ML_Finance_HW1

February 6, 2023

0.0.1 • Goal: Use machine learning to solve a binary classification problem using provided dataset.

0.0.2 • Background: The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).

0.0.3 • Author: Dhyey Dharmendrakumar Mavani

[40]:
```
# checking the current working directory access
import os
print(os.getcwd())
```

```
/Volumes/GoogleDrive/My Drive/Columbia VUS/SPRING2023/MATH GR 5430 MACHINE
LEARNING FOR FINANCE/ML_Finance_HW1
```

1 1. Setup and Data Fetching

- **Obtain and import the dataset (bank-additional-full.csv) from Courseworks as a Pandas dataframe.**

[42]:
```
# importing the data from the given csv (with cols separated by ";")
import pandas as pd
bank_data = pd.read_csv("./bank-additional-full.csv", sep = ";")
bank_data
```

[42]:
```
       age          job  marital            education  default housing loan  \
0       56    housemaid  married             basic.4y       no      no   no
1       57     services  married          high.school  unknown      no   no
2       37     services  married          high.school       no     yes   no
3       40       admin.  married             basic.6y       no      no   no
4       56     services  married          high.school       no      no  yes
…      …           …        …                   …       …      …    …
41183   73      retired  married  professional.course       no     yes   no
41184   46  blue-collar  married  professional.course       no      no   no
41185   56      retired  married    university.degree       no     yes   no
41186   44   technician  married  professional.course       no      no   no
41187   74      retired  married  professional.course       no     yes   no

          contact month day_of_week  …  campaign  pdays  previous  \
0      telephone   may         mon  …        1    999         0
1      telephone   may         mon  …        1    999         0
2      telephone   may         mon  …        1    999         0
3      telephone   may         mon  …        1    999         0
4      telephone   may         mon  …        1    999         0
…          …   …           …  …        …    …       …
41183   cellular   nov         fri  …        1    999         0
41184   cellular   nov         fri  …        1    999         0
41185   cellular   nov         fri  …        2    999         0
41186   cellular   nov         fri  …        1    999         0
41187   cellular   nov         fri  …        3    999         1

          poutcome emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
0      nonexistent          1.1          93.994          -36.4      4.857
1      nonexistent          1.1          93.994          -36.4      4.857
2      nonexistent          1.1          93.994          -36.4      4.857
3      nonexistent          1.1          93.994          -36.4      4.857
4      nonexistent          1.1          93.994          -36.4      4.857
…             …          …              …             …         …
41183  nonexistent         -1.1          94.767          -50.8      1.028
41184  nonexistent         -1.1          94.767          -50.8      1.028
41185  nonexistent         -1.1          94.767          -50.8      1.028
41186  nonexistent         -1.1          94.767          -50.8      1.028
41187      failure         -1.1          94.767          -50.8      1.028

       nr.employed    y
0           5191.0   no
1           5191.0   no
2           5191.0   no
3           5191.0   no
4           5191.0   no
…             …   …
41183       4963.6  yes
41184       4963.6   no
41185       4963.6   no
41186       4963.6  yes
41187       4963.6   no

[41188 rows x 21 columns]
```

• **Use bank-additional-names.txt doc associated with the dataset as complement information to get familiar with attributes.** From the (bank-additional-names.txt) file we can see that input and output variables are defined as follows:

1.0.1 Input variables:

Bank client data:

1 - `age` (numeric)

2 - `job`: type of job (categorical: “admin.”,“blue-collar”,“entrepreneur”,“housemaid”,“management”,“retired”,“self-employed”,“services”,“student”,“technician”,“unemployed”,“unknown”)

3 - `marital` : marital status (categorical: “divorced”,“married”,“single”,“unknown”; note: “divorced” means divorced or widowed)

4 - `education` (categorical: “basic.4y”,“basic.6y”,“basic.9y”,“high.school”,“illiterate”,“professional.course”,“univer

5 - `default`: has credit in default? (categorical: “no”,“yes”,“unknown”)

6 - `housing`: has housing loan? (categorical: “no”,“yes”,“unknown”)

7 - `loan`: has personal loan? (categorical: “no”,“yes”,“unknown”)

Related with the last contact of the current campaign:

8 - `contact`: contact communication type (categorical: “cellular”,“telephone”)

9 - `month`: last contact month of year (categorical: “jan”, “feb”, “mar”, …, “nov”, “dec”)

10 - `day_of_week`: last contact day of the week (categorical: “mon”,“tue”,“wed”,“thu”,“fri”)

11 - `duration`: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y=“no”). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

Other attributes:

12 - `campaign`: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13 - `pdays`: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

14 - `previous`: number of contacts performed before this campaign and for this client (numeric)

15 - `poutcome`: outcome of the previous marketing campaign (categorical: “failure”,“nonexistent”,“success”)

Social and economic context attributes

16 - `emp.var.rate`: employment variation rate - quarterly indicator (numeric)

17 - `cons.price.idx`: consumer price index - monthly indicator (numeric)

18 - `cons.conf.idx`: consumer confidence index - monthly indicator (numeric)

19 - `euribor3m`: euribor 3 month rate - daily indicator (numeric)

20 - `nr.employed`: number of employees - quarterly indicator (numeric)

1.0.2 Output variable (desired target):

21 - `y` - has the client subscribed a term deposit? (binary: “yes”,“no”)

2 2. Data Preprocessing

- **What is the balance within the y variable? (i.e. how many positives and negatives are there, is this relationship balanced?) How would you consider improving the dataset given your observations?**

[43]:
```
print(bank_data['y'].value_counts())
```

```
no     36548
yes     4640
Name: y, dtype: int64
```

We already know that there are a total of 41188 rows, which means we have 41188 different observations. From the output of the code chunk above, we can see that there are a total of 4640 positives ("yes" observations) and 36548 neagtives ("no" observations). I don't think the relationship is balanced. And, to further improve the dataset, I would try to have around equal number of positives and negatives.

- **Missing Attribute Values: There are several missing values in some categorical attributes, all coded with the "unknown" label. These missing values can be treated as a possible class label or using deletion or imputation techniques. (To-do) Choose the way you see as appropriate to clean the dataframe so that there are no missing values.**

[45]:
```
# I have chosen to delete the observations in which at least one of the column␣
 ↪values is set to "unknown"
import numpy as np
bank_data.replace("unknown", np.nan, inplace=True)
bank_data.dropna(inplace=True)
bank_data
```

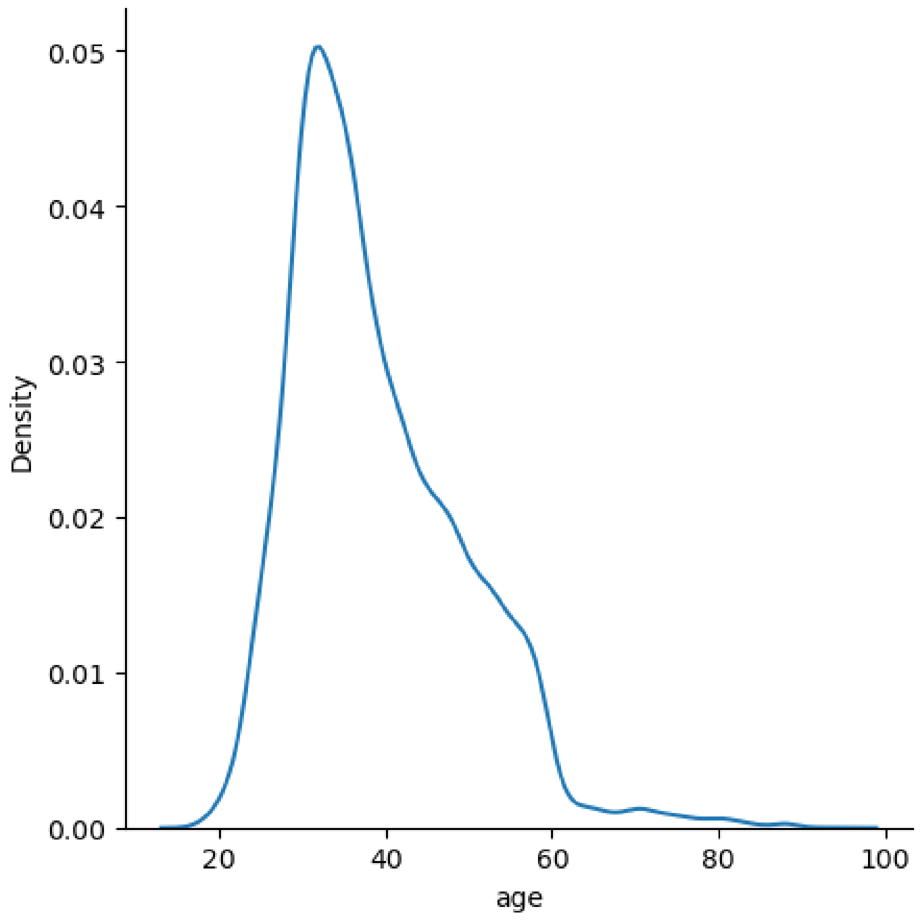
[45]:
```
       age          job  marital          education default housing loan  \
0       56    housemaid  married           basic.4y      no      no   no
2       37     services  married        high.school      no     yes   no
3       40       admin.  married           basic.6y      no      no   no
4       56     services  married        high.school      no      no  yes
6       59       admin.  married  professional.course      no      no   no
…     …          …       …                  …       …       …    …
41183   73      retired  married  professional.course      no     yes   no
41184   46  blue-collar  married  professional.course      no      no   no
41185   56      retired  married    university.degree      no     yes   no
41186   44   technician  married  professional.course      no      no   no
41187   74      retired  married  professional.course      no     yes   no

         contact month day_of_week  …  campaign  pdays  previous  \
0      telephone   may         mon  …         1    999         0
2      telephone   may         mon  …         1    999         0
3      telephone   may         mon  …         1    999         0
4      telephone   may         mon  …         1    999         0
```

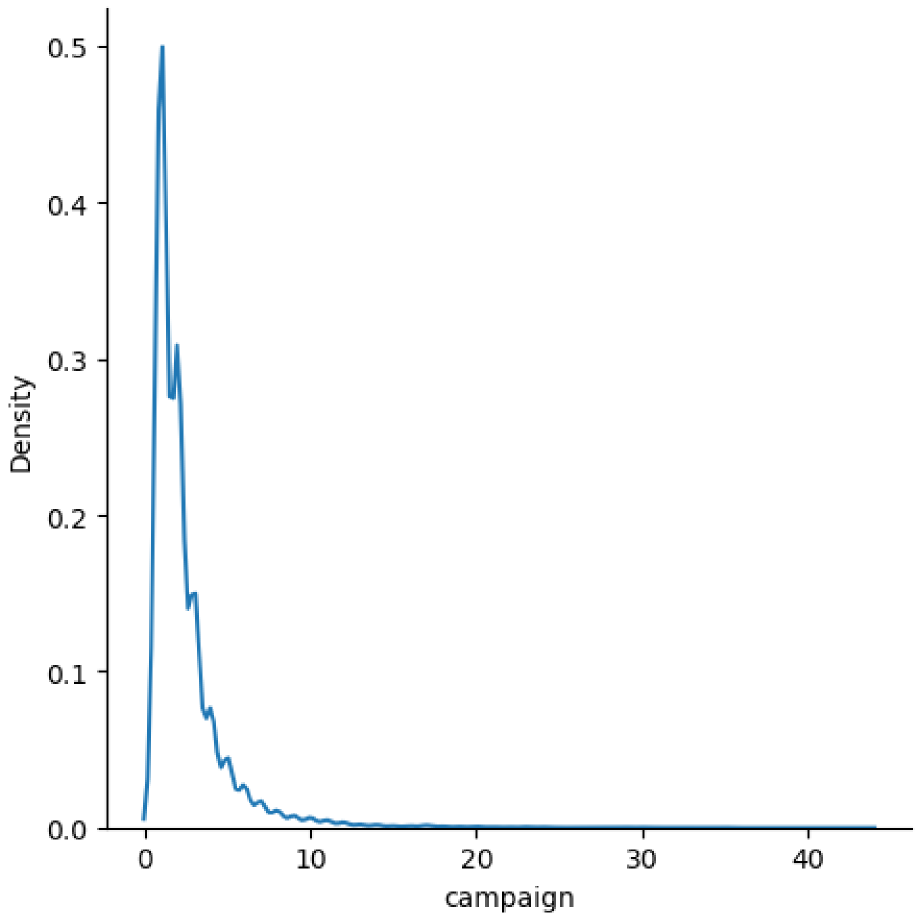
```
6      telephone   may        mon  …        1    999         0
…           …   …          …   …         …     …        …
41183   cellular   nov        fri  …        1    999         0
41184   cellular   nov        fri  …        1    999         0
41185   cellular   nov        fri  …        2    999         0
41186   cellular   nov        fri  …        1    999         0
41187   cellular   nov        fri  …        3    999         1

          poutcome emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
0      nonexistent          1.1          93.994          -36.4      4.857
2      nonexistent          1.1          93.994          -36.4      4.857
3      nonexistent          1.1          93.994          -36.4      4.857
4      nonexistent          1.1          93.994          -36.4      4.857
6      nonexistent          1.1          93.994          -36.4      4.857
…              …          …              …             …        …
41183  nonexistent         -1.1          94.767          -50.8      1.028
41184  nonexistent         -1.1          94.767          -50.8      1.028
41185  nonexistent         -1.1          94.767          -50.8      1.028
41186  nonexistent         -1.1          94.767          -50.8      1.028
41187      failure         -1.1          94.767          -50.8      1.028

       nr.employed    y
0           5191.0   no
2           5191.0   no
3           5191.0   no
4           5191.0   no
6           5191.0   no
…              …   …
41183       4963.6  yes
41184       4963.6   no
41185       4963.6   no
41186       4963.6  yes
41187       4963.6   no

[30488 rows x 21 columns]
```

- **Plot the distributions of two numeric independent variables of your choice. What do you discover about those variables?**

[46]:
```python
import seaborn as sns
sns.displot(bank_data, x="age", kind="kde")
sns.displot(bank_data, x="campaign", kind="kde")
```

[46]:
```
<seaborn.axisgrid.FacetGrid at 0x7fb193872550>
```

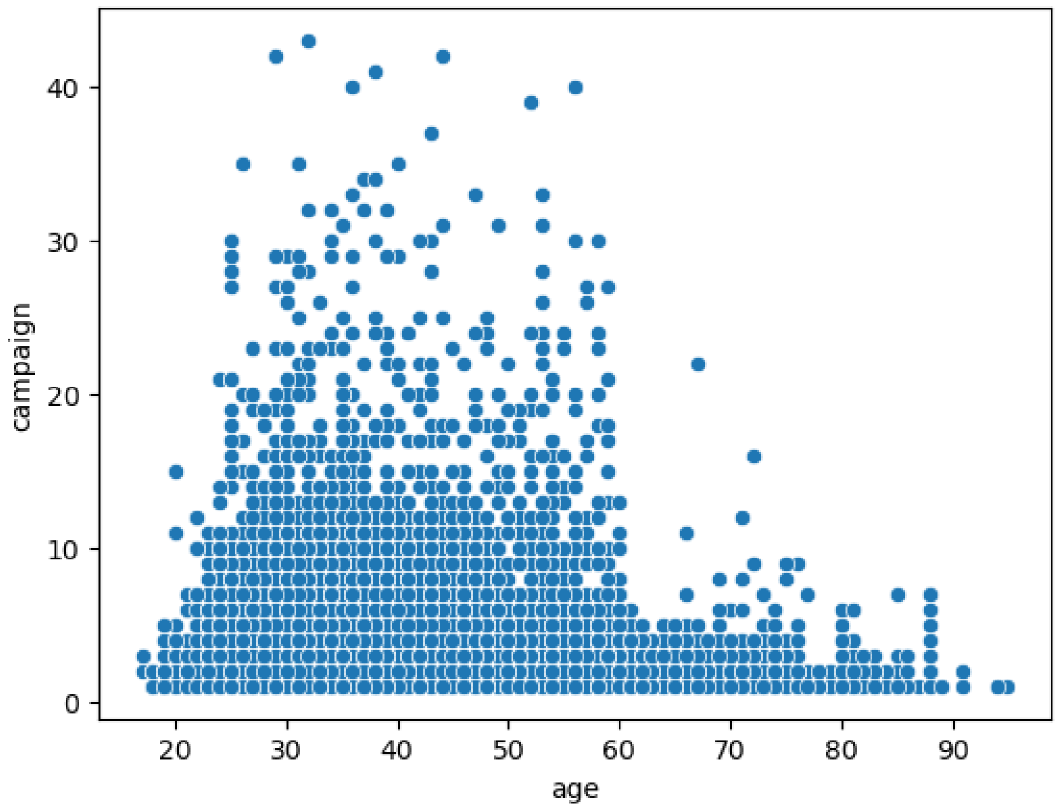

0.05
0.04
0.03
0.02
0.01
0.00
Density
20
40
60
80
100
age



We have chosen here to plot the density distributions of numeric variables age and campaign in our filtered dataset. We can see that both the distributions are right-skewed, while that of campaign is more so. Furthermore, we can see that the age with most density is around 35, while the density of campaign variable peaks when campaign takes a value of 2. We can also see that age mostly lies between 20 to 60, while campaign values mostly lie between 0 to 10.

```
[47]: sns.scatterplot(data=bank_data, x="age", y="campaign")
```

```
[47]: <AxesSubplot: xlabel='age', ylabel='campaign'>
```



Furthermore, on plotting the numeric variables of interest as a scatterplot with `campaign` on the y-axis and `age` on x-axis as shown above, we can see that most of the observations with more than 20 value of campaign is observed with ages below 60.

- **Plot the correlation matrix of all numeric features. What do you discover? How should you handle it in that situation?**

[48]:
```
bank_data.corr()
```

[48]:
```
                     age  duration  campaign     pdays  previous  \
age             1.000000  0.007910 -0.002364 -0.050891  0.049231
duration        0.007910  1.000000 -0.068406 -0.046239  0.018772
campaign       -0.002364 -0.068406  1.000000  0.054312 -0.080766
pdays          -0.050891 -0.046239  0.054312  1.000000 -0.590248
previous        0.049231  0.018772 -0.080766 -0.590248  1.000000
emp.var.rate   -0.050409 -0.023374  0.157739  0.268763 -0.403502
cons.price.idx -0.035762  0.012768  0.127260  0.068010 -0.176775
cons.conf.idx   0.125017 -0.009525 -0.011664 -0.102368 -0.027930
euribor3m      -0.036481 -0.028922  0.140836  0.295188 -0.438863
nr.employed    -0.064586 -0.040367  0.148069  0.370845 -0.488365

                emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
age                 -0.050409       -0.035762       0.125017  -0.036481
duration            -0.023374        0.012768      -0.009525  -0.028922
campaign             0.157739        0.127260      -0.011664   0.140836
pdays                0.268763        0.068010      -0.102368   0.295188
previous            -0.403502       -0.176775      -0.027930  -0.438863
emp.var.rate         1.000000        0.766055       0.157593   0.969412
cons.price.idx       0.766055        1.000000       0.027217   0.667292
cons.conf.idx        0.157593        0.027217       1.000000   0.243637
euribor3m            0.969412        0.667292       0.243637   1.000000
nr.employed          0.900390        0.488871       0.075283   0.944871

                nr.employed
age               -0.064586
duration          -0.040367
campaign           0.148069
pdays              0.370845
previous          -0.488365
emp.var.rate       0.900390
cons.price.idx     0.488871
cons.conf.idx      0.075283
euribor3m          0.944871
nr.employed        1.000000
```

We can see that some of the correlations are unusually high, for example, 0.969412 between `euribor3m` and `emp.var.rate`, 0.900390 between `nr.employed` and `emp.var.rate`, and 0.944871 between `euribor3m` and `nr.employed`. This might be causing multi-collinearity issues down the line. Some of the solutions which we can employ in this case range from removing the highly correlated independent variables to introducing a linear combination of the highly correlated independent variables instead of having them individually present as columns.

- **Appropriately encode all categorical features in the dataframe**

[49]:
```
cleanup_bank_data = {"job": {"admin.": 1,
                             "blue-collar": 2,
                             "entrepreneur": 3,
                             "housemaid": 4,
                             "management": 5,
                             "retired": 6,
                             "self-employed": 7,
                             "services": 8,
                             "student": 9,
                             "technician": 10,
                             "unemployed": 11
                            },
                     "marital": {"divorced": 1,
                                 "married": 2,
                                 "single": 3
                                },
 "education": {"basic.4y": 1,
               "basic.6y": 2,
               "basic.9y": 3,
               "high.school": 4,
               "illiterate": 5,
               "professional.course": 6,
               "university.degree": 7
              },
 "default": {"no": 1,
             "yes": 2
            },
 "housing": {"no": 1,
             "yes": 2
            },
 "loan": {"no": 1,
          "yes": 2
         },
 "contact": {"cellular": 1,
             "telephone": 2
            },
 "month": {"jan": 1,
           "feb": 2,
           "mar": 3,
           "apr": 4,
           "may": 5,
           "jun": 6,
           "jul": 7,
           "aug": 8,
           "sep": 9,
           "oct": 10,
           "nov": 11,
           "dec": 12
          },
 "day_of_week": {"mon": 1,
                 "tue": 2,
                 "wed": 3,
                 "thu": 4,
                 "fri": 5
                },
 "poutcome": {"failure": 1,
              "nonexistent": 2,
              "success": 3
             },
 "y": {"no": 0,
       "yes": 1
      }
}
```

```
bank_data = bank_data.replace(cleanup_bank_data)
bank_data
```

```
[49]:        age  job  marital  education  default  housing  loan  contact  month  \
0       56    4        2          1        1        1     1        2      5
2       37    8        2          4        1        2     1        2      5
3       40    1        2          2        1        1     1        2      5
4       56    8        2          4        1        1     2        2      5
6       59    1        2          6        1        1     1        2      5
…     …  …    …        …       …      …    …       …   …
41183   73    6        2          6        1        2     1        1     11
41184   46    2        2          6        1        1     1        1     11
41185   56    6        2          7        1        2     1        1     11
41186   44   10        2          6        1        1     1        1     11
41187   74    6        2          6        1        2     1        1     11

       day_of_week  …  campaign  pdays  previous  poutcome  emp.var.rate  \
0                1  …         1    999         0         2           1.1
2                1  …         1    999         0         2           1.1
3                1  …         1    999         0         2           1.1
4                1  …         1    999         0         2           1.1
6                1  …         1    999         0         2           1.1
…              …  …       …     …       …        …           …
41183            5  …         1    999         0         2          -1.1
41184            5  …         1    999         0         2          -1.1
41185            5  …         2    999         0         2          -1.1
41186            5  …         1    999         0         2          -1.1
41187            5  …         3    999         1         1          -1.1

       cons.price.idx  cons.conf.idx  euribor3m  nr.employed  y
0              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
6              93.994          -36.4      4.857       5191.0  0
…                 …              …          …          … ..
41183          94.767          -50.8      1.028       4963.6  1
41184          94.767          -50.8      1.028       4963.6  0
41185          94.767          -50.8      1.028       4963.6  0
41186          94.767          -50.8      1.028       4963.6  1
41187          94.767          -50.8      1.028       4963.6  0

[30488 rows x 21 columns]
```

3 3. Model Configuration and Testing

- **Split the data into train and test using 20% test size.** splitting the `bank_data` into 80% `training_bank_data` and 20% `testing_bank_data`

[54]:
```
training_bank_data = bank_data.sample(frac=0.8, random_state=8990)
testing_bank_data = bank_data.drop(training_bank_data.index)

print(f"No. of training examples: {training_bank_data.shape[0]}")
print(f"No. of testing examples:   {testing_bank_data.shape[0]}")

# No. of training examples: 24390
# No. of testing examples:   6098
```

```
No. of training examples: 24390
No. of testing examples:   6098
```

[55]:
```
training_bank_data
```

[55]:
```
       age  job  marital  education  default  housing  loan  contact  month  \
15303   25    8        3          4        1        2     1        1      7
1014    38    2        3          3        1        1     1        2      5
36221   35    1        2          4        1        2     2        1      5
21297   31   10        2          6        1        1     1        1      8
36669   50    3        2          3        1        1     1        2      6
…      …    …       …          …        …       …     …       …      …
12277   50    3        2          6        1        2     1        2      7
5319    46    1        2          4        1        2     1        2      5
20294   36   10        3          6        1        1     1        1      8
10168   45    2        3          2        1        1     1        2      6
31578   36    1        3          4        1        2     2        1      5

       day_of_week  …  campaign  pdays  previous  poutcome  emp.var.rate  \
15303            5  …         2    999         0         2           1.4
1014             3  …         5    999         0         2           1.1
36221            4  …         2    999         0         2          -1.8
21297            1  …         1    999         0         2           1.4
36669            5  …         1    999         0         2          -2.9
…              …  …       …      …        …         …
12277            4  …         2    999         0         2           1.4
5319             5  …         4    999         0         2           1.1
20294            1  …         4    999         0         2           1.4
10168            4  …         2    999         0         2           1.4
31578            4  …         1    999         0         2          -1.8

       cons.price.idx  cons.conf.idx  euribor3m  nr.employed  y
15303          93.918          -42.7      4.957       5228.1  0
1014           93.994          -36.4      4.856       5191.0  0
```

```
36221          92.893          -46.2      1.270       5099.1  0
21297          93.444          -36.1      4.963       5228.1  0
36669          92.963          -40.8      1.268       5076.2  0
…               …             …        …          … ..
12277          93.918          -42.7      4.966       5228.1  0
5319           93.994          -36.4      4.857       5191.0  0
20294          93.444          -36.1      4.965       5228.1  0
10168          94.465          -41.8      4.958       5228.1  0
31578          92.893          -46.2      1.327       5099.1  0

[24390 rows x 21 columns]
```

[56]:
```
testing_bank_data
```

[56]:
```
       age  job  marital  education  default  housing  loan  contact  month  \
3       40    1        2          2        1        1     1        2      5
4       56    8        2          4        1        1     2        2      5
8       24   10        3          6        1        2     1        2      5
13      57    4        1          1        1        2     1        2      5
16      35    2        2          2        1        2     1        2      5
…     …   …      …        …       …      …    …       …     …
41173   62    6        2          7        1        2     1        1     11
41177   57    6        2          6        1        2     1        1     11
41179   64    6        1          6        1        2     1        1     11
41186   44   10        2          6        1        1     1        1     11
41187   74    6        2          6        1        2     1        1     11

       day_of_week  …  campaign  pdays  previous  poutcome  emp.var.rate  \
3                1  …         1    999         0         2           1.1
4                1  …         1    999         0         2           1.1
8                1  …         1    999         0         2           1.1
13               1  …         1    999         0         2           1.1
16               1  …         1    999         0         2           1.1
…              …  …      …     …       …       …         …
41173            4  …         1    999         2         1          -1.1
41177            4  …         6    999         0         2          -1.1
41179            5  …         3    999         0         2          -1.1
41186            5  …         1    999         0         2          -1.1
41187            5  …         3    999         1         1          -1.1

       cons.price.idx  cons.conf.idx  euribor3m  nr.employed  y
3              93.994          -36.4      4.857       5191.0  0
4              93.994          -36.4      4.857       5191.0  0
8              93.994          -36.4      4.857       5191.0  0
13             93.994          -36.4      4.857       5191.0  0
16             93.994          -36.4      4.857       5191.0  0
…               …             …        …          … ..
```

```
41173          94.767          -50.8      1.031       4963.6  1
41177          94.767          -50.8      1.031       4963.6  0
41179          94.767          -50.8      1.028       4963.6  0
41186          94.767          -50.8      1.028       4963.6  1
41187          94.767          -50.8      1.028       4963.6  0

[6098 rows x 21 columns]
```

- **Fit a logistic regression model using the Scikit-learn library (Hint: from sklearn.linear model import LogisticRegression).**

[78]:
```
from sklearn.linear_model import LogisticRegression

logistic_model = LogisticRegression(solver='lbfgs',class_weight='balanced',␣
 ↪max_iter=10000)

# split dataset in features and target variable
# note that we got rid of euribor3m and nr.employed
# in order to fix the multicollinearity issue
feature_cols = ['age', 'job', 'marital', 'education',
                'default','housing','loan','contact',
                'month','day_of_week','poutcome',
                'duration','campaign','pdays','previous',
                'emp.var.rate','cons.price.idx',
                'cons.conf.idx']

X = bank_data[feature_cols] # Features
Y = bank_data.y # Target variable

X_train = training_bank_data[feature_cols] # Features
Y_train = training_bank_data.y # Target variable

X_test = testing_bank_data[feature_cols] # Features
Y_test = testing_bank_data.y # Target variable

logistic_model.fit(X_train, Y_train)
```

[78]:
```
LogisticRegression(class_weight='balanced', max_iter=10000)
```

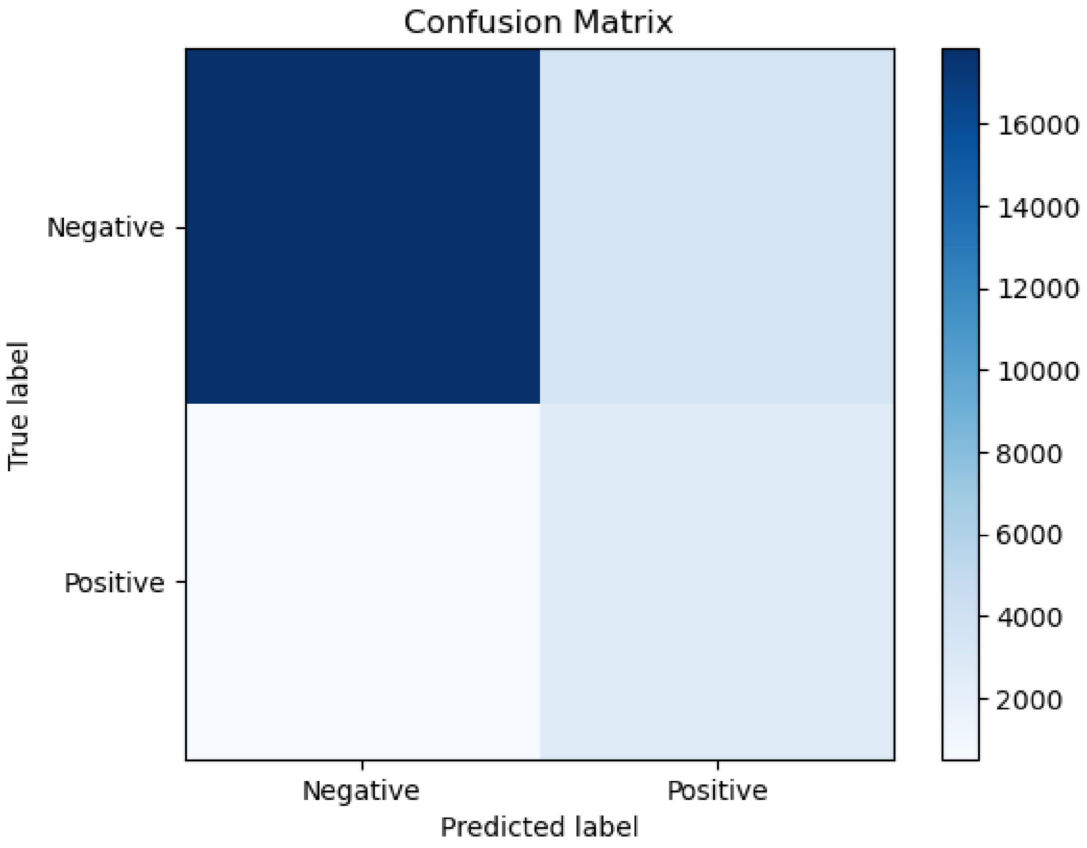
- **How does the model perform? Plot the confusion matrix and the ROC curve of both train and test dataset.**

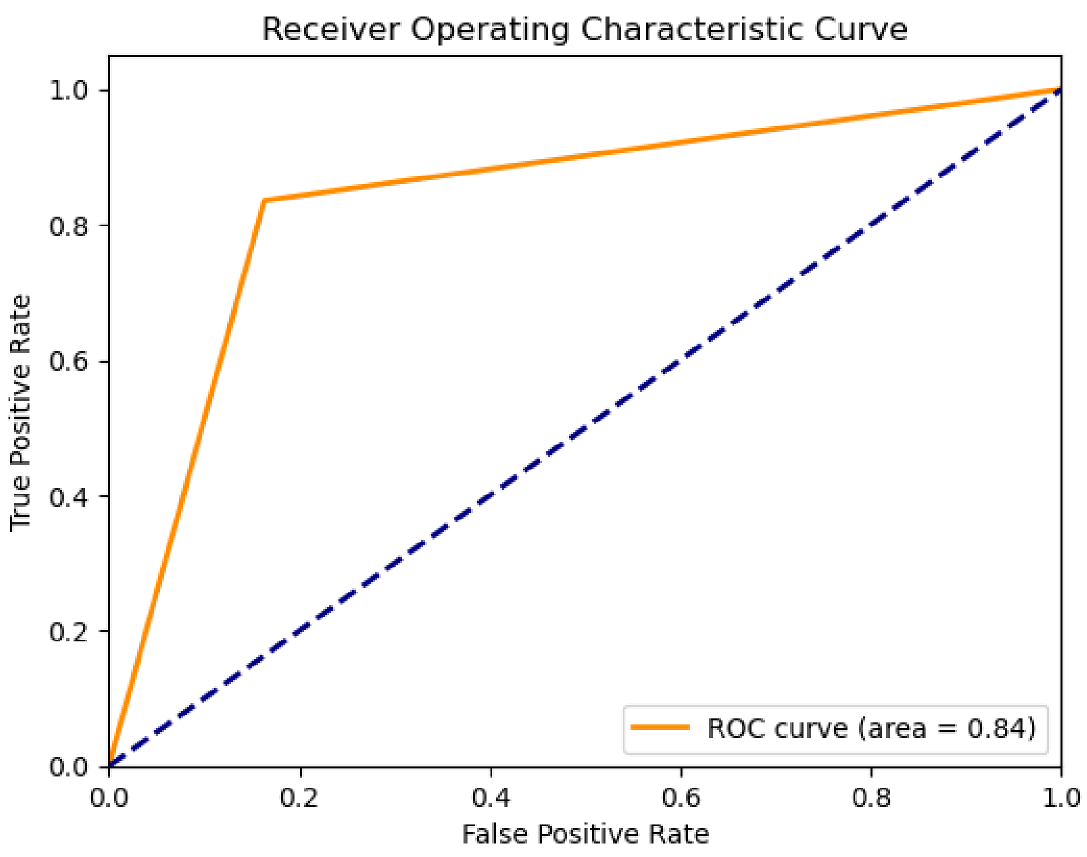
Model Performance (confusion matrix and ROC curve) with training dataset

[79]:
```
from sklearn.metrics import confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
```

```
# Predict the classes for the test data
Y_pred = logistic_model.predict(X_train)

# Compute the confusion matrix
conf_matrix = confusion_matrix(Y_train, Y_pred)
print(conf_matrix)
print("Accuracy score is ", accuracy_score(Y_train, Y_pred))
print("F1 score is ", f1_score(Y_train, Y_pred))
# Plot the confusion matrix
plt.imshow(conf_matrix, cmap=plt.cm.Blues)
plt.colorbar()
plt.xticks([0, 1], ['Negative', 'Positive'])
plt.yticks([0, 1], ['Negative', 'Positive'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()

# Compute the false positive rate, true positive rate and threshold for the ROC␣
 ↪curve
fpr, tpr, thresholds = roc_curve(Y_train, Y_pred)

# Compute the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Plot the ROC curve for the test data
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve')
plt.legend(loc="lower right")
plt.show()
```

```
[[17843  3482]
 [  504  2561]]
Accuracy score is  0.8365723657236572
F1 score is  0.5623627580149319
```


Confusion Matrix
True label
Negative
Positive
Predicted label
Negative
Positive
16000
14000
12000
10000
8000
6000
4000
2000



Overall, as we can see in the output above, the model did perform pretty well with the training data as per the metrics mentioned below.

Specifically, after looking at the output of the confusion matrix it is clear that there are total 17843 True Negatives (TN), 2561 True Positives (TP), 504 False Negatives (FN), and 3482 False Positives (FP).

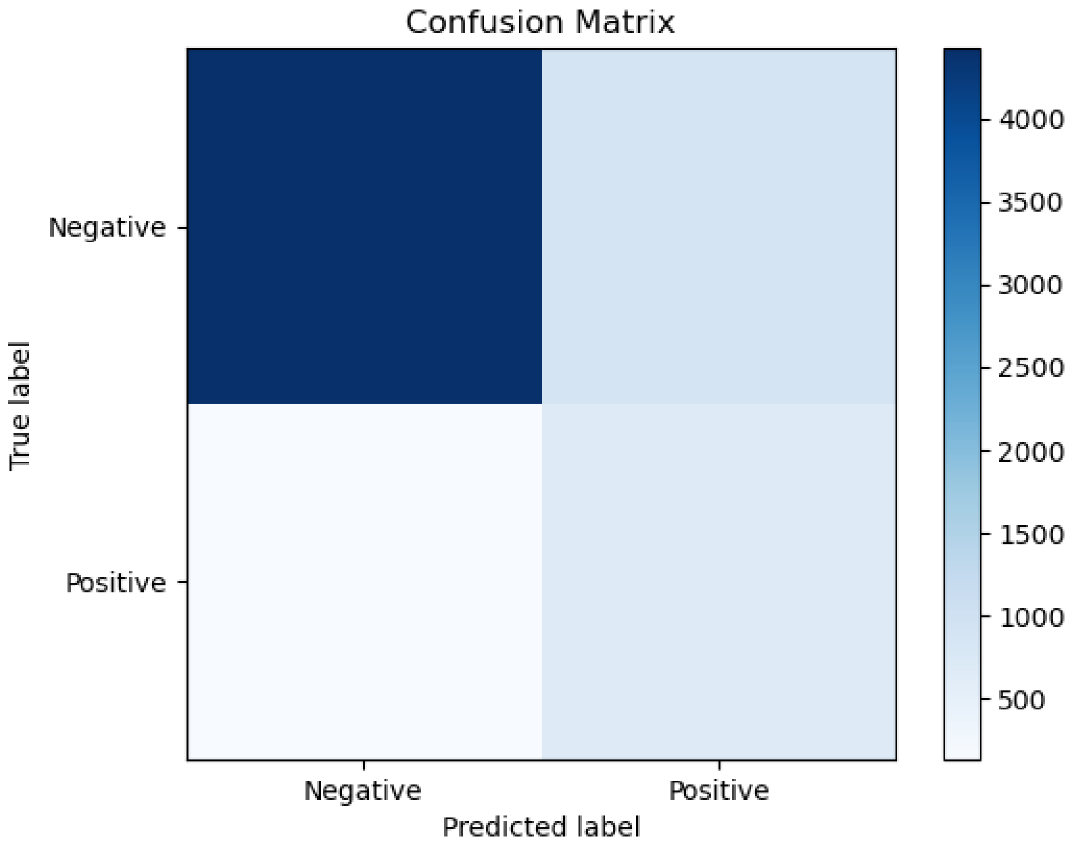
- Accuracy Rate = (TP + TN)/total = 0.837 (rounded to 3 dp)
- Misclassification Rate = 1 - Accuracy Rate = 0.163 (rounded to 3 dp)
- True Positive Rate = TP/actual positive = 0.836 (rounded to 3 dp)
- False Positive Rate = FP/actual negative = 0.163 (rounded to 3 dp)
- True Negative Rate = TN/actual negative = 0.837 (rounded to 3 dp)
- Precision = TP/predicted positive = 0.424 (rounded to 3 dp)
- Prevalence = actual positive/total = 0.126 (rounded to 3 dp)

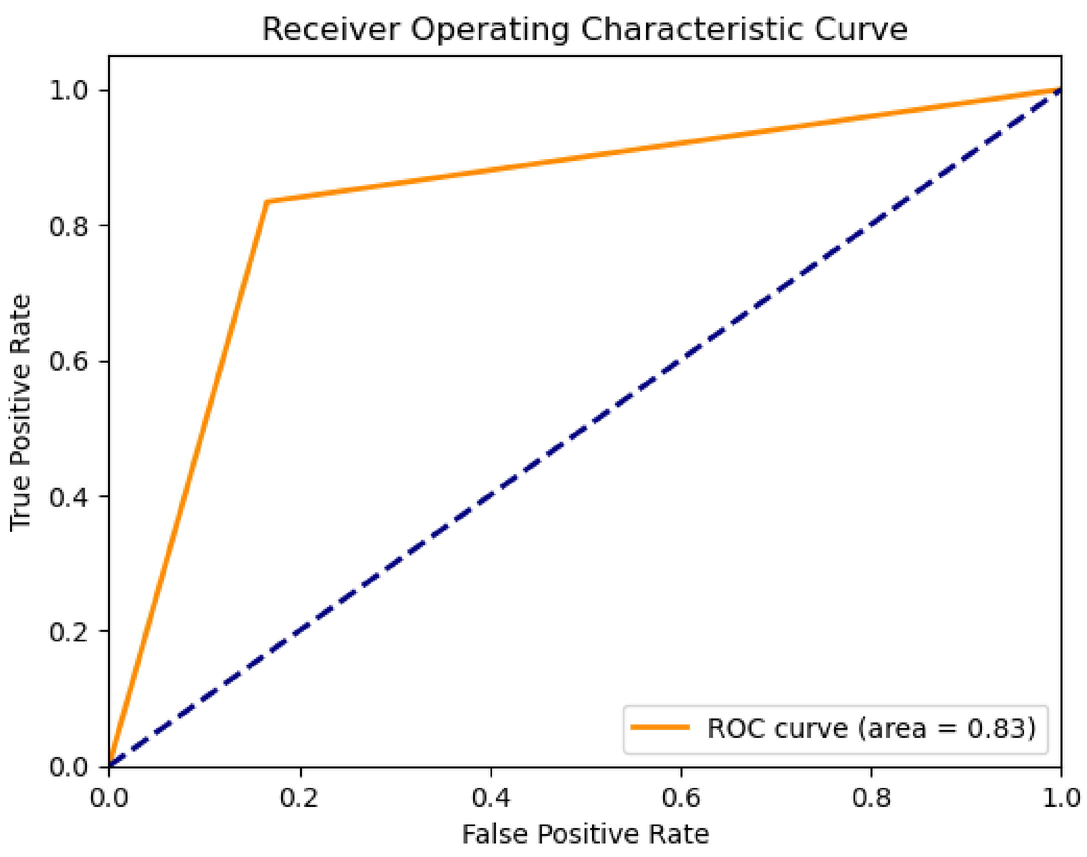
Model Performance (confusion matrix and ROC curve) with testing dataset

[80]:
```
from sklearn.metrics import confusion_matrix, roc_curve, auc, accuracy_score,␣
 ↪f1_score
import matplotlib.pyplot as plt
```

```
# Predict the classes for the test data
Y_pred = logistic_model.predict(X_test)

# Compute the confusion matrix
conf_matrix = confusion_matrix(Y_test, Y_pred)
print(conf_matrix)
print("Accuracy score is ", accuracy_score(Y_test, Y_pred))
print("F1 score is ", f1_score(Y_test, Y_pred))
# Plot the confusion matrix
plt.imshow(conf_matrix, cmap=plt.cm.Blues)
plt.colorbar()
plt.xticks([0, 1], ['Negative', 'Positive'])
plt.yticks([0, 1], ['Negative', 'Positive'])
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix')
plt.show()

# Compute the false positive rate, true positive rate and threshold for the ROC␣
 ↪curve
fpr, tpr, thresholds = roc_curve(Y_test, Y_pred)

# Compute the area under the ROC curve
roc_auc = auc(fpr, tpr)

# Plot the ROC curve for the test data
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve')
plt.legend(loc="lower right")
plt.show()
```

```
[[4423  881]
 [ 132  662]]
Accuracy score is  0.8338799606428338
F1 score is  0.566538296961917
```


Confusion Matrix
True label
Negative
Positive
Predicted label
Negative
Positive
4000
3500
3000
2500
2000
1500
1000
500



Overall, as we can see in the output above, the model did perform pretty well with the testing data as per the metrics mentioned below.

Specifically, after looking at the output of the confusion matrix it is clear that there are total 4423 True Negatives (TN), 662 True Positives (TP), 132 False Negatives (FN), and 881 False Positives (FP).

- Accuracy Rate = (TP + TN)/total = 0.834 (rounded to 3 dp)
- Misclassification Rate = 1 - Accuracy Rate = 0.166 (rounded to 3 dp)
- True Positive Rate = TP/actual positive = 0.834 (rounded to 3 dp)
- False Positive Rate = FP/actual negative = 0.166 (rounded to 3 dp)
- True Negative Rate = TN/actual negative = 0.834 (rounded to 3 dp)
- Precision = TP/predicted positive = 0.429 (rounded to 3 dp)
- Prevalence = actual positive/total = 0.150 (rounded to 3 dp)

- **Print the classification report of the model.**

[81]:
```
from sklearn.metrics import classification_report

logistic_model_classification_report = classification_report(Y_test, Y_pred)

print(logistic_model_classification_report)
```

```
              precision    recall  f1-score   support

           0       0.97      0.83      0.90      5304
           1       0.43      0.83      0.57       794

    accuracy                           0.83      6098
   macro avg       0.70      0.83      0.73      6098
weighted avg       0.90      0.83      0.85      6098
```

- **What do you find in the report? And what is the reason behind that? What metric would you maximize in search of the best-performing model?** The classification report provides a summary of the performance of a binary classifier. The precision, recall, and F1-score are the three main metrics that are used to evaluate the performance of a classifier.

In this report, precision is the ratio of correctly predicted positive instances to the total predicted positive instances. Recall is the ratio of correctly predicted positive instances to the total actual positive instances. F1-score is the harmonic mean of precision and recall, and it is a good measure to use when the positive class is rare or when the classes are imbalanced.

In this report, the classifier has a precision of 0.97 for class 0 and 0.43 for class 1. This means that the classifier is able to correctly identify 97% of the instances it predicts as class 0, and 43% of the instances it predicts as class 1.

The recall for class 0 is 0.83 and for class 1 is 0.83, which means that the classifier is able to correctly identify 83% of the actual instances of class 0, and 83% of the actual instances of class 1.

The F1-score for class 0 is 0.90 and for class 1 is 0.57, which is a weighted average of precision and recall.

The accuracy is 0.83, which is the ratio of correctly classified instances to the total instances.

The weighted average takes into account the imbalance between the two classes. In this case, class 0 is much more prevalent than class 1, so the weighted average F1-score is 0.85, which is higher than the F1-score for class 1.

In search of the best-performing model, the goal is to maximize the F1-score. This is because the F1-score is a balance between precision and recall, and it is a good metric to use when the positive class is rare or when the classes are imbalanced. However, depending on the problem maximizing other metrics such as precision, recall, or accuracy may also be appropriate.

4 4. Logistic Regression

- **Explore attributes of your model. Print out the coefficients, intercept of the model, as well as probability estimates of test dataset.**

[82]:
```python
# Print the coefficients
print("Coefficients:", logistic_model.coef_)
```

```
Coefficients: [[ 0.00930807  0.01364383  0.1354264   0.09686469 -0.04105604
0.05736587
```

```
 -0.23808015 -0.37449616 -0.02606264 -0.01260072  0.45454824  0.00622255
 -0.04590558 -0.0016117   0.0507441  -0.74237643 -0.011165    0.03056868]]
```

[83]:
```
# Print the intercept
print("Intercept:", logistic_model.intercept_)
```

```
Intercept: [-0.04063569]
```

[84]:
```
# Predict the probabilities of the test data
Y_probs = logistic_model.predict_proba(X_test)

# Print the probability estimates
print("Probability Estimates:", Y_probs)
```

```
Probability Estimates: [[0.95647064 0.04352936]
 [0.87202819 0.12797181]
 [0.75226604 0.24773396]
 …
 [0.77355391 0.22644609]
 [0.34965445 0.65034555]
 [0.70186639 0.29813361]]
```

From the outputs above we can explore the attributes of our model.

Use linear combination and sigmoid function to calculate probability estimates of test dataset manually and check if the two match.

- intercept: a0

- coefficients: a1, a2, · · · , am

- input: x_1, x_2, **· · ·** , x_n**, where** $x_i = (x_{i1}, ..., x_{im}), i = 1, ..., n$

- probability estimates:

$$y_i = \frac{1}{1 + e^{-(a_0 + \Sigma_{k=1}^{m} a_k x_{ik})}}$$

What is the relationship between logistic regression and linear regression?

[85]:
```
# Get the coefficients and intercept
coef = logistic_model.coef_[0]
intercept = logistic_model.intercept_[0]

# Calculate the linear combination
linear_combination = np.dot(X_test, coef) + intercept

# Calculate the probability estimates using the sigmoid function
y_probs_manual = 1 / (1 + np.exp(-linear_combination))
```

```
# Print out the manually calculated y_probs
print(y_probs_manual)
```

```
[0.04352936 0.12797181 0.24773396 … 0.22644609 0.65034555 0.29813361]
```

From the outputs above, we can see that they do match!!

The relationship between logistic regression and linear regression is that logistic regression is a special case of linear regression where the dependent variable is binary and the predicted values are transformed using the logistic (or sigmoid) function to lie between 0 and 1.

In linear regression, the goal is to predict a continuous dependent variable using a linear combination of the independent variables. In logistic regression, the goal is to predict a binary dependent variable (i.e., class labels) using a linear combination of the independent variables.

The probability estimates of logistic regression are calculated using a linear combination of the input variables and the coefficients, followed by transforming the result using the sigmoid function to ensure that the predicted values lie between 0 and 1.

Therefore, logistic regression can be seen as a variation of linear regression that is specifically designed for classification problems. The main difference is the use of the sigmoid function to ensure that the predicted values are probabilities.

5 5. Bonus

- **What is overfitting? What is underfitting? Does the model you trained exhibit any of these problem?** Overfitting is a phenomenon in machine learning where a model performs well on the training data but poorly on unseen or new data. This happens when a model is too complex or has too many parameters and is able to capture the noise or random fluctuations in the training data. This makes the model fit the training data too closely, leading to poor generalization performance on new data.

Underfitting is the opposite of overfitting and occurs when a model is too simple and unable to capture the underlying pattern in the data. This leads to high training error and poor generalization performance.

Whether the model exhibits overfitting or underfitting can be determined by comparing the performance of the model on the training data and test data. If the model performs well on the training data but poorly on the test data, it may be overfitting. If the model performs poorly on both the training data and the test data, it may be underfitting.

I don't think our model exhibits overfitting, but it may lean a little towards exhibiting slight underfitting because the accuracy rate is only around 85%

- **Use the sklearn documentation to perform a simple cross validation and grid search to optimize the parameters to your model. What is the lift you get from performing these additional steps? (Hint: from sklearn.model selection import GridSearchCV, cross val score)**

[92]:
```python
from sklearn.model_selection import GridSearchCV, KFold, cross_val_score

# Define the parameter grid to search
param_grid = {'C': [0.1, 1, 10, 100], 'penalty': ['l2']} # l1 lasso l2 ridge

# Create a 5-fold cross-validation object
kfold = KFold(n_splits = 5, shuffle = True, random_state = 42)

scores = cross_val_score(logistic_model, X, Y, cv = kfold)
print("Cross-validation scores:", scores)
print("Mean cross-validation score:", np.mean(scores))

# Create a grid search object
grid_search = GridSearchCV(logistic_model, param_grid, cv = kfold,
                           scoring = 'accuracy', return_train_score = True)

# Fit the grid search object to the data
grid_search.fit(X[feature_cols], Y)

# Print the best parameters and best score
print('Best parameters:', grid_search.best_params_)
print('Best score:', grid_search.best_score_)
```

```
Cross-validation scores: [0.84355526 0.83371597 0.83978354 0.84336559
0.81941939]
Mean cross-validation score: 0.8359679496548449
Best parameters: {'C': 10, 'penalty': 'l2'}
Best score: 0.83596796041346
```

The GridSearchCV function is initialized with the logistic regression model, the parameter grid, and the number of folds for cross-validation (5 in this case). The grid search is then fit to the data and the best parameters and best score are printed.

Performing cross-validation and grid search can help to lift the performance of the model by choosing the best parameters for the model and by estimating the performance of the model more accurately by considering multiple folds of the data. The lift in performance would depend on the specifics of the problem and the data and would need to be evaluated on a case-by-case basis.

In this specific case, we can see that our best score is still close to 85% so we can conclude that the lift of around 2% we got was not that significant at the cost of additional steps we performed.