Support Vector Regression Or SVR

In machine learning we use SVM or Support Vector Machine. But SVR is a bit different from SVM. As the name suggest the SVR is a regression algorithm, so we can use SVR for working with continuous Values instead of Classification which is SVM.

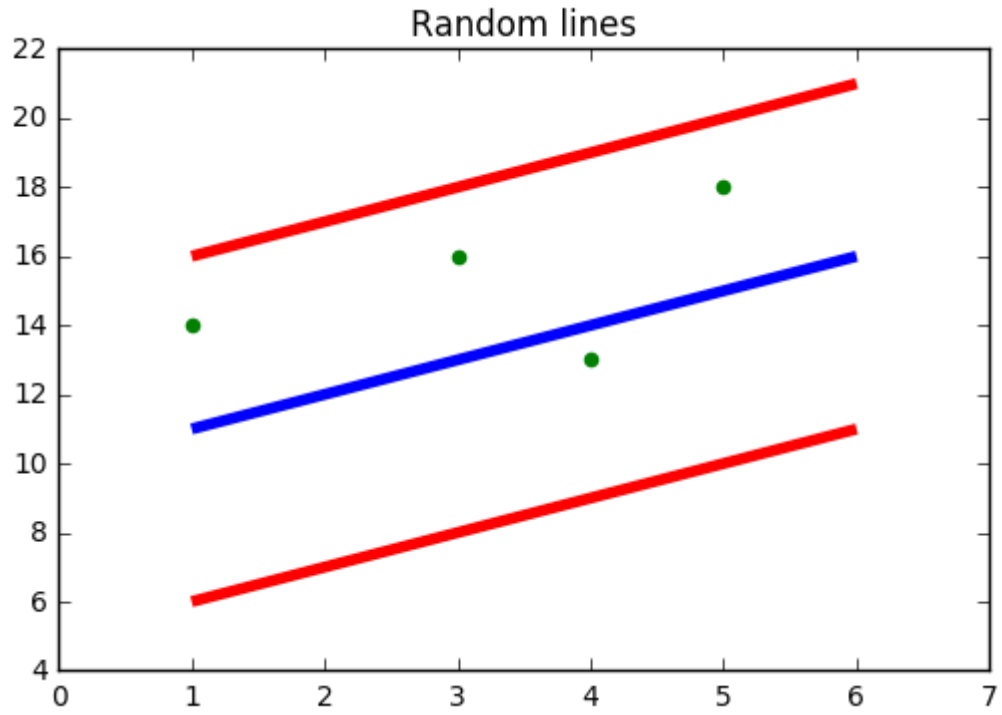
Terminologies used:

1. **Kernel**: The function used to map a lower dimensional data into a higher dimensional data.
2. **Hyper Plane**: In SVM this is basically the separation line between the data classes. Although in SVR we are going to define it as the line that will help us predict the continuous value or target value
3. **Boundary line**: In SVM there are two lines other than Hyper Plane which creates a margin . The support vectors can be on the Boundary lines or outside it. This boundary line separates the two classes. In SVR the concept is same.
4. **Support vectors**: This are the data points which are closest to the boundary. The distance of the points is minimum or least.

Why SVR ? Whats the main difference between SVR and a simple regression model?

In simple regression we try to minimise the error rate. While in SVR we try to fit the error within a certain threshold

Blue line: Hyper Plane; Red Line: Boundary Line



Now see the points

See fig 2 see how all the points are within the boundary line(Red Line). Our objective when we are moving on with SVR is to basically consider the points that are within the boundary line. Our best fit line is the line hyperplane that has maximum number of points.

Explaination:

what is this boundary line ?(yes! that red line). Think of it as to lines which are at a distance of 'e' (though not e its basically epsilon) but for simplicity lets say its **'e'.**

So the lines that we draw are at **'+e'** and **'-e'** distance from Hyper Plane.

Assuming our hyper plane is a straight line going through the Y axis

We can say that the Equation of the hyper plane is

wx+b=o

So we can state that the two the equation of the boundary lines are

Wx+b=+e

Wx+b=-e

respectively

Thus coming in terms with the fact that for any linear hyper plane the equation that satisfy our SVR is:

e≤y-Wx-b≤+e

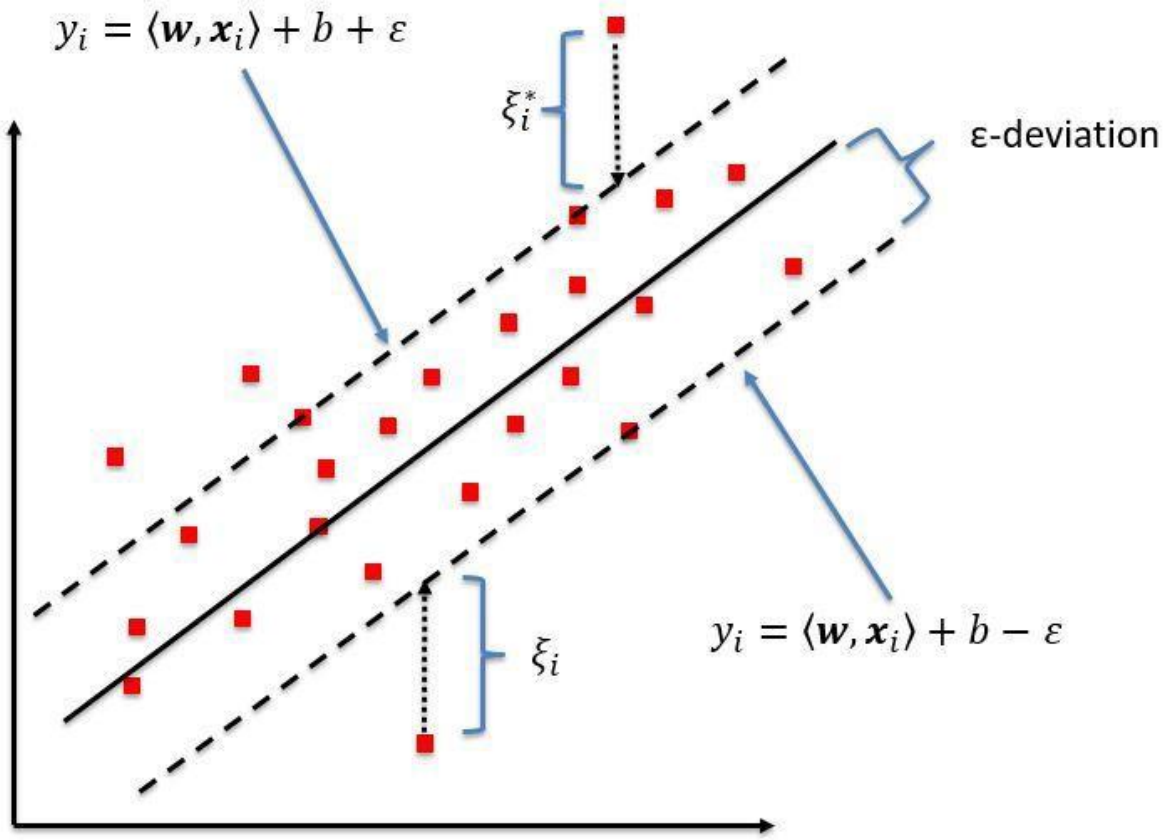
stating the fact that y=Wx+b

y-Wx-b=o

This applies for all other type of regression (non-linear,polynomial)

What we are trying to do here is basically trying to decide a decision boundary at ‘e’ distance from the original hyper plane such that data points closest to the hyper plane or

the support vectors are within that boundary line



Thus the decision boundary is our Margin of tolerance that is We are going to take only those points who are within this boundary.

Or in simple terms that we are going to take only those points which have least error rate. Thus, giving us a better fitting model.

DATA SET DESCRIPTION:

```
Predicting the age of abalone from physical measurements.
Name          Data Type     Meas.  Description
      ----          ---------     -----  -----------
      Sex           nominal                    M, F, and I
(infant)
      Length        continuous    mm     Longest shell measurement
```

```
       Diameter     continuous   mm     perpendicular to length
       Height       continuous   mm     with meat in shell
       Whole weight continuous   grams  whole abalone
       Shucked weight      continuous   grams  weight of meat
       Viscera weight      continuous   grams  gut weight (after
bleeding)
       Shell weight continuous   grams  after being dried
       Rings        integer                  +1.5 gives the age
in years
```

This time lets take it all the way! From Scaling to Feature Selection What say!

import pandas as pd

df=pd.read_csv('./age_mod.csv')

df.head()

df=df.drop(['Sex'],axis=1)

from sklearn.svm import SVR

regressor=SVR(kernel='linear',degree=1)

import matplotlib.pyplot as plt

plt.scatter(df['Shucked weight'],df['Age'])

from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest=train_test_split(x,y)

regressor.fit(xtrain,ytrain)

pred=regressor.predict(xtest)

print(regressor.score(xtest,ytest))

from sklearn.metrics import r2_score

print(r2_score(ytest,pred))

regressor=SVR(kernel='rbf',epsilon=1.0)
regressor.fit(xtrain,ytrain)
pred=regressor.predict(xtest)
print(regressor.score(xtest,ytest))
print(r2_score(ytest,pred))

READ THE CSV FILE

import pandas as pd

df=pd.read_csv('./age_mod.csv')

df.head()

WE DON'T NEED THE 'Sex' COLUMN SO DELETE

df=df.drop(['Sex'],axis=1)

LOADING THE SVR MODEL FROM sklearn.svm

from sklearn.svm import SVR

regressor=SVR(kernel='linear')

#NOTE: kernel='linear' → we are setting the kernel to a linear kernel

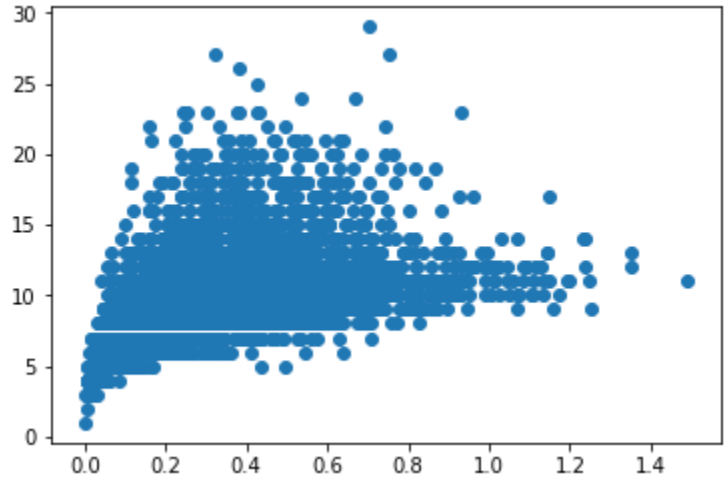
#DEFAULT: kernel='rbf'

PLOT THE RELATION:

import matplotlib.pyplot as plt

plt.scatter(df['Shucked weight'],df['Age'])

#try it for other parameters



SPLIT INTO TRAIN AND TEST SET

from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest=train_test_split(x,y)

FIT THE MODEL DO THE PREDICTION

regressor.fit(xtrain,ytrain)

pred=regressor.predict(xtest)

CHECK THE ACCURACY

print(regressor.score(xtest,ytest))

from sklearn.metrics import r2_score

print(r2_score(ytest,pred))

#NOTE: Both .score() and r2_score gives us a accuracy score prediction

LET'S TUNE SO PARAMETERS TO SEE IF WE CAN GET BETTER SCORE:

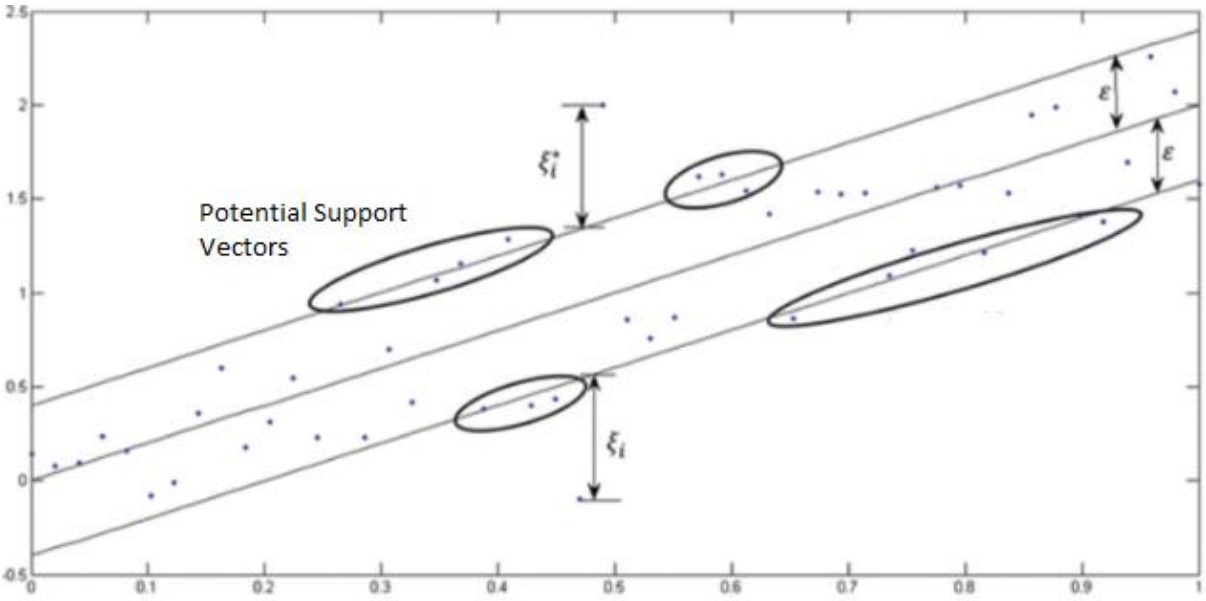
regressor=SVR(kernel='rbf',epsilon=1.0)
regressor.fit(xtrain,ytrain)
pred=regressor.predict(xtest)
print(regressor.score(xtest,ytest))
print(r2_score(ytest,pred))

LOOK HERE:

SVR(kernel='rbf',epsilon=1.0,degree=3)

#here we set the kernel to 'rbf' of degree 3 and a epsilon value of 1.0
#by default the kernel is 'rbf' degree is 3 and epsilon is 0.1
#other kernels are → 'linear','poly'(for polynomial),' rbf'

Support Vector regression is a type of Support vector machine that supports linear and non-linear regression. As it seems in the below graph, the mission is to fit as many instances as possible between the lines while limiting the margin violations. The violation concept in this example represents as ε (epsilon).



SVR requires the training data:{ X, Y} which covers the domain of interest and is accompanied by solutions on that domain. The work of the SVM is to approximate the function we used to generate the training set to reinforce some of the information we've already discussed in a classification problem.

$$F(\vec{X}) = \vec{Y}$$

How to Build a Support Vector Regression Model:

1. Collect a training ƈ = {X,Y}

2. Choose a kernel and parameter and regularization if needed. (Gaussian Kernel and noise regularization are an instance for both steps)

3. Form the correlation matrix:

$$K_{i,j} = \exp\left(\sum_k \theta_k \left|x_k^i - x_k^j\right|^2\right) + \epsilon\delta_{i,j}$$

4. Train your machine, exactly or approximately, to get contraction coefficient by using the main part of the algorithm.

$$\bar{K}\vec{\alpha} = \vec{y}$$

y vector → The vector of the values corresponding to your training dataset

$\vec{\alpha}$ → Set of unknowns we need to solve for

$$\vec{\alpha} = \bar{K}^{-1}\vec{y}$$

5. Use this coefficient to create an estimator.

To estimate the the y^* value for a test $\vec{x^*}$ point, compute the correlation vector $\vec{k}$,

$$y^* = \vec{\alpha} \cdot \vec{k} \qquad k_i = \exp\left(\sum_k \theta_k \left|x_k^i - x_k^\star\right|^2\right)$$

The goal in linear regression is to minimize the error between the prediction and data. In SVR, the goal is to make sure that the errors do not exceed the threshold.

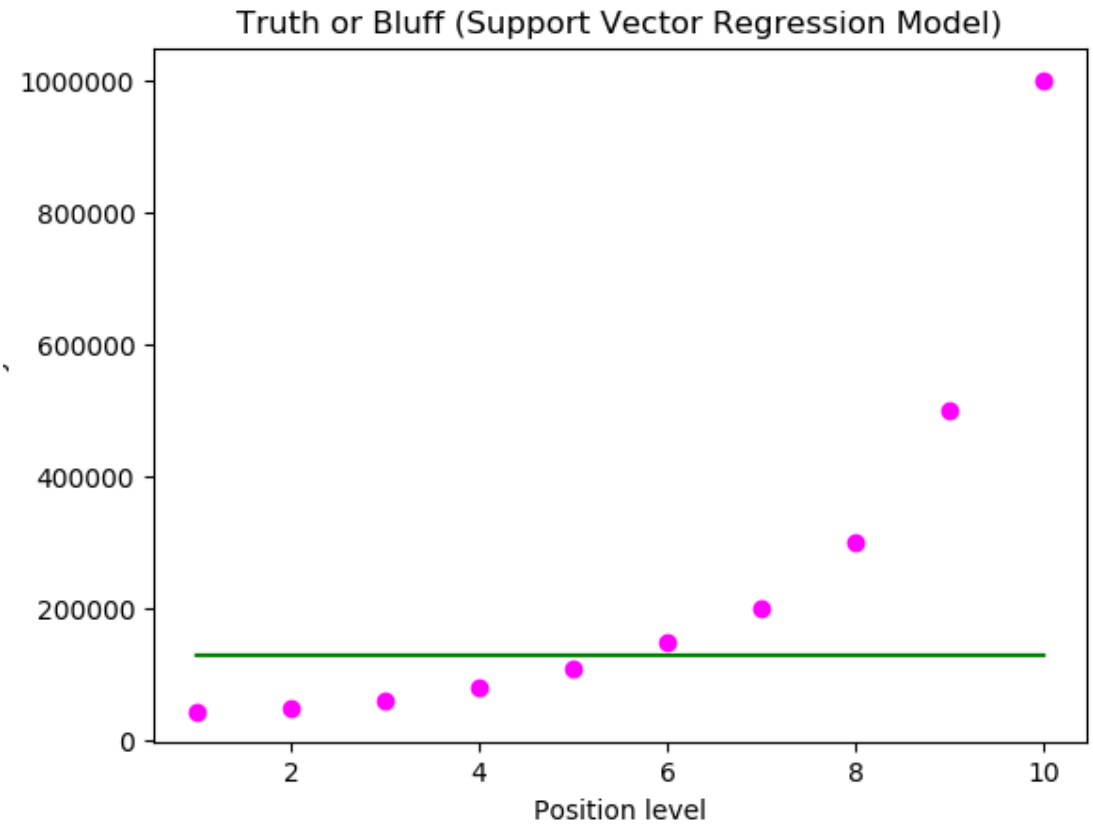
SVR in 6 Steps with Python:

Let's jump to the Python practice on this topic. Here is the link you can reach the dataset for this problem.

```
#1 Importing the librariesimport numpy as np
import matplotlib.pyplot as plt
import pandas as pd#2 Importing the datasetdataset =
pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:,1:2].values.astype(float)
y = dataset.iloc[:,2:3].values.astype(float)#3 Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)#4 Fitting the Support Vector
Regression Model to the dataset
# Create your support vector regressor herefrom sklearn.svm
import SVR# most important SVR parameter is Kernel type. It can
be #linear,polynomial or gaussian SVR. We have a non-linear
condition #so we can select polynomial or gaussian but here we
select RBF(a #gaussian type) kernel.regressor =
SVR(kernel='rbf')
regressor.fit(X,y)#5 Predicting a new result
y_pred = regressor.predict(6.5)
```

The prediction output is 130002. This prediction is not good enough.

```
#6 Visualising the Support Vector Regression
resultsplt.scatter(X, y, color = 'magenta')
plt.plot(X, regressor.predict(X), color = 'green')
plt.title('Truth or Bluff (Support Vector Regression Model)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

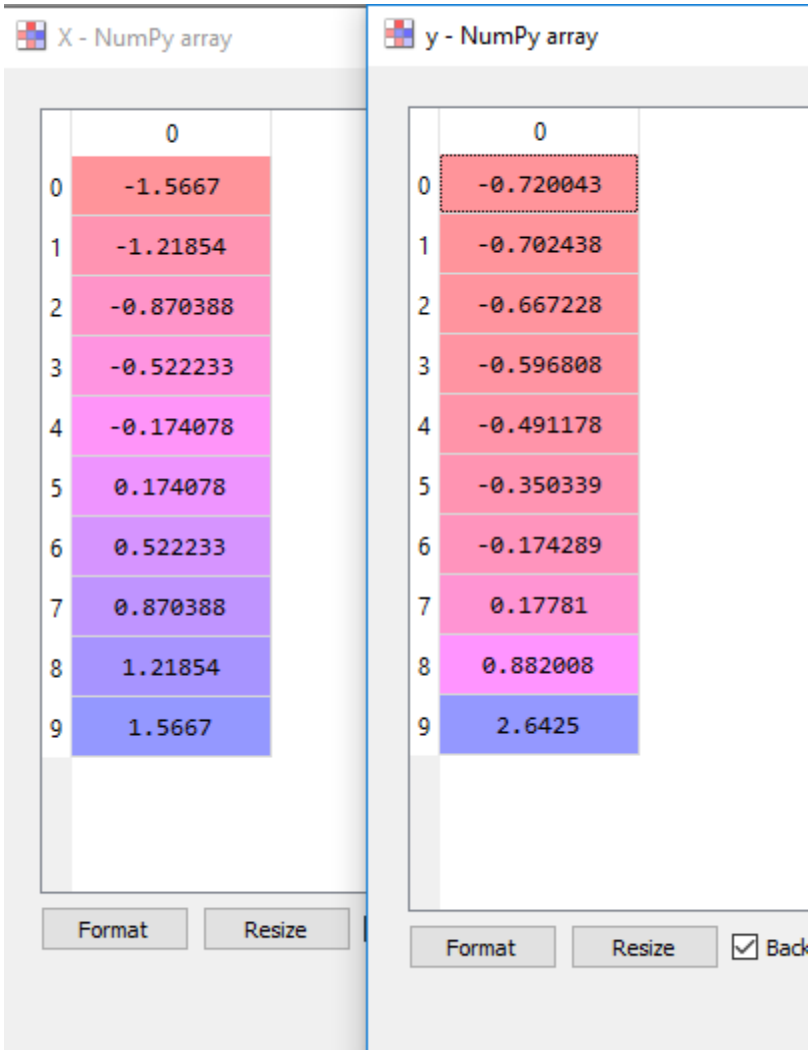


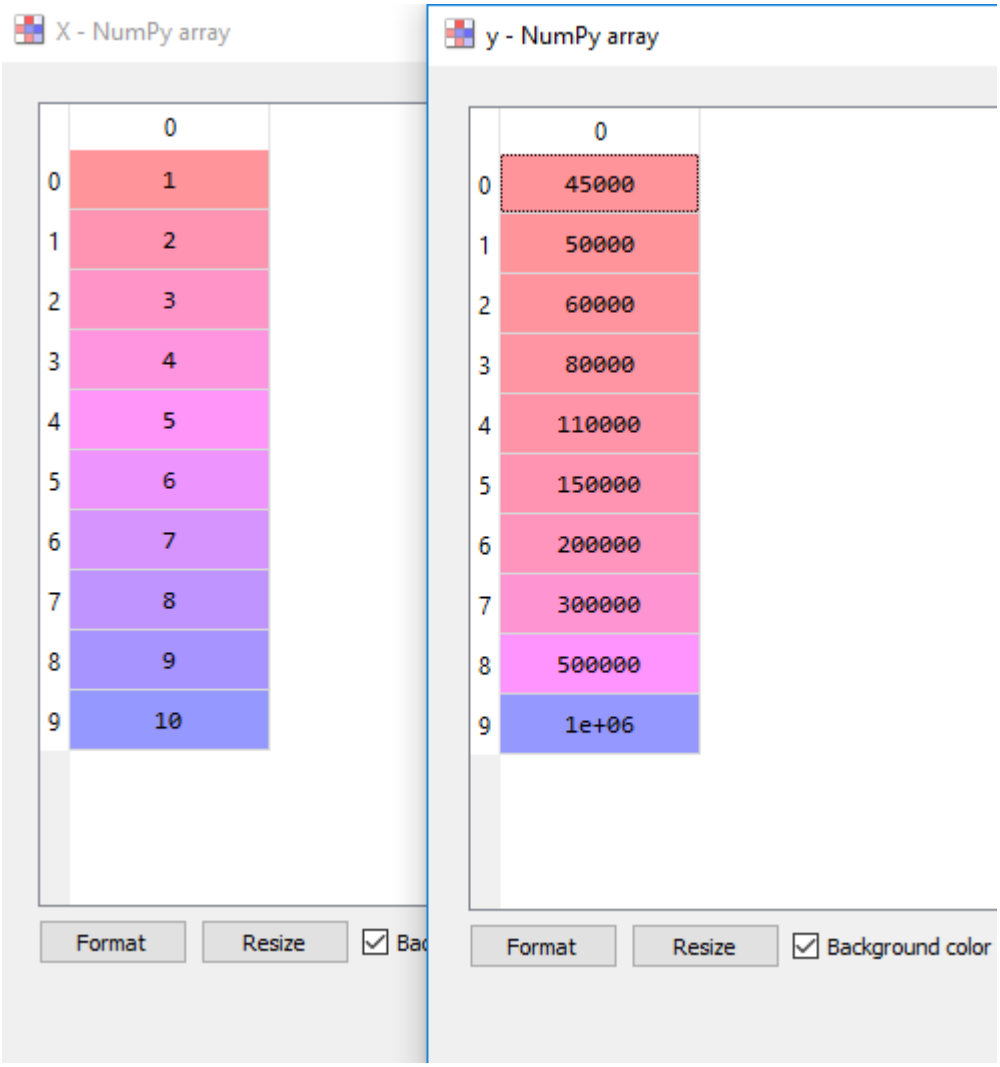
An interesting SVR line

The current problem causes from the unscaled dataset of our practice. 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 that we should make feature scaling by our codes.

Before coding feature scaling line, restart your kernel the Python IDE. Then put your code in the 3rd step of the code.

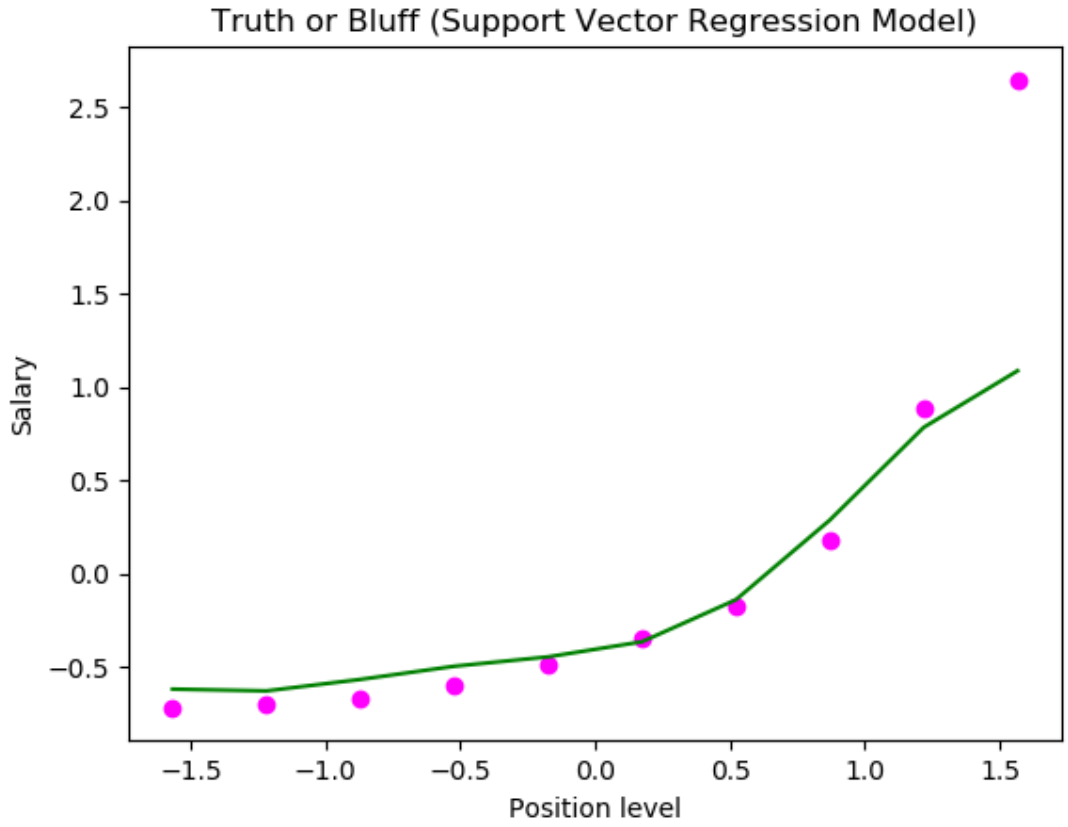
```
#3 Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
```


X - NumPy array
0
0 -1.5667
1 -1.21854
2 -0.870388
3 -0.522233
4 -0.174078
5 0.174078
6 0.522233
7 0.870388
8 1.21854
9 1.5667
Format
Resize
y - NumPy array
0
0 -0.720043
1 -0.702438
2 -0.667228
3 -0.596808
4 -0.491178
5 -0.350339
6 -0.174289
7 0.17781
8 0.882008
9 2.6425
Format
Resize



Compare the before and after the feature scaling

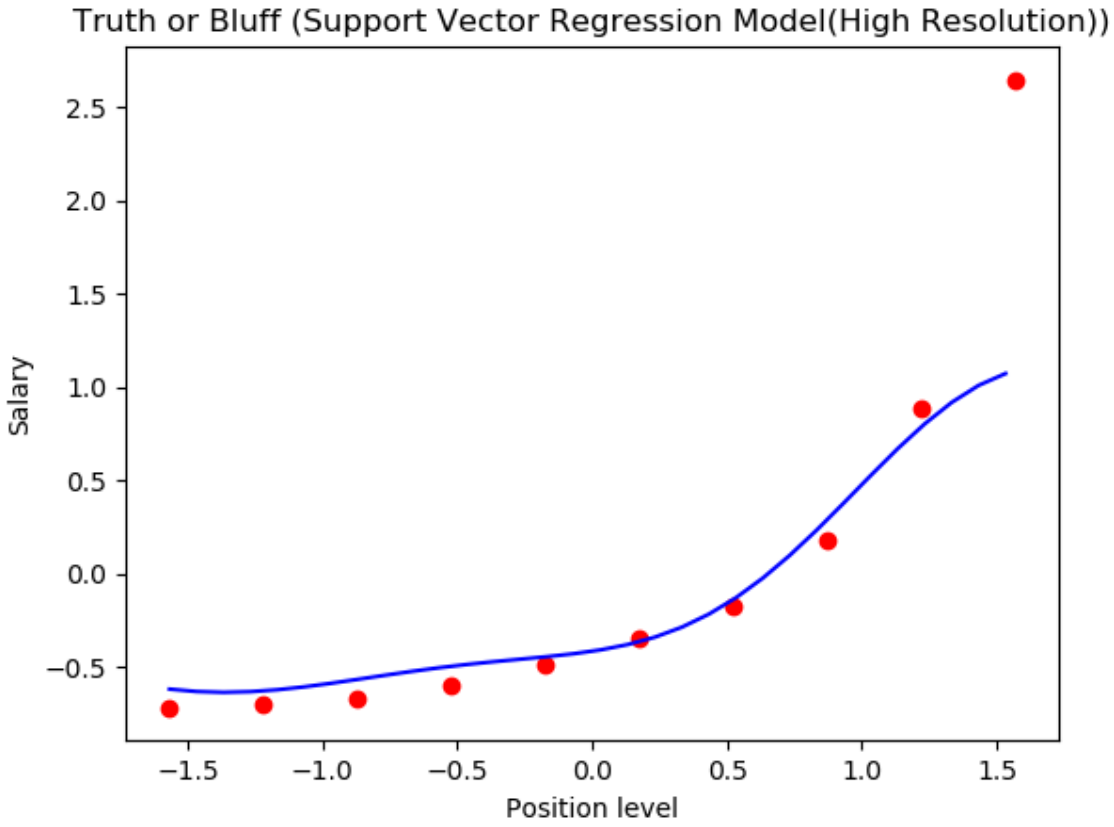
Then restart to run the code again. The final SVR graph will be like below. In this model the point on the far upper right side is CEO and that the model takes this value as an outlier.




```
#5 Predicting a new resulty_pred = sc_y.inverse_transform
((regressor.predict (sc_X.transform(np.array([[6.5]]))))
```

So basically what this means with the two pairs of brackets here is that it's an array of only one line in one column. That is one cell containing this 6.5 numerical value. By using transform and inverse_transform method to convert the feature scaled values into the normal values. So that the prediction for y_pred(6,5) will be 170370. So that it seems more accurate. The last line for our code is for a similar except that higher resolution

```
#6 Visualising the Regression results (for higher resolution and
#smoother curve)X_grid = np.arange(min(X), max(X), 0.1)
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 (Support Vector Regression Model(High
Resolution))')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```


Truth or Bluff (Support Vector Regression Model(High Resolution))
Salary
Position level
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

References :

https://medium.com/coinmonks/support-vector-regression-or-svr-8eb3acf6d0ff