

## 1 Introduction

« Support vector machine » pour la régression.

Les SVM (séparateur à vaste marge, machines à vecteurs de support, support vector machine en anglais) sont des méthodes bien connues en apprentissage supervisé. Leur utilisation est en revanche moins répandue en régression. On parle de « Support Vector Regression » (SVR) <sup>1</sup>.

La méthode est peu diffusée auprès des statisticiens. Pourtant, elle cumule des qualités qui la positionnent favorablement par rapport aux techniques existantes. Elle se comporte admirablement bien lorsque le ratio entre le nombre de variables et le nombre d'observations devient très défavorable, avec des prédicteurs fortement corrélés. Encore faut-il bien entendu trouver le paramétrage adéquat. Nous y reviendrons dans ce didacticiel. Autre atout, avec le principe des noyaux, il est possible de construire des modèles non linéaires sans avoir à produire explicitement de nouveaux descripteurs.

On se rend compte finalement qu'il y a une correspondance entre les SVR et la régression pénalisée, en particulier la Régression Ridge <sup>2</sup>. Ceci explique mieux la bonne tenue de l'approche aux yeux de ceux qui connaissent mal les SVM.

Le premier objectif de ce didacticiel est de montrer la mise en œuvre de deux nouveaux composants SVR de Tanagra : epsilon-SVR et nu-SVR. Ils sont issus de la bibliothèque LIBSVM (<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>) que nous utilisons par ailleurs pour l'apprentissage supervisé (voir le composant C-SVC, <http://tutoriels-data-mining.blogspot.com/2008/10/svm-comparaison-de-logiciels.html>). Nous comparerons nos résultats avec ceux du logiciel R (version 2.8.0 - <http://cran.r-project.org/>). Nous utilisons pour ce dernier le package e1071 (<http://cran.r-project.org/web/packages/e1071/index.html>) basé également sur la bibliothèque LIBSVM.

Le second objectif est de proposer un nouveau composant d'évaluation de la régression. Il est d'usage en apprentissage supervisé de scinder le fichier en deux parties, une pour la création du modèle, l'autre pour son évaluation, afin d'obtenir une estimation non biaisée des performances. Cette pratique est très peu répandue en régression. Pourtant, la procédure est nécessaire dès que nous sommes emmenés à comparer des prédicteurs de complexité différente. Nous constaterons ainsi dans ce didacticiel que les indicateurs usuels calculés sur les données d'apprentissage sont très trompeurs dans certaines situations.

## 2 Données

Nous utilisons le fichier QSAR.XLS <sup>3</sup>. Il s'agit de prédire l'activité biologique d'organismes à partir de leurs propriétés physico-chimiques <sup>4</sup>. Le fichier comporte 74 lignes et 25 colonnes : la variable à prédire est

---

<sup>1</sup> [http://fr.wikipedia.org/wiki/Machine\\_à\\_vecteurs\\_de\\_support](http://fr.wikipedia.org/wiki/Machine_à_vecteurs_de_support) ; <http://eprints.pascal-network.org/archive/00002057/01/SmoSch03b.pdf> ; <http://users.ecs.soton.ac.uk/srg/publications/pdf/SVM.pdf>

<sup>2</sup> N. Christianni, J. Shawe-Taylor, « An introduction to Support Vector Machines and other kernel-based learning methods », Cambridge University Press, 2000, section 6.2.2.

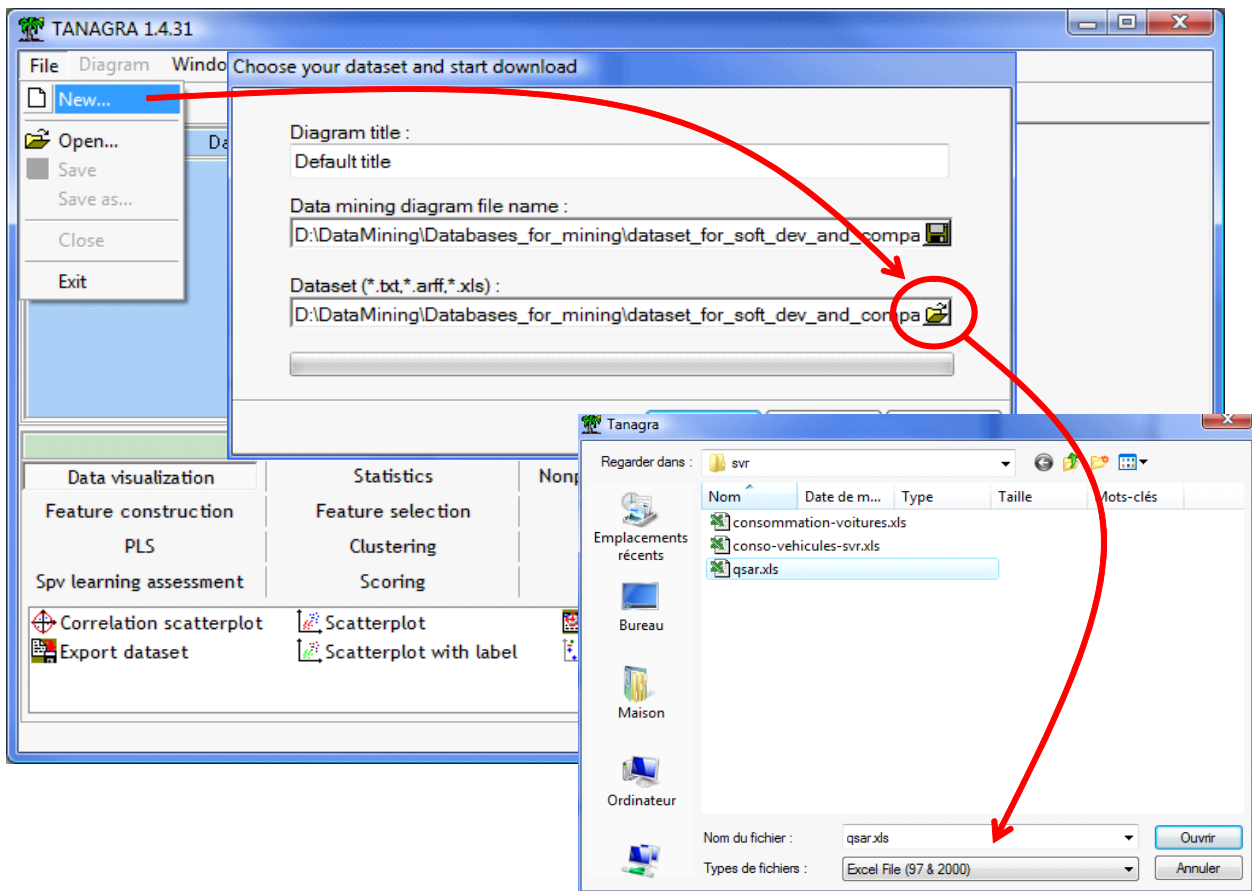
ACTIVITY ; les variables P1\_POLAR jusqu'à P3\_PI\_DONER sont les prédictives ; la colonne SUBSET sera utilisée pour subdiviser le fichier en « apprentissage » (30 observations) et « test » (44 observations).

### 3 Régression linéaire multiple avec Tanagra

Nous souhaitons tout d'abord analyser le comportement de la régression linéaire multiple, méthode de référence dans le domaine de la régression. Nous pourrions mieux situer les performances des SVR.

#### 3.1 Importation des données

Après avoir démarré Tanagra, nous actionnons le menu FILE / NEW. Nous sélectionnons le fichier QSAR.XLS<sup>5</sup>.

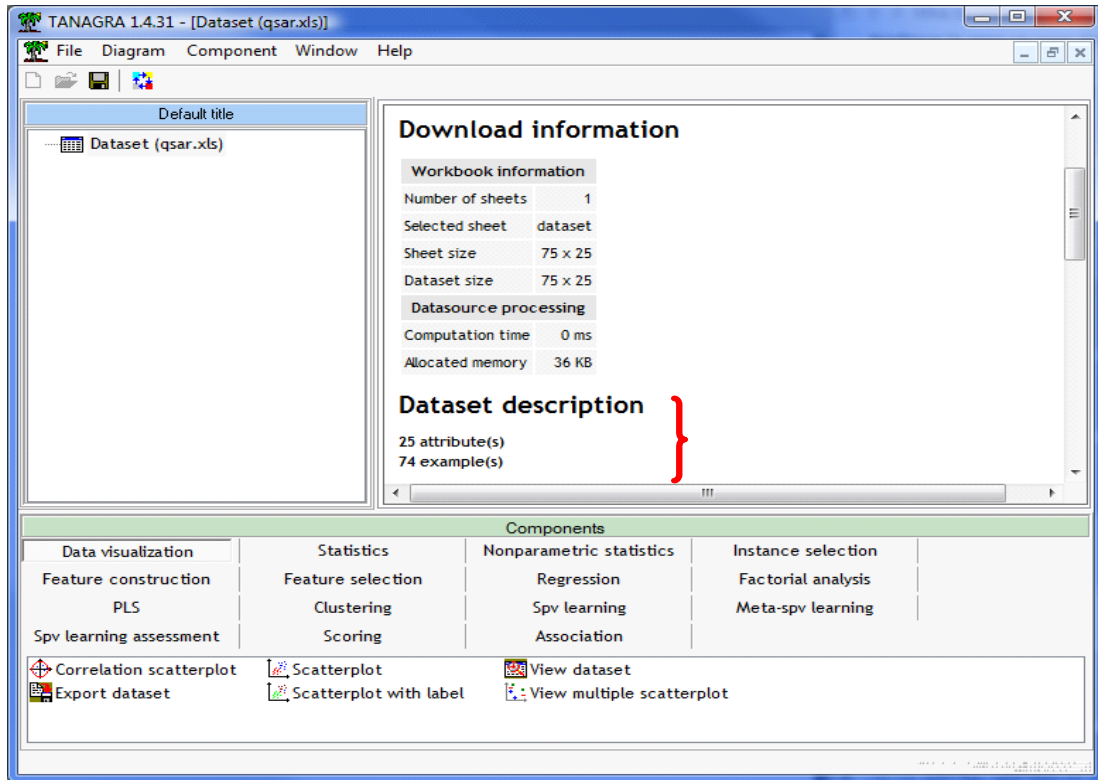


Un diagramme est directement créé. Les données sont importées. Tanagra nous indique qu'il y a 74 observations et 25 colonnes dans le fichier.

<sup>3</sup> <http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/qsar.zip>

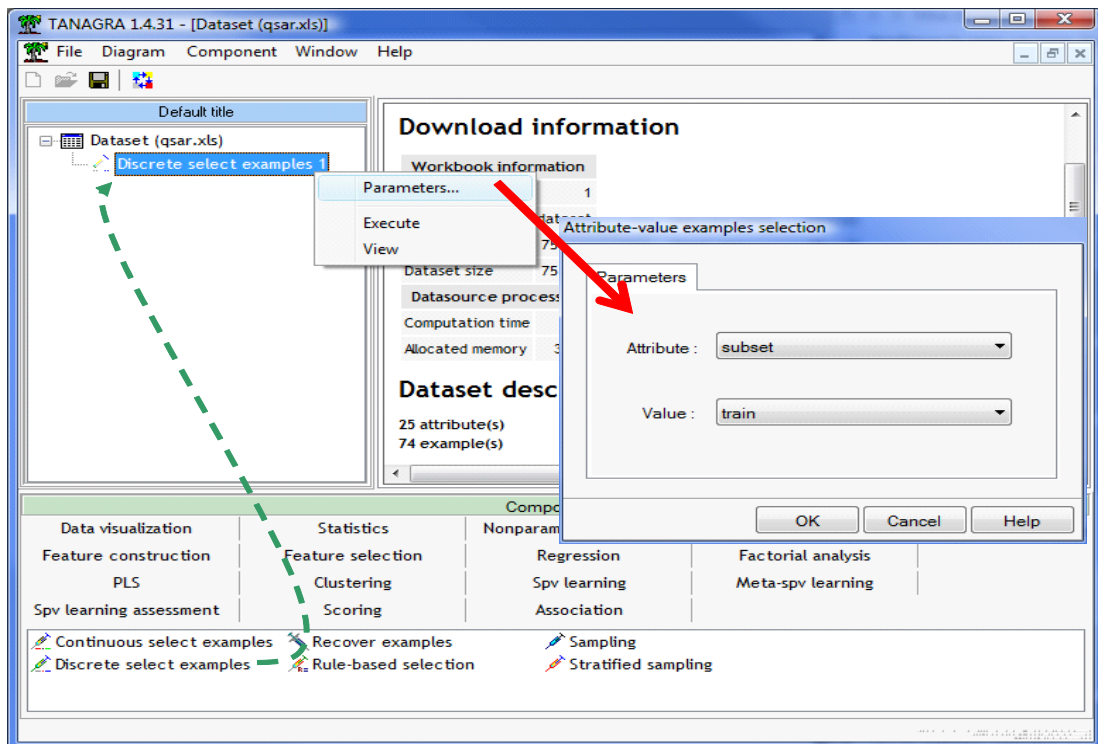
<sup>4</sup> [http://fr.wikipedia.org/wiki/Relation\\_quantitative\\_structure\\_à\\_activité](http://fr.wikipedia.org/wiki/Relation_quantitative_structure_à_activité)

<sup>5</sup> Tanagra sait importer directement le format XLS, il faut dans ce cas que le fichier ne soit pas en cours d'édition (<http://tutoriels-data-mining.blogspot.com/2008/03/importation-fichier-xls-excel-mode.html>). Il est aussi possible d'envoyer les données d'Excel à Tanagra via une macro complémentaire copiée automatiquement lors de l'installation de Tanagra (<http://tutoriels-data-mining.blogspot.com/2008/03/importation-fichier-xls-excel-macro.html>).

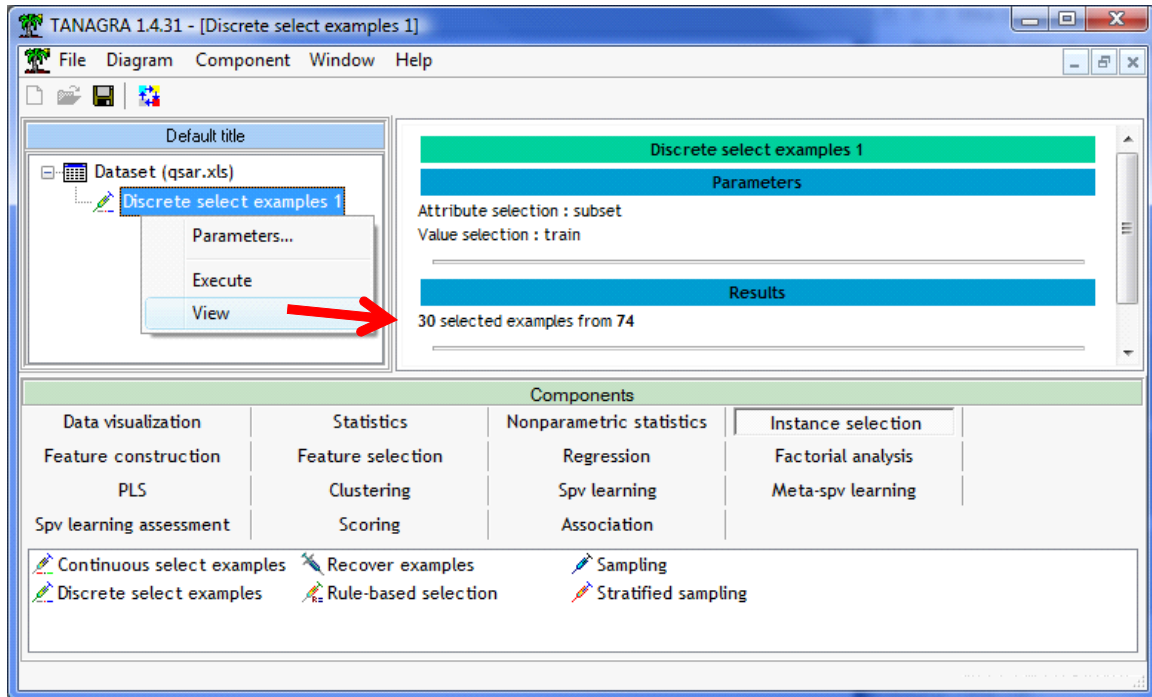


### 3.2 Subdivision apprentissage – test

Nous nous servons de la colonne SUBSET pour partitionner le fichier en données d'apprentissage et de test. Pour ce faire, nous introduisons le composant DISCRETE SELECT EXAMPLES (onglet INSTANCE SELECTION). Nous le paramétrons (menu contextuel PARAMETERS) de la manière suivante.

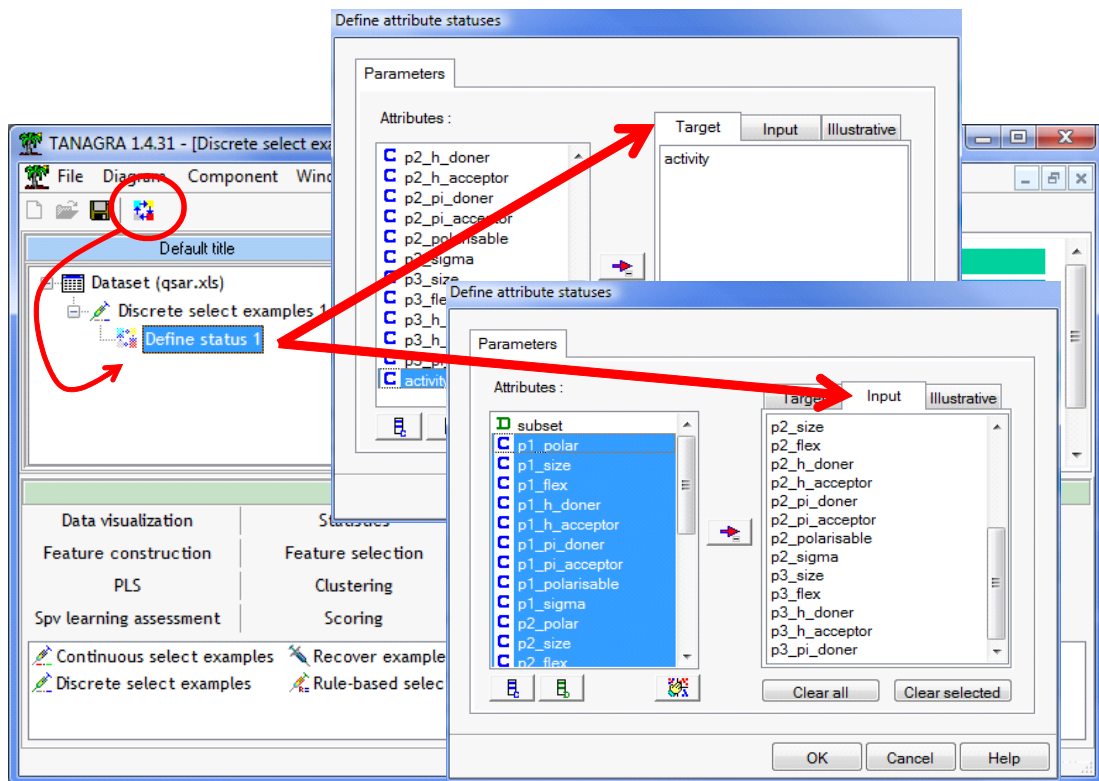


Nous validons et nous actionnons le menu VIEW pour accéder aux résultats : 30 observations sur les 74 sont réservées à la construction des modèles.



### 3.3 Variable à prédire et variables prédictives

Le composant DEFINE STATUS accessible dans la barre d'outils permet de définir le rôle des variables. Nous plaçons en TARGET la variable ACTIVITY, les autres en INPUT. La colonne SUBSET n'est pas exploitée à ce stade.



### 3.4 Régression linéaire multiple

Il reste à placer la régression linéaire multiple. Le composant MULTIPLE LINEAR REGRESSION est situé dans l'onglet REGRESSION. Nous l'insérons dans le diagramme. Puis nous actionnons le menu VIEW.

The screenshot shows the TANAGRA 1.4.31 interface. The 'Global results' panel contains the following table:

Endogenous attribute	activity
Examples	30
R <sup>2</sup>	0.988795
Adjusted-R <sup>2</sup>	0.945845
Sigma error	0.034922
F-Test (23,6)	23.0217 (0.000416)

The 'Components' panel at the bottom shows various statistical methods, including 'Multiple linear regression' which is highlighted in the 'Regression' sub-section.

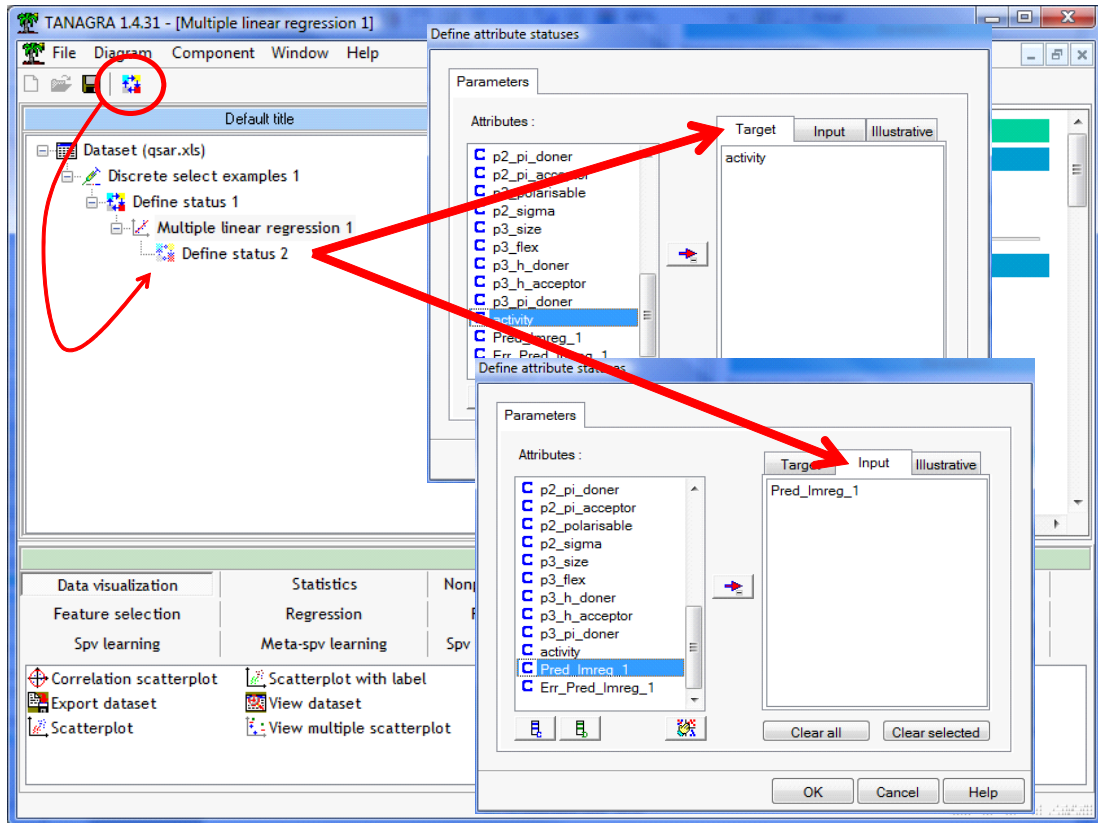
Les résultats paraissent excellents. Le modèle est globalement très significatif (la p-value du F de Fisher est égal à 0.0004),  $R^2 = 98.88\%$  de la variance de la variable à prédire est expliquée par le modèle.

S'arrêter à ce stade serait une grave erreur. Un degré de liberté de la somme des carrés des résidus égal à (d.f. = 6) devrait nous alerter. Il y a des chances que le modèle « colle » exagérément aux données d'apprentissage, ingérant ses spécificités. La fiabilité du modèle en généralisation, c.-à-d. lors de la prédiction sur de nouveaux individus, est vraisemblablement sujette à caution.

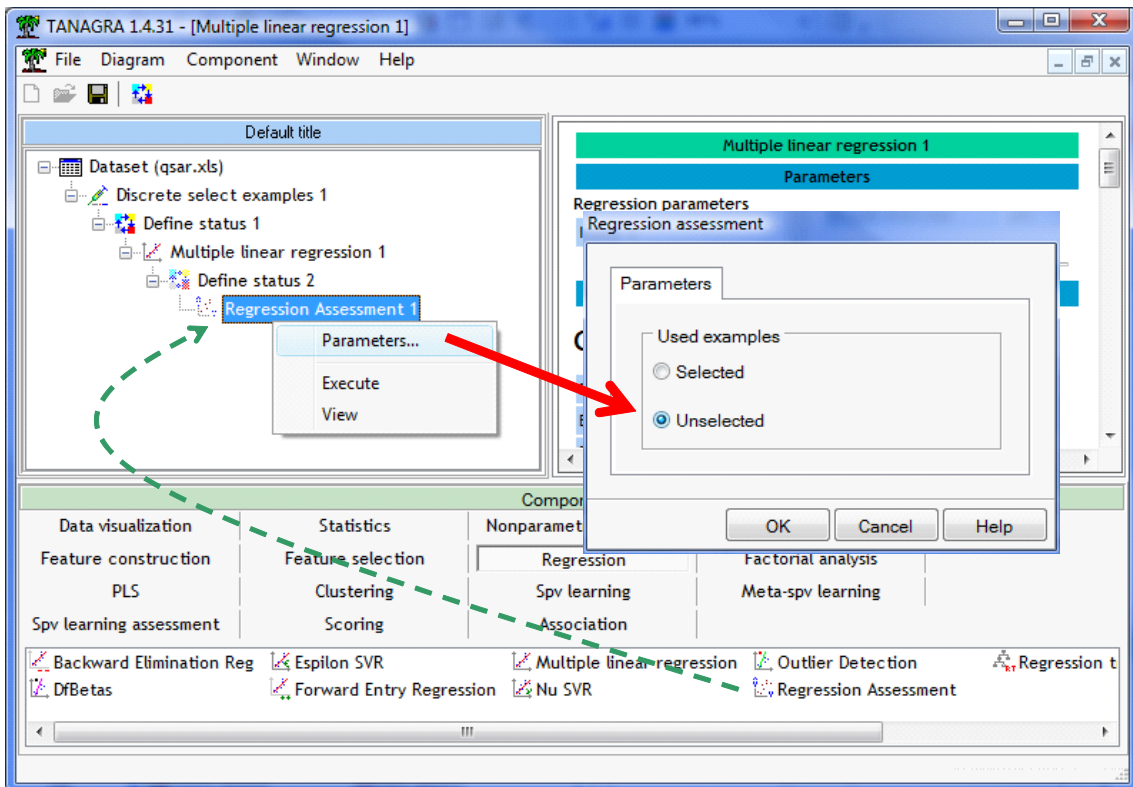
Pour s'en convaincre, rien de plus simple que d'évaluer les performances du modèle sur des observations qui n'ont pas participé à la construction du modèle. La partie « test » de notre fichier sert à cela.

### 3.5 Evaluation sur le fichier test

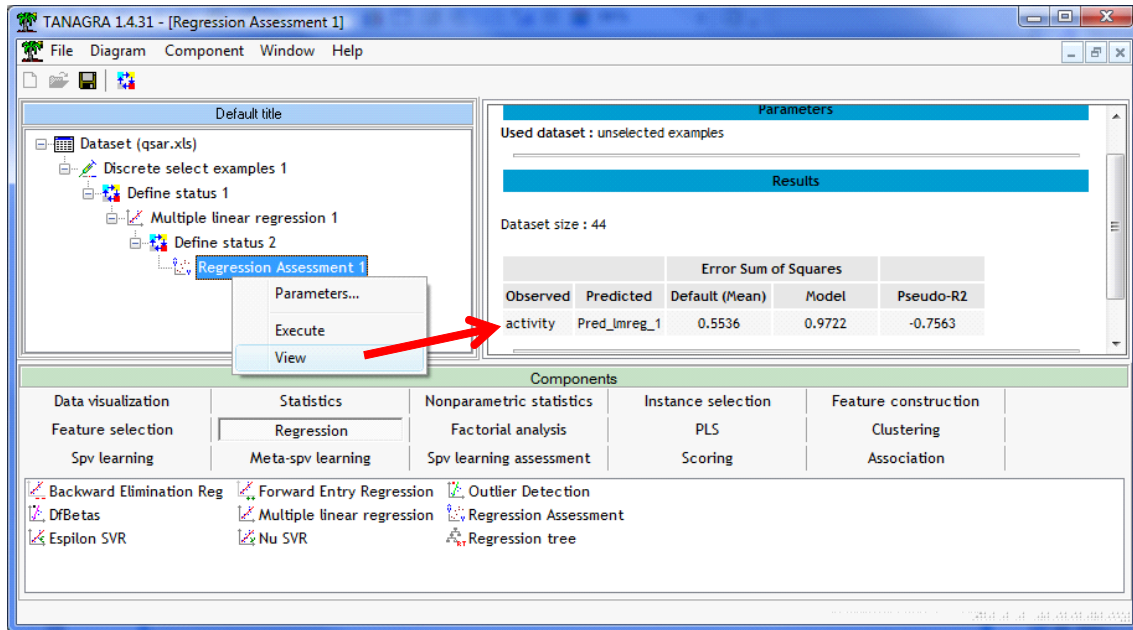
Nous souhaitons confronter les vraies valeurs de la variable ACTIVITY et la prédiction du modèle sur la partie du fichier de données correspondant à « SUBSET = TEST ». Nous devons tout d'abord spécifier les variables à confronter à l'aide du composant DEFINE STATUS. De nouveau, nous l'insérons via le raccourci dans la barre d'outils. Nous plaçons ACTIVITY en TARGET, et PRED\_LMREG\_1, produite automatiquement par la régression et calculée pour tous les individus (sélectionnés et non sélectionnés), en INPUT.



Nous pouvons maintenant introduire le composant REGRESSION ASSESSMENT. Nous le paramétrons de manière à calculer les indicateurs de performances sur les individus non sélectionnés, ceux qui ont été initialement mis à part au tout début du diagramme.



Nous cliquons sur VIEW. Les chiffres affichés méritent quelques explications.



La somme des carrés des résidus du modèle est

$$RSS_{\text{modèle}} = \sum_i (y_i - \hat{y}_{i,\text{modèle}})^2 = 0.9722$$

Plus cette valeur est faible, meilleur sera le modèle. Mais le chiffre en soi ne veut rien dire. Il faut pouvoir le comparer à une référence. On appelle « modèle par défaut », un modèle de prédiction qui n'utiliserait pas les informations en provenance des variables prédictives. Au sens des moindres carrés, la prédiction par défaut est égale à la moyenne de la variable à prédire calculée sur les données d'apprentissage c.-à-d.

$\hat{y}_{i,\text{default}} = \bar{y}_{\text{train}}$ . Les performances du modèle par défaut s'écrivent

$$RSS_{\text{default}} = \sum_i (y_i - \hat{y}_{i,\text{default}})^2 = 0.5536$$

On définit alors le pseudo-R<sup>2</sup> de la manière suivante

$$\text{Pseudo-R}^2 = 1 - \frac{RSS_{\text{modèle}}}{RSS_{\text{default}}}$$

Lorsque le modèle est parfait, nous avons Pseudo-R<sup>2</sup> = 1 ; si le modèle ne fait pas mieux que la prédiction par défaut, nous aurons Pseudo-R<sup>2</sup> = 0 ; si la valeur est négative, cela veut dire que le modèle est pire que la prédiction avec la simple moyenne de l'endogène calculée sur les données d'apprentissage, en l'absence de toute information fournie par les variables prédictives.

Dans notre cas, Pseudo-R<sup>2</sup> = 1 - 0.9722 / 0.5536 = -0.7563. Manifestement, le modèle issu de la régression linéaire multiple est à jeter aux orties. Alors qu'il semblait idyllique au regard des indicateurs de performances calculés sur les données d'apprentissage (98.8% de la variance expliquée), il se révèle particulièrement inefficace en prédiction, lorsqu'on souhaite l'appliquer sur un nouvel individu.

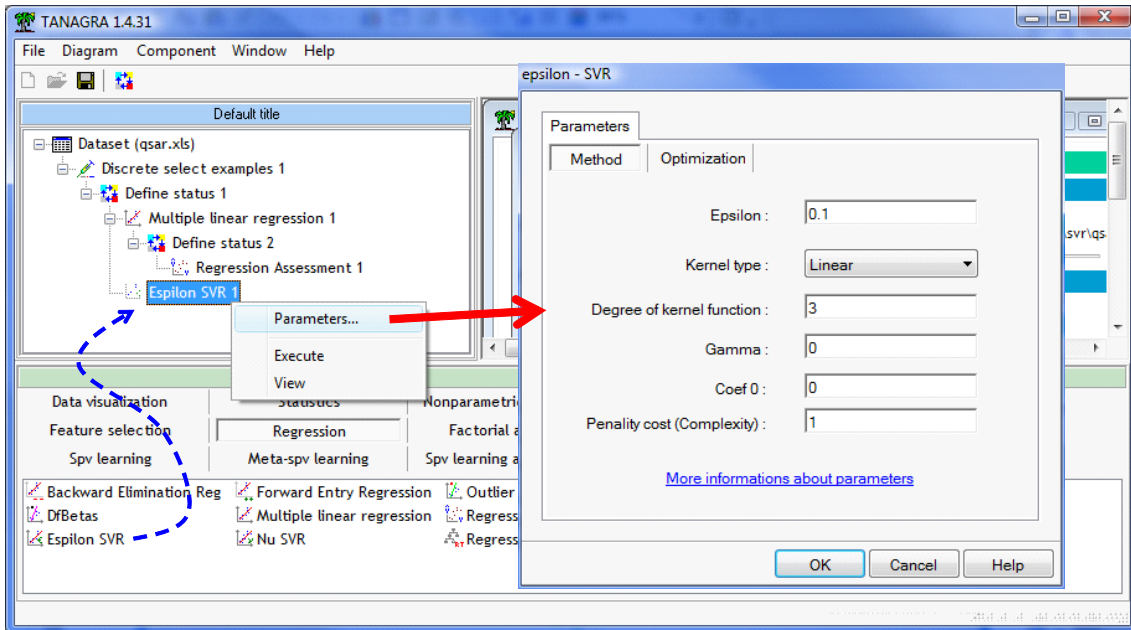
C'est toujours facile de le dire après coup. Mais dans notre cas, ce résultat était assez prévisible. Le nombre de variables prédictives est dangereusement élevé au regard du faible nombre d'observations, nous ne

pouvons qu'obtenir des résultats très instables avec la régression linéaire multiple, fortement dépendant des données d'apprentissage.

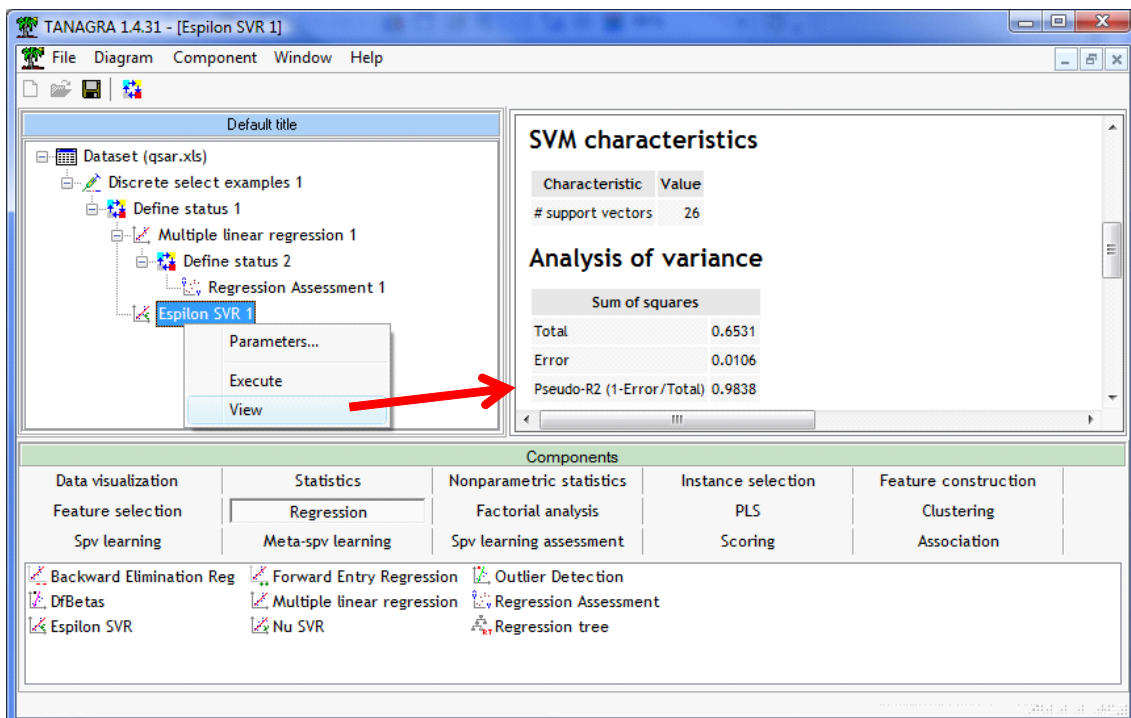
## 4 $\epsilon$ -SVR avec Tanagra

### 4.1 Apprentissage (Noyau linéaire)

Nous souhaitons maintenant introduire la méthode Epsilon-SVR (onglet REGRESSION). Nous l'insérons à la suite du composant DEFINE STATUS 1. Nous actionnons le menu PARAMETERS, les paramètres par défaut sont les suivants :



Nous ne les modifions pas dans un premier temps. Nous validons et nous actionnons le menu VIEW.



Il y a 26 observations supports dans la régression. Sur les données d'apprentissage, nous obtenons un Pseudo-R2 = 0.9838. Là également, les résultats semblent très encourageants.

## 4.2 Evaluation

Mais le comportement de la régression linéaire multiple nous invite à la prudence. Nous introduisons un dispositif d'évaluation similaire avec la séquence DEFINE STATUS (ACTIVITY en TARGET vs. PRED\_E\_SVR\_1 en INPUT) et REGRESSION ASSESSMENT (indicateurs calculés sur les données non sélectionnées).

The screenshot displays the TANAGRA 1.4.31 interface. The main workspace shows the 'Regression Assessment 2' component. The 'Parameters' section indicates 'Used dataset : unselected examples' and 'Dataset size : 44'. The 'Results' section contains a table with the following data:

Observed	Predicted	Error Sum of Squares		
		Default (Mean)	Model	Pseudo-R2
activity	Pred_e_svr_1	0.5536	0.7256	-0.3108

The 'View' menu option for the component is highlighted with a red arrow.

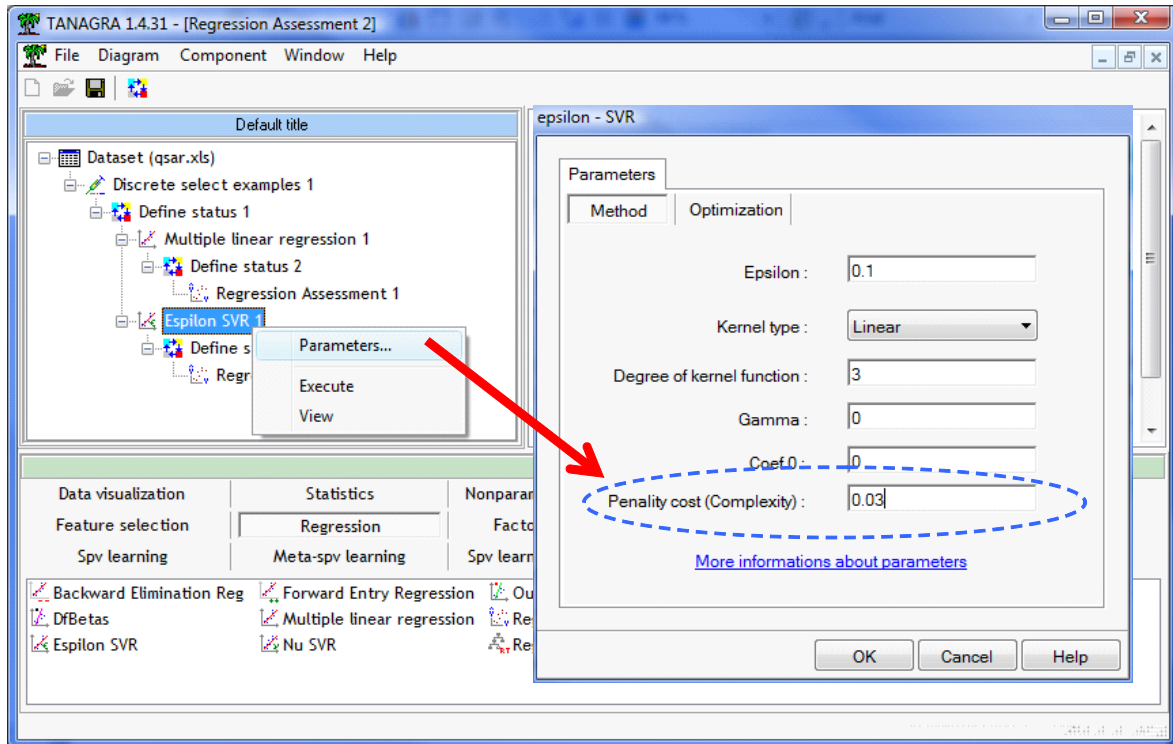
Nous obtenons un Pseudo-R2 =  $1 - 0.7256 / 0.5536 = -0.3108$ . C'est tout aussi catastrophique que pour la régression linéaire multiple. Voyons comment nous pouvons remédier à cela.

## 4.3 Modifier le paramètre de régularisation (C = 0.03)

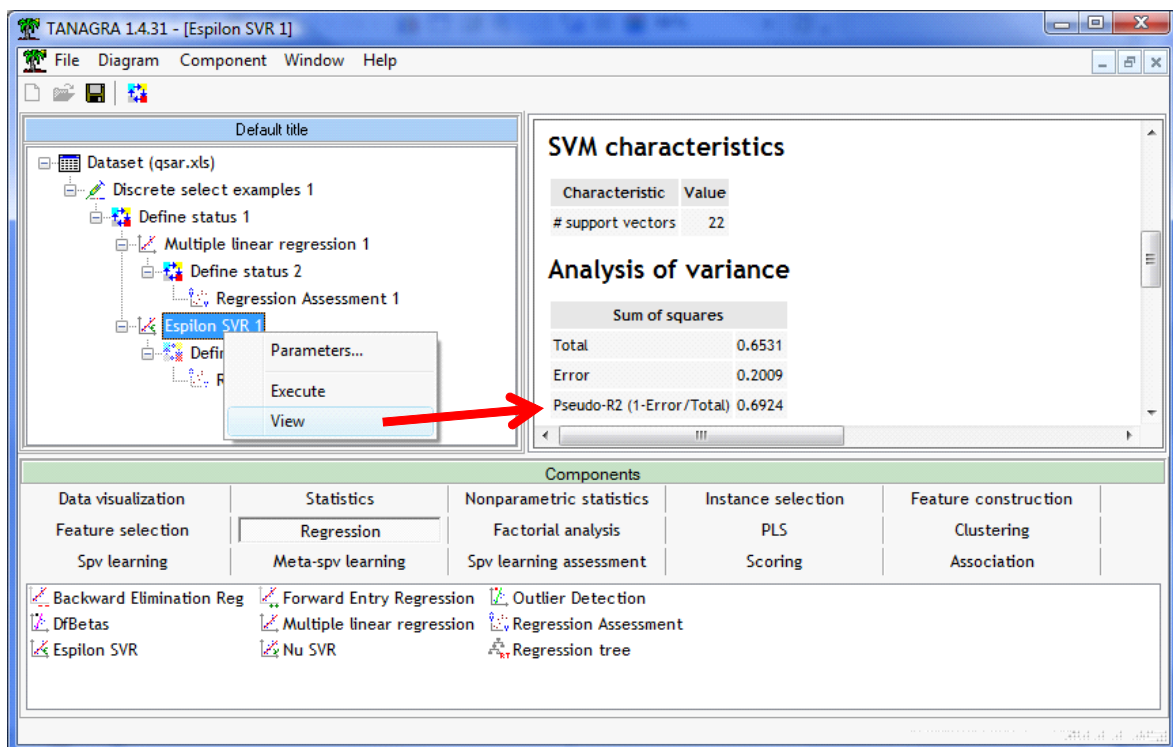
Ici commence le travail d'expertise. Manifestement, il y a sur ajustement. Nous devons lisser plus fortement le processus d'apprentissage c.-à-d. amoindrir le poids des données dans la définition du modèle de prédiction. Nous utilisons le paramètre de régularisation pour cela (Complexity Cost). Il définit le coût associé aux individus mal prédits, situés en dehors de la bande de largeur  $\epsilon$  autour de la droite de régression.

Nous cliquons sur le menu PARAMETERS du composant EPSILON SVR 1. Nous passons le paramètre « Complexity » Cost à 0.03.

**ⓘ Attention au point décimal, il dépend de la configuration de votre système d'exploitation.**



Il ne nous reste plus qu'à valider et à cliquer sur le menu VIEW. Les performances sur les données d'apprentissage sont naturellement moins bonnes avec un Pseudo-R2 = 0.6924.



Les performances en apprentissage semblent moins bonnes. Ce n'est guère étonnant, nous avons amoindri le rôle des données lors de l'élaboration du modèle. Mais c'est bien le comportement de ce dernier sur la partie test des données qui importe le plus. Nous cliquons donc sur le menu VIEW du composant REGRESSION ASSESSMENT 2.

The screenshot shows the TANAGRA 1.4.31 interface. The main window is titled 'Regression Assessment 2'. The left pane shows a workflow diagram with components like 'Dataset (qsar.xls)', 'Define status 1', 'Multiple linear regression 1', 'Define status 2', 'Regression Assessment 1', 'Epsilon SVR 1', 'Define status 3', and 'Regression Assessment 2'. The right pane shows the 'Results' section with a table of metrics. A red arrow points to the 'Pseudo-R2' value of 0.4763.

Observed	Predicted	Error Sum of Squares		
		Default (Mean)	Model	Pseudo-R2
activity	Pred_e_svr_1	0.5536	0.2899	0.4763

Sur la partie test, le Pseudo-R2 est maintenant égal à  $(1 - 0.2899 / 0.5536 =) 0.4763$ .

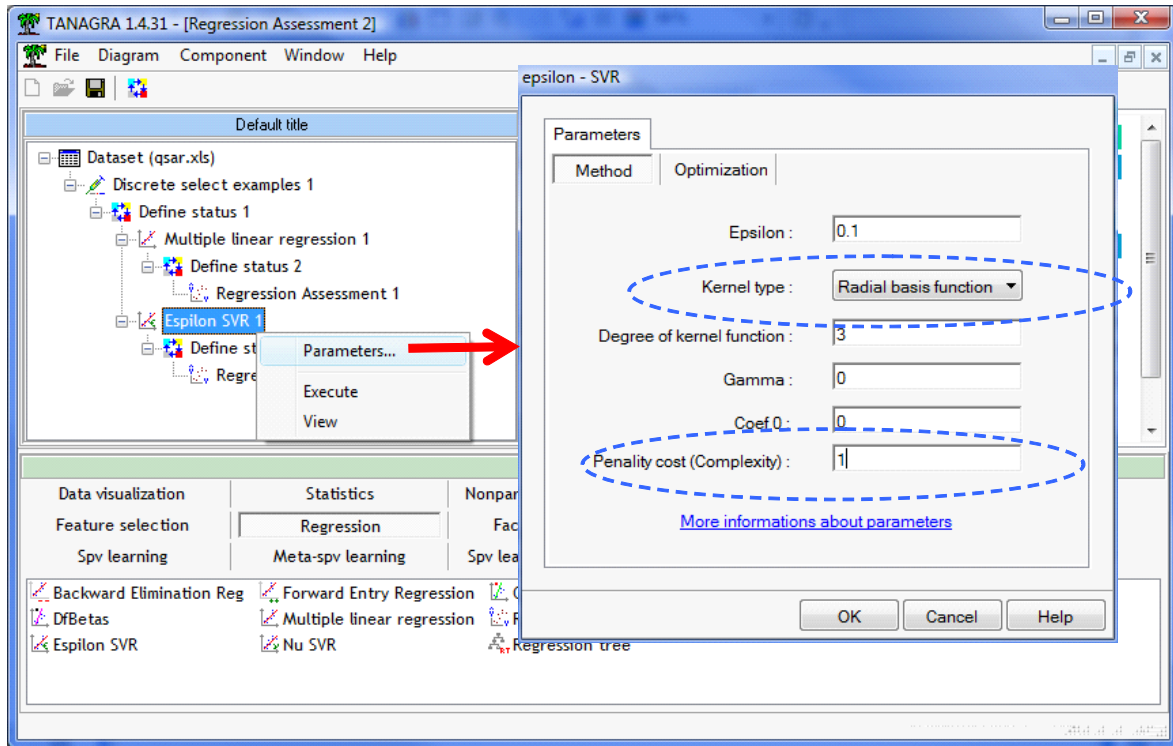
En étant moins dépendant des spécificités des données d'apprentissage, le modèle se révèle meilleur en généralisation c.-à-d. sur le fichier test. Tout le monde aura bien compris que la difficulté est, d'une part, de choisir le bon paramètre à manipuler et, d'autre part, de l'orienter dans la bonne direction.

Trouver la valeur adéquate est difficile. On n'a pas trouvé mieux que le tâtonnement à ce jour. Nous avons essayé plusieurs valeurs pour aboutir à  $C = 0.03$ , avec le risque que faire du sur apprentissage... sur les données tests, puisque nous le mettons à contribution pour choisir la meilleure solution. Plus loin, dans la section de ce tutoriel consacrée au logiciel R, nous montrons comment programmer une petite procédure qui permet de repérer automatiquement la valeur « optimale » du paramètre.

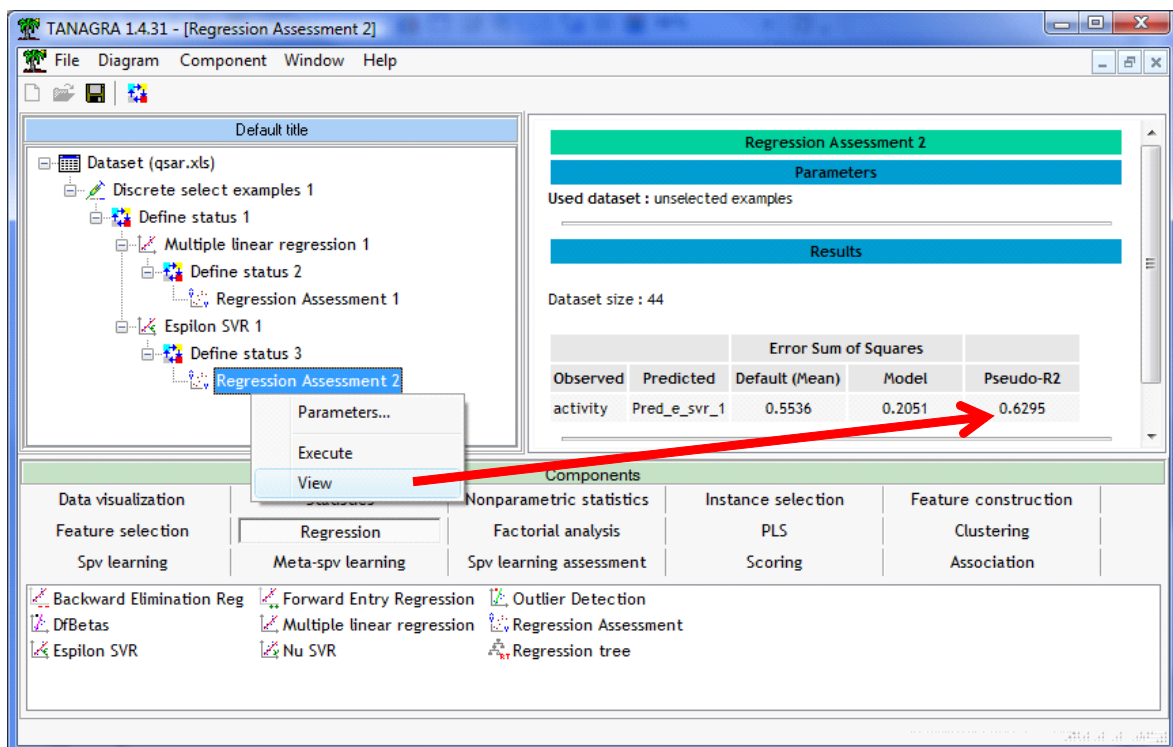
#### 4.4 Modifier le noyau (Noyau RBF avec $C = 1.0$ )

Un des principaux avantages des SVR est de pouvoir définir des modèles non linéaires disions nous plus haut. Essayons le noyau RBF (Radial Basis Function) sur nos données.

Nous actionnons de nouveau le menu PARAMETERS, nous effectuons les modifications suivantes. Nous remettons le paramètre de régularisation à 1.



Sur la partie apprentissage, le Pseudo-R2 = 0.7361. Voyons ce qu'il en est sur la partie test.



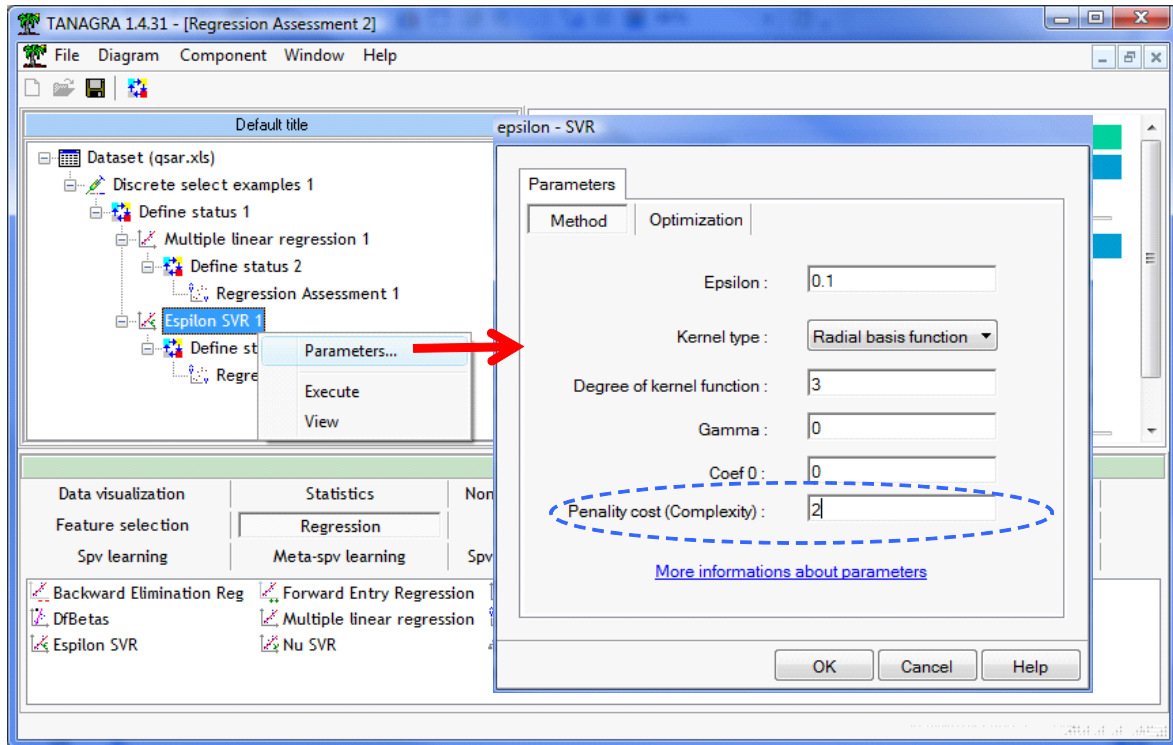
Nous obtenons Pseudo-R2 =  $1 - 0.2051 / 0.5536 = 0.6295$ . Nettement meilleur qu'avec le modèle linéaire. Dans notre situation, le noyau RBF se révèle particulièrement judicieux.

#### 4.5 Modifier le noyau et le paramètre de régularisation (Noyau RBF avec C = 2.0)

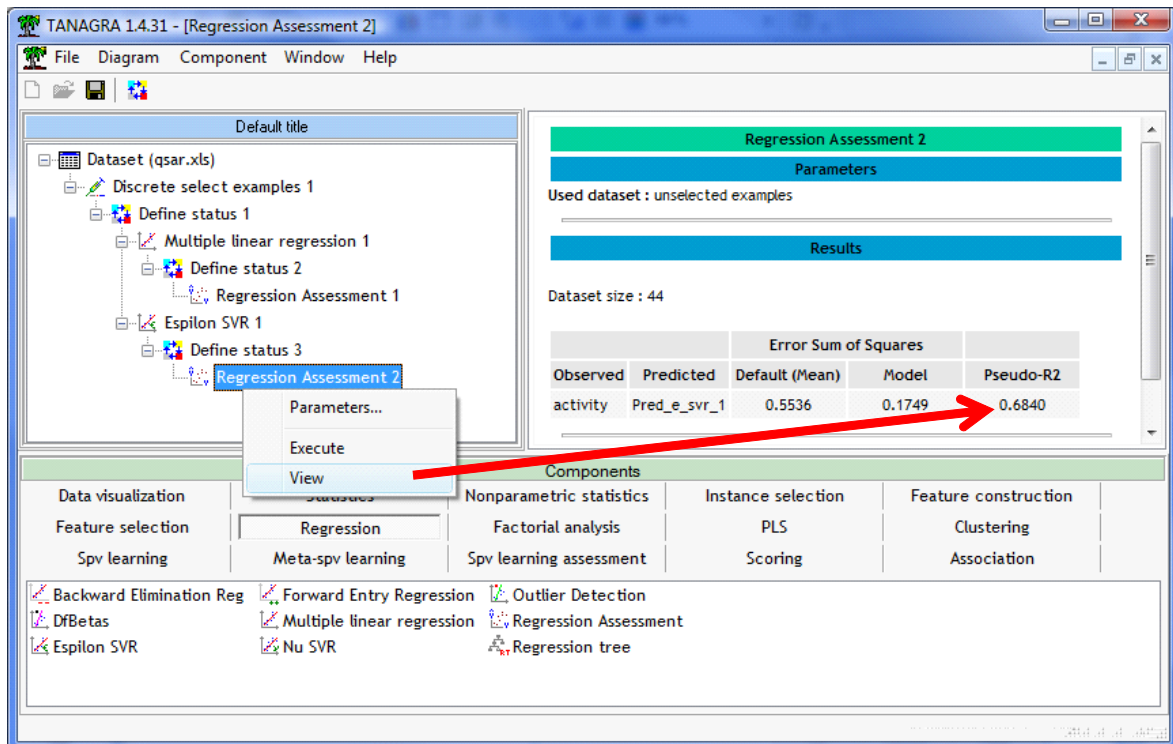
De nouveau, en manipulant intelligemment les paramètres, nous devons pouvoir améliorer la qualité de la prédiction. Ici, nous ne sommes pas en situation de sur apprentissage, c'est ce que semble nous indiquer le

faible écart entre les pseudo-R2 en apprentissage et en test. De plus, la capacité de représentation est plus élevée. Nous allons donc demander aux données de guider plus fermement la courbe de régression en passant le paramètre de complexité à 2.0.

Nous actionnons le menu PARAMETERS du composant EPSILON SVR 1, nous mettons « Complexity Cost » = 2.0. Nous validons et nous cliquons sur VIEW.



Le Pseudo-R2 en apprentissage est de 0.9227. Voyons ce qu'il en est en test.



Il est passé à  $\text{Pseudo-R}^2 = 1 - 0.1749 / 0.5536 = 0.6840$ . L'amélioration est peu spectaculaire. Il reste néanmoins qu'il s'agit là du meilleur résultat que nous avons enregistré sur nos données.

## 5 v-SVR avec Tanagra

La méthode Nu-SVR est une variante des SVR où l'on essaie de contrôler le nombre de vecteurs de support (<http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>). Par rapport à la méthode précédente, les performances ne se démarqueront pas. Le dispositif de paramétrage est différent simplement.

### 5.1 Apprentissage

Nous insérons le composant Nu-SVR dans le diagramme, à la suite de DEFINE STATUS 1. Nous actionnons directement VIEW sans manipuler les paramètres. Le noyau par défaut est linéaire.

The screenshot shows the TANAGRA 1.4.31 interface. On the left, a workflow diagram includes components like 'Dataset (qsar.xls)', 'Discrete select examples 1', 'Define status 1', 'Multiple linear regression 1', 'Define status 2', 'Regression Assessment 1', 'Epsilon SVR 1', 'Define status 3', 'Regression Assessment 2', and 'Nu SVR 1'. A context menu is open over 'Nu SVR 1', with 'View' selected. A red arrow points from the 'View' option to the 'SVM characteristics' panel on the right. This panel displays the following data:

Characteristic	Value
# support vectors	29

Below this, the 'Analysis of variance' panel shows the following data:

Sum of squares	
Total	0.6531
Error	0.0111
Pseudo-R2 (1-Error/Total)	0.9830

The bottom of the interface shows a 'Components' palette with various statistical and machine learning tools, including 'Nu SVR'.

Le Pseudo-R2 en apprentissage est de 0.9830.

### 5.2 Test

Pour le test, nous introduisons de nouveau la séquence DEFINE STATUS (TARGET = ACTIVITY, INPUT = PRED\_NU\_SVR\_1) et REGRESSION ASSESSMENT (UNSELECTED). Nous cliquons sur VIEW.

The screenshot displays the TANAGRA 1.4.31 interface for a regression assessment. The main window is titled "Regression Assessment 3". The left pane shows a hierarchical workflow starting with a dataset "qsar.xls", followed by discrete select examples, define status steps, multiple linear regression, epsilon SVR, nu SVR, and finally "Regression Assessment 3". A context menu is open over "Regression Assessment 3" with options: Parameters..., Execute, and View. A red arrow points from the "View" option to the results table.

The right pane shows the results for "Regression Assessment 3". It indicates that the used dataset is "unselected examples" and the dataset size is 44. The results table is as follows:

		Error Sum of Squares		
Observed	Predicted	Default (Mean)	Model	Pseudo-R2
activity	Pred_nu_svr_1	0.5536	0.7560	-0.3658

Additional information shown includes "Computation time : 0 ms." and "Created at 13/04/2009 19:52:43". The bottom section of the interface lists various components like Data visualization, Feature selection, Spv learning, Regression, Factorial analysis, Instance selection, Feature construction, etc.

Le Pseudo-R2 en test est égal à  $(1 - 0.7560 / 0.5536) = -0.3658$ . Ce qui n'est pas sans rappeler les premiers résultats obtenus avec le Epsilon-SVR.

Ici également, en définissant judicieusement les paramètres, nous améliorerons de manière spectaculaire les performances en prédiction.

## 6 Les SVR avec le package e1071 de R

La bibliothèque LIBSVM que nous avons utilisée dans Tanagra peut être intégrée dans le logiciel R (<http://www.r-project.org/>) via le système des packages. Il nous faut installer et charger le package **e1071** (<http://cran.r-project.org/web/packages/e1071/index.html>) avant de pouvoir travailler.

Nous essayons de reproduire les calculs de Tanagra ci-dessus, avec la même subdivision « apprentissage – test » des données. Nous devrions obtenir des résultats très similaires, mais pas forcément exactement les mêmes. En effet, l'optimisation étant basée sur des heuristiques, il est possible que la procédure, utilisée dans des environnements différents, ne produise pas le même optimum. Il y aurait matière à s'inquiéter cependant si les résultats étaient très dissemblables.

### 6.1 Importation des données et partition « apprentissage – test »

Dans un premier temps, nous importons le fichier Excel. Nous utilisons le package **xlsReadWrite**. Nous exploitons la colonne SUBSET pour partitionner les données.

```
library(xlsReadWrite)
library(e1071)

setwd("D:/DataMining/Databases_for_mining/dataset_for_soft_dev_and_comparison/regression/svr")
donnees <- read.xls(file="qsar.xls")
summary(donnees)

#partitioning into training and testing set
donnees.train <- donnees[donnees$subset=="train",2:ncol(donnees)]
donnees.test <- donnees[donnees$subset=="test",2:ncol(donnees)]
```

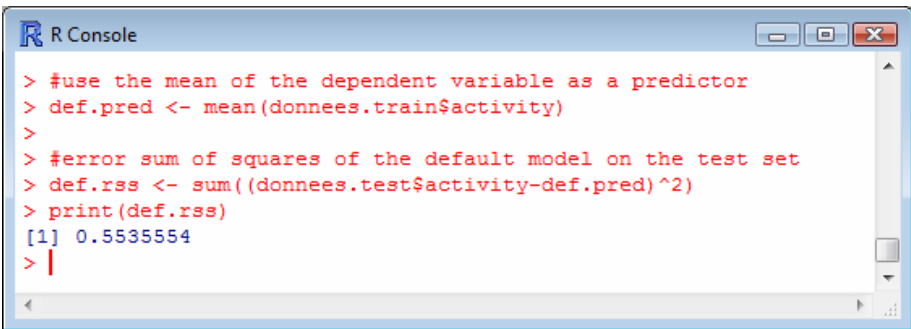
### 6.2 Définir le modèle par défaut

Nous définissons le modèle par défaut c.-à-d. la moyenne de la variable à prédire sur l'échantillon d'apprentissage. Nous mesurons ses performances en prédiction sur l'échantillon test. Nous calculons pour cela la somme des carrés des erreurs.

```
#use the mean of the dependent variable as a predictor
def.pred <- mean(donnees.train$activity)

#error sum of squares of the default model on the test set
def.rss <- sum((donnees.test$activity-def.pred)^2)
print(def.rss)
```

Nous obtenons  $RSS_{\text{default}} = \sum_i (y_i - \hat{y}_{i,\text{default}})^2 = 0.5536$ , identique à celui de Tanagra.



```
R Console
> #use the mean of the dependent variable as a predictor
> def.pred <- mean(donnees.train$activity)
>
> #error sum of squares of the default model on the test set
> def.rss <- sum((donnees.test$activity-def.pred)^2)
> print(def.rss)
[1] 0.5535554
> |
```

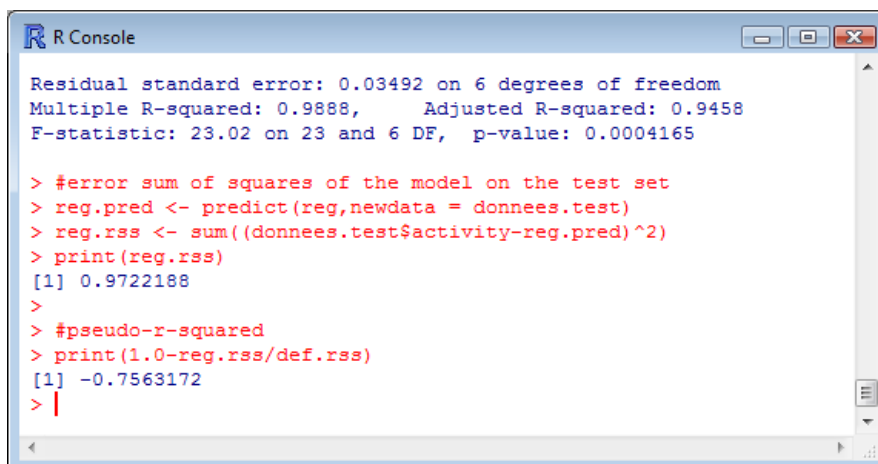
### 6.3 Régression linéaire multiple

Nous pouvons entamer la modélisation. Nous lançons la régression linéaire multiple sur les données d'apprentissage. Nous appliquons le modèle sur les données de test. Nous mesurons la somme des carrés des erreurs et nous en déduisons le Pseudo-R2.

```
#####
#linear regression
#####
reg <- lm(activity ~., data = donnees.train)
print(summary(reg))
#error sum of squares of the model on the test set
reg.pred <- predict(reg,newdata = donnees.test)
reg.rss <- sum((donnees.test$activity-reg.pred)^2)
print(reg.rss)

#pseudo-r-squared
print(1.0-reg.rss/def.rss)
```

Alors que la régression semble excellente sur les données d'apprentissage, les résultats s'avèrent catastrophiques en prédiction, avec un Pseudo-R2 = -0.7563, à l'instar de ce que nous avons pu le constater avec Tanagra.



```
R Console
Residual standard error: 0.03492 on 6 degrees of freedom
Multiple R-squared: 0.9888, Adjusted R-squared: 0.9458
F-statistic: 23.02 on 23 and 6 DF, p-value: 0.0004165

> #error sum of squares of the model on the test set
> reg.pred <- predict(reg,newdata = donnees.test)
> reg.rss <- sum((donnees.test$activity-reg.pred)^2)
> print(reg.rss)
[1] 0.9722188
>
> #pseudo-r-squared
> print(1.0-reg.rss/def.rss)
[1] -0.7563172
> |
```

### 6.4 $\epsilon$ -SVR linéaire avec deux scénarios de paramètre de complexité

**Cost = 1.0.** Nous souhaitons mettre en œuvre la méthode Epsilon-SVR. Nous appelons la procédure `svm` du package `e1071`.

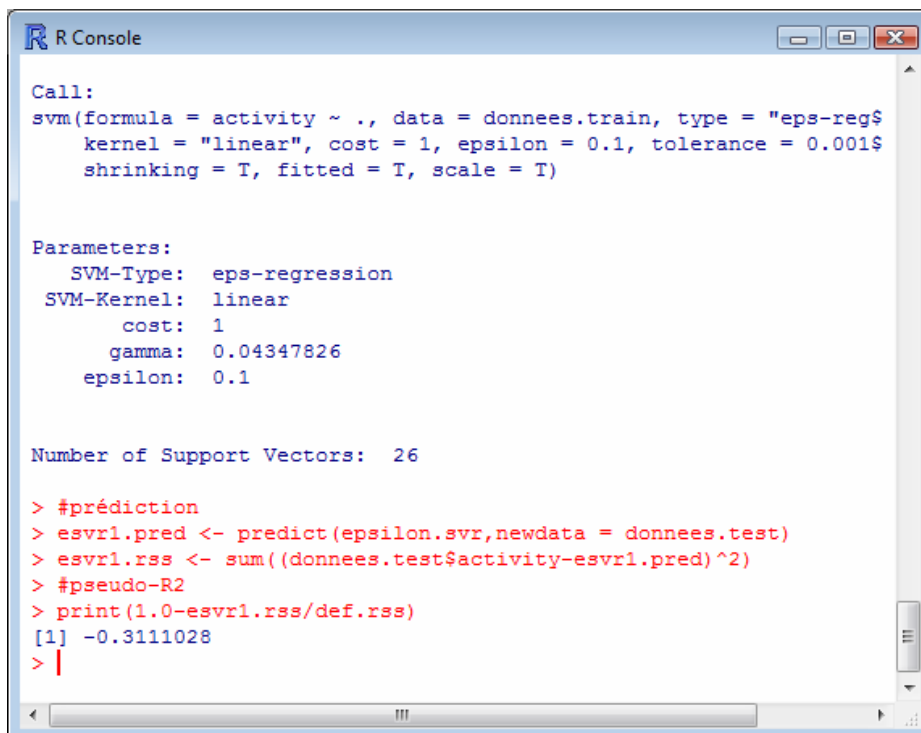
Nous définissons les mêmes paramètres qu'avec Tanagra. Nous fixons COST = 1.0 avec un noyau linéaire. Une fois le modèle construit, nous l'appliquons sur les données tests, puis nous calculons le Pseudo-R2.

```

#####
#linear epsilon-svr with cost = 1.0
#####
epsilon.svr <- svm(activity ~.,data = donnees.train, scale = T, type = "eps-regression",
                 kernel = "linear", cost = 1.0, epsilon=0.1,tolerance=0.001, shrinking=T,
                 fitted=T)
print(epsilon.svr)
#prédiction
esvr1.pred <- predict(epsilon.svr,newdata = donnees.test)
esvr1.rss <- sum((donnees.test$activity-esvr1.pred)^2)
#pseudo-R2
print(1.0-esvr1.rss/def.rss)

```

R nous fournit les résultats suivants :



```

R Console
Call:
svm(formula = activity ~ ., data = donnees.train, type = "eps-reg$
   kernel = "linear", cost = 1, epsilon = 0.1, tolerance = 0.001$
   shrinking = T, fitted = T, scale = T)

Parameters:
  SVM-Type:  eps-regression
  SVM-Kernel: linear
           cost: 1
           gamma: 0.04347826
           epsilon: 0.1

Number of Support Vectors: 26

> #prédiction
> esvr1.pred <- predict(epsilon.svr,newdata = donnees.test)
> esvr1.rss <- sum((donnees.test$activity-esvr1.pred)^2)
> #pseudo-R2
> print(1.0-esvr1.rss/def.rss)
[1] -0.3111028
> |

```

Le Pseud-R2 (-0.3110) est quasi identique à celui produit avec Tanagra. Manifestement, la technique est mal paramétrée pour le problème que nous traitons. Nous allons manipuler le paramètre de régularisation.

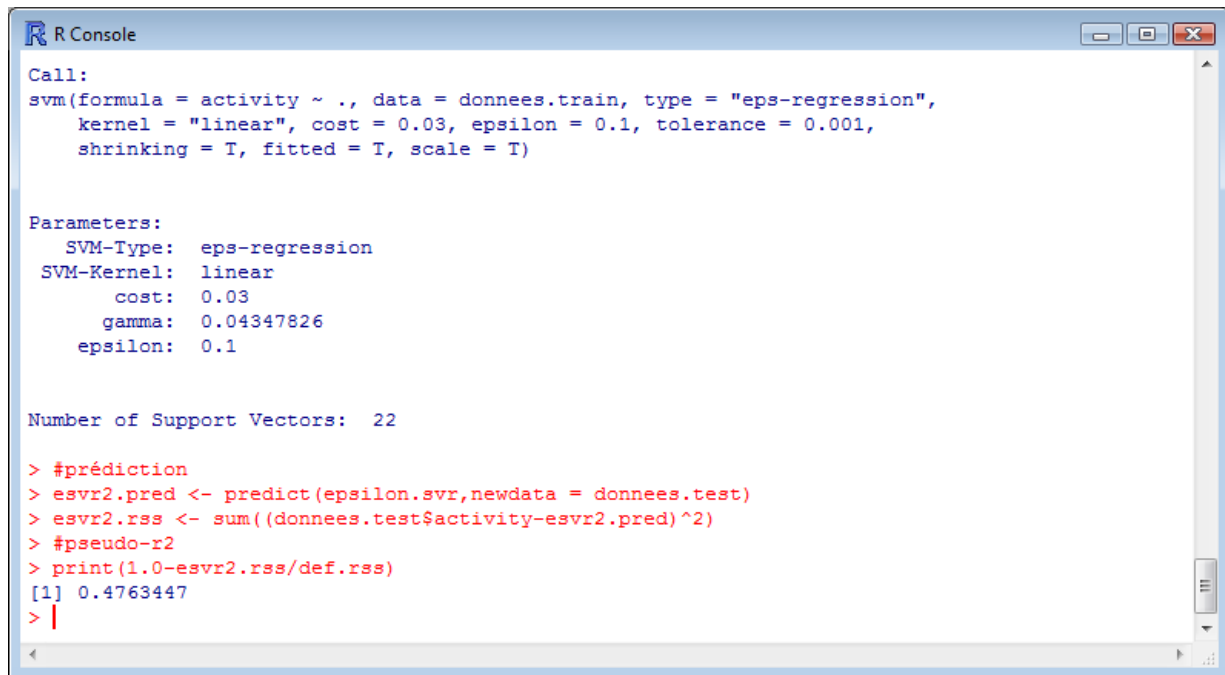
**Cost = 0.03.** A l'instar de ce que nous avons fait dans Tanagra, nous modifions le paramètre de régularisation (COST = 0.03). Nous observons les conséquences sur les données de test.

```

#####
#linear epsilon-svr with cost = 0.03
#####
epsilon.svr <- svm(activity ~.,data = donnees.train, scale = T, type = "eps-regression",
                 kernel = "linear", cost = 0.03, epsilon=0.1,tolerance=0.001, shrinking=T,
                 fitted=T)
print(epsilon.svr)
#prédiction
esvr2.pred <- predict(epsilon.svr,newdata = donnees.test)
esvr2.rss <- sum((donnees.test$activity-esvr2.pred)^2)
#pseudo-r2
print(1.0-esvr2.rss/def.rss)

```

L'amélioration est toujours aussi spectaculaire avec un Pseudo-R2 = 0.4763.



```

R Console
Call:
svm(formula = activity ~ ., data = donnees.train, type = "eps-regression",
     kernel = "linear", cost = 0.03, epsilon = 0.1, tolerance = 0.001,
     shrinking = T, fitted = T, scale = T)

Parameters:
  SVM-Type:  eps-regression
  SVM-Kernel: linear
           cost: 0.03
           gamma: 0.04347826
           epsilon: 0.1

Number of Support Vectors: 22

> #prédiction
> esvr2.pred <- predict(epsilon.svr,newdata = donnees.test)
> esvr2.rss <- sum((donnees.test$activity-esvr2.pred)^2)
> #pseudo-r2
> print(1.0-esvr2.rss/def.rss)
[1] 0.4763447
> |

```

## 6.5 Recherche automatique du paramètre de régularisation (1) – Noyau linéaire

L'énorme avantage de R, par rapport à tous les logiciels pilotés par diagramme de traitements, est que nous pouvons programmer, relativement facilement, des procédures complexes. Bien sûr, il faut faire l'apprentissage du langage de programmation. Mais, une fois passée cette barrière à l'entrée, les bénéfices que l'on peut en tirer sont considérables.

Dans notre cas, nous allons tester une série de valeurs du paramètre de régularisation COST. L'objectif est de détecter automatiquement la valeur adéquate pour le problème que nous traitons. Nous testons COST = 0.005 à 1.0 avec un pas de 0.005. La procédure s'écrit comme suit.

```

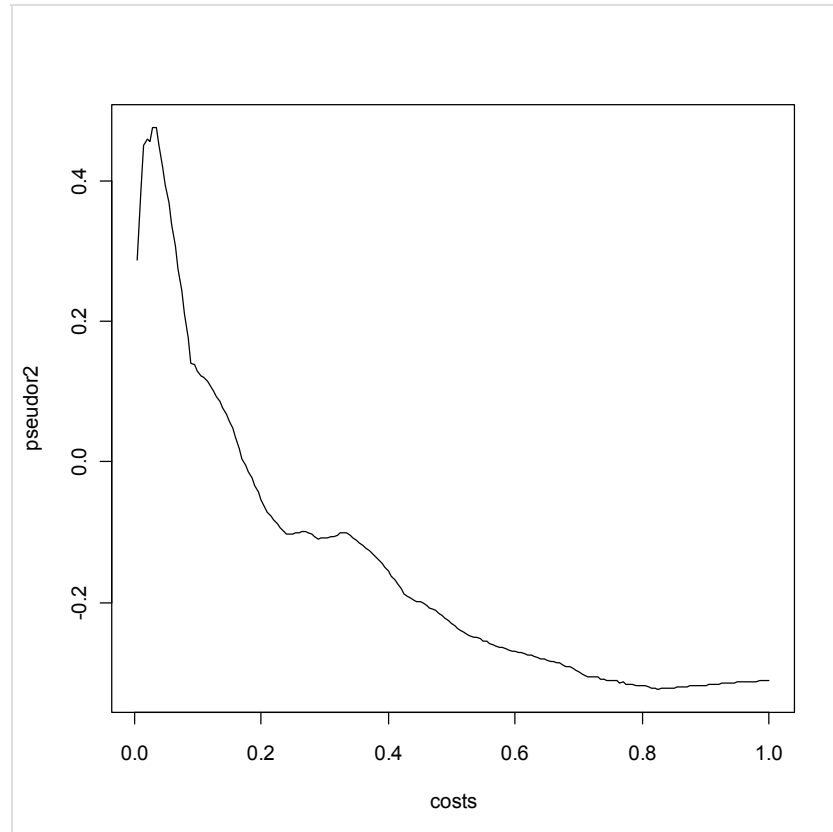
#*****
#detect the "best" cost parameter
#*****
costs <- seq(from=0.005,to=1.0,by=0.005)
pseudor2 <- double(length(costs))
for (c in 1:length(costs)){
  epsilon.svr <- svm(activity ~.,data = donnees.train, scale = T, type = "eps-regression",
                    kernel = "linear", cost = costs[c], epsilon=0.1,tolerance=0.001, shrinking=T,
                    fitted=T)

  #prédiction
  esvr.pred <- predict(epsilon.svr,newdata = donnees.test)
  esvr.rss <- sum((donnees.test$activity-esvr.pred)^2)
  pseudor2[c] <- 1.0-esvr.rss/def.rss
}

#graphical representation
plot(costs,pseudor2,type="l")
#show the max. of pseudo-r2 and the corresponding cost parameter
print(max(pseudor2))
k <- which.max(pseudor2)
print(costs[k])

```

Pour faciliter la lecture, nous mettons les résultats sous forme de graphique. Nous pourrions détecter non seulement la valeur optimale, mais aussi les plages de « bonnes » valeurs. C'est très important si l'on souhaite s'assurer de la fiabilité des résultats.



L'optimum est dans une zone assez étroite autour de COST = 0.03.

```
R Console
> #show the max. of pseudo-r2 and the corresponding cost parameter
> print(max(pseudor2))
[1] 0.4763447
> k <- which.max(pseudor2)
> print(costs[k])
[1] 0.03
>
> |
```

Nous remarquons également que la valeur par défaut COST = 1.0 proposée par les logiciels est particulièrement inadaptée pour le problème que nous traitons.

## 6.6 Recherche du paramètre de régularisation (2) – Noyau linéaire

Option très intéressante de la bibliothèque e1071, il existe déjà une procédure interne qui permet de retrouver les « bons » paramètres de la méthode par tâtonnement. Elle procède par validation croisée. Il n'est donc pas nécessaire de disposer d'un fichier test à part. Elle permet également d'explorer les combinaisons de paramètres (si l'on souhaite croiser les paramètres EPSILON et COST par exemple). La rapidité de la procédure est assez impressionnante si l'on considère tous les calculs réalisés en interne.

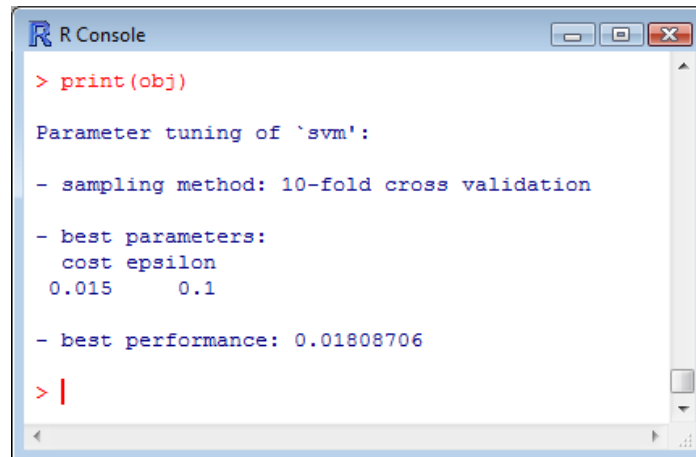
Le code source s'écrit comme suit.

```

#####
#other approach based on cross-validation in order to obtain the "best" cost parameter
#####
obj <- tune.svm(activity ~., data = donnees.train, scale = T, type = "eps-regression",
               kernel = "linear", cost = seq(from=0.005,to=1.0,by=0.005),
               epsilon=0.1, tolerance=0.001, shrinking=T, fitted=T)
print(obj)
plot(obj$performances[,1],obj$performances[,3],type="l")

```

La valeur « optimale » est assez différente (COST = 0.015) de celle obtenue avec la méthode externe basée sur les données tests que nous avons programmée dans la section précédente.



```

> print(obj)

Parameter tuning of `svm':

- sampling method: 10-fold cross validation

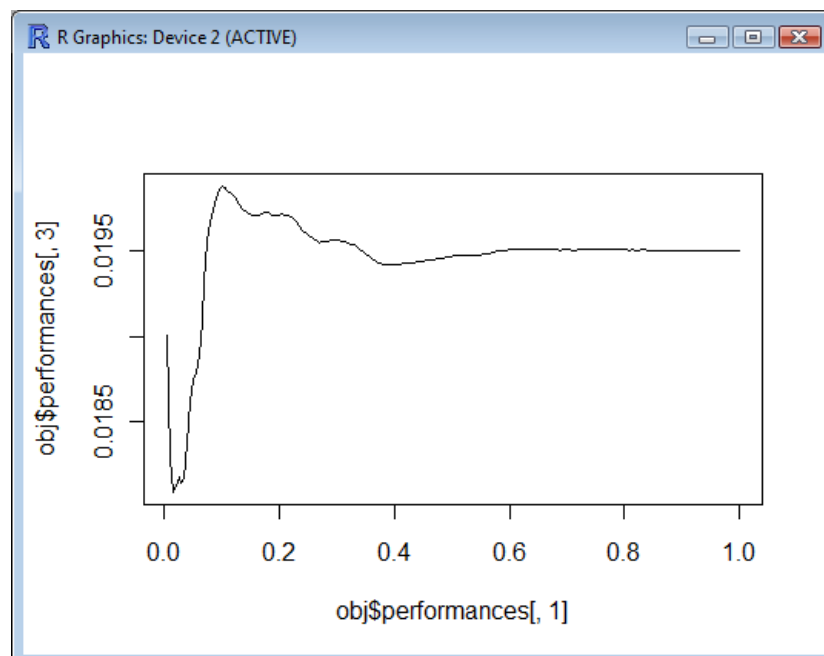
- best parameters:
  cost epsilon
  0.015      0.1

- best performance: 0.01808706

> |

```

On peut expliquer cette différence par la très faible taille du fichier d'apprentissage, engendrant une certaine instabilité des résultats. On constate toutefois que la « zone d'optimalité » est quasiment la même. C'est plutôt rassurant.



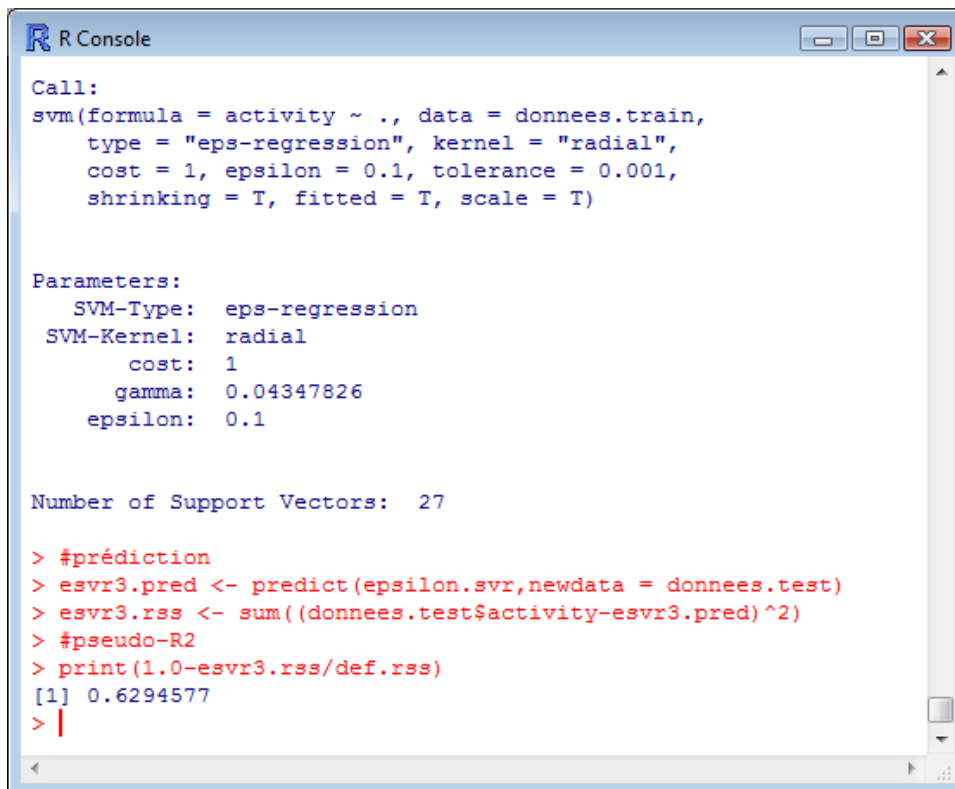
Attention, l'indicateur de performance utilisée par la procédure est le RMSE [root mean squared error]. Plus le RMSE sera faible, meilleur sera le modèle.

## 6.7 Utiliser un noyau RBF pour $\epsilon$ -SVR

Tout comme avec Tanagra, nous passons au noyau RBF. Nous construisons le modèle sur la partie apprentissage, nous le testons sur la seconde partie.

```
#####
#rbf epsilon-svr with cost = 1.0
#####
epsilon.svr <- svm(activity ~ ., data = donnees.train, scale = T, type = "eps-regression",
                  kernel = "radial", cost = 1.0, epsilon=0.1, tolerance=0.001, shrinking=T,
                  fitted=T)
print(epsilon.svr)
#prédiction
esvr3.pred <- predict(epsilon.svr, newdata = donnees.test)
esvr3.rss <- sum((donnees.test$activity-esvr3.pred)^2)
#pseudo-R2
print(1.0-esvr3.rss/def.rss)
```

Nous obtenons de bien meilleurs résultats qu'avec le noyau linéaire.



```
R Console
Call:
svm(formula = activity ~ ., data = donnees.train,
    type = "eps-regression", kernel = "radial",
    cost = 1, epsilon = 0.1, tolerance = 0.001,
    shrinking = T, fitted = T, scale = T)

Parameters:
  SVM-Type:  eps-regression
  SVM-Kernel: radial
           cost: 1
           gamma: 0.04347826
           epsilon: 0.1

Number of Support Vectors: 27

> #prédiction
> esvr3.pred <- predict(epsilon.svr, newdata = donnees.test)
> esvr3.rss <- sum((donnees.test$activity-esvr3.pred)^2)
> #pseudo-R2
> print(1.0-esvr3.rss/def.rss)
[1] 0.6294577
> |
```

## 6.8 Recherche automatique du paramètre de régularisation – Noyau RBF

Ici également, nous pouvons programmer une petite procédure ad hoc pour déterminer le paramètre de régularisation optimal pour notre problème. Nous nous basons uniquement sur le fichier test dans cette section.

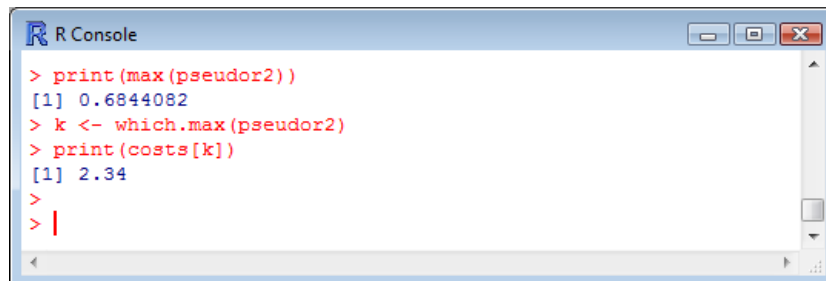
```

#####
#detect the "best" cost parameter for rbf epsilon-svr
#####
costs <- seq(from=0.05,to=3.0,by=0.005)
pseudor2 <- double(length(costs))
for (c in 1:length(costs)){
  epsilon.svr <- svm(activity ~.,data = donnees.train, scale = T, type = "eps-regression",
                    kernel = "radial", cost = costs[c], epsilon=0.1,tolerance=0.001, shrinking=T,
                    fitted=T)

  #prédiction
  esvr.pred <- predict(epsilon.svr,newdata = donnees.test)
  esvr.rss <- sum((donnees.test$activity-esvr.pred)^2)
  pseudor2[c] <- 1.0-esvr.rss/def.rss
}
#graphical representation
plot(costs,pseudor2,type="l")
#show the max. of pseudo-r2 and the corresponding cost parameter
print(max(pseudor2))
k <- which.max(pseudor2)
print(costs[k])

```

Le paramètre optimal est COST = 2.34.

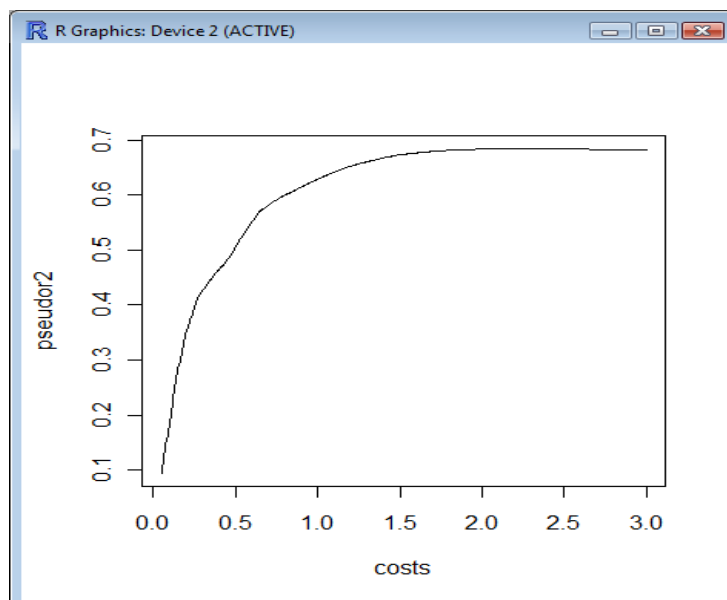


```

R Console
> print(max(pseudor2))
[1] 0.6844082
> k <- which.max(pseudor2)
> print(costs[k])
[1] 2.34
>
> |

```

Et surtout, l'allure de la courbe de performances est très différente. Manifestement, il faut augmenter la sensibilité aux points lorsque l'on travaille avec un noyau RBF pour traiter notre problème.



## 6.9 v-SVR avec R

Tout ce que l'on vient de mettre en place est bien entendu valable pour la méthode NU-SVR. Voici à titre d'exemple la mise en œuvre de la méthode avec les paramètres usuels.

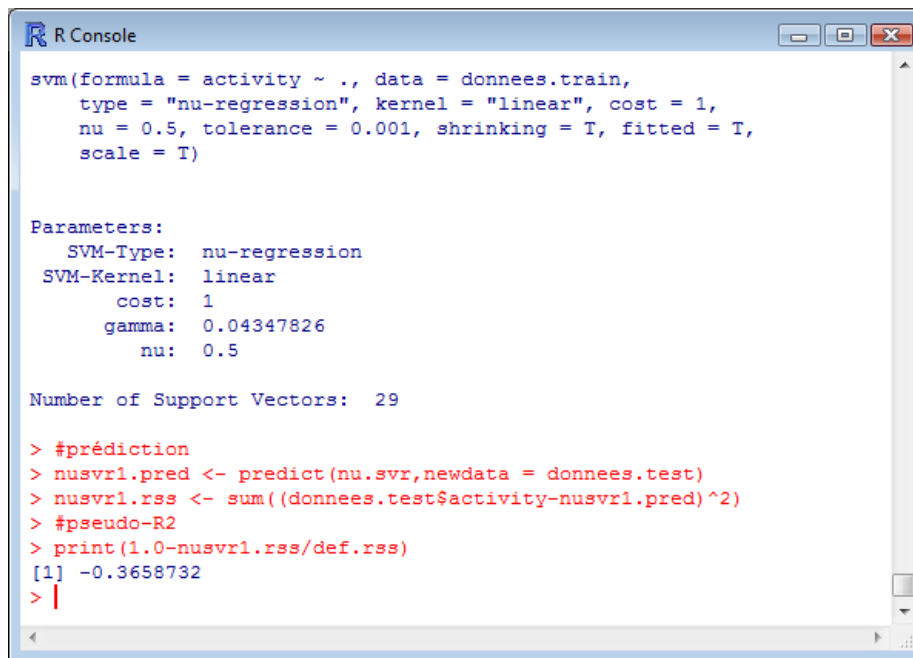
```

#####
#linear nu-svr with cost = 1.0
#####
nu.svr <- svm(activity ~., data = donnees.train, scale = T, type = "nu-regression",
              kernel = "linear", cost = 1.0, nu = 0.5, tolerance = 0.001, shrinking=T,
              fitted=T)

print(nu.svr)
#prédiction
nusvr1.pred <- predict(nu.svr, newdata = donnees.test)
nusvr1.rss <- sum((donnees.test$activity-nusvr1.pred)^2)
#pseudo-R2
print(1.0-nusvr1.rss/def.rss)

```

Nous obtenons un Pseudo-R2 = -0.3658, identique à celui de Tanagra.



```

R Console

svm(formula = activity ~ ., data = donnees.train,
    type = "nu-regression", kernel = "linear", cost = 1,
    nu = 0.5, tolerance = 0.001, shrinking = T, fitted = T,
    scale = T)

Parameters:
  SVM-Type:  nu-regression
  SVM-Kernel:  linear
            cost: 1
            gamma: 0.04347826
            nu: 0.5

Number of Support Vectors: 29

> #prédiction
> nusvr1.pred <- predict(nu.svr, newdata = donnees.test)
> nusvr1.rss <- sum((donnees.test$activity-nusvr1.pred)^2)
> #pseudo-R2
> print(1.0-nusvr1.rss/def.rss)
[1] -0.3658732
> |

```

## 7 Conclusion

Dans une démarche exploratoire, nous devons répondre à 3 questions pour aboutir à des résultats satisfaisants : (1) spécifier la famille de méthodes appropriée compte tenu du problème à résoudre (régression) ; (2) choisir la technique la plus efficace compte tenu des caractéristiques des données (SVR) ; (3) définir les paramètres adéquats pour un apprentissage efficace.

Ce dernier point n'est certainement pas le plus facile. Il faut une bonne connaissance des techniques, il faut la mettre en relation avec les caractéristiques des données. Souvent, nous devons nous en remettre à une démarche expérimentale pour trouver la solution « optimale ». Elle doit être la plus rigoureuse possible.

Dans ce didacticiel, nous avons montré comment mettre en œuvre les SVR (Support Vector Regression) dans Tanagra et R. Nous nous sommes attachés à orienter adroitement le paramètre de régularisation après avoir défini un critère d'évaluation calculé sur un fichier test.