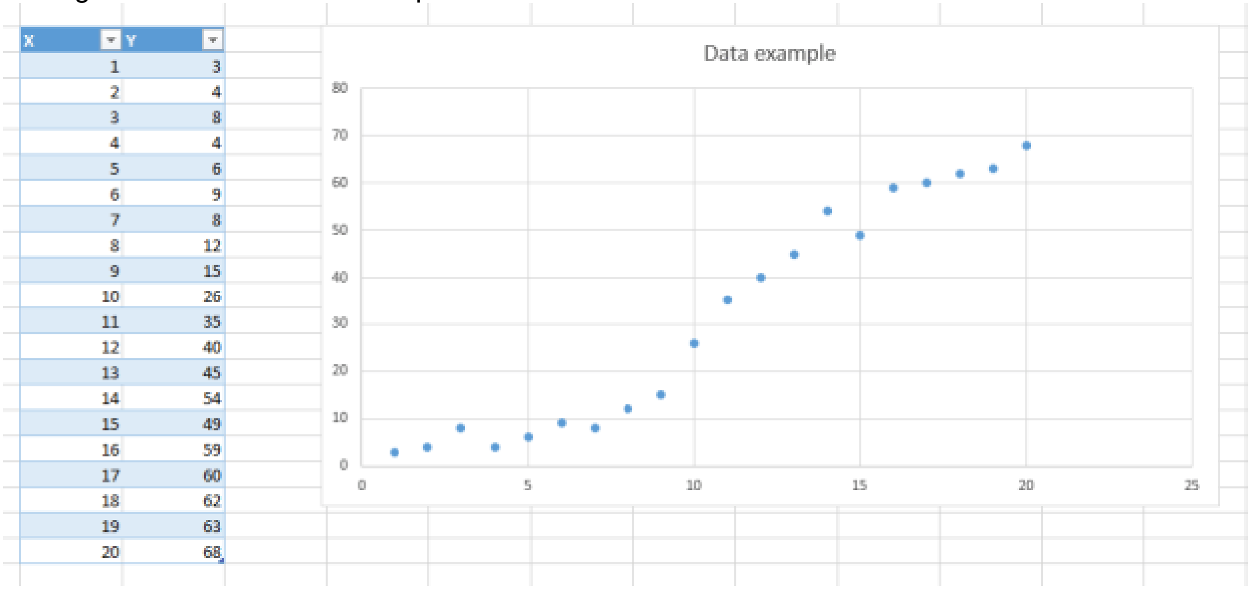
Support Vector Regression with R

77 Replies

In this article I will show how to use R to perform a Support Vector Regression.
We will first do a simple linear regression, then move to the Support Vector Regression so that you can see how the two behave with the same data.

A simple data set

To begin with we will use this simple data set:



X	Y
1	3
2	4
3	8
4	4
5	6
6	9
7	8
8	12
9	15
10	26
11	35
12	40
13	45
14	54
15	49
16	59
17	60
18	62
19	63
20	68

I just put some data in excel. I prefer that over using an existing well-known data-set because the purpose of the article is not about the data, but more about the models we will use.
As you can see there seems to be some kind of relation between our two variables X and Y, and it look like we could fit a line which would pass near each point.
Let's do that in R !

Step 1: Simple linear regression in R

Here is the same data in CSV format, I saved it in a file regression.csv :

```
X,Y
1,3
2,4
3,8
4,4
5,6
6,9
7,8
8,12
9,15
10,26
11,35
12,40
13,45
14,54
15,49
16,59
17,60
18,62
19,63
20,68
```

We can now use R to display the data and fit a line:

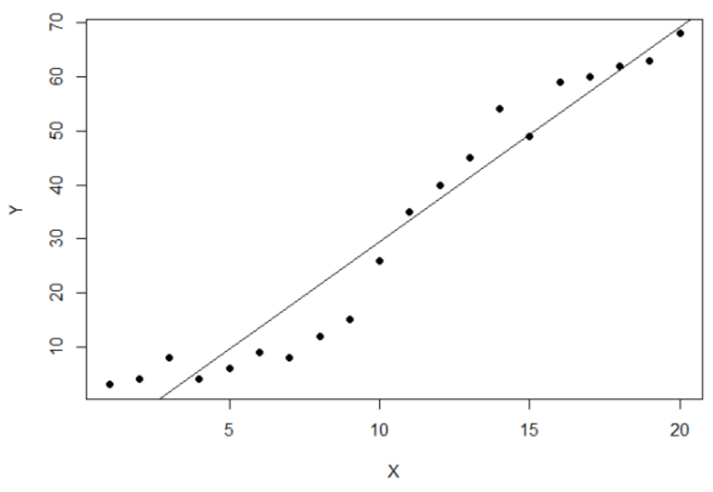
```r
# Load the data from the csv file
dataDirectory <- "D:/" # put your own folder here
data <- read.csv(paste(dataDirectory, 'regression.csv', sep=""), header = TRUE)

# Plot the data
plot(data, pch=16)

# Create a linear regression model
model <- lm(Y ~ X, data)

# Add the fitted line
abline(model)
```

The code above displays the following graph:



Step 2: How good is our regression ?

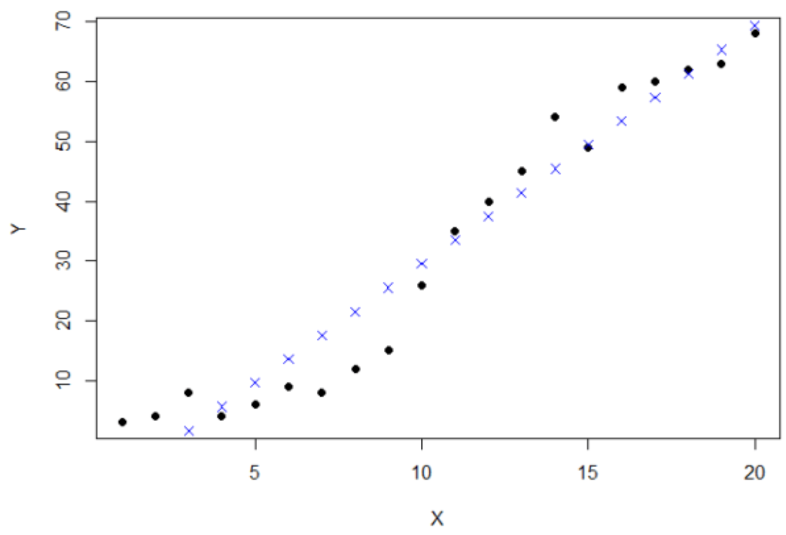
In order to be able to compare the linear regression with the support vector regression we first need a way to measure how good it is.
To do that we will change a little bit our code to visualize each prediction made by our model

```
dataDirectory <- "D:/"
data <- read.csv(paste(dataDirectory, 'regression.csv', sep=""), header = TRUE)

plot(data, pch=16)
model <- lm(Y ~ X , data)

# make a prediction for each X
predictedY <- predict(model, data)

# display the predictions
points(data$X, predictedY, col = "blue", pch=4)
```



This produces the following graph:
For each data point X_i Xi the model makes a prediction $\hat{Y}_i$ Y^i displayed as a blue cross on the graph. The only difference with the previous graph is that the dots are not connected with each other.
In order to measure how good our model is we will compute how much errors it makes.
We can compare each Y_i Yi value with the associated predicted value $\hat{Y}_i$ Y^i and see how far away they are with a simple difference.
Note that the expression $\hat{Y}_i - Y_i$ Y^i−Yi is the error, if we make a perfect prediction $\hat{Y}_i$ Y^i will be equal to Y_i Yi and the error will be zero.
If we do this for each data point and sum the error we will have the sum of the errors, and if we takes the mean we will get the Mean Squared Error (MSE)

$$MSE=\frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$ MSE=1n∑i=1n(Y^i−Yi)2

A common way to measure error in machine learning is to use the Root Mean Squared Error (RMSE) so we will use it instead.
To compute the RMSE we take the square root and we get the RMSE

$$RMSE=\sqrt{MSE}$$ RMSE=MSE

Using R we can come with the following code to compute the RMSE

```
rmse <- function(error)
{
  sqrt(mean(error^2))
}

error <- model$residuals  # same as data$Y - predictedY
```

```
predictionRMSE <- rmse(error)   # 5.703778
```

We know now that the RMSE of our linear regression model is 5.70. Let's try to improve it with SVR !

Step 3: Support Vector Regression

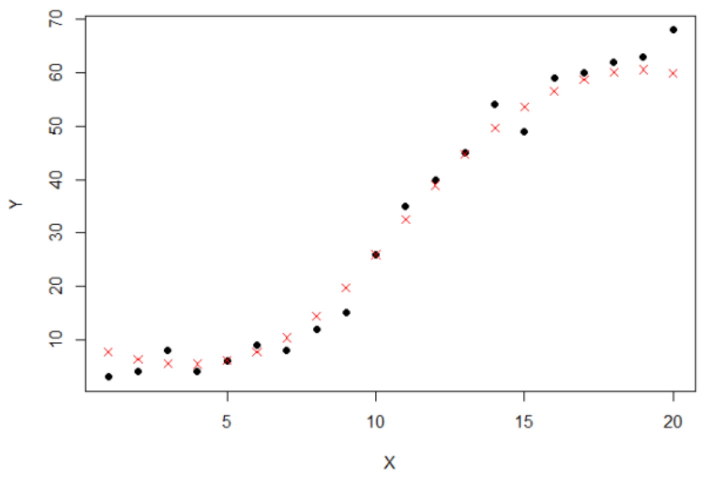
In order to create a SVR model with R you will need the package **e1071**. So be sure to install it and to add the **library(e1071)** line at the start of your file.
Below is the code to make predictions with Support Vector Regression:

```
model <- svm(Y ~ X , data)

predictedY <- predict(model, data)

points(data$X, predictedY, col = "red", pch=4)
```

As you can see it looks a lot like the linear regression code. Note that we called the **svm** function (not **svr** !) it's because this function can also be used to make classifications with Support Vector Machine. The function will automatically choose SVM if it detects that the data is categorical (if the variable is a factor in R).
The code draws the following graph:



This time the predictions is closer to the real values ! Let's compute the RMSE of our support vector regression model.

```
# /!\ this time  svrModel$residuals  is not the same as data$Y - predictedY
# so we compute the error like this
error <- data$Y - predictedY
svrPredictionRMSE <- rmse(error)  # 3.157061
```

As expected the RMSE is better, it is now 3.15 compared to 5.70 before.
But can we do better ?

Step 4: Tuning your support vector regression model

In order to improve the performance of the support vector regression we will need to select the best parameters for the model.

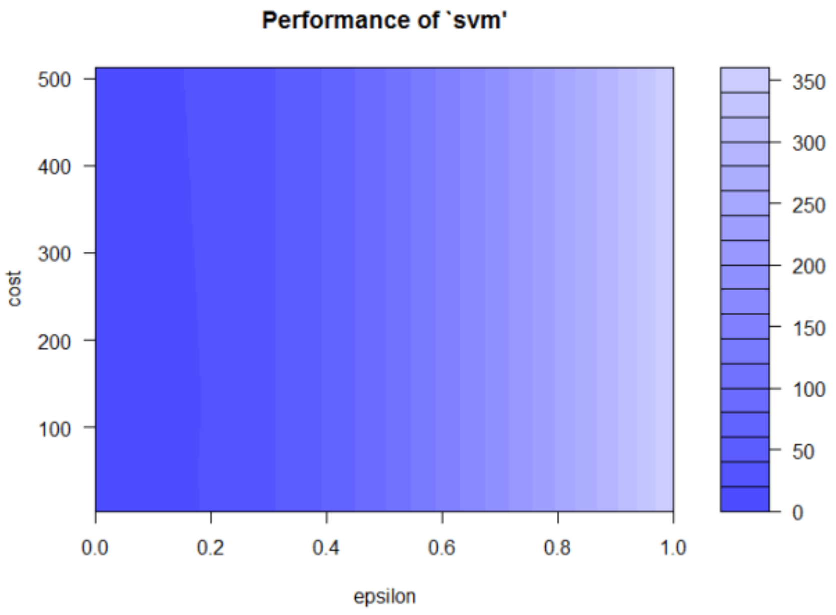
In our previous example, we performed an epsilon-regression, we did not set any value for **epsilon** (ϵ ϵ), but it took a default value of 0.1. There is also a **cost** parameter which we can change to avoid overfitting.
The process of choosing these parameters is called hyperparameter optimization, or **model selection**.
The standard way of doing it is by doing a **grid search**. It means we will train a lot of models for the different couples of ϵ ϵ and cost, and choose the best one.

```
# perform a grid search
tuneResult <- tune(svm, Y ~ X,  data = data,
              ranges = list(epsilon = seq(0,1,0.1), cost = 2^(2:9))
)
print(tuneResult)
# best performance: MSE = 8.371412, RMSE = 2.89 epsilon 1e-04 cost 4
# Draw the tuning graph
plot(tuneResult)
```

There is two important points in the code above:

- we use the tune method to train models with $\epsilon=0,0.1,0.2,...,1$ ϵ=0,0.1,0.2,...,1 and cost = $2^2,2^3,2^4,...,2^9$ 22,23,24,...,29 which means it will train 88 models (it can take a long time)
- the tuneResult returns the MSE, don't forget to convert it to RMSE before comparing the value to our previous model.

The last line plot the result of the grid search:



On this graph we can see that **the darker the region is the better our model is** (because the RMSE is closer to zero in darker regions).

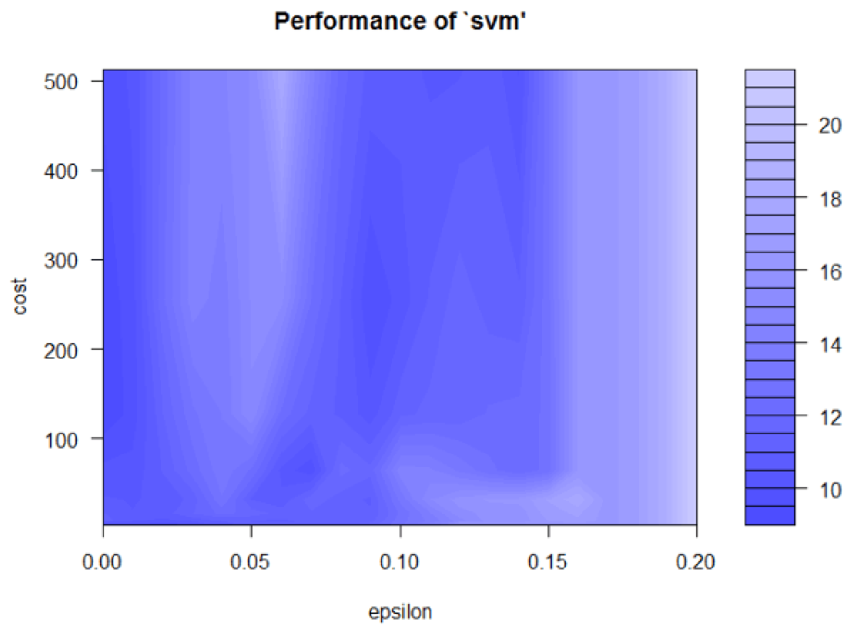
This means we can try another grid search in a narrower range we will try with ϵ ϵ values between 0 and 0.2. It does not look like the cost value is having an effect for the moment so we will keep it as it is to see if it changes.

```
tuneResult <- tune(svm, Y ~ X,  data = data,
                   ranges = list(epsilon = seq(0,0.2,0.01), cost = 2^(2:9))
)

print(tuneResult)
plot(tuneResult)
```

We trained different 168 models with this small piece of code.
As we zoomed-in inside the dark region we can see that there is several darker patch.

From the graph you can see that models with C between 200 and 300 and ϵ ϵ between 0.8 and 0.9 have less error.



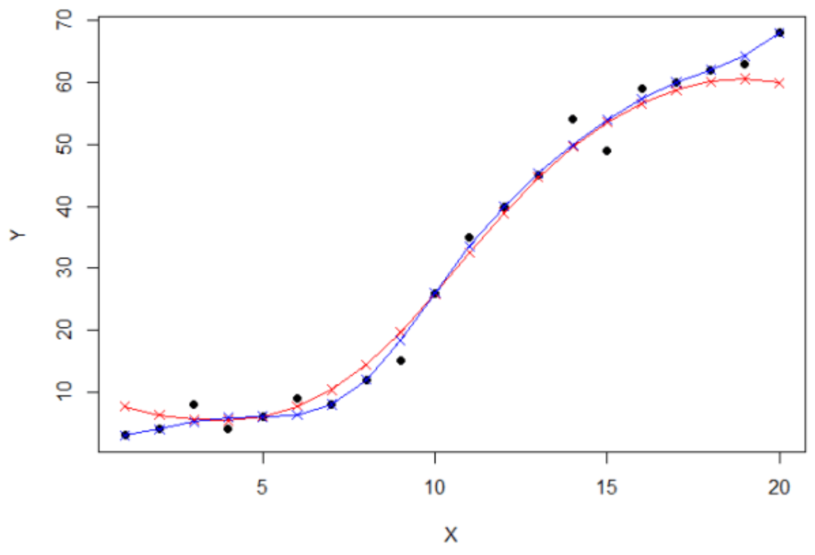
Hopefully for us, we don't have to select the best model with our eyes and R allows us to get it very easily and use it to make predictions.

```
tunedModel <- tuneResult$best.model
tunedModelY <- predict(tunedModel, data)

error <- data$Y - tunedModelY

# this value can be different on your computer
# because the tune method  randomly shuffles the data
tunedModelRMSE <- rmse(error)  # 2.219642
```

We improved again the RMSE of our support vector regression model !
If we want we can visualize both our models. The first SVR model is in red, and the tuned SVR model is in blue on the graph below :



Alexandre KOWALCZYK http://www.svm-tutorial.com/2014/10/support-vector-regression-r/