Problem_1

October 9, 2022

[13]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns
```

[14]:
```
dataset = pd.read_csv('./diabetes.csv')
dataset.head()
```

[14]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[15]:
```
diabetes = dataset.values[:,:]
print(diabetes[:20,:])
```

```
[[6.000e+00 1.480e+02 7.200e+01 3.500e+01 0.000e+00 3.360e+01 6.270e-01
  5.000e+01 1.000e+00]
 [1.000e+00 8.500e+01 6.600e+01 2.900e+01 0.000e+00 2.660e+01 3.510e-01
  3.100e+01 0.000e+00]
 [8.000e+00 1.830e+02 6.400e+01 0.000e+00 0.000e+00 2.330e+01 6.720e-01
  3.200e+01 1.000e+00]
```

```
 [1.000e+00 8.900e+01 6.600e+01 2.300e+01 9.400e+01 2.810e+01 1.670e-01
  2.100e+01 0.000e+00]
 [0.000e+00 1.370e+02 4.000e+01 3.500e+01 1.680e+02 4.310e+01 2.288e+00
  3.300e+01 1.000e+00]
 [5.000e+00 1.160e+02 7.400e+01 0.000e+00 0.000e+00 2.560e+01 2.010e-01
  3.000e+01 0.000e+00]
 [3.000e+00 7.800e+01 5.000e+01 3.200e+01 8.800e+01 3.100e+01 2.480e-01
  2.600e+01 1.000e+00]
 [1.000e+01 1.150e+02 0.000e+00 0.000e+00 0.000e+00 3.530e+01 1.340e-01
  2.900e+01 0.000e+00]
 [2.000e+00 1.970e+02 7.000e+01 4.500e+01 5.430e+02 3.050e+01 1.580e-01
  5.300e+01 1.000e+00]
 [8.000e+00 1.250e+02 9.600e+01 0.000e+00 0.000e+00 0.000e+00 2.320e-01
  5.400e+01 1.000e+00]
 [4.000e+00 1.100e+02 9.200e+01 0.000e+00 0.000e+00 3.760e+01 1.910e-01
  3.000e+01 0.000e+00]
 [1.000e+01 1.680e+02 7.400e+01 0.000e+00 0.000e+00 3.800e+01 5.370e-01
  3.400e+01 1.000e+00]
 [1.000e+01 1.390e+02 8.000e+01 0.000e+00 0.000e+00 2.710e+01 1.441e+00
  5.700e+01 0.000e+00]
 [1.000e+00 1.890e+02 6.000e+01 2.300e+01 8.460e+02 3.010e+01 3.980e-01
  5.900e+01 1.000e+00]
 [5.000e+00 1.660e+02 7.200e+01 1.900e+01 1.750e+02 2.580e+01 5.870e-01
  5.100e+01 1.000e+00]
 [7.000e+00 1.000e+02 0.000e+00 0.000e+00 0.000e+00 3.000e+01 4.840e-01
  3.200e+01 1.000e+00]
 [0.000e+00 1.180e+02 8.400e+01 4.700e+01 2.300e+02 4.580e+01 5.510e-01
  3.100e+01 1.000e+00]
 [7.000e+00 1.070e+02 7.400e+01 0.000e+00 0.000e+00 2.960e+01 2.540e-01
  3.100e+01 1.000e+00]
 [1.000e+00 1.030e+02 3.000e+01 3.800e+01 8.300e+01 4.330e+01 1.830e-01
  3.300e+01 0.000e+00]
 [1.000e+00 1.150e+02 7.000e+01 3.000e+01 9.600e+01 3.460e+01 5.290e-01
  3.200e+01 1.000e+00]]
```

[16]:
```
# Set the independent variables from Pregnancies to Age. Dependent variable is
 ↪outcome.
X = diabetes[:,0:8]
Y = diabetes[:,8]
```

[17]:
```
print('X=', X[0:5])
```

```
X= [[6.000e+00 1.480e+02 7.200e+01 3.500e+01 0.000e+00 3.360e+01 6.270e-01
  5.000e+01]
 [1.000e+00 8.500e+01 6.600e+01 2.900e+01 0.000e+00 2.660e+01 3.510e-01
  3.100e+01]
 [8.000e+00 1.830e+02 6.400e+01 0.000e+00 0.000e+00 2.330e+01 6.720e-01
  3.200e+01]
```

```
[1.000e+00 8.900e+01 6.600e+01 2.300e+01 9.400e+01 2.810e+01 1.670e-01
  2.100e+01]
 [0.000e+00 1.370e+02 4.000e+01 3.500e+01 1.680e+02 4.310e+01 2.288e+00
  3.300e+01]]
```

[18]:
```
print('Y=', Y[0:5])
```

```
Y= [1. 0. 1. 0. 1.]
```

[19]:
```
# Splitting the datasets to training and validation sets.

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state = 0)
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(614, 8) (154, 8) (614,) (154,)
```

[20]:
```
# Feature scaling between 0 and 1 for independent variables using␣
 ↪Standardization.

from sklearn.preprocessing import StandardScaler, MinMaxScaler
sc_X = StandardScaler()
X_train_sc = sc_X.fit_transform(X_train)
X_test_sc = sc_X.fit_transform(X_test)
```

[21]:
```
print('New X_train =', X_train_sc[0:5])
```

```
New X_train = [[ 0.90832902  0.91569367  0.44912368  0.52222619  0.3736349
0.37852648
   0.67740401  1.69955804]
 [ 0.03644676 -0.75182191 -0.47230103  0.14814855 -0.69965674 -0.50667229
  -0.07049698 -0.96569189]
 [-1.12606292  1.38763205  1.06340683  0.77161128  5.09271083  2.54094063
  -0.11855487 -0.88240283]
 [-0.8354355  -0.37427121 -0.67706208  0.02345601  0.45029859 -0.88604319
   1.10091422 -0.88240283]
 [ 1.19895644 -0.02818307 -3.54371676 -1.28581572 -0.69965674 -0.27904975
  -0.85143778  0.36693308]]
```

[22]:
```
# Construct a confusion matrix
from sklearn.model_selection import train_test_split
test_size = 0.2
seed = 0
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= test_size,␣
 ↪random_state = seed)
model = LogisticRegression(solver = 'liblinear')
model.fit(X_train_sc, Y_train)
predicted = model.predict(X_test_sc)
matrix = confusion_matrix(Y_test, predicted)
```

```
print(matrix)
```

```
[[94 13]
 [18 29]]
```

[23]:
```
from sklearn.model_selection import train_test_split
test_size = 0.2
seed = 0
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= test_size,␣
 ↪random_state = seed)
model = LogisticRegression(solver = 'liblinear')
model.fit(X_train_sc, Y_train)
predicted = model.predict(X_test_sc)
report = classification_report(Y_test, predicted)
print(report)
```

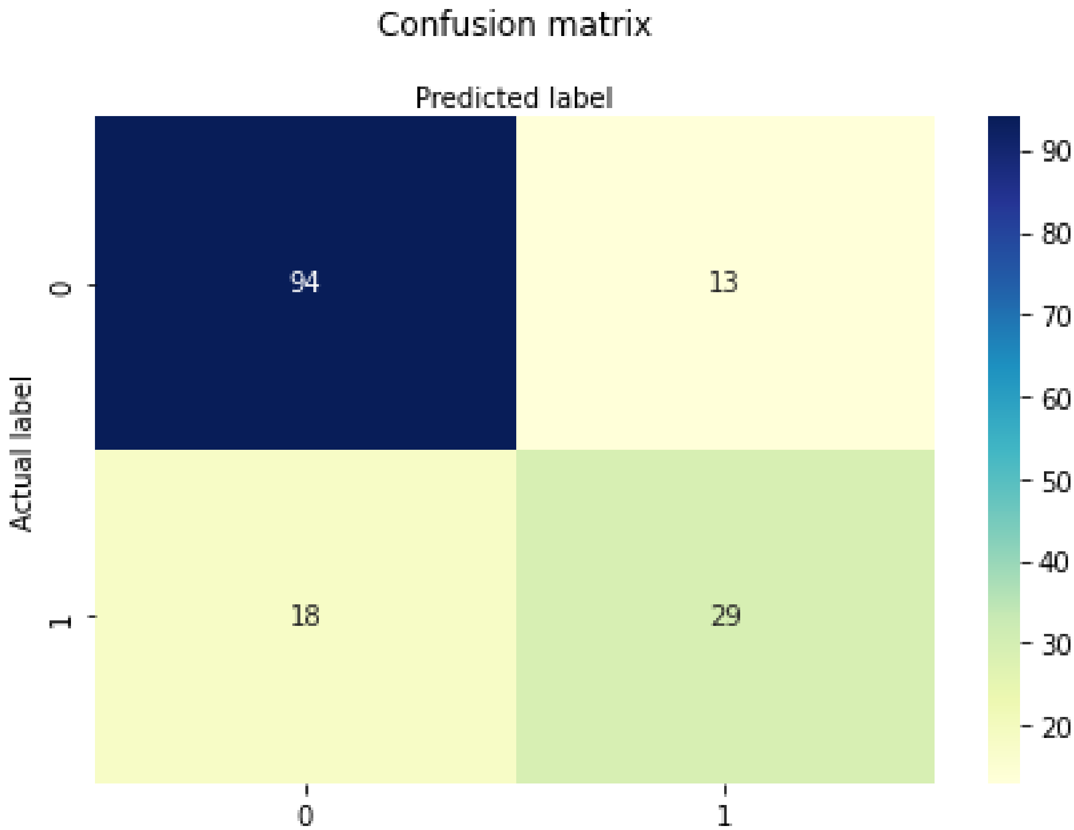
```
              precision    recall  f1-score   support

         0.0       0.84      0.88      0.86       107
         1.0       0.69      0.62      0.65        47

    accuracy                           0.80       154
   macro avg       0.76      0.75      0.76       154
weighted avg       0.79      0.80      0.80       154
```

[24]:
```
# Visualize the confusion matrix using Heatmap
import seaborn as sns
from matplotlib.colors import ListedColormap
class_names=[0,1] # name of classes
fid, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu", fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y = 1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[24]:
```
Text(0.5, 257.44, 'Predicted label')
```


Confusion matrix
Predicted label
Actual label
0
1
94
13
18
29
0
1
90
80
70
60
50
40
30
20

[ ]:

Problem_2

October 9, 2022

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns
```

[2]:
```
dataset = pd.read_csv('./diabetes.csv')
dataset.head()
```

[2]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[3]:
```
diabetes = dataset.values[:,:]
print(diabetes[:20,:])
```

```
[[6.000e+00 1.480e+02 7.200e+01 3.500e+01 0.000e+00 3.360e+01 6.270e-01
  5.000e+01 1.000e+00]
 [1.000e+00 8.500e+01 6.600e+01 2.900e+01 0.000e+00 2.660e+01 3.510e-01
  3.100e+01 0.000e+00]
 [8.000e+00 1.830e+02 6.400e+01 0.000e+00 0.000e+00 2.330e+01 6.720e-01
  3.200e+01 1.000e+00]
```

```
 [1.000e+00 8.900e+01 6.600e+01 2.300e+01 9.400e+01 2.810e+01 1.670e-01
  2.100e+01 0.000e+00]
 [0.000e+00 1.370e+02 4.000e+01 3.500e+01 1.680e+02 4.310e+01 2.288e+00
  3.300e+01 1.000e+00]
 [5.000e+00 1.160e+02 7.400e+01 0.000e+00 0.000e+00 2.560e+01 2.010e-01
  3.000e+01 0.000e+00]
 [3.000e+00 7.800e+01 5.000e+01 3.200e+01 8.800e+01 3.100e+01 2.480e-01
  2.600e+01 1.000e+00]
 [1.000e+01 1.150e+02 0.000e+00 0.000e+00 0.000e+00 3.530e+01 1.340e-01
  2.900e+01 0.000e+00]
 [2.000e+00 1.970e+02 7.000e+01 4.500e+01 5.430e+02 3.050e+01 1.580e-01
  5.300e+01 1.000e+00]
 [8.000e+00 1.250e+02 9.600e+01 0.000e+00 0.000e+00 0.000e+00 2.320e-01
  5.400e+01 1.000e+00]
 [4.000e+00 1.100e+02 9.200e+01 0.000e+00 0.000e+00 3.760e+01 1.910e-01
  3.000e+01 0.000e+00]
 [1.000e+01 1.680e+02 7.400e+01 0.000e+00 0.000e+00 3.800e+01 5.370e-01
  3.400e+01 1.000e+00]
 [1.000e+01 1.390e+02 8.000e+01 0.000e+00 0.000e+00 2.710e+01 1.441e+00
  5.700e+01 0.000e+00]
 [1.000e+00 1.890e+02 6.000e+01 2.300e+01 8.460e+02 3.010e+01 3.980e-01
  5.900e+01 1.000e+00]
 [5.000e+00 1.660e+02 7.200e+01 1.900e+01 1.750e+02 2.580e+01 5.870e-01
  5.100e+01 1.000e+00]
 [7.000e+00 1.000e+02 0.000e+00 0.000e+00 0.000e+00 3.000e+01 4.840e-01
  3.200e+01 1.000e+00]
 [0.000e+00 1.180e+02 8.400e+01 4.700e+01 2.300e+02 4.580e+01 5.510e-01
  3.100e+01 1.000e+00]
 [7.000e+00 1.070e+02 7.400e+01 0.000e+00 0.000e+00 2.960e+01 2.540e-01
  3.100e+01 1.000e+00]
 [1.000e+00 1.030e+02 3.000e+01 3.800e+01 8.300e+01 4.330e+01 1.830e-01
  3.300e+01 0.000e+00]
 [1.000e+00 1.150e+02 7.000e+01 3.000e+01 9.600e+01 3.460e+01 5.290e-01
  3.200e+01 1.000e+00]]
```

[4]:
```
# Set the independent variables from Pregnancies to Age. Dependent variable is␣
↪outcome.
X = diabetes[:,0:8]
Y = diabetes[:,8]
```

[5]:
```
print('X=', X[0:5])
```

```
X= [[6.000e+00 1.480e+02 7.200e+01 3.500e+01 0.000e+00 3.360e+01 6.270e-01
  5.000e+01]
 [1.000e+00 8.500e+01 6.600e+01 2.900e+01 0.000e+00 2.660e+01 3.510e-01
  3.100e+01]
 [8.000e+00 1.830e+02 6.400e+01 0.000e+00 0.000e+00 2.330e+01 6.720e-01
  3.200e+01]
```

```
[1.000e+00 8.900e+01 6.600e+01 2.300e+01 9.400e+01 2.810e+01 1.670e-01
 2.100e+01]
[0.000e+00 1.370e+02 4.000e+01 3.500e+01 1.680e+02 4.310e+01 2.288e+00
 3.300e+01]]
```

[6]:
```
print('Y=', Y[0:5])
```

```
Y= [1. 0. 1. 0. 1.]
```

[7]:
```
# Splitting the datasets to training and validation sets.

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state = 0)
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(614, 8) (154, 8) (614,) (154,)
```

[8]:
```
# Feature scaling between 0 and 1 for independent variables using␣
 ↪Standardization.

from sklearn.preprocessing import StandardScaler, MinMaxScaler
sc_X = StandardScaler()
X_train_sc = sc_X.fit_transform(X_train)
X_test_sc = sc_X.fit_transform(X_test)
```

[9]:
```
print('New X_train =', X_train_sc[0:5])
```

```
New X_train = [[ 0.90832902  0.91569367  0.44912368  0.52222619  0.3736349
0.37852648
   0.67740401  1.69955804]
 [ 0.03644676 -0.75182191 -0.47230103  0.14814855 -0.69965674 -0.50667229
  -0.07049698 -0.96569189]
 [-1.12606292  1.38763205  1.06340683  0.77161128  5.09271083  2.54094063
  -0.11855487 -0.88240283]
 [-0.8354355  -0.37427121 -0.67706208  0.02345601  0.45029859 -0.88604319
   1.10091422 -0.88240283]
 [ 1.19895644 -0.02818307 -3.54371676 -1.28581572 -0.69965674 -0.27904975
  -0.85143778  0.36693308]]
```

[10]:
```
# K-fold: 5
kfold = KFold(n_splits = 5, random_state = 0, shuffle = True)
model = LogisticRegression(solver = 'liblinear')
results = cross_val_score(model, X, Y, cv=kfold)
# Output the accuracy. Calculate the mean and std across all folds.
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 76.555% (3.185%)
```

[11]:
```python
# K-fold: 10
kfold = KFold(n_splits = 10, random_state = 0, shuffle = True)
model = LogisticRegression(solver = 'liblinear')
results = cross_val_score(model, X, Y, cv=kfold)
# Output the accuracy. Calculate the mean and std across all folds.
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 76.818% (3.744%)
```

[12]:
```python
# Construct a confusion matrix
from sklearn.model_selection import train_test_split
test_size = 0.2
seed = 0
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= test_size,␣
 ↪random_state = seed)
model = LogisticRegression(solver = 'liblinear')
model.fit(X_train_sc, Y_train)
predicted = model.predict(X_test_sc)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

```
[[94 13]
 [18 29]]
```

[13]:
```python
from sklearn.model_selection import train_test_split
test_size = 0.2
seed = 0
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= test_size,␣
 ↪random_state = seed)
model = LogisticRegression(solver = 'liblinear')
model.fit(X_train_sc, Y_train)
predicted = model.predict(X_test_sc)
report = classification_report(Y_test, predicted)
print(report)
```

```
              precision    recall  f1-score   support

         0.0       0.84      0.88      0.86       107
         1.0       0.69      0.62      0.65        47

    accuracy                           0.80       154
   macro avg       0.76      0.75      0.76       154
weighted avg       0.79      0.80      0.80       154
```

[ ]:
```python

```

Problem_3

October 9, 2022

[50]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns
from sklearn.datasets import load_breast_cancer
```

[51]:
```
breast = load_breast_cancer()
breast_data = breast.data
breast_data.shape
```

[51]: (569, 30)

[52]:
```
breast_input = pd.DataFrame(breast_data)
breast_input.head()
```

[52]:
```
       0      1       2       3        4        5       6        7       8  \
0  17.99  10.38  122.80  1001.0  0.11840  0.27760  0.3001  0.14710  0.2419
1  20.57  17.77  132.90  1326.0  0.08474  0.07864  0.0869  0.07017  0.1812
2  19.69  21.25  130.00  1203.0  0.10960  0.15990  0.1974  0.12790  0.2069
3  11.42  20.38   77.58   386.1  0.14250  0.28390  0.2414  0.10520  0.2597
4  20.29  14.34  135.10  1297.0  0.10030  0.13280  0.1980  0.10430  0.1809

         9  …     20     21      22      23      24      25      26      27  \
0  0.07871  …  25.38  17.33  184.60  2019.0  0.1622  0.6656  0.7119  0.2654
1  0.05667  …  24.99  23.41  158.80  1956.0  0.1238  0.1866  0.2416  0.1860
2  0.05999  …  23.57  25.53  152.50  1709.0  0.1444  0.4245  0.4504  0.2430
3  0.09744  …  14.91  26.50   98.87   567.7  0.2098  0.8663  0.6869  0.2575
4  0.05883  …  22.54  16.67  152.20  1575.0  0.1374  0.2050  0.4000  0.1625

       28       29
0  0.4601  0.11890
```

```
1  0.2750  0.08902
2  0.3613  0.08758
3  0.6638  0.17300
4  0.2364  0.07678

[5 rows x 30 columns]
```

[53]:
```python
breast_labels = breast.target
breast_labels.shape
```

[53]:
```
(569,)
```

[54]:
```python
labels = np.reshape(breast_labels,(569,1))
```

[55]:
```python
final_breast_data = np.concatenate([breast_data,labels],axis=1)
final_breast_data.shape
```

[55]:
```
(569, 31)
```

[56]:
```python
breast_dataset = pd.DataFrame(final_breast_data)
features = breast.feature_names
features
```

[56]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

[57]:
```python
features_labels = np.append(features,'label')
breast_dataset.columns = features_labels
breast_dataset.head()
```

[57]:
```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
```

```
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                 0.07871  …          17.33           184.60      2019.0
1                 0.05667  …          23.41           158.80      1956.0
2                 0.05999  …          25.53           152.50      1709.0
3                 0.09744  …          26.50            98.87       567.7
4                 0.05883  …          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  label
0          0.4601                  0.11890    0.0
1          0.2750                  0.08902    0.0
2          0.3613                  0.08758    0.0
3          0.6638                  0.17300    0.0
4          0.2364                  0.07678    0.0

[5 rows x 31 columns]
```

[58]:
```
breast_dataset.shape
```

[58]:
```
(569, 31)
```

[59]:
```
X = breast_dataset.values[:,0:30]
print('X =', X[0:5])
```

```
X = [[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
```

```
 3.613e-01 8.758e-02]
[1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
 1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
 9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
 2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
 6.638e-01 1.730e-01]
[2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.980e-01
 1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.444e+01
 1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.254e+01
 1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.625e-01
 2.364e-01 7.678e-02]]
```

[60]:

```
Y = breast_dataset.values[:,30]
print('Y =', Y[0:5])
```

```
Y = [0. 0. 0. 0. 0.]
```

[61]:

```
# Splitting the datasets to training and validation sets.

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state = 0)
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(455, 30) (114, 30) (455,) (114,)
```

[62]:

```
# Feature scaling between 0 and 1 for independent variables using␣
 ↪Standardization.

from sklearn.preprocessing import StandardScaler, MinMaxScaler
sc_X = MinMaxScaler()
X_train_sc = sc_X.fit_transform(X_train)
X_test_sc = sc_X.fit_transform(X_test)
```

[63]:

```
print('New X_train =', X_train_sc[0:5])
```

```
New X_train = [[0.1452506  0.32448133 0.14249188 0.07096501 0.52210275
0.18450791
  0.05883318 0.08822068 0.41919192 0.28117102 0.05446315 0.36571782
  0.04810818 0.01798599 0.28172272 0.09191276 0.04267677 0.1523584
  0.2448393  0.06506087 0.11490573 0.39498934 0.10742567 0.04885961
  0.45585419 0.10954585 0.08426518 0.22387186 0.26197516 0.14167651]
 [0.18074684 0.50912863 0.17275931 0.09179215 0.38427284 0.13029929
  0.08467666 0.06978131 0.48282828 0.20661331 0.07104834 0.27864215
  0.06987702 0.024248   0.22343526 0.14441073 0.0755303  0.19795416
  0.19705849 0.06202065 0.17182497 0.53358209 0.16574531 0.07478864
  0.39047745 0.13806987 0.15391374 0.25783672 0.27597083 0.14154532]
 [0.43348005 0.21369295 0.41814664 0.27847296 0.45965027 0.22474488
  0.12886598 0.2250497  0.34090909 0.18513058 0.04606192 0.06121818
  0.04579937 0.02729489 0.10385345 0.07666657 0.04623737 0.16569426
```

```
  0.13285304 0.02508879 0.34791889 0.20149254 0.32616166 0.18745085
  0.32642145 0.14059241 0.18450479 0.38890803 0.23910901 0.09891119]
 [0.24605992 0.3373444  0.23495266 0.1304772  0.56337569 0.17529621
  0.05834114 0.14617296 0.42424242 0.34519798 0.09219627 0.25433168
  0.0778872  0.03217365 0.17208678 0.07163457 0.02856061 0.28774389
  0.26723655 0.08682614 0.17431519 0.23720682 0.1580258  0.07618954
  0.28283695 0.06431489 0.03977636 0.20261798 0.13049478 0.1227863 ]
 [0.2493729  0.52821577 0.23764771 0.13700954 0.31812751 0.11170468
  0.04015933 0.06267396 0.24444444 0.2064027  0.04070252 0.1721181
  0.03863733 0.0202104  0.1564972  0.07358729 0.02376263 0.08620951
  0.15301056 0.05196717 0.22198506 0.53224947 0.21081727 0.10757471
  0.35944001 0.14854809 0.09824281 0.21822253 0.3025823  0.17703004]]
```

[64]:
```python
# Construct a confusion matrix
from sklearn.model_selection import train_test_split
test_size = 0.2
seed = 0
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= test_size,␣
 ↪random_state = seed)
model = LogisticRegression(solver = 'liblinear')
model.fit(X_train_sc, Y_train)
predicted = model.predict(X_test_sc)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

```
[[47  0]
 [ 9 58]]
```

[65]:
```python
from sklearn.model_selection import train_test_split
test_size = 0.2
seed = 0
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= test_size,␣
 ↪random_state = seed)
model = LogisticRegression(solver = 'liblinear')
model.fit(X_train_sc, Y_train)
predicted = model.predict(X_test_sc)
report = classification_report(Y_test, predicted)
print(report)
```

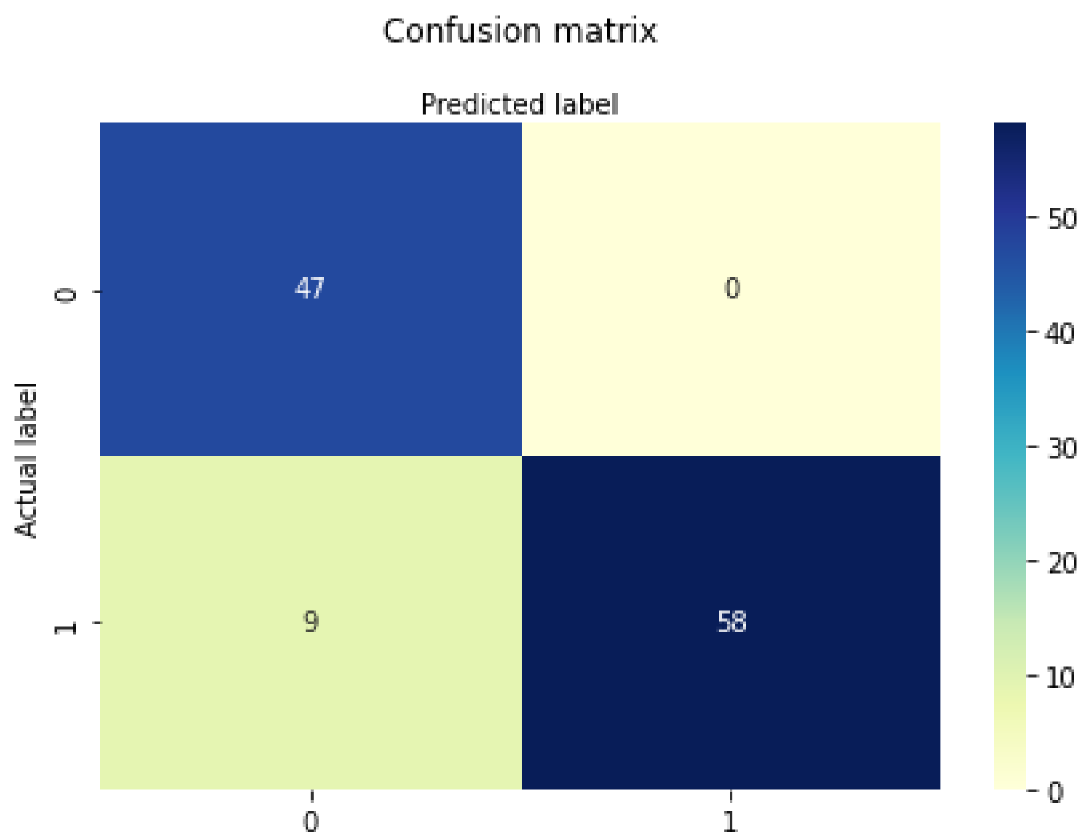
```
              precision    recall  f1-score   support

         0.0       0.84      1.00      0.91        47
         1.0       1.00      0.87      0.93        67

    accuracy                           0.92       114
   macro avg       0.92      0.93      0.92       114
weighted avg       0.93      0.92      0.92       114
```

[66]:
```python
# Visualize the confusion matrix using Heatmap
import seaborn as sns
from matplotlib.colors import ListedColormap
class_names=[0,1] # name of classes
fid, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="YlGnBu", fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y = 1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[66]: Text(0.5, 257.44, 'Predicted label')



[67]:
```python
C = [100, 50, 30, 10, 1]

for c in C:
```

```
    clf = LogisticRegression(penalty = 'l1', C=c, solver = 'liblinear')
    clf.fit(X_train, Y_train)
    print('C:', c)
    print('Training accuracy:', clf.score(X_train_sc, Y_train))
    print('Test accuracy:', clf.score(X_test_sc, Y_test))
    print('')
```

```
C: 100
Training accuracy: 0.7274725274725274
Test accuracy: 0.7807017543859649

C: 50
Training accuracy: 0.7208791208791209
Test accuracy: 0.8333333333333334

C: 30
Training accuracy: 0.7120879120879121
Test accuracy: 0.8157894736842105

C: 10
Training accuracy: 0.6967032967032967
Test accuracy: 0.7719298245614035

C: 1
Training accuracy: 0.6505494505494506
Test accuracy: 0.5789473684210527
```

Problem_4

October 9, 2022

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns
from sklearn.datasets import load_breast_cancer
```

[3]:
```
breast = load_breast_cancer()
breast_data = breast.data
breast_data.shape
```

[3]:
```
(569, 30)
```

[4]:
```
breast_input = pd.DataFrame(breast_data)
breast_input.head()
```

[4]:
```
       0      1       2       3        4        5       6        7       8   \
0  17.99  10.38  122.80  1001.0  0.11840  0.27760  0.3001  0.14710  0.2419
1  20.57  17.77  132.90  1326.0  0.08474  0.07864  0.0869  0.07017  0.1812
2  19.69  21.25  130.00  1203.0  0.10960  0.15990  0.1974  0.12790  0.2069
3  11.42  20.38   77.58   386.1  0.14250  0.28390  0.2414  0.10520  0.2597
4  20.29  14.34  135.10  1297.0  0.10030  0.13280  0.1980  0.10430  0.1809

        9   …     20     21      22      23      24      25      26      27  \
0  0.07871  …  25.38  17.33  184.60  2019.0  0.1622  0.6656  0.7119  0.2654
1  0.05667  …  24.99  23.41  158.80  1956.0  0.1238  0.1866  0.2416  0.1860
2  0.05999  …  23.57  25.53  152.50  1709.0  0.1444  0.4245  0.4504  0.2430
3  0.09744  …  14.91  26.50   98.87   567.7  0.2098  0.8663  0.6869  0.2575
4  0.05883  …  22.54  16.67  152.20  1575.0  0.1374  0.2050  0.4000  0.1625

       28       29
0  0.4601  0.11890
```

```
1  0.2750   0.08902
2  0.3613   0.08758
3  0.6638   0.17300
4  0.2364   0.07678

[5 rows x 30 columns]
```

[5]:
```
breast_labels = breast.target
breast_labels.shape
```

[5]:
```
(569,)
```

[6]:
```
labels = np.reshape(breast_labels,(569,1))
```

[7]:
```
final_breast_data = np.concatenate([breast_data,labels],axis=1)
final_breast_data.shape
```

[7]:
```
(569, 31)
```

[8]:
```
breast_dataset = pd.DataFrame(final_breast_data)
features = breast.feature_names
features
```

[8]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

[9]:
```
features_labels = np.append(features,'label')
breast_dataset.columns = features_labels
breast_dataset.head()
```

[9]:
```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
```

```
2           0.15990          0.1974              0.12790          0.2069
3           0.28390          0.2414              0.10520          0.2597
4           0.13280          0.1980              0.10430          0.1809

   mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                 0.07871  …          17.33           184.60      2019.0
1                 0.05667  …          23.41           158.80      1956.0
2                 0.05999  …          25.53           152.50      1709.0
3                 0.09744  …          26.50            98.87       567.7
4                 0.05883  …          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  label
0          0.4601                  0.11890    0.0
1          0.2750                  0.08902    0.0
2          0.3613                  0.08758    0.0
3          0.6638                  0.17300    0.0
4          0.2364                  0.07678    0.0

[5 rows x 31 columns]
```

[10]:
```python
breast_dataset.shape
```

[10]:
```
(569, 31)
```

[11]:
```python
X = breast_dataset.values[:,0:30]
print('X =', X[0:5])
```

```
X = [[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
```

```
 3.613e-01 8.758e-02]
[1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
 1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
 9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
 2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
 6.638e-01 1.730e-01]
[2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.980e-01
 1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.444e+01
 1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.254e+01
 1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.625e-01
 2.364e-01 7.678e-02]]
```

[12]:
```python
Y = breast_dataset.values[:,30]
print('Y =', Y[0:5])
```

```
Y = [0. 0. 0. 0. 0.]
```

[13]:
```python
# Splitting the datasets to training and validation sets.

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state = 0)
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(455, 30) (114, 30) (455,) (114,)
```

[14]:
```python
# Feature scaling between 0 and 1 for independent variables using␣
 ↪Standardization.

from sklearn.preprocessing import StandardScaler, MinMaxScaler
sc_X = MinMaxScaler()
X_train_sc = sc_X.fit_transform(X_train)
X_test_sc = sc_X.fit_transform(X_test)
```

[15]:
```python
print('New X_train =', X_train_sc[0:5])
```

```
New X_train = [[0.1452506  0.32448133 0.14249188 0.07096501 0.52210275
0.18450791
  0.05883318 0.08822068 0.41919192 0.28117102 0.05446315 0.36571782
  0.04810818 0.01798599 0.28172272 0.09191276 0.04267677 0.1523584
  0.2448393  0.06506087 0.11490573 0.39498934 0.10742567 0.04885961
  0.45585419 0.10954585 0.08426518 0.22387186 0.26197516 0.14167651]
 [0.18074684 0.50912863 0.17275931 0.09179215 0.38427284 0.13029929
  0.08467666 0.06978131 0.48282828 0.20661331 0.07104834 0.27864215
  0.06987702 0.024248   0.22343526 0.14441073 0.0755303  0.19795416
  0.19705849 0.06202065 0.17182497 0.53358209 0.16574531 0.07478864
  0.39047745 0.13806987 0.15391374 0.25783672 0.27597083 0.14154532]
 [0.43348005 0.21369295 0.41814664 0.27847296 0.45965027 0.22474488
  0.12886598 0.2250497  0.34090909 0.18513058 0.04606192 0.06121818
  0.04579937 0.02729489 0.10385345 0.07666657 0.04623737 0.16569426
```

```
 0.13285304 0.02508879 0.34791889 0.20149254 0.32616166 0.18745085
 0.32642145 0.14059241 0.18450479 0.38890803 0.23910901 0.09891119]
[0.24605992 0.3373444  0.23495266 0.1304772  0.56337569 0.17529621
 0.05834114 0.14617296 0.42424242 0.34519798 0.09219627 0.25433168
 0.0778872  0.03217365 0.17208678 0.07163457 0.02856061 0.28774389
 0.26723655 0.08682614 0.17431519 0.23720682 0.1580258  0.07618954
 0.28283695 0.06431489 0.03977636 0.20261798 0.13049478 0.1227863 ]
[0.2493729  0.52821577 0.23764771 0.13700954 0.31812751 0.11170468
 0.04015933 0.06267396 0.24444444 0.2064027  0.04070252 0.1721181
 0.03863733 0.0202104  0.1564972  0.07358729 0.02376263 0.08620951
 0.15301056 0.05196717 0.22198506 0.53224947 0.21081727 0.10757471
 0.35944001 0.14854809 0.09824281 0.21822253 0.3025823  0.17703004]]
```

[16]:

```python
# Construct a confusion matrix
from sklearn.model_selection import train_test_split
test_size = 0.2
seed = 0
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= test_size,␣
 ↪random_state = seed)
model = LogisticRegression(solver = 'liblinear')
model.fit(X_train_sc, Y_train)
predicted = model.predict(X_test_sc)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

```
[[47  0]
 [ 9 58]]
```

[17]:

```python
# K-fold: 5
kfold = KFold(n_splits = 5, random_state = 0, shuffle = True)
model = LogisticRegression(solver = 'liblinear')
results = cross_val_score(model, X, Y, cv=kfold)
# Output the accuracy. Calculate the mean and std across all folds.
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 95.434% (2.737%)
```

[18]:

```python
# K-fold: 10
kfold = KFold(n_splits = 10, random_state = 0, shuffle = True)
model = LogisticRegression(solver = 'liblinear')
results = cross_val_score(model, X, Y, cv=kfold)
# Output the accuracy. Calculate the mean and std across all folds.
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 95.432% (3.858%)
```

[19]:

```python
from sklearn.model_selection import train_test_split
test_size = 0.2
seed = 0
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= test_size,␣
 ↪random_state = seed)
model = LogisticRegression(solver = 'liblinear')
model.fit(X_train_sc, Y_train)
predicted = model.predict(X_test_sc)
report = classification_report(Y_test, predicted)
print(report)
```

```
              precision    recall  f1-score   support

         0.0       0.84      1.00      0.91        47
         1.0       1.00      0.87      0.93        67

    accuracy                           0.92       114
   macro avg       0.92      0.93      0.92       114
weighted avg       0.93      0.92      0.92       114
```

[21]:
```
C = [100, 50, 30, 10, 1]

for c in C:
    clf = LogisticRegression(penalty = 'l1', C=c, solver = 'liblinear')
    clf.fit(X_train, Y_train)
    print('C:', c)
    print('Training accuracy:', clf.score(X_train_sc, Y_train))
    print('Test accuracy:', clf.score(X_test_sc, Y_test))
    print('')
```

```
C: 100
Training accuracy: 0.734065934065934
Test accuracy: 0.7982456140350878

C: 50
Training accuracy: 0.7164835164835165
Test accuracy: 0.8333333333333334

C: 30
Training accuracy: 0.7120879120879121
Test accuracy: 0.8333333333333334

C: 10
Training accuracy: 0.6791208791208792
Test accuracy: 0.7631578947368421

C: 1
Training accuracy: 0.6483516483516484
Test accuracy: 0.5789473684210527
```

```
[ ]:
```