Machine Learning for Economics and Finance

01_Auto_data_2_solution

Ole Wilms

July 29, 2024

Setup for Tasks 2

The following steps will outline the initial setup of the Auto dataset, including the creation of separate training (train data) and test (test data) sets.

Get and Set working directory

[2]:
```
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
```

```
/mnt/ds/home/UHH_MLSJ_2024/Code/Python/01_SupLearn_Regression
```

Loading the package and data

[3]:
```
from ISLP import load_data  # Package which contains the data
Auto = load_data('Auto')    # Loading the data
Auto.head()                 # Showing the first 5 Lines of Data.
```

[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

[5]:
```
n  = int(len(Auto)) # Number of observations in the dataset
nT = int(n/2)       # training sample size
nV = int(n/2)       # validation sample size
```

Define training and test set

[6]:
```
import pandas as pd
import numpy as np

np.random.seed(2)  # set seed

# Define training and test sets
train_sample = np.random.choice(n, nT, replace=False)  # indices for training␣
 ↪data
train_data = Auto.iloc[train_sample]                   # training dataset
test_data = Auto.drop(train_sample)                    # test dataset
```

We start with the following univariate linear regression:

$$\text{mpg} = \beta_0 + \beta_1 \text{horsepower} + \varepsilon$$

Fit model on training data and calculate training MSE

[7]:
```
import statsmodels.formula.api as smf

# fit model on training data and calculate training MSE
fit_lm = smf.ols(formula='mpg ~ horsepower', data = train_data).fit()
```

[9]:
```
print(fit_lm.summary().tables[1])
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     39.0131      0.994     39.245      0.000      37.053      40.974
horsepower    -0.1510      0.009    -17.040      0.000      -0.168      -0.134
==============================================================================
```

[10]:
```
# Predictions for training data
y_head_train = fit_lm.predict(train_data)
```

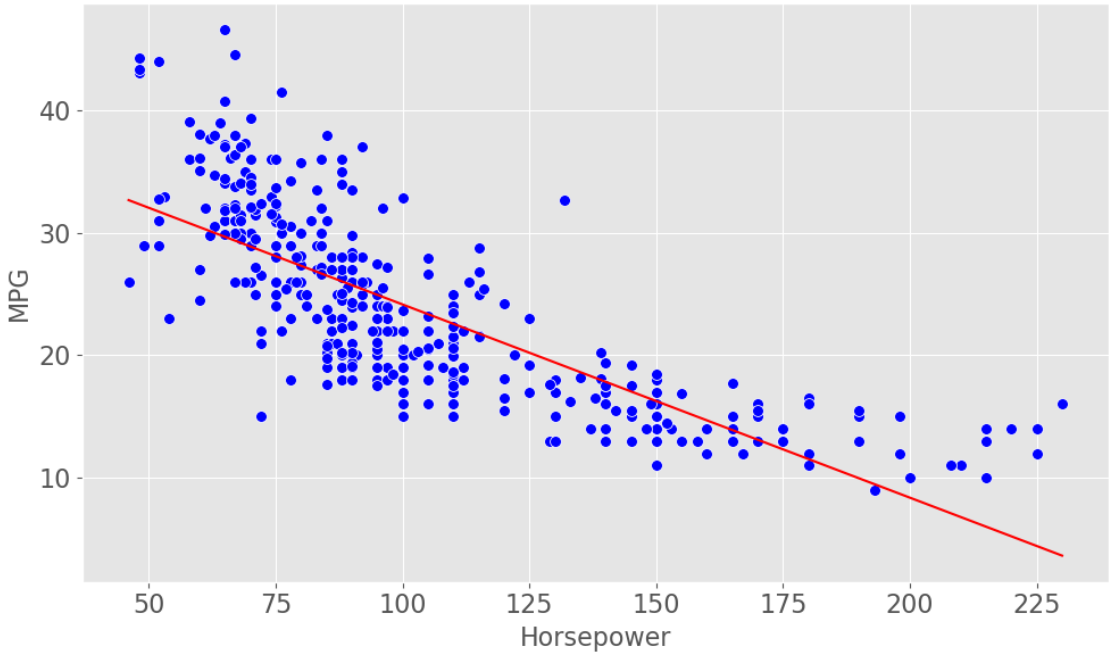
[11]:
```
# Function to compute the mean squared error (MSE)
# Takes realized values y and corresponding predictions y_head
# as inputs and returns MSE as output
def MSE(y, y_head):
  return((y - y_head)**2).mean()

# Compute the mean squared error
MSE_train = MSE(train_data['mpg'], y_head_train)
print(f"Mean Squared Error:  {MSE_train:.3f}")
```

```
Mean Squared Error:  23.002
```

Extra visualisation

[12]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression

# Plotting
plt.figure(figsize=(10, 6), tight_layout=True)
plt.style.use('ggplot')

# Scatter plot of the Auto data
sns.scatterplot(x=Auto['horsepower'], y=Auto['mpg'], color='blue', s=50)

X = Auto[['horsepower']]  # Make sure X is a 2D array
y = Auto['mpg']
lm = LinearRegression().fit(X,y)

# Generate predictions across the range of horsepower values
x_range = np.linspace(Auto['horsepower'].min(), Auto['horsepower'].max(), 100)
x_range_reshaped = x_range.reshape(-1, 1)  # Reshape x_range to a 2D array
y_pred = lm.predict(x_range_reshaped)

# Plot the predicted line together with the data
plt.plot(x_range, y_pred, color='red', label='Fitted line')

# Adjust the text size
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.xlabel('Horsepower', fontsize=16)
plt.ylabel('MPG', fontsize=16)
plt.title('', fontsize=18)

# Show plot
plt.show()
```

```
/usr/lib/python3.12/site-packages/sklearn/base.py:493: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```



Task 2

1. Compute the test MSE for the univariate linear model and note the training and test MSE.

Hint: First copy line 84 and generate predictions $\hat{y}$ for the test data. Then copy line 98 and change it correspondingly to compute the test MSE.

[14]:
```
# Predictions for test data
y_head_test = fit_lm.predict(test_data)
```

[23]:
```
# Compute the "Mean Squared Error"
print(f"Train MSE:       {MSE_train:.3f}")

MSE_test = MSE(test_data['mpg'], y_head_test)
print(f"Validation MSE:  {MSE_test:.3f}")
```

```
Train MSE:       23.002
Validation MSE:  25.109
```

2. Compute and note the training and test MSE for a quadratic model:

$$\text{mpg} = \beta_0 + \beta_1 \text{horsepower} + \beta_2 \text{horsepower}^2 + \varepsilon$$

[19]:
```
# Define polynomial function
def poly(x, degree):
    return np.vander(x, degree + 1, increasing=True)[:, 1:]
```

```
# Quadratic model
fit_lm2 = smf.ols(formula='mpg ~ poly(horsepower, 2)', data = train_data).fit()
print(fit_lm2.summary().tables[1])
```

```
==============================================================================
==========
                               coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
----------
Intercept                   56.0467      2.594     21.610      0.000      50.931
61.162
poly(horsepower, 2)[0]     -0.4566      0.044    -10.278      0.000      -0.544
-0.369
poly(horsepower, 2)[1]      0.0012      0.000      6.992      0.000       0.001
0.002
==============================================================================
==========
```

[20]:
```
# Predictions for test data
y_head_train_2 = fit_lm2.predict(train_data)
# Predictions for test data
y_head_test_2 = fit_lm2.predict(test_data)
```

[24]:
```
# Compute the "Mean Squared Error"
MSE_train_2 = MSE(train_data['mpg'], y_head_train_2)
print(f"Train MSE:       {MSE_train_2:.3f}")

MSE_test_2 = MSE(test_data['mpg'], y_head_test_2)
print(f"Validation MSE:  {MSE_test_2:.3f}")
```

```
Train MSE:       18.353
Validation MSE:  19.723
```

3. Redo Step 2 but add the term horsepower^3 to the regression. Then add horsepower^4 and so on. For each model note the training and test MSE. How do the two MSE change with the flexibility of the method?

- horsepower^3 model:

[25]:
```
# Quadratic model using poly() function.
# (yields same predictions as the quadratic model above but
# the polynomial degree can be easily adjusted)
fit_lm3 = smf.ols(formula='mpg ~ poly(horsepower, 3)', data = train_data).fit()
print(fit_lm3.summary().tables[1])
```

```
==============================================================================
==========
                               coef    std err          t      P>|t|      [0.025
```

```
0.975]
------------------------------------------------------------------------------
----------
Intercept                 64.6844      6.696      9.660      0.000      51.477
77.891
poly(horsepower, 3)[0]    -0.6878      0.171     -4.019      0.000      -1.025
-0.350
poly(horsepower, 3)[1]     0.0031      0.001      2.270      0.024       0.000
0.006
poly(horsepower, 3)[2] -4.746e-06   3.39e-06     -1.399      0.164   -1.14e-05
1.95e-06
==============================================================================
==========
```

[27]:
```
# Predictions for test data
y_head_train_3 = fit_lm3.predict(train_data)
# Predictions for test data
y_head_test_3 = fit_lm3.predict(test_data)
```

[29]:
```
# Compute the "Mean Squared Error"
MSE_train_3 = MSE(train_data['mpg'], y_head_train_3)
print(f"Train MSE:       {MSE_train_3:.3f}")

MSE_test_3 = MSE(test_data['mpg'], y_head_test_3)
print(f"Validation MSE:  {MSE_test_3:.3f}")
```

```
Train MSE:       18.168
Validation MSE:  19.921
```

- horsepower^{10} model:

[26]:
```
fit_lm10 = smf.ols(formula='mpg ~ poly(horsepower, 10)', data = train_data).
↪fit()
print(fit_lm10.summary().tables[1])
```

```
==============================================================================
===========
                               coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
Intercept                2.256e-12    1.1e-13     20.570      0.000    2.04e-12
2.47e-12
poly(horsepower, 10)[0]   3.91e-10    1.9e-11     20.569      0.000    3.53e-10
4.28e-10
poly(horsepower, 10)[1]  8.099e-09   3.94e-10     20.570      0.000    7.32e-09
8.88e-09
poly(horsepower, 10)[2]  3.176e-07   1.54e-08     20.570      0.000    2.87e-07
3.48e-07
```

```
poly(horsepower, 10)[3]   8.66e-06   4.21e-07     20.571      0.000    7.83e-06
9.49e-06
poly(horsepower, 10)[4] -2.016e-07   1.11e-08    -18.115      0.000   -2.24e-07
-1.8e-07
poly(horsepower, 10)[5]  1.748e-09   1.07e-10     16.372      0.000    1.54e-09
1.96e-09
poly(horsepower, 10)[6] -6.662e-12   4.42e-13    -15.058      0.000   -7.54e-12
-5.79e-12
poly(horsepower, 10)[7]  9.413e-15   6.71e-16     14.022      0.000    8.09e-15
1.07e-14
poly(horsepower, 10)[8] -2.054e-19   1.96e-19     -1.046      0.297   -5.93e-19
1.82e-19
poly(horsepower, 10)[9] -1.066e-20   1.15e-19     -0.093      0.926   -2.37e-19
2.16e-19
==============================================================================
===========
```

[30]:
```python
# Predictions for test data
y_head_train_10 = fit_lm10.predict(train_data)
# Predictions for test data
y_head_test_10 = fit_lm10.predict(test_data)
```

[31]:
```python
# Compute the "Mean Squared Error"
MSE_train_10 = MSE(train_data['mpg'], y_head_train_10)
print(f"Train MSE:       {MSE_train_10:.3f}")

MSE_test_10 = MSE(test_data['mpg'], y_head_test_10)
print(f"Validation MSE:  {MSE_test_10:.3f}")
```

```
Train MSE:       57.118
Validation MSE:  79.702
```

- Setup for 10 models with different polynomial degrees:

[33]:
```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

# Lists to store models and MSE values
models = []
test_mse = []
train_mse = []

# Try 10 models
for i in range(1, 11):
    # Polynomial features
    poly = PolynomialFeatures(degree=i)
    X_train_poly = poly.fit_transform(train_data[['horsepower']])
    X_test_poly = poly.transform(test_data[['horsepower']])
```

```
    # Linear regression model
    model = LinearRegression()
    model.fit(X_train_poly, train_data['mpg'])

    # Store the model
    models.append(model)

    # Predict on train and test data
    y_train_pred = model.predict(X_train_poly)
    y_test_pred = model.predict(X_test_poly)

    # Calculate MSE
    train_mse.append(mean_squared_error(train_data['mpg'], y_train_pred))
    test_mse.append(mean_squared_error(test_data['mpg'], y_test_pred))

# Create a DataFrame for MSE values
mse = pd.DataFrame({'x': range(1, 11), 'testmse': test_mse, 'trainmse':␣
 ↪train_mse})
print(mse)
```

```
    x    testmse   trainmse
0   1  25.108539  23.002395
1   2  19.722533  18.352971
2   3  19.921368  18.167881
3   4  19.823280  18.151318
4   5  19.115938  17.951430
5   6  18.891373  17.927435
6   7  19.129574  17.955439
7   8  19.341569  17.985236
8   9  19.322458  17.976080
9  10  19.132870  17.964630
```

[34]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create the plot
plt.figure(figsize=(10, 6))

# Plot Test MSE
plt.plot(mse['x'], mse['testmse'], color='red', label='Validation MSE')
plt.scatter(mse['x'], mse['testmse'], color='red')

# Plot Train MSE
plt.plot(mse['x'], mse['trainmse'], color='blue', label='Train MSE')
plt.scatter(mse['x'], mse['trainmse'], color='blue')
```

```
# Add labels and title
plt.xlabel('Flexibility (Degree of Polynomial)', fontsize=18)
plt.ylabel('MSE', fontsize=18)
plt.legend()
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

# Show the plot
plt.show()

# Save the plot as EPS file
#plt.savefig("01_auto_mse_seed3.eps", format='eps')
```