Support Vector Regression (SVR)

for regression

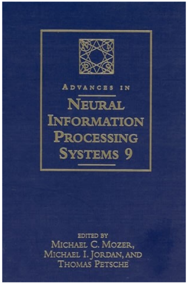
Freddy Hernández

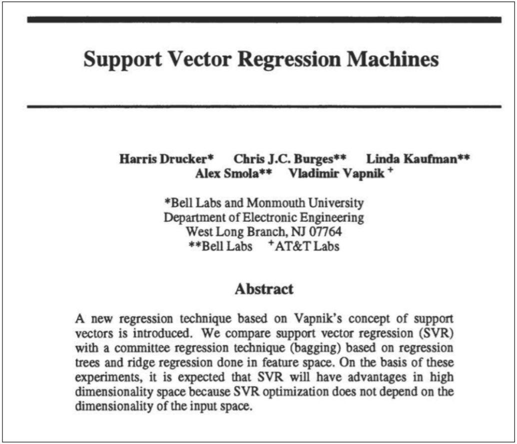
6 de noviembre de 2019

Support Vector Regression

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

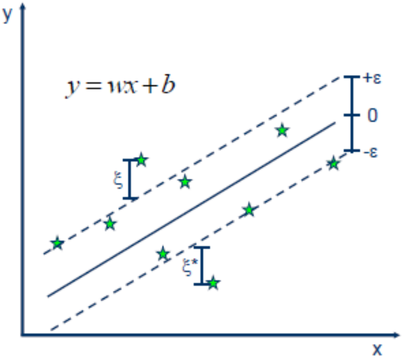
Abstract

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.

An overview of Support Vector Machines

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

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:

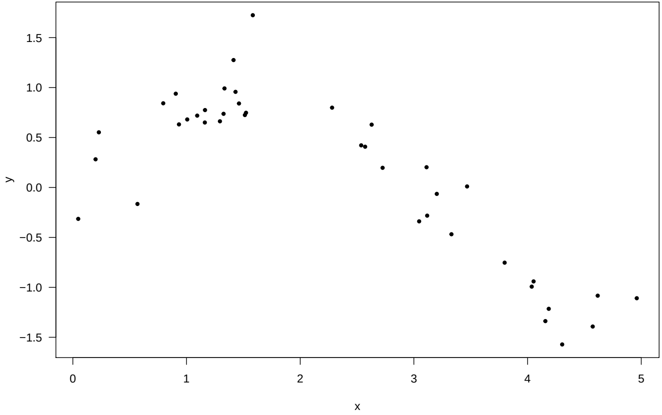
```
svm(formula, data, scale=TRUE,
    kernel=linear or polynomial or radial or sigmoid,
    degree=3, gamma=1/n, coef0=0, cost=1, ...)
```

Consult the vignette:

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

Example with simulated data

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



MSE function

This function can be used to obtain the MSE.

```
mse <- function(y, y_hat) mean((y - y_hat)^2)
```

Linear kernel

```r
mod_lin <- svm(y ~ x, kernel="linear")
```

To obtain $\hat{y}$.

```r
y_hat_lin <- predict(mod_lin)
```

To obtain the correlation coefficient and MSE.

```r
cor(y, y_hat_lin)
```

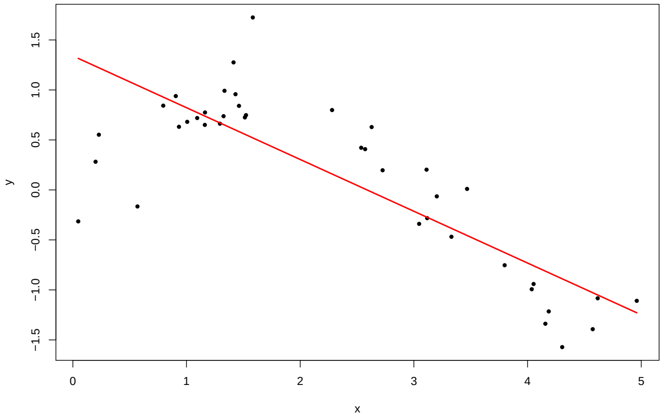
```
## [1] 0.7852445
```

```r
mse(y, y_hat_lin)
```

```
## [1] 0.2687447
```

Linear kernel

```
plot(x, y, pch=20)
points(x=x, y=y_hat_lin, type="l", lwd=2, col="red")
```



Polynomial kernel

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

To obtain $\hat{y}$.

```
y_hat_pol <- predict(mod_pol)
```

To obtain the correlation coefficient and MSE.

```
cor(y, y_hat_pol)
```

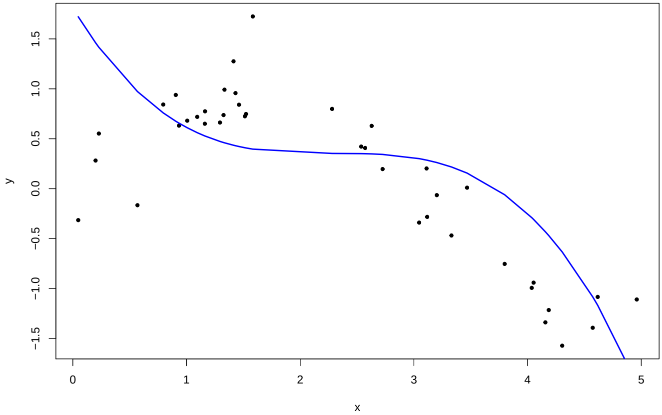
```
## [1] 0.6468922
```

```
mse(y, y_hat_pol)
```

```
## [1] 0.4344635
```

Polynomial kernel

```
plot(x, y, pch=20)
points(x=x, y=y_hat_pol, type="l", lwd=2, col="blue")
```



RBF kernel

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

To obtain $\hat{y}$.

```
y_hat_rad <- predict(mod_rad)
```

To obtain the correlation coefficient and MSE.

```
cor(y, y_hat_rad)
```

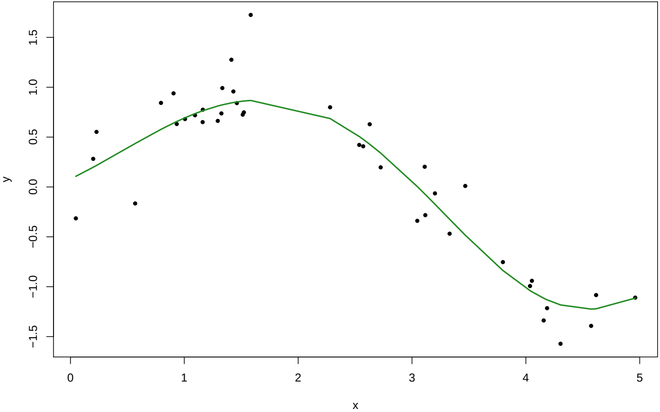
```
## [1] 0.9506258
```

```
mse(y, y_hat_rad)
```

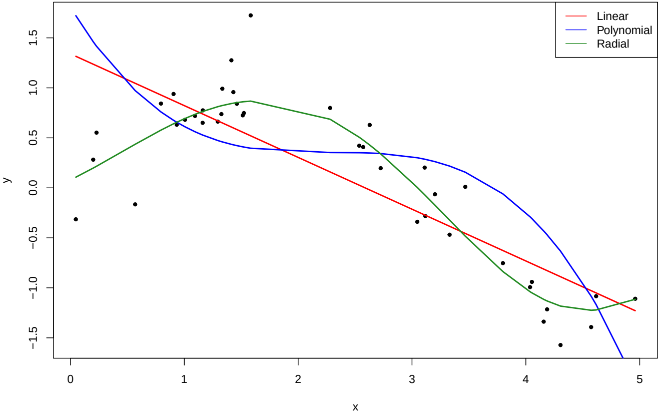
```
## [1] 0.06821404
```

RBF kernerl

```
plot(x, y, pch=20)
points(x=x, y=y_hat_rad, type="l", lwd=2, col="forestgreen")
```



Comparisons



Animations

Below we have some .gif figures to explore the tune parameter effects.

- 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.

```
tune.svm(formula, data, kernel,
         degree, gamma, coef0, cost)
```

- the classification error is used for classification.
- the mean squared error for regression.

Tuning parameters for polynomial

```r
pol_tune = tune.svm(y~x, kernel="polynomial",
                    degree=c(2, 3, 4),
                    gamma=c(0.1, 1, 2),
                    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 gamma coef0
##       4     1     2
##
## - best performance: 0.08341557
##
## - Detailed performance results:
##    degree gamma coef0      error dispersion
## 1       2   0.1   0.1 0.32067808 0.21307867
## 2       3   0.1   0.1 0.65578295 0.45230278
## 3       4   0.1   0.1 0.78137035 0.54532617
## 4       2   1.0   0.1 0.16156050 0.15002103
## 5       3   1.0   0.1 0.32109973 0.35392559
## 6       4   1.0   0.1 0.71894997 1.10410176
## 7       2   2.0   0.1 0.15656789 0.15172337
## 8       3   2.0   0.1 0.24171162 0.25211007
```

Tuning parameters for radial

```r
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
## 5   2.0 0.10722114 0.07893561
```

Best

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

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

```r
mse(y, y1)
```

```
## [1] 0.06545502
```

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

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

```r
mse(y, y2)
```

```
## [1] 0.07281149
```

Comparisons