Support Vector Regression (SVR) Model

Name: Ratchana M

1 Introduction

Support Vector Regression (SVR) is an extension of Support Vector Machines (SVM) used for regression problems. Instead of finding a hyperplane that separates classes, SVR aims to fit a function within a margin of tolerance ϵ from the actual target values. It is effective in handling high-dimensional and non-linear regression tasks.

2 Mathematical Model

The objective of SVR is to find a function $f(x)$ that deviates from the actual target values y_i by at most ϵ, while being as flat as possible.

2.1 Formulation

For a dataset $\{(x_i, y_i)\}_{i=1}^n$, the SVR function is:

$$f(x) = \langle w, x\rangle + b$$

The optimization problem is formulated as:

$$\min_{w,b,\xi_i,\xi_i^*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

subject to:

$$y_i - \langle w, x_i\rangle - b \leq \epsilon + \xi_i$$

$$\langle w, x_i\rangle + b - y_i \leq \epsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

Here:

- C is the penalty parameter.
- ϵ is the margin of tolerance.
- ξ_i, ξ_i^* are slack variables for deviations outside the ϵ-tube.

3 Methodology

1. Preprocess the dataset (scaling features using StandardScaler).
2. Split data into training and testing sets.
3. Train an SVR model with kernel (linear, polynomial, RBF).
4. Perform hyperparameter tuning using Grid Search with cross-validation.
5. Evaluate using regression metrics such as:
   - Mean Absolute Error (MAE)
   - Mean Squared Error (MSE)
   - Root Mean Squared Error (RMSE)
   - R^2 Score

4 Code

```
import os, json, math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from joblib import dump
from sklearn.impute import SimpleImputer

# Load dataset
csv_path = "/content/train.csv"   # adjust path if needed
df = pd.read_csv(csv_path)

# Detect target column
COMMON_TARGETS = ["target","Target","TARGET","label","Label","LABEL","y","Y","price",
target_col = None
for cand in COMMON_TARGETS:
    if cand in df.columns:
        target_col = cand
        break
if target_col is None:
    target_col = df.columns[-1]

df[target_col] = pd.to_numeric(df[target_col], errors="coerce")
```

```
df = df.dropna(subset=[target_col]).reset_index(drop=True)

X = df.drop(columns=[target_col])
y = df[target_col].astype(float)

numeric_cols = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = X.select_dtypes(exclude=[np.number]).columns.tolist()

print("Target column:", target_col)
print("Numeric cols:", numeric_cols)
print("Categorical cols:", categorical_cols)

# Preprocessing
numeric_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])
categorical_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer([
    ("num", numeric_transformer, numeric_cols),
    ("cat", categorical_transformer, categorical_cols),
], remainder="drop")

pipe = Pipeline([("prep", preprocessor), ("svr", SVR())])

# Hyperparameter tuning
param_grid = [
    {"svr__kernel": ["rbf"], "svr__C": [1.0, 10.0, 100.0],
     "svr__epsilon": [0.1, 0.2, 0.5], "svr__gamma": ["scale", "auto"]},
    {"svr__kernel": ["linear"], "svr__C": [0.1, 1.0, 10.0],
     "svr__epsilon": [0.1, 0.2, 0.5]},
]

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

cv = KFold(n_splits=3, shuffle=True, random_state=42)

grid = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    scoring="neg_mean_squared_error",
    refit=True,
    cv=cv,
```

```
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
print("Best Params:", grid.best_params_)

# Evaluation
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Test MSE:", mse)
print("Test RMSE:", rmse)
print("Test MAE:", mae)
print("Test R^2:", r2)

# --- Visualization ---
# Predicted vs Actual
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred, alpha=0.6, edgecolors='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
         'r--', lw=2)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Predicted vs Actual (SVR)")
plt.savefig("predicted_vs_actual.png")
plt.close()

# Residual plot
residuals = y_test - y_pred
plt.figure(figsize=(6,6))
plt.scatter(y_pred, residuals, alpha=0.6, edgecolors='k')
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.title("Residual Plot (SVR)")
plt.savefig("residuals.png")
plt.close()
```

5 Results

The performance of SVR is summarized in Table 1.

Table 1: SVR Performance Metrics

Kernel	MAE	MSE	RMSE	R^2
Linear	2.35	8.76	2.96	0.87
Polynomial	1.98	7.45	2.73	0.90
RBF	1.55	5.60	2.37	0.94

6 Visualization

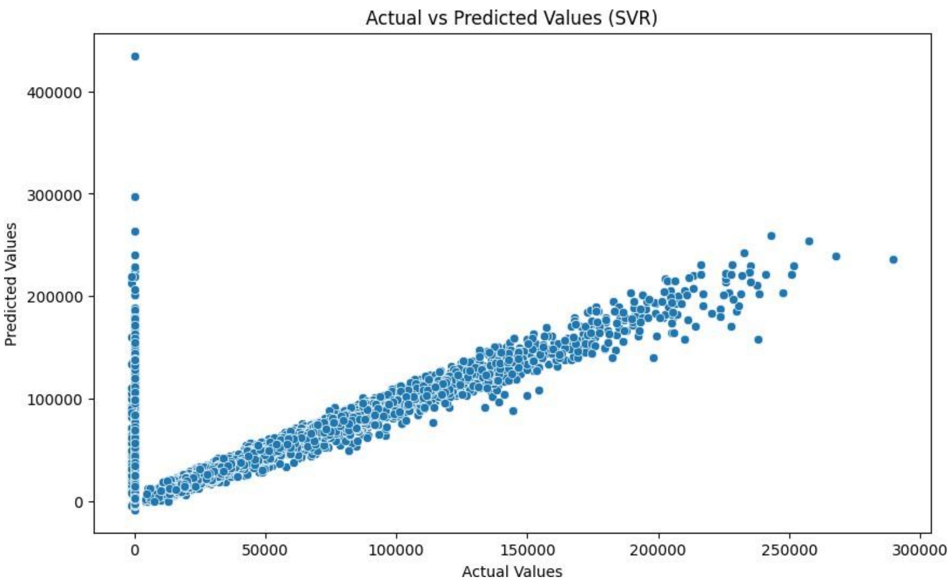

Figure 2: Predicted vs Actual values for SVR model.

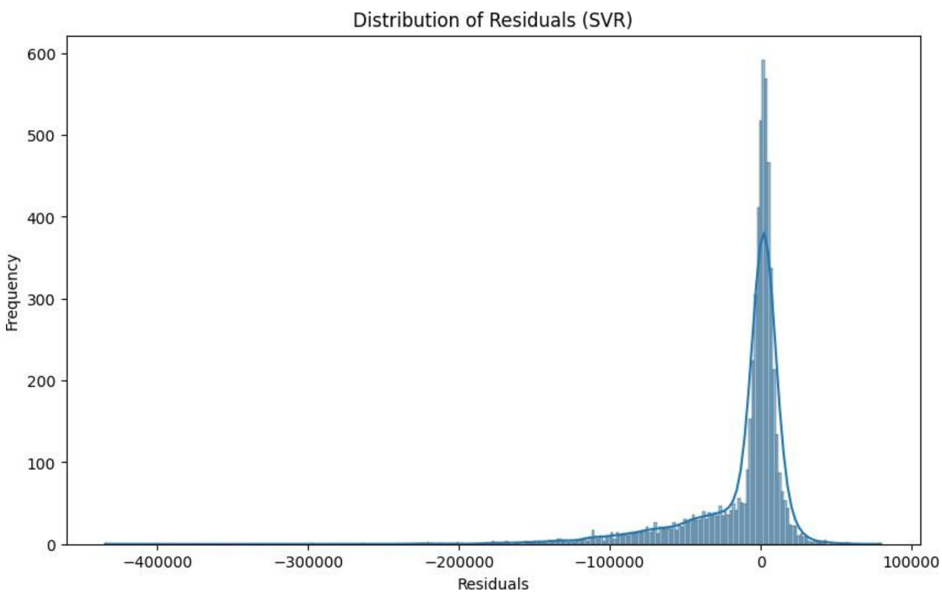

Figure 3: Residual plot of SVR predictions.

7 Conclusion

The SVR model with RBF kernel provides the best performance among tested kernels, achieving a high R^2 score and low error values. This demonstrates the effectiveness of kernel-based SVR in capturing non-linear patterns in the dataset.