Machine Learning for Economics and Finance

03_Auto_data_val_set

Ole Wilms

July 29, 2024

[1]:
```
import os           # Package to access system related information
print(os.getcwd())  # Prints the current working directory
path = os.getcwd()
os.chdir(path)      # Set the working directory

from ISLP import load_data           # Package which contains the data
Auto_data = load_data('Auto')  # Loading the data
Auto_data.head()               # Showing the first 5 Lines of Data.
```

```
/mnt/ds/home/UHH_MLSJ_2024/Code/Python/03-CrossValidation
```

[1]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	\
0	18.0	8	307.0	130	3504	12.0	70	
1	15.0	8	350.0	165	3693	11.5	70	
2	18.0	8	318.0	150	3436	11.0	70	
3	16.0	8	304.0	150	3433	12.0	70	
4	17.0	8	302.0	140	3449	10.5	70	

	origin	name
0	1	chevrolet chevelle malibu
1	1	buick skylark 320
2	1	plymouth satellite
3	1	amc rebel sst
4	1	ford torino

[2]:
```
print(Auto_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    int64
 4   weight        392 non-null    int64
```

```
 5   acceleration  392 non-null    float64
 6   year          392 non-null    int64
 7   origin        392 non-null    int64
 8   name          392 non-null    object
dtypes: float64(3), int64(5), object(1)
memory usage: 27.7+ KB
None
```

[3]:
```
print(Auto_data.describe())
```

```
              mpg   cylinders  displacement  horsepower       weight  \
count  392.000000  392.000000    392.000000  392.000000   392.000000
mean    23.445918    5.471939    194.411990  104.469388  2977.584184
std      7.805007    1.705783    104.644004   38.491160   849.402560
min      9.000000    3.000000     68.000000   46.000000  1613.000000
25%     17.000000    4.000000    105.000000   75.000000  2225.250000
50%     22.750000    4.000000    151.000000   93.500000  2803.500000
75%     29.000000    8.000000    275.750000  126.000000  3614.750000
max     46.600000    8.000000    455.000000  230.000000  5140.000000

       acceleration        year      origin
count    392.000000  392.000000  392.000000
mean      15.541327   75.979592    1.576531
std        2.758864    3.683737    0.805518
min        8.000000   70.000000    1.000000
25%       13.775000   73.000000    1.000000
50%       15.500000   76.000000    1.000000
75%       17.025000   79.000000    2.000000
max       24.800000   82.000000    3.000000
```

[4]:
```
import numpy as np
import pandas as pd

# set seed
np.random.seed(1)

# Number of observations in the dataset
n = len(Auto_data)

# Shuffle the dataset using np.random.permutation
shuffled_indices = np.random.permutation(n)

# Split data into training, validation, and test sets
n = len(Auto_data)
n_train = 150                  # Training set size
n_val = 150                    # Validation set size
n_test = n - n_train - n_val   # Remaining for the test set
```

```
train_data = Auto_data.iloc[:n_train]
val_data = Auto_data.iloc[n_train:n_train + n_val]
test_data = Auto_data.iloc[n_train + n_val:]
```

[5]:
```
from sklearn.metrics import mean_squared_error

# Define a function to compute MSE
def compute_mse(y_true, y_pred):
    return mean_squared_error(y_true, y_pred)
```

[6]:
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Try models with polynomial degrees from 1 to 10
num_models = 10
train_mse = []
val_mse = []
lm_list = []  # List to save all models

for i in range(1, num_models + 1):
    # Create polynomial features
    poly = PolynomialFeatures(degree=i)

    # Training data: Transform horsepower to polynomial features
    X_train_poly = poly.fit_transform(train_data[['horsepower']])
    y_train = train_data['mpg']

    # Validation data: Transform horsepower to polynomial features
    X_val_poly = poly.transform(val_data[['horsepower']])
    y_val = val_data['mpg']

    # Train the model on the training set
    model = LinearRegression().fit(X_train_poly, y_train)
    lm_list.append(model)  # Save the model

    # Make predictions for training and validation sets
    y_train_pred = model.predict(X_train_poly)
    y_val_pred = model.predict(X_val_poly)

    # Compute MSE for training and validation sets
    train_mse.append(compute_mse(y_train, y_train_pred))
    val_mse.append(compute_mse(y_val, y_val_pred))
```

[7]:
```
# Select the best model (with minimum validation MSE)
best_model_index = np.argmin(val_mse) + 1  # Adding 1 to match polynomial degree
print(f"Best model is polynomial degree: {best_model_index}")
```

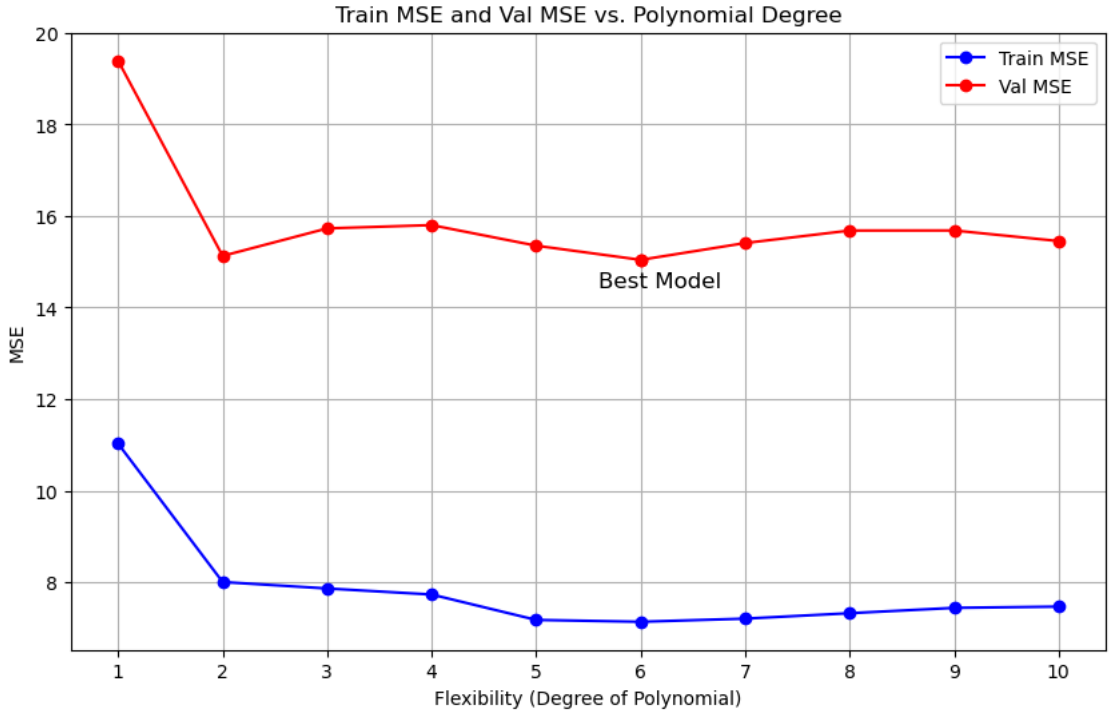
```
Best model is polynomial degree: 6
```

[8]:
```
# Output the MSE values for each model
for degree, train_error, val_error in zip(range(1, num_models + 1), train_mse,␣
 ↪val_mse):
    print(f"Degree {degree}:   Train MSE: {train_error:.5f},  Validation MSE:␣
 ↪{val_error:.5f}")
```

```
Degree 1:   Train MSE: 11.02522,  Validation MSE: 19.39243
Degree 2:   Train MSE: 8.00342,  Validation MSE: 15.12801
Degree 3:   Train MSE: 7.86238,  Validation MSE: 15.72968
Degree 4:   Train MSE: 7.73052,  Validation MSE: 15.80150
Degree 5:   Train MSE: 7.17526,  Validation MSE: 15.35206
Degree 6:   Train MSE: 7.13461,  Validation MSE: 15.04239
Degree 7:   Train MSE: 7.20354,  Validation MSE: 15.40961
Degree 8:   Train MSE: 7.32205,  Validation MSE: 15.68285
Degree 9:   Train MSE: 7.43871,  Validation MSE: 15.68433
Degree 10:   Train MSE: 7.46632,  Validation MSE: 15.45292
```

[9]:
```
import matplotlib.pyplot as plt

# Convert train MSE and validation MSE to a DataFrame
mse_df = pd.DataFrame({
    'Degree': np.arange(1, num_models + 1),
    'Val MSE': val_mse,
    'Train MSE': train_mse
})

# Create a subset with only the best model
mse_best = mse_df[mse_df['Degree'] == best_model_index]

# Plotting Train MSE and Validation MSE
plt.figure(figsize=(10, 6))

# Plot the training MSE
plt.plot(mse_df['Degree'], mse_df['Train MSE'], label="Train MSE",␣
 ↪color='blue', marker='o')
plt.plot(mse_df['Degree'], mse_df['Val MSE'], label="Val MSE", color='red',␣
 ↪marker='o')

# Mark the best model with a red point and annotate it
plt.scatter(mse_best['Degree'], mse_best['Val MSE'], color='red', zorder=5)
plt.text(mse_best['Degree'].values[0] - 0.4, mse_best['Val MSE'].values[0] - 0.
 ↪6,
         'Best Model', color='black', fontsize=12)

# Labels and legends
plt.xlabel('Flexibility (Degree of Polynomial)')
plt.ylabel('MSE')
```

```
plt.legend()
plt.xticks(np.arange(1, num_models + 1, step=1))
plt.title('Train MSE and Val MSE vs. Polynomial Degree')
plt.grid(True)
plt.show()
```



[10]:
```
# Fit the best model again using both the training and validation data
# Combine training and validation sets
train_val_data = pd.concat([train_data, val_data])

# Polynomial features for the best model
poly_best = PolynomialFeatures(degree=best_model_index)
X_train_val_poly = poly_best.fit_transform(train_val_data[['horsepower']])
y_train_val = train_val_data['mpg']

# Train the best model on the combined training + validation data
best_model = LinearRegression().fit(X_train_val_poly, y_train_val)

# Compute Test MSE
X_test_poly = poly_best.transform(test_data[['horsepower']])
y_test = test_data['mpg']
y_test_pred = best_model.predict(X_test_poly)
```

```python
test_mse = mean_squared_error(y_test, y_test_pred)

# Display Training, Validation and Test MSE for the best model
print(f"Train MSE: {round(train_mse[best_model_index-1], 5)}")
print(f"Val MSE:   {round(val_mse[best_model_index-1], 5)}")
print(f"Test MSE:  {round(test_mse, 5)}")
```

```
Train MSE: 7.13461
Val MSE:   15.04239
Test MSE:  57.43518
```