Multivariate statistical analysis

June 23, 2023

1 Exploratory Data Analysis (EDA)

Firstly, let's load the bank-additional-full.csv data and perform some exploratory data analysis on it. In Python, we can use the Pandas library to load and manipulate datasets. Here's code:

[20]:
```
import pandas as pd

# Load the dataset
data = pd.read_csv('bank-additional-full.csv')

# Print the first 5 rows of the dataset
print(data.head())

# Check for missing values
print(data.isnull().sum())

# Get descriptive statistics of the numerical variables
print(data.describe())
```

```
   age        job  marital    education  default housing loan    contact  \
0   56  housemaid  married     basic.4y       no      no   no  telephone
1   57   services  married  high.school  unknown      no   no  telephone
2   37   services  married  high.school       no     yes   no  telephone
3   40     admin.  married     basic.6y       no      no   no  telephone
4   56   services  married  high.school       no      no  yes  telephone

  month day_of_week  …  campaign  pdays  previous     poutcome emp.var.rate  \
0   may         mon  …         1    999         0  nonexistent          1.1
1   may         mon  …         1    999         0  nonexistent          1.1
2   may         mon  …         1    999         0  nonexistent          1.1
3   may         mon  …         1    999         0  nonexistent          1.1
4   may         mon  …         1    999         0  nonexistent          1.1

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y
0          93.994          -36.4      4.857       5191.0  no
1          93.994          -36.4      4.857       5191.0  no
2          93.994          -36.4      4.857       5191.0  no
3          93.994          -36.4      4.857       5191.0  no
```

```
4          93.994          -36.4      4.857       5191.0  no

[5 rows x 21 columns]
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
               age      duration      campaign         pdays      previous  \
count  41188.00000  41188.000000  41188.000000  41188.000000  41188.000000
mean      40.02406    258.285010      2.567593    962.475454      0.172963
std       10.42125    259.279249      2.770014    186.910907      0.494901
min       17.00000      0.000000      1.000000      0.000000      0.000000
25%       32.00000    102.000000      1.000000    999.000000      0.000000
50%       38.00000    180.000000      2.000000    999.000000      0.000000
75%       47.00000    319.000000      3.000000    999.000000      0.000000
max       98.00000   4918.000000     56.000000    999.000000      7.000000

       emp.var.rate  cons.price.idx  cons.conf.idx     euribor3m   nr.employed
count  41188.000000    41188.000000   41188.000000  41188.000000  41188.000000
mean       0.081886       93.575664     -40.502600      3.621291   5167.035911
std        1.570960        0.578840       4.628198      1.734447     72.251528
min       -3.400000       92.201000     -50.800000      0.634000   4963.600000
25%       -1.800000       93.075000     -42.700000      1.344000   5099.100000
50%        1.100000       93.749000     -41.800000      4.857000   5191.000000
75%        1.400000       93.994000     -36.400000      4.961000   5228.100000
max        1.400000       94.767000     -26.900000      5.045000   5228.100000
```

The above code will load the dataset, print the first 5 rows, check for missing values, and get descriptive statistics of the numerical variables.

2 Label Encoding

Next, we need to encode the categorical variables using label encoding. Label encoding is a process of converting categorical variables into numerical form so that they can be used in machine learning models. In Python, we can use Scikit-learn's LabelEncoder class to perform label encoding. Here's the code:

[21]:
```python
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object
le = LabelEncoder()

# Encode the categorical variables
data['job'] = le.fit_transform(data['job'])
data['marital'] = le.fit_transform(data['marital'])
data['education'] = le.fit_transform(data['education'])
data['default'] = le.fit_transform(data['default'])
data['housing'] = le.fit_transform(data['housing'])
data['loan'] = le.fit_transform(data['loan'])
data['contact'] = le.fit_transform(data['contact'])
data['month'] = le.fit_transform(data['month'])
data['day_of_week'] = le.fit_transform(data['day_of_week'])
data['poutcome'] = le.fit_transform(data['poutcome'])
data['y'] = le.fit_transform(data['y'])

# Print the first 5 rows of the dataset after encoding
print(data.head())
```

```
   age  job  marital  education  default  housing  loan  contact  month  \
0   56    3        1          0        0        0     0        1      6
1   57    7        1          3        1        0     0        1      6
2   37    7        1          3        0        2     0        1      6
3   40    0        1          1        0        0     0        1      6
4   56    7        1          3        0        0     2        1      6

   day_of_week  …  campaign  pdays  previous  poutcome  emp.var.rate  \
0            1  …         1    999         0         1           1.1
1            1  …         1    999         0         1           1.1
2            1  …         1    999         0         1           1.1
3            1  …         1    999         0         1           1.1
4            1  …         1    999         0         1           1.1

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed  y
0          93.994          -36.4      4.857       5191.0  0
1          93.994          -36.4      4.857       5191.0  0
2          93.994          -36.4      4.857       5191.0  0
3          93.994          -36.4      4.857       5191.0  0
4          93.994          -36.4      4.857       5191.0  0
```

```
[5 rows x 21 columns]
```

The above code will create a LabelEncoder object, encode the categorical variables, and print the first 5 rows of the dataset after encoding.

3 Train Test Split

After encoding the dataset, the next step is to split the data into training and testing sets. In Python, we can use Scikit-learn's train_test_split function to perform this operation. Here's the code:

[22]:
```python
from sklearn.model_selection import train_test_split

# Split the data into X (features) and y (target)
X = data.drop('y', axis=1)
y = data['y']

# Split the data into training and testing sets with a 65:35 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, 
 ↪random_state=42)

# Print the shape of the training and testing sets
print("Training set shape:", X_train.shape, y_train.shape)
print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (26772, 20) (26772,)
Testing set shape: (14416, 20) (14416,)
```

3.1 randomization

`random_state = 42` is a parameter commonly used in machine learning algorithms, including scikit-learn, to ensure reproducibility of the results. When this parameter is set to a specific value, such as 42, it will initialize the random number generator used by the algorithm with this seed value. This means that every time the algorithm is run with the same seed value, it will produce the same sequence of random numbers, which in turn will result in the same set of outputs.

The way the randomization occurs depends on the specific algorithm being used. However, in general, the algorithm will use a pseudo-random number generator (PRNG) to generate a sequence of apparently random numbers based on a predetermined algorithm. The PRNG generates these numbers deterministically, based on an initial seed value and a mathematical formula, so they are not truly random. However, the resulting sequence appears to be random for practical purposes.

By setting the `random_state` parameter to a fixed value, we can ensure that the same sequence of "random" numbers is generated each time we run the algorithm. This can be useful for testing and debugging, as well as ensuring that results are consistent across different runs.

4 Logistic Regression Model

Now that we have our training and testing sets, we can build a logistic regression model. Logistic regression is a method for analyzing a dataset in which there are one or more independent variables that determine an outcome. It is commonly used for binary classification problems. In Python, we can use Scikit-learn's LogisticRegression class to build the model. Here's The code:

[23]:
```
from sklearn.linear_model import LogisticRegression

# Create a Logistic Regression object
lr = LogisticRegression()

# Fit the model using the training data
lr.fit(X_train, y_train)

# Find the coefficients
coefs = lr.coef_

# Predict the target variable using the testing data
y_pred = lr.predict(X_test)

# Print the accuracy score of the model
print("Accuracy score:", lr.score(X_test, y_test))
print("The coefficients are: {}".format(coefs))
```

```
Accuracy score: 0.9088512763596004
The coefficients are: [[ 0.00190464  0.02648694  0.02244785  0.09336503
-0.0258161   0.00675698
  -0.00245879 -0.04032106 -0.03759216  0.02210143  0.00456907 -0.079183
  -0.00164802  0.01626544  0.00058085 -0.26201575  0.19371844  0.04785373
  -0.27560908 -0.00349051]]
```

```
/home/parsa/miniconda3/envs/ai/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

The above code will create a Logistic Regression object, fit the model using the training data, predict the target variable using the testing data, and print the accuracy score of the model.

5 Summary of the Model

We can get a summary of the logistic regression model using the statsmodels library. Here's The example code:

[24]:
```
import statsmodels.api as sm
```

[25]:
```
# Add a constant column to the X_train dataset
X_train_sm = sm.add_constant(X_train)

# Create a Logistic Regression object using statsmodels
lr_sm = sm.Logit(y_train, X_train_sm)

# Fit the model using the training data
lr_sm_fit = lr_sm.fit()

# Print the summary of the model
print(lr_sm_fit.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.214037
         Iterations 9
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                26772
Model:                          Logit   Df Residuals:                    26751
Method:                           MLE   Df Model:                           20
Date:                Fri, 23 Jun 2023   Pseudo R-squ.:                  0.3907
Time:                        20:52:24   Log-Likelihood:                -5730.2
converged:                       True   LL-Null:                       -9403.9
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
==
                     coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--
const            -10.6339     23.920     -0.445      0.657     -57.517
36.249
age                0.0017      0.002      0.733      0.464      -0.003
0.006
job                0.0020      0.007      0.292      0.770      -0.012
0.016
marital            0.0503      0.045      1.123      0.261      -0.037
0.138
education          0.0449      0.012      3.673      0.000       0.021
0.069
default           -0.3532      0.081     -4.342      0.000      -0.513
```

```
-0.194
housing           -0.0312      0.025     -1.238      0.216      -0.081
0.018
loan              -0.0139      0.035     -0.404      0.686      -0.082
0.054
contact           -0.7680      0.080     -9.584      0.000      -0.925
-0.611
month             -0.1134      0.012     -9.724      0.000      -0.136
-0.091
day_of_week        0.0635      0.018      3.530      0.000       0.028
0.099
duration           0.0045   8.99e-05     50.078      0.000       0.004
0.005
campaign          -0.0442      0.014     -3.086      0.002      -0.072
-0.016
pdays             -0.0011      0.000     -5.429      0.000      -0.001
-0.001
previous          -0.0758      0.069     -1.099      0.272      -0.211
0.059
poutcome           0.4230      0.095      4.447      0.000       0.237
0.609
emp.var.rate      -1.0436      0.085    -12.336      0.000      -1.209
-0.878
cons.price.idx     0.7980      0.148      5.391      0.000       0.508
1.088
cons.conf.idx      0.0155      0.008      1.855      0.064      -0.001
0.032
euribor3m          0.7583      0.127      5.993      0.000       0.510
1.006
nr.employed       -0.0134      0.002     -5.893      0.000      -0.018
-0.009
================================================================================
==
```

6 Discriminant Analysis

Another classification method we can use is discriminant analysis. Discriminant analysis is a statistical technique used to identify the underlying factors that differentiate between two or more groups. In Python, we can use Scikit-learn's LinearDiscriminantAnalysis class to perform discriminant analysis. Here's theexample code:

[26]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Create a Linear Discriminant Analysis object
lda = LinearDiscriminantAnalysis()

# Fit the model using the training data
```

```python
lda.fit(X_train, y_train)

# Predict the target variable using the testing data
y_pred_lda = lda.predict(X_test)

# Print the accuracy score of the model
print("Accuracy score:", lda.score(X_test, y_test))
```

```
Accuracy score: 0.9070477247502775
```

The above code will create a Linear Discriminant Analysis object, fit the model using the training data, predict the target variable using the testing data, and print the accuracy score of the model.

7 Evaluation of the Model

To evaluate the performance of the logistic regression and discriminant analysis models, we can use metrics such as accuracy score, precision, recall, and F1-score. In Python, we can use Scikit-learn's classification_report function to get these metrics. Here's an example code:

[27]:
```python
from sklearn.metrics import classification_report

# Print the classification report for logistic regression model
print("Logistic Regression")
print(classification_report(y_test, y_pred))

# Print the classification report for LDA model
print("Linear Discriminant Analysis")
print(classification_report(y_test, y_pred_lda))
```

```
Logistic Regression
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     12782
           1       0.66      0.40      0.50      1634

    accuracy                           0.91     14416
   macro avg       0.79      0.69      0.73     14416
weighted avg       0.90      0.91      0.90     14416

Linear Discriminant Analysis
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     12782
           1       0.61      0.48      0.54      1634

    accuracy                           0.91     14416
   macro avg       0.78      0.72      0.74     14416
weighted avg       0.90      0.91      0.90     14416
```

The above code will print the classification report for both the logistic regression and LDA models. The classification report includes information such as precision, recall, F1-score, and support for each class.