Support Vector Regression (SVR)

for regression

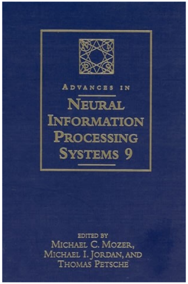
Freddy Hernández

4 de noviembre de 2019

Rday Medellín

SVR were developed by Drucker et al. (1996)

Url: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.5909



Support Vector Regression Machines

Harris Drucker* **Chris J.C. Burges**** **Linda Kaufman****
Alex Smola** **Vladimir Vapnik** [+]

*Bell Labs and Monmouth University
Department of Electronic Engineering
West Long Branch, NJ 07764
**Bell Labs [+]AT&T Labs

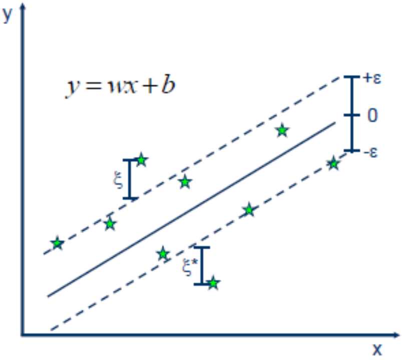
Abstract

A new regression technique based on Vapnik's concept of support vectors is introduced. We compare support vector regression (SVR) with a committee regression technique (bagging) based on regression trees and ridge regression done in feature space. On the basis of these experiments, it is expected that SVR will have advantages in high dimensionality space because SVR optimization does not depend on the dimensionality of the input space.

https://www.svm-tutorial.com/2017/02/svms-overview-support-vector-machines/

Idea

The objective of the Support Vector Machine algorithm is to find a margin of tolerance (ϵ) to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.



- Minimize:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\left(\xi_i + \xi_i^*\right)$$

- Constraints:

$$y_i - wx_i - b \leq \varepsilon + \xi_i$$

$$wx_i + b - y_i \leq \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

R package

The package **e1071** contains the `svm` function used for SVMs. To install the package:

```
install.packages("e1071")
```

To load the package:

```
library(e1071)
```

The main function is:

```
svm(formula, data, scale=TRUE,
    kernel=linear or polynomial or radial or sigmoid,
    degree=3, gamma=1/n, coef0=0, cost=1, ...)
```

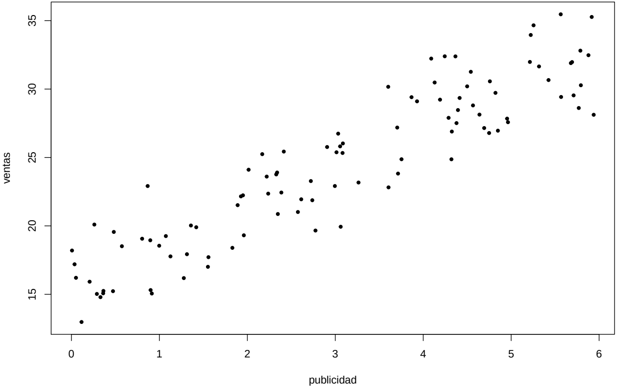
Consult the vignette:

https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf

Example 1

Let's first download some `Train` data.

```
url <- "https://raw.githubusercontent.com/rdaymedellin/
tutoriales_Rday_2019/master/Machine%20Learning%20SVM/publi_ventas.txt"
Train <- read.table(url, sep=";", header=TRUE)
with(Train, plot(x=publicidad, y=ventas, pch=20))
```



Example 1

Exploring the dimensions and the content of `Train` dataset.

```
Train <- Train[order(Train$publicidad), ]
dim(Train)
```

```
## [1] 100   2
```

```
head(Train)
```

```
##     publicidad   ventas
## 14 0.006819519 18.20038
## 32 0.035925780 17.19406
## 83 0.051887480 16.20541
## 84 0.115168600 12.97619
## 11 0.207212600 15.92082
## 48 0.260738700 20.09766
```

Example 1

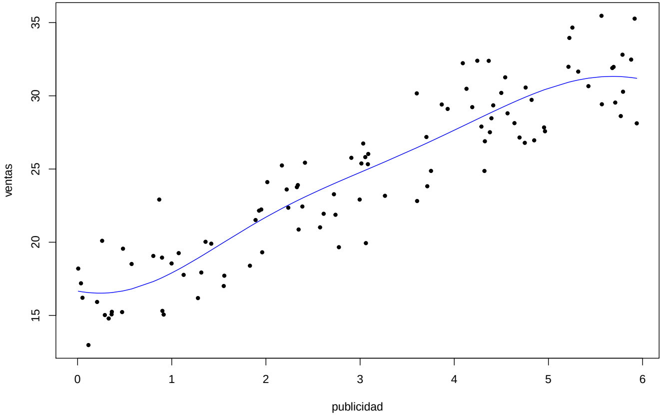
```r
svm_lin <- svm(ventas ~ publicidad, data=Train, scale=TRUE)
print(svm_lin)
```

```
##
## Call:
## svm(formula = ventas ~ publicidad, data = Train, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##     epsilon:  0.1
##
##
## Number of Support Vectors:  90
```

Example 1

The midline.

```
y_hat <- predict(svm_lin, Train)
with(Train, plot(x=publicidad, y=ventas, pch=20))
points(x=Train$publicidad, y=y_hat, col="blue", type="l")
```



The tune parameters can be found using the tune function. As performance measure, the classification error is used for classification, and the mean squared error for regression.

```
obj <- tune(method=svm, ventas ~ publicidad, data=Train,
            ranges=list(epsilon=seq(from=0.1, to=1, length.out=10),
                        cost=seq(from=0.1, to=100, length.out=10)),
            tunecontrol=tune.control(sampling="fix"))
obj
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: fixed training/validation set
##
## - best parameters:
##  epsilon cost
##      0.6 11.2
##
## - best performance: 5.191462
```

Example 1

```r
svm_lin <- svm(ventas ~ publicidad, data=Train, scale=TRUE,
               epsilon=obj$best.parameters[1],
               cost=obj$best.parameters[2])

y_hat <- predict(svm_lin, Train)

cor(Train$publicidad, y_hat)
```

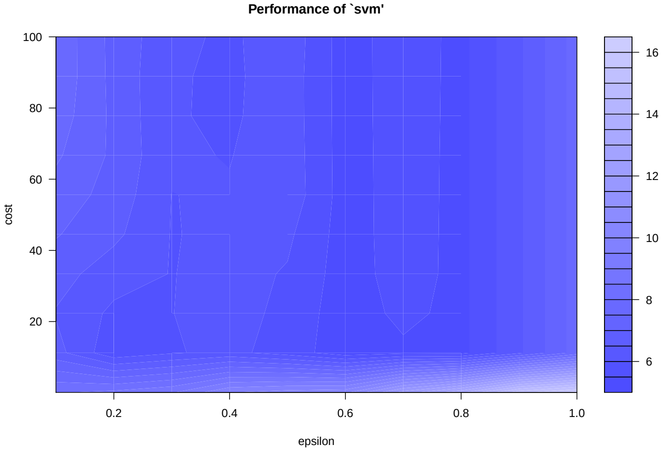
```
## [1] 0.9953371
```

```r
sqrt(mean((Train$publicidad - y_hat)^2))
```

```
## [1] 21.86075
```

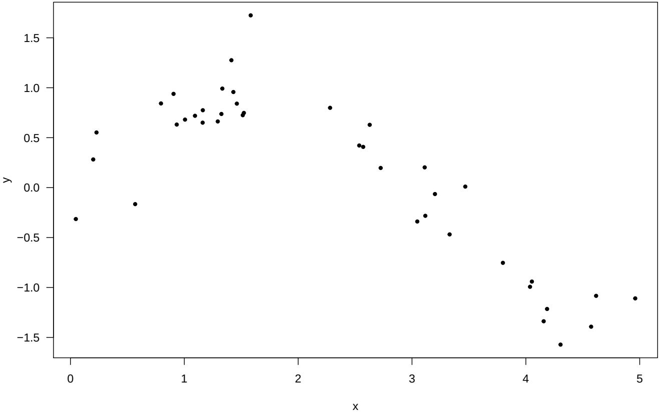
Example 1

```r
plot(obj)
```



Example

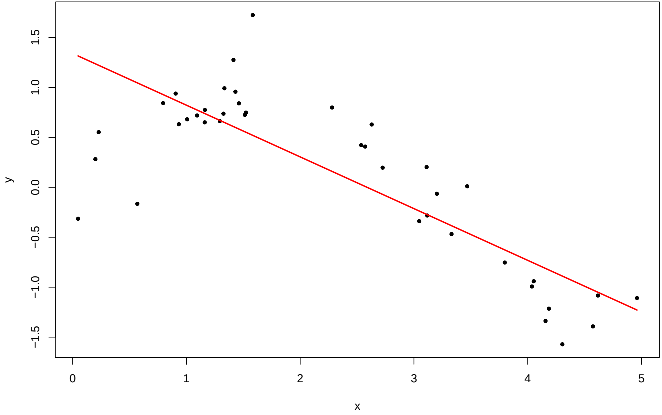
```r
set.seed(1234)
x <- sort(runif(n=40, min=0, max=5))
set.seed(1234)
y <- sin(x) + rnorm(40, sd=0.3)
plot(x, y, pch=20, las=1)
```



Linear kernel

```
mod_lin <- svm(y ~ x, kernel="linear")
y_hat_lin <- predict(mod_lin)
cor(y, y_hat_lin)
```

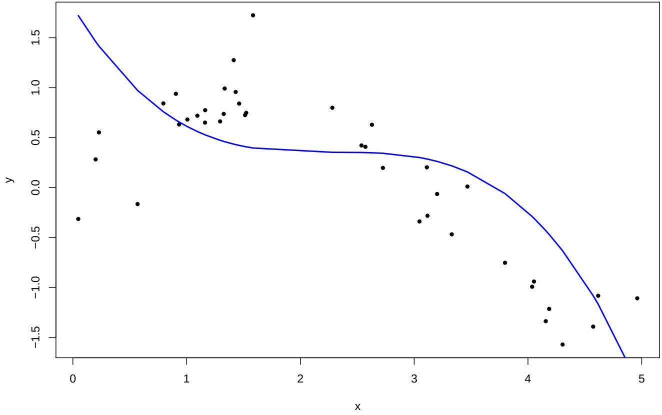
```
## [1] 0.7852445
```



Polynomial kernel

```
mod_pol <- svm(y ~ x, kernel="polynomial")
y_hat_pol <- predict(mod_pol)
cor(y, y_hat_pol)
```

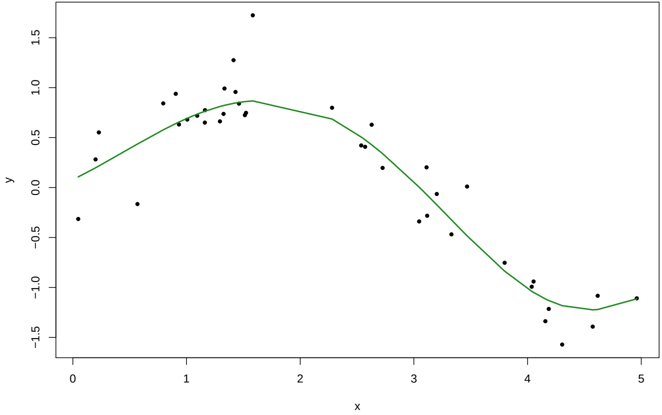
```
## [1] 0.6468922
```



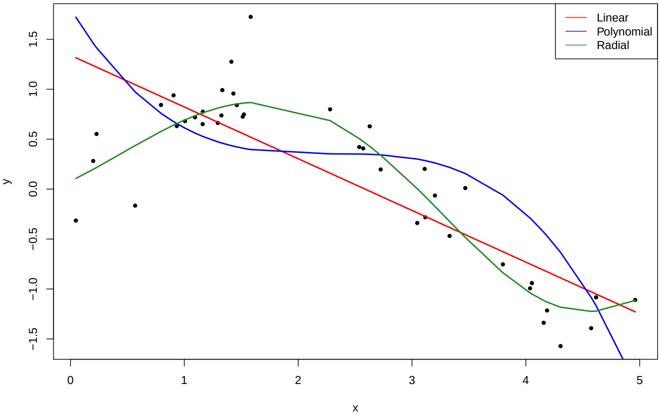
RBF kernel

```
mod_rad <- svm(y ~ x, kernel="radial")
y_hat_rad <- predict(mod_rad)
cor(y, y_hat_rad)
```

```
## [1] 0.9506258
```



Comparisons



Animations

https://github.com/rdaymedellin/tutoriales_Rday_2019/blob/master/Machine%20Learning%20SVM/animation_svm_pol_reg.gif

https://github.com/rdaymedellin/tutoriales_Rday_2019/blob/master/Machine%20Learning%20SVM/animation_svm_rad_reg.gif

Tuning parameters

We can use the `tune.svm` function.

```
pol_tune = tune.svm(y~x, kernel="polynomial",
                    degree=c(2, 3, 4), coef0=c(0.1, 0.5, 1, 2, 3))
summary(pol_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  degree coef0
##       4     2
##
## - best performance: 0.08341557
##
## - Detailed performance results:
##    degree coef0      error dispersion
## 1       2   0.1 0.16156050 0.15002103
## 2       3   0.1 0.32109973 0.35392559
## 3       4   0.1 0.71894997 1.10410176
## 4       2   0.5 0.16190742 0.16890318
```

Tuning parameters

We can use the `tune.svm` function.

```
rad_tune = tune.svm(y~x, kernel="radial",
                    gamma=c(0.1, 0.5, 1, 1.5, 2))
summary(rad_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma
##    0.5
##
## - best performance: 0.09273725
##
## - Detailed performance results:
##   gamma      error dispersion
## 1   0.1 0.16878872 0.12622542
## 2   0.5 0.09273725 0.08338983
## 3   1.0 0.09790368 0.08082934
## 4   1.5 0.10448328 0.07865747
```

Best

```r
best_pol <-  pol_tune$best.model
y1 <- predict(best_pol)
cor(y, y1)
```

```
## [1] 0.9512053
```

```r
best_rad <-  rad_tune$best.model
y2 <- predict(best_rad)
cor(y, y2)
```

```
## [1] 0.9468661
```

Comparisons