**Unleashing the Power of Non-Linear Predictive Modeling**

[[](https://medium.com/@vk.viswa?source=post_page-----d4495836884--------------------------------)](https://medium.com/@vk.viswa?source=post_page-----d4495836884--------------------------------)

[Viswa](https://medium.com/@vk.viswa?source=post_page-----d4495836884--------------------------------)

·

Follow

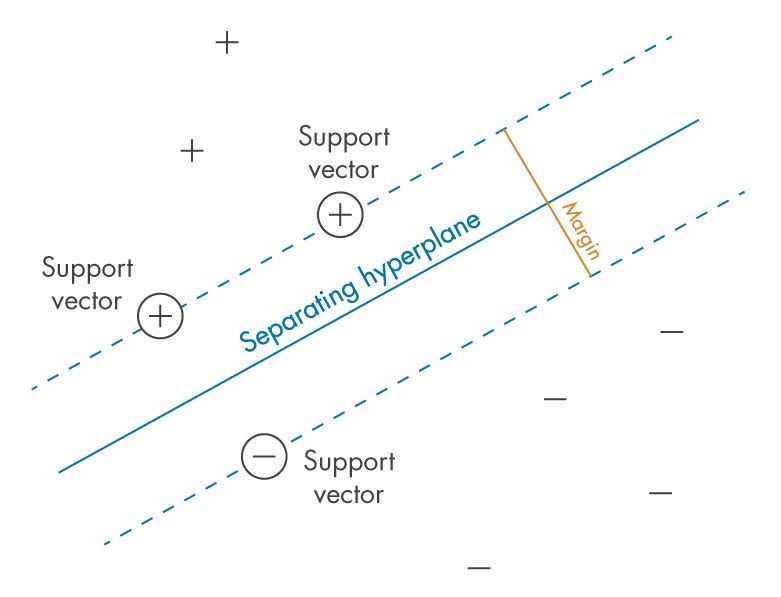
8 min read

·

Jul 28

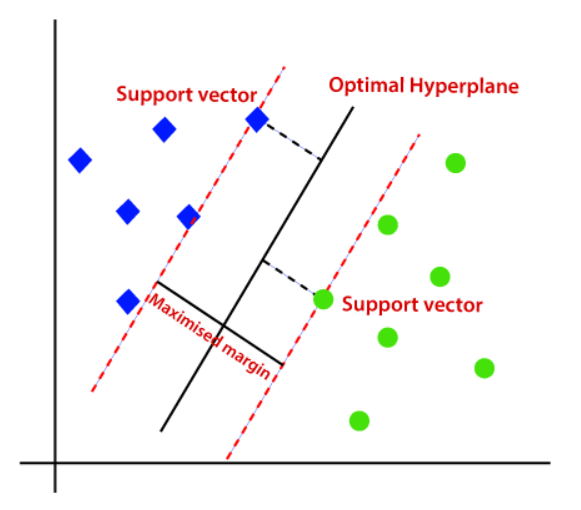
4

Support Vector Regression (SVR) is a powerful machine learning technique used for regression tasks, particularly in scenarios where linear regression may not be sufficient due to complex relationships or non-linear patterns in the data. SVR is an extension of the Support Vector Machine (SVM) algorithm, which is primarily used for classification tasks. SVR’s ability to handle both linear and non-linear data makes it a tool for various real-world applications, including finance, economics, engineering, and more. In this article, we will explore about SVR.



**Table of contents:**

1. Introduction
2. Understanding SVR — Hyperplane Concept
3. Types of Support Vector Regression
4. Implementing Support Vector Regression in Python
5. Practical Example: Building a Predictive Model for Output Power in a Combined Cycle Power Plant
6. Conclusion



**Introduction**

At its core, SVR aims to find a function that maps input features to corresponding output values, making it a regression task. Unlike traditional linear regression, SVR allows for the identification of non-linear relationships by introducing a mapping into a higher-dimensional space through the use of kernel functions. The primary objective of SVR is to minimize the error between the predicted and actual output values while maximizing the margin around the regression line.

**Key Components of Support Vector Regression**

1. **Hyperplane**: In SVR, the hyperplane represents the regression line that best fits the data. However, in contrast to linear regression, SVR allows for a “margin” of tolerance, which is controlled by two hyperparameters: epsilon (ε) and C. Epsilon determines the width of the margin, while C controls the trade-off between maximizing the margin and minimizing the error.

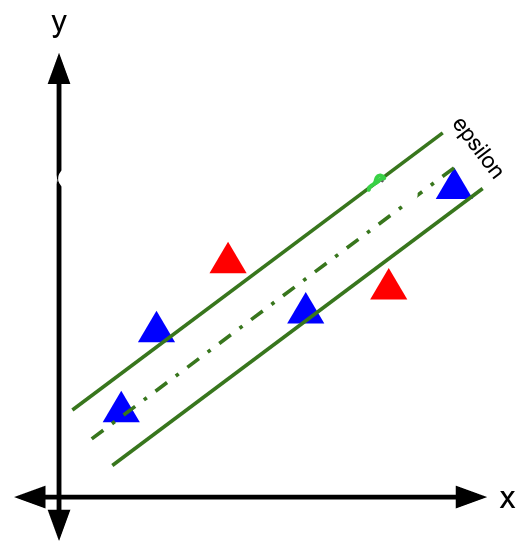
2.**Kernel Functions**: To handle non-linear relationships, SVR uses kernel functions to transform the data into a higher-dimensional space, where linear separation becomes possible. Commonly used kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid.

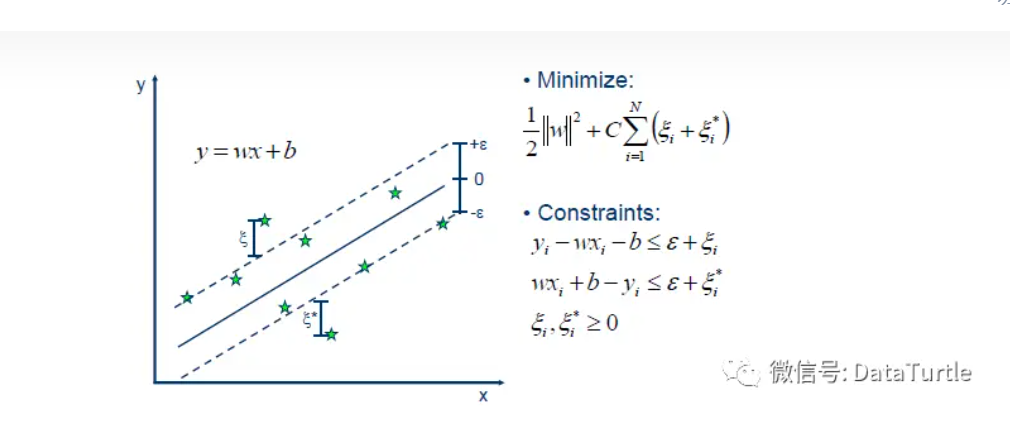
3. **Support Vectors**: These are the data points closest to the hyperplane and play a vital role in defining the SVR model. Only support vectors significantly affect the model, while other data points have little or no impact.

**Types of Support Vector Regression**

There are mainly three types of SVR models:

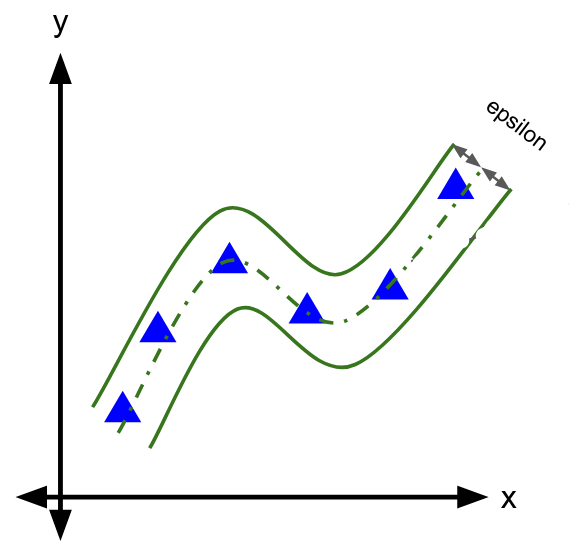
1. **Linear SVR**: This type of SVR uses a linear kernel function and aims to find a linear hyperplane that best fits the data.





The residuals of the data points in between the dashed lines are ignored. Epsilon is a hyperparameter. We want to find an orientation of the separating plane to minimize the residuals above or below the dashed lines. (Yungho)

2. **Nonlinear SVR**: Nonlinear SVR employs various kernel functions (e.g., polynomial, Gaussian radial basis function) to capture complex nonlinear relationships between variables.



3. **Epsilon-Support Vector Regression**: Epsilon-SVR introduces an additional parameter called epsilon, which controls the width of the margin and allows for a certain tolerance of errors.

**Advantages of Support Vector Regression**

1. **Robustness to Outliers**: SVR is less sensitive to outliers compared to traditional regression techniques. The margin around the hyperplane is determined by support vectors, and outliers are often not considered as support vectors, making the model more resilient to noisy data.

2. **Flexibility in Handling Non-Linear Data**: The ability to use different kernel functions allows SVR to capture complex, non-linear relationships in the data. This flexibility makes it suitable for a wide range of regression tasks with varying complexities.

3. **Regularization**: SVR incorporates regularization through the C parameter, preventing overfitting and promoting a more generalizable model.

**Support Vector Regression Practical**

This is the section where you’ll find out how to perform the support vector regression in Python.

We will analyze data from a combined cycle power plant to attempt to build a predictive model for output power.

**Step 1: Importing Python Libraries**

The first step is to start your Jupyter notebook and load all the prerequisite libraries in your Jupyter notebook. Here are the important libraries that we will be needing for this linear regression.

* **NumPy**(to perform certain mathematical operations)
* **pandas**(to store the data in a pandas Data Frames)
* **matplotlib.pyplot** (you will use matplotlib to plot the data)

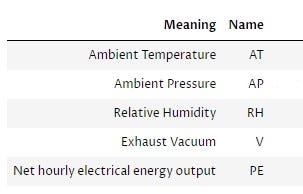
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt

**Step 2: Loading the Dataset**

Let us now import data into a DataFrame. A DataFrame is a data type in Python. The simplest way to understand it would be that it stores all your data in tabular format.

df = pd.read\_csv('Data[1].csv')  
df.head()  
X = df.iloc[:,:-1].values  
y = df.iloc[:,-1].values





**Step 3 : Splitting the dataset into the Training and Test set**

from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.25,random\_state=42)

This line imports the function train\_test\_split from the sklearn.model\_selection module. This module provides various methods for splitting data into subsets for model training, evaluation, and validation.

Here, X and y represent your input features and corresponding target values, respectively. The test\_size parameter specifies the proportion of the data that should be allocated for testing. In this case, test\_size=0.25 means that 25% of the data will be reserved for testing, while the remaining 75% will be used for training.

The random\_state parameter is an optional argument that allows you to set a seed value for the random number generator. By providing a specific random\_state value (e.g., random\_state=42), you ensure that the data is split in a reproducible manner

The train\_test\_split function returns four separate arrays: X\_train, X\_test, y\_train, and y\_test. X\_train and y\_train represent the training data, while X\_test and y\_test represent the testing data.

**Step 4 : Feature Scaling**

from sklearn.preprocessing import StandardScaler  
sc\_x = StandardScaler()  
sc\_y = StandardScaler()  
X\_train = sc\_x.fit\_transform(X\_train)  
y\_train = sc\_y.fit\_transform(y\_train)

Standardization is a common preprocessing technique used in machine learning to transform data into a standard scale. The scikit-learn library provides a StandardScaler class that performs standardization on numerical data.

In the code snippet you provided, the StandardScaler class is used to standardize both the input features (X\_train) and the target variable (y\_train).

**Standardization**involves subtracting the mean and dividing by the standard deviation for each feature. This process ensures that all features have a mean of zero and a standard deviation of one. Standardizing the data can be beneficial for certain machine learning models and algorithms as it helps to bring the features to a comparable scale and prevents any single feature from dominating the learning process.

By creating an instance of the StandardScaler class, such as sc\_x for features and sc\_y for the target variable, you can fit the scalers to the training data using the fit\_transform() method. This method calculates the mean and standard deviation for each feature or the target variable and applies the standardization formula.

Once the standardization is applied, the transformed features and target variable are stored back into X\_train and y\_train , respectively. The standardized data can now be used for training machine learning models.

**Step 5: Training the Support vector regression model on the Training set**

from sklearn.svm import SVR  
regressor = SVR(kernel = 'rbf')  
regressor.fit(X\_train,y\_train)

The first line of code imports the SVR class from the sklearn.svm module, which provides implementation for Support Vector Regression.

Next, you create an instance of the SVR class and assign it to the variable regrssor. The kernel parameter is set to “rbf”, indicating that you want to use the radial basis function (RBF) kernel for the SVR model. The RBF kernel is a popular choice for SVR as it is capable of modeling complex nonlinear relationships between the features and the target variable.

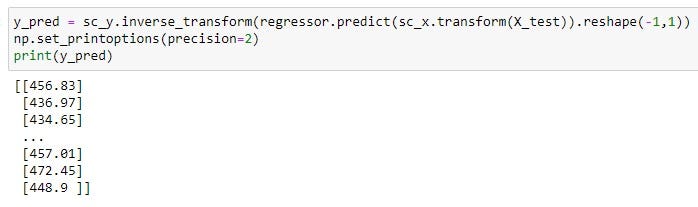
The fit() method of regressor is then called, with X\_train (the input features) and y\_train (the target variable) as the arguments. This method trains the SVR model on the provided training data, using the RBF kernel to capture the underlying patterns and relationships between the features and the target variable.

During the training process, the SVR model will determine the support vectors and learn the optimal hyperplane that best fits the training data, aiming to minimize the error between the predicted and actual target values.

Once the fit() method completes, the SVR model (regressor) will have learned from the training data and be ready to make predictions on new, unseen data.

**Step 6: Predicting the Test set results**

y\_pred = sc\_y.inverse\_transform(regressor.predict(sc\_x.transform(X\_test)).reshape(-1,1))  
np.set\_printoptions(precision=2)  
print(y\_pred)



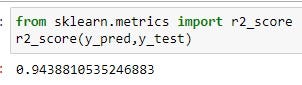
The transform method of sc\_x is applied to X\_test to standardize the new data using the same scaling factors computed on the training data. Then, the predict() method of regressor is used to make predictions on the standardized X\_test.

The predicted target variable (y\_pred) is passed to the inverse\_transform() method of sc\_y. This method undoes the standardization performed on the training target variable, bringing the predicted values back to their original scale. The reshape(-1,1)is used to reshape the predicted values to a column vector.

The code ensures that the predicted values are transformed back to their original scale using the inverse transformation of the target variable’s standard scaler (sc\_y). This is important to provide predictions in the original units of the target variable rather than the standardized values.

**Step 6 : Evaluating the Model Performance**

from sklearn.metrics import r2\_score  
r2\_score(y\_pred,y\_test)



This code imports the r2\_score function from scikit-learn's metrics module. The r2\_score function is commonly used as an evaluation metric for regression models, including linear regression. It measures the proportion of the variance in the target variable that is predictable from the input features.

A higher R-squared score indicates a better fit of the regression model to the data, where 1 represents a perfect fit and 0 represents no relationship between the predicted and actual values.

An R-squared score of 0.9438 indicates that approximately 94.38% of the variance in the target variable can be explained by the predictions of the SVR model. This suggests a strong fit of the model to the data.

Link to the code:

**[Regression/Support\_Vector\_Regression.ipynb at main · ViswaKiranAndraju/Regression](https://github.com/ViswaKiranAndraju/Regression/blob/main/Support_Vector_Regression.ipynb?source=post_page-----d4495836884--------------------------------" \t "_blank)**

[All codes of regression module. Contribute to ViswaKiranAndraju/Regression development by creating an account on…](https://github.com/ViswaKiranAndraju/Regression/blob/main/Support_Vector_Regression.ipynb?source=post_page-----d4495836884--------------------------------" \t "_blank)

[github.com](https://github.com/ViswaKiranAndraju/Regression/blob/main/Support_Vector_Regression.ipynb?source=post_page-----d4495836884--------------------------------" \t "_blank)

**Conclusion**

Support Vector Regression is a valuable tool in predictive modeling, particularly when dealing with complex and non-linear datasets. By introducing a margin of tolerance and employing kernel functions, SVR can handle diverse real-world regression problems effectively. Its robustness to outliers, flexibility in handling non-linear data, and guarantee of a global optimum make it a preferred choice for many researchers and practitioners. As the field of machine learning continues to evolve, SVR remains an essential technique in the data scientist’s toolkit for accurate and reliable regression analysis.