Support Vector Regression using Scikit-Learn

Project Description

This project demonstrates the use of Support Vector Regression (SVR) for predicting continuous target variables. SVR is an extension of Support Vector Machines (SVM) tailored for regression tasks, aiming to find a function that deviates from the actual observed values by a value no greater than a specified margin (epsilon). It is particularly effective in high-dimensional spaces and when the relationship between features and the target is non-linear.

Why Support Vector Regression?

- **Flexibility:** SVR can model complex, non-linear relationships using various kernel functions.
- **Robustness:** By introducing a margin of tolerance (epsilon), SVR can handle outliers and prevent overfitting.
- **Versatility:** Effective in high-dimensional spaces and applicable to various regression problems.

Prerequisites

Required Libraries

- **Python 3.7 or later**
- `pandas` : For data manipulation and analysis.
- `numpy` : For numerical computations.
- `scikit-learn` : For machine learning models and evaluation metrics.
- `matplotlib` : For data visualization.

Installation

Run the following command to install the necessary libraries:

```
pip install pandas numpy scikit-learn matplotlib
```

Files Included

- `your_dataset.csv` : A placeholder dataset (replace with your actual dataset file).
- Python code for Support Vector Regression.

Code Description

Steps in the Code

1. **Dataset Loading:**

```
 import pandas as pd
data = pd.read_csv('your_dataset.csv')
print(data.head())
```

- The dataset is loaded using pandas, and the first few rows are displayed for initial inspection.

2. **Handling Missing Values:**

```
data.fillna(data.mean(), inplace=True)
```

- Missing values in the dataset are filled with the mean of their respective columns to maintain data integrity.

3. **Splitting Features and Target:**

```
 X = data.iloc[:, :-1]  # All columns except the last one as features
y = data.iloc[:, -1]   # The last column as the target
```

- The dataset is divided into features (`X`) and the target variable (`y`).

4. **Splitting into Training and Test Sets:**

```
 from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

- The data is split into training (80%) and testing (20%) sets.

5. **Initializing and Training the SVR Model:**

```
 from sklearn.svm import SVR
svr_model = SVR(kernel='rbf', C=1.0, epsilon=0.1)
svr_model.fit(X_train, y_train)
```

- An SVR model with a **Radial Basis Function (RBF) kernel** is initialized and trained on the training data.

6. **Making Predictions:**

```
 y_pred = svr_model.predict(X_test)
```

- Predictions are made on the test dataset.

7. **Calculating Evaluation Metrics:**

    - **Mean Squared Error (MSE):**

```
 from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
```

- **R^2 Score:**

```
 from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
```

- The model's performance is evaluated using **MSE** and **R^2 score**.

8. **Visualizing Results:**

    - **Scatter Plot of Actual vs. Predicted Values:**

```
 import matplotlib.pyplot as plt
plt.scatter(y_test, y_pred)
plt.title('Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

- **Residual Plot:**

```
 residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.title('Residuals vs Predicted')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```

- Visualizations are created to assess the model's predictions and residuals.

Outputs

Metrics

- **Mean Squared Error (MSE)**
- **R² Score**

Visualizations

- **Actual vs. Predicted Values:** Scatter plot comparing actual target values to the model's predictions.
- **Residuals vs. Predicted Values:** Plot displaying residuals to check for patterns that might indicate issues with the model.

Example Output

```
 Mean Squared Error: 0.25
R² Score: 0.85
```

Note: These values are hypothetical and will vary based on the actual dataset used.

Use Cases

This project is applicable for:

- **Predicting continuous outcomes** where the relationship between features and the target is complex and potentially non-linear.
- **Situations requiring robust models** that can handle outliers and prevent overfitting.
- **Regression tasks in high-dimensional feature spaces.**

Future Enhancements

- **Hyperparameter Tuning:** Experiment with different kernel functions and parameters (`C` , `epsilon` , `gamma`) to optimize model performance.
- **Feature Scaling:** Implement feature scaling techniques to improve model convergence and accuracy.
- **Cross-Validation:** Use cross-validation methods to ensure the model generalizes well to unseen data.