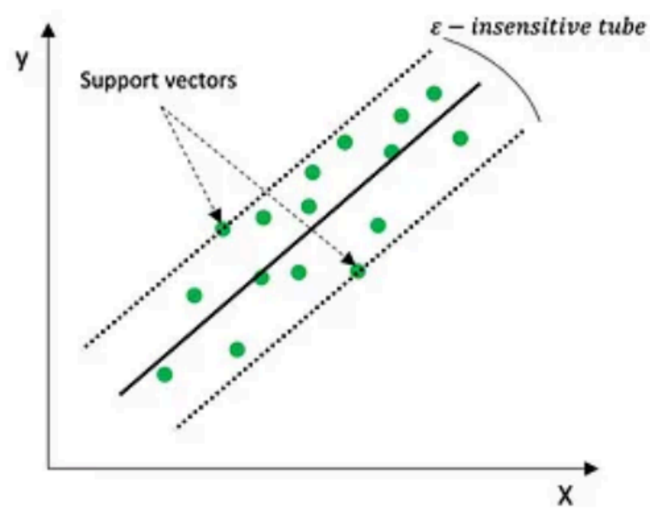
Support Vector Regression

- Used for **regression tasks.**
- Useful when you want to model **non-linear relationships** in your data.



Regression problem using SVR

Key Concept of SVR

Instead of classifying data points like SVM, **SVR fits a regression line** within a **margin of tolerance** (epsilon, **ε**).

- The goal is to **ignore small errors** (less than ε) and focus only on large deviations.
- The **model does not care about small errors**, unlike traditional regression models that minimize Mean Squared Error (MSE).
- Uses **support vectors** to determine the final regression function.
- SVR aims to find a function f(x) that approximates the relationship between the input features *xx* and the target variable *y*, while minimizing the prediction

error.

Mathematical Formulation

SVR finds a function **f(x) = w·x + b** such that most data points lie **within ε of the true values**.

It **minimizes** the following cost function:

$$\min \frac{1}{2}||w||^2 + C\sum(\xi + \xi^*)$$

Symbol	Meaning
w	Weight vector (defines regression line)
b	Bias term
C	Regularization parameter (controls trade-off between error and margin)
ε (epsilon)	Tolerance margin (only points outside this margin are penalized)
ξ, ξ*	Slack variables (for handling points outside the margin)

How SVR Works

1. Define a **margin of tolerance (ε)** around the regression line.
2. Ignore points **inside the margin** (they are not penalized).
3. Penalize points **outside the margin** using **slack variables ξ**.
4. Optimize the function **to maximize margin** and **minimize error**.

Types of SVR Kernels

Like SVM, SVR can use **different kernels**:

- **Linear SVR:** Simple linear relationship.
- **Polynomial SVR:** Captures non-linearity with polynomial features.

- **Radial Basis Function (RBF) SVR:** Most commonly used for complex, non-linear relationships.

```
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Example dataset
X = np.sort(5 * np.random.rand(100, 1), axis=0)
y = np.sin(X).ravel()

# Add noise to the target
y[::5] += 3 * (0.5 - np.random.rand(20))

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the SVR model
svr = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)
svr.fit(X_train, y_train)

# Make predictions
y_pred = svr.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print("R2 score: ",r2_score(y_test, y_pred))

Output:
```

```
Mean Squared Error: 0.18746631319423807
R2 score:  0.7263727663546142
```

Using GridSearchCV:

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np


# Example dataset
X = np.sort(5 * np.random.rand(100, 1), axis=0)
y = np.sin(X).ravel()
# Add noise to the target
y[::5] += 3 * (0.5 - np.random.rand(20))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svr = SVR()


parameters= {'C':[10,30,50,70,100,150,200,250,300], 'gamma':[0.1,0.01,1,1.1,1.5,2],'epsilon':[0.1,0.01,1,1.1,1.5,2]}

GSS = GridSearchCV(svr, parameters, scoring='r2', cv=10)
GSS.fit(X_train, y_train)

print(GSS.best_params_)
print(GSS.best_score_)
```

```
{'C': 50, 'epsilon': 0.1, 'gamma': 1}
0.7873348359865278
```