Untitled

May 1, 2024

[115]:

```python
# 1
# a
import pandas as pd
import statsmodels.api as sm

default_data = pd.read_csv("/Users/rouren/Desktop/24S ML/HW/hw5/Data-Default.
 ↪csv")

# Convert 'default' column to binary numeric values
default_data['default'] = default_data['default'].map({'No': 0, 'Yes': 1})

import numpy as np
np.random.seed(42)

X = default_data[['income', 'balance']]
y = default_data['default']
X = sm.add_constant(X)  # Adding a constant to the model

logit_model = sm.Logit(y, X)
result = logit_model.fit()

print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.078948
         Iterations 10
                           Logit Regression Results
==============================================================================
Dep. Variable:                default   No. Observations:                10000
Model:                          Logit   Df Residuals:                     9997
Method:                           MLE   Df Model:                            2
Date:                Wed, 01 May 2024   Pseudo R-squ.:                  0.4594
Time:                        21:12:17   Log-Likelihood:                -789.48
converged:                       True   LL-Null:                       -1460.3
Covariance Type:            nonrobust   LLR p-value:                 4.541e-292
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
const         -11.5405      0.435    -26.544      0.000     -12.393     -10.688
income       2.081e-05   4.99e-06      4.174      0.000     1.1e-05    3.06e-05
balance         0.0056      0.000     24.835      0.000       0.005       0.006
==============================================================================

Possibly complete quasi-separation: A fraction 0.14 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

[116]:
```python
# For the 'income' predictor: 2.081 x 10 ^(-5) with a stand error of 4.99 x␣
↪10^(-6)
# For the 'balance ' predictor:  with a stand error of
```

[117]:
```python
# b
import pandas as pd
import numpy as np
import statsmodels.api as sm

def boot_fn(data, index):
    # Subset the data
    sampled_data = data.iloc[index]

    X = sampled_data[['income', 'balance']]
    y = sampled_data['default']
    X = sm.add_constant(X)  # Adding a constant to the model
    logit_model = sm.Logit(y, X)
    result = logit_model.fit(disp=0)  # Suppress output to console for fitting

    coefs = result.params[['income', 'balance']]

    return coefs
```

[118]:
```python
# c
import pandas as pd
import numpy as np
import statsmodels.api as sm

def boot_fn(data, index):
    # Subset the data
    sampled_data = data.iloc[index]

    X = sampled_data[['income', 'balance']]
    y = sampled_data['default']
    X = sm.add_constant(X)  # Adding a constant to the model
    logit_model = sm.Logit(y, X)
    result = logit_model.fit(disp=0)  # Suppress output to console for fitting
```

```
    coefs = result.params[['income', 'balance']]

    return coefs

def bootstrap_standard_errors(data, num_iterations=1000):
    n = len(data)
    boot_index = np.random.randint(0, n, size=(num_iterations, n))
    boot_coefs = np.array([boot_fn(data, idx) for idx in boot_index])
    return boot_coefs.std(axis=0)

default_data = pd.read_csv("/Users/rouren/Desktop/24S ML/HW/hw5/Data-Default.
 ↪csv")

# Convert 'default' column to binary numeric values
default_data['default'] = default_data['default'].map({'No': 0, 'Yes': 1})

np.random.seed(42)

boot_standard_errors = bootstrap_standard_errors(default_data,␣
 ↪num_iterations=1000)
print("Bootstrap Standard Errors:")
print("Income:", boot_standard_errors[0])
print("Balance:", boot_standard_errors[1])
```

```
Bootstrap Standard Errors:
Income: 4.968586838704015e-06
Balance: 0.00023209224235493296
```

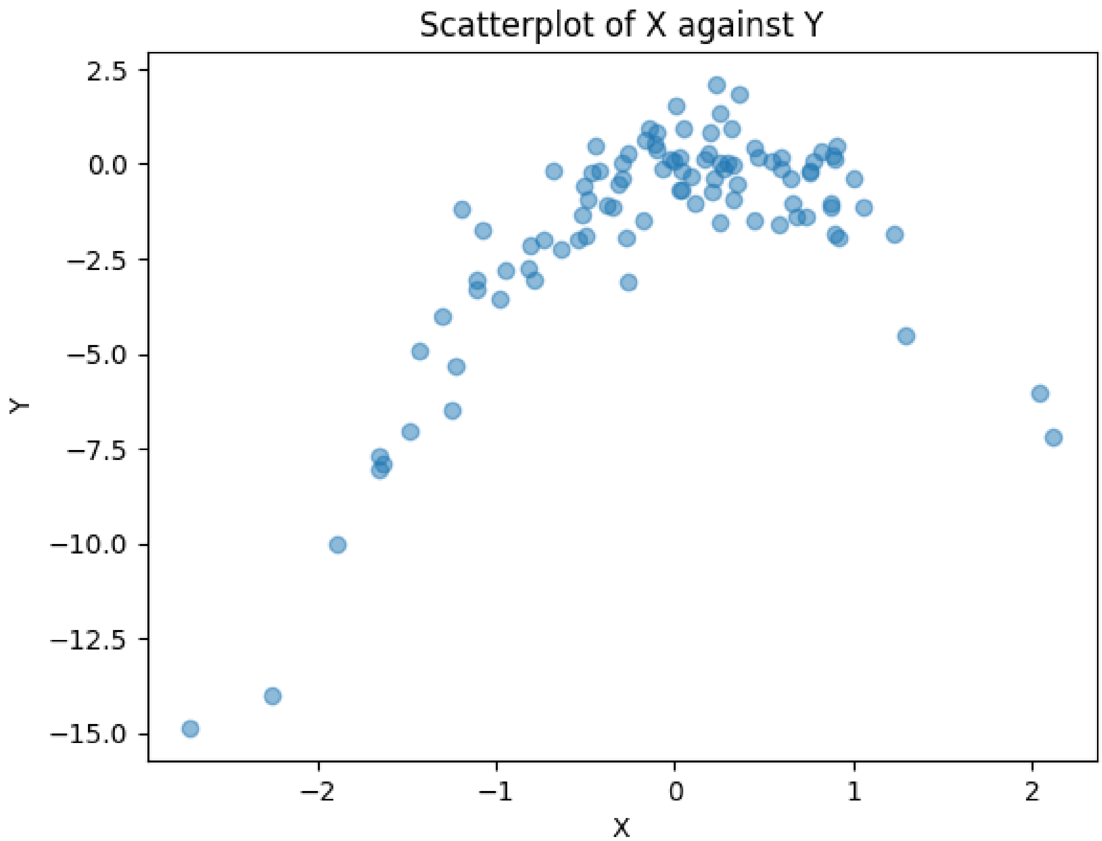
[119]:
```
# d
# The standard errors derived through bootstrap resampling closely resemble␣
 ↪those calculated using the statistical models underlying the glm() function.␣
 ↪This demonstrates the practical applicability of the bootstrap method.
```

[120]:
```
# 2
# a
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.default_rng(1)
x = rng.normal(size=100)
y = x - 2 * x**2 + rng.normal(size=100)
# Model equation
print("Model equation: y = x - 2x^2 + epsilon")
```

```
Model equation: y = x - 2x^2 + epsilon
```

[121]:
```
# n=100, p=1
```

[122]:
```
# b
import matplotlib.pyplot as plt

plt.scatter(x, y, alpha=0.5)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Scatterplot of X against Y')
plt.show()
```



[123]:
```
# The scatterplot reveals a non-linear relationship between the variables,␣
↪showing a quadratic trend.
```

[124]:
```
# c
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

rng = np.random.default_rng(10)

# Fit a linear regression model with degree 1 polynomial features
model_1 = make_pipeline(PolynomialFeatures(degree=1), LinearRegression())
```

```python
model_1.fit(x.reshape(-1, 1), y)

# Compute LOOCV error for model 1
loo = LeaveOneOut()
mse_model_1 = []
for train_index, test_index in loo.split(x):
    X_train, X_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]
    model_1.fit(X_train.reshape(-1, 1), y_train)
    y_pred = model_1.predict(X_test.reshape(-1, 1))
    mse_model_1.append(mean_squared_error(y_test, y_pred))
cv_error_1 = np.mean(mse_model_1)

# Fit models with polynomial features of degree 2, 3, and 4
cv_errors = []
for degree in range(1, 5):
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    mse_degree = []
    for train_index, test_index in loo.split(x):
        X_train, X_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]
        model.fit(X_train.reshape(-1, 1), y_train)
        y_pred = model.predict(X_test.reshape(-1, 1))
        mse_degree.append(mean_squared_error(y_test, y_pred))
    cv_errors.append(np.mean(mse_degree))

cvDF = pd.DataFrame({'degree': range(1, 5), 'cv.error': cv_errors})
print(cvDF)
```

```
   degree  cv.error
0       1  6.633030
1       2  1.122937
2       3  1.301797
3       4  1.332394
```

[125]:
```python
# d
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

rng = np.random.default_rng(40)

# Fit a linear regression model with degree 1 polynomial features
model_1 = make_pipeline(PolynomialFeatures(degree=1), LinearRegression())
model_1.fit(x.reshape(-1, 1), y)

# Compute LOOCV error for model 1
loo = LeaveOneOut()
```

```python
mse_model_1 = []
for train_index, test_index in loo.split(x):
    X_train, X_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]
    model_1.fit(X_train.reshape(-1, 1), y_train)
    y_pred = model_1.predict(X_test.reshape(-1, 1))
    mse_model_1.append(mean_squared_error(y_test, y_pred))
cv_error_1 = np.mean(mse_model_1)

# Fit models with polynomial features of degree 2, 3, and 4
cv_errors = []
for degree in range(1, 5):
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    mse_degree = []
    for train_index, test_index in loo.split(x):
        X_train, X_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]
        model.fit(X_train.reshape(-1, 1), y_train)
        y_pred = model.predict(X_test.reshape(-1, 1))
        mse_degree.append(mean_squared_error(y_test, y_pred))
    cv_errors.append(np.mean(mse_degree))

cvDF = pd.DataFrame({'degree': range(1, 5), 'cv.error': cv_errors})
print(cvDF)
```

```
   degree  cv.error
0       1  6.633030
1       2  1.122937
2       3  1.301797
3       4  1.332394
```

[126]:
```python
# The outcomes remain consistent across both seeds. This consistency arises␣
↪from the nature of LOOCV, which systematically evaluates each iteration of␣
↪data partitioning, ensuring that each observation serves as a test sample␣
↪exactly once while the rest form the training set. Consequently, the␣
↪specific order of data partitioning does not influence the results.
```

[127]:
```python
# e
# The second-degree polynomial model had the smallest LOOCV error, which was␣
↪expected since the data was simulated with a second-degree polynomial␣
↪relationship between x and y. Incorporating the square of x in the model␣
↪captures the underlying nonlinear pattern better, closely resembling the␣
↪observed data distribution.
```

[128]:
```python
# f
import statsmodels.api as sm
from scipy import stats
```

```python
# Fit the linear regression model with fourth-degree polynomial features
x_poly = PolynomialFeatures(degree=4).fit_transform(x.reshape(-1, 1))
lm4 = sm.OLS(y, x_poly).fit()

print(lm4.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.894
Model:                            OLS   Adj. R-squared:                  0.890
Method:                 Least Squares   F-statistic:                     200.2
Date:                Wed, 01 May 2024   Prob (F-statistic):           2.22e-45
Time:                        21:12:28   Log-Likelihood:                -137.74
No. Observations:                 100   AIC:                             285.5
Df Residuals:                      95   BIC:                             298.5
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.1008      0.136      0.743      0.460      -0.169       0.370
x1             0.9050      0.205      4.423      0.000       0.499       1.311
x2            -2.5059      0.221    -11.336      0.000      -2.945      -2.067
x3             0.0338      0.073      0.466      0.642      -0.110       0.178
x4             0.1042      0.045      2.309      0.023       0.015       0.194
==============================================================================
Omnibus:                        2.476   Durbin-Watson:                   2.163
Prob(Omnibus):                  0.290   Jarque-Bera (JB):                2.097
Skew:                           0.118   Prob(JB):                        0.351
Kurtosis:                       3.669   Cond. No.                         19.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[129]:
```python
# In the OLS regression results, the coefficients for x1 and x2 have low
↪p-values, indicating statistical significance, while the coefficients for
↪higher-degree terms may not be significant. This suggests that x1 and x2
↪have significant effects on the response variable.
```

[130]:
```python
# 3
# a
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.feature_selection import SequentialFeatureSelector
```

```python
import numpy as np

file_path = "/Users/rouren/Desktop/24S ML/HW/hw5/Boston/Boston.csv"
data = pd.read_csv(file_path)

X = data.drop(columns=['CRIM'])
y = data['CRIM']

kf = KFold(n_splits=5, shuffle=True, random_state=28)
lr = LinearRegression()

# Forward Stepwise Selection
fss_errors = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    fss = SequentialFeatureSelector(lr, direction='forward')
    fss.fit(X_train, y_train)
    selected_features = fss.transform(X_train)

    lr.fit(selected_features, y_train)
    y_pred = lr.predict(fss.transform(X_test))

    fss_errors.append(mean_squared_error(y_test, y_pred))

fss_avg_error = np.mean(fss_errors)

# Backward Stepwise Selection
bss_errors = []
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    bss = SequentialFeatureSelector(lr, direction='backward')
    bss.fit(X_train, y_train)
    selected_features = bss.transform(X_train)

    lr.fit(selected_features, y_train)
    y_pred = lr.predict(bss.transform(X_test))

    bss_errors.append(mean_squared_error(y_test, y_pred))

bss_avg_error = np.mean(bss_errors)

print("Average MSE for Forward Stepwise Selection:", fss_avg_error)
print("Average MSE for Backward Stepwise Selection:", bss_avg_error)
```

```
/Users/rouren/anaconda3/lib/python3.10/site-
packages/sklearn/feature_selection/_sequential.py:206: FutureWarning: Leaving
`n_features_to_select` to None is deprecated in 1.0 and will become 'auto' in
1.3. To keep the same behaviour as with None (i.e. select half of the features)
and avoid this warning, you should manually set `n_features_to_select='auto'`
and set tol=None when creating an instance.
  warnings.warn(
/Users/rouren/anaconda3/lib/python3.10/site-
packages/sklearn/feature_selection/_sequential.py:206: FutureWarning: Leaving
`n_features_to_select` to None is deprecated in 1.0 and will become 'auto' in
1.3. To keep the same behaviour as with None (i.e. select half of the features)
and avoid this warning, you should manually set `n_features_to_select='auto'`
and set tol=None when creating an instance.
  warnings.warn(
/Users/rouren/anaconda3/lib/python3.10/site-
packages/sklearn/feature_selection/_sequential.py:206: FutureWarning: Leaving
`n_features_to_select` to None is deprecated in 1.0 and will become 'auto' in
1.3. To keep the same behaviour as with None (i.e. select half of the features)
and avoid this warning, you should manually set `n_features_to_select='auto'`
and set tol=None when creating an instance.
  warnings.warn(
/Users/rouren/anaconda3/lib/python3.10/site-
packages/sklearn/feature_selection/_sequential.py:206: FutureWarning: Leaving
`n_features_to_select` to None is deprecated in 1.0 and will become 'auto' in
1.3. To keep the same behaviour as with None (i.e. select half of the features)
and avoid this warning, you should manually set `n_features_to_select='auto'`
and set tol=None when creating an instance.
  warnings.warn(
/Users/rouren/anaconda3/lib/python3.10/site-
packages/sklearn/feature_selection/_sequential.py:206: FutureWarning: Leaving
`n_features_to_select` to None is deprecated in 1.0 and will become 'auto' in
1.3. To keep the same behaviour as with None (i.e. select half of the features)
and avoid this warning, you should manually set `n_features_to_select='auto'`
and set tol=None when creating an instance.
  warnings.warn(
/Users/rouren/anaconda3/lib/python3.10/site-
packages/sklearn/feature_selection/_sequential.py:206: FutureWarning: Leaving
`n_features_to_select` to None is deprecated in 1.0 and will become 'auto' in
1.3. To keep the same behaviour as with None (i.e. select half of the features)
and avoid this warning, you should manually set `n_features_to_select='auto'`
and set tol=None when creating an instance.
  warnings.warn(
/Users/rouren/anaconda3/lib/python3.10/site-
packages/sklearn/feature_selection/_sequential.py:206: FutureWarning: Leaving
`n_features_to_select` to None is deprecated in 1.0 and will become 'auto' in
1.3. To keep the same behaviour as with None (i.e. select half of the features)
and avoid this warning, you should manually set `n_features_to_select='auto'`
and set tol=None when creating an instance.
```

```
  warnings.warn(
/Users/rouren/anaconda3/lib/python3.10/site-
packages/sklearn/feature_selection/_sequential.py:206: FutureWarning: Leaving
`n_features_to_select` to None is deprecated in 1.0 and will become 'auto' in
1.3. To keep the same behaviour as with None (i.e. select half of the features)
and avoid this warning, you should manually set `n_features_to_select='auto'`
and set tol=None when creating an instance.
  warnings.warn(
/Users/rouren/anaconda3/lib/python3.10/site-
packages/sklearn/feature_selection/_sequential.py:206: FutureWarning: Leaving
`n_features_to_select` to None is deprecated in 1.0 and will become 'auto' in
1.3. To keep the same behaviour as with None (i.e. select half of the features)
and avoid this warning, you should manually set `n_features_to_select='auto'`
and set tol=None when creating an instance.
  warnings.warn(
/Users/rouren/anaconda3/lib/python3.10/site-
packages/sklearn/feature_selection/_sequential.py:206: FutureWarning: Leaving
`n_features_to_select` to None is deprecated in 1.0 and will become 'auto' in
1.3. To keep the same behaviour as with None (i.e. select half of the features)
and avoid this warning, you should manually set `n_features_to_select='auto'`
and set tol=None when creating an instance.
  warnings.warn(

Average MSE for Forward Stepwise Selection: 46.39471179430429
Average MSE for Backward Stepwise Selection: 45.53566952450801
```

[131]:

```
# Though the difference is minimal, BSS tended to perform slightly better.␣
↪However, both methods offer reasonable approaches for feature selection,␣
↪with the choice potentially depending on factors like interpretability or␣
↪computational efficiency.
```

[132]:

```
# b
fss_lr = LinearRegression()
bss_lr = LinearRegression()

fss_selected_features = fss.transform(X)
fss_lr.fit(fss_selected_features, y)

bss_selected_features = bss.transform(X)
bss_lr.fit(bss_selected_features, y)

# Calculate AIC for FSS model
fss_train_pred = fss_lr.predict(fss_selected_features)
fss_residuals = y - fss_train_pred
fss_mse = np.mean(fss_residuals ** 2)
fss_n = len(y)
fss_k = fss_selected_features.shape[1]
fss_aic = fss_n * np.log(fss_mse) + 2 * fss_k
```

```python
# Calculate AIC for BSS model
bss_train_pred = bss_lr.predict(bss_selected_features)
bss_residuals = y - bss_train_pred
bss_mse = np.mean(bss_residuals ** 2)
bss_n = len(y)
bss_k = bss_selected_features.shape[1]
bss_aic = bss_n * np.log(bss_mse) + 2 * bss_k

print("AIC for Forward Stepwise Selection (FSS):", fss_aic)
print("AIC for Backward Stepwise Selection (BSS):", bss_aic)

if fss_aic < bss_aic:
    print("The model selected by Forward Stepwise Selection (FSS) performs␣
 ↪better based on AIC.")
else:
    print("The model selected by Backward Stepwise Selection (BSS) performs␣
 ↪better based on AIC.")
```

```
AIC for Forward Stepwise Selection (FSS): 1928.8133827616027
AIC for Backward Stepwise Selection (BSS): 1907.1129894324838
The model selected by Backward Stepwise Selection (BSS) performs better based on
AIC.
```

[133]:
```python
# Lower AIC values indicate a better trade-off between goodness of fit and␣
 ↪model complexity. Therefore, the model selected by BSS is preferred as it␣
 ↪achieves a lower AIC, suggesting a more optimal balance between model␣
 ↪performance and simplicity.
# Different feature selection methods like FSS and BSS may select varying␣
 ↪models due to their different approaches. FSS adds features iteratively,␣
 ↪while BSS removes them. Using AIC is favored over training error because it␣
 ↪balances model fit and complexity, aiding in model comparison and␣
 ↪generalization.
```

[134]:
```python
# The chosen model may not include all features in the dataset. This is because␣
 ↪feature selection methods like FSS and BSS aim to identify a subset of␣
 ↪features that best explain the target variable while minimizing complexity.␣
 ↪Exclusion of some features could be due to redundancy, irrelevance, or␣
 ↪capturing interactions between features.
```