

# 1 Objectif

## **Comparer les performances des différentes implémentations libres (gratuites) des SVM.**

Les machines à vecteurs de support (ou séparateur à vaste marge) sont des techniques d'apprentissage supervisé qui s'appuient sur deux idées fortes : (1) le principe de la maximisation de la marge ; (2) lorsque les données ne sont pas linéairement séparables, il est possible, par le principe des noyaux, de se projeter dans un espace de plus grande dimension pour trouver la solution adéquate, sans avoir à former explicitement ce nouvel espace de représentation (Voir [Wikipédia](#)).

De par sa remarquable stabilité (faible variance), les SVM sont très performants dans les espaces à très grande dimension, lorsque le ratio nombre de variables / nombre d'observations est très élevé. Dans le cas contraire, sur des fichiers « classiques » où le nombre d'observations est comparativement plus élevé que le nombre de variables, il est illusoire d'espérer voir les SVM, linéaires tout du moins, surclasser des techniques telles que la régression logistique ou l'analyse discriminante linéaire.

Nous nous plaçons dans un cadre particulièrement favorable aux SVM dans ce didacticiel. Nous souhaitons prédire la famille d'appartenance de séquences de protéines à partir de la présence - absence de suites de 4 acides aminées (4-grams). Nous traitons un problème à 2 classes, nous disposons de 135 observations et 31809 descripteurs. **Notre objectif est de comparer le comportement de quelques implémentations libres des SVM.** Ce document vient en complément d'autres comparaisons que nous avons réalisés dans des contextes différents (<http://tutoriels-data-mining.blogspot.com/search/label/Tanagra%20et%20les%20autres>).

Ce comparatif est intéressant à plus d'un titre. Tout d'abord, nous aurons la possibilité d'évaluer les différentes implémentations des SVM, tant en temps de calcul qu'en qualité de prédiction. L'optimisation reposant sur des heuristiques, il est normal que les temps de calcul soient différents, mais il se peut également que les performances en classement qui en résultent ne soient pas identiques. Pouvoir les situer est une démarche importante. Dans les publications utilisant les SVM, on devrait non seulement dire « nous avons utilisé les SVM, avec tel noyau », mais aussi préciser « quelle implémentation des SVM », tant parfois les résultats peuvent diverger d'un logiciel à l'autre.

Autre point important. A y regarder de plus près, nous sommes également confrontés à un problème de volumétrie ici. Même si le nombre d'observations est faible, le nombre de variables, lui, est élevé. Or, les logiciels évalués chargent la totalité des données en mémoire vive. De nouveau la mémoire disponible devient un goulot d'étranglement. De plus, certains logiciels sont calibrés pour gérer un grand nombre d'observations avec relativement peu de variables (de l'ordre d'une centaine). C'est le cas clairement pour Tanagra. En les plaçant dans un contexte tout à fait inhabituel, nous pourrions évaluer les performances et la robustesse des structures internes.

Nous en tiendrons aux SVM linéaires dans ce didacticiel qui s'inscrit dans la série dédiée aux comparaisons des logiciels. Les descriptions de la machine utilisée et des logiciels mis en compétition sont disponibles dans <http://tutoriels-data-mining.blogspot.com/2008/09/traitement-de-gros-volumes-comparaison.html>

De manière générale, nos comparaisons reposent essentiellement sur le temps de traitement et l'occupation mémoire (maximale). Nous ferons confiance à l'indication fournie par les logiciels lorsqu'ils affichent le temps de calcul. Sinon nous utiliserons simplement un chronomètre. Il est de toute manière absurde de vouloir se battre au centième de seconde près, c'est plutôt l'ordre de

grandeur qui est important pour nous. Quant à l'occupation mémoire, nous nous fierons aux indications du questionnaire de tâche de Windows. Nous relèverons la mémoire allouée au chargement des données et le pic durant le traitement.

Dans le cas précis des SVM, comme les implémentations peuvent influencer sur les performances en prédiction, nous mesurerons également le taux d'erreur en validation croisée (5-fold). Il ne faut pas trop se focaliser sur ce résultat toutefois, sauf si les écarts sont manifestement importants. En effet, les subdivisions pour la validation croisée étant aléatoires, les partitions ne sont pas les mêmes d'un logiciel à l'autre, les résultats sont empreints d'une certaine variabilité. Nous utiliserons surtout ce critère pour repérer les éventuels « plantages » lors de l'apprentissage du modèle.

Voici la liste des logiciels utilisés dans ce comparatif :

Logiciel	Version	URL
ORANGE	1.0b2	<a href="http://www.ailab.si/orange/">http://www.ailab.si/orange/</a>
RAPIDMINER	Community Edition 4.2	<a href="http://rapid-i.com/">http://rapid-i.com/</a>
TANAGRA	1.4.27	<a href="http://eric.univ-lyon2.fr/~ricco/tanagra/">http://eric.univ-lyon2.fr/~ricco/tanagra/</a>
WEKA	3.5.6	<a href="http://www.cs.waikato.ac.nz/ml/weka/">http://www.cs.waikato.ac.nz/ml/weka/</a>

Ces logiciels proposent leurs propres implémentations. Ils intègrent également des bibliothèques externes. Parmi les plus utilisées figure LIBSVM (<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>). Son succès repose à la fois sur ses performances (rapidité, qualité de l'optimisation), mais aussi, de manière tout à fait pragmatique, sur la facilité avec laquelle il est possible de l'intégrer dans les logiciels. Concernant Tanagra, il a suffi de récupérer la DLL compilée, d'écrire l'unité d'import en DELPHI avec les prototypes des fonctions, le tout est directement fonctionnel.

D'autres logiciels ont été testés. Ils ont échoué, soit au chargement des données, soit au lancement des calculs. Nous les recenserons dans la section 4. Il se peut que ces plantages soient consécutifs à une mauvaise utilisation ou un mauvais paramétrage de notre part. Ce n'est pas à exclure. Les données de ce comparatif étant accessibles en ligne, tout un chacun pourra réaliser l'expérimentation et éventuellement trouver les bons paramétrages. Un retour sur la bonne manière de faire serait très apprécié.

## 2 Données

Nous traitons un problème de discrimination de 2 familles de protéines à partir de la présence ou absence de suites d'acides aminés de longueur 4. Le nombre potentiel de descripteurs est  $20^4 = 160.000$ , bon nombre d'entre eux sont absents du fichier, nous « n'avons » que 31809 descripteurs finalement, ce qui reste assez considérable encore, surtout si l'on tient compte du fait que nous ne disposons que de 135 observations.

D'emblée, nous savons que la maîtrise de la variance est l'enjeu principal de l'apprentissage dans ce cas de figure, un terrain privilégié pour les SVM.

Le fichier WIDE\_PROTEIN\_CLASSIFICATION.TXT est compressé dans une archive<sup>1</sup>.

**Attention**, si votre système utilise « , » comme point décimal, il faudra (pour certains logiciels)

<sup>1</sup> [http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/wide\\_protein\\_classification.zip](http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/wide_protein_classification.zip)

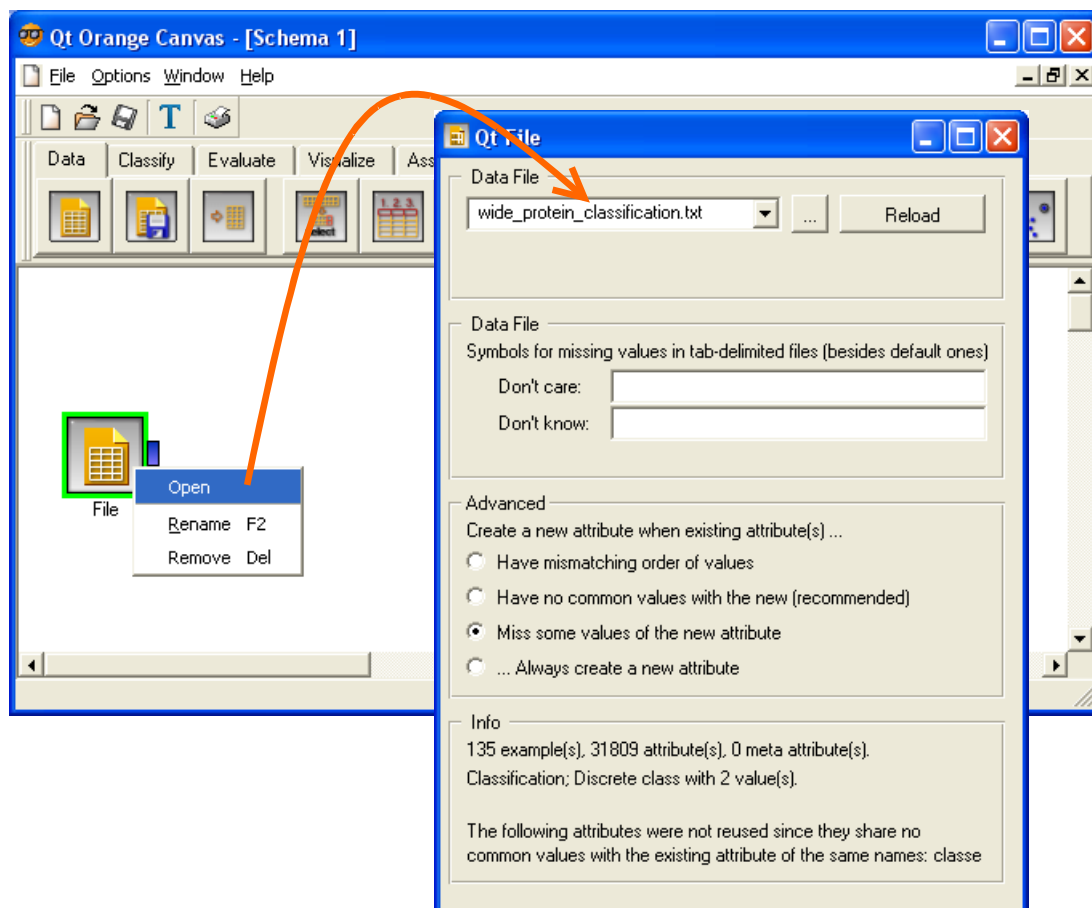
remplacer les « . » par « , » dans le fichier, en utilisant la fonction RECHERCHER/REEMPLACER d'un éditeur de texte.

## 3 Résultats des expérimentations

### 3.1 ORANGE

Au démarrage du logiciel, nous constatons immédiatement que PYTHON, qui encapsule le programme, est aussi lancé. L'occupation mémoire est de 25 Mo.

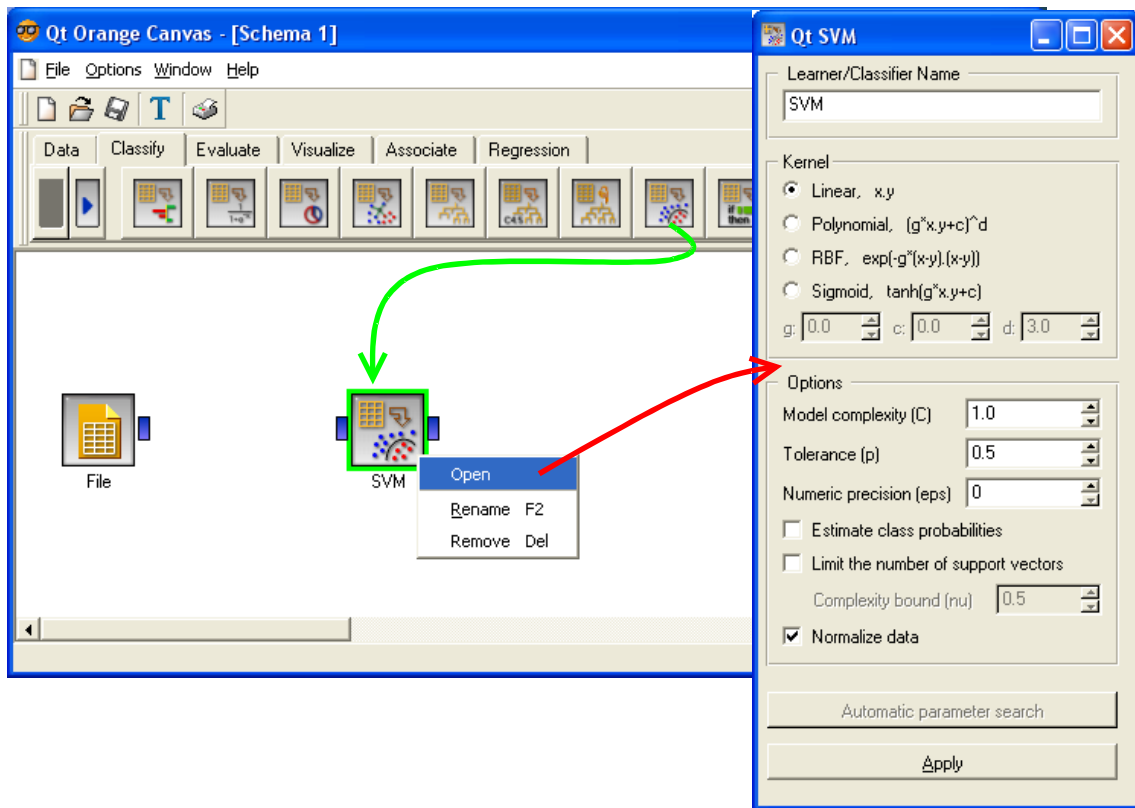
ORANGE fonctionne sous forme de filières, appelées « schémas ». Les composants sont réunis dans la barre d'outils. Nous introduisons dans un premier temps le composant FILE d'accès aux données, nous activons le menu contextuel OPEN pour désigner le fichier à charger. Attention, dès que le fichier est sélectionné, le chargement est démarré. Il n'y a pas d'inquiétude à avoir même si le logiciel semble se figer avec la boîte de dialogue encore ouverte. La durée de l'importation est de 95 secondes, l'occupation mémoire est passée à 118 Mo.



Nous insérons le composant SVM (onglet CLASSIFY) dans le schéma. Nous actionnons le menu contextuel OPEN pour accéder à la boîte de paramétrage. Nous souhaitons mettre en œuvre un SVM linéaire, avec une pénalité des mal classés (MODEL COMPLEXITY) à 1.0. Nous essaierons d'utiliser ces paramètres pour tous les logiciels de ce comparatif.

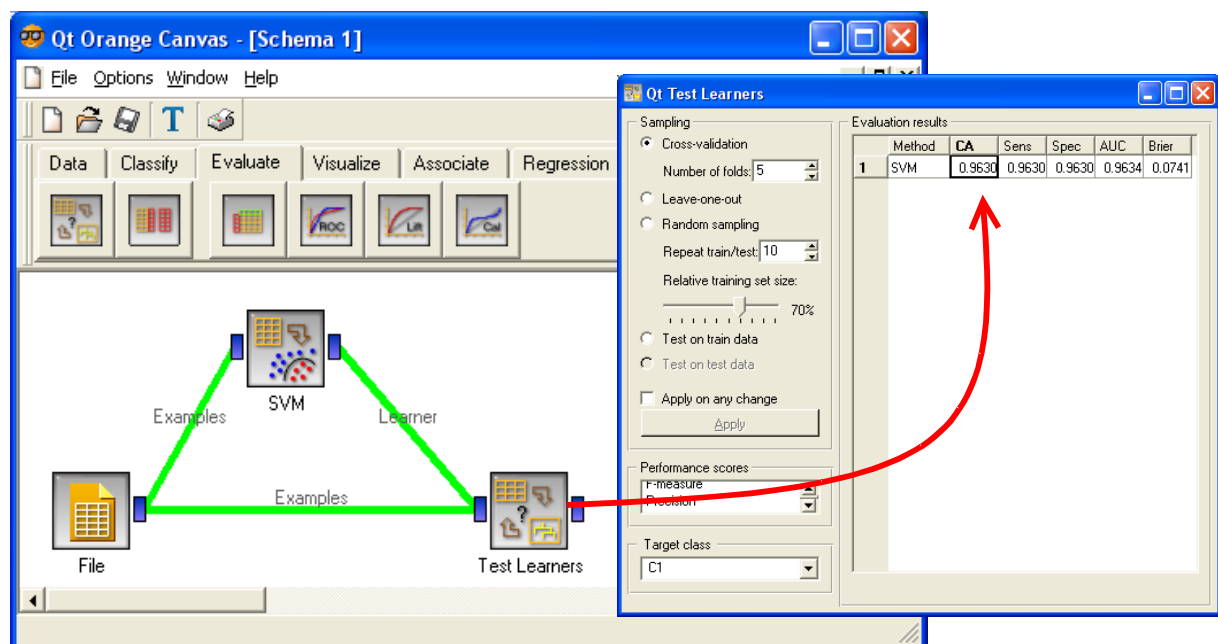
Concernant ORANGE, un fichier d'aide au format HTML décrit l'usage et les paramètres du composant SVM C:\Python25\Lib\site-packages\orange\doc\widgets\catalog\Classify\SVM.htm. Il est accessible en appuyant la touche F1 lorsque la boîte de paramétrage est ouverte.

**Remarque :** On notera une option bluffante de ORANGE dans l'aide en ligne. Il est possible de distinguer les points supports dans les graphiques nuages de points en 2 dimensions. Pédagogiquement, c'est un plus extraordinaire.



Il ne reste plus qu'à relier le composant FILE à SVM. Le calcul est lancé dès que la connexion est établie, il durera 690 secondes (plus de 11 minutes).

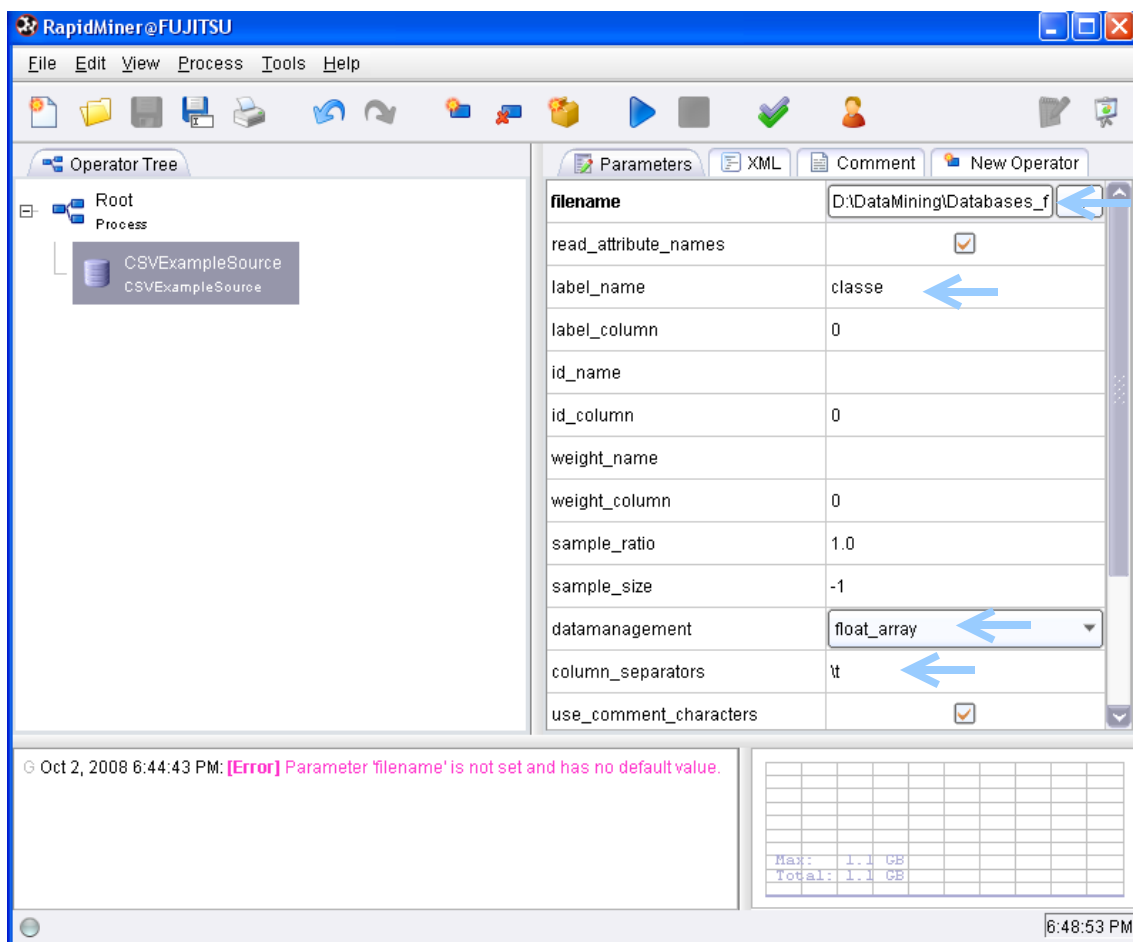
Pour obtenir le taux d'erreur en validation croisée, nous insérons le composant TEST LEARNERS dans le schéma. Là aussi, nous avons intérêt à le paramétrer (5 - validation croisée) avant de procéder aux branchements. Un taux d'erreur de 4% est annoncé.



L'occupation mémoire au plus fort de tous les calculs, lors de la validation croisée notamment, est de 406 Mo.

## 3.2 RAPIDMINER

RAPIDMINER est développé en JAVA, la machine virtuelle est automatiquement démarrée au lancement du logiciel, l'occupation mémoire est de 124 Mo. Après avoir créé un nouveau diagramme, nous insérons un CSVEXAMPLESOURCE. Attention, le paramétrage est délicat. Après avoir précisé le fichier (FILENAME), nous devons impérativement indiquer la variable à prédire « classe » dans LABEL\_NAME, les données sont stockées en simple précision (DATAMANAGEMENT ; ça ne sert à rien de stocker des 0/1 en double précision), le caractère « tabulation » sert de séparateur de colonnes (COLUMN\_SEPARATORS).



Le temps de chargement est fulgurant (5 secondes), l'occupation mémoire passe à 210 Mo.

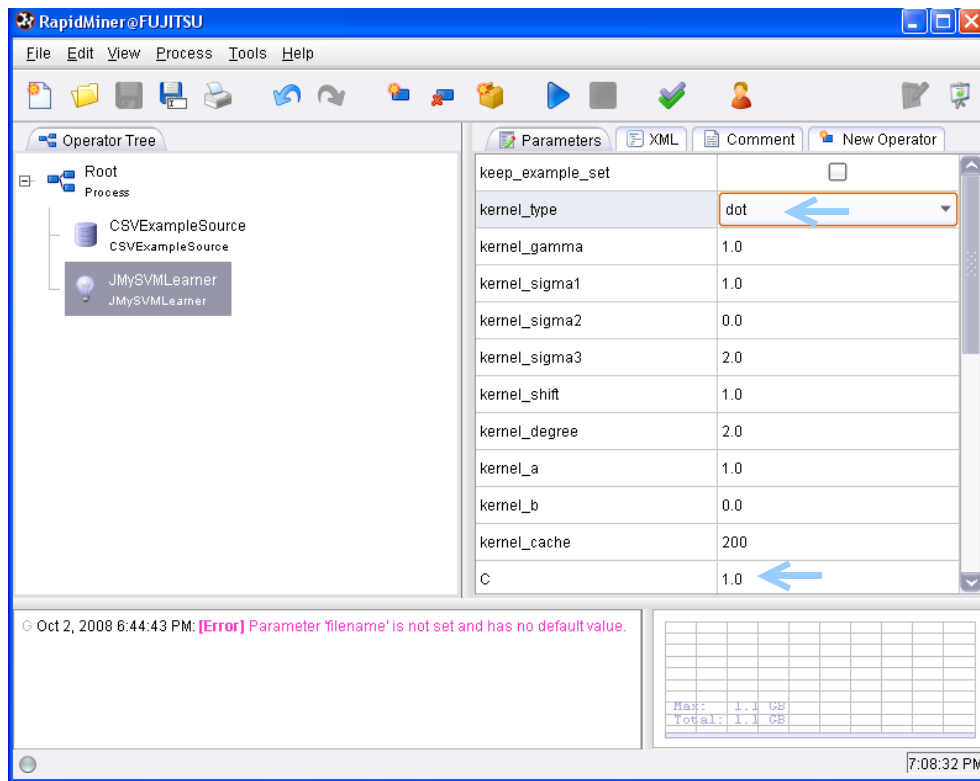
RAPIDMINER intègre deux composants dédiés aux SVM. Nous les testons tour à tour.

### 3.2.1 RAPIDMINER – JMYSVMLEARNER

Le composant JMYSVMLEARNER est une implémentation JAVA de MYSVM de Stefan Rüping (<http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/index.html>). On le retrouve dans le sous-menu NEW OPERATOR / LEARNER / SUPERVISED / FUNCTIONS.

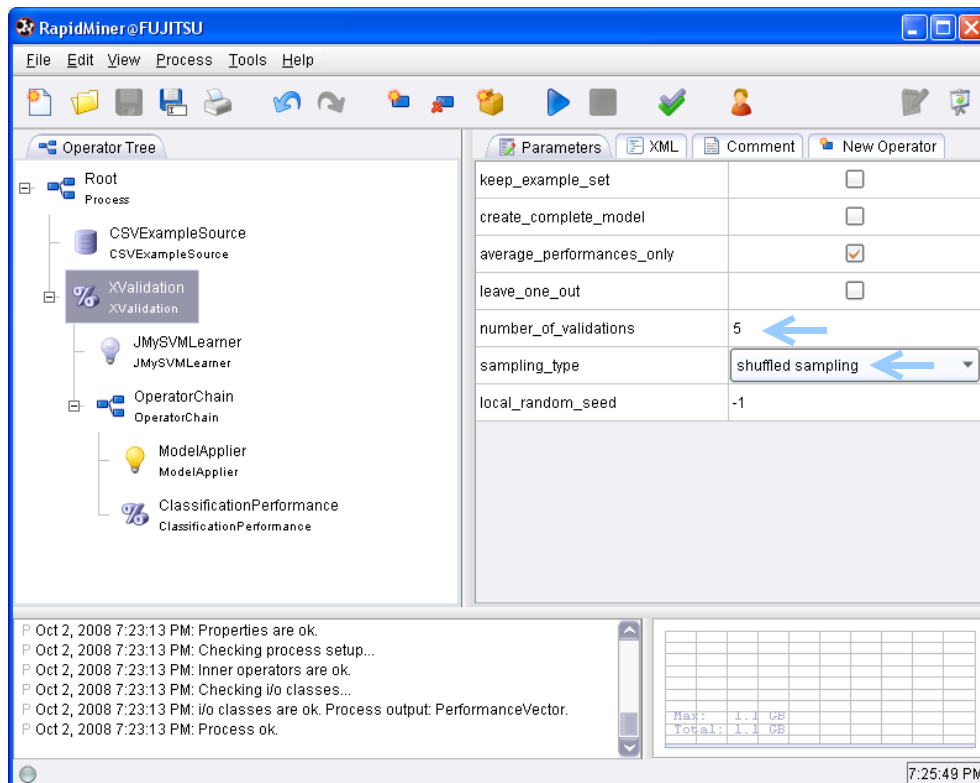
Le paramétrage s'apparente à un schéma de croix. Même si une description complète est accessible en ligne, il est difficile d'apprécier avec exactitude le rôle de chaque paramètre sur la

qualité et la rapidité de l'apprentissage. Nous nous contenterons pour notre part de préciser un noyau linéaire, avec un paramètre de complexité  $C = 1.0$ , sans rien modifier d'autre.



Il ne reste plus qu'à lancer le traitement à l'aide du bouton PLAY dans la barre d'outils. Le calcul dure 29 secondes, l'occupation mémoire passe à 338 Mo.

Pour évaluer les performances en classement, il faut introduire la validation croisée.



Etrangement, alors que la structure du diagramme RAPIDMINER était très facile à appréhender jusqu'ici, la mise en place de la validation croisée nous oblige à des contorsions assez phénoménales. J'imagine que les auteurs du logiciel ont souhaité introduire un haut niveau de généralité dans la définition des opérations, nous permettant de mettre en oeuvre la validation croisée à la fois pour le classement et la régression. Après coup, on se rend compte de la cohérence de la démarche. La validation croisée est bien la répétition de deux opérations ; apprentissage sur une fraction des données, application du modèle sur une autre fraction. Mais encore fallait-il trouver le bon enchaînement des composants.

Nous avons demandé `NUMBER_OF_VALIDATIONS = 5` et `SAMPLING_TYPE = SHUFFLED SAMPLING` pour le composant `XVALIDATION`, coché `CLASSIFICATION_ERROR` pour `CLASSIFICATIONPERFORMANCE`.

The screenshot shows the RapidMiner PerformanceVector window. The 'Criterion Selector' on the left lists 'classification\_error'. The main area displays the following summary and table:

**classification\_error: 11.11% +/- 6.20% (mikro: 11.11%)**

	true C1	true C2	class precision
pred. C1	39	0	100.00%
pred. C2	15	81	84.38%
class recall	72.22%	100.00%	

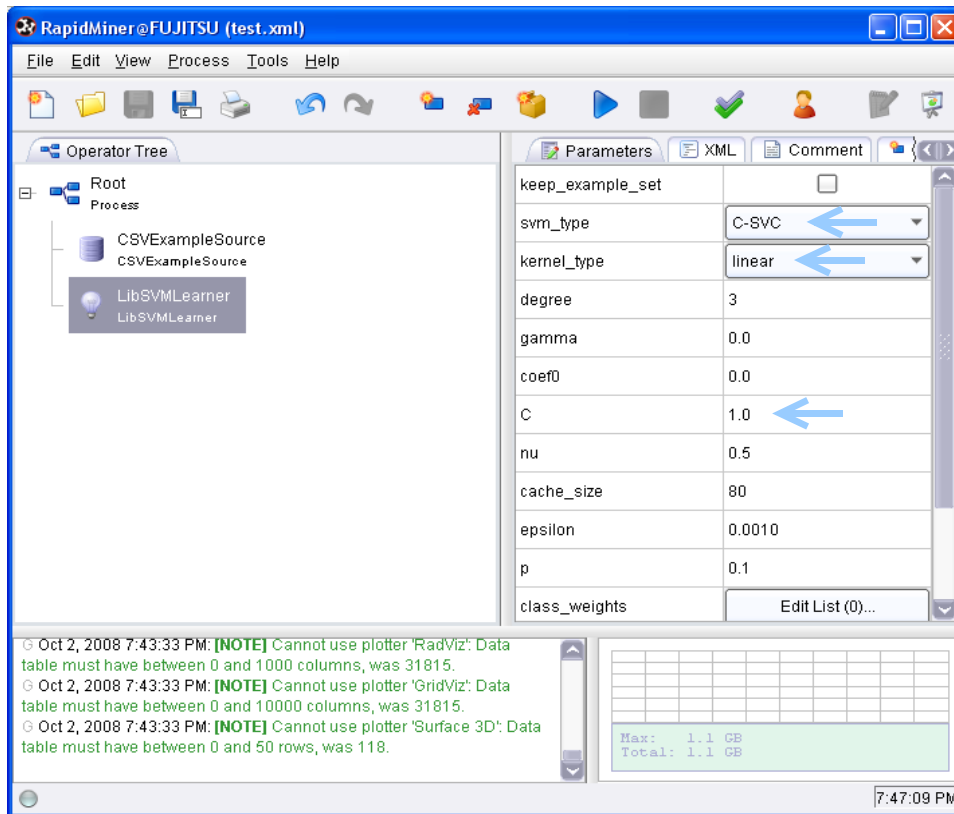
Below the table, the process flow is visible, including XValidation, JMySVM Learner, OperatorChain, ModelApplier, and ClassificationPerformance. A status message indicates the process finished after 445 seconds. The bottom right corner shows the time 7:36:41 PM.

Après 445 secondes, RAPIDMINER nous fournit la matrice de confusion synthétique, avec un taux d'erreur estimé de 11% (15 erreurs sur 135 observations).

### 3.2.2 RAPIDMINER – LIBSVM

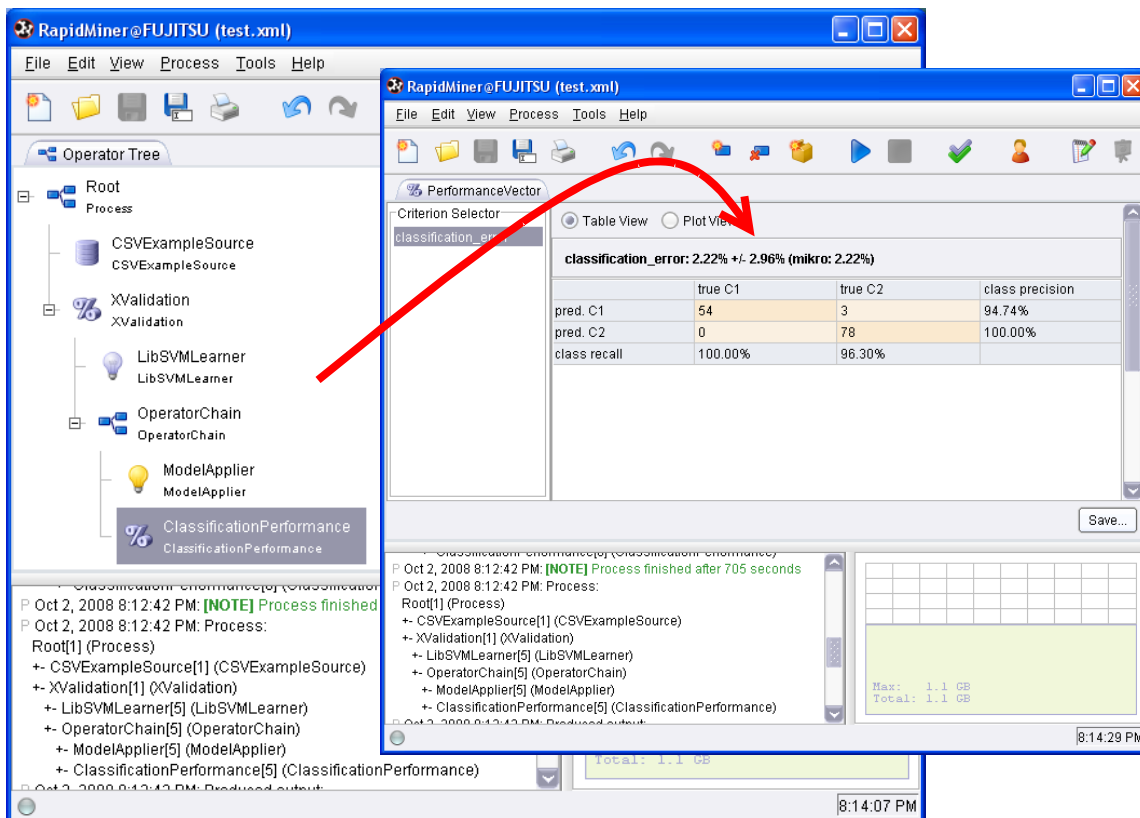
RAPIDMINER intègre la bibliothèque LIBSVM sous une forme native. Le code original a été traduit en JAVA. Il n'y a donc pas d'appel à une DLL externe compilée.

Nous construisons un nouveau diagramme. Après avoir re-précisé la source de données, nous introduisons le composant LIBSVMLEARNER. Les paramètres sont décrits sur le site de LIBSVM, le plus important pour nous est de demander un noyau linéaire avec un paramètre de pénalité égal à 1 (<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>).



Le temps d'apprentissage est de 9 secondes (!). Mais le résultat ne s'affiche pas immédiatement. Un séparateur linéaire ayant été produit, RAPIDMINER s'évertue à en énumérer les coefficients, et ils sont nombreux, retardant l'affichage. L'occupation mémoire est de 442 Mo.

Pour estimer le taux d'erreur en validation croisée, nous devons reproduire le schéma ci-dessus.

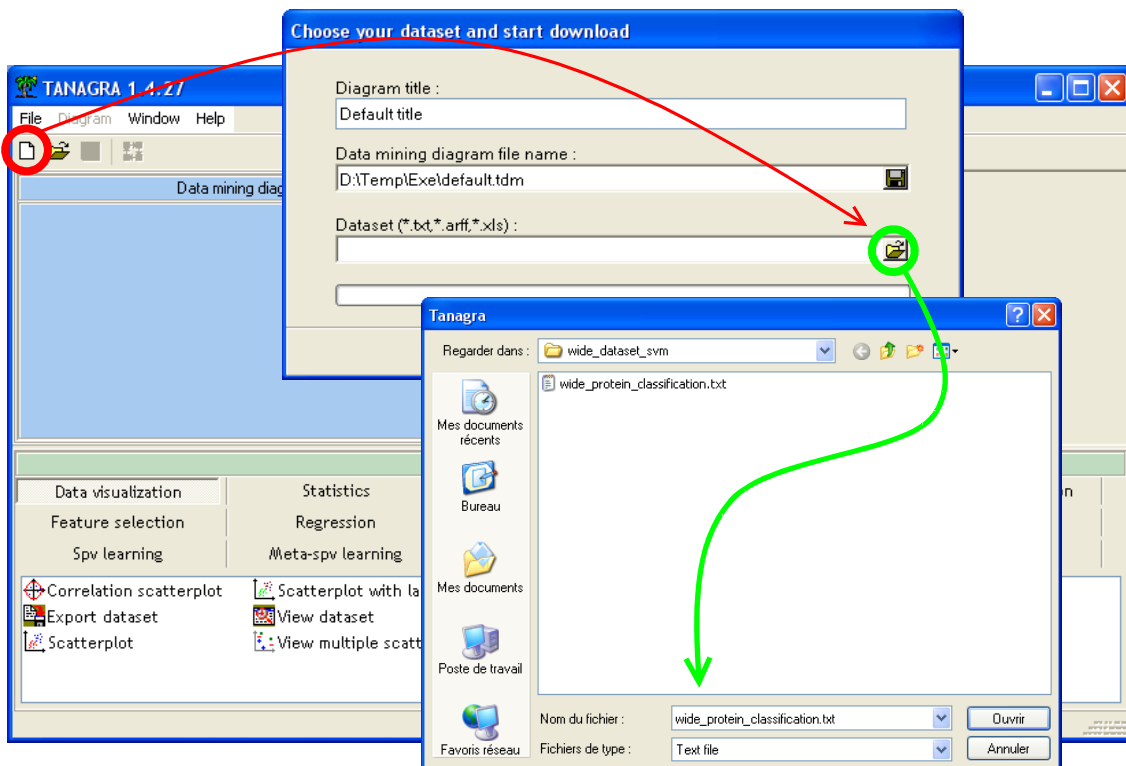


Nous obtenons 2% (3 observations mal classées sur 135). Curieusement, alors que le temps d'apprentissage précédent était très rapide, la validation croisée prend beaucoup de temps (705 secondes) avec une occupation mémoire décuplée (870 Mo).

### 3.3 TANAGRA

TANAGRA est un programme DELPHI compilé pour Windows. Il peut néanmoins fonctionner sur d'autres systèmes via un émulateur.

Première étape, nous devons créer un diagramme de traitements et importer les données. Nous actionnons le menu FILE / NEW. Dans la boîte de dialogue qui apparaît, nous sélectionnons notre fichier de données.



Le temps de chargement est relativement rapide (12 secondes). L'affichage en HTML dans la fenêtre dédiée prend un peu plus de temps en revanche. Nous constatons que nous disposons de 135 observations et 31810 variables (1 variable à prédire + 31809 descripteurs).

**Download information**

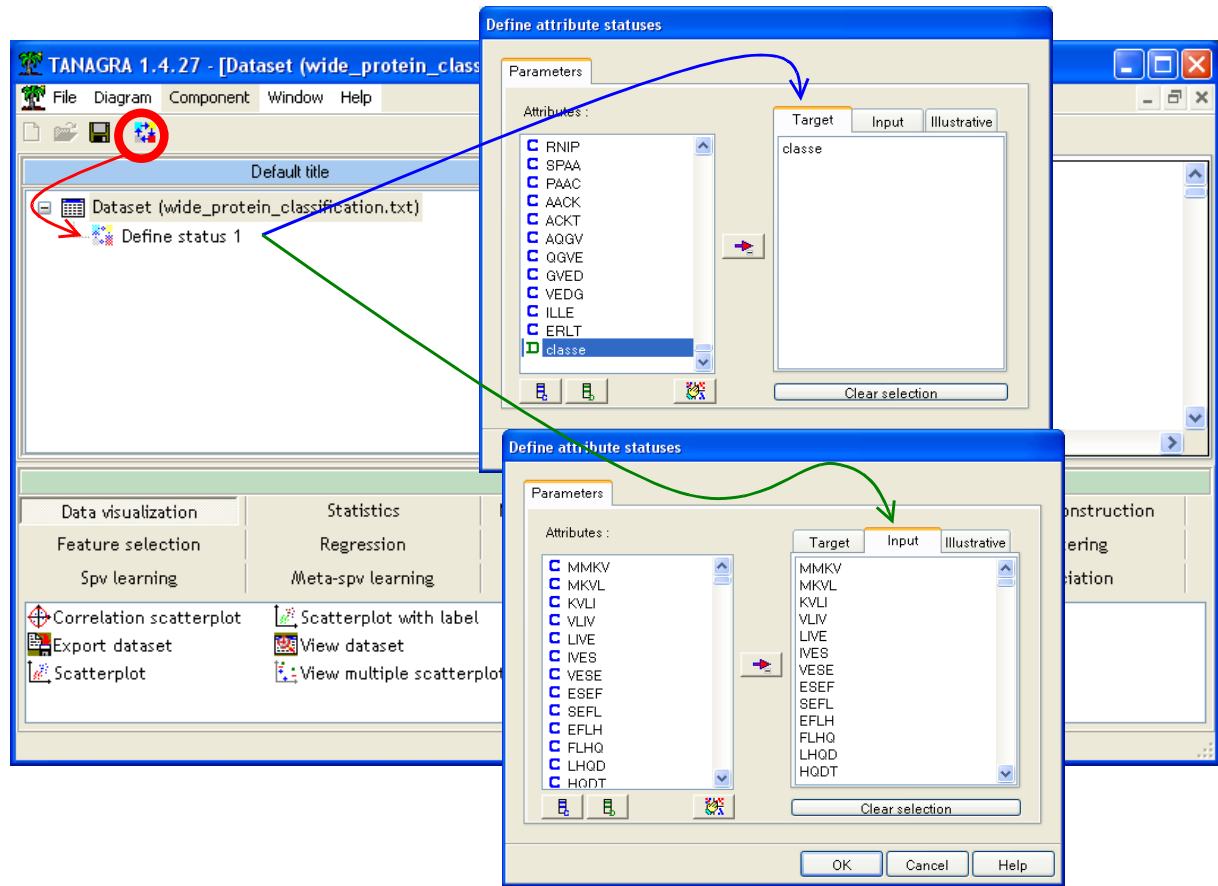
**Datasource processing**

Computation time	12187 ms
Allocated memory	52466 KB

**Dataset description**

**31810 attribute(s)**  
**135 example(s)**

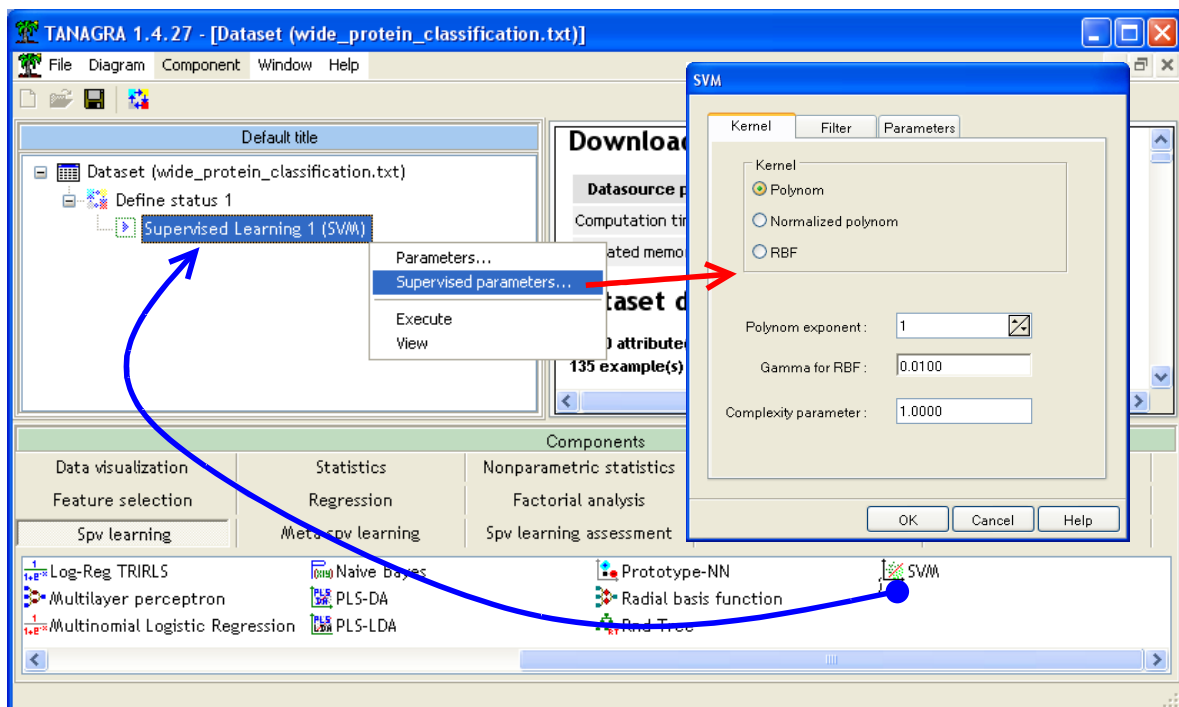
Nous définissons le rôle des variables via le composant DEFINE STATUS accessible dans la barre d'outils : « classe » est placée en TARGET, les autres variables en INPUT.



TANAGRA propose deux composants pour les SVM.

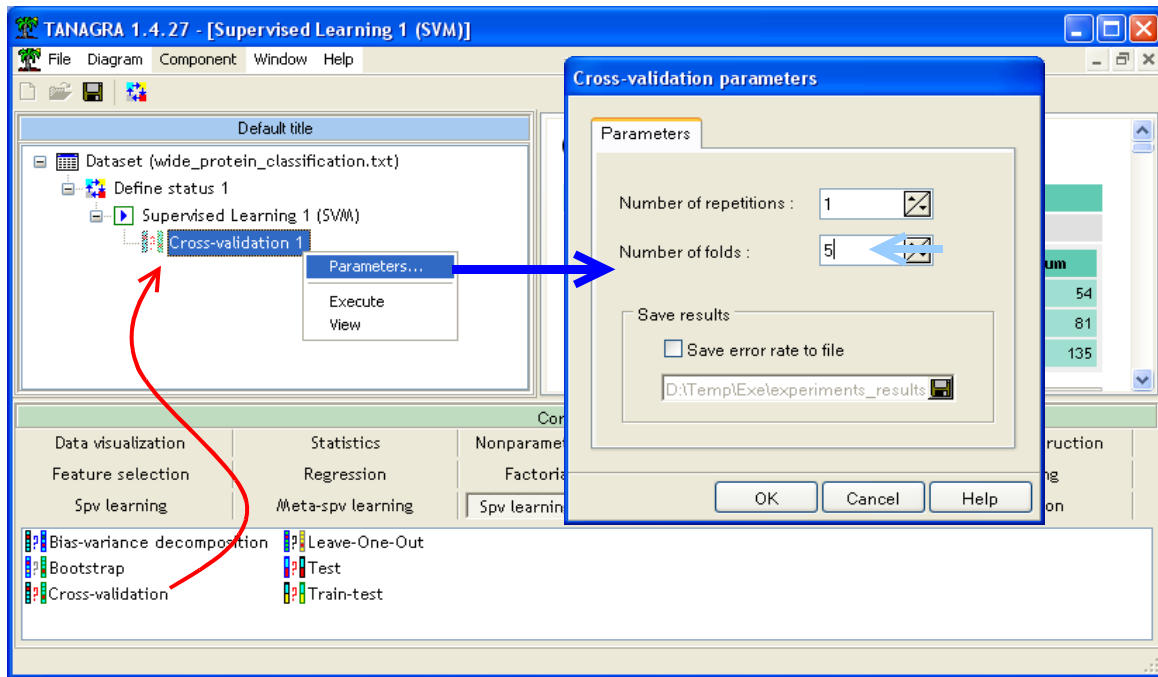
### 3.3.1 TANAGRA – SVM

SVM est une implémentation native de l’algorithme SMO. Nous l’insérons dans le diagramme, nous actionnons le menu SUPERVISED PARAMETERS pour inspecter les paramètres.

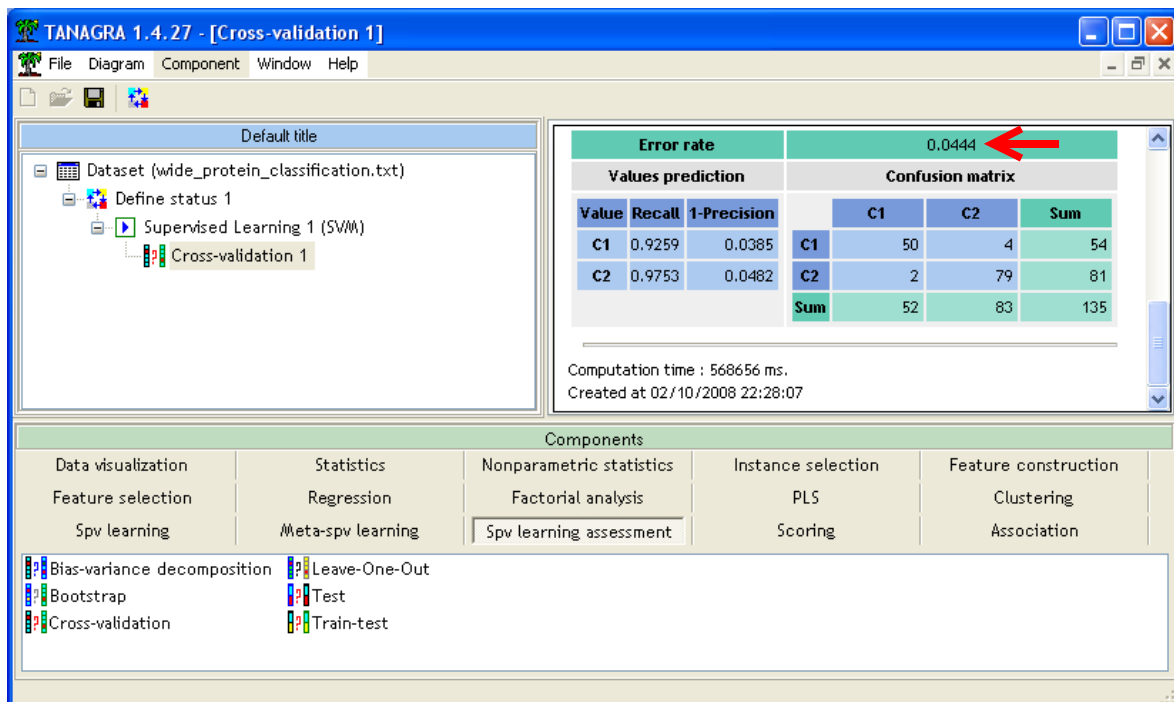


Le paramétrage par défaut nous convient. Nous cliquons sur VIEW pour obtenir les résultats. Ils arrivent au bout de 130 secondes. Mis à part ORANGE, c'est relativement long par rapport aux autres implémentations. Nous verrons pourquoi plus loin. L'occupation mémoire reste très raisonnable avec 393 Mo.

Pour obtenir le taux d'erreur en validation croisée, nous intégrons le composant CROS-VALIDATION dans le diagramme. Nous spécifions 5 FOLDS dans la boîte de paramétrage.

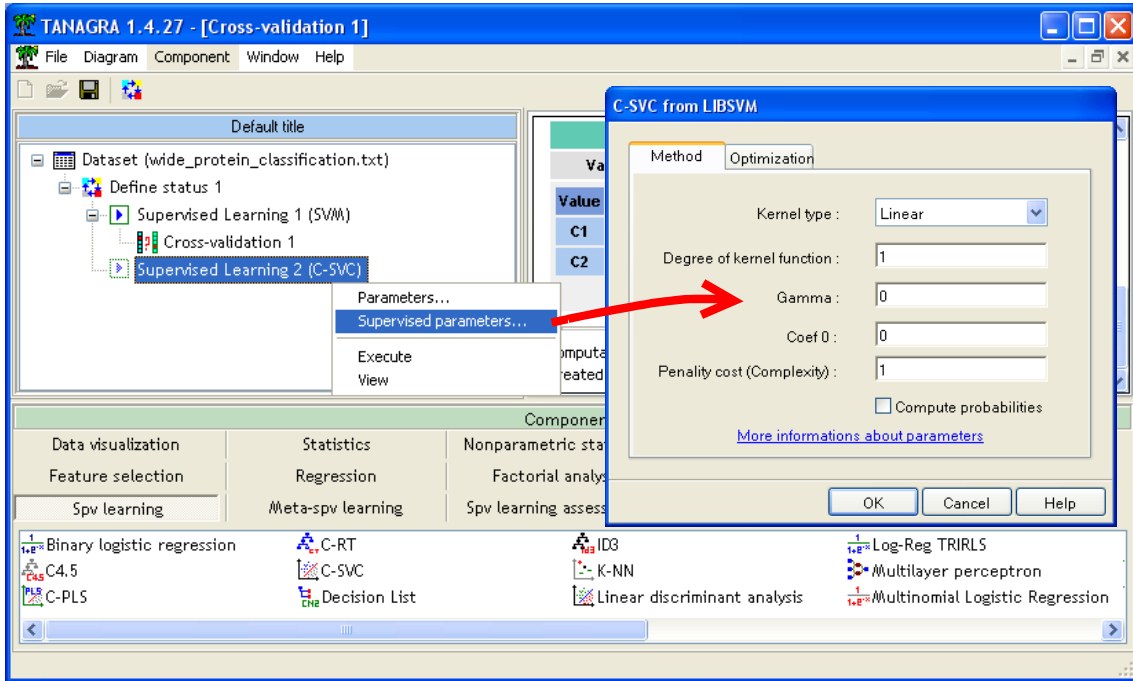


Il ne reste plus qu'à activer le menu VIEW pour accéder aux résultats. Le taux d'erreur est de 4% (6 mal classés). Contrairement aux autres implémentations où la validation croisée accroît parfois exagérément l'occupation mémoire, cette dernière est restée stable pour TANAGRA (avec 393 Mo toujours).



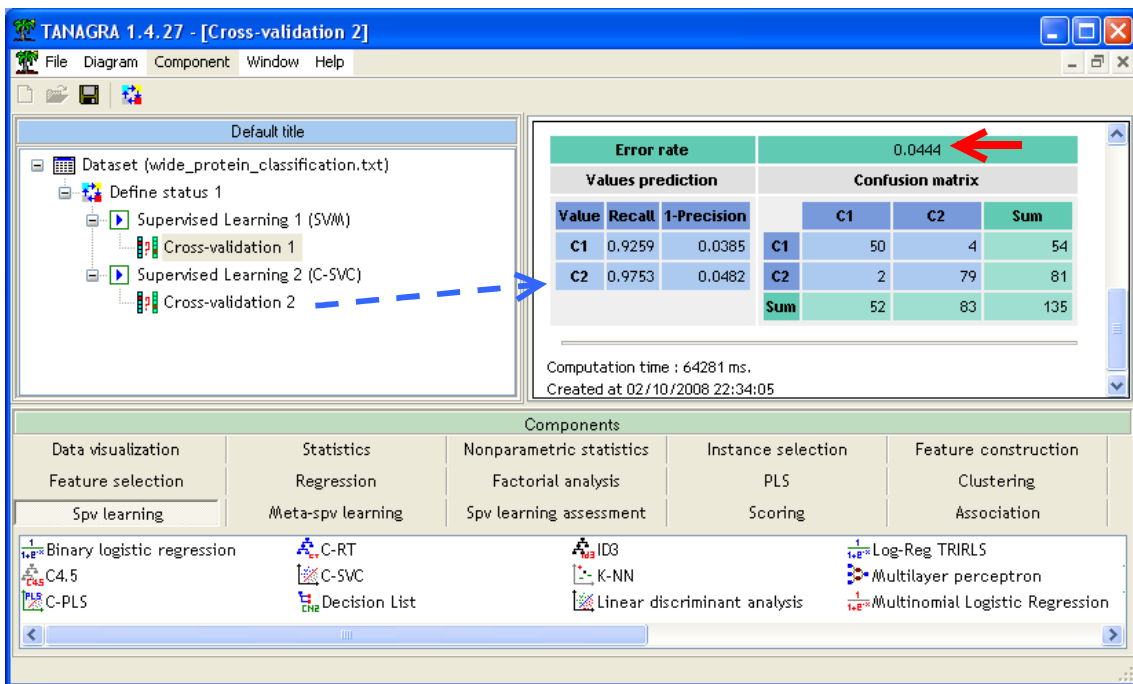
### 3.3.2 TANAGRA – LIBSVM

TANAGRA fait appel à la librairie LIBSVM compilée sous forme de DLL. Nous insérons le composant C-SVC à la suite de DEFINE STATUS 1. Nous actionnons le menu SUPERVISED PARAMETERS pour inspecter les paramètres toujours.



Il ne reste plus qu'à cliquer sur le menu VIEW. Le résultat est quasi-immédiat (11 secondes). Les données ont été dupliquées lors de l'appel de la bibliothèque, pour respecter le format propre à LIBSVM. Cela ne s'avère pas trop pénalisant puisque l'occupation mémoire est passée seulement à 406 Mo durant le traitement, à peu près au même niveau que pour les autres implémentations.

Nous demandons la validation croisée en 5 FOLDS.

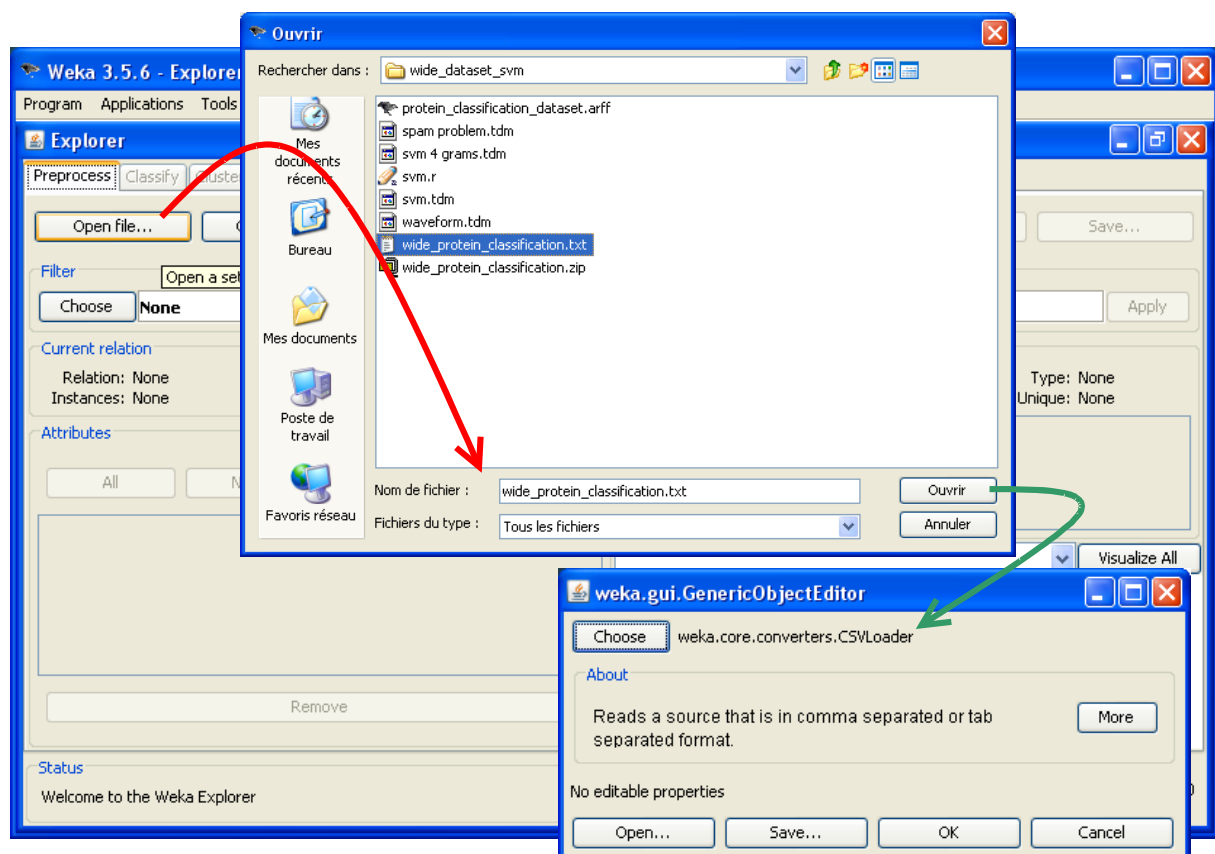


Nous retrouvons le taux d'erreur à 4% (6 individus mal classés), le temps de calcul est de 64 secondes. L'occupation mémoire est restée stable avec 406 Mo.

### 3.4 WEKA

WEKA est programmé en JAVA, la machine virtuelle est démarrée lors du lancement du logiciel. Nous utilisons le module EXPLORER dans ce comparatif.

Première étape, nous importons les données. Le format texte avec séparateur « tabulation » n'apparaît pas dans la liste des formats. Il faut simplement sélectionner le fichier. WEKA dit ne pas reconnaître automatiquement le fichier, il nous demande de confirmer le format, nous choisissons le CSVLoader.

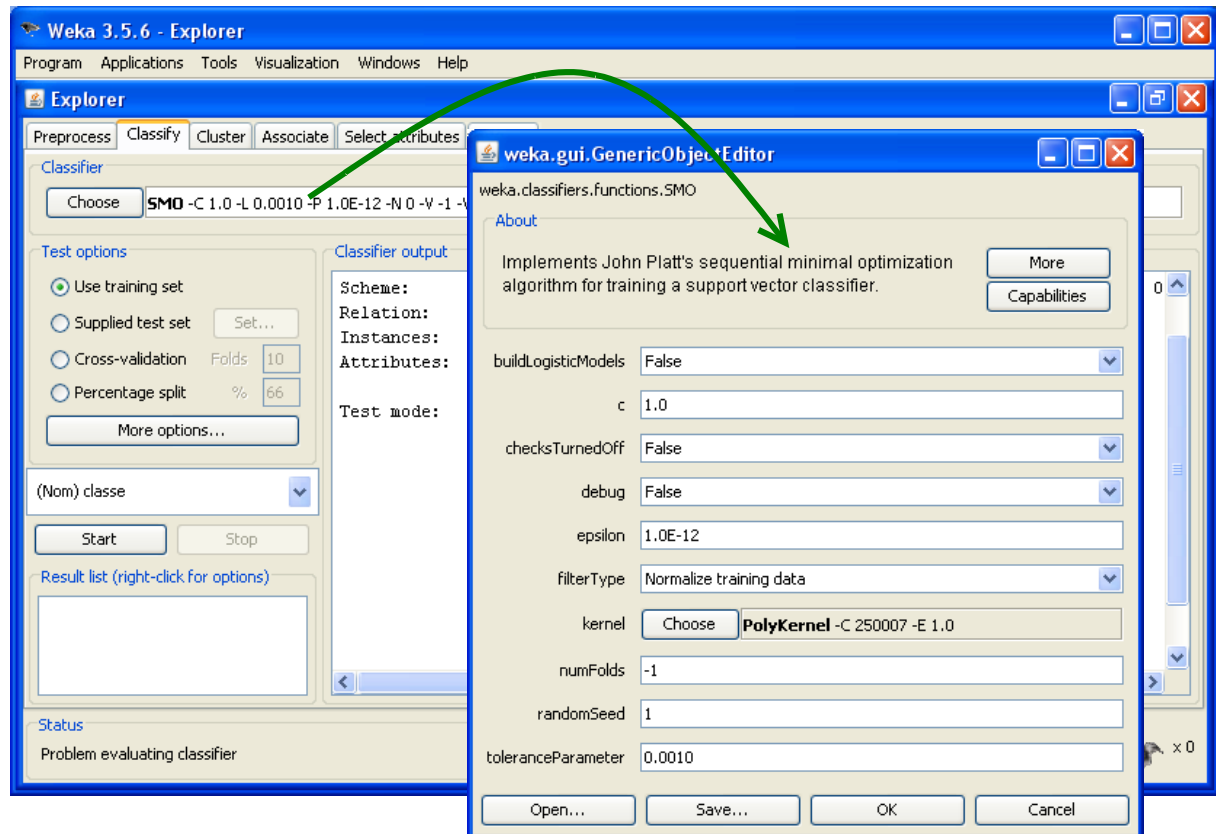


Le temps de chargement est rapide (10 secondes). L'occupation mémoire est passée de 54 Mo à 243 Mo avec les données.

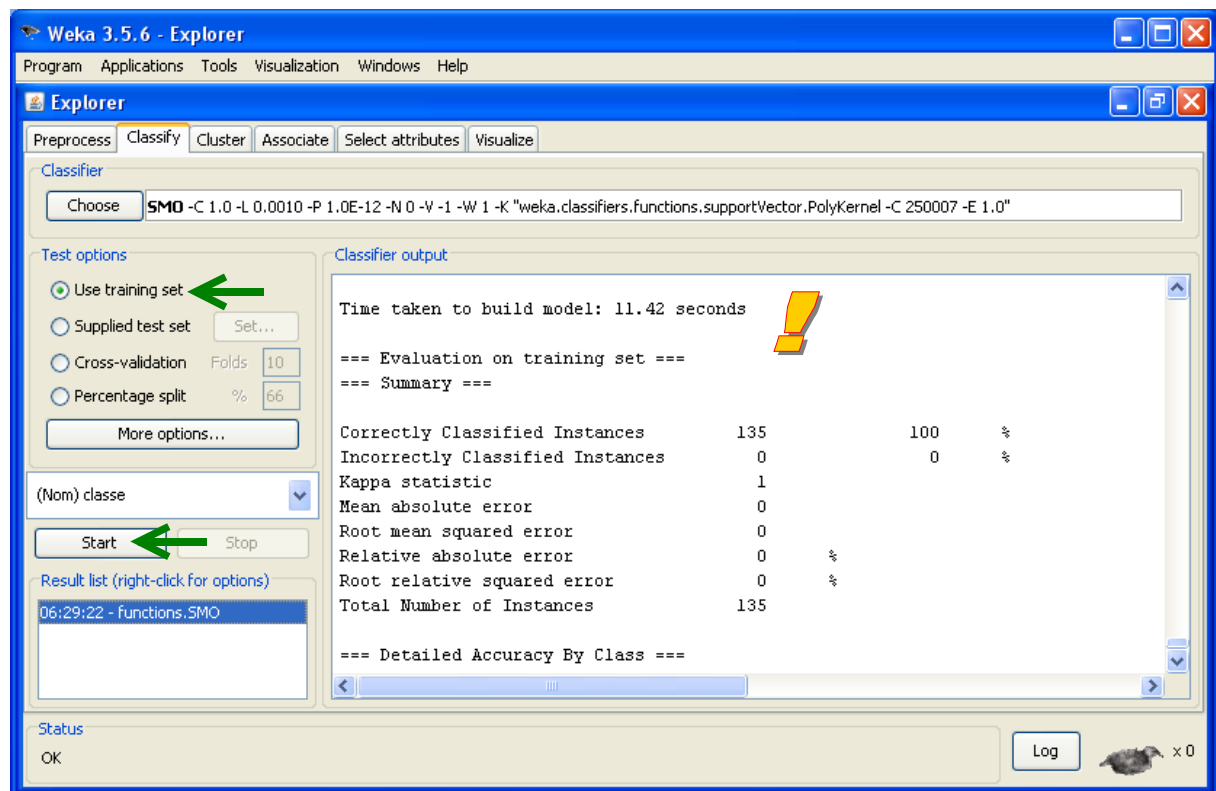
Nous passons dans l'onglet CLASSIFY pour initier une analyse supervisée. En cliquant sur le bouton CHOOSE, la liste des méthodes disponibles apparaît. Elles sont très nombreuses. Nous choisissons la méthode SMO parmi FUNCTIONS.

**Remarque :** On remarquera que LIBSVM est aussi accessible dans WEKA. Il semble cependant qu'elle ne soit pas automatiquement installée. Lorsque nous avons tenté de la lancer sur nos données, un message d'erreur est apparu, indiquant que la bibliothèque n'est pas disponible dans le chemin de recherche usuel (CLASSPATH).

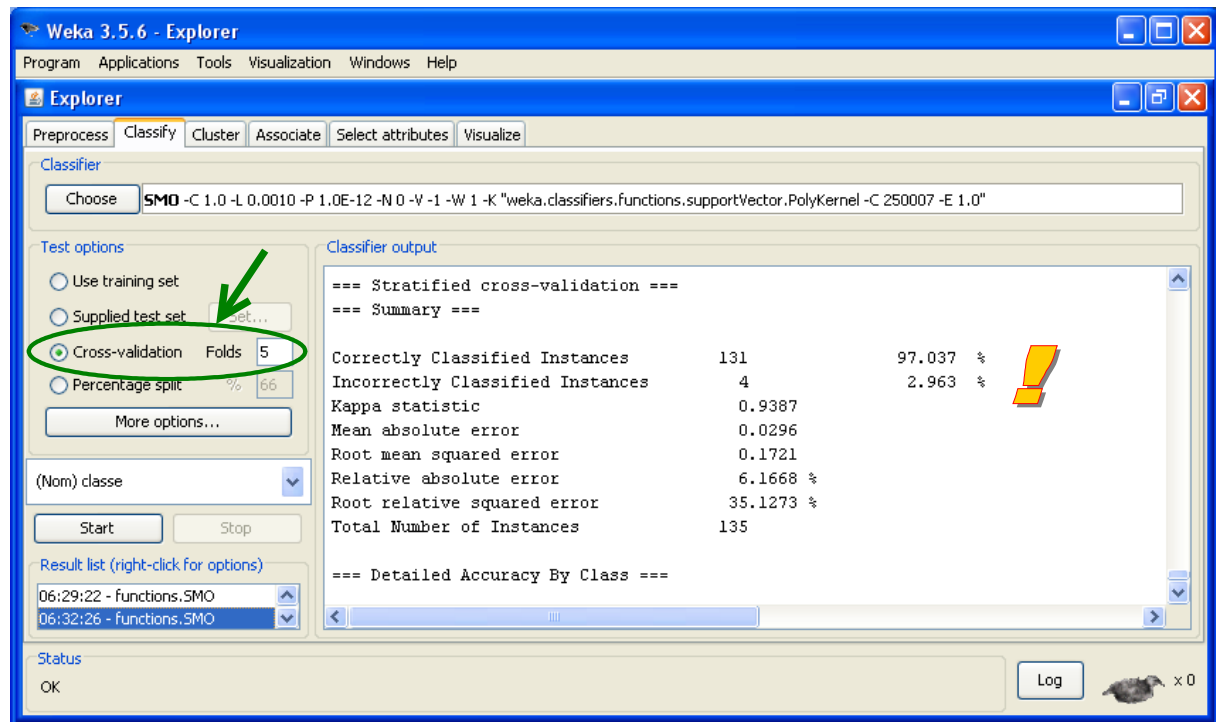
Revenons à SMO. Après l'avoir sélectionnée, nous pouvons la paramétrer en cliquant sur la barre de description de la méthode. Le paramétrage par défaut nous convient, il s'agit d'un SVM linéaire (polynôme de degré 1) avec un paramètre de complexité  $C = 1.0$ .



Dans un premier temps, nous demandons un apprentissage / test sur l'intégralité du fichier (USE TRAINING SET). Nous cliquons sur le bouton START pour lancer les calculs, 11 secondes plus tard les résultats s'affichent.



Pour la validation croisée, il suffit de modifier le paramétrage et de relancer les calculs à l'aide du bouton START. Le taux d'erreur obtenu est de 3% (4 mal classés sur 135). L'occupation mémoire est passée à 595 Mo.



### 3.5 Récapitulatif

Un petit tableau récapitulatif permet de situer les différentes implémentations des SVM :

Logiciel	Temps de traitement (sec.)		Taux d'erreur en validation croisée (%)	Occupation mémoire (Mo)			
	Importation	Calcul		Au lancement	Avec les données	Durant le traitement	Durant la validation croisée
ORANGE	95	690	4% (6/135)	25	118	317	406
RAPIDMINER JMySVM Learner	5	29	11% (15/135)	124	210	338	608
RAPIDMINER C-SVC (LIBSVM)	5	9	2% (3/135)	124	210	442	870
TANAGRA - SVM	12	130	4% (6/135)	7	337	393	393
TANAGRA C-SVC (LIBSVM)	12	11	4% (6/135)	7	337	406	406
WEKA - SMO	11	12	3% (4/135)	54	243	489	595

**Taux d'erreur.** Des logiciels censés implémenter les mêmes techniques produisent des résultats différents en termes de précision de classement. Cela peut laisser perplexe. Ce n'est pourtant pas étonnant. Tout d'abord, comme nous le disions plus haut, les subdivisions ne sont pas les mêmes lors de la validation croisée. Il faudrait utiliser un fichier test identique pour obtenir des résultats directement comparables d'un logiciel à l'autre. Mais surtout, les SVM reposent sur des heuristiques

d'optimisation. Les logiciels ne trouvent simplement pas le même optimum. D'autant plus qu'il y a pléthore de paramètres dont on ne maîtrise pas toujours très bien les effets sur les résultats.

Des implémentations que nous avons étudiées dans ce comparatif, seule JMYSVMLEARNER de RAPIDMINER semble réellement moins performante que les autres. Il faudrait se plonger dans la littérature pour comprendre le pourquoi de cette défaillance dans le contexte précis de nos données.

**Temps de traitement.** J'ai toujours pensé que la librairie LIBSVM était extraordinaire. C'est pour cela d'ailleurs que j'ai consenti à l'intégrer comme librairie externe (DLL). De fait, malgré la duplication et le transfert de données, la méthode reste très rapide dans TANAGRA. Son avantage est encore plus marqué en ce qui concerne son intégration dans RAPIDMINER. En effet, les classes de calcul sont intégrées directement dans le logiciel (voir le code source JAVA dans le sous répertoire SRC de RAPIDMINER).

SMO de WEKA vient tout de suite après. Ici également le niveau de performances est bluffant. D'autant plus que je m'en suis beaucoup inspiré pour implémenter le composant SVM de TANAGRA. Un tel écart de performances, 12 secondes contre 138, mérite que l'on s'y attarde un peu.

Après quelques heures sur un profiler, je me suis rendu compte que la vraie différence se situait dans la méthode **dotProd(.)** de la classe **CachedKernel** c.-à-d. le calcul du produit scalaire entre les couples d'individus. Nos codes semblent très similaires. Mais les performances sont très différentes. En effet, le stockage en lignes des données de WEKA fait merveille pour passer très rapidement d'une variable à l'autre lors du produit scalaire. A cela, WEKA y ajoute un codage *sparse* des données, ignorant les colonnes comportant la valeur 0. Au final, des structures adaptées et une stratégie judicieuse divise par 10 le temps de traitement.

A la lumière de ce résultat, on comprend mieux les écarts de performances entre TANAGRA et WEKA lors de l'induction des arbres de décision (voir <http://tutoriels-data-mining.blogspot.com/2008/09/traitement-de-gros-volumes-comparaison.html>). La structure en colonnes de TANAGRA (classe **TAttribute**) lui permet de parcourir très rapidement les individus dans chaque colonne, opération typique de l'induction par arbres lors de la recherche des variables de segmentation sur un sommet. C'est TANAGRA qui divise par 10 le temps de calcul dans ce cas.

Moralité de tout ceci, avant de s'énerver inutilement sur les langages de programmation ou les compilateurs, on devrait avant tout se pencher attentivement sur les structures de données lorsque l'on souhaite travailler sur les temps de traitement.

Concernant ORANGE, je n'ai pas assez d'éléments pour expliquer cet écart. Je n'ai pas eu le courage d'inspecter en détail le code source. Peut être est-ce du à un mauvais paramétrage de ma part. Il faudrait multiplier les tests sur différentes configurations de données pour mieux comprendre son comportement.

**Occupation mémoire.** On constate de nouveau dans ce comparatif la différence de comportement entre les logiciels fonctionnant à l'aide d'une machine virtuelle et les autres. L'occupation mémoire est légèrement plus importante par rapport aux programmes compilés, malgré un stockage interne des données performant (cf. l'écart entre l'occupation mémoire avec les données seules et après le traitement).

Le comportement de RAPIDMINER-LIBSVM en validation croisée laisse à penser qu'un grand nombre de résultats intermédiaires sont conservés, gonflant inutilement la mémoire utilisée.

## 4 Les autres logiciels

D'autres logiciels ont été testés. Ils n'ont pas pu mener à bien l'analyse pour différentes raisons.

- **KNIME (1.5.31)** : Problème à l'importation des données, dès la sélection du fichier dans la boîte de paramétrage du FILE READER. En scannant le fichier pour l'afficher dans la grille de pré visualisation, le logiciel se fige. Au bout de 10 minutes d'attente, j'avoue avoir perdu patience. Il semble que ce soit la grille qui pose problème. En effet, il n'y aucune raison que les structures internes ne soient pas en mesure d'accueillir les données.
- **R (package e1071)** : Nous avons utilisé la version 2.7.2 de R, avec le package « e1071 » issue de la bibliothèque LIBSVM. Le chargement a été réalisé sans aucun problème en 35 secondes, la mémoire occupée dans Windows est passée de 19 Mo à 156 Mo. La construction du modèle a en revanche échoué. Il semble que R a tenté d'allouer un vecteur de 3.8 Go si l'on se fie au message d'erreur renvoyé. Ce qui n'a pas été possible sur ma machine. On ne voit pas bien le pourquoi de cette erreur. S'il s'agit d'envoyer les données à LIBSVM dans un format adéquat, ce que fait Tanagra par exemple, au pire on multiplie par 2 l'espace mémoire occupé. Le chiffre de 3.8 Go reste mystérieux.
- Un grand nombre de librairies de calcul sont accessibles en ligne. Des sites les recensent (par exemple [http://www.support-vector-machines.org/SVM\\_soft.html](http://www.support-vector-machines.org/SVM_soft.html)). La principale difficulté est de préparer les données dans le format adéquat pour pouvoir les lancer correctement. Certaines librairies sont associées à des logiciels commerciaux tels que MATLAB, leur diffusion est forcément restreinte.

## 5 Conclusion

Pour réellement apprécier ce didacticiel, il faudrait faire le parallèle avec celui consacré aux arbres de décision (<http://tutoriels-data-mining.blogspot.com/2008/09/traitement-de-gros-volumes-comparaison.html>). Ce qui est une vérité pour une configuration (arbre de décision + beaucoup d'observations et relativement peu de variables) n'en est pas une pour l'autre (SVM + peu d'observations et beaucoup de variables). Cela confirme encore une fois qu'il n'y a jamais de solution universelle. Il n'y a que des solutions adaptées à des problèmes précis, dont il importe de circonscrire au mieux les contours si l'on souhaite pouvoir reproduire avantageusement la solution. Dans les deux cas étudiés, des structures internes plus ou moins adaptées peuvent faire varier les temps de traitement d'un facteur de 10. C'est quand même considérable.

Tout un chacun peut reproduire l'expérimentation décrite dans ce didacticiel. Nous pouvons aussi l'orienter à notre guise en utilisant des données présentant des caractéristiques qui sont au centre de nos préoccupations. Finalement, c'est ce qui importe.