Machine Learning for Economics and Finance
Task 1: Logistic Regressions - solution

02_Default_data_solution

Ole Wilms

July 29, 2024

Task 1: Logistic Regressions

1.1 Randomly split the data into 7000 observations for training and 3000 observations for testing and set the seed to 1 before sampling the data. Call these two datasets *train_data* and *test_data* respectively. (**Hint**: use the code to split the data from *01_Auto_data.ipynb*)

[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

# pip install ISLP --break-system-packages
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/02-SupLearning_Class
```

[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.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]:
```
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
```

Compute the percentage of defaulting ("Yes") values in the `default` column of the "train_data" and "test_data" datasets.

[5]:
```
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
```

1.2 Fit a logistic regression of default on *income* using the *train_data*. Analyze the significance of the estimated coefficients.

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

# Create a copy of the train_data DataFrame
# preventing to overwrite the original data
train_data_copy = train_data.copy()

# Ensure the target variable is numeric (binary)
train_data_copy['default'] = train_data_copy['default'].map({'No': 0, 'Yes': 1})
```

```
# Logistic regression model:
X_train = train_data_copy[['income']]
X_train = sm.add_constant(X_train)  # 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()

# Alternative:
#glm_fit = sm.Logit(y_train, X_train).fit()

print(glm_fit.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                default   No. Observations:                 7000
Model:                            GLM   Df Residuals:                     6998
Model Family:                Binomial   Df Model:                            1
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -979.69
Date:                Wed, 29 Jan 2025   Deviance:                       1959.4
Time:                        23:11:20   Pearson chi2:                 7.01e+03
No. Iterations:                     6   Pseudo R-squ. (CS):          0.0004155
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -3.1353      0.179    -17.492      0.000      -3.487      -2.784
income     -8.81e-06   5.18e-06     -1.700      0.089    -1.9e-05    1.35e-06
==============================================================================
```

1.3 Compute the *out-of-sample accuracy* and *error rate* and compare to the *in-sample statistics*. Do you think this is a good model to predict default?

[7]:
```
# Predict probabilities on the training data
logit_probs_train = glm_fit.predict(X_train)

# Predict default status based on the probability threshold (0.5)
logit_pred_train = np.where(logit_probs_train > 0.5, 1, 0)  # Ternary operator␣
 ↪usage:
# (If "logit_probs_train" is greater than "0.5" apply "1" else "0")

#np.unique(logit_pred_train)
#result: array([0, 1])  at logit_probs_train > 0.03

# Compute accuracy and error rate for the training set
accuracy_train = np.mean(logit_pred_train == train_data_copy['default'])
```

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

error_rate_train = np.mean(logit_pred_train != train_data_copy['default'])
print(f"In-sample error rate: {round(error_rate_train, 5)}")
```

```
In-sample accuracy:   0.96843
In-sample error rate: 0.03157
```

[8]:
```
# Create a copy of the test_data DataFrame
# preventing to overwrite the original data
test_data_copy = test_data.copy()

# Ensure the target variable is numeric (binary)
test_data_copy['default'] = test_data_copy['default'].map({'No': 0, 'Yes': 1})

# Logistic regression model:
X_test = test_data_copy[['income']]
X_test = sm.add_constant(X_test)  # Adds an intercept term to the model
y_test = test_data_copy['default']

logit_probs_test = glm_fit.predict(X_test)
logit_pred_test = [ 0 if x < 0.5 else 1 for x in logit_probs_test]

## EXTRA:
#from sklearn.metrics import classification_report
#print(classification_report(y_test,
#                            logit_pred_test,
#                            digits = 3))
```

[9]:
```
# Compute accuracy and error rate for the training set
accuracy_test = np.mean(logit_pred_test == test_data_copy['default'])
print(f"Out-of-sample accuracy:   {round(accuracy_test, 5)}")

error_rate_test = np.mean(logit_pred_test != test_data_copy['default'])
print(f"Out-of-sample error rate: {round(error_rate_test, 5)}")
```

```
Out-of-sample accuracy:   0.96267
Out-of-sample error rate: 0.03733
```

1.4 Add balance as a predictor and compute the *out-of-sample error rate* and *accuracy*. Do you think this is a good model to predict *default*?

[10]:
```
# Second logistic regression model:
X_train2 = train_data_copy[['income','balance']]
X_train2 = sm.add_constant(X_train2)  # Adds an intercept term to the model
X_test2 = test_data_copy[['income','balance']]
X_test2 = sm.add_constant(X_test2)  # Adds an intercept term to the model

# Fit the logistic regression model
```

```
glm_fit2 = sm.GLM(y_train, X_train2, family=sm.families.Binomial()).fit()

# Alternative:
#glm_fit = sm.Logit(y_train, X_train2).fit()

print(glm_fit2.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:                Wed, 29 Jan 2025   Deviance:                       1084.3
Time:                        23:11:25   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
==============================================================================
```

[11]:
```
logit_probs_test2 = glm_fit2.predict(X_test2)
logit_pred_test2 = [ 0 if x < 0.5 else 1 for x in logit_probs_test2]

# Compute accuracy and error rate for the training set
accuracy_test2 = np.mean(logit_pred_test2 == test_data_copy['default'])
print(f"Out-of-sample accuracy:   {round(accuracy_test2, 5)}")

error_rate_test2 = np.mean(logit_pred_test2 != test_data_copy['default'])
print(f"Out-of-sample error rate: {round(error_rate_test2, 5)}")
```

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

1.5 Compare the results for Task 1.4 to a model with only balance as a predictor. Which model would you choose?

[12]:
```
# Third logistic regression model:
X_train3 = train_data_copy[['balance']]
X_train3 = sm.add_constant(X_train3)  # Adds an intercept term to the model
X_test3 = test_data_copy[['balance']]
X_test3 = sm.add_constant(X_test3)  # Adds an intercept term to the model

# Fit the logistic regression model
```

```
glm_fit3 = sm.GLM(y_train, X_train3, family=sm.families.Binomial()).fit()

# Alternative:
#glm_fit = sm.Logit(y_train, X_train3).fit()

print(glm_fit3.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                default   No. Observations:                 7000
Model:                            GLM   Df Residuals:                     6998
Model Family:                Binomial   Df Model:                            1
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -546.92
Date:                Wed, 29 Jan 2025   Deviance:                       1093.8
Time:                        23:11:28   Pearson chi2:                 5.73e+03
No. Iterations:                     9   Pseudo R-squ. (CS):             0.1167
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -10.5908      0.435    -24.330      0.000     -11.444      -9.738
balance        0.0054      0.000     20.441      0.000       0.005       0.006
==============================================================================
```

[13]:
```
logit_probs_test3 = glm_fit3.predict(X_test3)
logit_pred_test3 = [ 0 if x < 0.5 else 1 for x in logit_probs_test3]

# Compute accuracy and error rate for the training set
accuracy_test3 = np.mean(logit_pred_test3 == test_data_copy['default'])
print(f"Out-of-sample accuracy:   {round(accuracy_test3, 5)}")

error_rate_test3 = np.mean(logit_pred_test3 != test_data_copy['default'])
print(f"Out-of-sample error rate: {round(error_rate_test3, 5)}")
```

```
Out-of-sample accuracy:   0.97033
Out-of-sample error rate: 0.02967
```

Extra:

[14]:
```
import pandas as pd

# Define the data
data = {
    "Model": ["income", "income + balance", "balance"],
    "In-sample error rate": [error_rate_train, '', ''],
    "Out-of-sample error rate": [error_rate_test, error_rate_test2,␣
 ↪error_rate_test3]
```

```
}

# Create a DataFrame
df = pd.DataFrame(data)

# Display the DataFrame
print(df)
```

```
              Model  In-sample error rate  Out-of-sample error rate
0            income              0.031571                  0.037333
1  income + balance                                        0.029333
2           balance                                        0.029667
```

1.6 Take the model from Task 1.4 but now re-estimate the model using different *seeds* to draw your *training* and *test data*. Does your *test error rate* change with the seed? What's going on here?

[15]:
```
# CHANGE SEED HERE:
np.random.seed(123)

# 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

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_train1_6 = train_data_copy[['income','balance']]
X_train1_6 = sm.add_constant(X_train1_6)  # Adds an intercept term to the model
X_test1_6 = test_data_copy[['income','balance']]
X_test1_6 = sm.add_constant(X_test1_6)  # Adds an intercept term to the model
y_train1_6 = train_data_copy['default']

# Fit the logistic regression model
glm_fit1_6 = sm.GLM(y_train1_6, X_train1_6, family=sm.families.Binomial()).fit()
```

```
logit_probs_test1_6 = glm_fit1_6.predict(X_test1_6)
logit_pred_test1_6 = [ 0 if x < 0.5 else 1 for x in logit_probs_test1_6]

error_rate_test1_6 = np.mean(logit_pred_test1_6 != test_data_copy['default'])
print(f"Out-of-sample error rate: {round(error_rate_test1_6, 10)}")
```

```
Out-of-sample error rate: 0.0253333333
```

[ ]:

[ ]:

[26]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from math import ceil
import io
from PIL import Image

# Assuming your DataFrame is named 'paneldata09_SL'
# If not, replace 'paneldata09_SL' with your actual DataFrame name

# Function to create a single plot
def create_plot(data, var):
    fig, ax = plt.subplots(figsize=(5, 4))
    if data[var].dtype in ['int64', 'float64']:
        sns.kdeplot(data=data, x=var, ax=ax)
        ax.set_title(f"Density Plot of {var}")
    else:
        sns.countplot(data=data, x=var, ax=ax)
        ax.set_title(f"Bar Plot of {var}")

    # Save the plot to a buffer
    buf = io.BytesIO()
    fig.savefig(buf, format='png')
    plt.close(fig)
    buf.seek(0)
    return Image.open(buf)

# Create a plot for each variable
plots = [create_plot(train_data, var) for var in train_data.columns]

# Calculate grid dimensions
n = len(plots)
cols = 3  # You can adjust this to change the number of columns
rows = ceil(n / cols)
```

```
# Create the grid plot
fig, axes = plt.subplots(rows, cols, figsize=(5*cols, 4*rows))
fig.suptitle("Plots of All Variables", fontsize=16)

# Add plots to the grid
for i, plot in enumerate(plots):
    row = i // cols
    col = i % cols
    axes[row, col].imshow(plot)
    axes[row, col].axis('off')
    axes[row, col].set_title(f"$\\mathbf{{{chr(65+i)}}}$: $\\quad\\quad\\quad$␣
 ↪{train_data.columns[i]}")

# Remove any empty subplots
for i in range(n, rows*cols):
    row = i // cols
    col = i % cols
    fig.delaxes(axes[row, col])

plt.tight_layout()
plt.show()
```