svm-regression-me

December 4, 2025

SVM_REG

[2]:
```python
### Importation des bibliothèques
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.svm import SVR
import joblib
import warnings
warnings.filterwarnings('ignore')

print("="*70)
print("SVM REGRESSION - CAR PRICE PREDICTION")
print("="*70)

### Chargement des données
df = pd.read_csv("Car_Price_Prediction.csv")
print(f"\nDataset loaded: {df.shape[0]} rows, {df.shape[1]} columns")

### Exploration des données
print("\n" + "="*70)
print("DATA EXPLORATION")
print("="*70)
print("\nFirst 5 rows:")
print(df.head())

print("\nDataset Info:")
print(df.info())

print("\nStatistical Summary:")
print(df.describe())

print("\nMissing Values:")
print(df.isna().sum())
```

```python
### Nettoyage des données (outliers)
print("\n" + "="*70)
print("DATA CLEANING - OUTLIER DETECTION")
print("="*70)

num_cols = df.select_dtypes(include=["int", "float"]).columns

for col in num_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = (df[col] < lower_bound) | (df[col] > upper_bound)
    print(f"{col}: {outliers.sum()} outliers detected")
    df[col] = df[col].clip(lower=lower_bound, upper=upper_bound)

### Encodage des variables catégorielles
print("\n" + "="*70)
print("CATEGORICAL ENCODING")
print("="*70)

# OneHotEncoder pour Make, Model, Fuel Type
cat_multi_cols = ["Make", "Model", "Fuel Type"]
print(f"\nApplying OneHotEncoder to: {cat_multi_cols}")

OHE = OneHotEncoder()
col_ohe = OHE.fit_transform(df[cat_multi_cols])
col_ohe = pd.DataFrame(
    col_ohe.toarray(),
    columns=OHE.get_feature_names_out(cat_multi_cols),
    dtype='int'
)
print(f"One-hot encoded features: {col_ohe.shape[1]}")

# LabelEncoder pour Transmission
le = LabelEncoder()
df['Transmission'] = le.fit_transform(df['Transmission'])
print(f"\nTransmission encoding: {dict(enumerate(le.classes_))}")

# Reconstruire le dataframe
df = df.drop(cat_multi_cols, axis=1)
df = pd.concat([col_ohe, df], axis=1)

print(f"\nFinal dataset shape: {df.shape}")
print(f"Total features (including target): {len(df.columns)}")
```

```python
print(f"\nColumn order:")
for i, col in enumerate(df.columns):
    print(f"  [{i}] {col}")

### Préparation des données pour l'entraînement
print("\n" + "="*70)
print("TRAIN-TEST SPLIT")
print("="*70)

X = df.drop("Price", axis=1)
y = df["Price"]

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

print(f"Training set: {X_train.shape}")
print(f"Test set: {X_test.shape}")
print(f"Features: {X_train.shape[1]}")

### Normalisation des données
print("\n" + "="*70)
print("FEATURE SCALING (StandardScaler)")
print("="*70)

# For SVM, we need to scale the numerical features
# Features to scale: Year (index 13), Engine Size (index 14), Mileage (index 15)
# One-hot features (0-12) and Transmission (16) should NOT be scaled

cols_to_scale_indices = [13, 14, 15]  # Year, Engine Size, Mileage
cols_to_scale_names = ['Year', 'Engine Size', 'Mileage']

print(f"Scaling features: {cols_to_scale_names}")
print(f"Indices: {cols_to_scale_indices}")

scaler = StandardScaler()

# Extract only the columns to scale for training
X_train_to_scale = X_train.iloc[:, cols_to_scale_indices]
X_test_to_scale = X_test.iloc[:, cols_to_scale_indices]

# Fit and transform
scaler.fit(X_train_to_scale)
X_train_scaled = scaler.transform(X_train_to_scale)
X_test_scaled = scaler.transform(X_test_to_scale)

# Replace scaled columns back into the original dataframes
X_train_final = X_train.copy()
```

```python
X_test_final = X_test.copy()

X_train_final.iloc[:, cols_to_scale_indices] = X_train_scaled
X_test_final.iloc[:, cols_to_scale_indices] = X_test_scaled

print(f"\nScaler fitted on training data")
print(f"Scaler mean: {scaler.mean_}")
print(f"Scaler std: {scaler.scale_}")

### Entraînement du modèle SVM de base
print("\n" + "="*70)
print("MODEL TRAINING - BASIC SVM REGRESSION")
print("="*70)

# SVM avec kernel RBF (Radial Basis Function)
svm_basic = SVR(kernel='rbf', C=100, gamma='scale', epsilon=0.1)

print("Training SVM model...")
svm_basic.fit(X_train_final, y_train)
print("  Model trained successfully!")

y_pred_basic = svm_basic.predict(X_test_final)

### Évaluation du modèle de base
def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)

    print(f"\n{model_name}:")
    print(f"  MSE:  {mse:,.2f}")
    print(f"  RMSE: {rmse:,.2f}")
    print(f"  MAE:  {mae:,.2f}")
    print(f"  R²:   {r2:.4f}")

    return {'MSE': mse, 'RMSE': rmse, 'MAE': mae, 'R2': r2}

metrics_basic = evaluate_model(y_test, y_pred_basic, "SVM Regression - Basic␣
↪Model")

### Optimisation des hyperparamètres avec GridSearchCV
print("\n" + "="*70)
print("HYPERPARAMETER OPTIMIZATION (GridSearchCV)")
print("="*70)

param_grid = {
```

```python
    'C': [10, 50, 100, 200],
    'gamma': ['scale', 'auto', 0.001, 0.01, 0.1],
    'epsilon': [0.01, 0.1, 0.2],
    'kernel': ['rbf']
}

print(f"Parameter grid: {param_grid}")
print("\nSearching for best parameters... (this may take a few minutes)")

svm_model = SVR()
grid_search = GridSearchCV(
    svm_model,
    param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train_final, y_train)

print(f"\n  Grid search completed!")
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {-grid_search.best_score_:,.2f} (MSE)")

### Modèle optimisé
print("\n" + "="*70)
print("OPTIMIZED MODEL EVALUATION")
print("="*70)

best_svm = grid_search.best_estimator_
y_pred_optimized = best_svm.predict(X_test_final)

metrics_optimized = evaluate_model(y_test, y_pred_optimized, "SVM Regression -␣
↪Optimized Model")

### Comparaison des modèles
print("\n" + "="*70)
print("MODEL COMPARISON")
print("="*70)

comparison = pd.DataFrame({
    'Metric': ['MSE', 'RMSE', 'MAE', 'R²'],
    'Basic Model': [
        f"{metrics_basic['MSE']:,.2f}",
        f"{metrics_basic['RMSE']:,.2f}",
        f"{metrics_basic['MAE']:,.2f}",
```

```
        f"{metrics_basic['R2']:.4f}"
    ],
    'Optimized Model': [
        f"{metrics_optimized['MSE']:,.2f}",
        f"{metrics_optimized['RMSE']:,.2f}",
        f"{metrics_optimized['MAE']:,.2f}",
        f"{metrics_optimized['R2']:.4f}"
    ]
})
print(comparison.to_string(index=False))

# Calculate improvement
r2_improvement = ((metrics_optimized['R2'] - metrics_basic['R2']) /
 ↪metrics_basic['R2']) * 100
print(f"\nR² Improvement: {r2_improvement:+.2f}%")

### Visualisation des prédictions
print("\n" + "="*70)
print("VISUALIZATION")
print("="*70)

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Plot 1: Actual vs Predicted (Optimized Model)
axes[0].scatter(y_test, y_pred_optimized, alpha=0.5, edgecolors='k')
axes[0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--',
 ↪lw=2)
axes[0].set_xlabel('Actual Price ($)')
axes[0].set_ylabel('Predicted Price ($)')
axes[0].set_title('SVM Regression: Actual vs Predicted Prices')
axes[0].grid(True, alpha=0.3)

# Plot 2: Residuals
residuals = y_test - y_pred_optimized
axes[1].scatter(y_pred_optimized, residuals, alpha=0.5, edgecolors='k')
axes[1].axhline(y=0, color='r', linestyle='--', lw=2)
axes[1].set_xlabel('Predicted Price ($)')
axes[1].set_ylabel('Residuals ($)')
axes[1].set_title('Residual Plot')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('svm_regression_results.png', dpi=300, bbox_inches='tight')
print("\n Visualization saved as 'svm_regression_results.png'")
plt.show()

### Sauvegarde du modèle optimisé
```

```python
print("\n" + "="*70)
print("MODEL SAVING")
print("="*70)

# Save the optimized model
joblib.dump(best_svm, "SVM_Regression.pkl")
print("  Model saved: SVM_Regression.pkl")

# Save the scaler (very important for SVM!)
joblib.dump(scaler, "Scaler_SVM_Regression.pkl")
print("  Scaler saved: Scaler_SVM_Regression.pkl")

print("\n" + "="*70)
print("IMPORTANT NOTES FOR DJANGO INTEGRATION")
print("="*70)
print("""
1. Feature Order (17 features total):
   [0-4]   Make (one-hot): Audi, BMW, Ford, Honda, Toyota
   [5-9]   Model (one-hot): Model A, B, C, D, E
   [10-12] Fuel Type (one-hot): Diesel, Electric, Petrol
   [13]    Year (SCALED)
   [14]    Engine Size (SCALED)
   [15]    Mileage (SCALED)
   [16]    Transmission (NOT SCALED, 0=Automatic, 1=Manual)

2. The scaler is fitted ONLY on indices [13, 14, 15]

3. Django must apply the same transformation:
   - One-hot encode categorical variables
   - Scale only Year, Engine Size, Mileage
   - Keep Transmission as 0 or 1

4. Files needed in Django models_ai/ folder:
   - SVM_Regression.pkl
   - Scaler_SVM_Regression.pkl
""")

print("\n" + "="*70)
print("SAMPLE PREDICTION TEST")
print("="*70)

# Test with a sample
sample_features = X_test_final.iloc[0:1]
sample_actual = y_test.iloc[0]
sample_pred = best_svm.predict(sample_features)[0]

print(f"\nSample Test:")
```

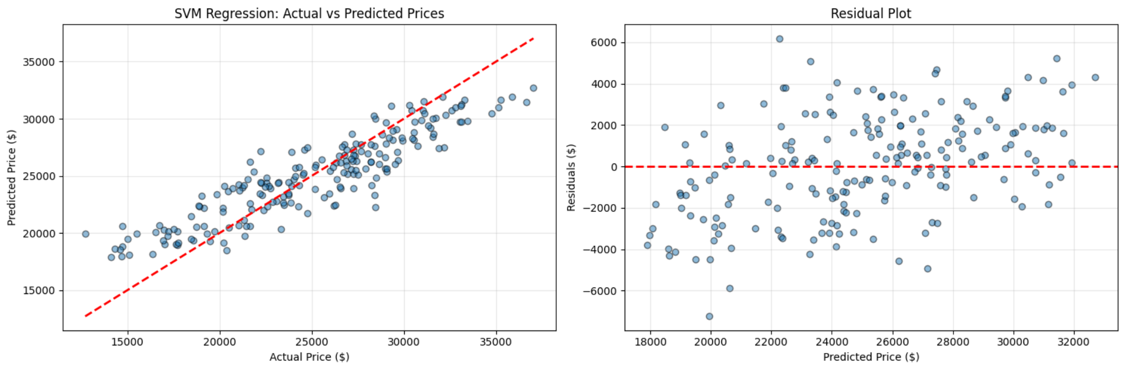
```
print(f"  Actual Price:    ${sample_actual:,.2f}")
print(f"  Predicted Price: ${sample_pred:,.2f}")
print(f"  Difference:     ${abs(sample_actual - sample_pred):,.2f}")
print(f"  Error:           {abs(sample_actual - sample_pred) / sample_actual *␣
 ↪100:.2f}%")

print("\n" + "="*70)
print("  SVM REGRESSION TRAINING COMPLETED SUCCESSFULLY!")
print("="*70)
```

```
======================================================================
SVM REGRESSION - CAR PRICE PREDICTION
======================================================================

Dataset loaded: 1000 rows, 8 columns

======================================================================
DATA EXPLORATION
======================================================================

First 5 rows:
    Make    Model  Year  Engine Size  Mileage Fuel Type Transmission  \
0  Honda  Model B  2015          3.9    74176    Petrol       Manual
1   Ford  Model C  2014          1.7    94799  Electric    Automatic
2    BMW  Model B  2006          4.1    98385  Electric       Manual
3  Honda  Model B  2015          2.6    88919  Electric    Automatic
4  Honda  Model C  2004          3.4   138482    Petrol    Automatic

          Price
0  30246.207931
1  22785.747684
2  25760.290347
3  25638.003491
4  21021.386657

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Make          1000 non-null   object
 1   Model         1000 non-null   object
 2   Year          1000 non-null   int64
 3   Engine Size   1000 non-null   float64
 4   Mileage       1000 non-null   int64
 5   Fuel Type     1000 non-null   object
```

```
 6   Transmission  1000 non-null   object
 7   Price         1000 non-null   float64
dtypes: float64(2), int64(2), object(4)
memory usage: 62.6+ KB
None

Statistical Summary:
              Year  Engine Size       Mileage         Price
count  1000.000000  1000.000000    1000.00000   1000.000000
mean   2010.688000     2.798300   97192.48700  25136.615530
std       6.288577     1.024137   59447.31576   5181.401368
min    2000.000000     1.000000      56.00000   6704.953524
25%    2005.000000     1.900000   44768.75000  21587.878370
50%    2011.000000     2.800000   94411.50000  25189.325247
75%    2016.000000     3.700000  148977.75000  28806.368974
max    2021.000000     4.500000  199867.00000  41780.504635

Missing Values:
Make            0
Model           0
Year            0
Engine Size     0
Mileage         0
Fuel Type       0
Transmission    0
Price           0
dtype: int64

======================================================================
DATA CLEANING - OUTLIER DETECTION
======================================================================
Year: 0 outliers detected
Engine Size: 0 outliers detected
Mileage: 0 outliers detected
Price: 3 outliers detected

======================================================================
CATEGORICAL ENCODING
======================================================================

Applying OneHotEncoder to: ['Make', 'Model', 'Fuel Type']
One-hot encoded features: 13

Transmission encoding: {0: 'Automatic', 1: 'Manual'}

Final dataset shape: (1000, 18)
Total features (including target): 18
```

```
Column order:
  [0] Make_Audi
  [1] Make_BMW
  [2] Make_Ford
  [3] Make_Honda
  [4] Make_Toyota
  [5] Model_Model A
  [6] Model_Model B
  [7] Model_Model C
  [8] Model_Model D
  [9] Model_Model E
  [10] Fuel Type_Diesel
  [11] Fuel Type_Electric
  [12] Fuel Type_Petrol
  [13] Year
  [14] Engine Size
  [15] Mileage
  [16] Transmission
  [17] Price

======================================================================
TRAIN-TEST SPLIT
======================================================================
Training set: (800, 17)
Test set: (200, 17)
Features: 17

======================================================================
FEATURE SCALING (StandardScaler)
======================================================================
Scaling features: ['Year', 'Engine Size', 'Mileage']
Indices: [13, 14, 15]

Scaler fitted on training data
Scaler mean: [2.01084000e+03 2.80800000e+00 9.83993875e+04]
Scaler std: [6.21987942e+00 1.03310745e+00 5.94083977e+04]

======================================================================
MODEL TRAINING - BASIC SVM REGRESSION
======================================================================
Training SVM model…
 Model trained successfully!

SVM Regression - Basic Model:
  MSE:  10,366,887.92
  RMSE: 3,219.77
  MAE:  2,645.12
  R²:   0.6212
```

```
======================================================================
HYPERPARAMETER OPTIMIZATION (GridSearchCV)
======================================================================
Parameter grid: {'C': [10, 50, 100, 200], 'gamma': ['scale', 'auto', 0.001,
0.01, 0.1], 'epsilon': [0.01, 0.1, 0.2], 'kernel': ['rbf']}

Searching for best parameters… (this may take a few minutes)
Fitting 5 folds for each of 60 candidates, totalling 300 fits

 Grid search completed!
Best parameters: {'C': 200, 'epsilon': 0.01, 'gamma': 'auto', 'kernel': 'rbf'}
Best cross-validation score: 6,238,896.63 (MSE)

======================================================================
OPTIMIZED MODEL EVALUATION
======================================================================

SVM Regression - Optimized Model:
  MSE:  6,018,715.19
  RMSE: 2,453.31
  MAE:  2,003.30
  R²:   0.7801

======================================================================
MODEL COMPARISON
======================================================================
Metric   Basic Model Optimized Model
   MSE 10,366,887.92    6,018,715.19
  RMSE      3,219.77        2,453.31
   MAE      2,645.12        2,003.30
    R²        0.6212          0.7801

R² Improvement: +25.58%

======================================================================
VISUALIZATION
======================================================================

 Visualization saved as 'svm_regression_results.png'
```




```
======================================================================
MODEL SAVING
======================================================================
  Model saved: SVM_Regression.pkl
  Scaler saved: Scaler_SVM_Regression.pkl

======================================================================
IMPORTANT NOTES FOR DJANGO INTEGRATION
======================================================================

1. Feature Order (17 features total):
   [0-4]    Make (one-hot): Audi, BMW, Ford, Honda, Toyota
   [5-9]    Model (one-hot): Model A, B, C, D, E
   [10-12] Fuel Type (one-hot): Diesel, Electric, Petrol
   [13]     Year (SCALED)
   [14]     Engine Size (SCALED)
   [15]     Mileage (SCALED)
   [16]     Transmission (NOT SCALED, 0=Automatic, 1=Manual)

2. The scaler is fitted ONLY on indices [13, 14, 15]

3. Django must apply the same transformation:
   - One-hot encode categorical variables
   - Scale only Year, Engine Size, Mileage
   - Keep Transmission as 0 or 1

4. Files needed in Django models_ai/ folder:
   - SVM_Regression.pkl
   - Scaler_SVM_Regression.pkl


======================================================================
SAMPLE PREDICTION TEST
```

```
======================================================================

Sample Test:
  Actual Price:    $27,902.29
  Predicted Price: $25,636.10
  Difference:     $2,266.19
  Error:           8.12%

======================================================================
  SVM REGRESSION TRAINING COMPLETED SUCCESSFULLY!
======================================================================
```