

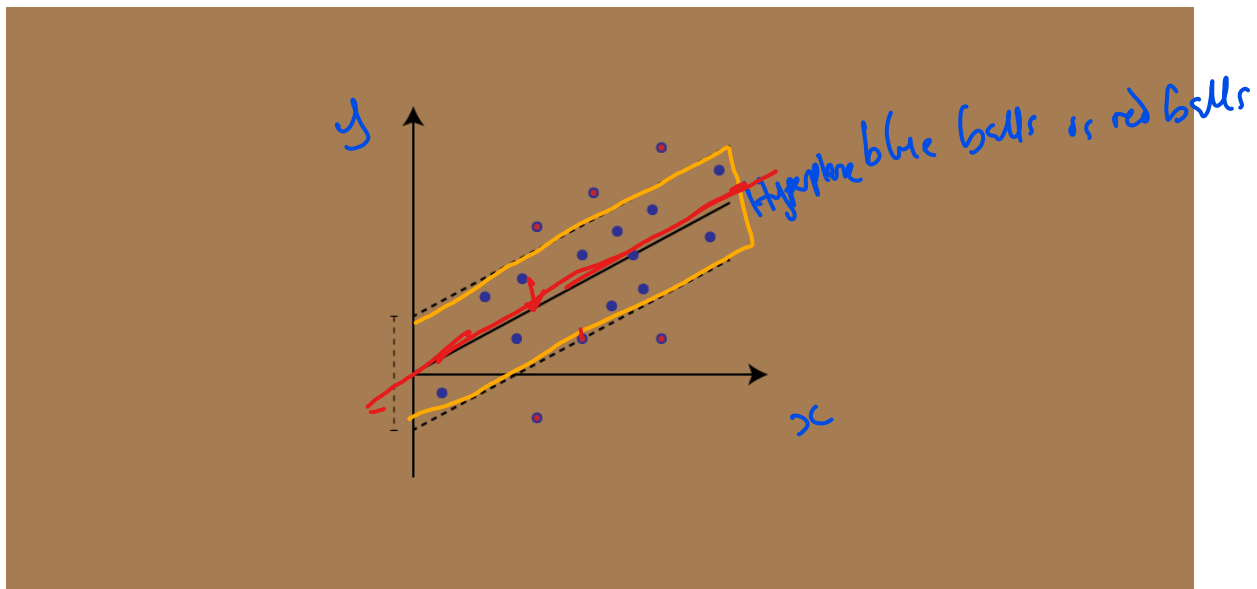
Support Vector Regression Tutorial for Machine Learning

Tue → 01/03/2022

Unlocking a New World with the Support Vector Regression Algorithm

Support Vector Machines (SVM) are popularly and widely used for classification problems in machine learning. I've often relied on this not just in machine learning projects but when I want a quick result in a hackathon.

But SVM for regression analysis? I hadn't even considered the possibility for a while! And even now when I bring up "Support Vector Regression" in front of machine learning beginners, I often get a bemused expression. I understand – most courses and experts don't even mention Support Vector Regression (SVR) as a machine learning algorithm.



But SVR has its uses as you'll see in this tutorial. We will first quickly understand what SVM is, before diving into the world of Support Vector Regression and how to implement it in Python!

Note: You can learn about Support Vector Machines and Regression problems in course format here (it's free!):

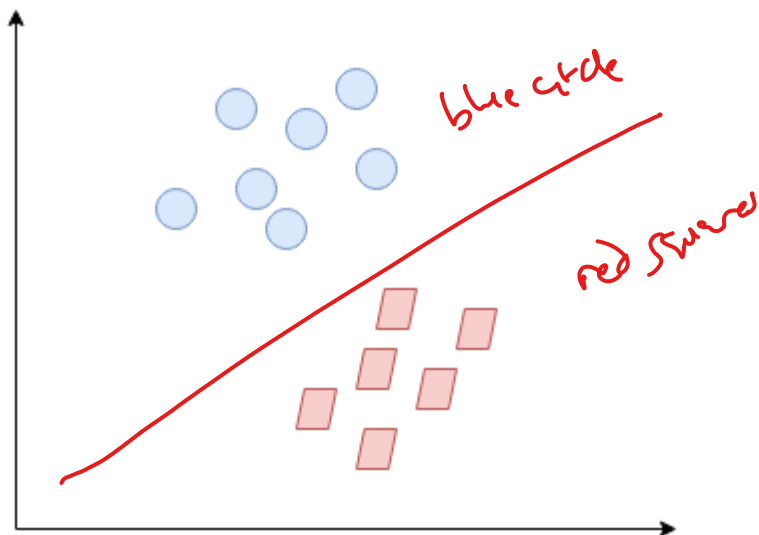
- [Support Vector Machine \(SVM\) in Python and R](#)
- [Fundamentals of Regression Analysis](#)

Here's what we'll cover in this Support Vector Regression tutorial:

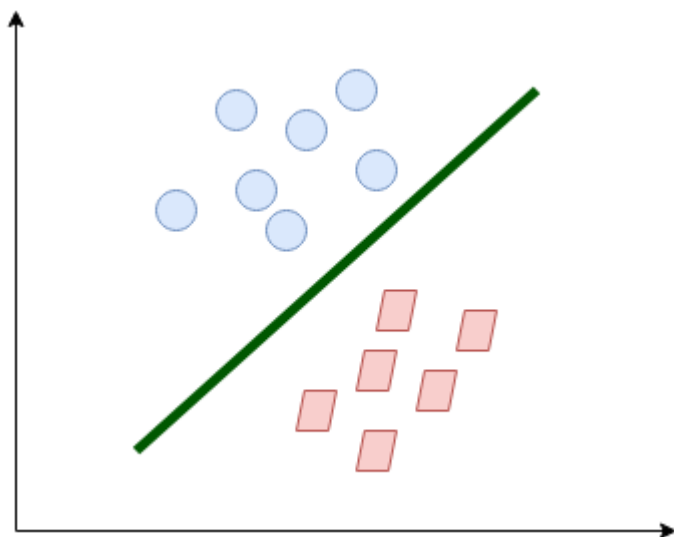
- What is a Support Vector Machine (SVM)?
- Hyperparameters of the Support Vector Machine Algorithm
- Introduction to Support Vector Regression (SVR)
- Implementing Support Vector Regression in Python

What is a Support Vector Machine (SVM)?

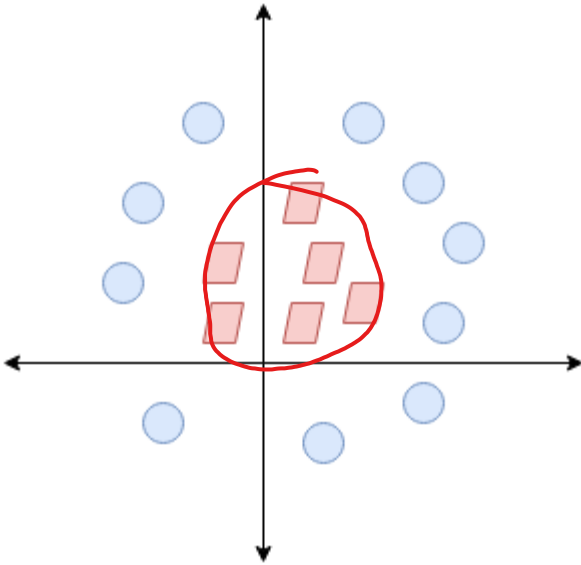
So what exactly is Support Vector Machine (SVM)? We'll start by understanding SVM in simple terms. Let's say we have a plot of two label classes as shown in the figure below:



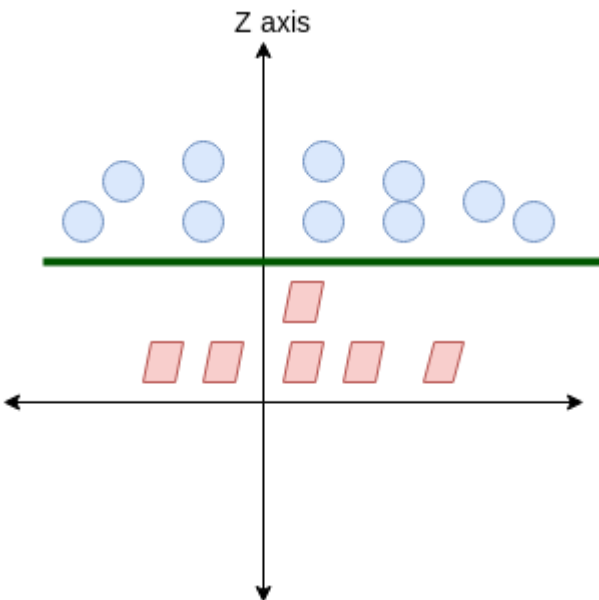
Can you decide what the separating line will be? You might have come up with this:



The line fairly separates the classes. This is what SVM essentially does – **simple class separation**. Now, what is the data was like this:

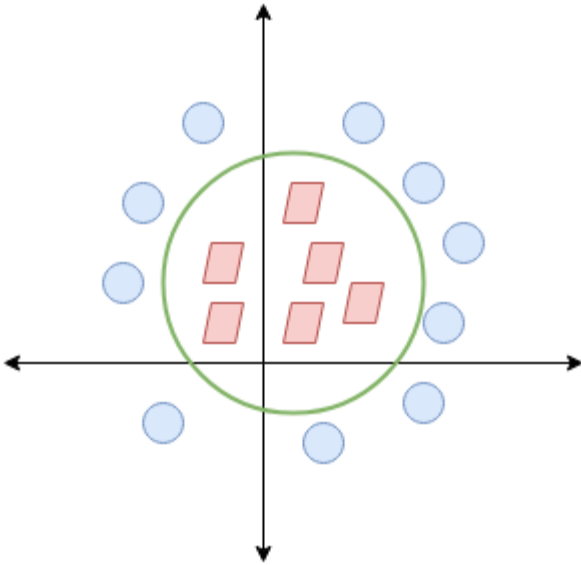


Here, we don't have a simple line separating these two classes. So we'll extend our dimension and introduce a new dimension along the z-axis. We can now separate these two classes:



When we transform this line back to the original plane, it maps to the circular boundary as I've shown here:



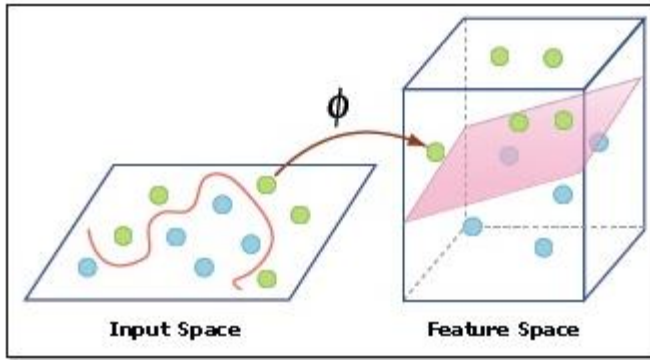


This is exactly what SVM does! It tries to find a line/hyperplane (in multidimensional space) that separates these two classes. Then it classifies the new point depending on whether it lies on the positive or negative side of the hyperplane depending on the classes to predict.

Hyperparameters of the Support Vector Machine (SVM) Algorithm

There are a few important parameters of SVM that you should be aware of before proceeding further:

- **Kernel:** A kernel helps us find a hyperplane in the higher dimensional space without increasing the computational cost. Usually, the computational cost will increase if the dimension of the data increases. This increase in dimension is required when we are unable to find a separating hyperplane in a given dimension and are required to move in a higher dimension:



- **Hyperplane:** This is basically a separating line between two data classes in SVM. But in Support Vector Regression, this is the line that will be used to predict the continuous output
- **Decision Boundary:** A decision boundary can be thought of as a demarcation line (for simplification) on one side of which lie positive examples and on the other side lie the negative examples. On this very line, the examples may be classified as either positive or negative. This same concept of SVM will be applied in Support Vector Regression as well

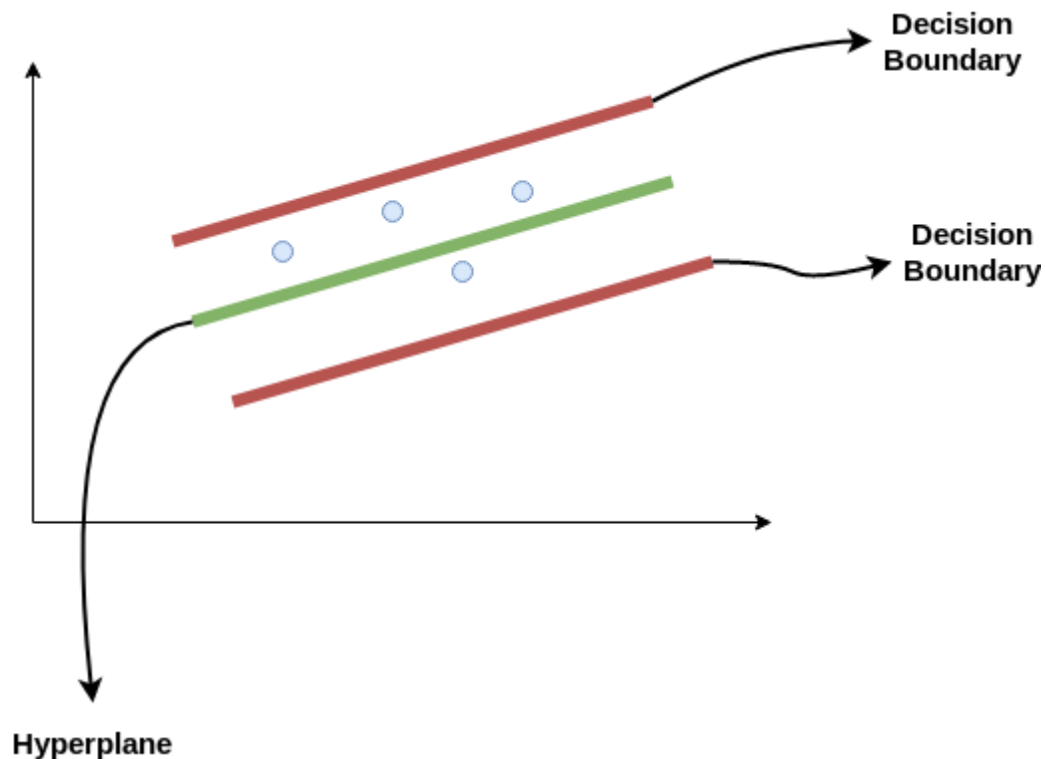
To understand SVM from scratch, I recommend this tutorial: [Understanding Support Vector Machine\(SVM\) algorithm from examples.](#)

✓ Introduction to Support Vector Regression (SVR)

Support Vector Regression (SVR) uses the same principle as SVM, but for regression problems. Let's spend a few minutes understanding the idea behind SVR.

The Idea Behind Support Vector Regression

The problem of regression is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample. So let's now dive deep and understand how SVR works actually.



Consider these two red lines as the decision boundary and the green line as the hyperplane. **Our objective, when we are moving on with SVR, is to basically consider the points that are within the decision boundary line.** Our best fit line is the hyperplane that has a maximum number of points.

The first thing that we'll understand is what is the decision boundary (the danger red line above!). Consider these lines as being at any distance, say 'a', from the hyperplane. So, these are the lines that we draw at distance '+a' and '-a' from the hyperplane. This 'a' in the text is basically referred to as epsilon.

Assuming that the equation of the hyperplane is as follows:

$$Y = wx + b \text{ (equation of hyperplane)}$$

Then the equations of decision boundary become:

$$wx + b = +a$$

$$wx+b = -a$$

Thus, any hyperplane that satisfies our SVR should satisfy:

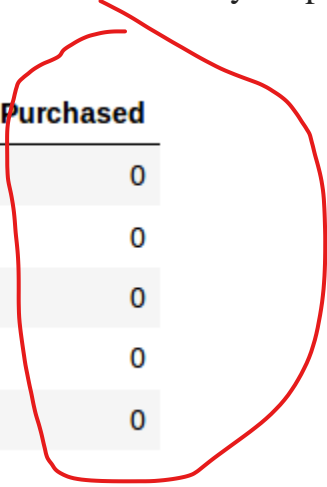
$$-a < Y - wx + b < +a$$

Our main aim here is to decide a decision boundary at 'a' distance from the original hyperplane such that data points closest to the hyperplane or the support vectors are within that boundary line.

Hence, we are going to take only those points that are within the decision boundary and have the least error rate, or are within the Margin of Tolerance. This gives us a better fitting model.

Implementing Support Vector Regression (SVR) in Python

Time to put on our coding hats! In this section, we'll understand the use of Support Vector Regression with the help of a dataset. Here, we have to predict the salary of an employee given a few independent variables. A classic HR analytics project!



	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Step 1: Importing the libraries


```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
view rawsvr1.py hosted with ❤ by GitHub
```

Step 2: Reading the dataset

```
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values
y = dataset.iloc[:, 2].values
view rawsv2.py hosted with ❤ by GitHub
```

Step 3: Feature Scaling

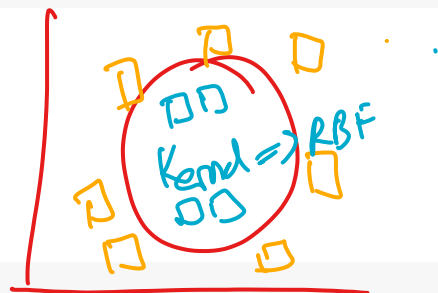
A real-world dataset contains features that vary in magnitudes, units, and range. I would suggest performing normalization when the scale of a feature is irrelevant or misleading.

Feature Scaling basically helps to normalize the data within a particular range. Normally several common class types contain the feature scaling function so that they make feature scaling automatically. However, the SVR class is not a commonly used class type so we should perform feature scaling using Python.

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
view rawsvr3.py hosted with ❤ by GitHub
```

Step 4: Fitting SVR to the dataset

```
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X, y)
view rawsvr4.py hosted with ❤ by GitHub
```



Kernel is the most important feature. There are many types of kernels – linear, Gaussian, etc. Each is used depending on the dataset. To learn more about this, read this: [Support Vector Machine \(SVM\) in Python and R](#)

Step 5. Predicting a new result

```
y_pred = regressor.predict(6.5)
y_pred = sc_y.inverse_transform(y_pred)
```

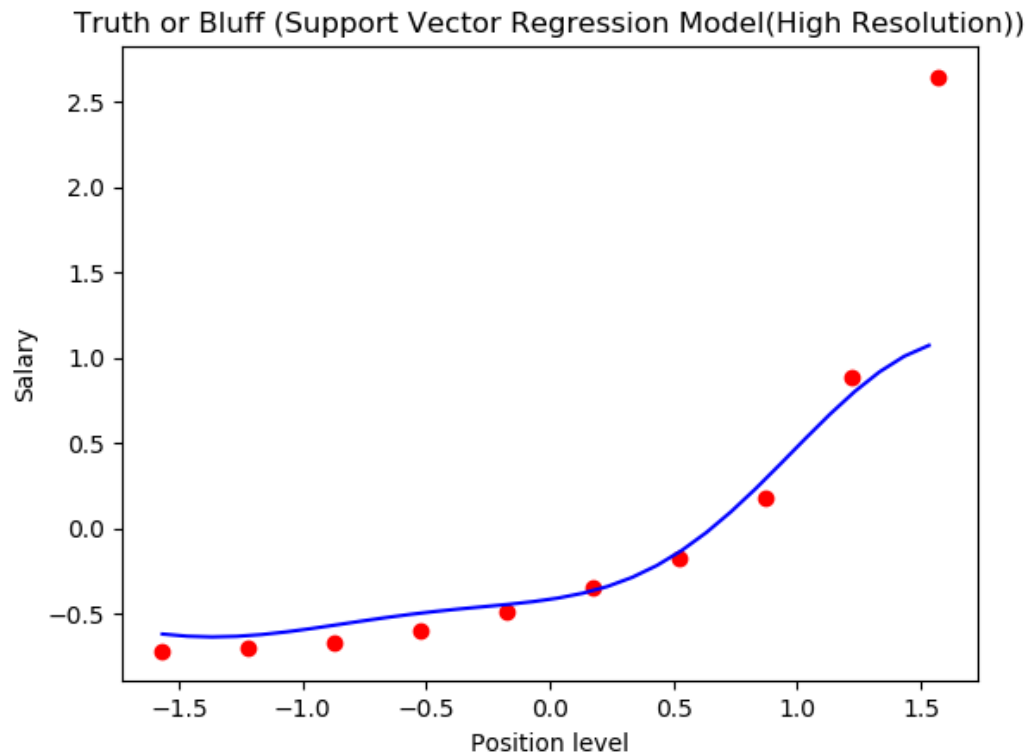
[view rawsvr5.py](#) hosted with ❤ by [GitHub](#)

So, the prediction for $y_{pred}(6, 5)$ will be 170,370.

Step 6. Visualizing the SVR results (for higher resolution and smoother curve)

```
X_grid = np.arange(min(X), max(X), 0.01) #this step required because data is feature
scaled.
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
plt.title('Truth or Bluff (SVR)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

[view rawsvr6.py](#) hosted with ❤ by [GitHub](#)



This is what we get as output- the best fit line that has a maximum number of points.
Quite accurate!

End Notes

We can think of Support Vector Regression as the counterpart of SVM for regression problems. SVR acknowledges the presence of non-linearity in the data and provides a proficient prediction model.