

1 Introduction

« Support vector machine » for regression.

Support Vector Machine (SVM) is a well-know approach in the machine learning community. It is usually implemented for a classification problem in a supervised learning framework. But SVM can be used also in a regression problem, where we want to predict or explain the values taken by a continuous dependent variable. We say Support Vector Regression in this context¹ (SVR).

The method is not widely diffused among statisticians. Yet it combines several desirable properties compared with existing techniques. It has a good behavior even if the ratio between the number of variables and the number of observations becomes very unfavorable, with highly correlated predictors. Another advantage is the principle of kernel (the famous "kernel trick"). It is possible to construct a non-linear model without explicitly having to produce new descriptors. A deeply study of the characteristics of the method allows to make comparison with penalized regression such as ridge regression².

The first subject of this tutorial is to show how to use two new SVR components of the 1.4.31 version of Tanagra. They are based on the famous LIBSVM library (<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>). We use the same library for the classification (see C-SVC component, <http://data-mining-tutorials.blogspot.com/2008/11/svm-using-libsvm-library.html>). We compare our results to those of the R software (version 2.8.0 - <http://cran.r-project.org/>). We utilize the e1071 package for R (<http://cran.r-project.org/web/packages/e1071/index.html>). It is also based on the LIBSVM library.

The second subject is to propose a new assessment component for the regression. It is usual in the supervised learning framework to split the dataset into two parts, the first for the learning process, the second for its evaluation, in order to obtain an unbiased estimation of the accuracy. We can implement the same approach for the regression. The procedure is even essential when we try to compare models with various complexities (or various degrees of freedom). We will see in this tutorial that the usual indicators computed on the learning data are highly misleading in some situations. We must use an independent test set when we want assess a model (or use a resampling scheme).

2 Dataset

We use the QSAR.XLS³ data file. We want to predict a biological activity from chemical structure⁴. The data file contains 74 examples and 25 variables: the dependent attribute is ACTIVITY; the variables from

¹ http://en.wikipedia.org/wiki/Support_vector_machine ; <http://eprints.pascal-network.org/archive/00002057/01/SmoSch03b.pdf> ; <http://users.ecs.soton.ac.uk/srg/publications/pdf/SVM.pdf>

² 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.

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

⁴ http://en.wikipedia.org/wiki/Quantitative_structure-activity_relationship

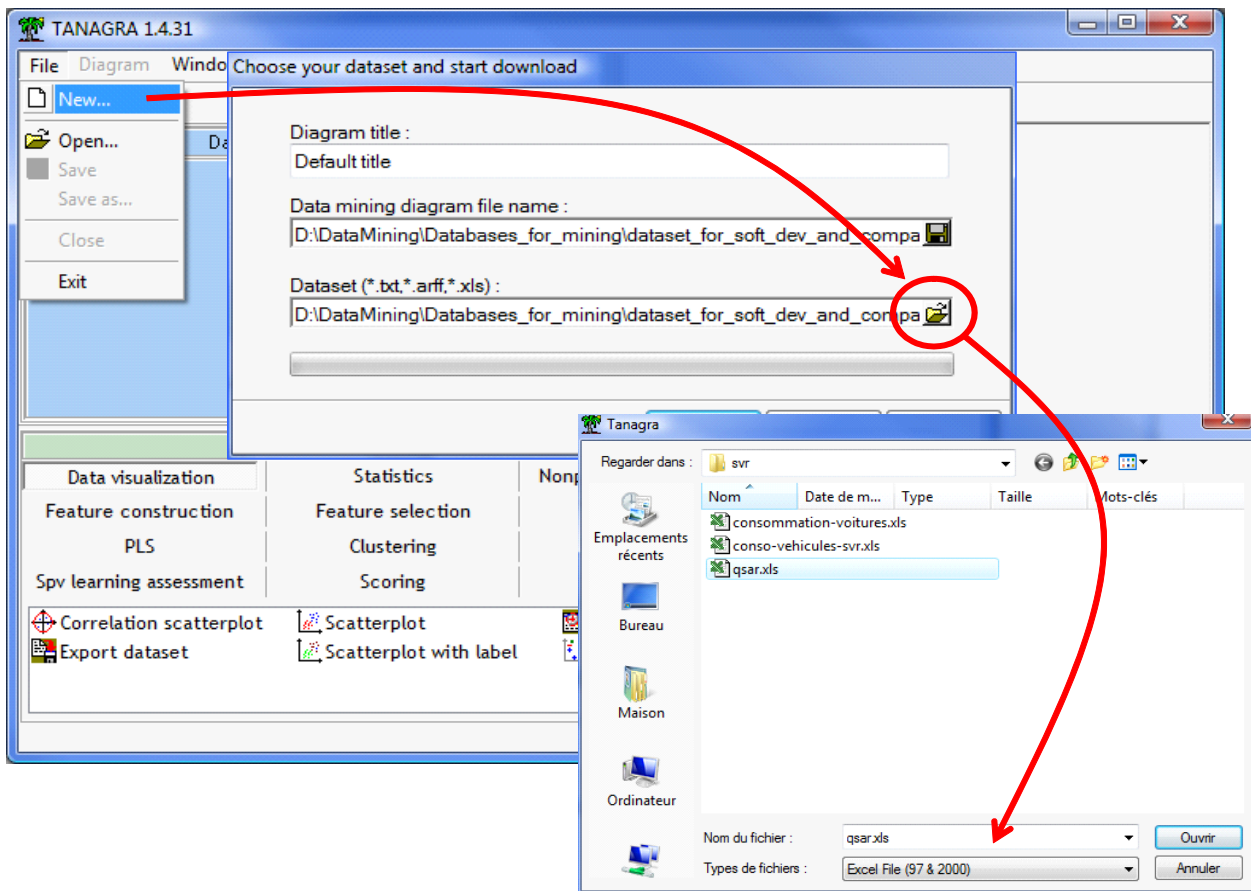
P1_POLAT to P3_PI_DONER are the independent attributes; SUBSET is used for subdividing the dataset in a train set (30 examples) and test set (44 examples).

3 Multiple linear regression with Tanagra

First we want to study the behavior of the state of the art regression technique i.e. the linear regression⁵. Our idea is to compare the behavior of the SVR with this method. Is the SVR is really better for our QSAR problem?

3.1 Data importation

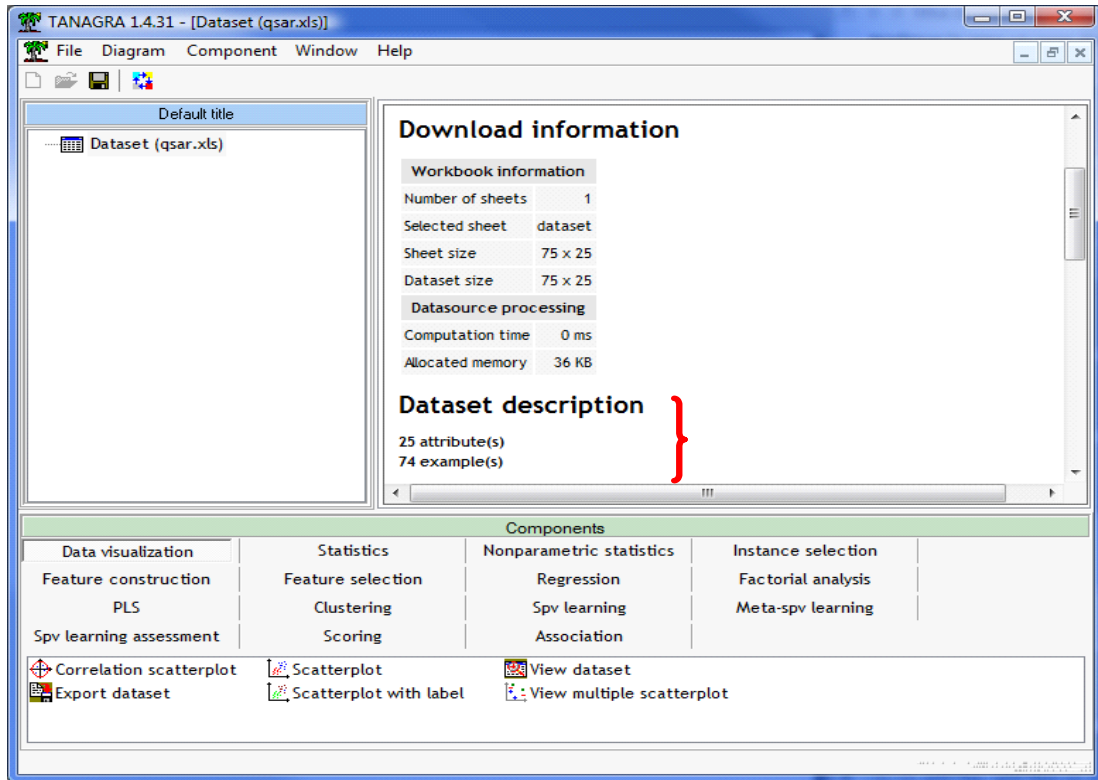
After we launch Tanagra, we click on the FILE / NEW menu. We select the data file QSAR.XLS⁶.



The diagram is automatically created. The dataset is loaded. Tanagra displays the number of examples and variables in the current data file.

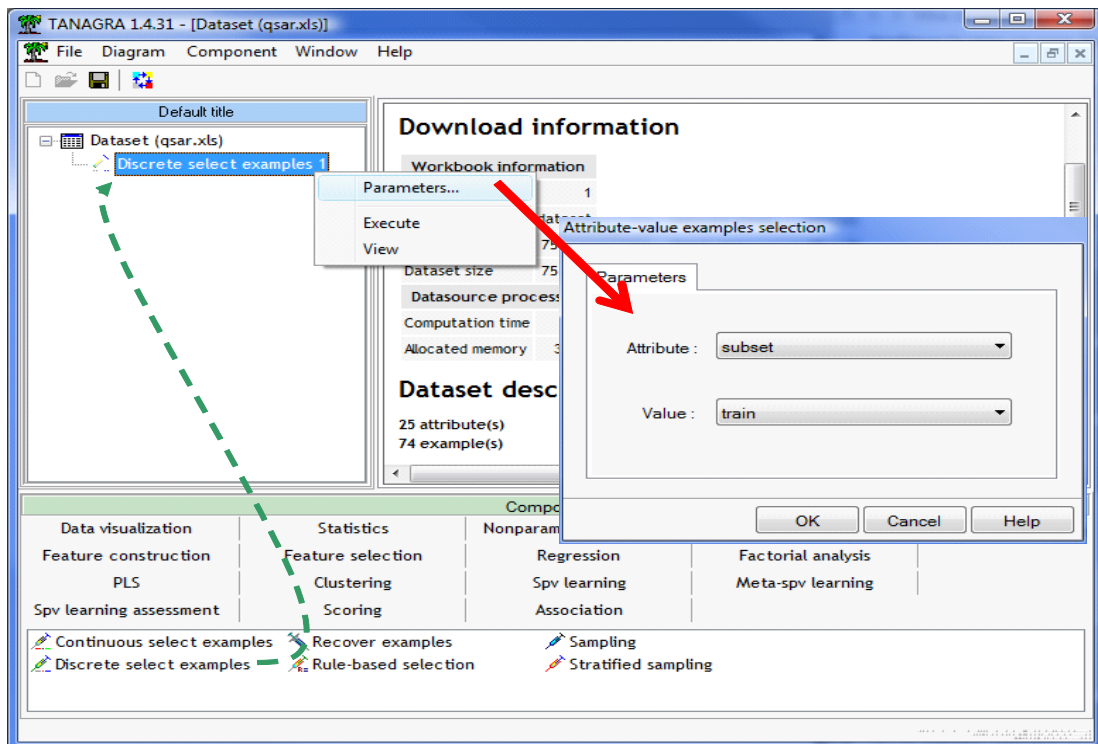
⁵ http://en.wikipedia.org/wiki/Linear_Regression

⁶ Tanagra can handle directly the XLS file format. In this context, the dataset had not in the same time to be loaded into Excel (<http://data-mining-tutorials.blogspot.com/2008/10/excel-file-format-direct-importation.html>). We can also use an add-on to send the dataset from Excel to Tanagra. The process is very simple in this situation (<http://data-mining-tutorials.blogspot.com/2008/10/excel-file-handling-using-add-in.html>)

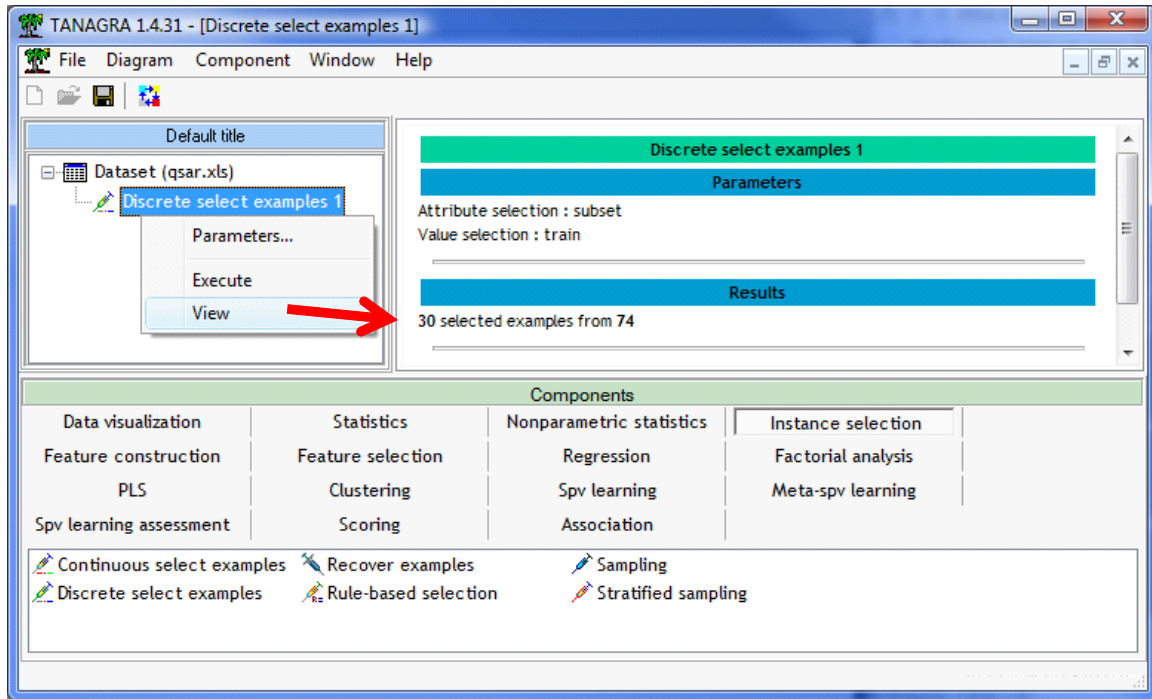


3.2 Subdividing the dataset into train and test sets

We want to use the SUBSET column for the dataset subdivision. We insert the DISCRETE SELECT EXAMPLES (INSTANCE SELECTION tab) component into the diagram. We set the following parameters by clicking on the PARAMETERS contextual menu.

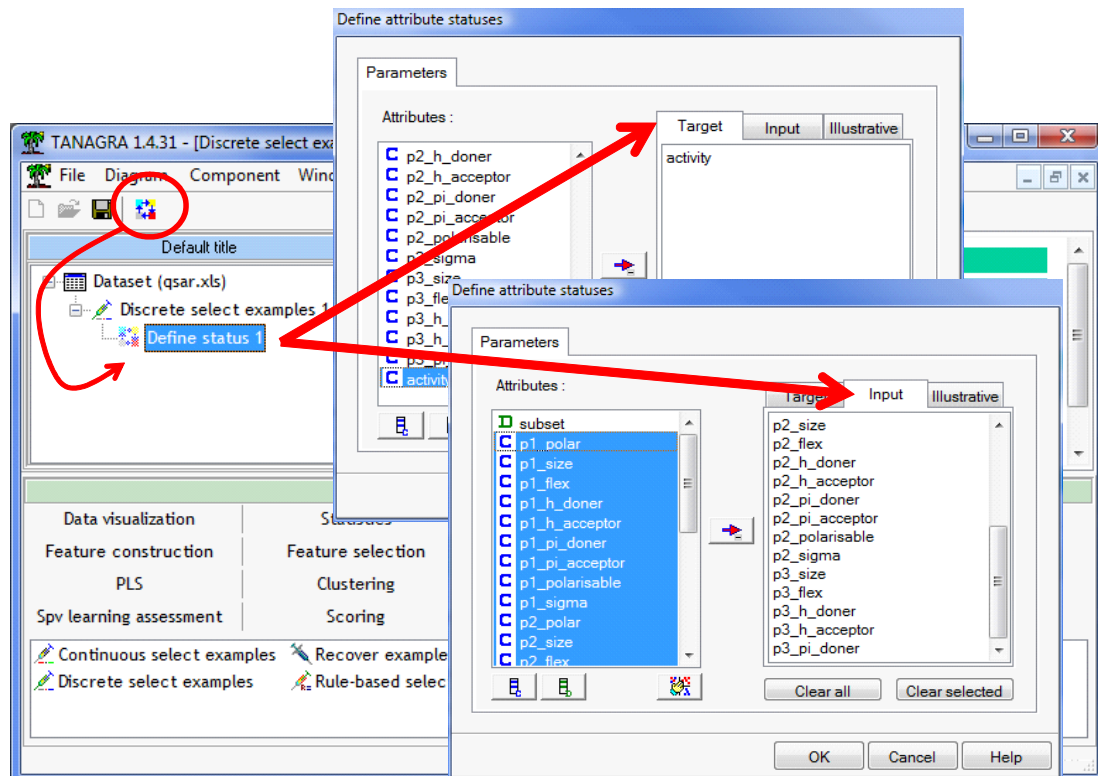


We validate and we click on the VIEW menu. We see that 30 examples are assigned to the learning phase; the others (44 examples) will be used for the assessment of models.



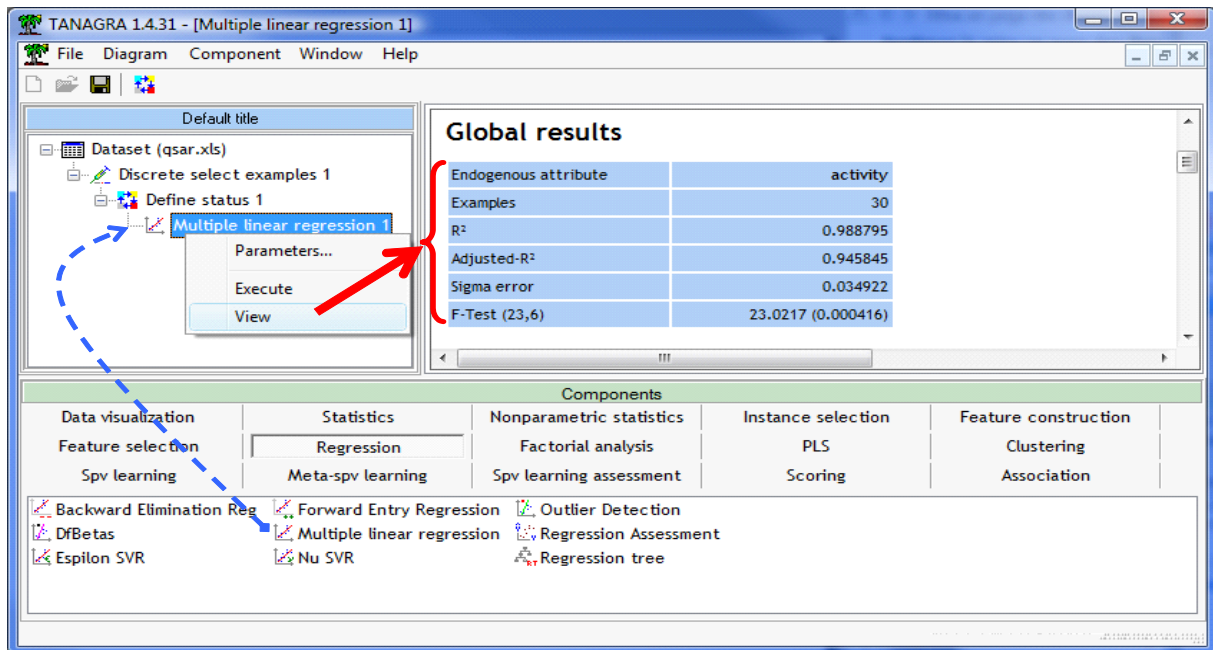
3.3 Dependent variable and independent variables

The DEFINE STATUS component allows to specify the types of variables. We set ACTIVITY as TARGET; the others as INPUT. The SUBSET column is not used here.



3.4 Linear regression

We insert the MULTIPLE LINEAR REGRESSION component (REGRESSION tab) into the diagram. We click on the VIEW menu in order to launch the calculations.

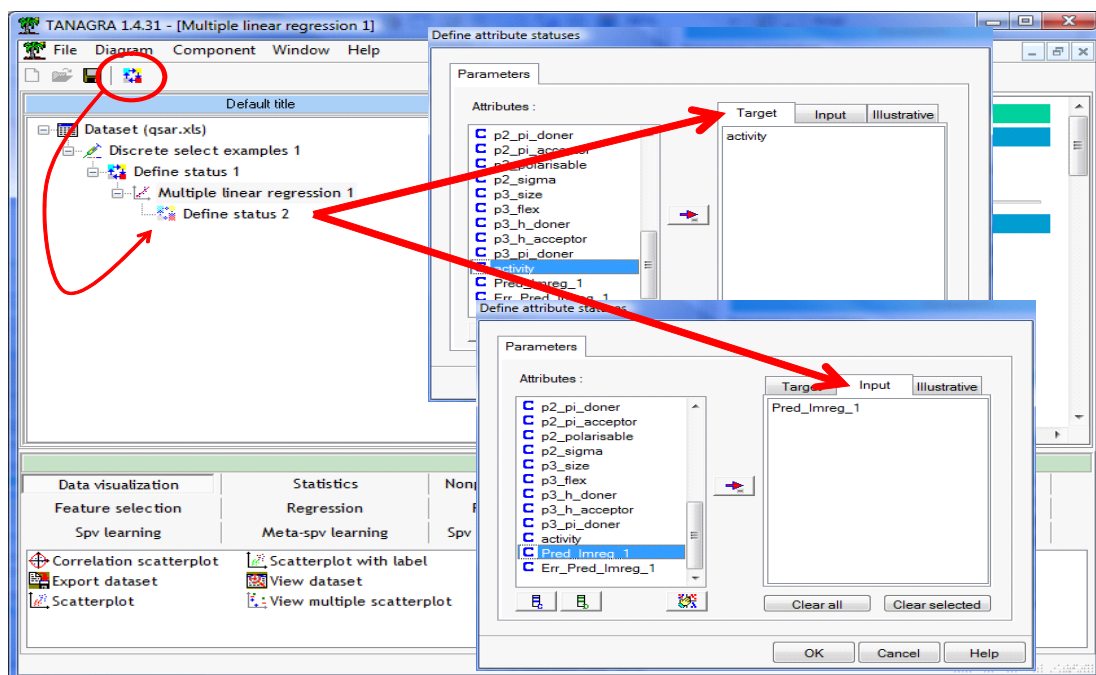


The model seems very good, the coefficient of determination is $R^2 = 98.88\%$ i.e. 98.88% of the variability of the dependent variable is explained by the model.

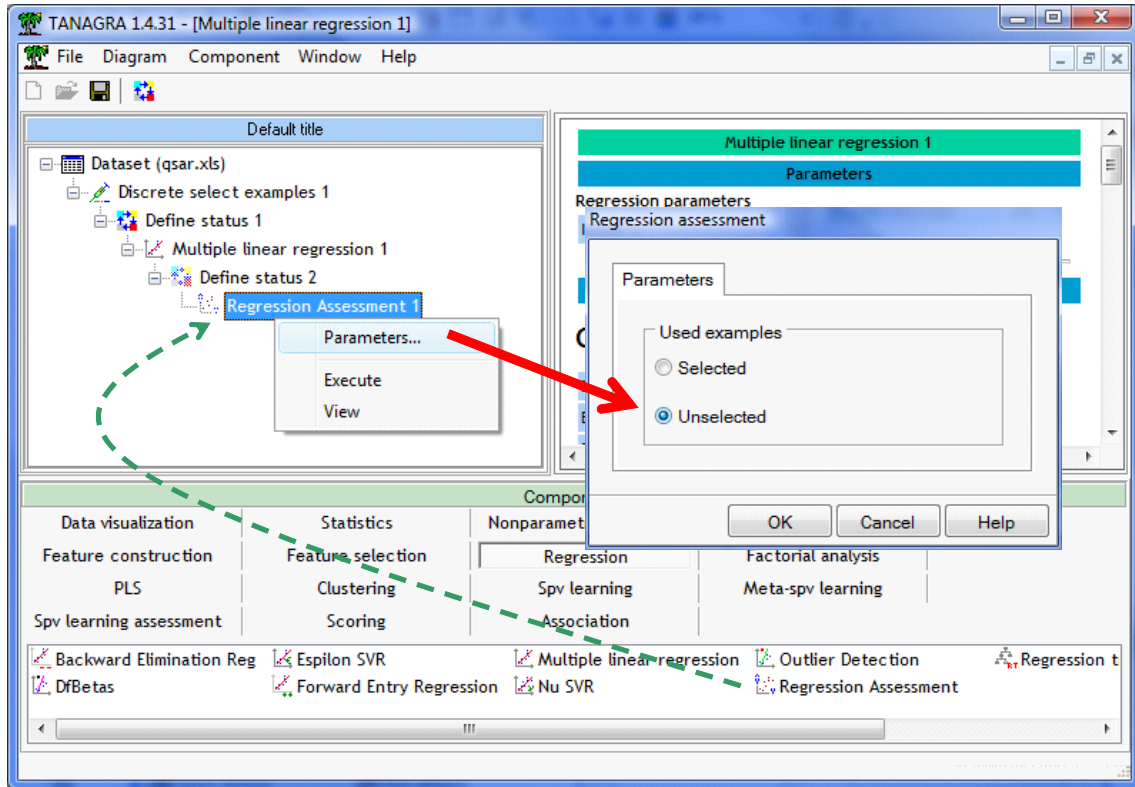
But stopping the analysis at this stage would be a mistake. The number of independent variables is high in relation to the number of examples. An overfitting can occur. It would be judicious to assess the quality of the model on a dataset which has not contributed to its construction. It is the purpose of the test sample.

3.5 Evaluation on the test sample

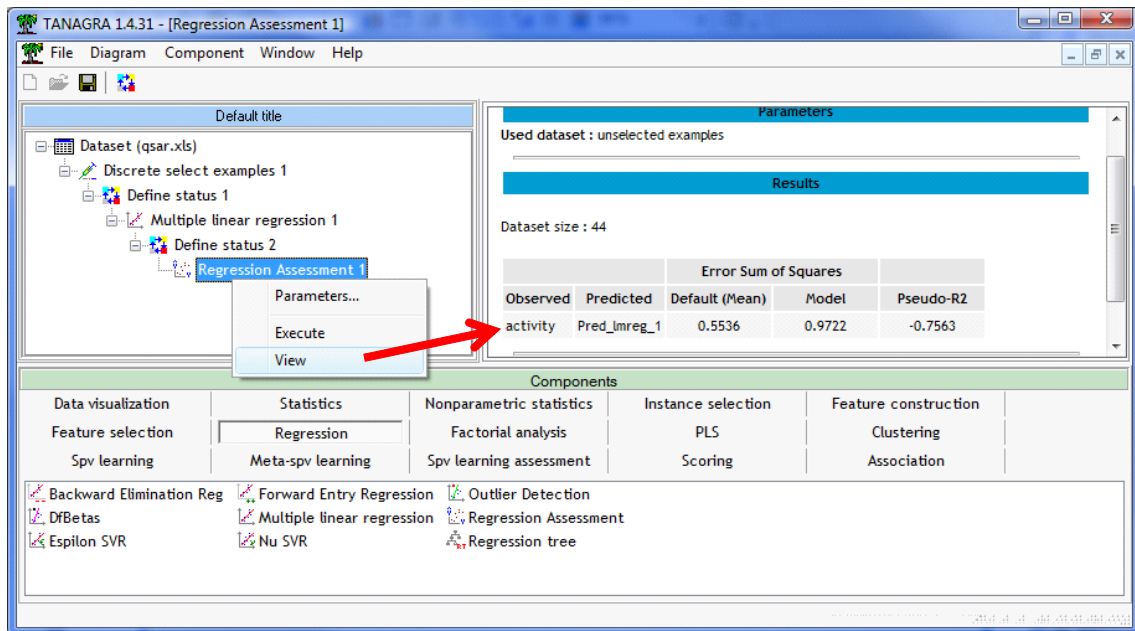
We want to compare the observed values of the dependent attribute (ACTIVITY) with the predicted values of the model on the test sample defined by SUBSET = TEST. We must specify the type of variables with the DEFINE STATUS component: we set ACTIVITY as TARGET; PRED_LMREG_1, automatically added by the regression component, as INPUT.



We can now insert the REGRESSION ASSESSMENT component. We set the parameters in order to make the comparison on the unselected examples at the beginning of the diagram i.e. the test sample.



We click on the VIEW menu. These results deserve some explanation.



The residual sum of squares of the model is computed as follows

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

The smaller is the obtained value, the better is the model. But the value in itself is not really informative. We must compare this one to a reference situation.

We call "default model" the model which does not use the information from the independent variables for the prediction of the dependent variable. According to least square principle, the default model is simply the mean of the dependent variable computed on the training sample i.e. $\hat{y}_{i, default} = \bar{y}_{train}$.

So, the residual sum of squares of the default model is

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

We can now define the pseudo-R2 criterion as follows

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

When we have a perfect model, Pseudo-R2 = 1; if our model is not better than the default model, Pseudo-R2 = 0; if our model is worse than the default model, Pseudo-R2 < 0.

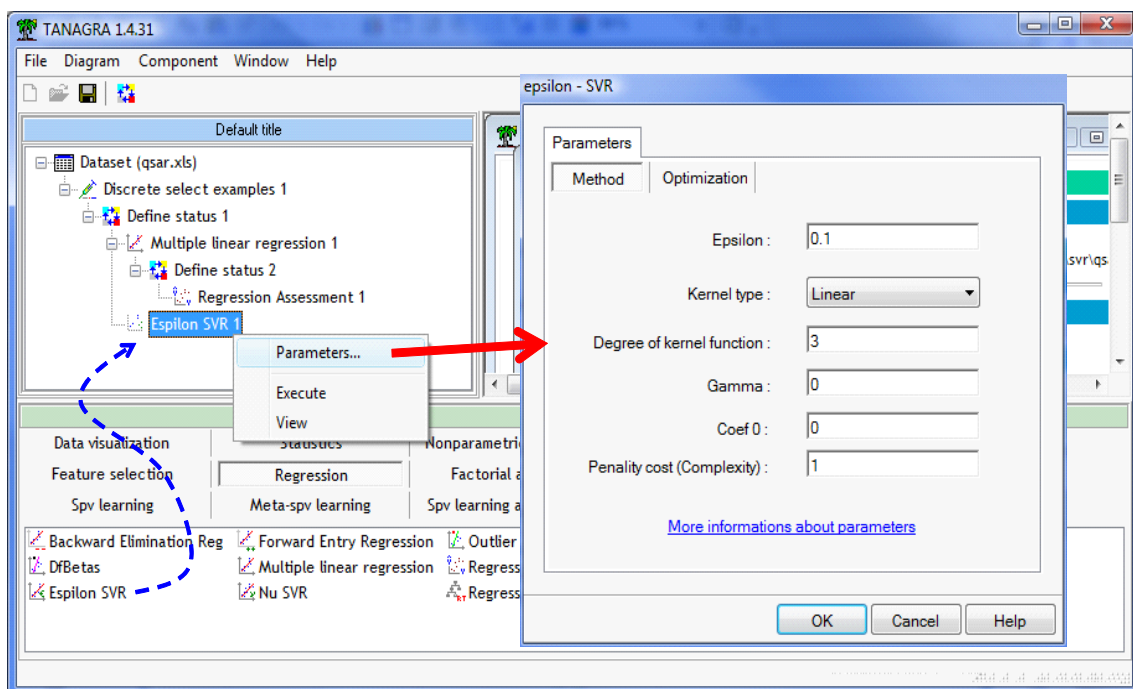
In our situation, Pseudo-R2 = 1 - 0.9722 / 0.5536 = -0.7563. Our model, which seems accurate on the train set, is in reality very bad, worse than the default model, when we measure the accuracy on the test set.

A posteriori, this result is obvious. The number of variables is high compared with the number of examples, an overfitting on the learning sample occurs during the learning process. The model is not reliable. It predicts badly on the population. We must regularize more strongly the learning process.

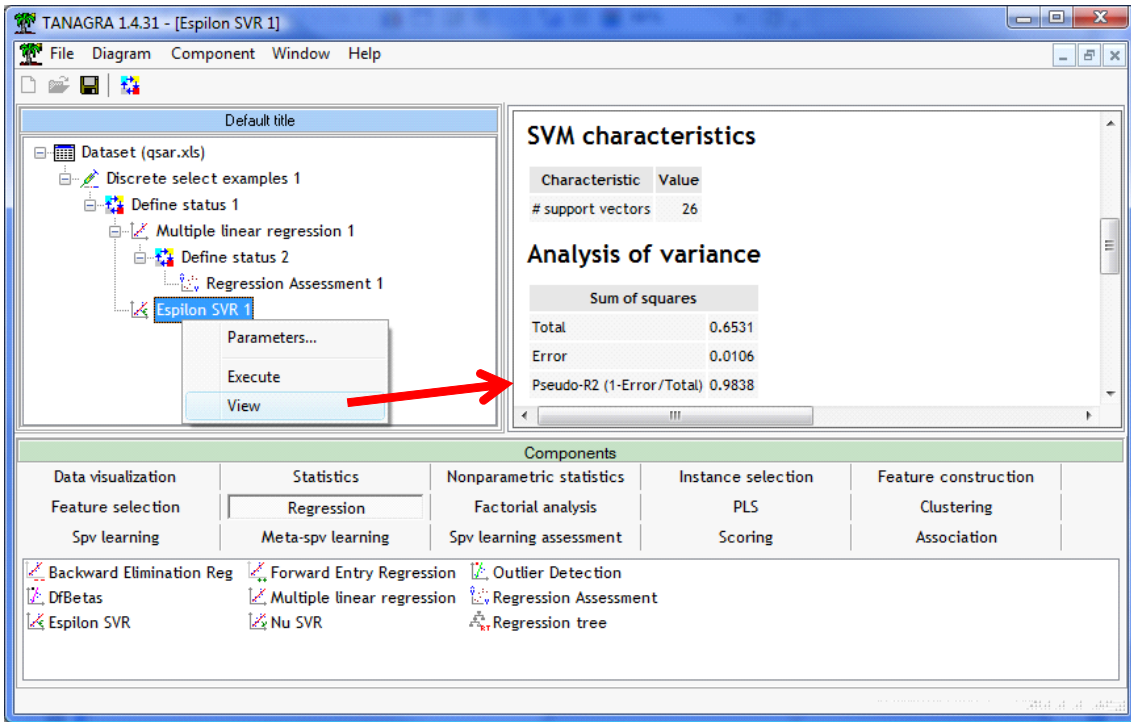
4 ϵ -SVR avec Tanagra

4.1 Training phase - Linear Kernel

We want to implement now the Epsilon-SVR component (REGRESSION tab). We insert it after the DEFINE STATUS 1 into the diagram. We click on the PARAMETERS menu. The default parameters are the following:



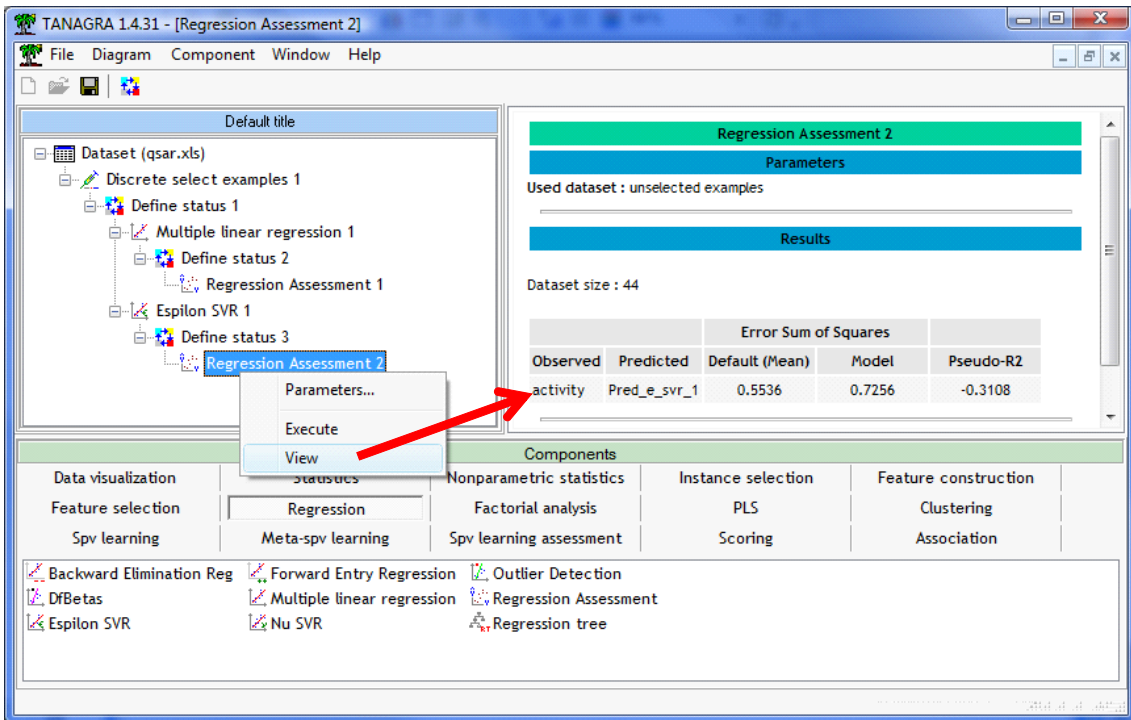
We do not modify these settings. We validate and we click on the VIEW menu. We obtain:



The number of support vectors is 26. The Pseudo-R2 on the training sample is 0.9838. The regression seems very good. But we know that this indicator, computed on the train set, is not really reliable.

4.2 Assessment

We want to use the test sample in order to evaluate the regression. We insert into the diagram: the DEFINE STATUS component (ACTIVITY as TARGET and PRED_E_SVR_1 ad INPUT) and the REGRESSION ASSESSMENT component (we use the unselected dataset i.e. the test sample).

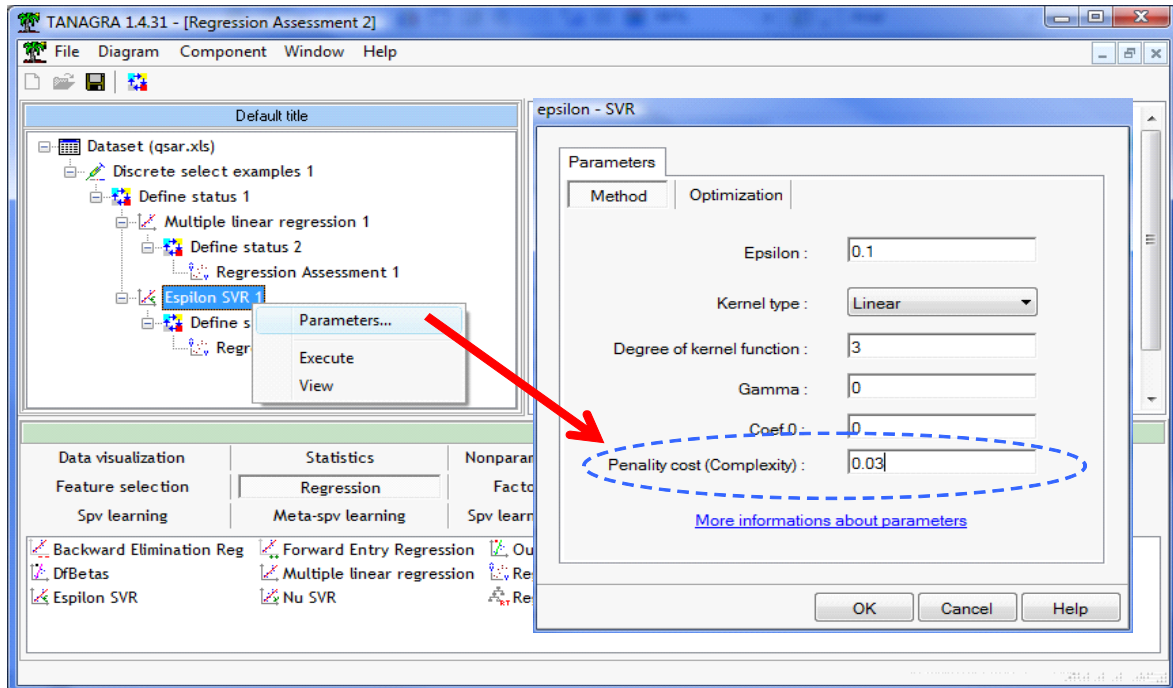


We obtain $Pseudo-R2 = 1 - 0.7256 / 0.5536 = -0.3108$. The result is as wrong as that of the linear regression. Let us see how to remedy this.

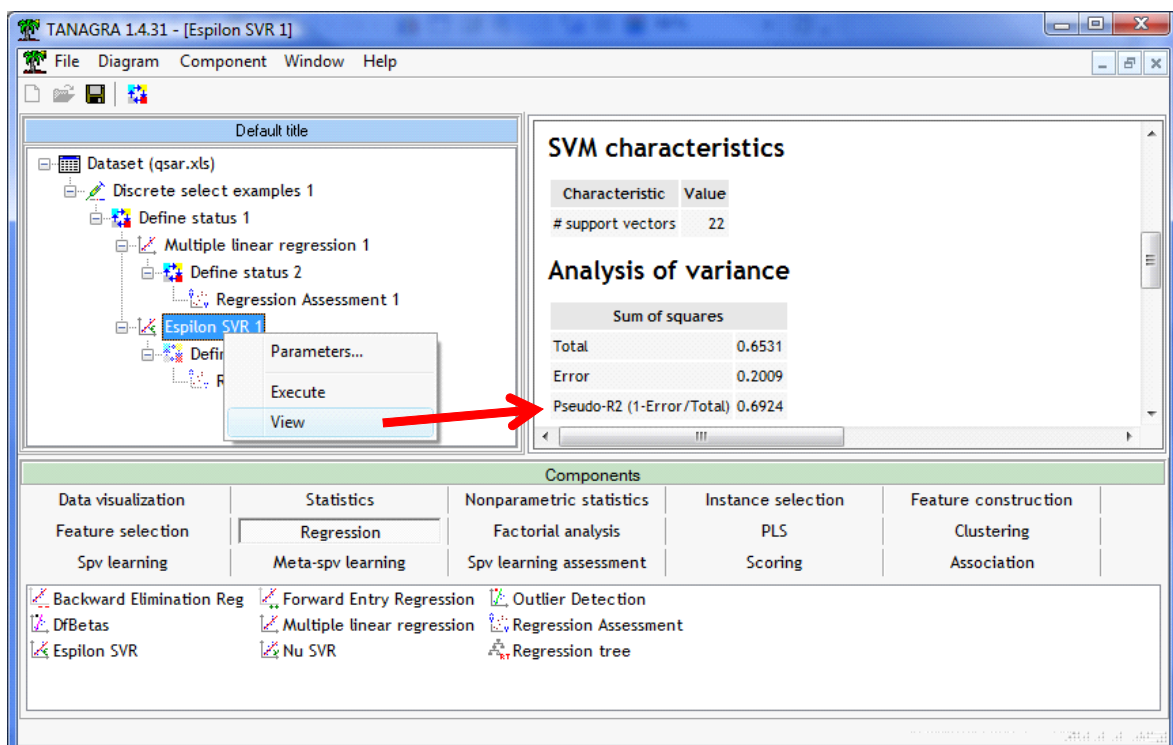
4.3 Modifying the regularization parameter (Cost $C = 0.03$)

Clearly, there is overfitting on the train sample. We must smooth the influence of the train sample. We use the regularization parameter for that i.e. the Complexity Cost parameter. It defines the cost associated to observations outside the epsilon band width around the regression line. We click on the PARAMETERS menu of the EPSILON SVR 1 component. We set Complexity Cost to 0.03.

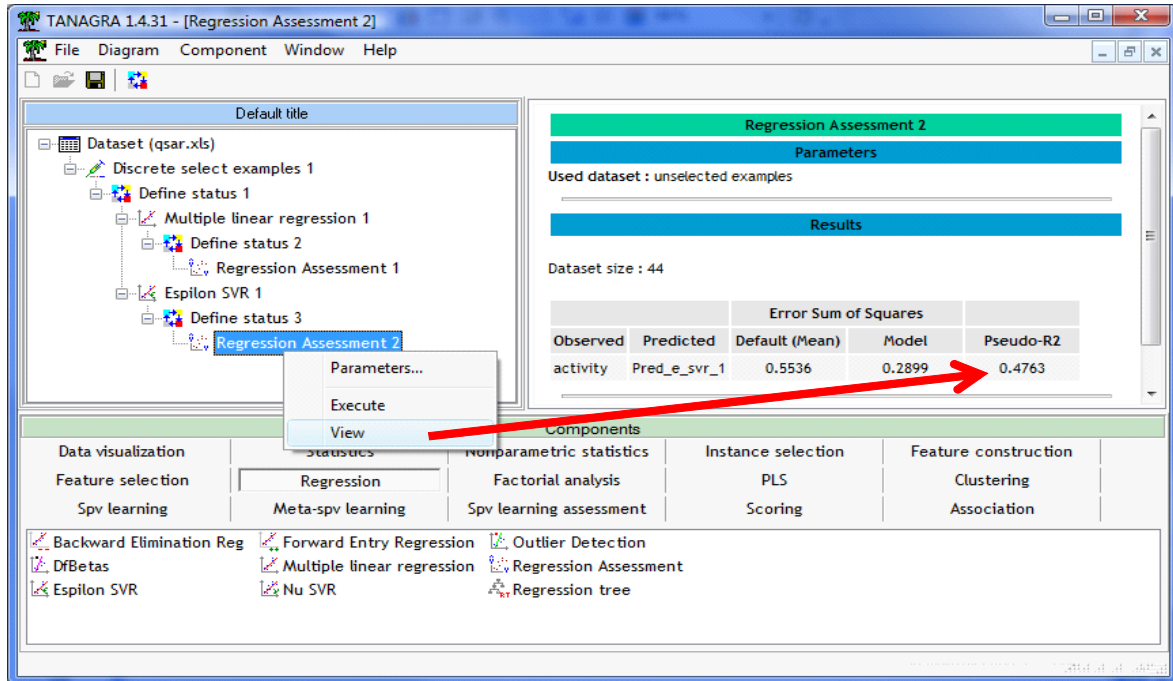
ⓘ Warning! The decimal separator may be different according to your OS.



We validate and we click on the VIEW menu. Of course, the Pseudo-R2 computed on the train sample is worse (0.6924).



But what about the test sample? We click on the VIEW menu of REGRESSION ASSESSMENT 2.

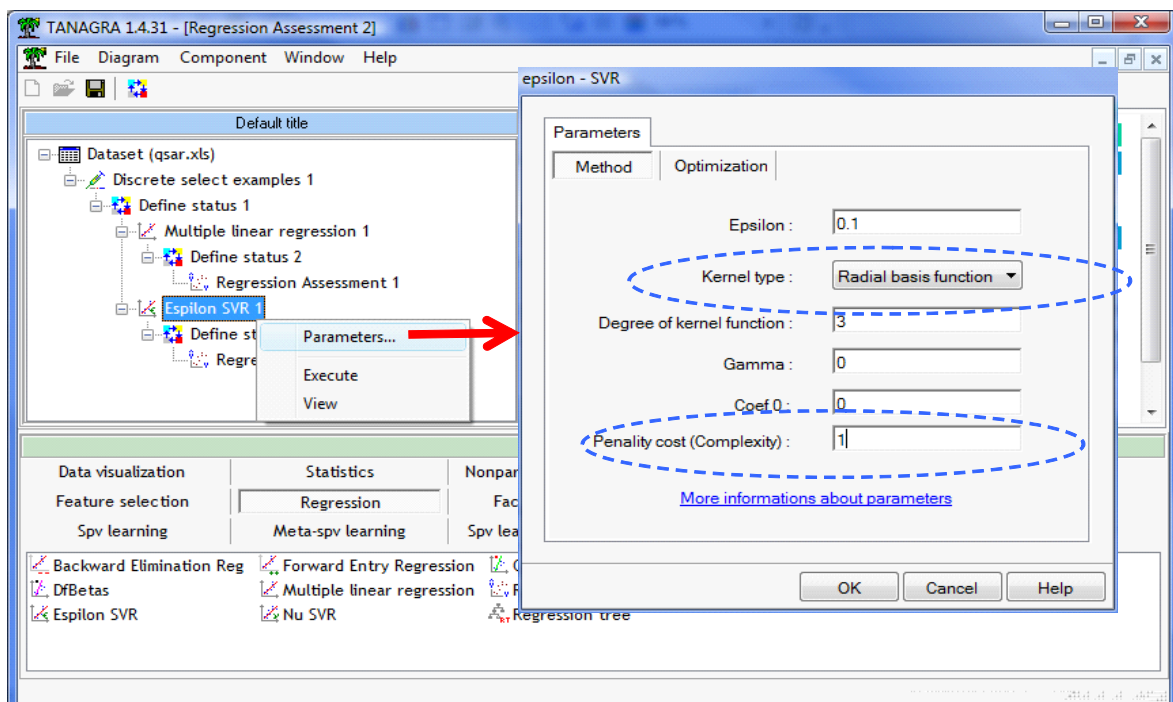


The Pseudo-R2 is now $(1 - 0.2899 / 0.5536) = 0.4763$. By being less dependent on the specificities of the training set, the model is better on the test set.

But finding the right parameter and set the right value are not really obvious. We try various values in order to find $C = 0.03$. In the next section, we show how to try a range of values with the R software.

4.4 Using a RBF kernel (with $C = 1.0$)

One of the main advantages of the SVR is the kernel trick property. We can implement a non linear model without creating explicitly new variables. We click on the PARAMETERS menu. We set new settings.



On the train set, we obtain Pseudo-R2 = 0.7361. What about the test set?

The screenshot shows the TANAGRA 1.4.31 interface. The main window displays the results of a 'Regression Assessment 2'. The 'Results' section shows a table with the following data:

Observed	Predicted	Error Sum of Squares		
		Default (Mean)	Model	Pseudo-R2
activity	Pred_e_svr_1	0.5536	0.2051	0.6295

A red arrow points to the Pseudo-R2 value of 0.6295. The interface also shows a tree view of the workflow on the left and a components palette at the bottom.

We have Pseudo-R2 = $1 - 0.2051 / 0.5536 = 0.6295$, clearly better than the linear model.

4.5 RBF kernel and C = 2.0

Again, if we set a judicious value for the regularization parameter, perhaps can we obtain better performance? By using the trial and error scheme, it seems that C = 2.0 is an “optimal” value for the complexity parameter. We click on the PARAMETERS menu and set the following modification.

The screenshot shows the TANAGRA 1.4.31 interface with the 'Epsilon SVR' parameters dialog box open. The 'Parameters' tab is selected, and the 'Optimization' section is visible. The 'Penalty cost (Complexity)' parameter is set to 2.0, which is circled in blue. The other parameters are: Epsilon: 0.1, Kernel type: Radial basis function, Degree of kernel function: 3, Gamma: 0, and Coef 0: 0. A red arrow points to the 'Parameters...' menu item in the workflow tree.

We click on VIEW menu. The Pseudo-R2 on the train set is 0.9227.

The screenshot shows the TANAGRA 1.4.31 interface. The 'Regression Assessment 2' component is selected, and its 'View' menu is open. The 'Results' section displays the following table:

Error Sum of Squares				
Observed	Predicted	Default (Mean)	Model	Pseudo-R2
activity	Pred_e_svr_1	0.5536	0.1749	0.6840

On the test sample, we obtain $\text{Pseudo-R2} = 1 - 0.1749 / 0.5536 = 0.6840$. The improvement is not very impressive. But, however, this is the best result that we obtained on our problem.

5 v-SVR with Tanagra

Nu-SVR is a variant of SVR (see <http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>).

5.1 Training phase

We insert the Nu-SVR component into the diagram. We click on the VIEW menu.

The screenshot shows the TANAGRA 1.4.31 interface with the 'Nu SVR 1' component selected. The 'View' menu is open, and a red arrow points to the 'Pseudo-R2 (1-Error/Total)' value of 0.9830 in the 'Analysis of variance' table.

Characteristic	Value
# support vectors	29

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

The default kernel is linear. The Pseudo-R2 on the train set is 0.9830.

5.2 Assessment on the test sample

For the evaluation, we insert again the DEFINE STATUS component (TARGET = ACTIVITY, INPUT = PRED_NU_SVR_1) and the REGRESSION ASSESSMENT component (UNSELECTED).

The screenshot displays the TANAGRA 1.4.31 interface for a regression assessment. The main window shows a workflow diagram on the left and a results panel on the right. The results panel, titled 'Regression Assessment 3', shows the following data:

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

A red arrow points to the Pseudo-R2 value of -0.3658. Below the results panel, there is a 'Components' section with a grid of available components:

Components		
parametric statistics	Instance selection	Feature construction
Regression	PLS	Clustering
Meta-spv learning	Scoring	Association
Factorial analysis		

At the bottom of the interface, there is a list of components that can be added to the workflow, including 'Backward Elimination Reg', 'Forward Entry Regression', 'Outlier Detection', 'DfBetas', 'Multiple linear regression', 'Regression Assessment', 'Epsilon SVR', 'Nu SVR', and 'Regression tree'.

The Pseudo-R2 on the test set is $(1 - 0.7560 / 0.5536) = -0.3658$.

Here again, we can improve dramatically the performance of the model by defining the appropriate settings.

6 SVR with the package e1071 of the R software

The LIBSVM library can be also implemented from R (<http://www.r-project.org/>) using the package **e1071** (<http://cran.r-project.org/web/packages/e1071/index.html>). We try to make again the same analysis as above with the same train-test subdivision of the dataset. Perhaps, the results can be slightly different because the calculations rely on a heuristic. But, we would have to obtain comparable performances.

6.1 Loading the data file and subdivision of the dataset

First, we import the file using the **xlsReadWrite** package. The column SUBSET allows to subdivide the dataset.

```
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)]
```

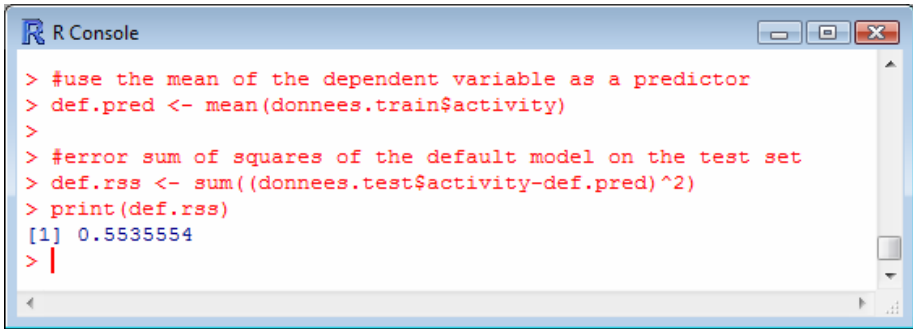
6.2 Computing the default model

We compute the default model i.e. the predicted value of the model is always the mean computed on the train set. We measure the accuracy on the test set.

```
#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)
```

We obtain $RSS_{\text{default}} = \sum_i (y_i - \hat{y}_{i,\text{default}})^2 = 0.5536$, the same value as 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 Linear regression

We perform a linear regression on the train set. Then, we compute the RSS and the Pseudo-R2 on the test set.

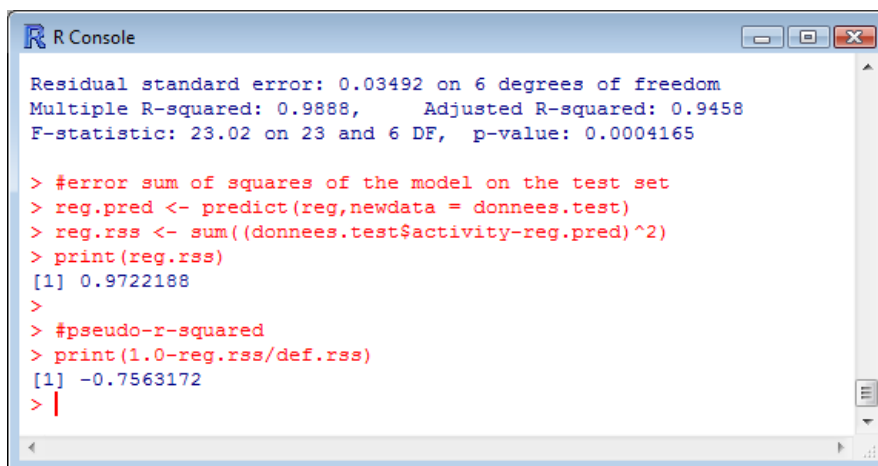
```

#####
#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)

```

While the regression seems excellent on the train set ($R^2 = 0.9888$), we note that the accuracy is poor on the test set (Pseudo- $R^2 = -0.7563$). This approach is not very suitable on our problem. The results are exactly the same as 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 ϵ -SVR with a linear kernel

Cost = 1.0. We want to implement the Epsilon-SVR approach. We call the `svm` procedure from the `e1071` package. We set the same settings as for the previous analysis with Tanagra i.e. `COST = 1.0` and linear kernel. Then, we apply the model on the test set.

```

#####
#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 supplies the following results:

```

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
> |

```

The Pseudo-R2 (-0.3110) is very similar to this the one of Tanagra. Clearly, our settings for this problem are not appropriate. We modify the value of the regularization parameter.

Cost = 0.03. We set COST = 0.03. We observe the consequence on the accuracy of the model.

```

#*****
#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)

```

The improvement is always spectacular, the 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 Detecting the suitable value of the regularization parameter (1) – Linear Kernel

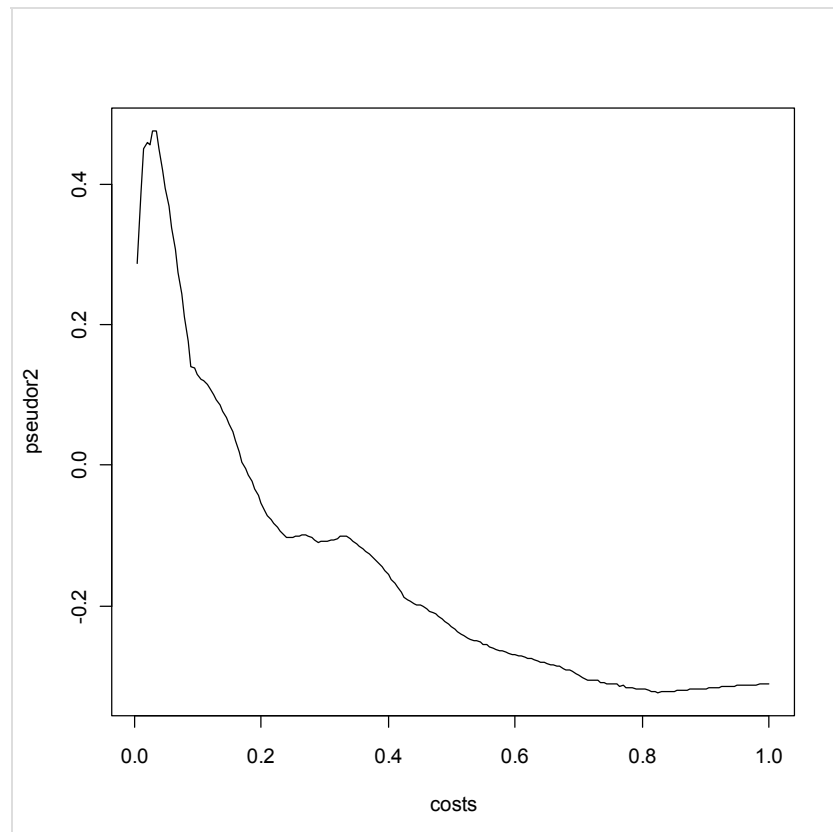
One of the main advantages of R is that we can implement complex processes using the internal programming language. We must know the syntax of this programming language but it is not really difficult.

For our problem, we want to evaluate the effect of the regularization parameter on the accuracy. This last one is measured on the test set. The program is the following.

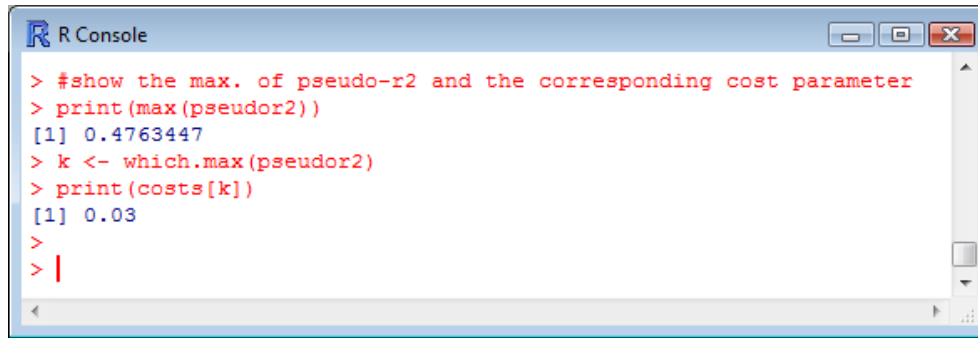
```
#####
#detect the "best" cost parameter
#####
costs <- seq(from=0.005,to=1.0,by=0.005)
pseudor2 <- double(length(costs))
for (c in 1:length(costs)){
  epsilon.svr <- svm(activity ~.,data = donnees.train, scale = T, type = "eps-regression",
                    kernel = "linear", cost = costs[c], epsilon=0.1,tolerance=0.001,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

In order to make easier the analysis of the results, we summarize the values in a graphical representation.



The “optimal” value seems to be 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
>
> |

```

We note that COST = 1.0 is particularly inappropriate for our problem.

6.6 Finding the suitable value of the regularization parameter (2) – Linear kernel

The library includes a very interesting functionality. There is a predefined function which can try to detect, using a cross validation (resampling approach), the appropriate values of the COST parameter (in fact the best combination of cost and epsilon). We do not need a separate test set in this context. It can be useful when we have a small dataset.

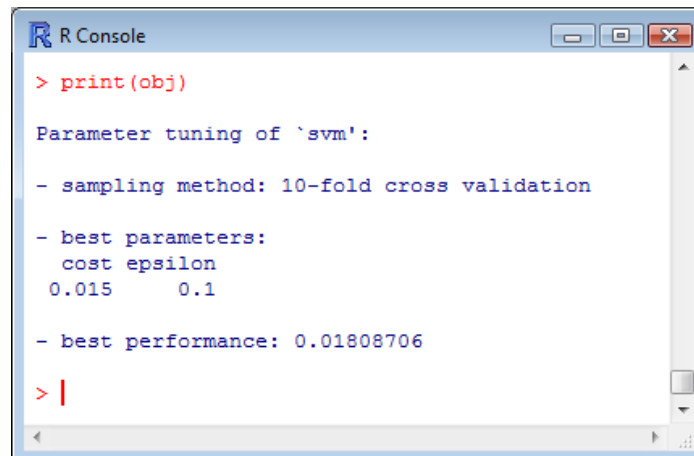
In our experiment, we control only the COST parameter.

```

#####
#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")

```

The optimal value (COST = 0.015) is different compared with this one obtained on the separate test set.



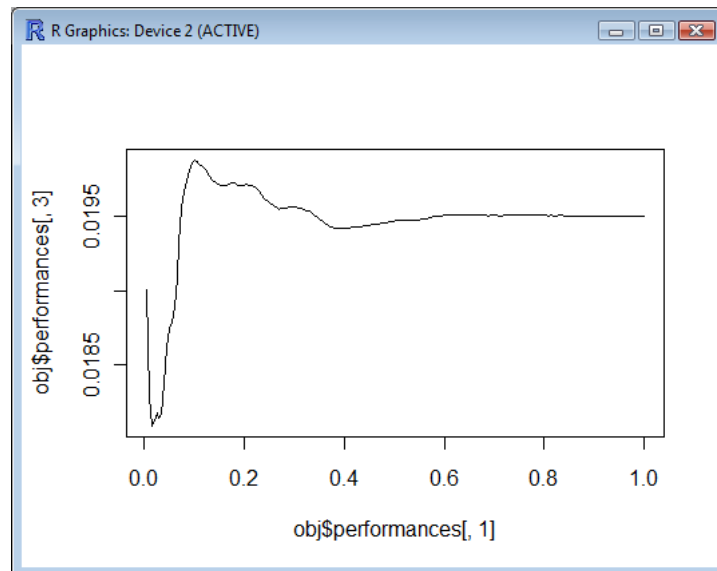
```

R Console
> print(obj)
Parameter tuning of `svm':
- sampling method: 10-fold cross validation
- best parameters:
  cost epsilon
  0.015      0.1
- best performance: 0.01808706
> |

```

We can explain this difference by the small size of the dataset, the results are very unstable. We note however that the area of the optimal value is the same one.

Note: The computed performance indicator is the RMSE (root mean squared error) for R; the lower is the value, the better is the result.



6.7 RBF kernel for ϵ -SVR

As with Tanagra, we try the RBF kernel.

```
#####
#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)
```

We obtain.

```
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
> |
```

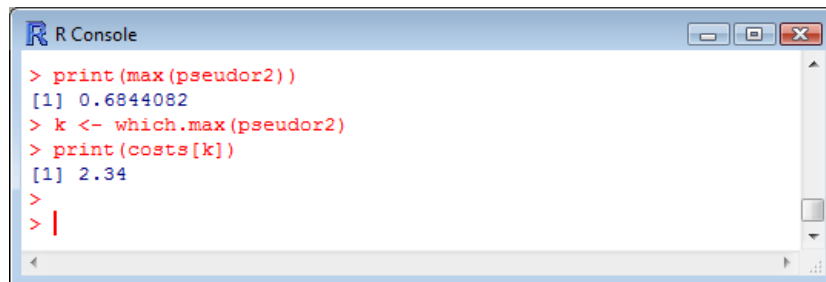
6.8 Finding the best value for the regularization parameter – RBF kernel

Here also, we program a simple procedure in order to detect the appropriate value for the regularization parameter. We use only the strategy based on the separate test set.

```
#####
#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)

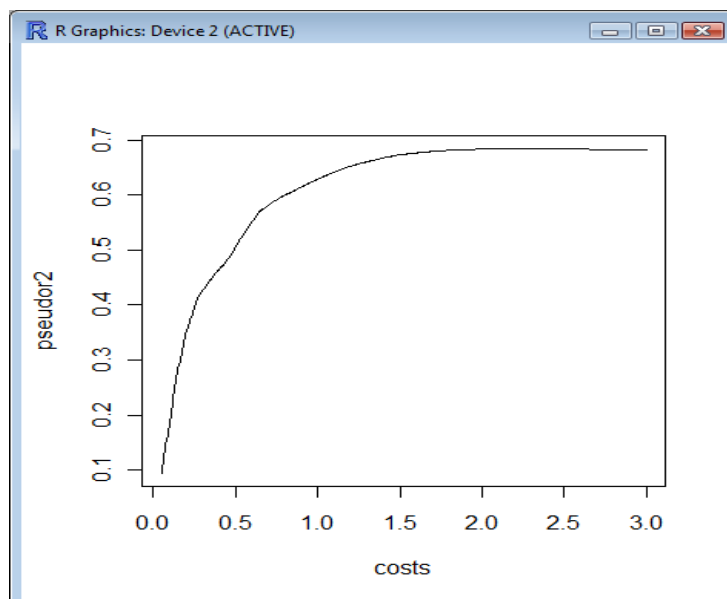
  #prédiction
  esvr.pred <- predict(epsilon.svr,newdata = donnees.test)
  esvr.rss <- sum((donnees.test$activity-esvr.pred)^2)
  pseudor2[c] <- 1.0-esvr.rss/def.rss
}
#graphical representation
plot(costs,pseudor2,type="l")
#show the max. of pseudo-r2 and the corresponding cost parameter
print(max(pseudor2))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

The optimal value is COST = 2.34.



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

The shape of the performance curve is very different compared to the previous one. We must increase the value of COST for this kind of kernel for our problem.

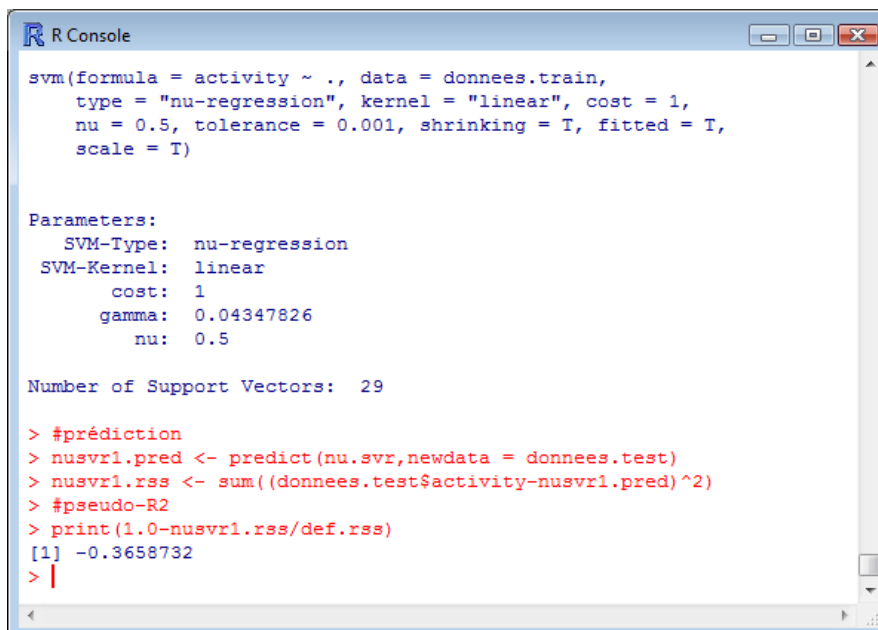


6.9 v-SVR with R

We can also implement the NU-SVR approach with R.

```
#####
#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)
```

The Pseudo-R2 computed on the test sample is -0.3658.



```
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

In the exploratory data analysis domain, we must answer 3 questions for a successful modeling process: (1) specifying the suitable family of methods to the problem to be solved (here, regression); (2) selecting the most effective technique given the characteristics of data (among the regression approaches, choosing the SVR); (3) defining the appropriate values of the parameters.

This last point is not the easiest. It takes a good knowledge of techniques; it should be put in relation with the characteristics of data. Often, we must use an experimental approach to find the "optimal" solution. It should be as rigorous as possible.

In this tutorial, we showed how to implement the SVR (Support Vector Regression) with Tanagra and R. We are attached to skillfully guide the regularization parameter after defining an evaluation criterion computed on a test sample.