Machine Learning for Economics and Finance

01_Auto_data

Ole Wilms

July 29, 2024

Get and Set working directory:

[3]:
```
import os           # Library 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
```

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

[5]:

	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

[11]:
```
n  = int(len(Auto)) # Number of observations in the dataset
nT = int(n/2)       # training sample size
nV = int(n/2)       # validation sample size

#nT = int(0.7 * n)  # or any other value for training sample size
#nV = int(0.3 * n)  # validation
```

[15]:
```
import pandas as pd
import numpy as np
```

```
# set seed
np.random.seed(1)

# 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
```

[16]:
```
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()
```

[25]:
```
print(fit_lm.summary().tables[1])
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     40.3338      1.023     39.416      0.000      38.316      42.352
horsepower    -0.1596      0.009    -17.788      0.000      -0.177      -0.142
==============================================================================
```

[17]:
```
import statsmodels.api as sm

# Predictions for training data
y_head_train = fit_lm.predict(sm.add_constant(train_data))

# OR Alternatively
#y_head_train = fit_lm.predict(train_data)
```

[18]:
```
# 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:  24.623
```

[19]:
```
# Predictions for test data
y_head_test = fit_lm.predict(sm.add_constant(test_data))
```

[20]:
```
# MSE in the test data
MSE_test = MSE(test_data['mpg'], y_head_test)
```

```
print(f"Mean Squared Error:  {MSE_test:.3f}")
```

```
Mean Squared Error:  23.362
```

[38]:
```
def poly(x, degree):
    return np.vander(x, degree + 1, increasing=True)[:, 1:]

fit_lm2 = smf.ols(formula='mpg ~ poly(horsepower, 10)', data = train_data).fit()
print(fit_lm2.summary().tables[1])
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
Intercept                2.267e-12   1.17e-13     19.397      0.000    2.04e-12
2.5e-12
poly(horsepower, 10)[0] -1.047e-09    5.4e-11    -19.397      0.000   -1.15e-09
-9.4e-10
poly(horsepower, 10)[1]  5.523e-09   2.85e-10     19.398      0.000    4.96e-09
6.08e-09
poly(horsepower, 10)[2]   3.23e-07   1.67e-08     19.398      0.000     2.9e-07
3.56e-07
poly(horsepower, 10)[3]  8.839e-06   4.56e-07     19.399      0.000    7.94e-06
9.74e-06
poly(horsepower, 10)[4] -2.061e-07   1.21e-08    -17.074      0.000    -2.3e-07
-1.82e-07
poly(horsepower, 10)[5]  1.789e-09   1.16e-10     15.420      0.000    1.56e-09
2.02e-09
poly(horsepower, 10)[6] -6.836e-12   4.82e-13    -14.175      0.000   -7.79e-12
-5.88e-12
poly(horsepower, 10)[7]  9.682e-15   7.34e-16     13.197      0.000    8.23e-15
1.11e-14
poly(horsepower, 10)[8] -8.958e-20   2.41e-19     -0.372      0.711   -5.65e-19
3.86e-19
poly(horsepower, 10)[9]  2.196e-20   1.28e-19      0.172      0.864    -2.3e-19
2.74e-19
==============================================================================
```

[27]:
```
# Predictions for test data
y_head_test2 = fit_lm2.predict(sm.add_constant(test_data))

# MSE in the test data
MSE_test2 = MSE(test_data['mpg'], y_head_test2)
print(f"Mean Squared Error:  {MSE_test2:.3f}")
```

```
Mean Squared Error:  73.668
```

[28]:
```
# Find the minimum value of the 'horsepower' column
hp_min = Auto['horsepower'].min()
print(hp_min)

# Find the maximum value of the 'horsepower' column
hp_max = Auto['horsepower'].max()
print(hp_max)
```

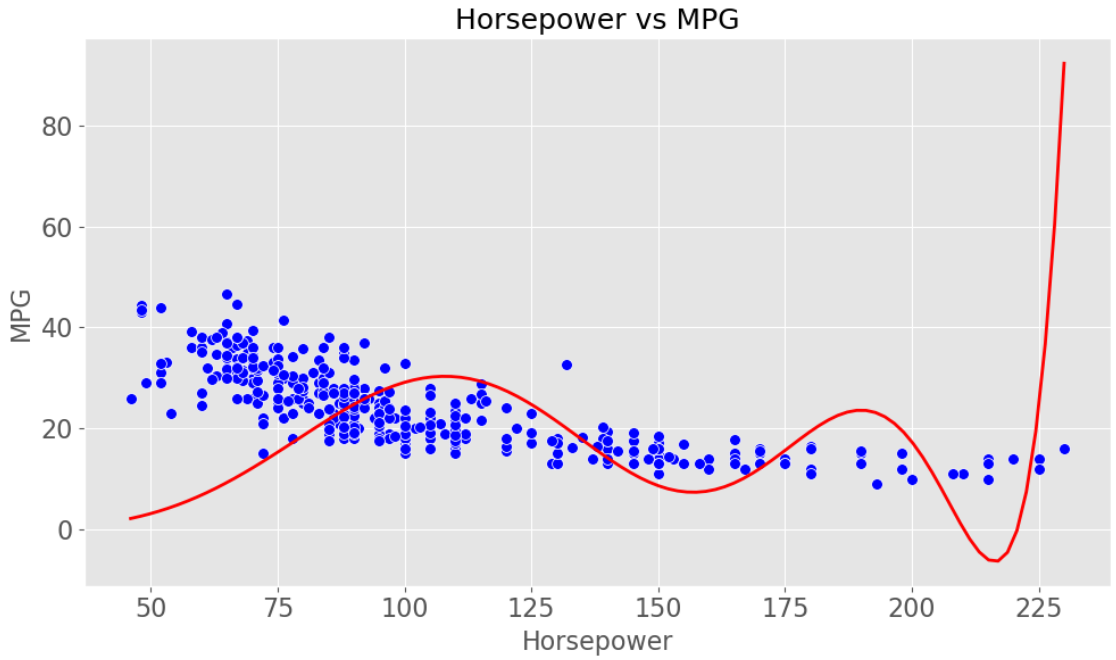
```
46
230
```

[50]:
```
from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm

# Create a grid of horsepower values
hp_grid = np.linspace(hp_min, hp_max, 100)

# Create a DataFrame for the grid
hp_grid_df = pd.DataFrame(hp_grid, columns=['horsepower'])

# Create polynomial features up to degree 10 for the grid
poly = PolynomialFeatures(degree=10)
hp_poly_grid = poly.fit_transform(hp_grid_df)

# Fit the polynomial regression model using statsmodels
train_data = Auto.iloc[train_sample]  # Use the train_sample index to get␣
 ↪training data
X_train_poly = poly.fit_transform(train_data[['horsepower']])
y_train = train_data['mpg']
fit2 = sm.OLS(y_train, X_train_poly).fit()

# Predict using the fitted model on the grid
hp_grid_df['mpg_pred'] = fit2.predict(hp_poly_grid)

print(hp_grid_df)
```

```
    horsepower   mpg_pred
0    46.000000   2.133755
1    47.858586   2.561017
2    49.717172   3.043722
3    51.575758   3.584271
4    53.434343   4.184536
..         …         …
95  222.565657   7.456441
96  224.424242  19.414182
97  226.282828  36.696028
98  228.141414  60.525143
99  230.000000  92.310918
```

```
[100 rows x 2 columns]
```

[57]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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

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

# Line plot of the predicted values
plt.plot(hp_grid_df['horsepower'], hp_grid_df['mpg_pred'], color='red',␣
 ↪linewidth=2)

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

# Show plot
plt.show()

# Save the plot as EPS file
#plt.savefig("01_auto_poly10.eps", format='eps')
```



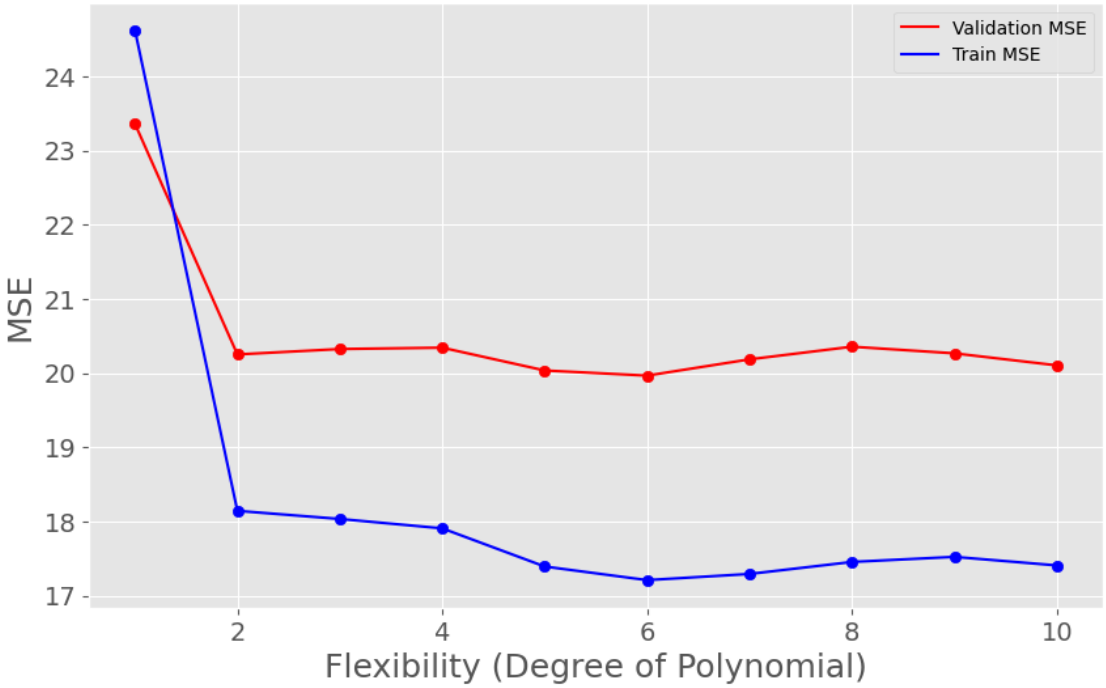
[60]:
```
from sklearn.metrics import mean_squared_error

# 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  23.361903  24.623010
1   2  20.252691  18.144194
2   3  20.325609  18.035030
3   4  20.343887  17.908378
4   5  20.036431  17.393915
5   6  19.966946  17.210659
6   7  20.186598  17.293690
7   8  20.357129  17.455614
8   9  20.267491  17.524121
9  10  20.105590  17.408553
```

[62]:
```
# 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')
```

```
The PostScript backend does not support transparency; partially transparent
artists will be rendered opaque.
```


Validation MSE
Train MSE
MSE
24
23
22
21
20
19
18
17
2
4
6
8
10
Flexibility (Degree of Polynomial)