Claude Shyaka

ID#: 801326243

Homework 2: Logistic Regression and K-Fold Cross Validation

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

In [2]:
```
# Load the diabetes dataset
ds = pd.read_csv('./data/diabetes.csv')
ds.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288

In [3]:
```
# Seed for random state
seed = 42
# Extract the features and labels
X = ds.iloc[:, :8].values
Y = ds.iloc[:, 8].values
```

In [4]:
```
# Split the data into training and test set.
# Train with 80% and testing with 20%.
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, trai
```

In [5]:
```
# create a scaler object,
# Fit the scaler on the training data and transform
sc = MinMaxScaler()
X_train_sc = sc.fit_transform(X_train)

# Apply the scaler to the test data
X_test_sc = sc.transform(X_test)
```

Problem 1

In [6]:
```
# Create and train the classifier using different
# regularization strengths.
from sklearn import metrics
C = np.arange(1., 2., 0.1)
print(C)
for c in C:
    clf = LogisticRegression(penalty='l2', random_state=seed, C=c,solver='liblin
    clf.fit(X_train_sc, Y_train)
    Y_pred = clf.predict(X_test_sc)
    print("C =", c)
    # print('Training accuracy:', clf.score(X_train_sc, Y_train))
    # print('Test accuracy:', clf.score(X_test_sc, Y_test))
    print("Accuracy:", metrics.accuracy_score(Y_test, Y_pred))
    print("Precision:", metrics.precision_score(Y_test, Y_pred))
    print("Recall:", metrics.recall_score(Y_test, Y_pred))
    print()
```

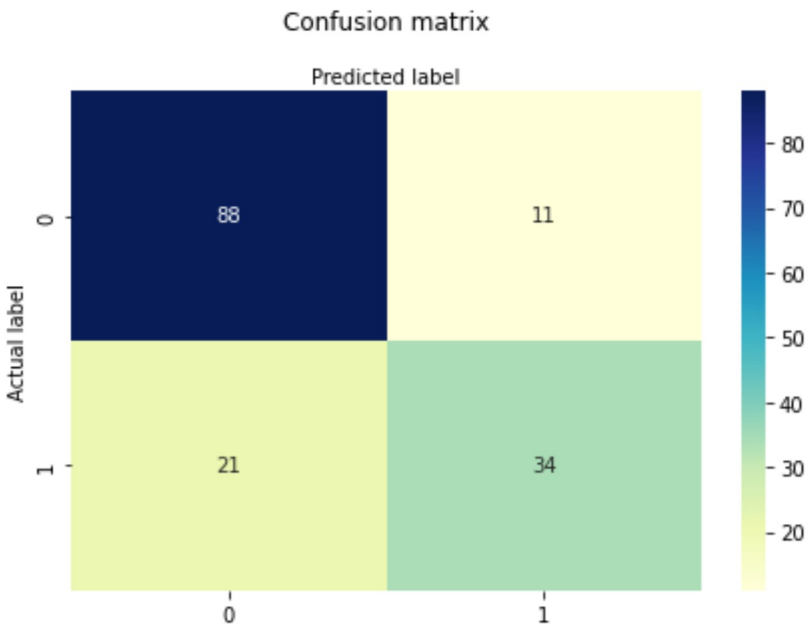
```
[1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9]
C = 1.0
Accuracy: 0.7792207792207793
Precision: 0.7333333333333333
Recall: 0.6

C = 1.1
Accuracy: 0.7857142857142857
Precision: 0.75
Recall: 0.6

C = 1.2000000000000002
Accuracy: 0.7922077922077922
Precision: 0.7555555555555555
Recall: 0.6181818181818182

C = 1.3000000000000003
Accuracy: 0.7922077922077922
Precision: 0.7555555555555555
Recall: 0.6181818181818182

C = 1.4000000000000004
Accuracy: 0.7857142857142857
Precision: 0.7391304347826086
Recall: 0.6181818181818182

C = 1.5000000000000004
Accuracy: 0.7727272727272727
Precision: 0.7083333333333334
Recall: 0.6181818181818182

C = 1.6000000000000005
Accuracy: 0.7662337662337663
Precision: 0.6938775510204082
Recall: 0.6181818181818182

C = 1.7000000000000006
Accuracy: 0.7662337662337663
Precision: 0.6938775510204082
Recall: 0.6181818181818182

C = 1.8000000000000007
Accuracy: 0.7597402597402597
Precision: 0.68
Recall: 0.6181818181818182

C = 1.9000000000000008
Accuracy: 0.7532467532467533
Precision: 0.6666666666666666
Recall: 0.6181818181818182
```

In [7]:

```
clf = LogisticRegression(penalty='l2', random_state=seed, C=1.2, solver='libline
clf.fit(X_train_sc, Y_train)
Y_pred = clf.predict(X_test_sc)
matrix = confusion_matrix(Y_test, Y_pred)
report = classification_report(Y_test, Y_pred)
print("Accuracy:", metrics.accuracy_score(Y_test, Y_pred))
print("Precision:", metrics.precision_score(Y_test, Y_pred))
print("Recall:", metrics.recall_score(Y_test, Y_pred))
print('\nConfusion matrix:\n', matrix)
print('\nClassification metrics:\n', report)
```

```
Accuracy: 0.7922077922077922
Precision: 0.7555555555555555
Recall: 0.6181818181818182

Confusion matrix:
 [[88 11]
 [21 34]]

Classification metrics:
              precision    recall  f1-score   support

           0       0.81      0.89      0.85        99
           1       0.76      0.62      0.68        55

    accuracy                           0.79       154
   macro avg       0.78      0.75      0.76       154
weighted avg       0.79      0.79      0.79       154
```

In [8]:
```
# create heatmap
def create_heatmap(matrix):
    import seaborn as sns
    from matplotlib.colors import ListedColormap
    class_names = [0, 1]
    fig, 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")

create_heatmap(matrix)
```



Problem 2

In [9]:
```
# scale the features
X_sc = sc.fit_transform(X)
```

In [10]:
```
# Select 5 and 10 folds
k_val = (5, 10)
for k in k_val:
    kfold = KFold(n_splits=k, random_state=seed, shuffle=True)
    clf = LogisticRegression(penalty='l2', random_state=seed, C=1.2, solver='lib
    results = cross_val_score(clf, X_sc, Y, cv=kfold)
    print("K =", k)
    print("Average Accuracy:", results.mean())
    print("Standard deviation:", results.std())
    print()
```

```
K = 5
Average Accuracy: 0.7643069348951702
Standard deviation: 0.021316135048651527

K = 10
Average Accuracy: 0.7655502392344498
Standard deviation: 0.056414015882504054
```

Problem 3

In [11]:
```
from sklearn.datasets import load_breast_cancer
```

In [12]:
```
breast_cancer = load_breast_cancer()
breast_cancer_data = breast_cancer.data
breast_cancer_data.shape
```

Out[12]:
```
(569, 30)
```

In [13]:
```
breast_cancer_data_df = pd.DataFrame(breast_cancer_data)
breast_cancer_data_df.head()
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9	...	20	
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	1
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	2
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	2
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	1

5 rows × 30 columns

In [14]:
```
breast_cancer_labels = breast_cancer.target
breast_cancer_labels.shape
```

Out[14]:
```
(569,)
```

In [15]:
```
labels = np.reshape(breast_cancer_labels, (569,1))
```

In [16]:
```
final_breast_cancer_data = np.concatenate([breast_cancer_data, labels], axis=1)
final_breast_cancer_data.shape
```

Out[16]:
```
(569, 31)
```

In [17]:
```
breast_cancer_dataset = pd.DataFrame(final_breast_cancer_data)
```

In [18]:
```
features = breast_cancer.feature_names
features
```

Out[18]:
```
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')
```

In [19]:
```
features_labels = np.append(features, 'label')
```

In [20]:
```
breast_cancer_dataset.columns = features_labels
```

In [21]:
```
breast_cancer_dataset.head()
```

Out[21]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.241
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.181
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.206
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.259
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.180

5 rows × 31 columns

In [22]:
```
breast_cancer_dataset['label'].replace(0, 'Benign', inplace=True)
breast_cancer_dataset['label'].replace(1, 'Malignant', inplace=True)
```

In [23]:
```
breast_cancer_dataset.tail()
```

Out[23]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symme
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1

5 rows × 31 columns

In [24]:
```
breast_cancer_dataset['label'].replace('Benign', 0, inplace=True)
breast_cancer_dataset['label'].replace('Malignant', 1, inplace=True)
```

In [25]:
```
breast_cancer_dataset.tail()
```

Out[25]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	m symme
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1

5 rows × 31 columns

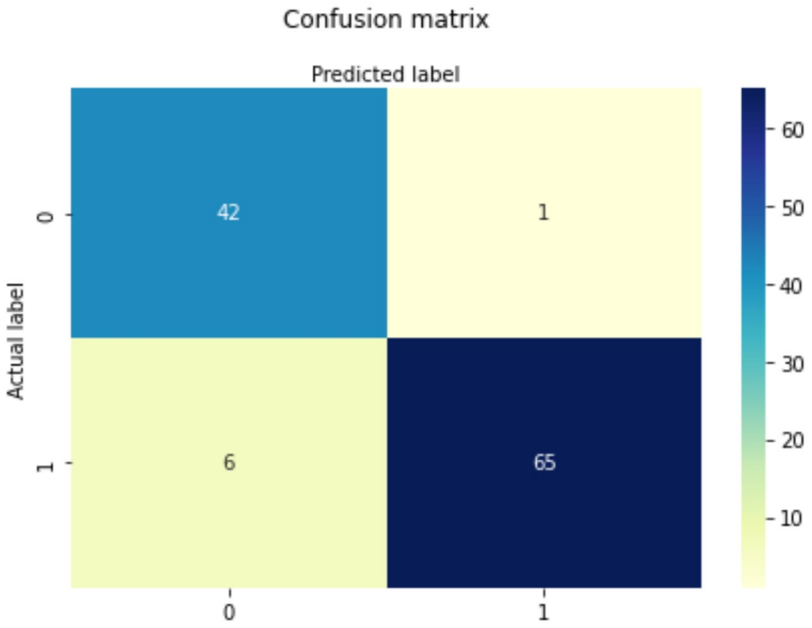
In [26]:
```
X = breast_cancer_dataset.iloc[:, :30].values
Y = breast_cancer_dataset.iloc[:, 30].values
```

In [27]:
```
# Split the data into training and test set.
# Train with 80% and testing with 20%.
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, trai
```

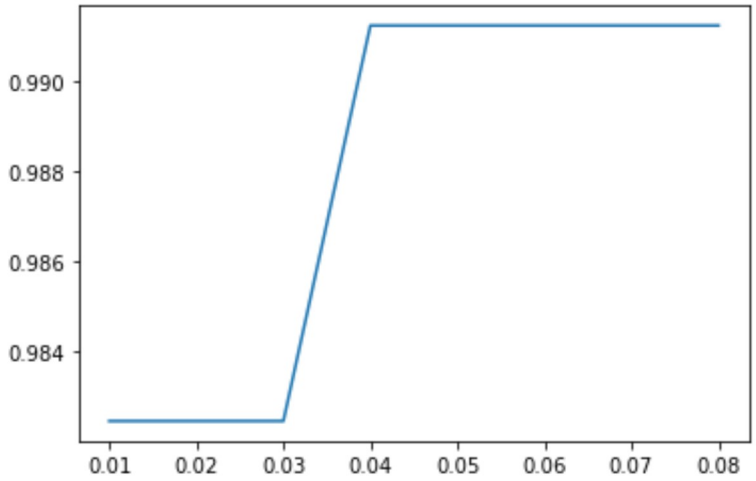
In [28]:
```
# create a scaler object,
# Fit the scaler on the training data and transform
sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)

# Apply the scaler to the test data
X_test_sc = sc.transform(X_test)
```

In [29]:
```
clf = LogisticRegression(penalty='none', random_state=seed, solver='lbfgs')
clf.fit(X_train_sc, Y_train)
Y_pred = clf.predict(X_test_sc)
matrix = confusion_matrix(Y_test, Y_pred)
report = classification_report(Y_test, Y_pred)
print("Accuracy:", metrics.accuracy_score(Y_test, Y_pred))
print("Precision:", metrics.precision_score(Y_test, Y_pred))
print("Recall:", metrics.recall_score(Y_test, Y_pred))
print('\nConfusion matrix:\n', matrix)
print('\nClassification metrics:\n', report)
print()
create_heatmap(matrix)
```

```
Accuracy: 0.9385964912280702
Precision: 0.9848484848484849
Recall: 0.9154929577464789

Confusion matrix:
 [[42  1]
 [ 6 65]]

Classification metrics:
              precision    recall  f1-score   support

           0       0.88      0.98      0.92        43
           1       0.98      0.92      0.95        71

    accuracy                           0.94       114
   macro avg       0.93      0.95      0.94       114
weighted avg       0.94      0.94      0.94       114
```



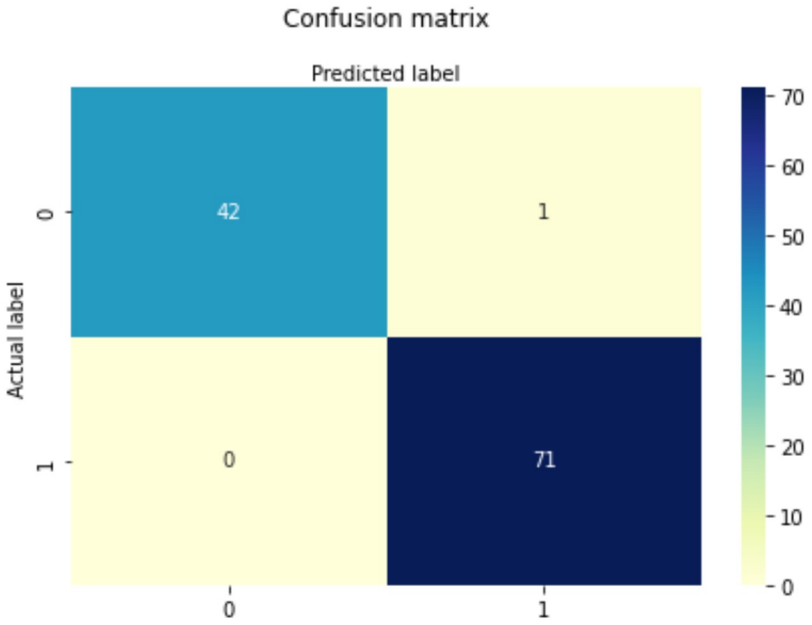
In [30]:
```
# Create and train the classifier using different
# regularization strengths.
from sklearn import metrics
C = np.arange(0.01, 0.09, 0.01)
acc = {}
for c in C:
    clf = LogisticRegression(penalty='l2', random_state=seed, C=c,solver='liblin
    clf.fit(X_train_sc, Y_train)
    Y_pred = clf.predict(X_test_sc)
    acc[c] = metrics.accuracy_score(Y_test, Y_pred)
c_val = max(acc, key=acc.get)
plt.plot(acc.keys(), acc.values())
plt.show()
print("C value corresponding to max accuracy/precision:", c_val)
```




```
C value corresponding to max accuracy/precision: 0.04
```

In [31]:
```
clf = LogisticRegression(penalty='l2', random_state=seed, C=c_val, solver='libli
clf.fit(X_train_sc, Y_train)
Y_pred = clf.predict(X_test_sc)
matrix = confusion_matrix(Y_test, Y_pred)
report = classification_report(Y_test, Y_pred)
print("Accuracy:", metrics.accuracy_score(Y_test, Y_pred))
print("Precision:", metrics.precision_score(Y_test, Y_pred))
print("Recall:", metrics.recall_score(Y_test, Y_pred))
print('\nConfusion matrix:\n', matrix)
print('\nClassification metrics:\n', report)
print()
create_heatmap(matrix)
```

```
Accuracy: 0.9912280701754386
Precision: 0.9861111111111112
Recall: 1.0

Confusion matrix:
 [[42  1]
 [ 0 71]]

Classification metrics:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        43
           1       0.99      1.00      0.99        71

    accuracy                           0.99       114
   macro avg       0.99      0.99      0.99       114
weighted avg       0.99      0.99      0.99       114
```



Problem 4

In [32]:
```
# scale the features
X_sc = sc.fit_transform(X)
```

In [33]:
```
# Cross validation without weight penality.
# Select 5 and 10 folds
k_val = (5, 10)
for k in k_val:
    kfold = KFold(n_splits=k, random_state=seed, shuffle=True)
    clf = LogisticRegression(penalty='none', random_state=seed, solver='lbfgs')
    results = cross_val_score(clf, X_sc, Y, cv=kfold)
    print("K =", k)
    print("Average Accuracy:", results.mean())
    print("Standard deviation:", results.std())
    print()
```

```
K = 5
Average Accuracy: 0.9542617605961807
Standard deviation: 0.020459657053801185

K = 10
Average Accuracy: 0.9543546365914788
Standard deviation: 0.03155122766467967
```

```
C:\Users\eliec\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:81
4: 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-regressio
n
  n_iter_i = _check_optimize_result(
C:\Users\eliec\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:81
4: 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-regressio
n
  n_iter_i = _check_optimize_result(
C:\Users\eliec\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:81
4: 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-regressio
n
  n_iter_i = _check_optimize_result(
C:\Users\eliec\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:81
4: 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-regressio
n
  n_iter_i = _check_optimize_result(
C:\Users\eliec\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:81
4: 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-regressio
n
  n_iter_i = _check_optimize_result(
C:\Users\eliec\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:81
4: 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-regressio
n
  n_iter_i = _check_optimize_result(
C:\Users\eliec\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:81
4: 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-regressio
n
  n_iter_i = _check_optimize_result(
C:\Users\eliec\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:81
4: 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-regressio
n
  n_iter_i = _check_optimize_result(
```

In [34]:

```
# Cross validation with weight penality and lib
# Select 5 and 10 folds
k_val = (5, 10)
for k in k_val:
    kfold = KFold(n_splits=k, random_state=seed, shuffle=True)
    clf = LogisticRegression(penalty='l2', random_state=seed, solver='liblinear'
    results = cross_val_score(clf, X_sc, Y, cv=kfold)
    print("K =", k)
    print("Average Accuracy:", results.mean())
    print("Standard deviation:", results.std())
    print()
```

```
K = 5
Average Accuracy: 0.9771464058376029
Standard deviation: 0.008964382519164905

K = 10
Average Accuracy: 0.9771616541353383
Standard deviation: 0.01577930735487523
```

In [ ]:

In [ ]: