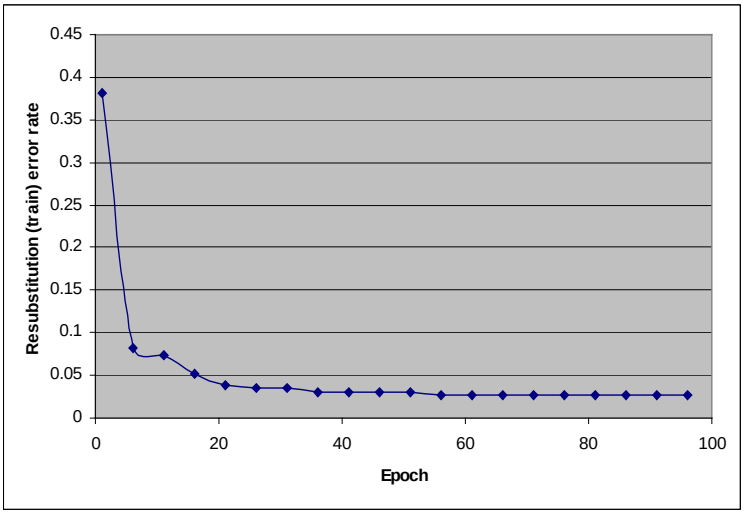
Nous observons par exemple dans le tableau ci-dessus que le taux d'erreur du perceptron est de **0.026** ; si nous désactivons la variable « a01 », c.-à-d. nous substituons par une constante toutes les valeurs de cet attribut lors du classement des observations, le taux d'erreur passe à **0.1169**.

La différence est égale à **0.0909** ; la statistique du test de comparaison entre ces deux taux, qui sont en fait des proportions, est de **8.6868**. Bien entendu, il ne s'agit pas d'un test statistique rigoureux, il faut plutôt le prendre comme un indicateur d'intensité de l'écart, il nous donne une idée du rôle joué par l'attribut évalué. Si « **statistics** » est égal à 0, l'attribut joue un rôle négligeable dans la prédiction, plus sera grand, plus la variable tient un rôle important.

Enfin, dernier tableau, ERROR RATE DECREASING indique l'évolution de l'erreur au fil des itérations. Nous disposons du taux d'erreur et de l'erreur quadratique sur les données d'apprentissage, et éventuellement, du taux d'erreur sur les données de validation. Il est possible de retracer la courbe dans un graphique en utilisant un tableur.

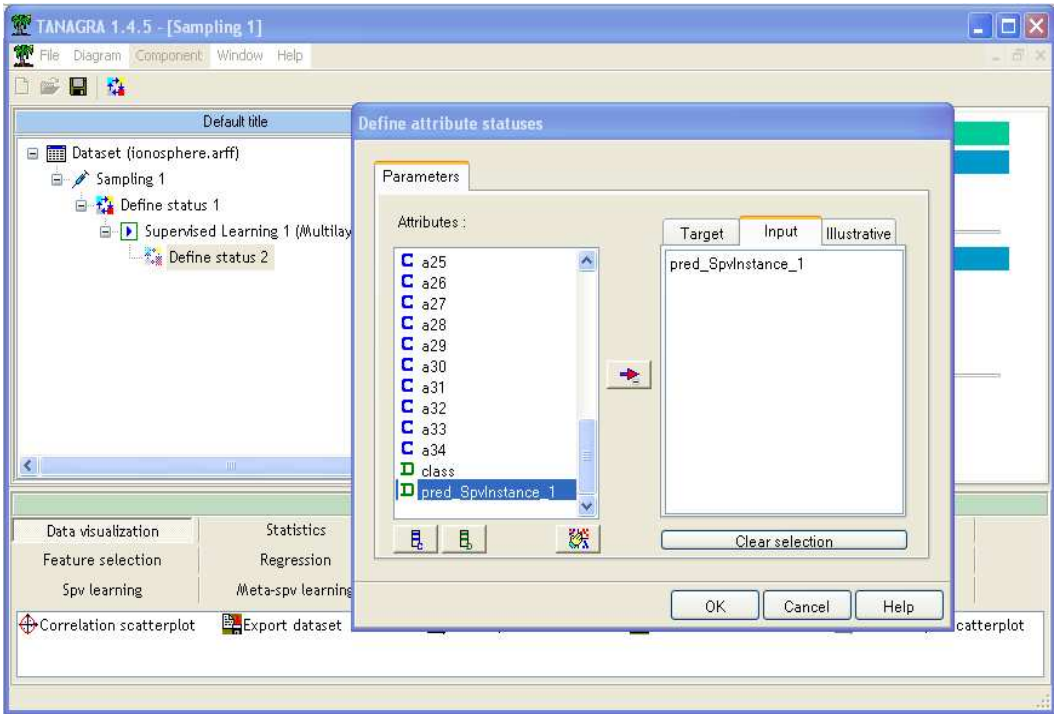
Error rate decreasing

Error evaluation			
Epoch	Train err	Val. err	MSE
1	0.3810	1.0000	54.4613
6	0.0823	1.0000	23.2339
11	0.0736	1.0000	16.4484
16	0.0519	1.0000	13.4766
21	0.0390	1.0000	11.7270
26	0.0346	1.0000	10.5781
31	0.0346	1.0000	9.8356
36	0.0303	1.0000	9.3085
41	0.0303	1.0000	8.9049
46	0.0303	1.0000	8.5669
51	0.0303	1.0000	8.1741
56	0.0260	1.0000	7.6697
61	0.0260	1.0000	7.4119
66	0.0260	1.0000	7.2302
71	0.0260	1.0000	7.0859
76	0.0260	1.0000	6.9666
81	0.0260	1.0000	6.8660
86	0.0260	1.0000	6.7802
91	0.0260	1.0000	6.7064
96	0.0260	1.0000	6.6424

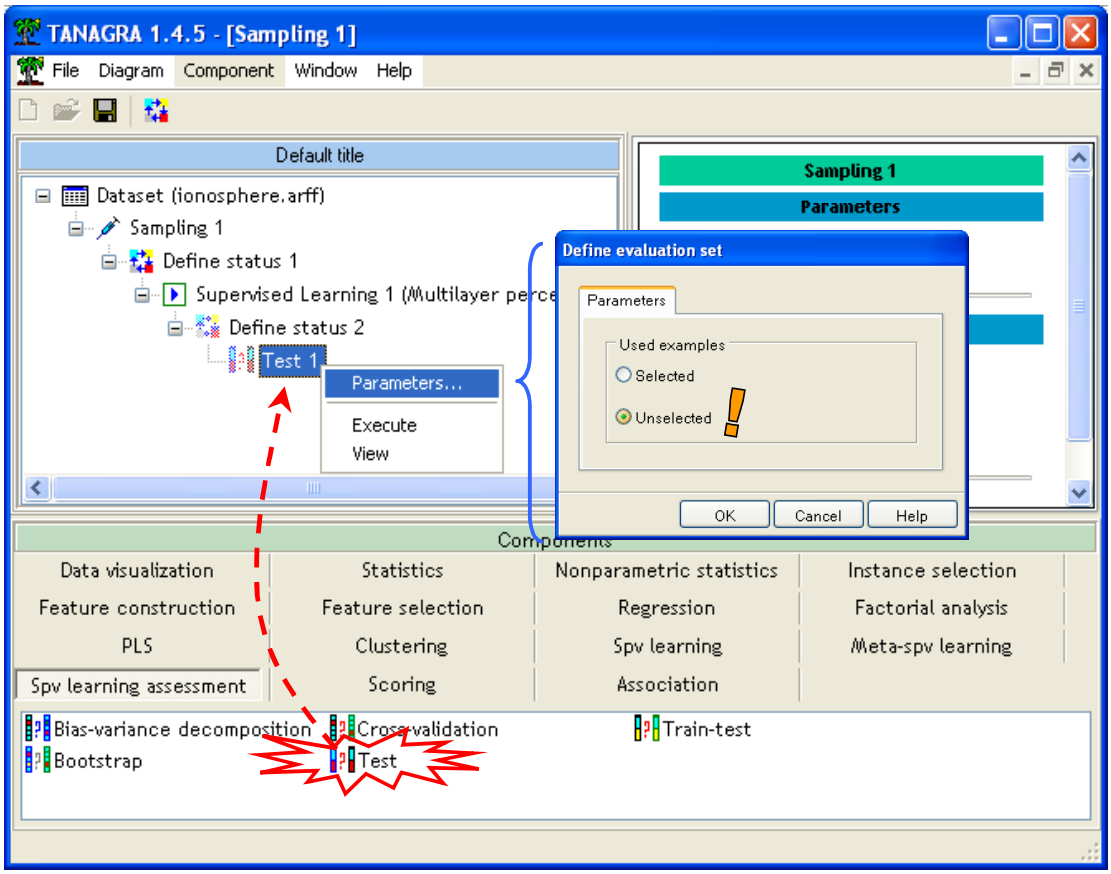


Evaluation des performances sur l'ensemble test

Il nous reste maintenant à évaluer le taux d'erreur sur les données que nous avions mis de côté au tout début du diagramme (34% des observations, soit 120 exemples). Pour ce faire, nous insérons de nouveau un composant DEFINE STATUS dans le diagramme, nous plaçons en TARGET la variable à prédire (CLASS), et en INPUT la prédiction proposée par le perceptron (PRED_SPVINSTANCE_1).



Puis, nous insérons le composant TEST de la palette SPV LEARNING ASSESMENT en veillant à ce que le taux soit bien calculé sur les individus non-sélectionnés, n'ayant pas participé à l'apprentissage.



Nous cliquons sur le menu VIEW pour accéder aux résultats, la matrice de confusion nous annonce que le taux d'erreur en test est **0.125**, à comparer avec le taux en apprentissage qui était de 0.026. Ce décalage n'est pas étonnant, il est assez fréquent avec les réseaux de neurones.

Test 1

Parameters

Evaluation set : **unselected** examples

Results

pred_SpvInstance_1

Error rate	0.1250

Values prediction

Value	Recall	1-Precision
good	0.9634	0.1319
bad	0.6842	0.1034

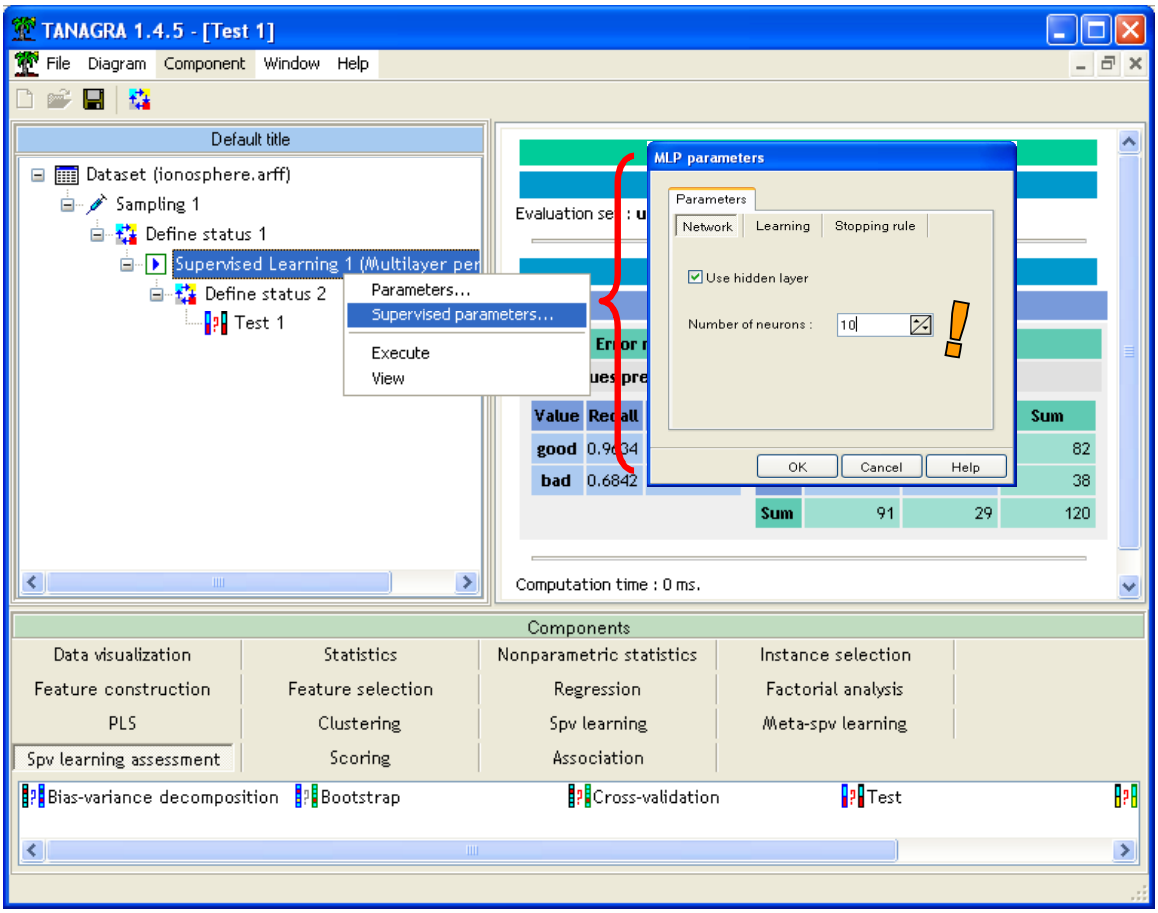
Confusion matrix

	good	bad	Sum
good	79	3	82
bad	12	26	38
Sum	91	29	120

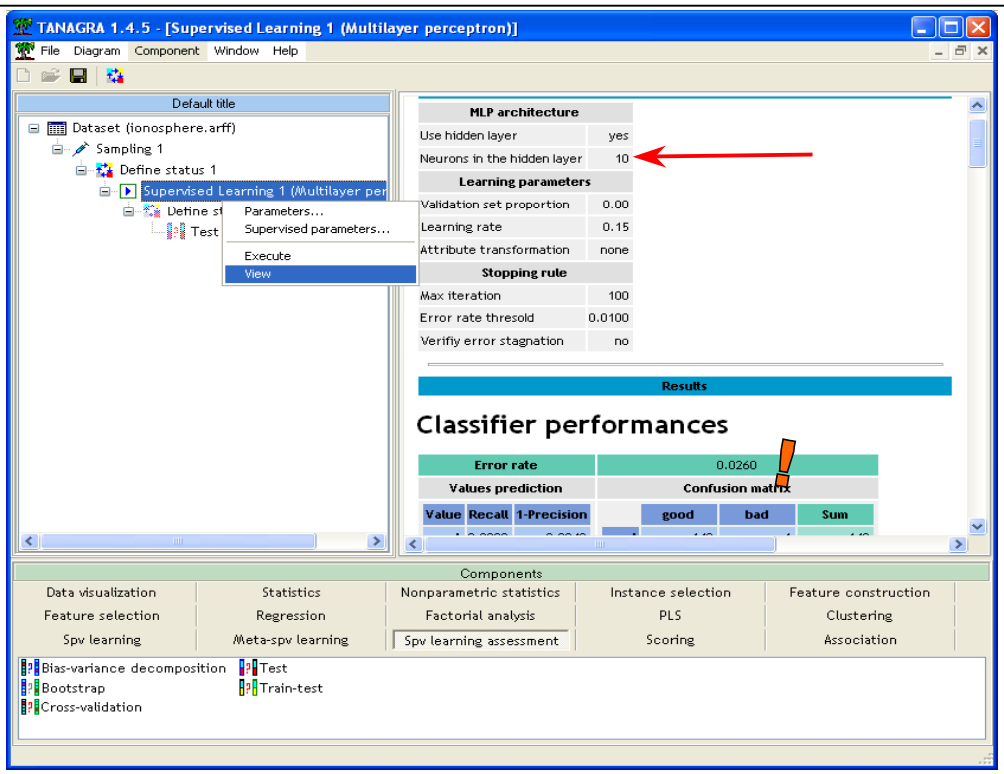
Modifier les paramètres du réseau

Un des avantages incontestables du perceptron est la possibilité de moduler la puissance du modèle de prédiction en modifiant le nombre de neurones dans la couche cachée. Plus nous ajoutons de neurones, plus le modèle pourra apprendre des formes complexes. Mais ce n'est pas anodin, en augmentant le nombre de neurones, nous prenons également le risque de trop coller aux données, construire un modèle qui s'attache aux spécificités du fichier d'apprentissage et non à la « vraie » causalité qui peut exister entre les descripteurs et la variable à prédire.

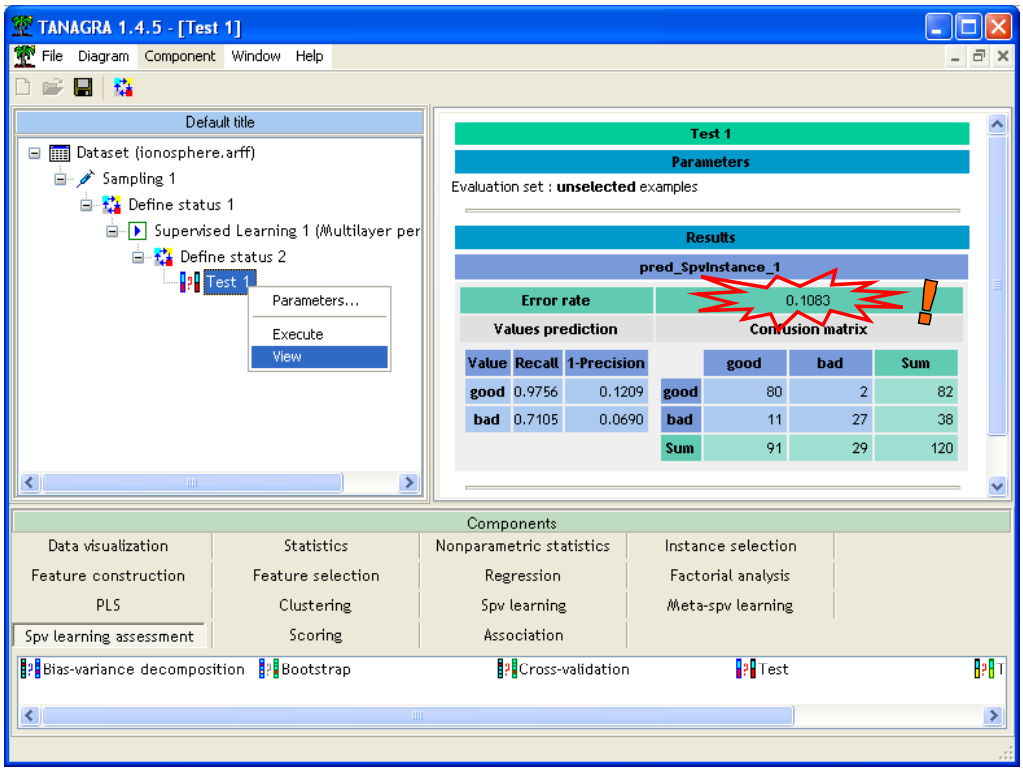
Dans notre cas, nous revenons sur le composant d'apprentissage et nous passons le nombre de neurones dans la couche cachée à 10. A priori, nous améliorons les capacités de prédiction du modèle.



Nous relançons les calculs en cliquant sur le menu VIEW, les performances ne semblent pas modifiés puisque nous obtenons de nouveau un taux d'erreur en apprentissage de 0.0206.



Voyons ce qu'il en est maintenant sur la partie test. Nous activons le menu VIEW du composant TEST du diagramme, le taux d'erreur en test est 0.1083.

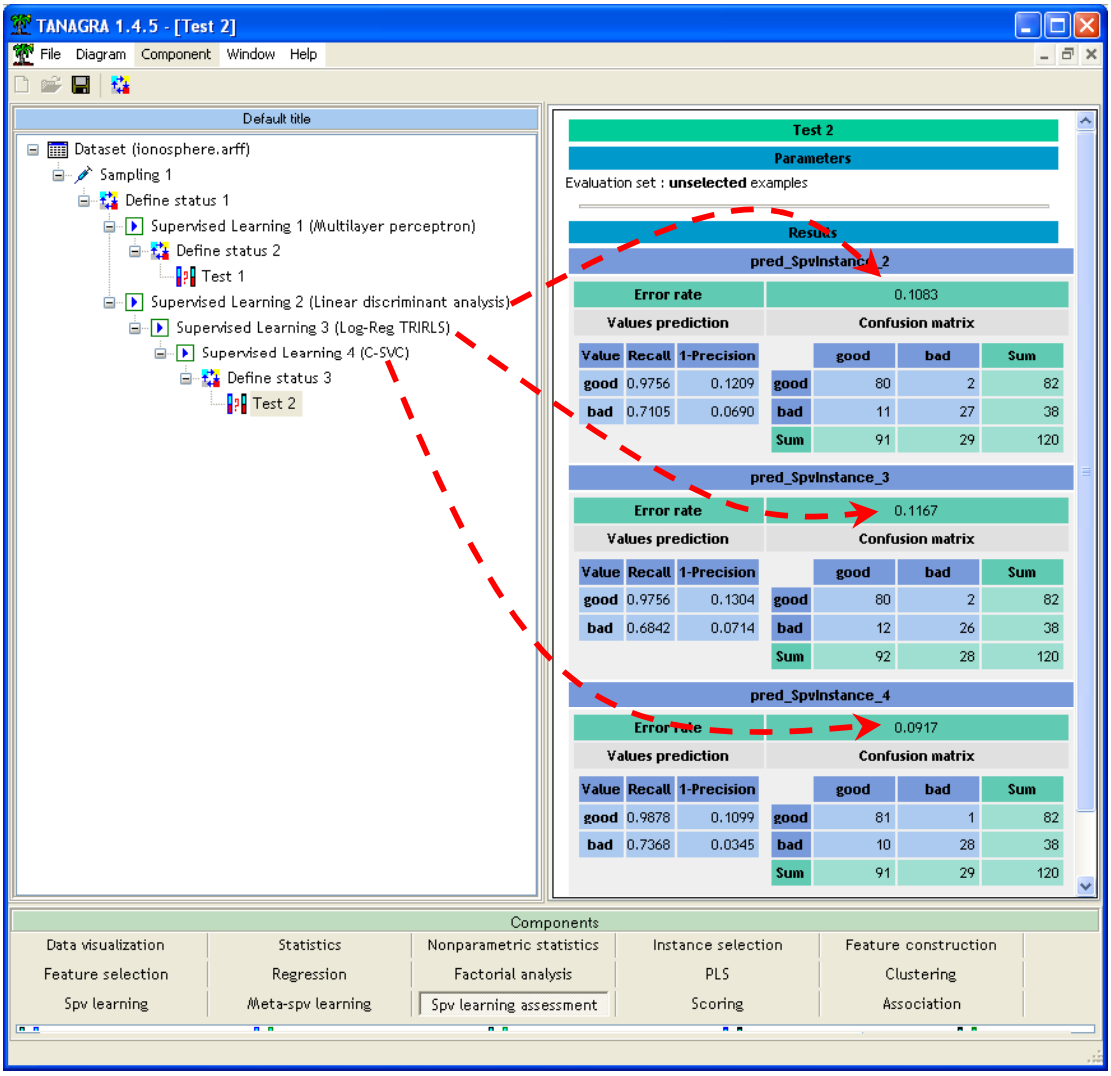


Ce qui ne transparaissait pas sur les données d'apprentissage, apparaît mieux ici. La faiblesse de l'effectif nous empêche quand même de prétendre à des conclusions définitives.

En testant différentes configurations (nombre de neurones dans la couche cachée variant de 2 à 100), nous nous sommes rendu compte qu'il n'était guère possible d'obtenir un réseau plus performant.

Autre évaluation intéressante, en désactivant la couche cachée, en utilisant un perceptron simple donc, le taux d'erreur en test est de 0.1250. Le modèle de prédiction est linéaire dans ce cas. Nous avons voulu comparer ces performances avec d'autres modèles linéaires, en l'occurrence l'analyse discriminante, la régression logistique et un SVM linéaire. Nous obtenons les résultats suivants. Il semble que le SVM soit le plus performant. Mais encore une fois, les écarts sont trop faibles (la différence se joue à un ou deux individus bien classés en plus ou en moins) pour que nous puissions raisonnablement statuer.

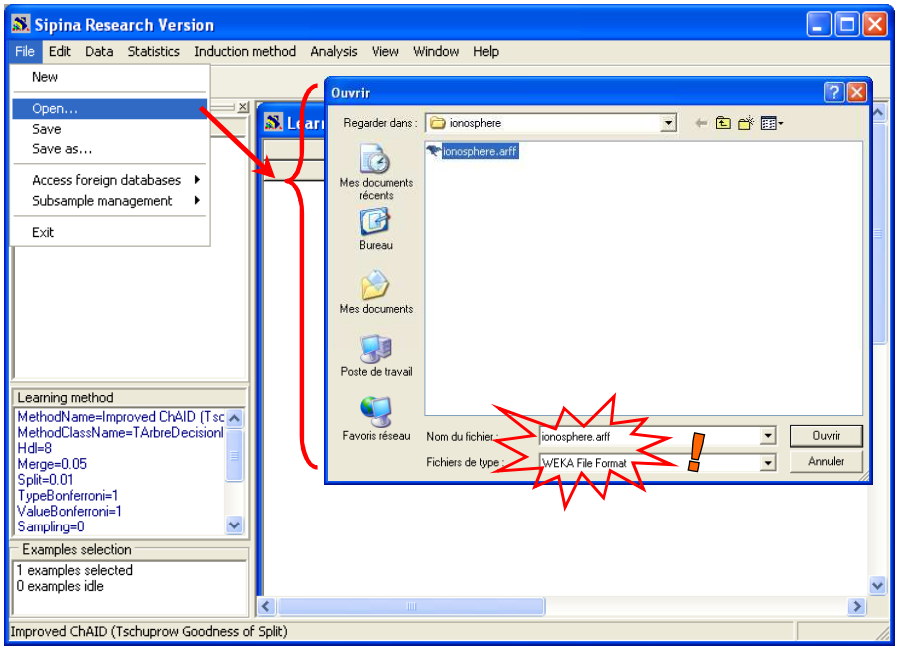
En tous les cas, la démarche est correcte, nous avons évalué les performances sur les mêmes individus en test.



Construire un Perceptron Multicouches avec SIPINA

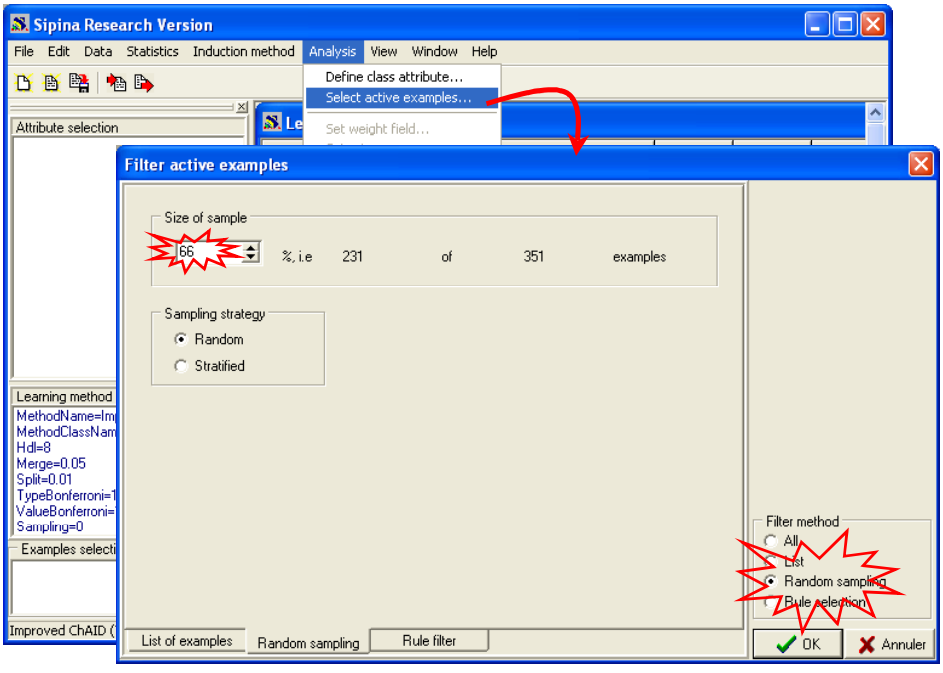
Importer les données

SIPINA est piloté par menu. La première étape est d'accéder au fichier de données. Pour ce faire, nous activons le menu FILE / OPEN, nous choisissons le format de fichier ARFF et sélectionnons le fichier IONOSPHERE.

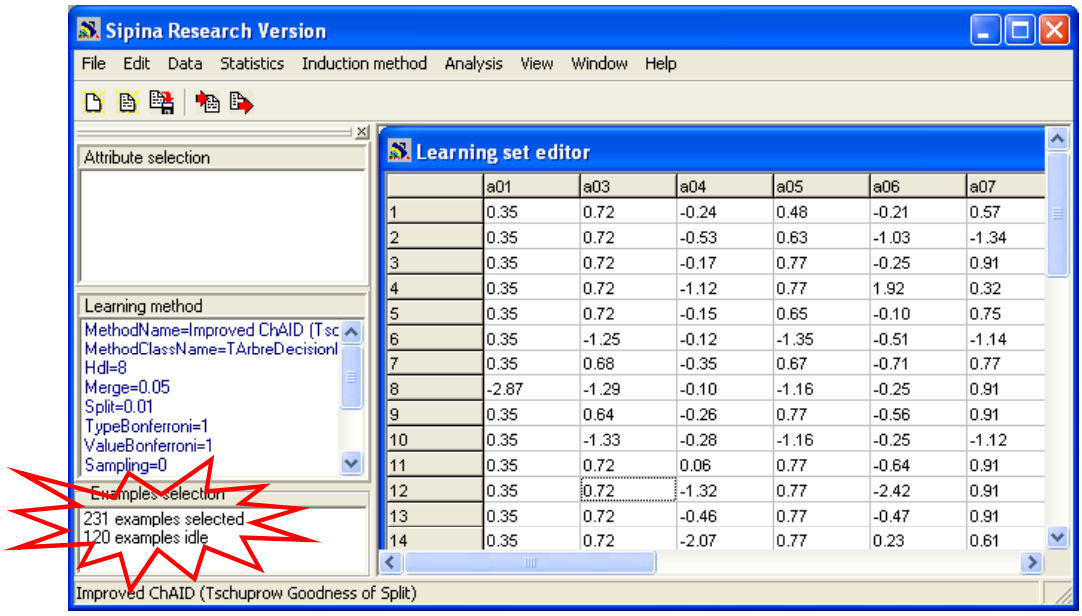


Subdiviser les données

Nous voulons réserver 66% des observations pour construire le modèle de prédiction, et le reste pour l'évaluer. Nous activons le menu ANALYSIS / SELECT ACTIVE EXAMPLES, et nous sélectionnons l'option RANDOM SAMPLING.

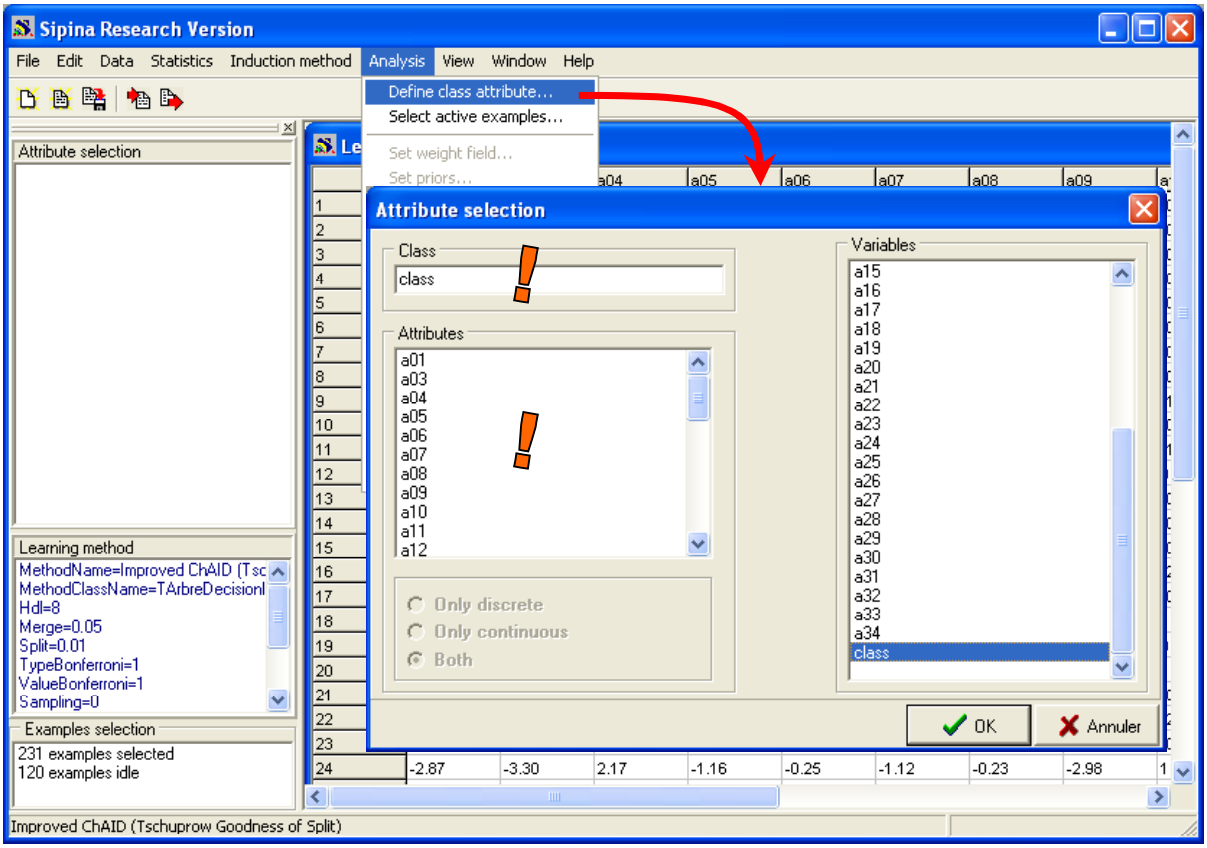


Le résultat de la sélection s'affiche dans la boîte liste située en bas et à gauche de la fenêtre principale.

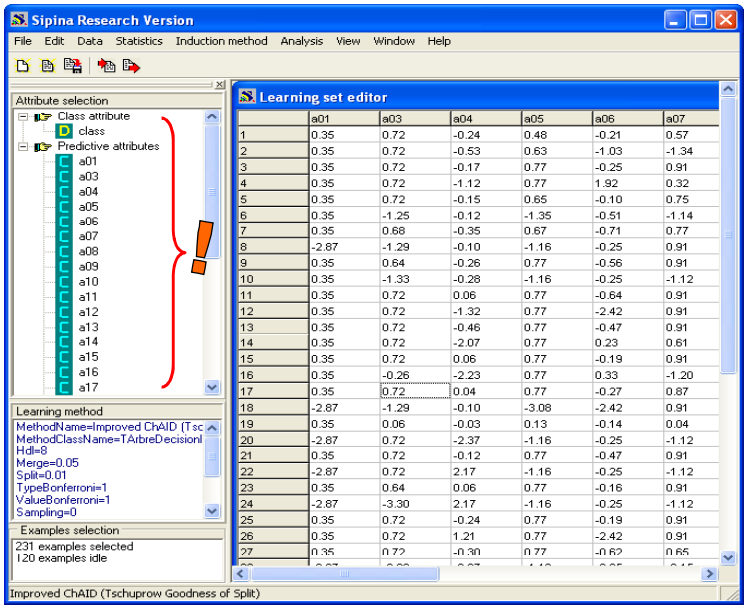


Sélectionner les variables

Pour définir les variables en entrée et la variable à prédire du réseau, nous activons le menu ANALYSIS / DEFINE CLASS ATTRIBUTE. Par glisser – déposer, nous plaçons en « attributs » toutes les variables sauf la dernière, qui elle doit être placée en « classe ».

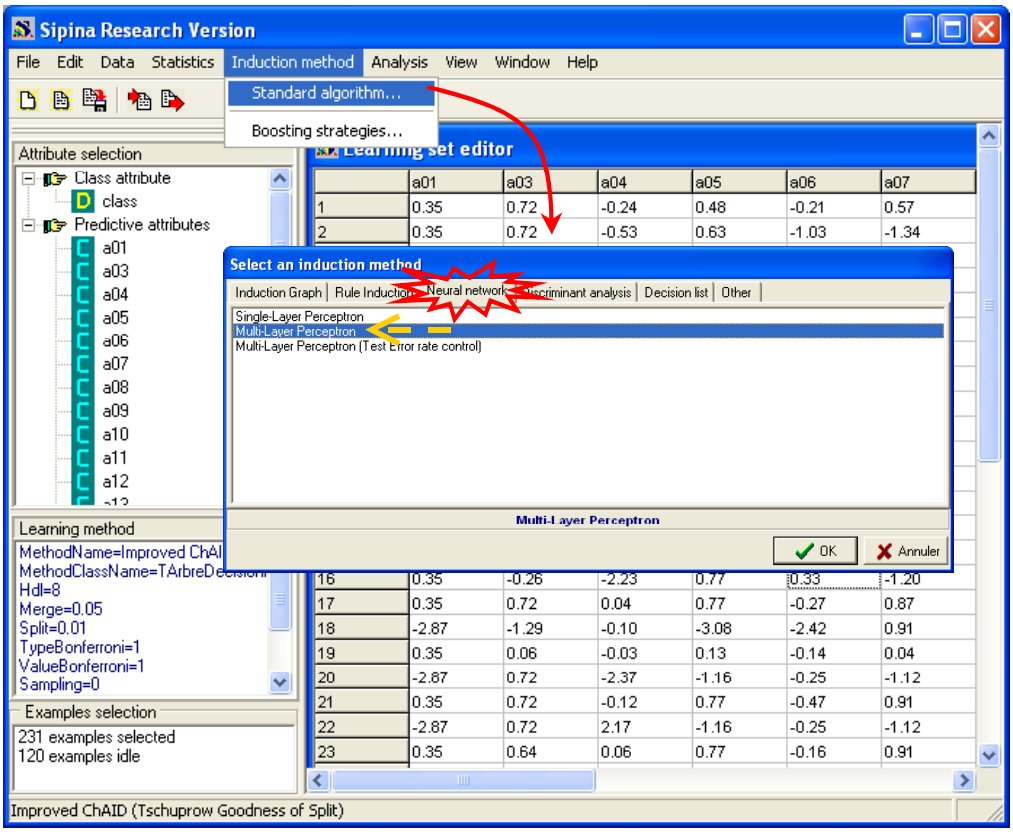


Le résultat de la sélection s'affiche dans la partie gauche de la fenêtre principale. Nous y observons également le type des variables : « C » pour les variables continues, « D » pour les discrètes.

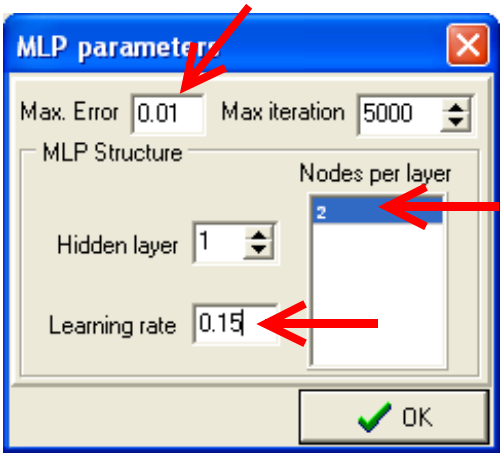


Définir et paramétrer la méthode d'apprentissage

Pour définir la méthode d'apprentissage, nous activons le menu INDUCTION METHOD / STANDARD ALGORITHM. Dans la boîte de dialogue qui apparaît, nous cliquons sur l'onglet NEURAL NETWORK puis nous sélectionnons la méthode MULTILAYER PERCEPTRON.



Après avoir validé, une seconde boîte de dialogue apparaît, nous pouvons paramétrer l'architecture du réseau et les options de l'apprentissage.

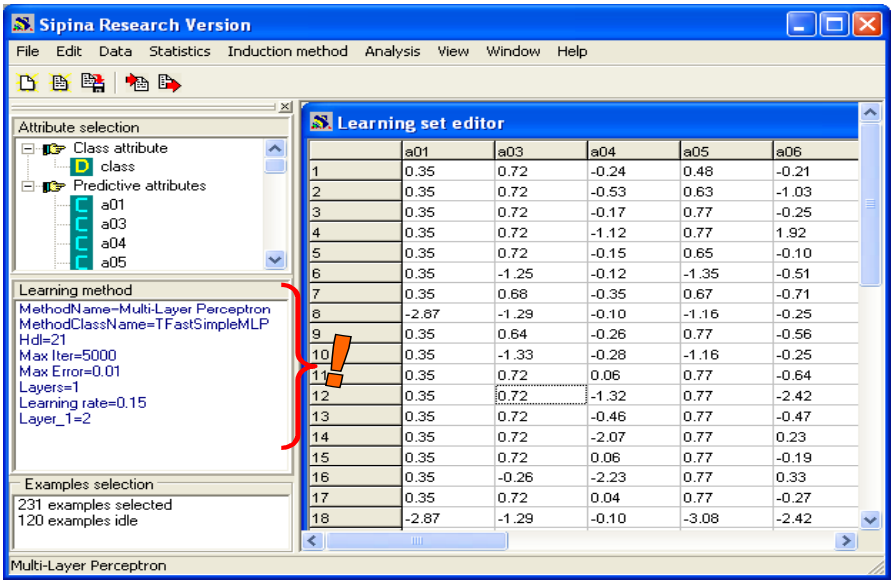


Concernant l'**architecture du réseau**, nous choisissons une seule couche cachée (HIDDEN LAYER = 1), et 2 neurones dans cette couche (NODES PER LAYER = 2).

Pour ce qui est du processus d'apprentissage, nous fixons la **constante d'apprentissage** (LEARNING RATE) à 0.15 ; nous définissons comme **règle d'arrêt** un taux d'erreur en apprentissage inférieur ou égal à 0.01 (MAX ERROR = 0.01, dès que le taux d'erreur calculé sur le fichier d'apprentissage est inférieur à ce seuil, l'induction est stoppée).

Remarquons que le nombre maximum d'itérations est assez élevé ici (MAX ITERATION = 5000). Cela peut entraîner une durée d'apprentissage excessivement longue. Fort heureusement, nous avons la possibilité de suivre le déroulement de l'apprentissage dans SIPINA, nous pourrons interrompre les calculs si nous jugeons que le processus a convergé.

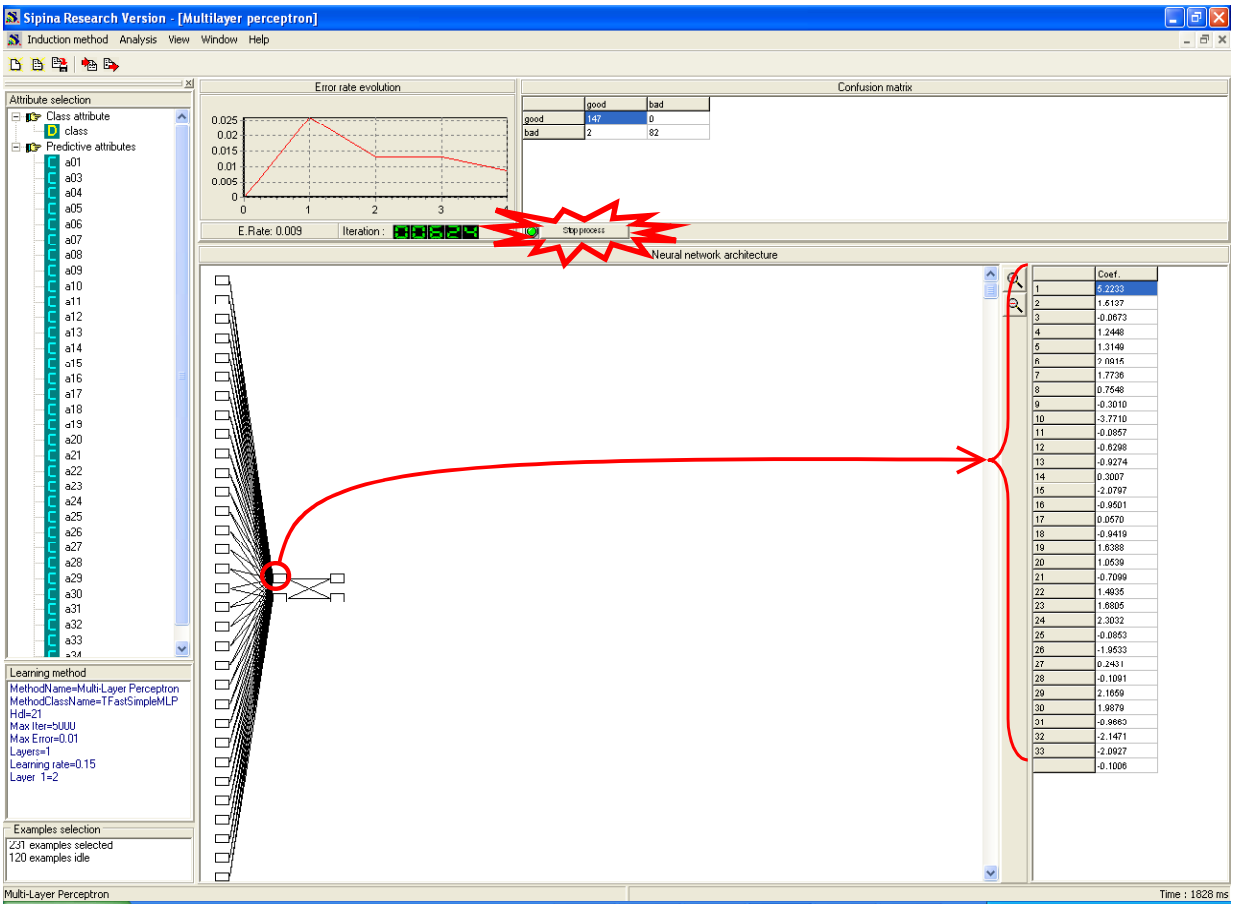
La méthode que nous avons choisie et les options associées sont décrites dans la partie gauche de la fenêtre principale de SIPINA.



	a01	a03	a04	a05	a06
1	0.35	0.72	-0.24	0.48	-0.21
2	0.35	0.72	-0.53	0.63	-1.03
3	0.35	0.72	-0.17	0.77	-0.25
4	0.35	0.72	-1.12	0.77	1.92
5	0.35	0.72	-0.15	0.65	-0.10
6	0.35	-1.25	-0.12	-1.35	-0.51
7	0.35	0.68	-0.35	0.67	-0.71
8	-2.87	-1.29	-0.10	-1.16	-0.25
9	0.35	0.64	-0.26	0.77	-0.56
10	0.35	-1.33	-0.28	-1.16	-0.25
11	0.35	0.72	0.06	0.77	-0.64
12	0.35	0.72	-1.32	0.77	-2.42
13	0.35	0.72	-0.46	0.77	-0.47
14	0.35	0.72	-2.07	0.77	0.23
15	0.35	0.72	0.06	0.77	-0.19
16	0.35	-0.26	-2.23	0.77	0.33
17	0.35	0.72	0.04	0.77	-0.27
18	-2.87	-1.29	-0.10	-3.08	-2.42

Apprentissage

Pour lancer l'apprentissage, nous sélectionnons le menu ANALYSIS / LEARNING. Une nouvelle fenêtre apparaît, décrivant le réseau dans sa partie basse et montrant l'évolution du taux d'erreur au fur et à mesure des itérations dans sa partie haute.



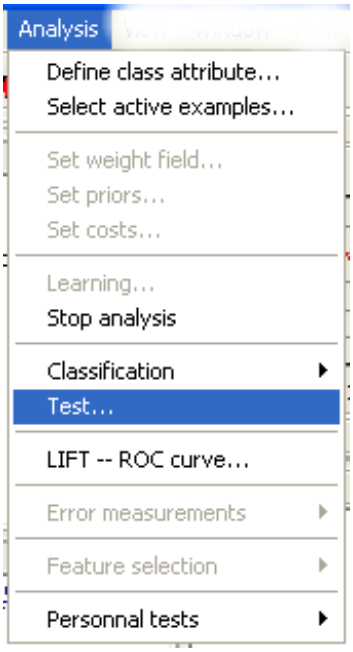
Error rate evolution indique la décroissance du taux d'erreur, dans notre cas, nous constatons qu'il est descendu à 0.009, déclenchant l'arrêt de l'apprentissage à la 624ème itération. La matrice de confusion associée est affichée dans la partie droite.

Le bouton **STOP PROCESS** joue un rôle important dans ce dispositif, il nous donne la possibilité d'interrompre le processus à tout moment. Ca peut être le cas par exemple si nous constatons que le taux d'erreur ne décroît plus au bout d'un certain nombre d'itérations.

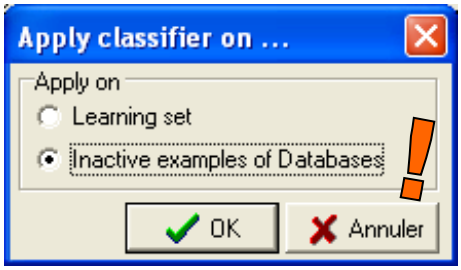
Enfin, si nous sélectionnons un des neurones du réseau, SIPINA affiche les poids synaptiques associés à ce neurone. Il est possible de les copier et de les coller dans un tableur. Nous devrons en revanche le faire manuellement pour chaque neurone, ce qui se révéler être très fastidieux lorsque le réseau est un tant soit peu complexe.

Evaluation sur le fichier test

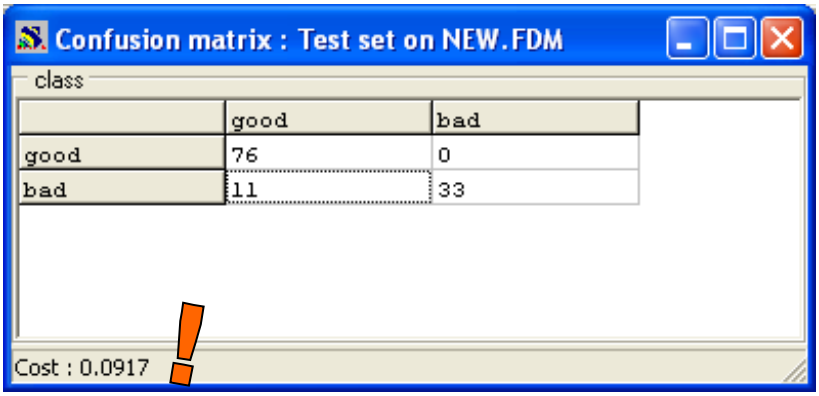
Reste maintenant à appliquer le modèle de prédiction sur l'ensemble test, n'ayant pas servi à la construction du réseau. Pour ce faire, nous cliquons sur le menu ANALYSIS / TEST.



Dans la boîte de dialogue qui apparaît nous choisissons d'appliquer le modèle sur les observations qui n'ont pas été sélectionnées pour l'apprentissage.



Une nouvelle fenêtre avec la matrice de confusion apparaît, dans notre cas le taux d'erreur en test est de 0.0917.

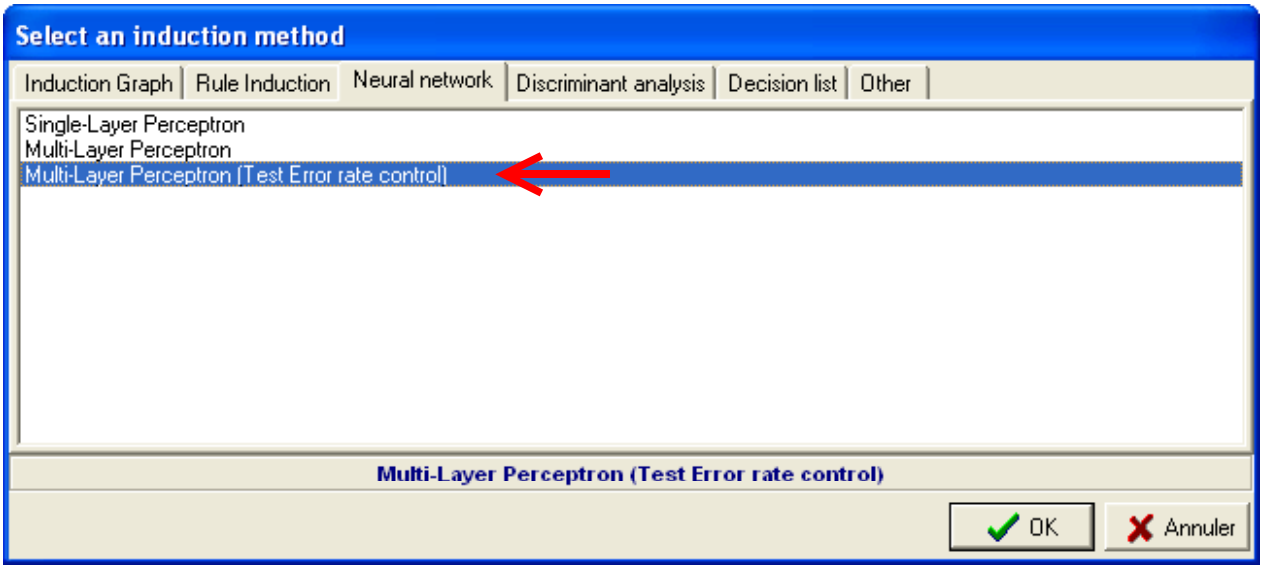


NB : Il ne faut pas s'émouvoir outre mesure des différences de performances avec TANAGRA, l'échantillon étant réduit, les résultats sont entachés d'une certaine variabilité.

Utilisation de l'échantillon de validation

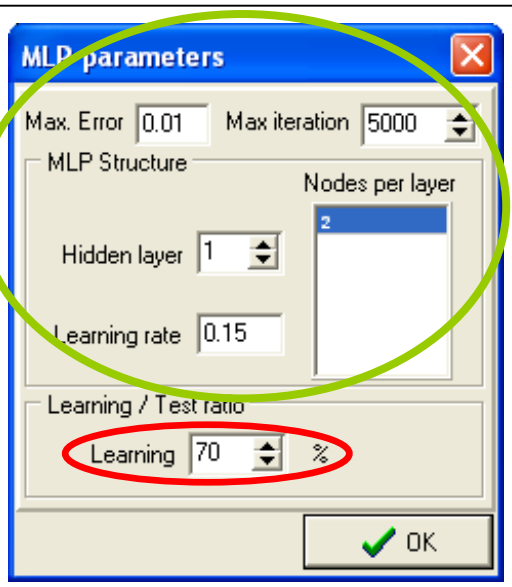
A la différence de TANAGRA, nous avons la possibilité de suivre le processus d'induction dans SIPINA. Nous constatons de visu la décroissance de l'erreur, nous pouvons même stopper l'apprentissage si nous jugeons que poursuivre n'améliorera pas le modèle. Dans ce cadre, disposer d'une courbe d'erreur non biaisée peut se révéler très utile, il devient réellement intéressant de subdiviser l'échantillon d'apprentissage en deux parties : une première qui sert à calculer les poids synaptiques, nous appellerons dorénavant cette partie « échantillon d'entraînement » ; une seconde, dite communément « échantillon de validation[2] », qui permet de suivre l'évolution de l'erreur.

Fermons toutes les fenêtres de résultats dans SIPINA en cliquant sur le menu WINDOW / CLOSE ALL. Pour intégrer la création de l'échantillon de validation dans la construction d'un réseau de neurones, nous activons de nouveau le menu INDUCTION METHOD / STANDARD ALGORITHM. Nous sélectionnons l'option MULTILAYER PERCEPTRON (TEST ERROR RATE CONTROL).

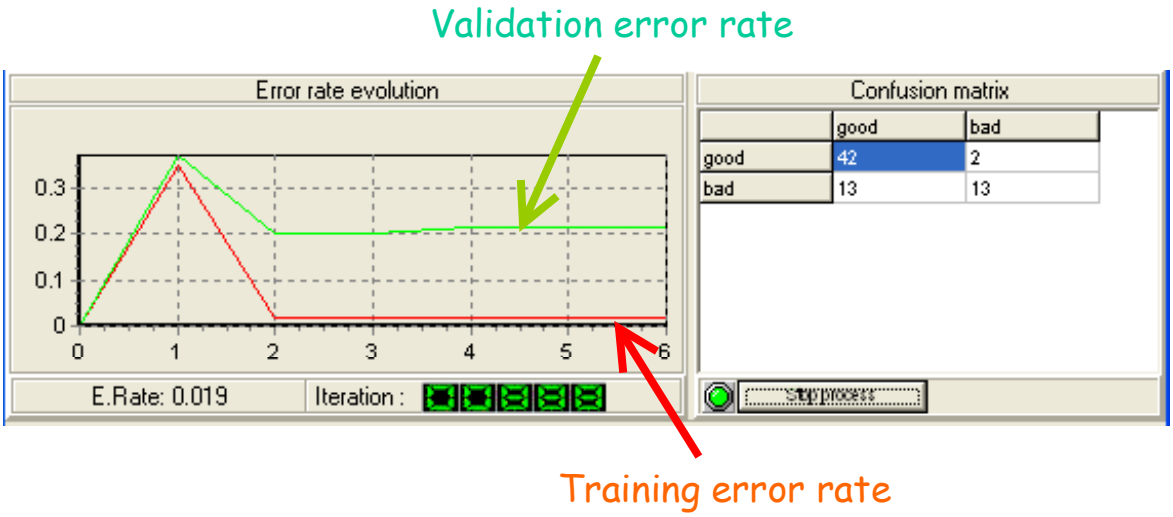


Dans la boîte de paramétrage, 70 % des données d'apprentissage seront réservés au calcul des poids synaptiques (70% de 231 = 161 exemples), le reste sera dédié à la validation (231 – 161 = 70 observations). Nous conservons les mêmes paramètres que précédemment pour ce qui est des autres options.

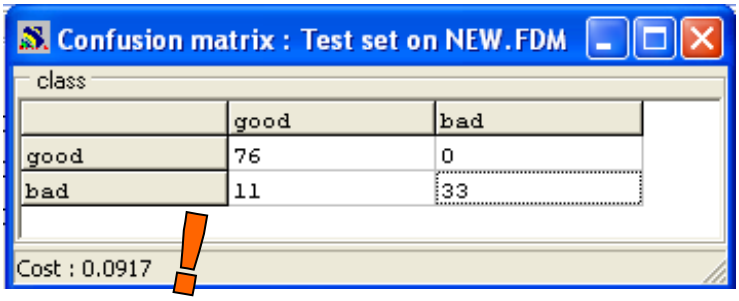
[2] Dans certains logiciels, cet échantillon est appelé « échantillon test », c'est le cas de SIPINA.



Pour relancer les calculs, nous activons le menu ANALYSIS / LEARNING, deux courbes maintenant apparaissent dans la fenêtre de suivi, nous constatons que si l'erreur calculée sur l'échantillon d'entraînement descend rapidement, l'erreur calculée sur les données de validation stagne très rapidement. Dans certains cas, ce taux en validation peut se dégrader lorsqu'il y a sur-apprentissage.

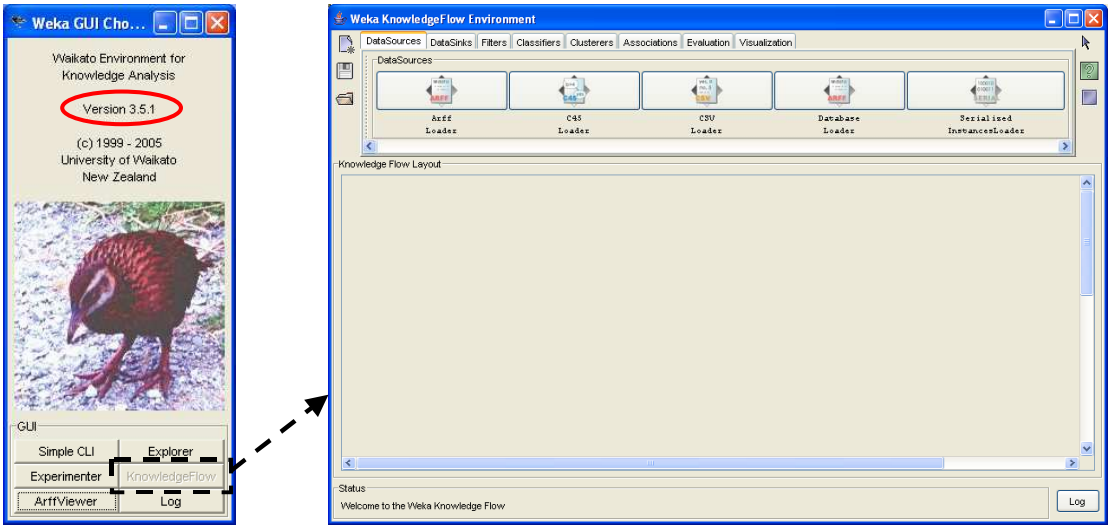


En appliquant ce réseau sur les données test (ANALYSIS / TEST), nous obtenons la matrice de confusion suivante, avec un taux d'erreur de 0.0917, identique à la précédente.



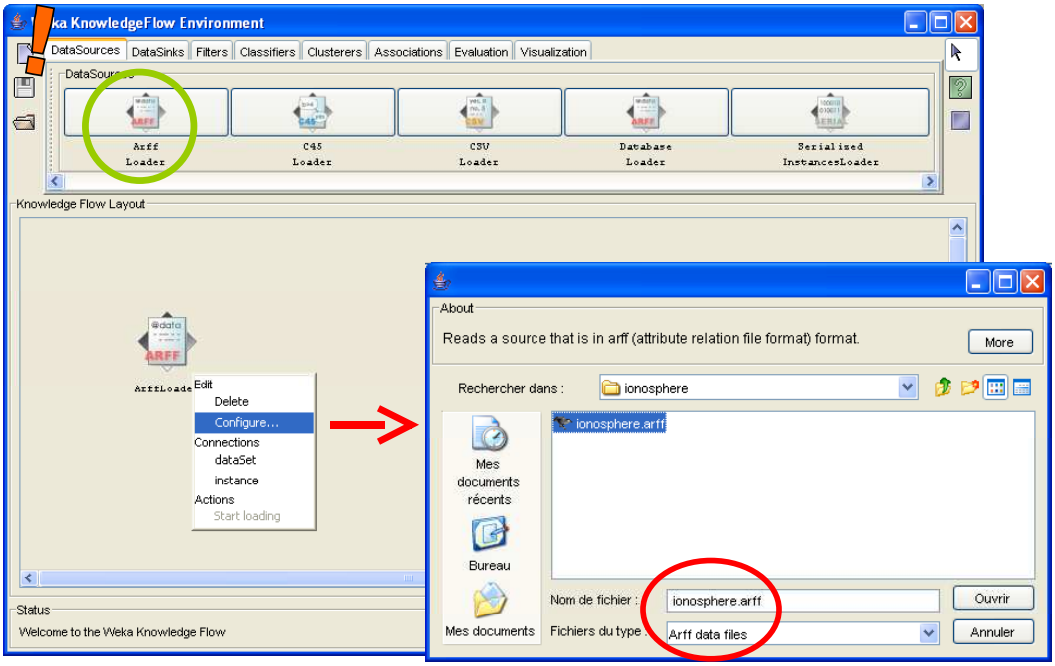
Construire un Perceptron Multicouches avec WEKA

Au lancement de WEKA (accessible sur le site → http://www.cs.waikato.ac.nz/ml/weka/), un panneau permet de choisir le mode d'exécution du logiciel. Nous choisissons le mode **KNOWLEDGE FLOW**. Nous avons utilisé la version **3.5.1** dans ce didacticiel. Nous parvenons alors dans l'espace de travail dans lequel nous allons définir nos traitements. Dans la partie haute, nous trouvons les icônes organisées dans des palettes, ils représentent les opérateurs de traitement.



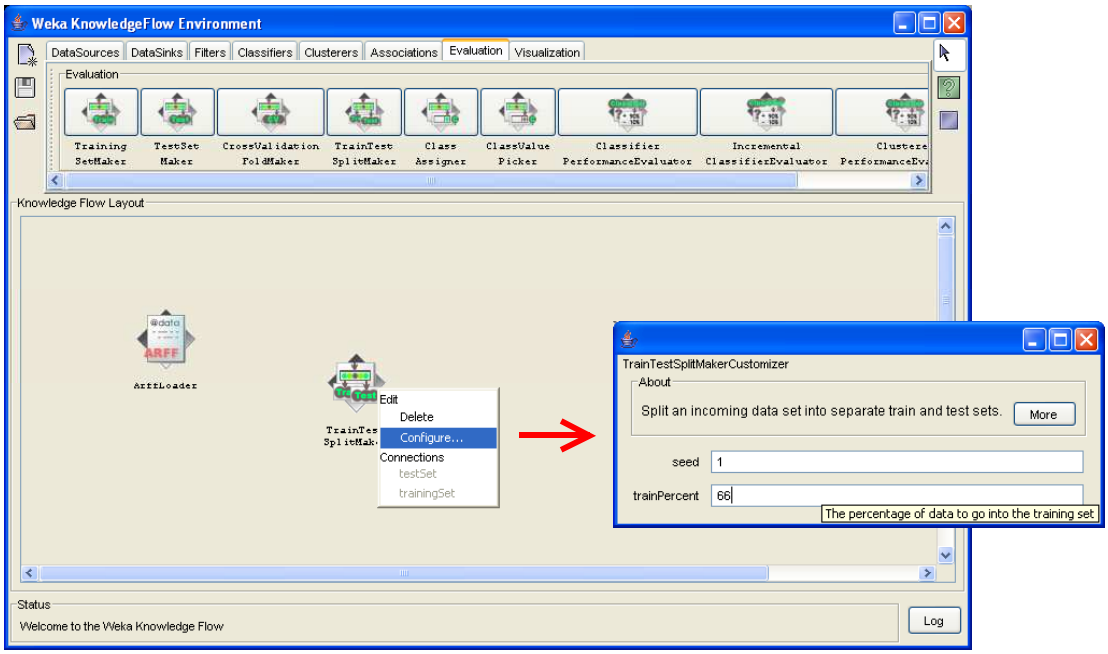
Charger les données

Le composant ARFF LOADER (onglet DATASOURCES) permet de charger les données. Nous le plaçons dans l'espace de travail et le configurons de manière à traiter le fichier IONOSPHERE.

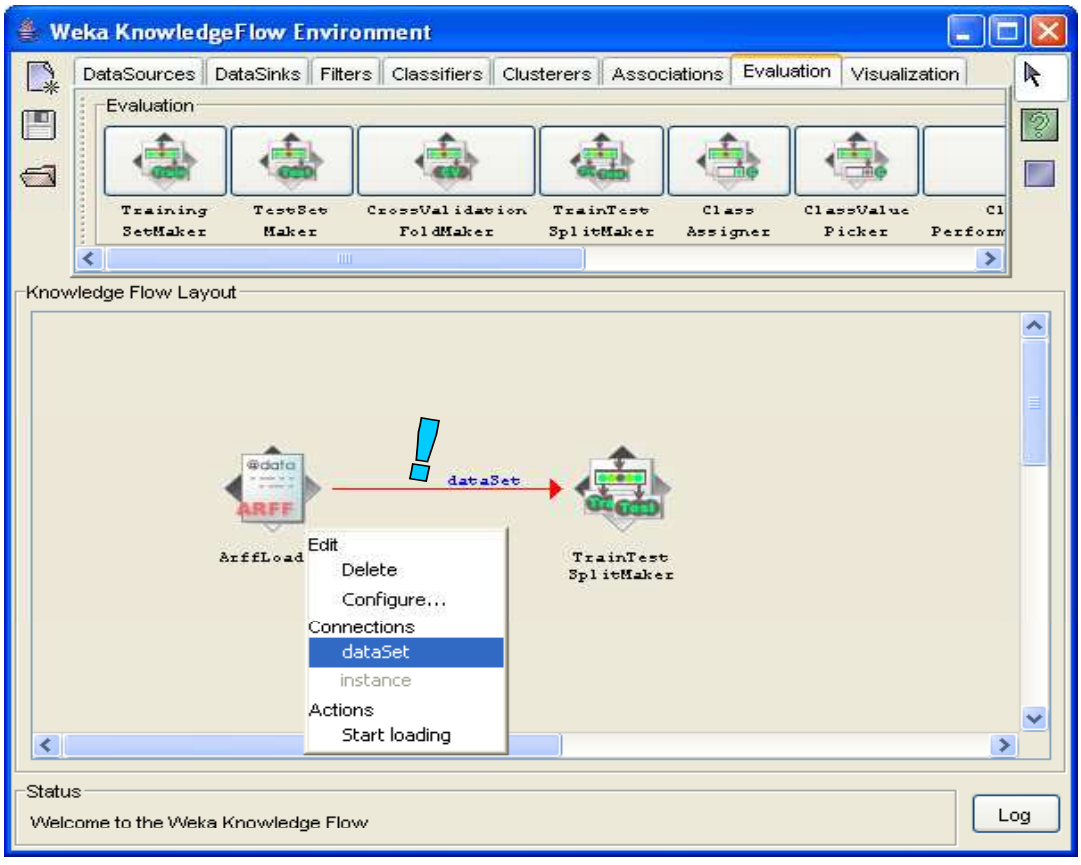


Subdiviser les données

Nous voulons subdiviser notre fichier en échantillon d'apprentissage et test. Le composant TRAINTEST SPLITMAKER (onglet EVALUATION) permet de le faire, nous le configurons de manière à ce que 66% des observations soient consacrés à la construction du modèle.



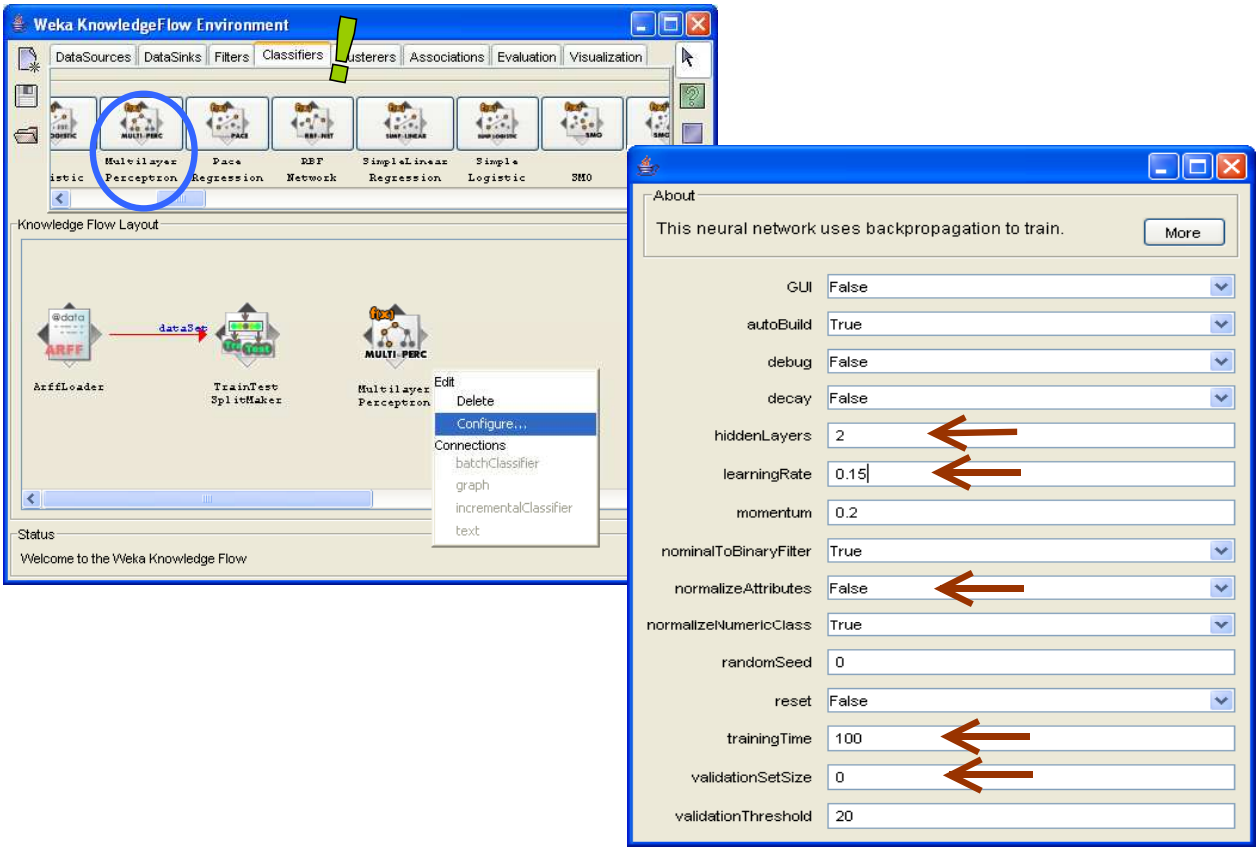
Nous pouvons alors connecter le composant ARFF LOADER à ce dernier composant en utilisant la connexion de type DATASET.



Définir et paramétrer la méthode d'apprentissage

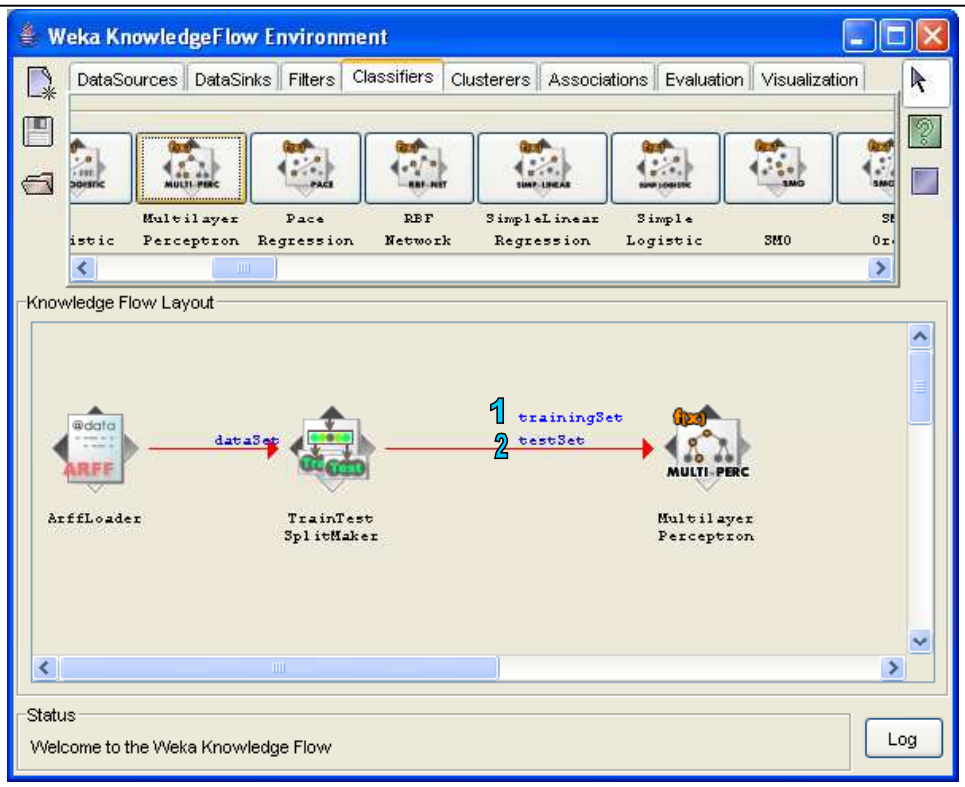
Il n'est pas nécessaire de désigner de manière explicite les descripteurs et la variable à prédire dans WEKA. Par défaut, la dernière colonne correspond à la variable à prédire, et toutes les autres variables représentent les attributs prédictifs. Si ce n'était pas le cas, nous aurions eu à utiliser le composant CLASS ASSIGNER.

Nous cherchons la méthode d'apprentissage dans l'onglet CLASSIFIERS, nous insérons le composant MULTILAYER PERCEPTRON. Nous la paramétrons en cliquant sur le menu CONFIGURE.

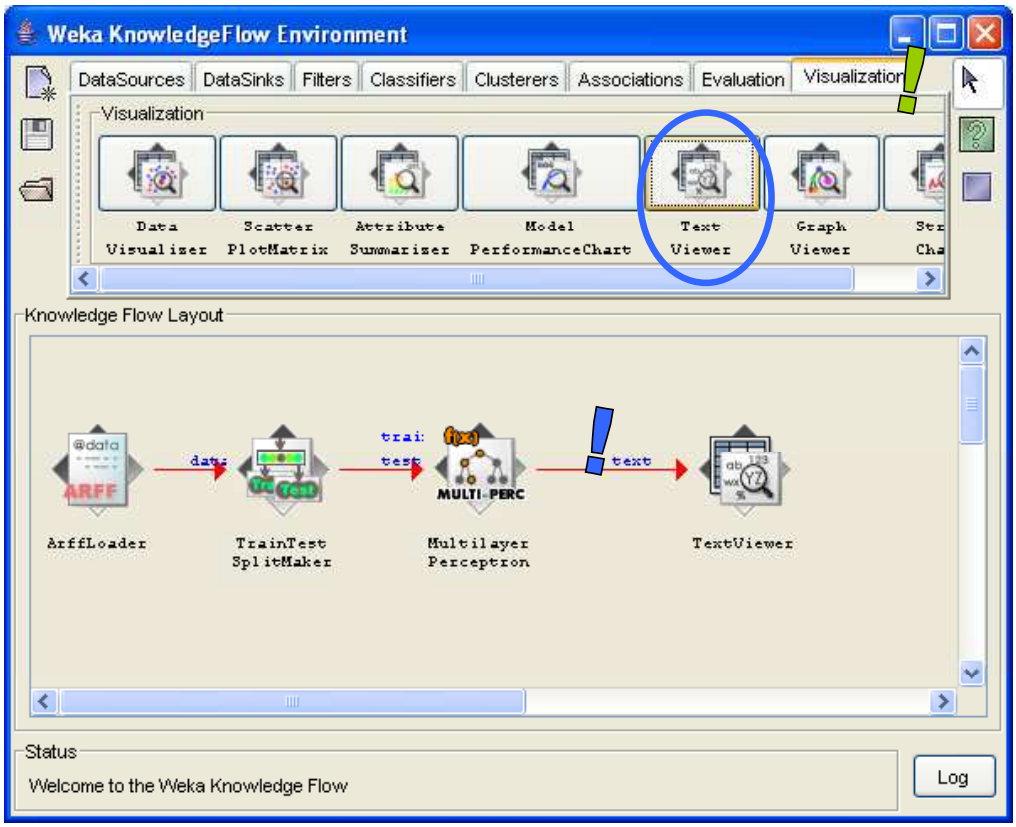


La couche cachée comporte 2 neurones (HIDDENLAYERS), la constante d'apprentissage est égale à 0.15 (LEARNING RATE), les descripteurs ne sont pas transformés (NORMALIZE ATTRIBUTES = FALSE), le nombre d'itérations est égal à 100 (TRAINING TIME), et enfin, nous n'utilisons pas d'échantillon de validation (VALIDATION SET SIZE = 0).

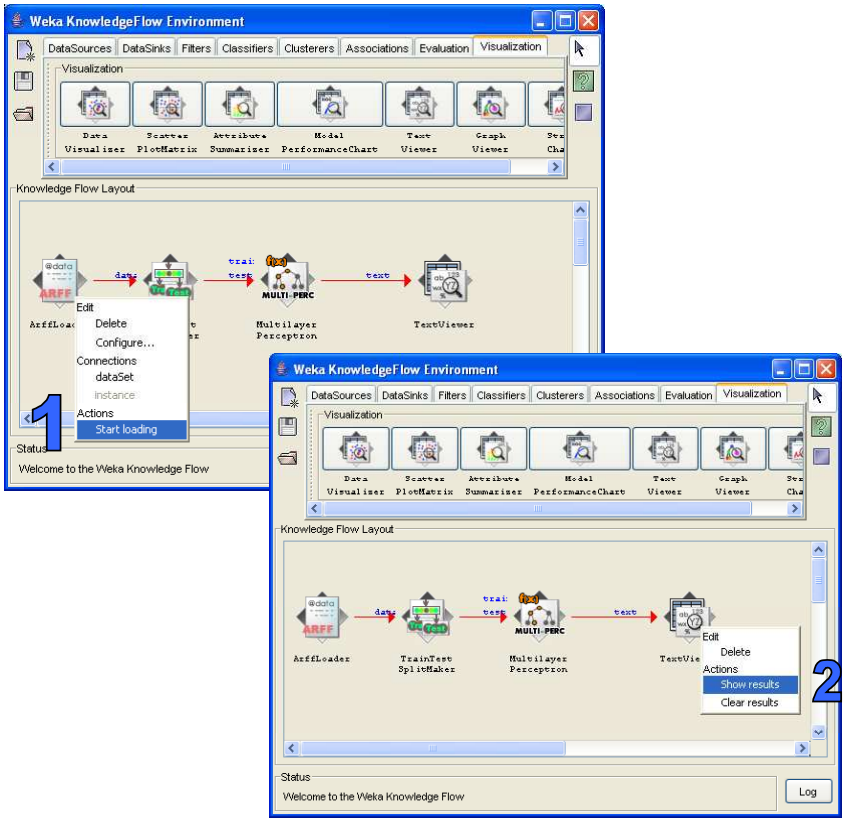
Nous connectons 2 fois le composant TRAINTEST SPLITMAKER à ce composant d'apprentissage pour passer les données d'apprentissage (1 – Training set) et de test (2 – Test set).



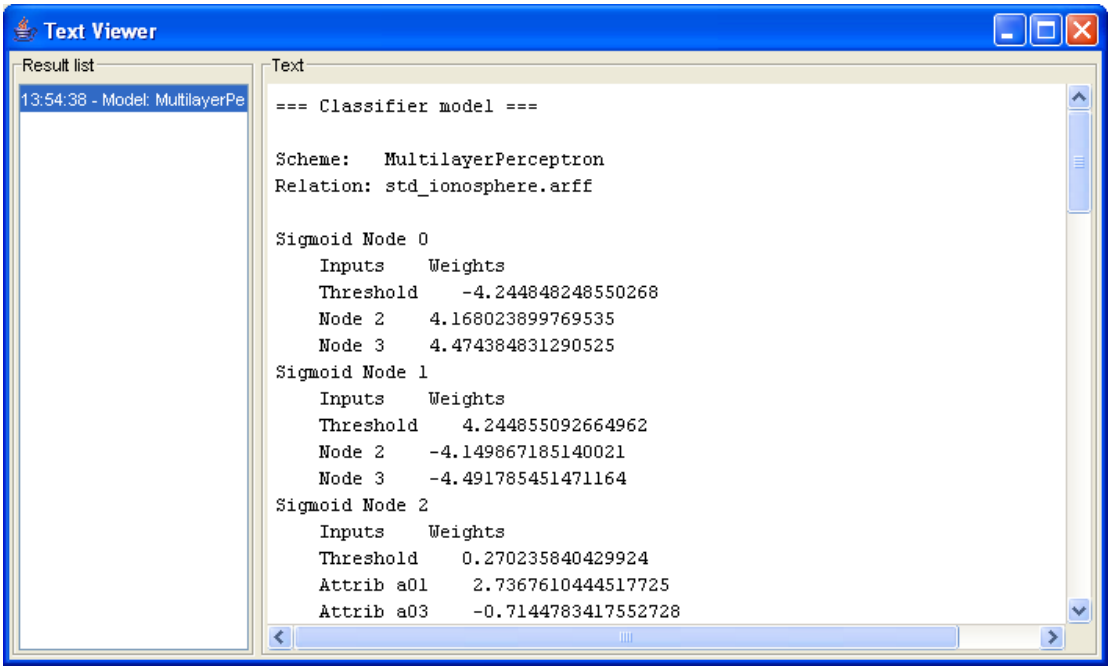
Nous devons placer un composant de visualisation pour accéder aux résultats. Le composant TEXT VIEWER convient pour cela, nous le trouvons dans l'onglet VISUALIZATION. La connexion est de type TEXT.



Reste à lancer l'apprentissage, nous devons pour ce faire activer le menu START LOADING du composant d'accès aux données. Et pour visualiser les résultats, nous cliquons sur le menu SHOW RESULTS du composant TEXT VIEWER.

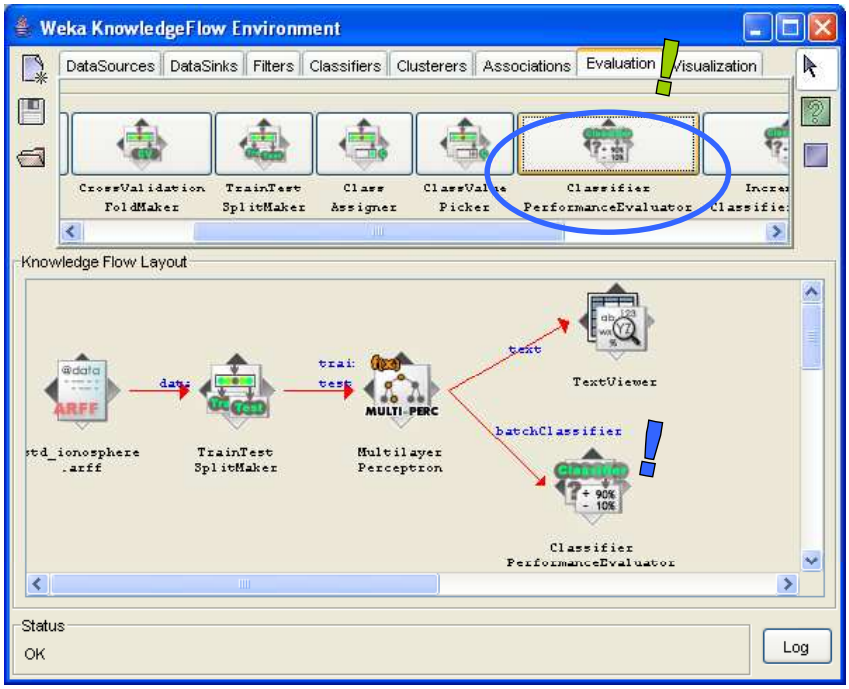


La fenêtre de résultats nous affiche essentiellement les poids synaptiques.

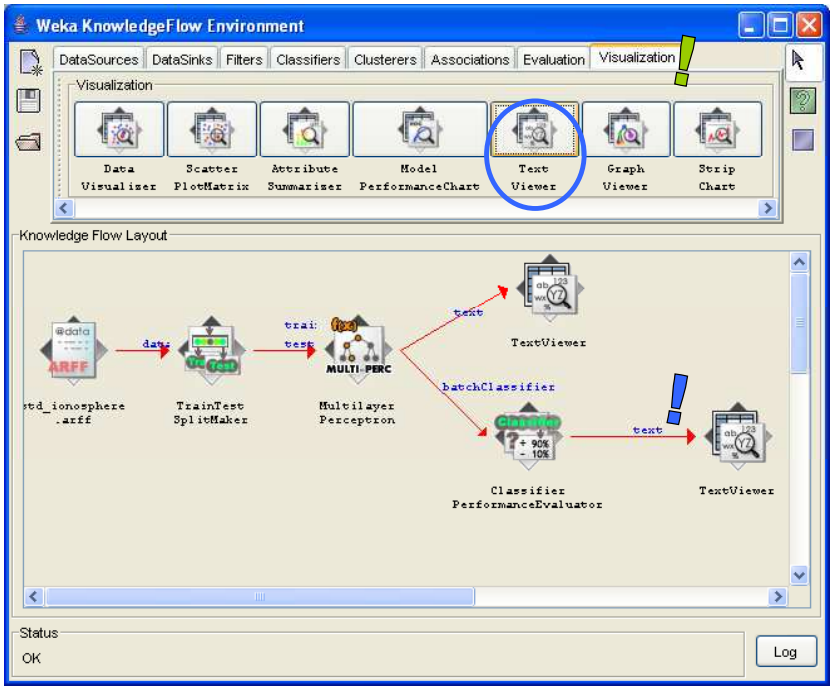


Evaluation du réseau

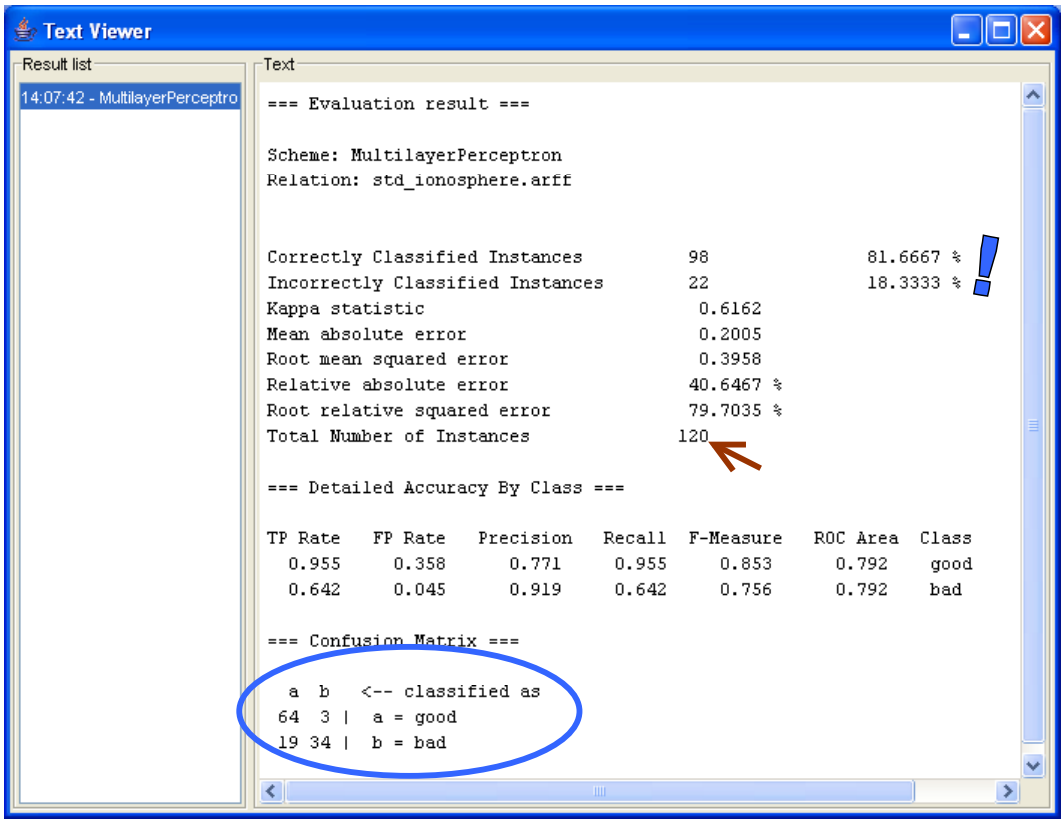
Nous voulons maintenant évaluer les performances du réseau sur les données test. Nous devons ajouter deux composants dans le diagramme. Le premier consiste à appliquer le modèle sur les données test, il s'agit du composant CLASSIFIER PERFORMANCE EVALUATOR (onglet EVALUATION), nous lui connectons la méthode d'apprentissage en utilisant la connexion de type BATCH CLASSIFIER.



Le second est destiné à l'affichage des résultats, nous utilisons une nouvelle fois le composant TEXT VIEWER. Nous lui connectons le composant CLASSIFIER PERFORMANCE EVALUATOR en utilisant la connexion de type TEXT.



Pour accéder aux résultats, nous devons tout d'abord ré-exécuter le diagramme en cliquant sur le menu START LOADING du composant ARFF LOADER, puis cliquer sur le menu SHOW RESULTS du composant TEXT VIEWER pour les visualiser.



Le réseau est bien appliqué sur les 120 observations qui composent l'échantillon test, nous disposons de la matrice de confusion, le taux d'erreur est de 18.33%.

Conclusion

Quel que soit le logiciel utilisé, nous constatons à travers cet exemple qu'ils obéissent à la même logique de fonctionnement. Le tout est de trouver les composants, les menus et les boîtes de dialogues adéquats pour définir les bons paramètres de l'analyse. A partir de là, la mise en œuvre des réseaux de neurones est finalement assez simple.

Après, il faut interpréter les résultats, c'est une toute autre histoire, surtout en ce qui concerne le perceptron multicouches.

Enfin, l'apprentissage étant basé sur des heuristiques et notre échantillon de taille réduite, il est tout à fait naturel que nous observions des différences assez sensibles dans les performances mesurées sur les échantillons tests (qui ne sont pas constitués des mêmes observations d'ailleurs d'un logiciel à l'autre).