

# Data Mining with Neural Networks and Support Vector Machines using the R/rminer Tool\*

Paulo Cortez

Department of Information Systems/R&D Centre Algoritmi,  
University of Minho, 4800-058 Guimarães, Portugal,  
pcortez@dsi.uminho.pt, WWW home page: <http://www3.dsi.uminho.pt/pcortez>

**Abstract.** We present rminer, our open source library for the R tool that facilitates the use of data mining (DM) algorithms, such as neural Networks (NNs) and support vector machines (SVMs), in classification and regression tasks. Tutorial examples with real-world problems (i.e. satellite image analysis and prediction of car prices) were used to demonstrate the rminer capabilities and NN/SVM advantages. Additional experiments were also held to test the rminer predictive capabilities, revealing competitive performances.

**Keywords:** Classification, Regression, Sensitivity Analysis, Neural Networks, Support Vector Machines.

## 1 Introduction

The fields of data mining (DM)/business intelligence (BI) arose due to the advances of information technology (IT), leading to an exponential growth of business and scientific databases. The aim of DM/BI is to analyze raw data and extract high-level knowledge for the domain user or decision-maker [16].

Due to its importance, there is a wide range of commercial and free DM/BI tools [7]. The R environment [12] is an open source, multiple platform (e.g. *Windows, Linux, Mac OS*) and high-level matrix programming language for statistical and data analysis. Although not specifically oriented for DM/BI, the R tool includes a high variety of DM algorithms and it is currently used by a large number of DM/BI analysts. For example, the 2008 DM survey [13] reported an increase in the R usage, with 36% of the responses [13]. Also, the 2009 KDnuggets pool, regarding DM tools used for a real project, ranked R as the second most used open source tool and sixth one overall [10]. When compared with commercial tools (e.g. offered by SAS: <http://www.sas.com/technologies/bi/>) or even open source environments (e.g. WEKA [18]), R presents the advantage of being more flexible and extensible by design, thus integration of statistics, programming and graphics is more natural. Also, due to its open source availability and users' activity, novel DM methods are in general more quickly encoded into R than into commercial tools. The R community is very active and new packages are being continuously created, with more than 2321 packages available

---

\* This work is supported by FCT grant PTDC/EIA/64541/2006.

at <http://www.r-project.org/>. Thus, R can be viewed as worldwide gateway for sharing computational algorithms.

DM software suites often present friendly graphical user interfaces (GUI). In contrast, the most common usage of R is under a console command interface, which may require a higher learning curve from the user. Yet, after mastering the R environment, the user achieves a better control (e.g. adaptation to a specific application) and understanding of what is being executed (in contrast with several “black-box” DM GUI products). Nevertheless, for those interested in graphical DM suites for R, there is the Rattle tool [17].

In this work, we present our `rminer` library, which is an integrated framework that uses a console based approach and that facilitates the use of DM algorithms in R. In particular, it addresses two important and common goals [16]:

**classification** – labeling a data item into one of several predefined classes; and  
**regression** – estimate a real-value (the dependent variable) from several (independent) input attributes.

While several DM algorithms are available for these tasks, the library is particularly suited for using neural networks (NNs) and support vector machines (SVMs). Both are flexible models that can cope with complex nonlinear mappings, potentially leading to more accurate predictions [8]. Also, it is possible to extract knowledge from NNs and SVMs, given in terms of input relevance [4]. When compared to Rattle, `rminer` can be viewed as a lightweight command based alternative, since it is easier to install and requires much less R packages. Moreover, `rminer` presents more NN and SVM capabilities (e.g. in Rattle version 2.5.26, SVM cannot be used for regression tasks). While adopting R packages for the DM algorithms, `rminer` provides new features:

- i) it simplifies the use of DM algorithms (e.g. NNs and SVMs) in classification and regression tasks by presenting a short and coherent set of functions (as shown in Section 3.1);
- ii) it performs an automatic model selection (i.e. tuning of NN/SVM);
- iii) it computes several classification/regression metrics and graphics, including the sensitivity analysis procedure for input relevance extraction.

The `rminer/R` tool has been used by both IT and non-IT specialists (e.g. managers, biologists or civil engineers), with applications in distinct domains, such as civil engineering [15], wine quality [4] or spam email detection [5]. In this paper, we address several real-world problems from the UCI repository [1] to show the `rminer` capabilities.

## 2 Data Mining

DM is an iterative process that consists of several steps. The CRISP-DM [2], a tool-neutral methodology supported by the industry (e.g. SPSS, DaimlerChrysler), partitions a DM project into 6 phases (Fig. 1): 1 - business understanding;

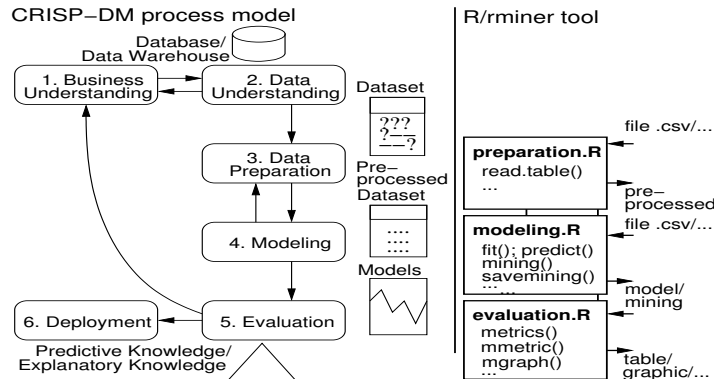


Fig. 1. The CRISP-DM and proposed R/rminer tool use

2 - data understanding; 3 - data preparation; 4 - modeling; 5 - evaluation; and 6 - deployment.

This work addresses steps 4 and 5, with an emphasis on the use of NNs and SVMs to solve classification and regression goals. Both tasks require a supervised learning, where a model is adjusted to a dataset of examples that map  $I$  inputs into a given target. The rminer models output a probability  $p(c)$  for each possible class  $c$ , such that  $\sum_{c=1}^{N_c} p(c) = 1$  (if classification) or a numeric value (for regression). For assigning a target class  $c$ , one option is to set a decision threshold  $D \in [0, 1]$  and then output  $c$  if  $p(c) > D$ , otherwise return  $\neg c$ . This method is used to build the receiver operating characteristic (ROC) curves. Another option is to output the class with the highest probability and this method allows the definition of a multi-class confusion matrix.

To evaluate a model, common metrics are [18]: ROC area (AUC), confusion matrix, accuracy (ACC), true positive/negative rates (TPR/TNR), for classification; and mean absolute deviation (MAD), relative absolute error (RAE), root mean squared (RMSE), root relative squared error (RRSE) and regression error characteristic (REC) curve, for regression. A classifier should present high values of ACC, TPR, TNR and AUC, while a regressor should present low predictive errors and an high REC area. The model's generalization performance is often estimated by the holdout validation (i.e. train/test split) or the more robust  $k$ -fold cross-validation [8]. The latter is more robust but requires around  $k$  times more computation, since  $k$  models are fitted.

Before fitting the DM models, the data needs to be preprocessed. This includes operations such as selecting the data (e.g. attributes or examples) or dealing with missing values. Since functional models (e.g. NN or SVM) only deal with numeric values, discrete variables need to be transformed. In R/rminer, the nominal attributes (with  $N_c = 3$  or more non-ordered values) are encoded with the common 1-of- $N_c$  transform, leading to  $N_c$  binary variables. Also, all attributes are standardized to a zero mean and one standard deviation [8].

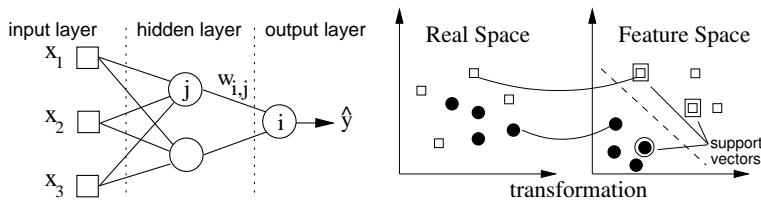
For NN, we adopt the popular multilayer perceptron, as coded in the **Rnnet** package. This network includes one hidden layer of  $H$  neurons with logistic functions (Fig 2). The overall model is given in the form:

$$y_i = f_i(w_{i,0} + \sum_{j=I+1}^{I+H} f_j(\sum_{n=1}^I x_n w_{m,n} + w_{m,0})w_{i,n}) \quad (1)$$

where  $y_i$  is the output of the network for node  $i$ ,  $w_{i,j}$  is the weight of the connection from node  $j$  to  $i$  and  $f_j$  is the activation function for node  $j$ . For a binary classification ( $N_c = 2$ ), there is one output neuron with a logistic function. Under multi-class tasks ( $N_c > 2$ ), there are  $N_c$  linear output neurons and the softmax function is used to transform these outputs into class probabilities:

$$p(i) = \frac{\exp(y_i)}{\sum_{c=1}^{N_c} \exp(y_c)} \quad (2)$$

where  $p(i)$  is the predicted probability and  $y_i$  is the NN output for class  $i$ . In regression, the output neuron uses a linear function. The training (BFGS algorithm) is stopped when the error slope approaches zero or after a maximum of  $M_e$  epochs. For regression tasks, the algorithm minimizes the squared error, while for classification it maximizes the likelihood [8]. Since NN training is not optimal, the final solution is dependent of the choice of starting weights. To solve this issue, the solution adopted is to train  $N_r$  different networks and then select the NN with the lowest error or use an ensemble of all NNs and output the average of the individual predictions [8]. In **rminer**, the former option is set using `model="mlp"`, while the latter is called using `model="mlpe"`. In general, ensembles are better than individual learners [14]. The final NN performance depends crucially on the number of hidden nodes. The simplest NN has  $H = 0$ , while more complex NNs use a high  $H$  value.



**Fig. 2.** Example of a multilayer perceptron (left) and SVM transformation (right)

When compared with NNs, SVMs present theoretical advantages, such as the absence of local minima in the learning phase [8]. The basic idea is transform the input  $\mathbf{x} \in \mathbb{R}^I$  into a high  $m$ -dimensional feature space by using a nonlinear mapping. Then, the SVM finds the best linear separating hyperplane, related to a set of support vector points, in the feature space (Fig. 2). The transformation ( $\phi(\mathbf{x})$ ) depends of a kernel function. In **rminer**, we use the **kernelab** package,

which uses the sequential minimal optimization (SMO) learning algorithm. We also adopt the popular gaussian kernel, which presents less parameters than other kernels (e.g. polynomial):  $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$ ,  $\gamma > 0$ .

The classification performance is affected by two hyperparameters:  $\gamma$ , the parameter of the kernel, and  $C$ , a penalty parameter. The probabilistic SVM output is given by [19]:

$$\begin{aligned} f(\mathbf{x}_i) &= \sum_{j=1}^m y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) + b \\ p(i) &= 1/(1 + \exp(Af(\mathbf{x}_i) + B)) \end{aligned} \quad (3)$$

where  $m$  is the number of support vectors,  $y_i \in \{-1, 1\}$  is the output for a binary classification,  $b$  and  $\alpha_j$  are coefficients of the model, and  $A$  and  $B$  are determined by solving a regularized maximum likelihood problem. When  $N_c > 2$ , the one-against-one approach is used, which trains  $N_c(N_c - 1)/2$  binary classifiers and the output is given by a pairwise coupling [19]. For regression there is an additional hyperparameter  $\epsilon$ , used to set an  $\epsilon$ -insensitive tube around the residuals, being the tiny errors within this tube discarded. The SVM algorithm finds the best linear separating hyperplane:

$$y_j = w_0 + \sum_{i=1}^m w_i \phi_i(x) \quad (4)$$

Since the search space for these parameters is high, we adopt by default the heuristics [3]:  $C = 3$  (for a standardized output) and  $\epsilon = 3\sigma_y \sqrt{\log(N)/N}$ , where  $\sigma_y$  denotes the standard deviation of the predictions of given by a 3-nearest neighbor and  $N$  is the dataset size.

In `rminer`, the NN and SVM hyperparameters (e.g.  $H$ ,  $\gamma$ ) are optimized using a grid search. To avoid overfitting, the training data is further divided into training and validation sets (holdout) or an internal  $k$ -fold is used. After selecting the best parameter, the model is retrained with all training data.

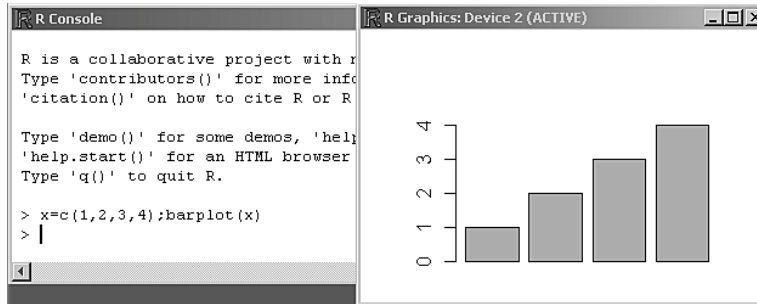
The sensitivity analysis is a simple procedure that is applied after the training procedure and analyzes the model responses when a given input is changed. Let  $y_{a,j}$  denote the output obtained by holding all input variables at their average values except  $x_a$ , which varies through its entire range ( $x_{a,j}$ , with  $j \in \{1, \dots, L\}$  levels). We use the variance ( $V_a$ ) of  $y_{a,j}$  as a measure of input relevance [9]. If  $N_c > 2$  (multi-class), we set it as the sum of the variances for each output class probability ( $p(c)_{a,j}$ ). A high variance ( $V_a$ ) suggests a high  $x_a$  relevance, thus the input relative importance ( $R_a$ ) is given by  $R_a = V_a / \sum_{i=1}^I V_i \times 100$  (%). For a more detailed analysis, we propose the variable effect characteristic (VEC) curve [6], which plots the  $x_{a,j}$  values ( $x$ -axis) versus the  $y_{a,j}$  predictions ( $y$ -axis).

### 3 Data Mining using R/`rminer`

#### 3.1 The R/`rminer` Tool

R works under a console interface (Fig. 3). Commands are typed after the prompt (`>`). An extensive help system is included (`help.start()` calls the full tutorial

in an HTML browser). R instructions can be separated using the ; or newline character. Everything that appears after the # character in a line is a comment. R commands can be edited in a file<sup>1</sup> and loaded with the `source` command.



**Fig. 3.** Example of the R tool in *Windows*

Data is stored in objects and the = operator can be used to assign an object to a variable. Atomic objects include the **character** (e.g. "day") and **numeric** (e.g. 0.2) types. There are also several containers, such as: **vector**, **factor**, **matrix**, **data.frame** and **list**. **Vectors** and **matrices** are indexed objects of atoms. A **factor** is a special vector that contains discrete values. A **data.frame** is a special matrix where the columns (vectors or factors) have names. Finally, a **list** is a collection of distinct objects (called components). Since R uses an object oriented language, there are important functions that can be applied to any of these objects (e.g. `summary` or `plot`).

Our `rminer` library<sup>2</sup> project started in 2006. All code is written in R and only a few packages need to be installed (e.g. `kernlab`). In this work, the `rminer` functions are underlined. The main functions are: fit – create and adjust a given DM model using a dataset (i.e. **data.frame**); predict – returns the predictions for new data; mining – a powerful function that trains and tests a particular model under several runs; mgraph, metrics and mmetric– which return several mining graphs (e.g. ROC) or metrics (e.g. ACC). All experiments were tested in *Windows*, *Linux* and *Mac OS*. The results reported here were conducted within a *Mac OS* Intel Core 2 Duo processor.

### 3.2 Classification Example

The satellite data was generated using **Landsat** multi-spectral images. The aim is to classify a tiny scene based on 36 numeric features that correspond to pixels from four spectral bands. In the original data, a numeric value was given to the

<sup>1</sup> In *Windows*, the Tinn-R editor can be used: <http://www.sciviews.org/Tinn-R/>

<sup>2</sup> Available at: <http://www3.dsi.uminho.pt/pcortez/rminer.html>

output variable (V37). Also, the training and test sets are already divided into two files: `sat.trn` (with 4435 samples) and `sat.tst` (2000 cases).

We propose that a DM process should be divided into 3 blocks (or files), with the CRISP-DM steps 3 to 5 (Fig. 1): preparation, modeling and evaluation (we only address the last two here). By separating the computation and generating intermediate outcomes, it is possible to later rerun only one of these steps (e.g. analyze a different metric), thus saving time. Our satellite modeling code is:

```
library(rminer) # load the library
# read the training and test sets:
tr=read.table("sat.trn",sep=" "); ts=read.table("sat.tst",sep=" ")
tr$V37=factor(tr$V37); ts$V37=factor(ts$V37) # convert output to factor
DT=fit(V37~.,tr,model="dt") # fit a Decision Tree with tr
NN=fit(V37~.,tr,model="mlp",search=10) # fit a NN with H=10
SV=fit(V37~.,tr,model="svm",search=2^c(-5,-3)) # fit the SVM
print(DT); print(NN); print(SV) # show and save the trained DM models:
savemodel(DT,"sat.dt"); savemodel(NN,"sat.nn"); savemodel(SV,"sat.sv")
# get the predictions:
PDT=predict(DT,ts); PNN=predict(NN,ts); PSV=predict(SV,ts)
P=data.frame(ts=ts$V37,dt=PDT,nn=PNN,svm=PSV) # create a data.frame
write.table(P,"sat.res",row.names=FALSE) # save output and predictions
```

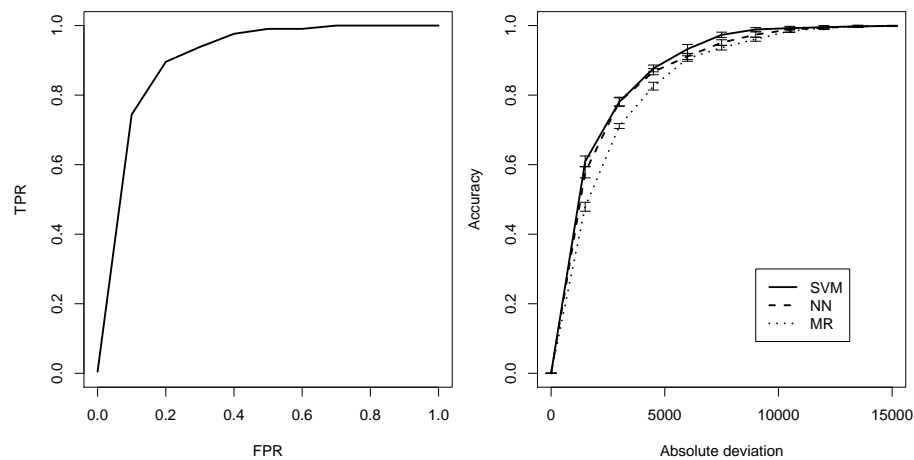
The `read.table` and `write.table` are functions that load/save a dataset from/to a text file (e.g. ".csv")<sup>3</sup>. The `tr` and `ts` objects are **data.frames**. Since the output target (V37) is encoded with numeric values, we converted it into a factor (i.e. set of classes). While `rminer` includes several classifiers, in the example we tested only a decision tree (DT) (`model="dt"`), a NN ("`mlp`") and a SVM ("`svm`"). The first parameter of the `fit` function is a R **formula**, which defines the output (V37) to be modeled (`~`) from the inputs (`.` means all other variables). The `search` parameter controls the NN and SVM hyperparameters ( $H$  or  $\gamma$ ). When `search` contains more than one value (e.g. SVM fit), then an internal grid search is performed. By default, `search` is set to  $H = I/2$  for NN and  $\gamma = 2^{-6}$  for SVM. Additional NN/SVM parameters can be set with the optional `mpar` (see Section 3.3). In this case, the default `mpar=c(3,100,"holdout",2/3,"AUC")` (for NN,  $N_r = 3$ ,  $M_e = 100$  and internal holdout with 2/3 train and 1/3 test split, while "AUC" means use the AUC metric for model selection during the grid search) or `mpar=c(NA,NA,"holdout",2/3,"AUC")` (for SVM, use default  $C/\epsilon$  heuristics) is assumed. The result of the `fit` function is a **model** object, which contains the adjusted model (e.g. `DT@object`) and other information (e.g. hyperparameters or fitting time). The execution times (in seconds) were 1.1s for DT (stored at `DT@time`), 15.9s for NN and 25s for SVM. In case of the SVM, the best  $\gamma$  is  $2^{-3}$  (`SV@mpar`). Next, we show the evaluation code:

```
P=read.table("sat.res",header=TRUE); P$ts=factor(P$ts); # read the results
# compute the test errors:
EDT=metrics(P$ts,P[,2:7]); ESV=metrics(P$ts,P[,14:19]);
ENN1=metrics(P$ts,P[,8:13]); ENN2=metrics(P$ts,P[,8:14],D=0.7,TC=4)
# show the full test errors:
print(EDT); print(ESV); print(ENN1); print(ENN2)
```

<sup>3</sup> Further loading functions (e.g. for SPSS files) are available in the **foreign** R package.

```
mgraph(P$ts,P[,8:13],graph="ROC",PDF="roc4",TC=4) # plot the ROC
NN=loadmodel("sat.nn") # load the best model
```

The predictions for each model are matrixes, where each column denotes the  $p_c$  for a given  $c \in \{“1”, “2”, “3”, “4”, “5”, “7”\}$  (there is no “6” class). The `metrics` function receives the target and predictions (e.g. P columns 8 to 13 for NN) and returns a list with several performance measures. Under the multi-class confusion matrix, the best accuracy result is given by NN, with an  $ACC = 86\%$  (`ENN$acc`), followed by the SVM (81%) and the DT (78%). The global AUC, which weights the AUC for each class  $c$  according to the prevalence of  $c$  in the data [11], also favors the NN, with a value of 98% (`ENN1$tauc`). For the target “4” class ( $TC=4$ ) and NN, we also computed the metrics using a threshold of ( $D = 0.7$ ), leading to the  $TPR = 41\%$  and  $TNR = 98\%$  values (`ENN2$tptr` and `ENN2$tnr`). This is a point of the ROC curve, whose full graph is created with the `mgraph` command (Fig. 4).



**Fig. 4.** Examples of the ROC (left) and REC (right) curves.

### 3.3 Regression Example

The **automobile** dataset goal is to estimate car prices using 16 continuous and 10 nominal attributes. The data includes 205 instances, although there are several missing values. We tested a multiple regression (MR), a NN and a SVM, during the modeling phase:

```
library(rminer) # load the library
d=read.table("imports-85.data",sep=",",na.strings="?") # load the data
d=d[,c(6:8,14,17,19,22,26)] # variable selection: 6,7,8,14,17,19,22,26
d=na.omit(d) # erases from d all examples with missing data
```



```

v=c("kfold",5) # external 5-fold validation
MR=mining(V26~.,d,model="mr",Runs=10,method=v) # 10 runs of 5-fold
m=c(3,100,"kfold",4,"RAE"); s=seq(1,8,1) # m=Nr,Me,... s=1,2,...,8
NN=mining(V26~.,d,model="mlpe",Runs=10,method=v,mpar=m,search=s,feat="s")
m=c(NA,NA,"kfold",4,"RAE"); s=2^seq(-15,3,2) # NA = C/epsilon heuristics
SV=mining(V26~.,d,model="svm",Runs=10,method=v,mpar=m,search=s,feat="s")
print(MR);print(NN);print(SV) # show mining results and save them:
savemining(MR,"imr"); savemining(NN,"inn"); savemining(SV,"isv")

```

Here, we selected only 7 variables as inputs (e.g. V8, the curb weight). Then, we deleted all examples with missing data. Next, the DM models were evaluated using 10 runs of a 5-fold cross validation scheme. The `mining` function performs several fits and returns a list with the obtained predictions and other fields (e.g. time for each run). The NN and SVM models were optimized (i.e. grid search for the best  $H$  and  $\gamma$  parameters) using an internal 4-fold. In this example and for NN, we used an ensemble of 3 networks (`model="mlpe"`). The `seq(from,to,by)` R function was used to define the search ranges for  $H$  and  $\gamma$ , while the `feat="s"` argument triggers the input sensitivity analysis. Next, we show the evaluation:

```

MR=loadmining("imr");NN=loadmining("inn");SV=loadmining("isv")
# show paired t-test and RAE mean and confidence intervals for SV:
print(t.test(mmetric(NN,metric="RAE"),mmetric(SV,metric="RAE")))
print(meanint(mmetric(SV,metric="RAE")))
# plot the average REC curves:
M=vector("list",3); # vector list of mining
M[[1]]=SV;M[[2]]=NN;M[[3]]=MR
mgraph(M,graph="REC",leg=c("SVM","NN","MR"),xval=15000,PDF="rec")
# plot the input relevance bars for SVM: (xval is the L x-axis position)
L=c("n-doors","body-style","drive-wheels","curb-weight","engine-size",
"bore","horsepower") # plot the input relevance (IMP) graph:
mgraph(SV,graph="IMP",leg=L,xval=0.3,PDF="imp")
# plot the VEC curve for the most relevant input (xval=4):
mgraph(SV,graph="VEC",leg=L,xval=4,PDF="vec")

```

In this example, the SVM model (median  $\gamma = 2^{-3}$ ,  $C = 3$  and  $\epsilon = 0.09$ , total execution time 148s) obtained the best predictive results. The average  $\overline{RAE} = 32.5\% \pm 1.4$  is better when statistically compared with NN (median  $H = 3$ ,  $\overline{RAE} = 35.5\% \pm 1.4$ ) and MR ( $41\% \pm 1.0$ ). In all graphs, whiskers denote the 95% t-student confidence intervals. The first `mgraph` function plots the vertically averaged REC curves ( $x$ -axis from 0 to 15000, Fig. 4) and confirms the SVM performance superiority. The next two graphs are based on the sensitivity analysis procedure and are useful for knowledge discovery. In this case, the relative input importances of the SVM model (ordered by importance, Fig. 5) show the curb weight as the most relevant input. The average VEC curve was plotted for this input (Fig. 5), showing a positive effect, where an increase of the curb weight leads to an higher price, particularly within the range [2519,3550].

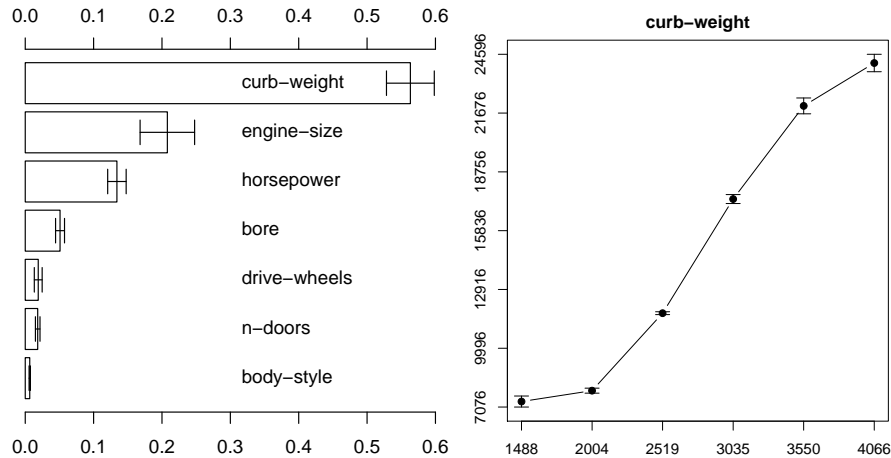


Fig. 5. The input relevances (left) and curb-weight VEC curve (right)

### 3.4 Predictive Performance

We selected 6 classification and 6 regression tasks from UCI [1] for a more detailed predictive performance measurement of the R/rminer capabilities. The aim is show that the R/rminer results are consistent when compared with other DM tools. For a baseline comparison, we adopted the WEKA environment with its default parameters [18]. The datasets main characteristics (e.g number of inputs, examples and classes) are shown in Table 1. For auto-mpg, all examples with missing data were removed (we used the R `na.omit` function).

Table 1. Summary of the UCI datasets used

Task	Description	$I$	Examples	$N_c$
<b>balance</b>	balance scale weight and distance	4	625	3
<b>cmc</b>	contraceptive method choice	9	1473	3
<b>german</b>	German credit data	20	1000	2
<b>heart</b>	Statlog heart disease	13	270	2
<b>house-votes</b>	congressional voting records	16	435	2
<b>sonar</b>	sonar classification (rocks vs mines)	60	208	2
<b>abalone</b>	age of abalone	8	4177	$\mathfrak{R}$
<b>auto-mpg</b>	miles per gallon prediction	7	392	$\mathfrak{R}$
<b>concrete</b>	concrete compressive strength	8	1030	$\mathfrak{R}$
<b>housing</b>	housing prices in suburbs of Boston	13	506	$\mathfrak{R}$
<b>servo</b>	rise time of a servomechanism	4	167	$\mathfrak{R}$
<b>white</b>	white wine quality	11	4899	$\mathfrak{R}$

For each task, we executed 10 runs of a 5-fold validation. The NN and SVM hyperparameters were ranged within  $H \in \{0, 1, 2, \dots, 9\}$  and  $\gamma \in \{2^{-15}, 2^{-13}, \dots, 2^3\}$ , in a total of 10 searches per DM model. For NN, we tested the ensemble variant with  $N_r=3$ . An internal 3-fold was used during the grid search, which optimized the global AUC (classification) and RRSE (regression) metrics (the code used is available at the rminer Web page).

Table 2 presents the test set results. In general, the R/rminer outperformed the baseline tool (the only exceptions are for NN and the house-votes and sonar tasks). In particular, a higher improvement was achieved for SVM, when compared with the WEKA SVM version, with differences ranging from 3.5 pp (house-votes) to 39.1 pp (servo). When comparing the two rminer methods, SVM outperforms NN in 4 classification cases, while NN is better in 4 regression datasets.

**Table 2.** Classification and regression results (average global AUC and RRSE values, in %; best values are in **bold**; underline denotes significant difference under a paired t-test between R/rminer and WEKA)

Task	WEKA		R/rminer	
	NN	SVM	NN	SVM
<b>balance</b>	97.5±0.2	88.1±0.2	<b>99.5</b> ±0.1	<u>98.9</u> ±0.2
<b>cmc</b>	71.4±0.3	63.8±0.3	<b>73.9</b> ±0.0	<u>72.9</u> ±0.2
<b>german</b>	73.5±0.7	67.2±0.7	<u>76.3</u> ±0.8	<b>77.9</b> ±0.5
<b>heart</b>	85.6±1.2	83.7±0.4	<u>88.5</u> ±1.4	<b>90.2</b> ±0.4
<b>house-votes</b>	98.6±0.2	95.7±0.3	98.0±0.5	<b>99.2</b> ±0.1
<b>sonar</b>	89.2±1.3	76.6±1.8	87.4±0.9	<b>95.6</b> ±0.8
<b>abalone</b>	72.7±2.1	69.9±0.1	<b>64.0</b> ±0.1	<u>66.0</u> ±0.1
<b>auto-mpg</b>	44.3±3.4	44.5±0.3	<u>37.4</u> ±3.1	<b>34.8</b> ±0.4
<b>concrete</b>	46.5±1.4	65.6±0.2	<b>31.8</b> ±0.4	<u>35.9</u> ±0.5
<b>housing</b>	49.9±3.0	55.2±0.4	<b>38.3</b> ±1.6	<u>40.1</u> ±1.3
<b>servo</b>	46.1±4.5	84.0±0.4	<b>40.8</b> ±6.0	<u>44.9</u> ±1.5
<b>white</b>	91.3±3.1	85.5±0.1	<u>79.0</u> ±0.3	<b>76.1</b> ±0.6

## 4 Conclusions

The R tool is an open source environment that is widely used for data analysis. In this work, we present our rminer library, which eases the use of R (e.g. for non-IT specialists) to solve DM classification and regression tasks. The library is particularly suited for NNs and SVMs, flexible and nonlinear learning techniques that are promising due to their predictive performances. Two tutorial examples (e.g. satellite image classification) were used to show the R/rminer potential under the CRISP-DM methodology. Additional experiments were held in order to measure the rminer library predictive performances. Overall, competitive results were obtained, in particular the SVM model for the classification tasks and NN

for the regression ones. In future work, we intend to expand the rminer capabilities (e.g. unsupervised learning) and applications (e.g. telecommunications).

## References

1. A. Asuncion and D. Newman. UCI Machine Learning Repository, Univ. of California Irvine, <http://www.ics.uci.edu/~mlearn/MLRepository.html>, 2007.
2. P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. CRISP-DM 1.0: Step-by-step data mining guide. *CRISP-DM consortium*, 2000.
3. V. Cherkassy and Y. Ma. Practical Selection of SVM Parameters and Noise Estimation for SVM Regression. *Neural Networks*, 17(1):113–126, 2004.
4. P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.
5. P. Cortez, C. Lopes, P. Sousa, M. Rocha, and M. Rio. Symbiotic Data Mining for Personalized Spam Filtering. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI-09)*, pages 149–156. IEEE, 2009.
6. P. Cortez, J. Teixeira, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Using data mining for wine quality assessment. In J. Gama et al., editors, *Discovery Science*, volume 5808 of *Lecture Notes in Computer Science*, pages 66–79. Springer, 2009.
7. M. Goebel and L. Gruenwald. A Survey of Data Mining and Knowledge Discovery Software Tools. *SIGKDD Explorations*, 1(1):20–33, June 1999.
8. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, NY, USA, 2nd ed., 2008.
9. R. Kewley, M. Embrechts, and C. Breneman. Data Strip Mining for the Virtual Design of Pharmaceuticals with Neural Networks. *IEEE Trans Neural Networks*, 11(3):668–679, May 2000.
10. G. Piatetsky-Shapiro. Data Mining Tools Used Poll. <http://www.kdnuggets.com/polls/2009/data-mining-tools-used.htm>, 2009.
11. F. Provost and P. Domingos. Tree Induction for Probability-Based Ranking. *Machine Learning*, 52(3):199–215, 2003.
12. R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0, <http://www.R-project.org>, 2009.
13. K. Rexer. Second annual data miner survey. Tech. report, Rexer Analytics, 2008.
14. M. Rocha, P. Cortez, and J. Neves. Evolution of Neural Networks for Classification and Regression. *Neurocomputing*, 70:2809–2816, 2007.
15. J. Tinoco, A.G. Correia, and P. Cortez. A Data Mining Approach for Jet Grouting Uniaxial Compressive Strength Prediction. In *World Congress on Nature and Biologically Inspired Computing (NaBIC'09)*, pages 553–558, Coimbatore, India, December 2009. IEEE.
16. E. Turban, R. Sharda, J. Aronson, and D. King. *Business Intelligence, A Managerial Approach*. Prentice-Hall, 2007.
17. G. Williams. Rattle: A Data Mining GUI for R. *The R Journal*, 1(2):45–55, 2009.
18. I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, SF, USA, 2005.
19. T.F. Wu, C.J. Lin, and R.C. Weng. Probability estimates for multi-class classification by pairwise coupling. *The Journal of Machine Learning Research*, 5:975–1005, 2004.