統計諮詢報告

變數選擇實際操作

Yu-Wei Kuo

2025-03-08

目錄

一、讀取資料

```
# Numerical libraries
import numpy as np

# Import Linear Regression machine learning library
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso

from sklearn.metrics import r2_score

# to handle data in form of rows and columns
import pandas as pd

# importing ploting libraries
import matplotlib.pyplot as plt

#importing seaborn for statistical plots
import seaborn as sns

mpg_df = pd.read_csv("C:/Users/ASUS/Downloads/auto-mpg.csv")
pd.set_option('display.max_columns', None)  # Show all columns
pd.set_option('display.expand_frame_repr', False)  # Prevent wrapping into multiple lines
mpg_df.head() #sample 5 records
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chev
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 3
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth sate

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
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

```
mpg_df.info() #information about the data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object 
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64  
 7   origin        398 non-null    int64  
 8   car name      398 non-null    object 
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

二、資料處理

```
mpg_df[mpg_df['horsepower'].str.isnumeric()==False]
mpg_df=mpg_df.replace('?',np.nan)
mpg_df=mpg_df.drop('car name',axis=1)
mpg_df = mpg_df.apply(pd.to_numeric, errors='coerce')
mpg_df=mpg_df.apply(lambda x: x.fillna(x.median()),axis=0)
```

```
mpg_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    float64
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64  
 7   origin        398 non-null    int64  
dtypes: float64(4), int64(4)
memory usage: 25.0 KB
```

```
mpg_df.head() #sample 5 records
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
0	18.0	8	307.0	130.0	3504	12.0	70	1
1	15.0	8	350.0	165.0	3693	11.5	70	1
2	18.0	8	318.0	150.0	3436	11.0	70	1
3	16.0	8	304.0	150.0	3433	12.0	70	1
4	17.0	8	302.0	140.0	3449	10.5	70	1

三、設定模型變數

```
#'mpg' is dependent variable so drop it . Copying rest of the columns to X
X = mpg_df.drop('mpg', axis=1)

#Copying the 'mpg' column alone into the y dataframe. This is the dependent variable
y = mpg_df[['mpg']]
```

四、假設變數獨立下進行模型訓練

```
from sklearn import preprocessing

# scale all the columns of the mpg_df. This will produce a numpy array
X_scaled = preprocessing.scale(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)  # ideally the training and test should be

y_scaled = preprocessing.scale(y)
y_scaled = pd.DataFrame(y_scaled, columns=y.columns)  # ideally the training and test should be
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.30, random_state=1)
```

1. 線性模型

```
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)

for i, col in enumerate(X_train.columns):
    print(f"Regression model coefficient for {col} is {regression_model.coef_[0][i]}")
intercept = regression_model.intercept_[0]

print("Regression model intercept is:", regression_model.intercept_)
```

```
Regression model coefficient for cylinders is -0.08561436895562707
Regression model coefficient for displacement is 0.30441822535930246
Regression model coefficient for horsepower is -0.09718466302484263
Regression model coefficient for weight is -0.7628632829136761
Regression model coefficient for acceleration is 0.021591275172924626
Regression model coefficient for model year is 0.3749408074118714
Regression model coefficient for origin is 0.12302637024556856
Regression model intercept is: [0.01283313]
```

2. Ridge模型

```
ridge = Ridge(alpha=0.3)  # coefficients are prevented to become too big by this alpha value
ridge.fit(X_train, y_train)
```

```
# Iterate through the columns and print out the corresponding coefficient
for i, col in enumerate(X_train.columns):
    print(f"Ridge model coefficient for {col} is {ridge.coef_[i]}")
print("Ridge model intercept is:", ridge.intercept_)
```

```
Ridge model coefficient for cylinders is -0.08073001909556088
Ridge model coefficient for displacement is 0.2882220741723193
Ridge model coefficient for horsepower is -0.09985675156606261
Ridge model coefficient for weight is -0.7510550916024498
Ridge model coefficient for acceleration is 0.019127698314832648
Ridge model coefficient for model year is 0.3737831248958428
Ridge model coefficient for origin is 0.1221271728580032
Ridge model intercept is: [0.01285216]
```

3. Lasso模型

```
lasso = Lasso(alpha=0.1)
lasso.fit(X_train,y_train)
for i,col in enumerate(X_train):
    print (f"Lasso model coefficients for {col} is {lasso.coef_[i]}:")
print("Lasso model intercept is:", lasso.intercept_)
```

```
Lasso model coefficients for cylinders is -0.0:
Lasso model coefficients for displacement is -0.0:
Lasso model coefficients for horsepower is -0.013280002937314536:
Lasso model coefficients for weight is -0.6205207866794482:
Lasso model coefficients for acceleration is 0.0:
Lasso model coefficients for model year is 0.29198732924913484:
Lasso model coefficients for origin is 0.021567653979880638:
Lasso model intercept is: [0.01036444]
```

4. ElasticNet模型

```
from sklearn.linear_model import ElasticNet
elastic_net = ElasticNet(alpha=0.3, l1_ratio=0.5)
elastic_net.fit(X_train, y_train)
for i, col in enumerate(X_train.columns):
    print(f"ElasticNet model coefficient for {col} is {elastic_net.coef_[i]}")
print("ElasticNet model intercept is:", elastic_net.intercept_)
```

```
ElasticNet model coefficient for cylinders is -0.054359389774312886
ElasticNet model coefficient for displacement is -0.09160279755788328
ElasticNet model coefficient for horsepower is -0.10095171404332741
ElasticNet model coefficient for weight is -0.33584907028750005
ElasticNet model coefficient for acceleration is 0.0
ElasticNet model coefficient for model year is 0.20951348717019383
ElasticNet model coefficient for origin is 0.023067594598654484
ElasticNet model intercept is: [0.00656019]
```

5. 比較模型的r-square

```
print("Linear Regression R-squared (Train):", regression_model.score(X_train, y_train))
print("Linear Regression R-squared (Test):", regression_model.score(X_test, y_test))

print("Ridge Regression R-squared (Train):", ridge.score(X_train, y_train))
print("Ridge Regression R-squared (Test):", ridge.score(X_test, y_test))
```

```
print("Lasso Regression R-squared (Train):", lasso.score(X_train, y_train))
print("Lasso Regression R-squared (Test):", lasso.score(X_test, y_test))

print("ElasticNet R-squared (Train):", elastic_net.score(X_train, y_train))
print("ElasticNet R-squared (Test):", elastic_net.score(X_test, y_test))
```

```
Linear Regression R-squared (Train): 0.8081802739111359
Linear Regression R-squared (Test): 0.8472274567567305
Ridge Regression R-squared (Train): 0.8081651504849107
Ridge Regression R-squared (Test): 0.8475401122140553
Lasso Regression R-squared (Train): 0.7853770917055521
Lasso Regression R-squared (Test): 0.8277658025171161
ElasticNet R-squared (Train): 0.7375966120113489
ElasticNet R-squared (Test): 0.7902295317854511
```

五、假設變數非獨立下進行模型訓練

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree = 2, interaction_only=True)
X_poly = poly.fit_transform(X_scaled)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.30, random_state=1)
X_train.shape
```

```
(278, 29)
```

1. 線性模型

```
regression_model.fit(X_train, y_train)
print("Regression model:", (regression_model.coef_))
```

```
Regression model: [[-3.46148973e-13  6.50937571e-01 -2.17855687e-01 -2.45601437e+00
  -4.81187049e+00 -6.44775739e-01  2.85471189e+00  1.03974036e+00
  -4.73859713e-01 -6.74188889e-01  1.43519865e+00  1.13264258e+00
  -1.54521285e+00  9.04315573e-01 -1.42332439e-01  1.31740727e+00
  -1.25529739e+00  3.45224055e+00  1.32798427e+00  2.56954964e-01
  -9.14623227e-01 -1.37168588e+00  8.80133118e-01  6.59590958e-01
  -6.92229746e-01 -1.01303860e+00  5.50296551e-01  1.24973189e+00
   9.07830046e-01]]
```

2. Ridge模型

```
ridge = Ridge(alpha=0.3)
ridge.fit(X_train,y_train)
print ("Ridge model:", (ridge.coef_))
```

```
Ridge model: [ 0.          0.64813084 -0.36554878 -2.46919357 -4.70667241 -0.63510083
  2.8499319   0.96173248 -0.49248204 -0.57492325  1.37951065  1.0927984
 -1.42036657  0.89110432 -0.16927823  1.35119104 -1.1349459   3.21038276
  1.16514278  0.22836119 -0.89663938 -1.35284633  0.79573939  0.58576253
 -0.61776911 -0.88637153  0.54014514  1.23036832  0.88256903]
```

3. Lasso模型

```
lasso = Lasso(alpha=0.1)
lasso.fit(X_train,y_train)
print ("Lasso model:", (lasso.coef_))
```

```
Lasso model: [ 0.         -0.         -0.37802101 -2.04156884 -4.64843934 -0.
  2.78580954  0.05433586 -0.          0.          0.07086347  0.
 -0.          0.          0.          1.16818735  0.          0.
  0.          0.36475302 -0.         -0.61170462  0.         -0.
 -0.          0.          0.24737398  0.6555845   0.34807138]
```

4. ElasicNet模型

```
elastic_net = ElasticNet(alpha=0.3, l1_ratio=0.5)
elastic_net.fit(X_train, y_train)
print("ElasticNet model coefficients:", (elastic_net.coef_))
```

```
ElasticNet model coefficients: [ 0.         -0.62943469 -1.25029915 -1.86673058 -2.6819481  -0.
  2.30084771  0.34524843  0.          0.          0.05606344  0.
 -0.          0.          0.12754883  0.4451572  -0.         -0.
  0.          0.39320115 -0.03082228 -0.40837813 -0.         -0.09888907
 -0.16412644 -0.          0.22866038  0.60005713  0.2380177 ]
```

5. 比較模型

```
print("Linear Regression R-squared (Train):", regression_model.score(X_train, y_train))
print("Linear Regression R-squared (Test):", regression_model.score(X_test, y_test))

print("Ridge Regression R-squared (Train):", ridge.score(X_train, y_train))
print("Ridge Regression R-squared (Test):", ridge.score(X_test, y_test))

print("Lasso Regression R-squared (Train):", lasso.score(X_train, y_train))
print("Lasso Regression R-squared (Test):", lasso.score(X_test, y_test))

print("ElasticNet R-squared (Train):", elastic_net.score(X_train, y_train))
print("ElasticNet R-squared (Test):", elastic_net.score(X_test, y_test))
```

```
Linear Regression R-squared (Train): 0.8925011833410883
Linear Regression R-squared (Test): 0.8602464955563417
Ridge Regression R-squared (Train): 0.8924614223751737
Ridge Regression R-squared (Test): 0.8615025469455584
Lasso Regression R-squared (Train): 0.8783598485420683
Lasso Regression R-squared (Test): 0.8780043695312
ElasticNet R-squared (Train): 0.8579699617052333
ElasticNet R-squared (Test): 0.8830940161894539
```

Kaggle 提供的數據集 (Kaggle, 2025a, 2025b)。

六、參考文獻

Kaggle. (2025a). Auto-mpg Dataset. https://www.kaggle.com/datasets/uciml/autompg-dataset

Kaggle. (2025b). Ridge, Lasso, Linear Regression Models. https://www.kaggle.com/code/vignesh1609/ridge-lasso-linear-regression-models/notebook