Machine Learning for Economics and Finance

03_Auto_data_CV

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

# set seed
np.random.seed(2)
```

[5]:
```
import statsmodels.api as sm

# Logistic regression model:
X = Auto_data[['horsepower']]
X = sm.add_constant(X)  # Adds an intercept term to the model
y = Auto_data['mpg']

# Fit the model
glm_fit = sm.GLM(y, X, family=sm.families.Gaussian()).fit()
# Further function options can be found at:
# https://www.statsmodels.org/stable/glm.html  "Families"

print(glm_fit.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                    mpg   No. Observations:                  392
Model:                            GLM   Df Residuals:                      390
Model Family:                Gaussian   Df Model:                            1
Link Function:               Identity   Scale:                          24.066
Method:                          IRLS   Log-Likelihood:                -1178.7
Date:                Sat, 19 Oct 2024   Deviance:                       9385.9
Time:                        16:50:14   Pearson chi2:                 9.39e+03
No. Iterations:                     3   Pseudo R-squ. (CS):             0.7834
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         39.9359      0.717     55.660      0.000      38.530      41.342
horsepower    -0.1578      0.006    -24.489      0.000      -0.170      -0.145
==============================================================================
```

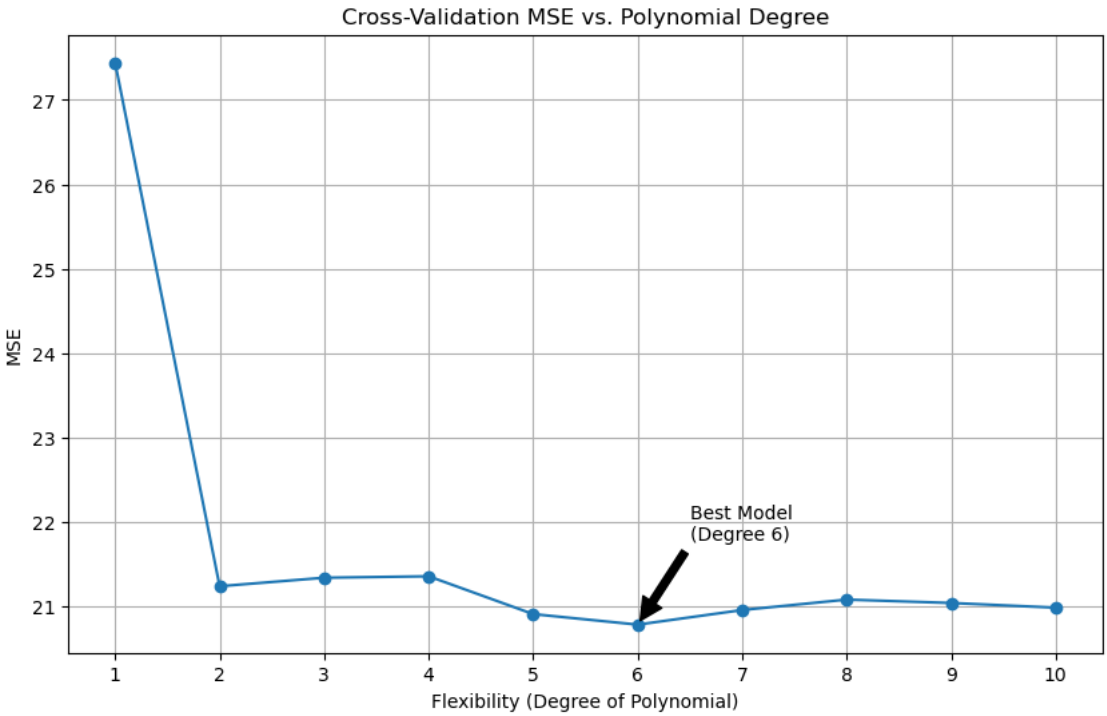
[6]:
```
print(glm_fit.params)  # print coefficients
```

```
const         39.935861
horsepower    -0.157845
dtype: float64
```

[7]:
```
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

# Number of folds for cross-validation
folds = 10
num_models = 10  # Number of polynomial degrees to fit
cv_errors = []

# Perform cross-validation for different polynomial degrees (flexibility)
for degree in range(1, num_models + 1):
    # Create a pipeline to add polynomial features and fit a linear regression␣
 ↪model
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())

    # Compute cross-validated MSE (negative mean squared error)
    mse_scores = -cross_val_score(model, X, y, cv=folds,␣
 ↪scoring='neg_mean_squared_error')

    # Save the average MSE for this model
    cv_errors.append(mse_scores.mean())
```

```
# Convert results into a DataFrame for easier plotting
cv_errors_df = pd.DataFrame({'Degree': np.arange(1, num_models + 1), 'MSE':␣
 ↪cv_errors})


# Find the best model (minimum MSE)
best_model_idx = cv_errors_df['MSE'].idxmin()
best_model_degree = cv_errors_df.loc[best_model_idx, 'Degree']
best_model_mse = cv_errors_df.loc[best_model_idx, 'MSE']
```

[8]:
```
import matplotlib.pyplot as plt

# Plot the cross-validation errors
plt.figure(figsize=(10, 6))
plt.plot(cv_errors_df['Degree'], cv_errors_df['MSE'], marker='o', linestyle='-')

# Mark the best model on the plot
plt.annotate(f'Best Model\n(Degree {best_model_degree})',
             xy=(best_model_degree, best_model_mse),
             xytext=(best_model_degree + 0.5, best_model_mse + 1),  # Adjust␣
 ↪position for better readability
             arrowprops=dict(facecolor='black', shrink=0.05), fontsize=10)

plt.xticks(np.arange(1, num_models + 1, step=1))
plt.xlabel('Flexibility (Degree of Polynomial)')
plt.ylabel('MSE')
plt.title('Cross-Validation MSE vs. Polynomial Degree')
plt.grid(True)
plt.show()
```



[9]:
```
# Output the cross-validation errors for each polynomial degree
print(cv_errors_df)
```

```
   Degree        MSE
0       1  27.439934
1       2  21.235840
2       3  21.336606
3       4  21.353887
4       5  20.905641
5       6  20.780511
6       7  20.953654
7       8  21.077388
8       9  21.037360
9      10  20.982162
```

[10]:
```
# Output the MSE values and best model
print(f"Best model is with degree {best_model_degree} with a corresponding MSE␣
  ↪of {round(best_model_mse, 3)}")
```

```
Best model is with degree 6 with a corresponding MSE of 20.781
```