Machine Learning for Economics and Finance

03_Default_data

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
default_data = load_data('Default')  # Loading the data
default_data.head()                  # Showing the first 5 Lines of Data.
```

```
/mnt/ds/home/UHH_MLSJ_2024/Code/Python/03-CrossValidation
```

[1]:

	default	student	balance	income
0	No	No	729.526495	44361.625074
1	No	Yes	817.180407	12106.134700
2	No	No	1073.549164	31767.138947
3	No	No	529.250605	35704.493935
4	No	No	785.655883	38463.495879

[2]:
```
print(default_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   default  10000 non-null  object
 1   student  10000 non-null  object
 2   balance  10000 non-null  float64
 3   income   10000 non-null  float64
dtypes: float64(2), object(2)
memory usage: 312.6+ KB
None
```

[3]:
```
print(default_data.describe())
```

```
            balance        income
count  10000.000000  10000.000000
mean     835.374886  33516.981876
```

```
std       483.714985  13336.639563
min         0.000000    771.967729
25%       481.731105  21340.462903
50%       823.636973  34552.644802
75%      1166.308386  43807.729272
max      2654.322576  73554.233495
```

[4]:
```python
import numpy as np

# set seed
np.random.seed(1)

# Number of observations in the dataset
n = len(default_data)

# Shuffle the dataset using np.random.permutation
shuffled_indices = np.random.permutation(n)

# Compute training and validation sample sizes
nT = int(0.7 * n)  # Training sample size

# Split the shuffled dataset based on the shuffled indices
train_data = default_data.iloc[shuffled_indices[:nT]]   # First 70% for training
test_data = default_data.iloc[shuffled_indices[nT:]]    # Remaining 30% for␣
 ↪validation
```

[5]:
```python
defaulting_train = (train_data['default'] == 'Yes').mean()
defaulting_test = (test_data['default'] == 'Yes').mean()
# The "train_data$default == "Yes": creates a logical vector where each element␣
 ↪is TRUE
# if the corresponding element.
# The outer mean() function than calculates the proportion of TRUE values
# in the logical vector.

# Output the results
print(f"Train data percentage of defaulting: {round(defaulting_train, 5)}")
print(f"Test data percentage of defaulting:  {round(defaulting_test, 5)}")
```

```
Train data percentage of defaulting: 0.03157
Test data percentage of defaulting:  0.03733
```

[6]:
```python
import statsmodels.api as sm

train_data_copy = train_data.copy()
train_data_copy['default'] = train_data_copy['default'].map({'No': 0, 'Yes': 1})

test_data_copy = test_data.copy()
test_data_copy['default'] = test_data_copy['default'].map({'No': 0, 'Yes': 1})
```

```
# Logistic regression model:
X_train = train_data_copy[['income','balance']]
X_train = sm.add_constant(X_train)  # Adds an intercept term to the model
X_test = test_data_copy[['income','balance']]
X_test = sm.add_constant(X_test)  # Adds an intercept term to the model
y_train = train_data_copy['default']

# Fit the logistic regression model
glm_fit = sm.GLM(y_train, X_train, family=sm.families.Binomial()).fit()
print(glm_fit.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                 default   No. Observations:                 7000
Model:                             GLM   Df Residuals:                     6997
Model Family:                Binomial   Df Model:                            2
Link Function:                   Logit   Scale:                          1.0000
Method:                           IRLS   Log-Likelihood:                -542.14
Date:                Sat, 19 Oct 2024   Deviance:                       1084.3
Time:                        16:53:00   Pearson chi2:                 5.42e+03
No. Iterations:                     9   Pseudo R-squ. (CS):             0.1179
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -11.3514      0.515    -22.060      0.000     -12.360     -10.343
income      1.847e-05   5.98e-06      3.091      0.002    6.76e-06    3.02e-05
balance        0.0055      0.000     20.428      0.000       0.005       0.006
==============================================================================
```

[7]:
```
print(glm_fit.params)  # print coefficients
```

```
const     -11.351394
income      0.000018
balance     0.005536
dtype: float64
```

[8]:
```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold

# ---- K-Fold Cross-Validation ----
folds = 10
kf = KFold(n_splits=folds, shuffle=True, random_state=12)
cv_errors = []

for train_index, test_index in kf.split(X_train):
```

```
    X_train_fold, X_test_fold = X_train.iloc[train_index], X_train.
 ↪iloc[test_index]
    y_train_fold, y_test_fold = y_train.iloc[train_index], y_train.
 ↪iloc[test_index]

    # Fit model on this fold
    glm_fold = sm.GLM(y_train_fold, X_train_fold, family=sm.families.
 ↪Binomial()).fit()

    # Compute the out-of-sample error for this fold
    preds_fold = glm_fold.predict(X_test_fold)
    pred_labels_fold = [1 if p > 0.5 else 0 for p in preds_fold]
    fold_error = np.mean(pred_labels_fold != y_test_fold)

    cv_errors.append(fold_error)

cv_error_rate = np.mean(cv_errors)
print(f"K-fold Cross-Validation Error Rate: {cv_error_rate:.5f}")
```

```
K-fold Cross-Validation Error Rate: 0.02571
```

[9]:
```
# ---- In-sample predictions ----
glm_probs_train = glm_fit.predict(X_train)
glm_pred_train = np.where(glm_probs_train > 0.5, 1, 0)  # ternary operator

# Compute in-sample accuracy and error rate
accuracy_train = accuracy_score(y_train, glm_pred_train)
error_rate_train = np.mean(glm_pred_train != y_train)

print(f"In-sample accuracy: {round(accuracy_train, 5)}")
print(f"In-sample error rate: {round(error_rate_train, 5)}")
```

```
In-sample accuracy: 0.97486
In-sample error rate: 0.02514
```

[10]:
```
# ---- Out-of-sample predictions ----
glm_probs_test = glm_fit.predict(X_test)
glm_pred_test = np.where(glm_probs_test > 0.5, 1, 0)  # ternary operator
```

[11]:
```
# Compute out-of-sample accuracy and error rate
accuracy_test = accuracy_score(test_data_copy['default'], glm_pred_test)
error_rate_test = np.mean(glm_pred_test != test_data_copy['default'])

print(f"Out-of-sample accuracy: {round(accuracy_test, 5)}")
print(f"Out-of-sample error rate: {round(error_rate_test, 5)}")
```

```
Out-of-sample accuracy: 0.97067
Out-of-sample error rate: 0.02933
```