# 1 Objectif

**Description des méthodes CVM (Core Vector Machine) et BVM (Ball Vector Machine) de la librairie LIBCVM (http://c2inet.sce.ntu.edu.sg/ivor/cvm.html).**

Les Support Vector Machine (SVM) constituent des méthodes particulièrement efficaces en apprentissage supervisé. De par leur grande stabilité, elles sont bien adaptées aux problèmes comportant un grand nombre de descripteurs relativement à la taille d'échantillon[1]. Elles sont moins à leur avantage lorsque le nombre d'observations « n » devient important. En effet, une implémentation naïve est de complexité $O(n^3)$ en temps de calcul et $O(n^2)$ en espace de stockage. De fait, obtenir la solution optimale n'est pas possible en pratique. Les implémentations recherchent des solutions approchées, et en profitent au passage pour réduire les complexités[2].

J'ai découvert récemment la librairie LIBCVM[3]. L'idée des auteurs est très astucieuse, ils tiennent le raisonnement suivant (traduction très libre) : « Puisqu'on ne peut obtenir que des solutions approchées, on peut s'appuyer sur une formulation équivalente de la recherche des points supports, la recherche de la plus petite boule englobante (*Minimum Enclosing Ball*) en géométrie computationnelle, et utiliser les résultats obtenus avec cette dernière ». A la sortie, tout comme les SVM, la technique produit une série de points supports utilisables pour la prédiction, les performances prédictives sont similaires voire améliorées, avec une capacité accrue d'appréhension des grandes bases (en nombre d'observations).

J'avais trouvé les articles de référence de LIBCVM très intéressants. J'ai d'autant plus été séduit que tous les outils qui permettent de reproduire les expérimentations sont en ligne. Cela nous change des trop nombreux travaux où des auteurs nous promettent monts et merveilles avec une nouvelle méthode qui surpasserait les autres, mais où ni les programmes (ne parlons même pas des codes sources), ni les données réellement utilisées[4], ne sont accessibles. Il est par conséquent impossible de reproduire (de vérifier) les performances qu'on nous exhibe dans les articles dits scientifiques. C'est un peu dommage. Et je doute fort que les relecteurs qui les valident demandent à accéder aux programmes pour examiner dans le détail les résultats. Ils devraient pourtant.

Les méthodes CVM et BVM dédiées à l'apprentissage supervisé sont implémentées dans la librairie **LIBCVM**. Il s'agit une extrapolation de la bibliothèque **LIBSVM** (version 2.85), bien connue des chercheurs (http://www.csie.ntu.edu.tw/~cjlin/libsvm/). **Le code source en C++ étant disponible, j'ai compilé LIBCVM (version 2.2 [beta], 29 août 2011) en DLL et je l'ai intégrée dans TANAGRA 1.4.44**. Dans ce tutoriel, nous décrivons le comportement des méthodes CVM et BVM sur la base « Web data set » accessible sur le site des auteurs. Nous comparons les performances (qualité de prédiction, temps de calcul) avec celles de la méthode C-SVC de la librairie LIBSVM.

[1] Ex. http://tutoriels-data-mining.blogspot.fr/2008/10/svm-comparaison-de-logiciels.html ; le fichier comporte n = 135 observations et p = 31809 descripteurs.

[2] Ce texte repose sur les deux articles accessibles sur le site de LIBCVM - http://c2inet.sce.ntu.edu.sg/ivor/cvm.html.

[3] Merci à **Vicent Pisetta** qui m'a indiqué ces travaux.

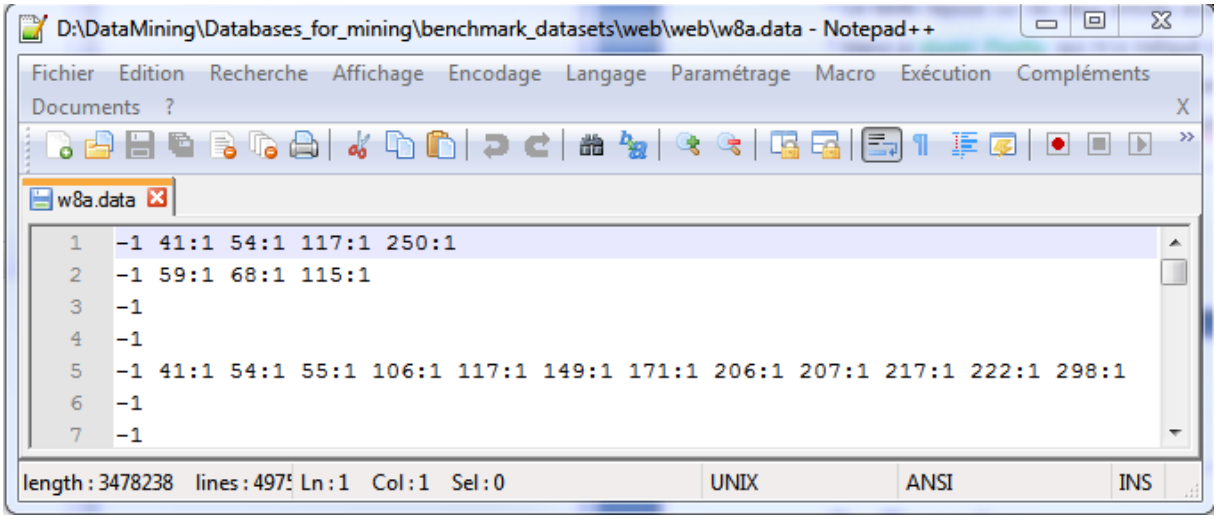
[4] Même si les bases UCI sont publiques (http://archive.ics.uci.edu/ml/), elles nécessitent souvent une préparation avant l'application des algorithmes (ex. données manquantes). Ce pré-traitement est souvent passé sous silence.

# 2 Données « web » au format sparse

La base « web » est initialement constituée de deux fichiers : « w8a.data » est l'échantillon d'apprentissage (49 749 observations) ; « w8a.test » l'échantillon test (14 951 observations). Tous deux sont au format « sparse », reconnu entres autres par les bibliothèques SVMlight et LIBSVM.

Chaque observation est décrite par une ligne. Le premier chiffre correspond à la classe d'appartenance (la valeur de la variable dépendante s'il s'agit d'un problème de régression). Ensuite, nous avons une série de couples de chiffres séparées par « : ». Celui de gauche correspond au numéro de variable, celui de droite à la valeur associée. De fait, la valeur d'une variable non référencée dans la ligne est implicitement nulle (égale à 0).

Prenons un exemple pour expliciter cela. Dans la copie d'écran suivante, nous observons les premières lignes du fichier d'apprentissage « w8a.data ».



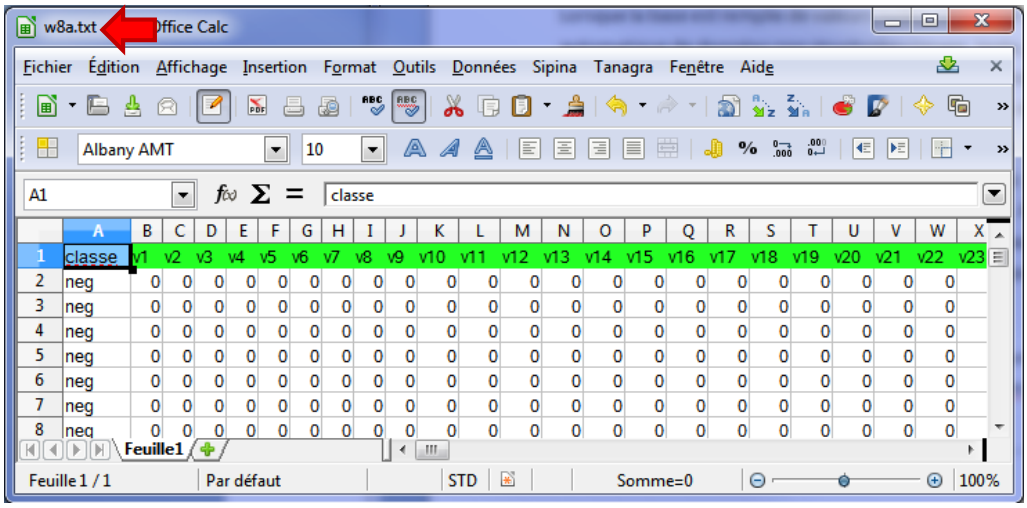

La première observation appartient à la classe « -1 » ; la valeur associée à la première variable est 0 (V1 = 0) puisque le code « 1 : … » est absent de la ligne; de même, V2 = 0 ; … ; V41 = 1 en revanche puisque nous observons « 41 : 1 » ; V42 = 0 ; … ; V54 = 1. Etc.

Lorsque la base est remplie de valeurs nulles, cela arrive souvent lorsque qu'elle est issue du codage automatique de données non structurées (image, texte, etc.), ce mode de représentation permet de réduire considérablement la taille du fichier. C'est un mode de compression très simple. Les données restent néanmoins consultables avec un simple éditeur de texte.

Par commodité, nous avons concaténé les deux fichiers (64 700 lignes) et nous l'avons transformé en format « dense » [5]. La variable cible « classe » est à deux modalités « pos = 1 / neg = -1 », les descripteurs (v1 à V300) sont tous binaires « 0 / 1 ». A la sortie, nous disposons donc d'un fichier avec le format suivant : la première ligne correspond aux noms des variables ; ensuite, chaque ligne décrit une observation toujours, mais avec systématiquement 301 colonnes.

[5] Pour ne pas disperser le lecteur, notre principal propos étant de présenter les méthodes CVM et BVM, nous ne nous attarderons pas sur la manipulation des fichiers « sparses » pour l'instant. Nous y reviendrons plus en détail dans un tutoriel à venir.

Les données du fichier « **w8a.txt** » peuvent être visualisées à l'aide du tableur LibreOffice Calc.

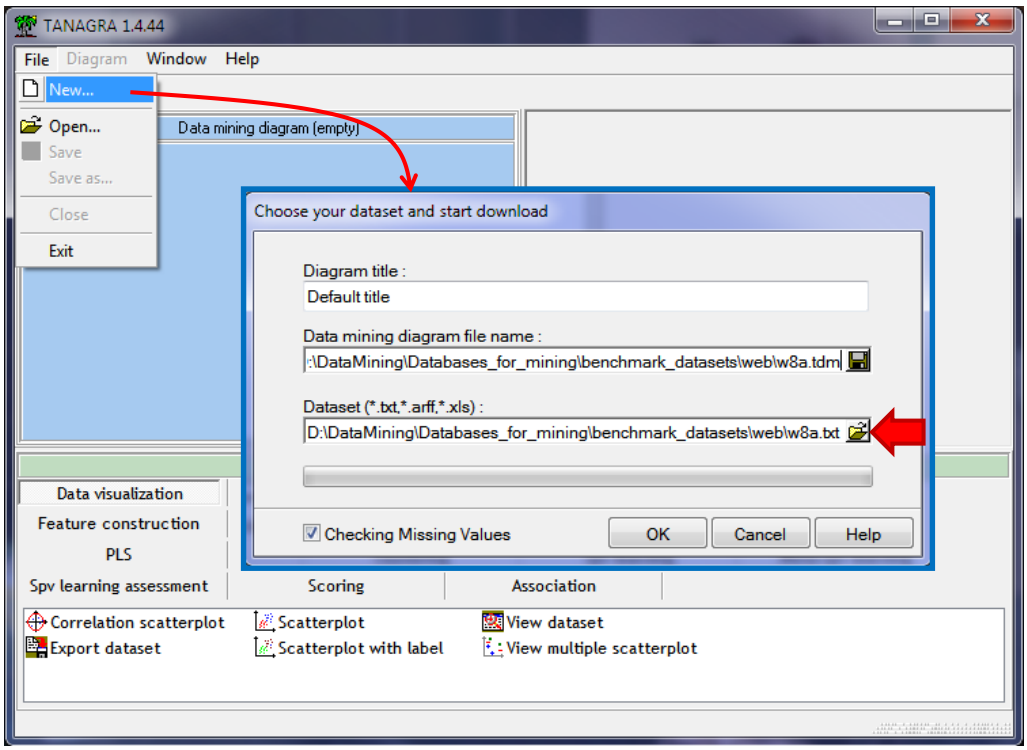



# 3 Core Vector Machine (CVM)

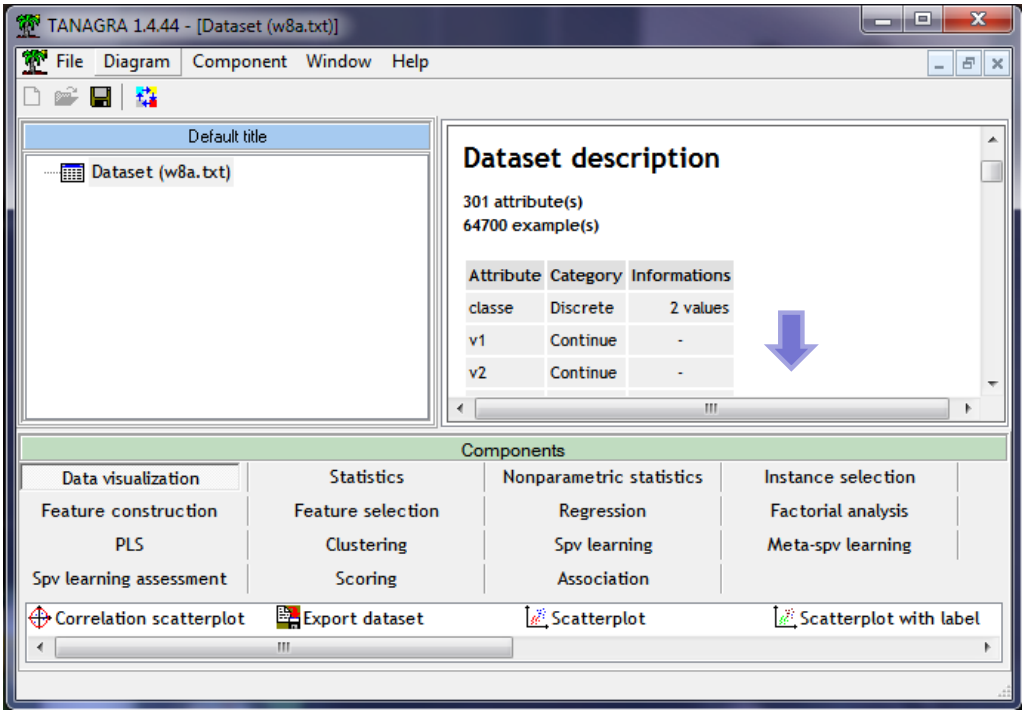
Nous ne décrirons pas la méthode CVM dans cette section, l'article de référence s'en charge très bien : Ivor W. Tsang, James T. Kwok, Pak-Ming Cheung. **Core vector machines: Fast SVM training on very large data sets.** *Journal of Machine Learning Research*, 6:363-392, 2005. La principale idée à retenir est que nous pouvons maintenant traiter de très grandes bases avec des résultats (points supports) et des performances en prédiction similaires à ceux des programmes implémentant les SVM. Nous prendrons en référence la méthode C-SVC du package LIBSVM plus loin (section 5).

## 3.1 Importation des données

Après avoir démarré TANAGRA, nous actionnons le menu FILE / NEW pour créer un nouveau diagramme. Nous sélectionnons le fichier de données « w8a.txt » et nous lançons l'importation.

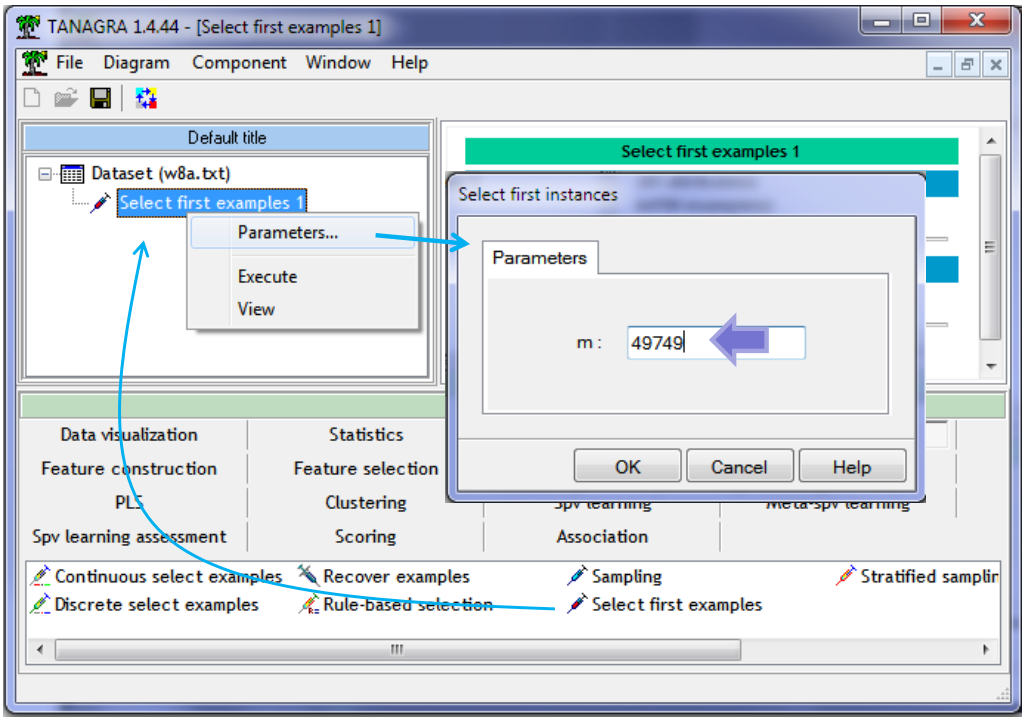


301 variables et 64700 observations sont chargées. La « classe » est catégorielle ; tous les autres attributs sont numériques. Il s'agit en réalité d'indicatrices 0/1. Cette remarque n'est pas anodine, elle déterminera le mode de transformation des données lors de la modélisation.

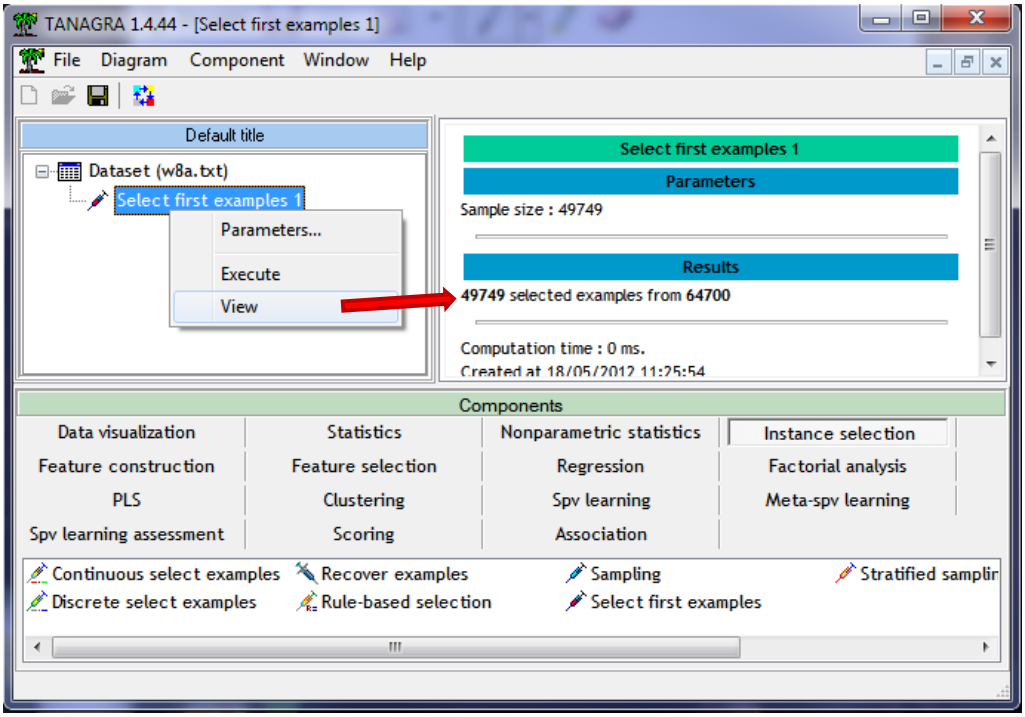



## 3.2 Partition apprentissage-test

La première étape consiste à scinder les données en deux parties pour préciser les individus dévolus à l'apprentissage et au test. Rappelons que nous avons concaténé les deux fichiers. L'échantillon d'apprentissage correspond donc aux 49 749 premières observations.

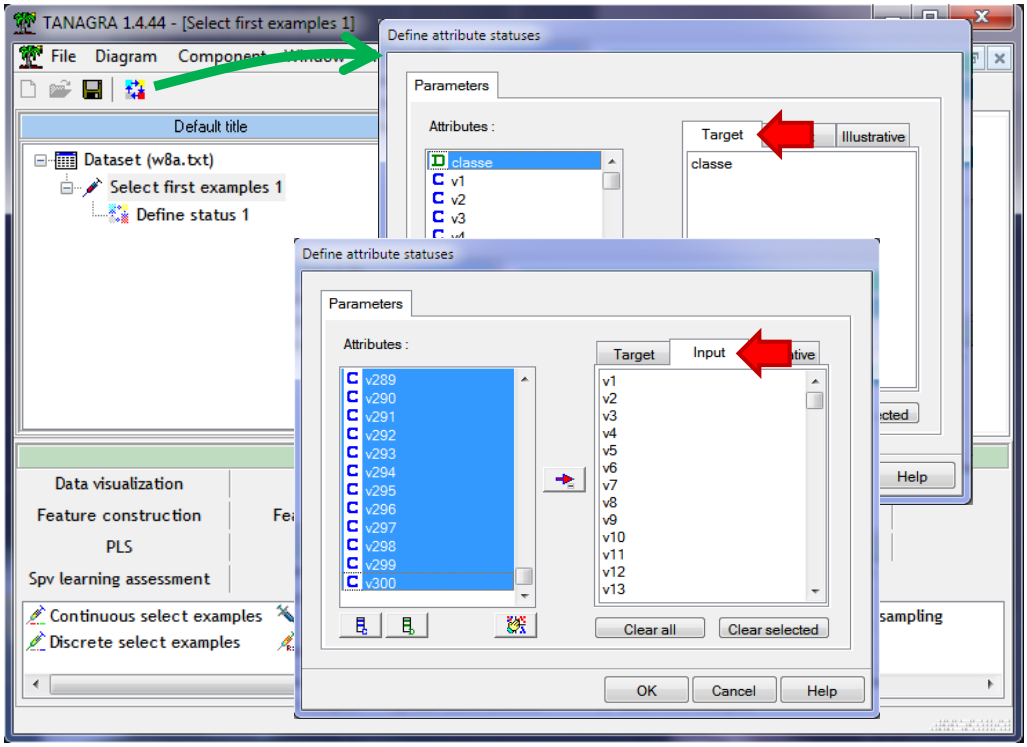


Nous ajoutons le composant SELECT FIRST EXAMPLES (onglet INSTANCE SELECTION) dans le diagramme. Nous le paramétrons (menu PARAMETERS). Nous validons notre choix et nous cliquons sur VIEW. Tanagra indique le nombre d'observations sélectionnées pour l'apprentissage.

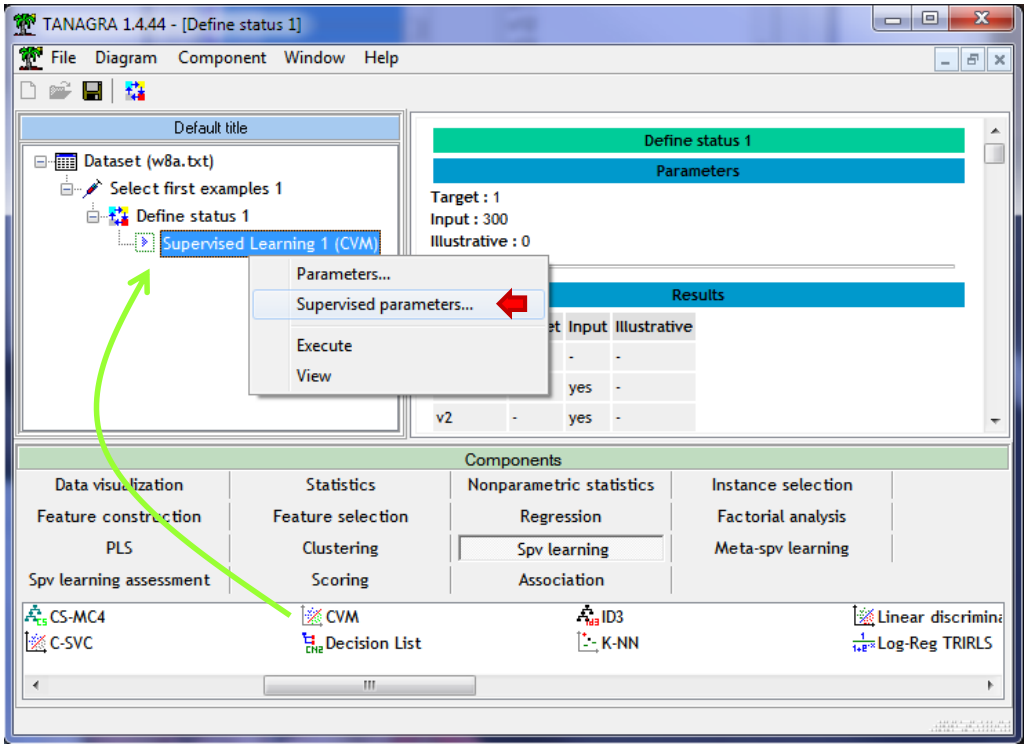



## 3.3 Paramétrage de la méthode

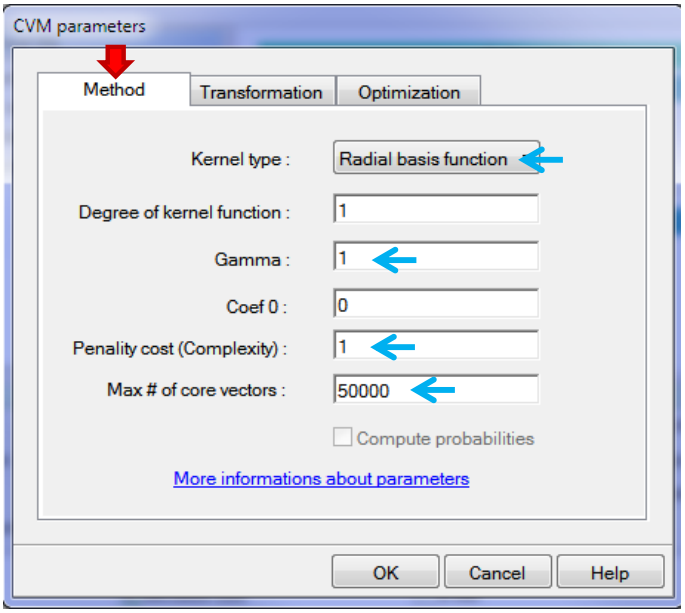
Nous introduisons le composant DEFINE STATUS pour désigner la variable cible « classe » et les variables prédictives « V1… V300 ».



Nous ajoutons alors le composant CVM (onglet SPV LEARNING). Nous le paramétrons en cliquant sur le menu contextuel SUPERVISED PARAMETERS.




Dans l'onglet **METHOD**, nous spécifions les options utilisées dans notre processus[6] :




- Type de noyau [Kernel Type] : RBF (Radial Basis Function, option : –t 2)
- Gamma : 1 (-g 1)
- Cost : 1 (–c 1)
- Max number of core vectors : 50000 (-f 50000)

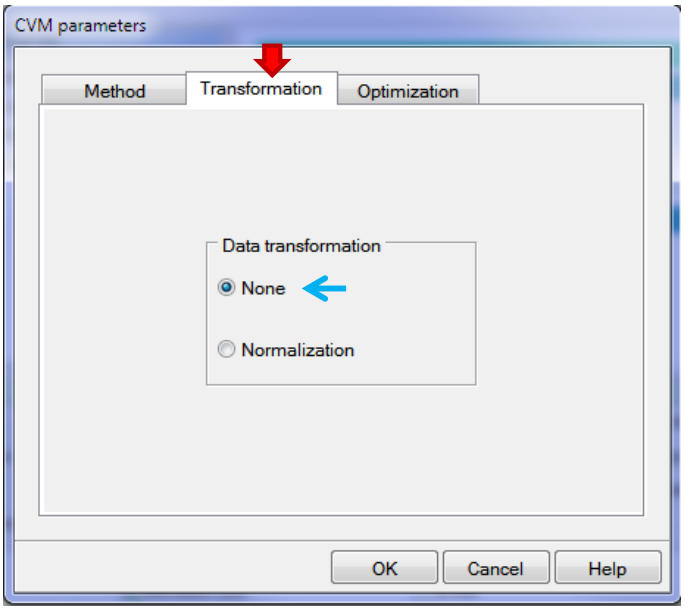
[6] Voir la description des options en ligne http://c2inet.sce.ntu.edu.sg/ivor/cvm.html (Section **How to Use**).

Dans l'onglet **TRANSFORMATION**, nous pouvons spécifier le mode de préparation des variables. **Cette option est propre à Tanagra**. Par défaut, NORMALIZATION est activée. Tanagra introduit la normalisation suivante pour chaque variable X :

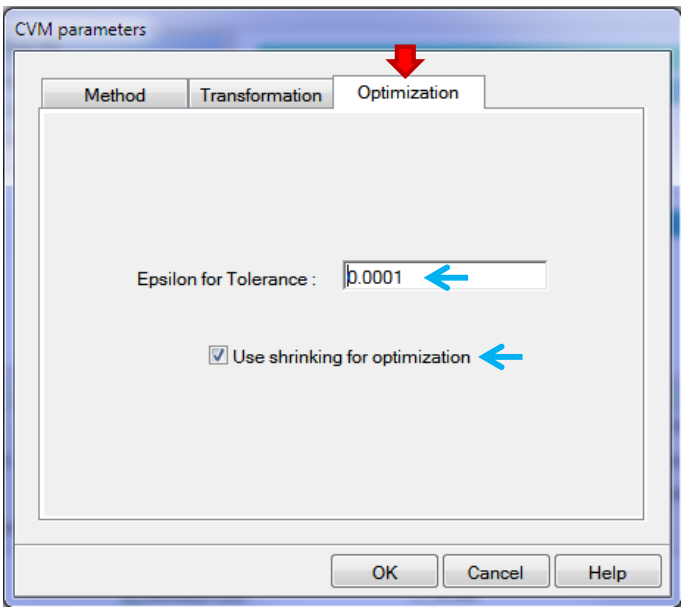
$$z = 2 \times \frac{x - x_{min}}{x_{max} - x_{min}} - 1$$

Ainsi, Z varie entre [-1 ; +1]. Cette transformation est importante lorsque les variables sont exprimées dans des unités différentes. Les domaines de variation n'étant pas les mêmes, celles qui ont une plus forte amplitude risquent de « tirer » indument sur les résultats.

Dans notre cas, toutes les variables sont des indicatrices définies sur {0, 1}. Cette normalisation, qui est consommatrice de temps calcul tant en apprentissage qu'en déploiement, n'a pas lieu d'être. C'est pour cette raison que nous sélectionnons l'option **NONE**.




Enfin, dans l'onglet **OPTIMIZATION**, nous avons les options qui pèsent sur la finesse de l'optimisation (et par conséquent sur le temps de calcul !) :

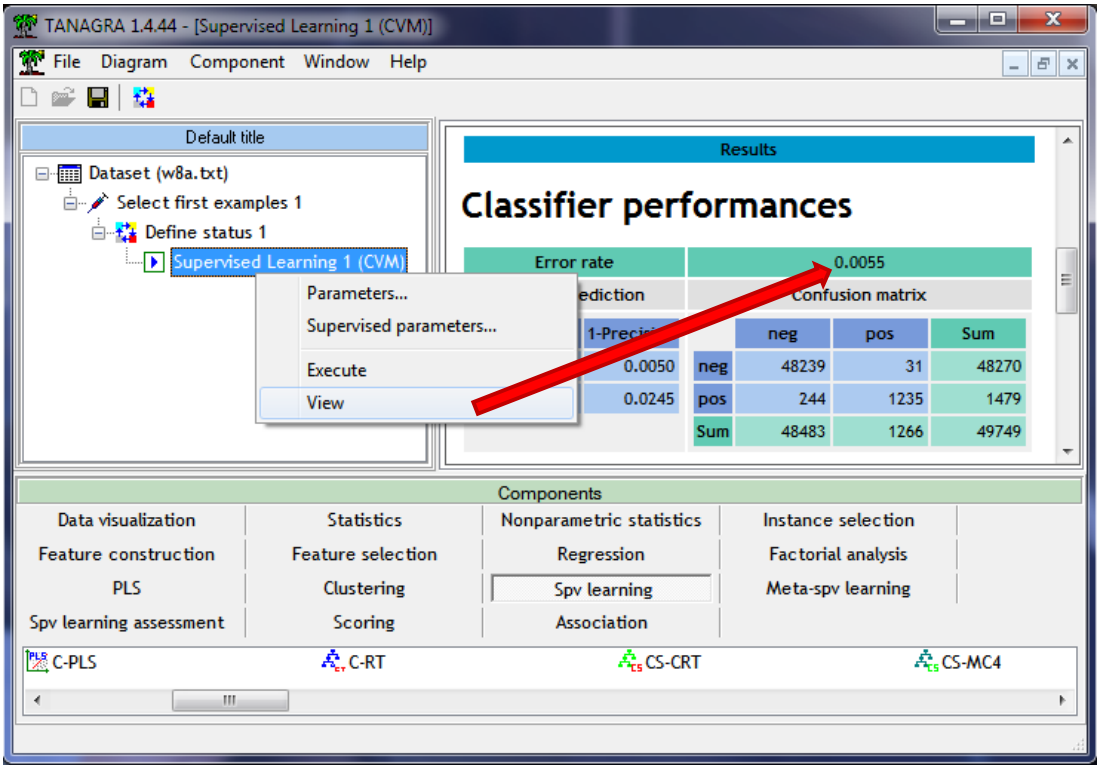


- Epsilon : 0.0001 (-e 0.0001)
- Use shrinking for optimization : cochée (-h 1).

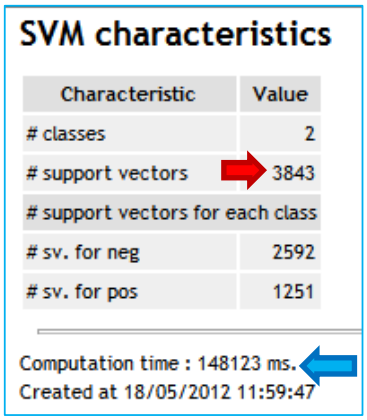
Nous validons. Puis nous lançons les calculs en actionnant le menu contextuel VIEW.

**Note** : *Précisons quand même que nous sommes en train de lancer un algorithme apparenté aux SVM sur 49 749 observations. Si nous devions réellement calculer et stocker en double précision le produit scalaire entre tous les individus pris deux à deux, en considérant que la matrice est symétrique, nous occuperons [(49749 x 49749)/2] x 8 ≈ 9.22 Go en mémoire centrale. Manifestement, une implémentation naïve n'est absolument pas tenable sur nos données. Et sur le site de LIBCVM, on constate que les auteurs ont appliqué leur méthode sur des fichiers allant jusqu'à 4 898 431 observations (fichier « KDD-99, intrusion detection ») !*

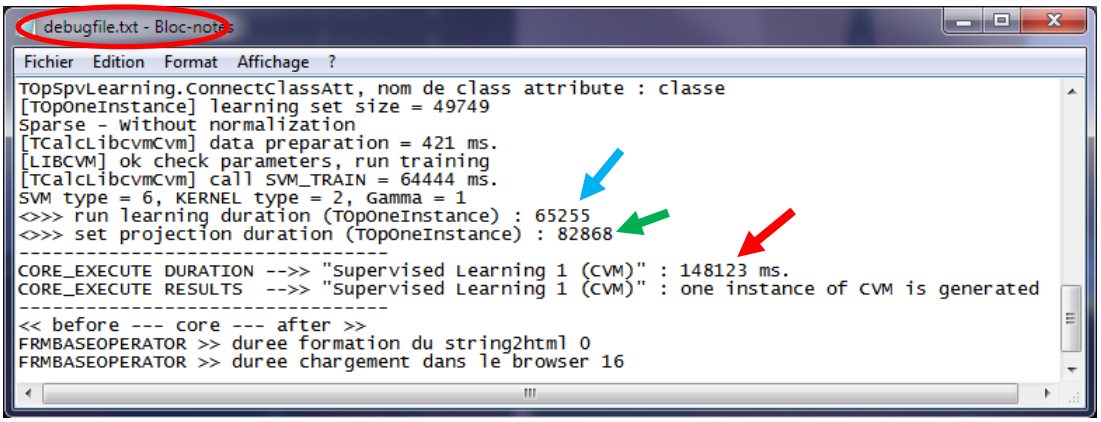
## 3.4 Apprentissage et évaluation



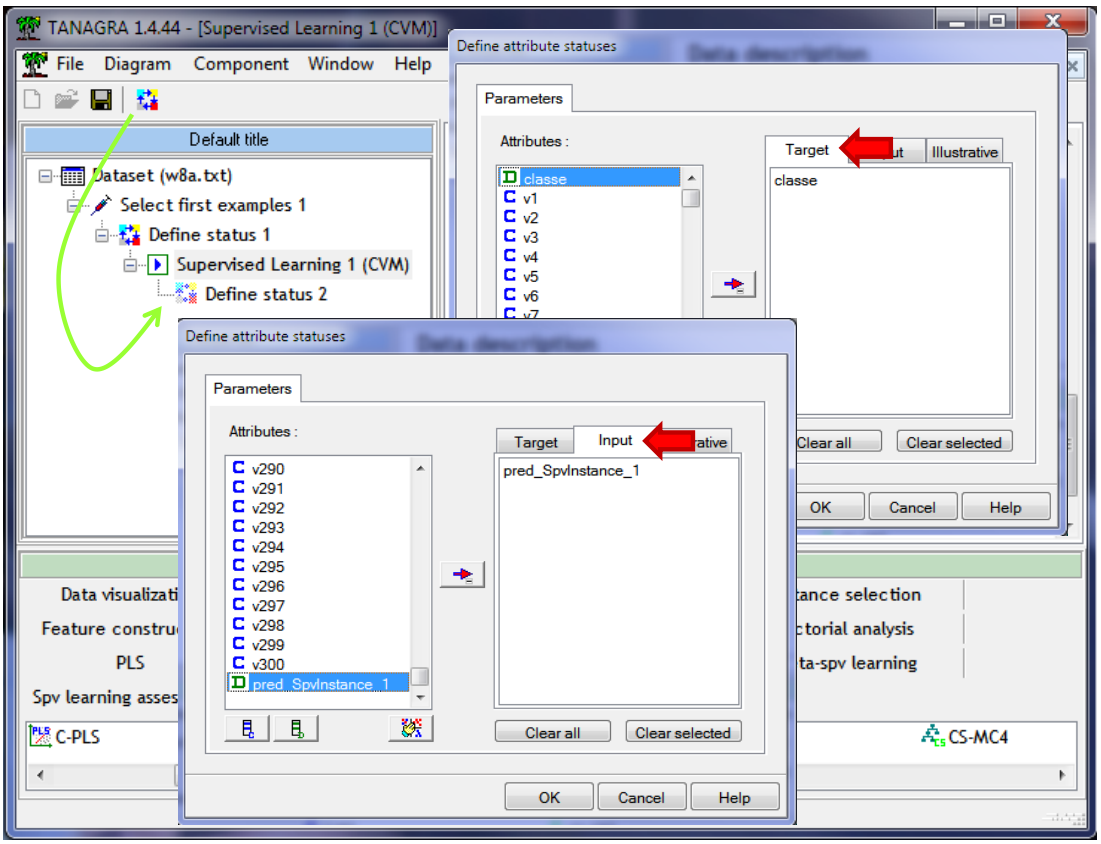

Le taux d'erreur en apprentissage est de 0.55% (244 + 31 = 275 individus mal classés sur 49 749). Le modèle s'appuie sur 3843 points supports.



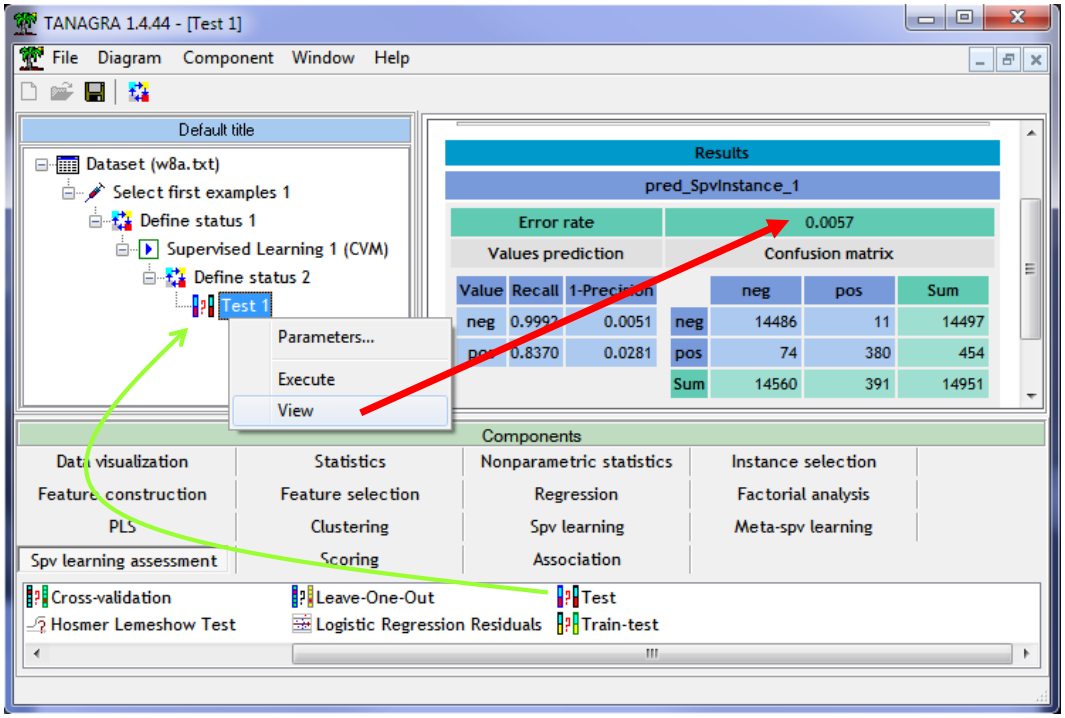
La durée du traitement est de 148 secondes. Il se décompose en deux parties, le détail est visible dans le fichier log (**debugfile.txt**) : 65 secondes sont consacrées au processus de modélisation proprement dit, c'est peu pour un (apparenté) SVM sur un fichier de cette taille ; 83 à l'application du modèle sur les 64700 individus (apprentissage + test) pour disposer de la colonne prédiction. Cette dernière est nécessaire pour obtenir le taux d'erreur en resubstitution ci-dessus, mais aussi le taux d'erreur en test que nous calculerons ci-après. Le temps de calcul pour cette seconde partie n'est pas négligeable. Il sera d'autant plus conséquent que le nombre de points supports est élevé.




Pour calculer le taux d'erreur en test, nous insérons de nouveau le composant DEFINE STATUS. Nous plaçons CLASSE en TARGET, et la colonne prédiction PRED_SPVINSTANCE_1 en INPUT.



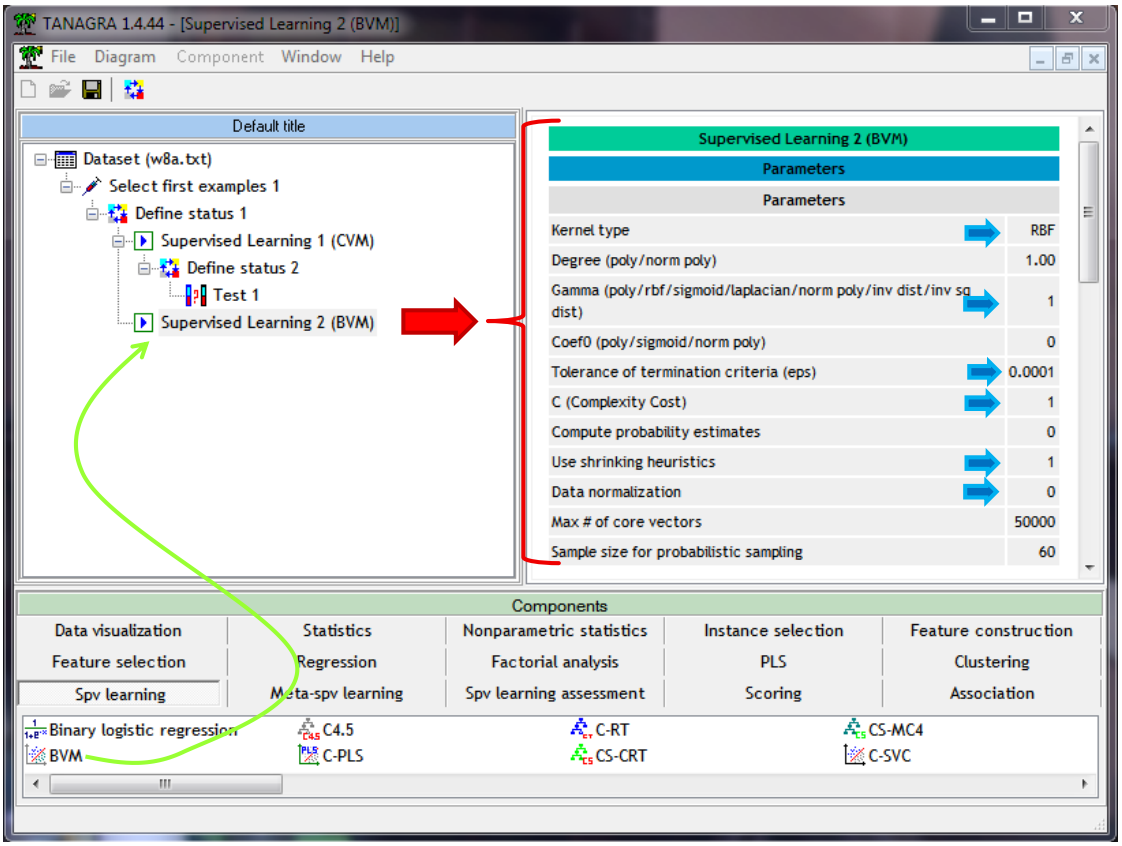

Puis, nous ajoutons le composant TEST (onglet SPV LEARNING ASSESSMENT). Par défaut, il réalise les calculs sur les individus non sélectionnés dans la branche du diagramme, c.-à-d. l'échantillon test.




Le taux d'erreur en test est de 0.57%, avec 85 (74 +11) individus mal classés sur 14951.

# 4 Ball Vector Machine

BVM est une variante de CVM. L'article de référence est : I. W. Tsang, A. Kocsor, J. T. Kwok. Simpler core vector machines with enclosing balls. *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML)*, Corvallis, Oregon, USA, June 2007.



Pour résumer, l'exigence de minimisation est relâchée lors de la recherche de la boule englobante. Il devrait en résulter un temps de calcul moins important lors de l'apprentissage. Nous introduisons le composant BVM (onglet SPV LEARNING) dans le diagramme. Nous le paramétrons exactement de la même manière que CVM (nous laissons les valeurs par défaut pour les paramètres additionnels).

La durée de calcul est de 84 secondes. La phase de construction proprement dite est de 27 secondes, le reclassement des individus de 57 secondes. Nous obtenons 2710 points supports, avec un taux d'erreur en resubstitution de 0.56%.

**SVM characteristics**

| Characteristic | Value |
|---|---|
| # classes | 2 |
| # support vectors | 2710 |
| # support vectors for each class | |
| # sv. for neg | 1487 |
| # sv. for pos | 1223 |

Computation time : 84662 ms.
Created at 18/05/2012 15:37:18

**Classifier performances**

| Error rate | 0.0056 |
|---|---|

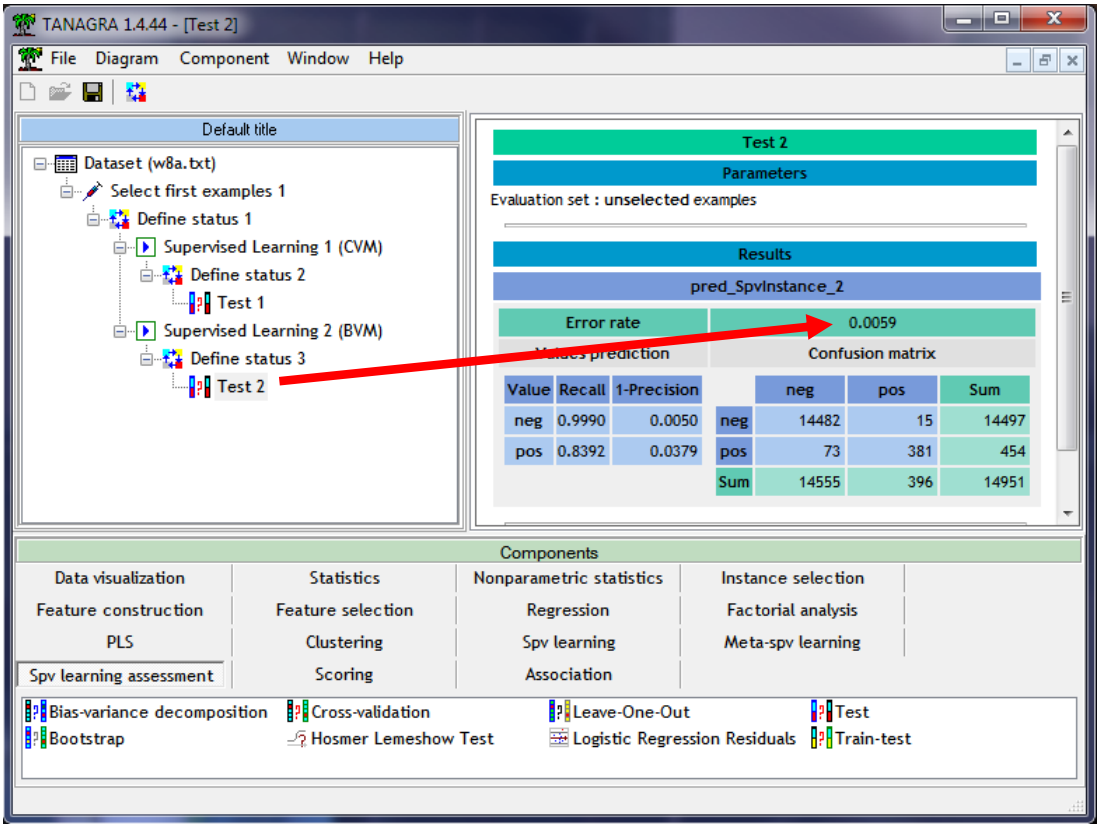
Values prediction

| Value | Recall | 1-Precision |
|---|---|---|
| neg | 0.9991 | 0.0049 |
| pos | 0.8404 | 0.0327 |

Confusion matrix

| | neg | pos | Sum |
|---|---|---|---|
| neg | 48228 | 42 | 48270 |
| pos | 236 | 1243 | 1479 |
| Sum | 48464 | 1285 | 49749 |

En test, le taux d'erreur est de 0.59% (88 individus mal classés sur 14951).




# 5 Récapitulatif – Comparaison avec C-SVC

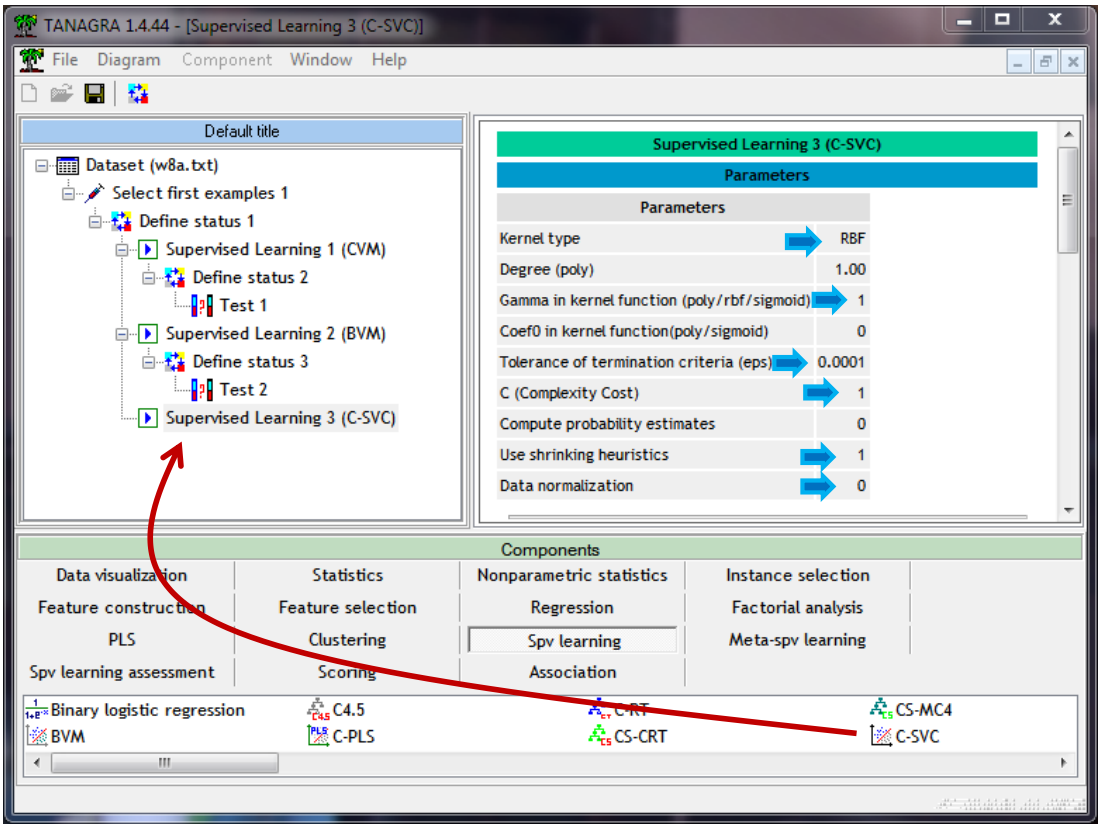
## 5.1 C-SVC de la librairie LIBSVM

CVM et BVM sont censés fournir des résultats comparables à ceux des SVM classiques, tout en étant capable de traiter plus rapidement les grandes bases de données. Pour le vérifier, nous appliquons la

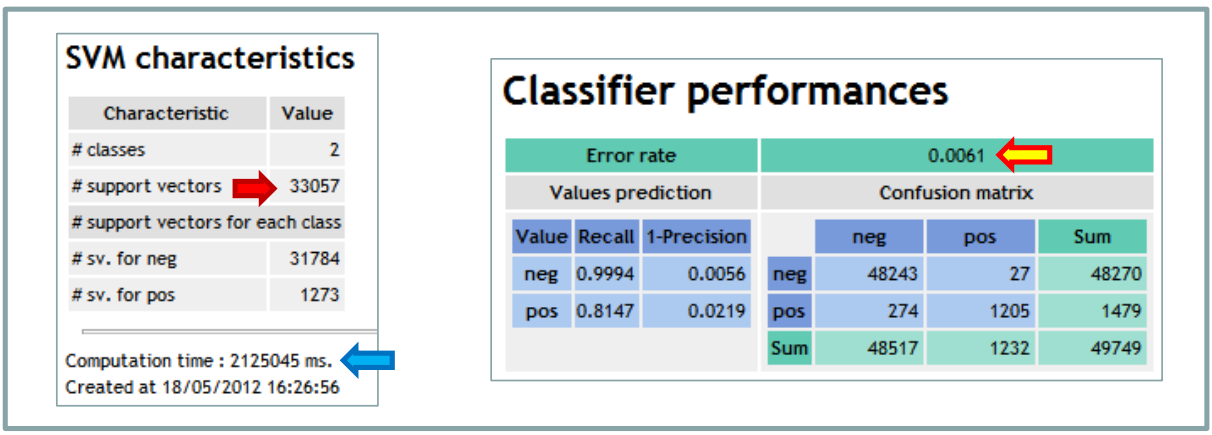
méthode C-SVC de la librairie LIBSVM, bien connue, et de ce fait très utilisée, dans la communauté de l'apprentissage supervisé.

**Note :** *Pour ma part, après avoir tenté de programmer mon propre SVM (composant SVM, onglet SPV LEARNING) dans Tanagra, j'avais constaté que j'étais réellement loin du compte (en rapidité de traitement) par rapport à C-SVC notamment. Cela m'avait persuadé d'inclure le package sous forme de DLL, chose que je m'étais toujours refusé à faire auparavant. L'intégration de LIBCVM aujourd'hui procède de la même démarche.*

Nous insérons le composant C-SVC dans le diagramme, toujours en veillant à reproduire le paramétrage de CVM et BVM ci-dessus.




Il ne nous reste plus qu'à lancer les calculs.



**SVM characteristics**

| Characteristic | Value |
|---|---|
| # classes | 2 |
| # support vectors | 33057 |
| # support vectors for each class | |
| # sv. for neg | 31784 |
| # sv. for pos | 1273 |

Computation time : 2125045 ms.
Created at 18/05/2012 16:26:56

**Classifier performances**

| Error rate | 0.0061 |
|---|---|

Values prediction

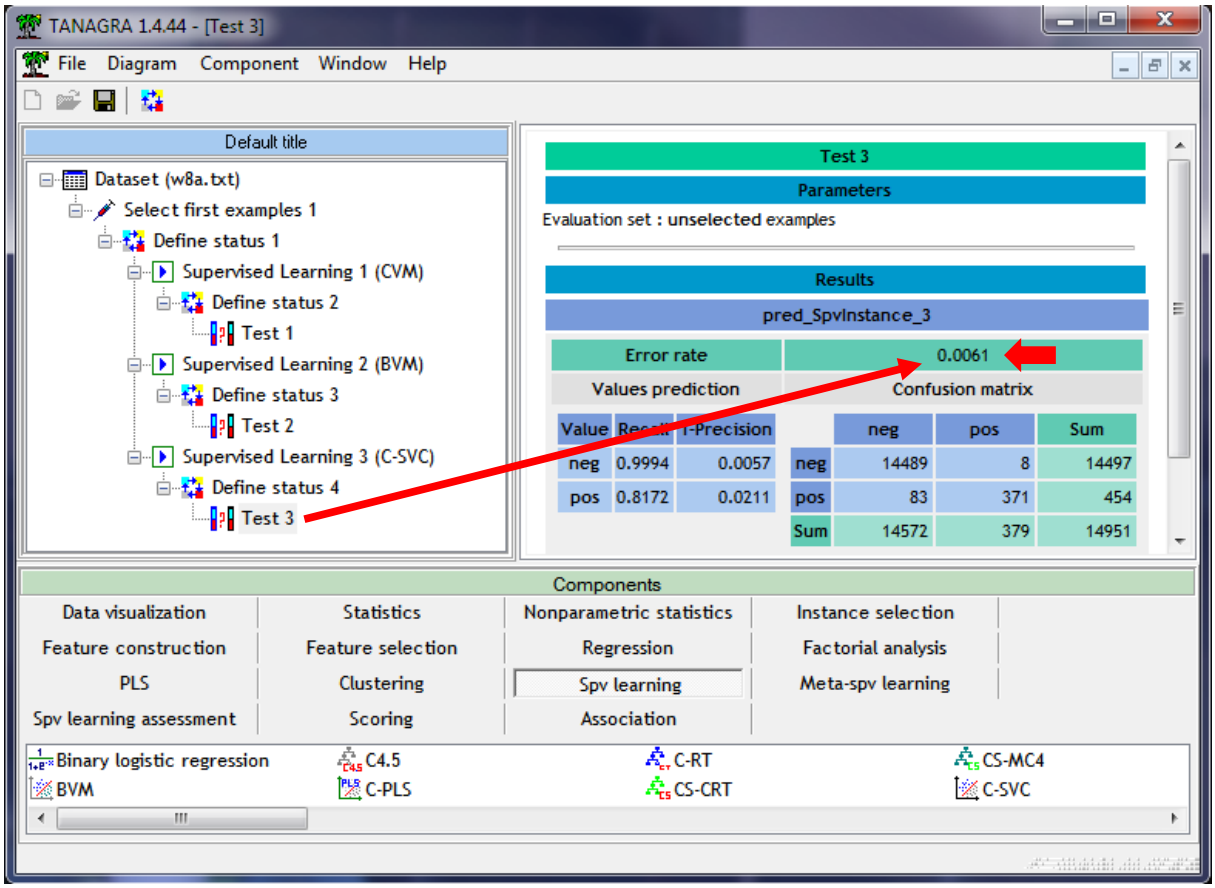
| Value | Recall | 1-Precision |
|---|---|---|
| neg | 0.9994 | 0.0056 |
| pos | 0.8147 | 0.0219 |

Confusion matrix

| | neg | pos | Sum |
|---|---|---|---|
| neg | 48243 | 27 | 48270 |
| pos | 274 | 1205 | 1479 |
| Sum | 48517 | 1232 | 49749 |

Nous obtenons 33 057 points supports au bout de 2125 secondes de calculs (≈ 35 minutes !). Le taux d'erreur en resubstitution, calculée sur l'échantillon d'apprentissage est de 0.61 %. En allant dans le détail, nous constatons que la durée de la modélisation en elle-même est de 1560 secondes, le reclassement des individus prend lui 564 secondes.

Sur l'échantillon test, le taux d'erreur est 0.61 %, comparable à ceux de CVM et BVM.




## 5.2 Comparaison des résultats

Pour situer les méthodes, nous récapitulons les principaux résultats dans le tableau suivant :

| Méthode | Modélisation | | Temps de traitement (sec.) | |
|---|---|---|---|---|
| | # points supports | Taux d'erreur en test | Apprentissage | Classement (64 700 obs.) |
| CVM [LIBCVM] | 3843 | 0.57 % | 60 | 80 |
| BVM [LIBCVM] | 2710 | 0.59 % | 27 | 57 |
| C-SVC [LIBSVM] | 33057 | 0.61 % | 1560 | 564[7] |

A paramétrage identique et performances en classement similaires par rapport C-SVC, qui est quand même une technique (et une implémentation) qui fait référence, le temps de traitement est divisé

[7] Le classement est lourdement pénalisé par la multiplication des points supports.

par 26 pour CVM, et par 57 pour BVM. De plus, la capacité à traiter des grandes bases est largement améliorée si l'on se réfère aux articles cités. En effet, seuls CVM et BVM ont pu traiter la base KDD-99 (n = 4 898 431 observations) durant leurs expérimentations.

Pour être tout à fait exhaustif, les écarts ne sont pas toujours aussi spectaculaires. En modifiant le paramétrage (ex. gamma = 0, ce qui revient à définir gamma = 1 / nombre de descripteurs), nous pouvons obtenir des classifieurs moins performants quelle que soit la méthode, mais avec des temps de calculs resserrés.

# 6 Conclusion

Les méthodes CVM et BVM de la librairie LIBCVM semblent particulièrement performantes. Elles constituent une alternative très crédible aux implémentations classiques des SVM lors du traitement des bases comportant un grand nombre d'observations.