# 第三次作品(專題)(3-3)：淺度機器學習分類器的評比實驗

學號：410978040

姓名：黃冠翔

作品目標：

- 利用多元羅吉斯回歸、支援向量機 SVM 和神經網路對資料進行分類學習與測試。
- 學習分類器的原理並進行評比實驗。

本專題計畫執行分類器比較，即採用三種分類器分別對三組資料進行分類學習與測試。其中分類器包括： (1)多元羅吉斯回歸 (Multinomial Logistic Regression) (2)支援向量機 (Support Vector Machine) (3)神經網路 (Neural Network)

三組資料包括： (1)來自 3 個產區，178 瓶葡萄酒，含 13 種葡萄酒成分。 (2)來自 AT&T 40 個人的人臉影像共 400 張，每張大小 64×64。 (3)來自 Yale Face 38 人的人臉影像共 2410 張，每張大小 192×168。

此檔案以 Yale Face 38 人的人臉影像資料進行分類學習與測試。

先讀取資料並設定變數，同時將資料標準化。

```python
import pandas as pd
import numpy as np
import scipy.io
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Read data
df = scipy.io.loadmat('allFaces.mat')
X = df['faces'] # 32256 x 2410, each column represents an image
y = np.ndarray.flatten(df['nfaces'])
m = int(df['m']) # 168
n = int(df['n']) # 192
n_persons = int(df['person']) # 38
Y=[]
i=0
for yi in y:
    i=i+1
    Y=Y+([i]*(yi))
Y=np.array(Y)
# Split data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(
X.T, Y, test_size=0.30)
#X_train, X_test, y_train, y_test = train_test_split(X.T, Y, 
test_size=0.30, random_state=100)
# Standardize data
scaler = StandardScaler()
X_train_ = scaler.fit_transform(X_train)
X_test_ = scaler.fit_transform(X_test)
```

(1)多元羅吉斯回歸 (Multinomial Logistic Regression)

(a)原始資料

1.使用 lbfgs 的演算法

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'lbfgs' # ’lbfgs’ is the default
# solver = ’liblinear’
# solver = ’newton−cg’
clf_original = LogisticRegression(solver = solver, **opts)
clf_original.fit(X_train_, y_train)
y_pred = clf_original.predict(X_test_)
# 測試資料之準確率回報
print(f"{accuracy_score(y_test, y_pred):.2%}\n")
print(f"{clf_original.score(X_test_, y_test):.2%}\n")
print(classification_report(y_test, y_pred))
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 35.2min finished

95.57%

95.57%

              precision    recall  f1-score   support

           1       1.00      0.95      0.97        20
           2       0.95      0.95      0.95        19
           3       1.00      1.00      1.00        23
           4       0.93      1.00      0.97        14
           5       0.91      1.00      0.95        10
           6       0.92      1.00      0.96        22
           7       1.00      0.95      0.98        21
           8       0.91      1.00      0.95        20
           9       0.95      1.00      0.97        19
          10       0.84      1.00      0.91        16
          11       1.00      0.95      0.97        20
          12       1.00      1.00      1.00        12
          13       0.94      0.88      0.91        17
          14       1.00      0.88      0.93        24
          15       1.00      1.00      1.00        14
          16       1.00      0.86      0.93        22
          17       1.00      1.00      1.00        14
          18       1.00      0.91      0.95        22
          19       1.00      0.96      0.98        24
          20       0.95      1.00      0.98        21
```

```
          21       0.95      0.83      0.89        24
          22       1.00      0.95      0.98        21
          23       1.00      1.00      1.00        20
          24       0.94      1.00      0.97        15
          25       1.00      0.94      0.97        18
          26       0.95      1.00      0.98        20
          27       0.95      1.00      0.97        19
          28       1.00      0.95      0.98        22
          29       0.82      1.00      0.90        14
          30       1.00      0.91      0.95        23
          31       0.95      1.00      0.98        21
          32       0.92      1.00      0.96        24
          33       0.88      1.00      0.94        15
          34       0.95      0.86      0.90        22
          35       1.00      0.96      0.98        23
          36       0.81      0.93      0.87        14
          37       0.93      0.87      0.90        15
          38       0.90      0.95      0.92        19

    accuracy                           0.96       723
   macro avg       0.95      0.96      0.96       723
weighted avg       0.96      0.96      0.96       723
```

2.使用 liblinear 的演算法

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'liblinear'
# solver = ’newton–cg’
clf_original = LogisticRegression(solver = solver, **opts)
clf_original.fit(X_train_, y_train)
y_pred = clf_original.predict(X_test_)
# 測試資料之準確率回報
print(f"{accuracy_score(y_test, y_pred):.2%}\n")
print(f"{clf_original.score(X_test_, y_test):.2%}\n")
print(classification_report(y_test, y_pred))
```

```
[LibLinear]98.76%

98.76%

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        20
           2       1.00      0.95      0.97        19
           3       1.00      1.00      1.00        23
           4       1.00      1.00      1.00        14
```

```
           5       0.91      1.00      0.95        10
           6       1.00      1.00      1.00        22
           7       1.00      1.00      1.00        21
           8       0.91      1.00      0.95        20
           9       1.00      1.00      1.00        19
          10       1.00      1.00      1.00        16
          11       1.00      1.00      1.00        20
          12       1.00      1.00      1.00        12
          13       1.00      0.94      0.97        17
          14       1.00      0.96      0.98        24
          15       1.00      1.00      1.00        14
          16       1.00      0.91      0.95        22
          17       1.00      1.00      1.00        14
          18       1.00      0.95      0.98        22
          19       1.00      1.00      1.00        24
          20       1.00      1.00      1.00        21
          21       1.00      1.00      1.00        24
          22       1.00      1.00      1.00        21
          23       1.00      1.00      1.00        20
          24       1.00      0.93      0.97        15
          25       1.00      1.00      1.00        18
          26       1.00      1.00      1.00        20
          27       1.00      1.00      1.00        19
          28       1.00      1.00      1.00        22
          29       1.00      1.00      1.00        14
          30       1.00      1.00      1.00        23
          31       1.00      1.00      1.00        21
          32       0.96      1.00      0.98        24
          33       0.94      1.00      0.97        15
          34       1.00      0.91      0.95        22
          35       1.00      1.00      1.00        23
          36       0.82      1.00      0.90        14
          37       0.94      1.00      0.97        15
          38       1.00      1.00      1.00        19

    accuracy                           0.99       723
   macro avg       0.99      0.99      0.99       723
weighted avg       0.99      0.99      0.99       723
```

3.使用 newton-cg 的演算法

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'newton-cg'
clf_original = LogisticRegression(solver = solver, **opts)
clf_original.fit(X_train_, y_train)
```

```python
y_pred = clf_original.predict(X_test_)
# 測試資料之準確率回報
print(f"{accuracy_score(y_test, y_pred):.2%}\n")
print(f"{clf_original.score(X_test_, y_test):.2%}\n")
print(classification_report(y_test, y_pred))
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 118.6min finished

95.57%

95.57%

              precision    recall  f1-score   support

           1       1.00      0.95      0.97        20
           2       0.95      0.95      0.95        19
           3       1.00      1.00      1.00        23
           4       0.93      1.00      0.97        14
           5       0.91      1.00      0.95        10
           6       0.92      1.00      0.96        22
           7       1.00      0.95      0.98        21
           8       0.91      1.00      0.95        20
           9       0.95      1.00      0.97        19
          10       0.84      1.00      0.91        16
          11       1.00      0.95      0.97        20
          12       1.00      1.00      1.00        12
          13       0.94      0.88      0.91        17
          14       1.00      0.88      0.93        24
          15       1.00      1.00      1.00        14
          16       1.00      0.86      0.93        22
          17       1.00      1.00      1.00        14
          18       1.00      0.91      0.95        22
          19       1.00      0.96      0.98        24
          20       0.95      1.00      0.98        21
          21       0.95      0.83      0.89        24
          22       1.00      0.95      0.98        21
          23       1.00      1.00      1.00        20
          24       0.94      1.00      0.97        15
          25       1.00      0.94      0.97        18
          26       0.95      1.00      0.98        20
          27       0.95      1.00      0.97        19
          28       1.00      0.95      0.98        22
          29       0.82      1.00      0.90        14
          30       1.00      0.91      0.95        23
          31       0.95      1.00      0.98        21
          32       0.92      1.00      0.96        24
          33       0.88      1.00      0.94        15
          34       0.95      0.86      0.90        22
```

```
          35       1.00      0.96      0.98        23
          36       0.81      0.93      0.87        14
          37       0.93      0.87      0.90        15
          38       0.90      0.95      0.92        19

    accuracy                           0.96       723
   macro avg       0.95      0.96      0.96       723
weighted avg       0.96      0.96      0.96       723
```

討論：

- 使用 lbfgs 的演算法時，準確率為 95.57%，執行時間約 35 分鐘。
- 使用 liblinear 的演算法時，準確率為 98.76%，執行時間約 92 分鐘。
- 使用 newton-cg 的演算法時，準確率為 95.57%，執行時間約 118 分鐘。
- 綜上所述，liblinear 之準確率最高，其餘兩者相同，但論執行時間，lbfgs 的執行時間最短，其餘兩者皆很費時。

(b)主成分資料

1.取 50 個主成分並使用 lbfgs 的演算法

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 50).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'lbfgs' # 'lbfgs' is the default
clf_PCA = LogisticRegression(solver = solver, **opts)
clf_PCA.fit(Z_train, y_train)
y_pred = clf_PCA.predict(Z_test)
print(f"{clf_PCA.score(Z_test, y_test):.2%}\n")

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.

90.32%


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   17.1s finished
```

2.取 100 個主成分並使用 lbfgs 的演算法

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 100).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'lbfgs' # 'lbfgs' is the default
clf_PCA = LogisticRegression(solver = solver, **opts)
```

```
clf_PCA.fit(Z_train, y_train)
y_pred = clf_PCA.predict(Z_test)
print(f"{clf_PCA.score(Z_test, y_test):.2%}\n")
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.

94.19%


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   26.7s finished
```

3.取 500 個主成分並使用 lbfgs 的演算法

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 500).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'lbfgs' # 'lbfgs' is the default
clf_PCA = LogisticRegression(solver = solver, **opts)
clf_PCA.fit(Z_train, y_train)
y_pred = clf_PCA.predict(Z_test)
print(f"{clf_PCA.score(Z_test, y_test):.2%}\n")
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.

94.19%


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.8min finished
```

4.取 50 個主成分並使用 liblinear 的演算法

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 50).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'liblinear' # 'lbfgs' is the default
clf_PCA = LogisticRegression(solver = solver, **opts)
clf_PCA.fit(Z_train, y_train)
y_pred = clf_PCA.predict(Z_test)
print(f"{clf_PCA.score(Z_test, y_test):.2%}\n")
```

```
[LibLinear]89.76%
```

5.取 100 個主成分並使用 liblinear 的演算法

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 100).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'liblinear' # 'lbfgs' is the default
clf_PCA = LogisticRegression(solver = solver, **opts)
clf_PCA.fit(Z_train, y_train)
y_pred = clf_PCA.predict(Z_test)
print(f"{clf_PCA.score(Z_test, y_test):.2%}\n")
```

```
[LibLinear]95.30%
```

6.取 500 個主成分並使用 liblinear 的演算法

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 500).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'liblinear' # 'lbfgs' is the default
clf_PCA = LogisticRegression(solver = solver, **opts)
clf_PCA.fit(Z_train, y_train)
y_pred = clf_PCA.predict(Z_test)
print(f"{clf_PCA.score(Z_test, y_test):.2%}\n")
```

```
[LibLinear]98.34%
```

7.取 50 個主成分並使用 newton-cg 的演算法

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 50).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'newton-cg' # 'lbfgs' is the default
clf_PCA = LogisticRegression(solver = solver, **opts)
clf_PCA.fit(Z_train, y_train)
y_pred = clf_PCA.predict(Z_test)
print(f"{clf_PCA.score(Z_test, y_test):.2%}\n")
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.

90.32%


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  5.1min finished
```

8.取 100 個主成分並使用 newton-cg 的演算法

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 100).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'newton-cg' # 'lbfgs' is the default
clf_PCA = LogisticRegression(solver = solver, **opts)
clf_PCA.fit(Z_train, y_train)
y_pred = clf_PCA.predict(Z_test)
print(f"{clf_PCA.score(Z_test, y_test):.2%}\n")
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.

94.61%


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.6min finished
```

9.取 500 個主成分並使用 newton-cg 的演算法

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 500).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
opts = dict(tol = 1e-6, max_iter = int(1e6), verbose=1)
solver = 'newton-cg' # 'lbfgs' is the default
clf_PCA = LogisticRegression(solver = solver, **opts)
clf_PCA.fit(Z_train, y_train)
y_pred = clf_PCA.predict(Z_test)
print(f"{clf_PCA.score(Z_test, y_test):.2%}\n")
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1
concurrent workers.

94.19%


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.3min finished
```

討論：

使用 lbfgs 的演算法：

- 取 50 個主成分時，準確率為 90.32%。
- 取 100 個主成分時，準確率為 94.19%。
- 取 500 個主成分時，準確率為 94.19%。

使用 liblinear 的演算法：

- 取 50 個主成分時，準確率為 89.76%。
- 取 100 個主成分時，準確率為 95.30%。
- 取 500 個主成分時，準確率為 98.34%。

使用 newton-cg 的演算法：

- 取 50 個主成分時，準確率為 90.32%。
- 取 100 個主成分時，準確率為 94.61%。
- 取 500 個主成分時，準確率為 94.19%。

小結：

- 使用 liblinear 的演算法時，準確率最高；使用 lbfgs 的演算法和使用 newton-cg 的演算法所得之準確率差不多，但 newton-cg 執行時間最長。
- 原則上，取愈多主成分，準確率愈高。

(2)支援向量機 (Support Vector Machine)

(a)原始資料

1.使用 kernel="linear"

```
from sklearn.svm import SVC, LinearSVC
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
# opts = dict(C = C, decision_function_shape = ’ovo’, \
# tol = 1e−6, max_iter = int(1e6))
clf_svm = SVC(kernel="linear", **opts)
# clf_svm = SVC(kernel=”rbf”, gamma=0.2, **opts)
# clf_svm = SVC(kernel=”poly”, degree=3, gamma=”auto”, **opts)
# clf_svm = LinearSVC(**opts) # one vs the rest
clf_svm.fit(X_train_, y_train)
predictions = clf_svm.predict(X_test_)
print(f"{accuracy_score(y_test, predictions):.2%}\n")
print(classification_report(y_test, predictions))
```

```
92.25%

              precision    recall  f1-score   support

           1       1.00      0.95      0.97        20
           2       0.86      0.95      0.90        19
           3       0.96      1.00      0.98        23
           4       0.87      0.93      0.90        14
           5       0.77      1.00      0.87        10
           6       0.88      1.00      0.94        22
           7       0.77      0.95      0.85        21
           8       0.83      0.95      0.88        20
           9       1.00      1.00      1.00        19
          10       0.76      1.00      0.86        16
          11       1.00      0.95      0.97        20
```

```
          12       1.00      1.00      1.00        12
          13       1.00      0.88      0.94        17
          14       1.00      0.83      0.91        24
          15       0.93      0.93      0.93        14
          16       1.00      0.91      0.95        22
          17       1.00      1.00      1.00        14
          18       0.95      0.86      0.90        22
          19       1.00      0.92      0.96        24
          20       0.91      1.00      0.95        21
          21       0.95      0.79      0.86        24
          22       1.00      0.95      0.98        21
          23       0.91      1.00      0.95        20
          24       0.88      1.00      0.94        15
          25       0.95      1.00      0.97        18
          26       0.90      0.95      0.93        20
          27       0.95      1.00      0.97        19
          28       1.00      0.95      0.98        22
          29       0.88      1.00      0.93        14
          30       0.84      0.91      0.87        23
          31       0.95      0.95      0.95        21
          32       0.92      1.00      0.96        24
          33       0.93      0.93      0.93        15
          34       0.90      0.82      0.86        22
          35       0.90      0.78      0.84        23
          36       0.86      0.86      0.86        14
          37       1.00      0.60      0.75        15
          38       1.00      0.63      0.77        19

    accuracy                           0.92       723
   macro avg       0.93      0.92      0.92       723
weighted avg       0.93      0.92      0.92       723
```

2.使用 kernel="rbf"

```
from sklearn.svm import SVC, LinearSVC
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
# opts = dict(C = C, decision_function_shape = ’ovo’, \
# tol = 1e−6, max_iter = int(1e6))
#clf_svm = SVC(kernel="linear", **opts)
clf_svm = SVC(kernel="rbf", gamma='auto', **opts)
# clf_svm = SVC(kernel=”poly”, degree=3, gamma=”auto”, **opts)
# clf_svm = LinearSVC(**opts) # one vs the rest
clf_svm.fit(X_train_, y_train)
predictions = clf_svm.predict(X_test_)
print(f"{accuracy_score(y_test, predictions):.2%}\n")
print(classification_report(y_test, predictions))
```

```
82.57%

              precision    recall  f1-score   support

           1       1.00      0.75      0.86        20
           2       1.00      0.63      0.77        19
           3       1.00      0.91      0.95        23
           4       0.85      0.79      0.81        14
           5       0.67      0.80      0.73        10
           6       0.95      0.86      0.90        22
           7       0.80      0.76      0.78        21
           8       0.76      0.95      0.84        20
           9       1.00      0.89      0.94        19
          10       0.94      0.94      0.94        16
          11       1.00      0.85      0.92        20
          12       0.86      1.00      0.92        12
          13       1.00      0.82      0.90        17
          14       0.89      0.71      0.79        24
          15       1.00      0.93      0.96        14
          16       1.00      0.68      0.81        22
          17       0.93      1.00      0.97        14
          18       0.94      0.73      0.82        22
          19       1.00      0.79      0.88        24
          20       1.00      0.76      0.86        21
          21       1.00      0.79      0.88        24
          22       1.00      0.90      0.95        21
          23       0.95      0.95      0.95        20
          24       1.00      0.73      0.85        15
          25       1.00      1.00      1.00        18
          26       0.94      0.80      0.86        20
          27       1.00      0.89      0.94        19
          28       0.85      0.77      0.81        22
          29       0.76      0.93      0.84        14
          30       0.83      0.83      0.83        23
          31       0.38      1.00      0.55        21
          32       0.83      0.79      0.81        24
          33       0.86      0.80      0.83        15
          34       0.77      0.77      0.77        22
          35       0.86      0.83      0.84        23
          36       0.27      0.86      0.41        14
          37       0.67      0.67      0.67        15
          38       0.87      0.68      0.76        19

    accuracy                           0.83       723
   macro avg       0.88      0.83      0.84       723
weighted avg       0.89      0.83      0.84       723
```

3.使用 kernel="poly"

```python
from sklearn.svm import SVC, LinearSVC
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
# opts = dict(C = C, decision_function_shape = ’ovo’, \
# tol = 1e−6, max_iter = int(1e6))
#clf_svm = SVC(kernel="linear", **opts)
# clf_svm = SVC(kernel=”rbf”, gamma=0.2, **opts)
clf_svm = SVC(kernel="poly", degree=1, gamma="auto", **opts)
# clf_svm = LinearSVC(**opts) # one vs the rest
clf_svm.fit(X_train_, y_train)
predictions = clf_svm.predict(X_test_)
print(f"{accuracy_score(y_test, predictions):.2%}\n")
print(classification_report(y_test, predictions))
```

```
79.94%

              precision    recall  f1-score   support

           1       1.00      0.75      0.86        20
           2       1.00      0.68      0.81        19
           3       1.00      0.87      0.93        23
           4       0.55      0.86      0.67        14
           5       0.78      0.70      0.74        10
           6       1.00      0.91      0.95        22
           7       0.78      0.67      0.72        21
           8       0.63      0.85      0.72        20
           9       1.00      0.89      0.94        19
          10       1.00      0.94      0.97        16
          11       1.00      0.85      0.92        20
          12       0.92      1.00      0.96        12
          13       1.00      0.71      0.83        17
          14       1.00      0.67      0.80        24
          15       1.00      0.79      0.88        14
          16       0.89      0.73      0.80        22
          17       0.93      1.00      0.97        14
          18       0.94      0.73      0.82        22
          19       1.00      0.58      0.74        24
          20       1.00      0.71      0.83        21
          21       1.00      0.71      0.83        24
          22       1.00      0.86      0.92        21
          23       1.00      0.95      0.97        20
          24       0.92      0.73      0.81        15
          25       1.00      1.00      1.00        18
          26       0.94      0.80      0.86        20
          27       1.00      0.89      0.94        19
          28       1.00      0.73      0.84        22
          29       0.60      0.86      0.71        14
          30       0.95      0.78      0.86        23
          31       0.49      1.00      0.66        21
          32       0.95      0.79      0.86        24
```

```
          33       0.92      0.80      0.86        15
          34       0.27      0.77      0.40        22
          35       0.90      0.78      0.84        23
          36       0.41      0.86      0.56        14
          37       0.43      0.67      0.53        15
          38       0.93      0.74      0.82        19

    accuracy                           0.80       723
   macro avg       0.87      0.81      0.82       723
weighted avg       0.88      0.80      0.82       723
```

討論：

- 使用 kernel="linear"時，準確率為 92.25%。
- 使用 kernel="rbf"時，準確率為 82.57%。
- 使用 kernel="poly"時，準確率為 79.94%。
- 綜上所述，kernel="linear"之準確率最高，kernel="rbf"次之，kernel="poly"之準確率最低。

(b)主成分資料

1.取 50 個主成分並使用 kernel="linear"

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC, LinearSVC

pca = PCA(n_components = 50).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
clf_svm = SVC(kernel="linear", **opts)
clf_svm.fit(Z_train, y_train)
predictions = clf_svm.predict(Z_test)
print(f"{clf_svm.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
89.90%

              precision    recall  f1-score   support

           1       1.00      0.85      0.92        20
           2       0.86      0.95      0.90        19
           3       1.00      0.96      0.98        23
           4       0.87      0.93      0.90        14
           5       0.83      1.00      0.91        10
           6       0.81      1.00      0.90        22
           7       0.70      0.90      0.79        21
           8       0.80      1.00      0.89        20
```

```
           9       1.00      0.95      0.97        19
          10       0.88      0.88      0.88        16
          11       0.95      0.95      0.95        20
          12       0.92      1.00      0.96        12
          13       1.00      0.94      0.97        17
          14       1.00      0.83      0.91        24
          15       0.92      0.79      0.85        14
          16       0.95      0.86      0.90        22
          17       1.00      1.00      1.00        14
          18       0.76      0.86      0.81        22
          19       0.85      0.92      0.88        24
          20       0.90      0.90      0.90        21
          21       0.90      0.75      0.82        24
          22       1.00      0.95      0.98        21
          23       0.95      1.00      0.98        20
          24       0.81      0.87      0.84        15
          25       1.00      1.00      1.00        18
          26       1.00      0.95      0.97        20
          27       0.90      1.00      0.95        19
          28       0.95      0.91      0.93        22
          29       0.78      1.00      0.88        14
          30       0.81      0.91      0.86        23
          31       1.00      0.90      0.95        21
          32       0.92      1.00      0.96        24
          33       0.81      0.87      0.84        15
          34       0.86      0.82      0.84        22
          35       1.00      0.78      0.88        23
          36       0.75      0.86      0.80        14
          37       1.00      0.47      0.64        15
          38       1.00      0.68      0.81        19

    accuracy                           0.90       723
   macro avg       0.91      0.90      0.90       723
weighted avg       0.91      0.90      0.90       723
```

2.取 100 個主成分並使用 kernel="linear"

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC, LinearSVC

pca = PCA(n_components = 100).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
clf_svm = SVC(kernel="linear", **opts)
clf_svm.fit(Z_train, y_train)
predictions = clf_svm.predict(Z_test)
```

```python
print(f"{clf_svm.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
90.59%

              precision    recall  f1-score   support

           1       0.90      0.90      0.90        20
           2       0.78      0.95      0.86        19
           3       1.00      0.96      0.98        23
           4       0.93      0.93      0.93        14
           5       0.77      1.00      0.87        10
           6       0.85      1.00      0.92        22
           7       0.67      0.95      0.78        21
           8       0.83      1.00      0.91        20
           9       0.94      0.89      0.92        19
          10       0.84      1.00      0.91        16
          11       0.95      0.95      0.95        20
          12       0.92      1.00      0.96        12
          13       1.00      0.88      0.94        17
          14       1.00      0.83      0.91        24
          15       0.87      0.93      0.90        14
          16       0.95      0.82      0.88        22
          17       1.00      1.00      1.00        14
          18       0.79      0.86      0.83        22
          19       0.96      0.92      0.94        24
          20       0.86      0.90      0.88        21
          21       0.95      0.79      0.86        24
          22       1.00      0.95      0.98        21
          23       0.95      1.00      0.98        20
          24       0.94      1.00      0.97        15
          25       0.95      1.00      0.97        18
          26       1.00      0.95      0.97        20
          27       1.00      1.00      1.00        19
          28       1.00      0.91      0.95        22
          29       0.82      1.00      0.90        14
          30       0.91      0.91      0.91        23
          31       0.95      0.90      0.93        21
          32       0.89      1.00      0.94        24
          33       0.93      0.87      0.90        15
          34       0.90      0.82      0.86        22
          35       1.00      0.70      0.82        23
          36       0.76      0.93      0.84        14
          37       0.89      0.53      0.67        15
          38       1.00      0.63      0.77        19

    accuracy                           0.91       723
   macro avg       0.91      0.91      0.90       723
weighted avg       0.92      0.91      0.90       723
```

3.取 500 個主成分並使用 kernel="linear"

```python
from sklearn.decomposition import PCA
from sklearn.svm import SVC, LinearSVC

pca = PCA(n_components = 500).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
clf_svm = SVC(kernel="linear", **opts)
clf_svm.fit(Z_train, y_train)
predictions = clf_svm.predict(Z_test)
print(f"{clf_svm.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
91.56%

              precision    recall  f1-score   support

           1       0.95      0.95      0.95        20
           2       0.75      0.95      0.84        19
           3       0.96      1.00      0.98        23
           4       0.93      0.93      0.93        14
           5       0.77      1.00      0.87        10
           6       0.88      1.00      0.94        22
           7       0.77      0.95      0.85        21
           8       0.83      0.95      0.88        20
           9       1.00      1.00      1.00        19
          10       0.73      1.00      0.84        16
          11       1.00      0.95      0.97        20
          12       1.00      1.00      1.00        12
          13       1.00      0.88      0.94        17
          14       1.00      0.83      0.91        24
          15       0.93      0.93      0.93        14
          16       1.00      0.91      0.95        22
          17       1.00      1.00      1.00        14
          18       0.90      0.86      0.88        22
          19       1.00      0.92      0.96        24
          20       0.91      0.95      0.93        21
          21       0.95      0.79      0.86        24
          22       1.00      0.95      0.98        21
          23       0.91      1.00      0.95        20
          24       0.88      1.00      0.94        15
          25       0.95      1.00      0.97        18
          26       0.90      0.95      0.93        20
          27       0.95      1.00      0.97        19
          28       1.00      0.95      0.98        22
          29       0.88      1.00      0.93        14
          30       0.84      0.91      0.87        23
```

```
          31       0.95      0.95      0.95        21
          32       0.89      1.00      0.94        24
          33       0.93      0.87      0.90        15
          34       0.90      0.82      0.86        22
          35       1.00      0.65      0.79        23
          36       0.80      0.86      0.83        14
          37       1.00      0.60      0.75        15
          38       1.00      0.63      0.77        19

    accuracy                           0.92       723
   macro avg       0.92      0.92      0.91       723
weighted avg       0.93      0.92      0.91       723
```

4.取 50 個主成分並使用 kernel="rbf"

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC, LinearSVC

pca = PCA(n_components = 50).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
clf_svm = SVC(kernel="rbf", gamma='scale', **opts)
clf_svm.fit(Z_train, y_train)
predictions = clf_svm.predict(Z_test)
print(f"{clf_svm.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
72.20%

              precision    recall  f1-score   support

           1       0.87      0.65      0.74        20
           2       1.00      0.47      0.64        19
           3       0.90      0.83      0.86        23
           4       0.50      0.79      0.61        14
           5       0.58      0.70      0.64        10
           6       0.81      0.77      0.79        22
           7       0.60      0.57      0.59        21
           8       0.50      0.85      0.63        20
           9       0.94      0.89      0.92        19
          10       1.00      0.88      0.93        16
          11       1.00      0.70      0.82        20
          12       0.86      1.00      0.92        12
          13       1.00      0.65      0.79        17
          14       0.81      0.54      0.65        24
          15       0.79      0.79      0.79        14
          16       1.00      0.55      0.71        22
```

```
          17       0.70      1.00      0.82        14
          18       0.80      0.55      0.65        22
          19       0.93      0.54      0.68        24
          20       1.00      0.67      0.80        21
          21       1.00      0.71      0.83        24
          22       0.88      0.67      0.76        21
          23       0.89      0.85      0.87        20
          24       1.00      0.53      0.70        15
          25       0.94      0.94      0.94        18
          26       0.94      0.75      0.83        20
          27       1.00      0.84      0.91        19
          28       0.80      0.73      0.76        22
          29       0.60      0.86      0.71        14
          30       0.79      0.65      0.71        23
          31       0.38      1.00      0.55        21
          32       0.82      0.75      0.78        24
          33       0.71      0.80      0.75        15
          34       0.59      0.77      0.67        22
          35       0.79      0.65      0.71        23
          36       0.18      0.64      0.28        14
          37       0.62      0.67      0.65        15
          38       0.73      0.58      0.65        19

    accuracy                           0.72       723
   macro avg       0.80      0.73      0.74       723
weighted avg       0.81      0.72      0.74       723
```

5.取 100 個主成分並使用 kernel="rbf"

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC, LinearSVC

pca = PCA(n_components = 100).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
clf_svm = SVC(kernel="rbf", gamma='scale', **opts)
clf_svm.fit(Z_train, y_train)
predictions = clf_svm.predict(Z_test)
print(f"{clf_svm.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
78.15%

              precision    recall  f1-score   support

           1       0.93      0.65      0.76        20
           2       1.00      0.53      0.69        19
```

```
           3       1.00      0.87      0.93        23
           4       0.85      0.79      0.81        14
           5       0.64      0.70      0.67        10
           6       0.95      0.82      0.88        22
           7       0.68      0.71      0.70        21
           8       0.64      0.90      0.75        20
           9       1.00      0.89      0.94        19
          10       1.00      0.94      0.97        16
          11       1.00      0.80      0.89        20
          12       0.92      1.00      0.96        12
          13       0.93      0.76      0.84        17
          14       0.88      0.62      0.73        24
          15       0.75      0.86      0.80        14
          16       0.94      0.68      0.79        22
          17       0.82      1.00      0.90        14
          18       0.87      0.59      0.70        22
          19       1.00      0.67      0.80        24
          20       1.00      0.67      0.80        21
          21       1.00      0.75      0.86        24
          22       1.00      0.86      0.92        21
          23       0.90      0.90      0.90        20
          24       1.00      0.67      0.80        15
          25       1.00      0.94      0.97        18
          26       0.94      0.80      0.86        20
          27       1.00      0.89      0.94        19
          28       0.89      0.77      0.83        22
          29       0.59      0.93      0.72        14
          30       0.81      0.74      0.77        23
          31       0.38      1.00      0.55        21
          32       0.95      0.79      0.86        24
          33       0.85      0.73      0.79        15
          34       0.65      0.77      0.71        22
          35       0.86      0.78      0.82        23
          36       0.22      0.79      0.34        14
          37       0.59      0.67      0.62        15
          38       0.76      0.68      0.72        19

    accuracy                           0.78       723
   macro avg       0.85      0.79      0.80       723
weighted avg       0.86      0.78      0.80       723
```

6.取 500 個主成分並使用 kernel="rbf"

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC, LinearSVC

pca = PCA(n_components = 500).fit(X_train_)
Z_train = pca.transform(X_train_)
```

```python
Z_test = pca.transform(X_test_)
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
clf_svm = SVC(kernel="rbf", gamma='scale', **opts)
clf_svm.fit(Z_train, y_train)
predictions = clf_svm.predict(Z_test)
print(f"{clf_svm.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
82.43%

              precision    recall  f1-score   support

           1       1.00      0.75      0.86        20
           2       1.00      0.63      0.77        19
           3       1.00      0.91      0.95        23
           4       0.85      0.79      0.81        14
           5       0.67      0.80      0.73        10
           6       0.95      0.91      0.93        22
           7       0.80      0.76      0.78        21
           8       0.76      0.95      0.84        20
           9       1.00      0.89      0.94        19
          10       1.00      0.94      0.97        16
          11       1.00      0.85      0.92        20
          12       0.92      1.00      0.96        12
          13       1.00      0.82      0.90        17
          14       0.89      0.67      0.76        24
          15       1.00      0.93      0.96        14
          16       1.00      0.68      0.81        22
          17       0.93      1.00      0.97        14
          18       0.94      0.73      0.82        22
          19       1.00      0.75      0.86        24
          20       1.00      0.76      0.86        21
          21       1.00      0.79      0.88        24
          22       1.00      0.90      0.95        21
          23       0.95      0.95      0.95        20
          24       1.00      0.73      0.85        15
          25       1.00      1.00      1.00        18
          26       0.94      0.80      0.86        20
          27       1.00      0.89      0.94        19
          28       0.85      0.77      0.81        22
          29       0.72      0.93      0.81        14
          30       0.83      0.83      0.83        23
          31       0.38      1.00      0.55        21
          32       0.79      0.79      0.79        24
          33       0.86      0.80      0.83        15
          34       0.74      0.77      0.76        22
          35       0.86      0.83      0.84        23
          36       0.27      0.86      0.41        14
          37       0.67      0.67      0.67        15
```

```
          38       0.87      0.68      0.76        19

    accuracy                           0.82       723
   macro avg       0.88      0.83      0.84       723
weighted avg       0.89      0.82      0.84       723
```

7.取 50 個主成分並使用 kernel="poly"

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC, LinearSVC

pca = PCA(n_components = 50).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
clf_svm = SVC(kernel="poly", degree=1, gamma="auto", **opts)
clf_svm.fit(Z_train, y_train)
predictions = clf_svm.predict(Z_test)
print(f"{clf_svm.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))

91.84%

              precision    recall  f1-score   support

           1       0.94      0.85      0.89        20
           2       0.95      0.95      0.95        19
           3       1.00      0.96      0.98        23
           4       0.88      1.00      0.93        14
           5       0.83      1.00      0.91        10
           6       0.81      1.00      0.90        22
           7       0.70      0.90      0.79        21
           8       0.95      1.00      0.98        20
           9       1.00      1.00      1.00        19
          10       0.93      0.88      0.90        16
          11       1.00      0.95      0.97        20
          12       0.92      1.00      0.96        12
          13       1.00      0.88      0.94        17
          14       1.00      0.83      0.91        24
          15       1.00      0.79      0.88        14
          16       0.95      0.86      0.90        22
          17       1.00      1.00      1.00        14
          18       0.79      0.86      0.83        22
          19       0.88      0.92      0.90        24
          20       0.95      0.90      0.93        21
          21       0.95      0.75      0.84        24
          22       1.00      0.95      0.98        21
          23       0.95      1.00      0.98        20
```

```
          24       0.88      1.00      0.94        15
          25       1.00      1.00      1.00        18
          26       1.00      0.95      0.97        20
          27       0.95      1.00      0.97        19
          28       0.95      0.91      0.93        22
          29       0.82      1.00      0.90        14
          30       0.91      0.91      0.91        23
          31       0.83      0.90      0.86        21
          32       0.92      1.00      0.96        24
          33       0.94      1.00      0.97        15
          34       0.86      0.82      0.84        22
          35       1.00      0.96      0.98        23
          36       0.72      0.93      0.81        14
          37       0.90      0.60      0.72        15
          38       1.00      0.79      0.88        19

    accuracy                           0.92       723
   macro avg       0.92      0.92      0.92       723
weighted avg       0.93      0.92      0.92       723
```

8.取 100 個主成分並使用 kernel="poly"

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC, LinearSVC

pca = PCA(n_components = 100).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
clf_svm = SVC(kernel="poly", degree=1, gamma="auto", **opts)
clf_svm.fit(Z_train, y_train)
predictions = clf_svm.predict(Z_test)
print(f"{clf_svm.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
93.08%

              precision    recall  f1-score   support

           1       0.95      0.90      0.92        20
           2       1.00      0.95      0.97        19
           3       1.00      0.96      0.98        23
           4       0.93      0.93      0.93        14
           5       0.91      1.00      0.95        10
           6       0.92      1.00      0.96        22
           7       0.74      0.95      0.83        21
           8       0.91      1.00      0.95        20
           9       1.00      0.95      0.97        19
```

```
          10       0.94      1.00      0.97        16
          11       0.95      0.95      0.95        20
          12       1.00      1.00      1.00        12
          13       1.00      0.88      0.94        17
          14       1.00      0.88      0.93        24
          15       1.00      0.93      0.96        14
          16       0.95      0.82      0.88        22
          17       1.00      1.00      1.00        14
          18       0.79      0.86      0.83        22
          19       0.96      0.92      0.94        24
          20       0.95      0.90      0.93        21
          21       0.95      0.79      0.86        24
          22       1.00      0.95      0.98        21
          23       0.95      1.00      0.98        20
          24       0.88      1.00      0.94        15
          25       0.95      1.00      0.97        18
          26       1.00      0.95      0.97        20
          27       1.00      1.00      1.00        19
          28       1.00      0.91      0.95        22
          29       0.82      1.00      0.90        14
          30       0.95      0.91      0.93        23
          31       0.81      1.00      0.89        21
          32       0.89      1.00      0.94        24
          33       0.93      0.93      0.93        15
          34       0.90      0.82      0.86        22
          35       0.96      0.96      0.96        23
          36       0.72      0.93      0.81        14
          37       1.00      0.67      0.80        15
          38       1.00      0.89      0.94        19

    accuracy                           0.93       723
   macro avg       0.94      0.93      0.93       723
weighted avg       0.94      0.93      0.93       723
```

9.取 500 個主成分並使用 kernel="poly"

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC, LinearSVC

pca = PCA(n_components = 500).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)
C = 1 # SVM regularization parameter
opts = dict(C = C, tol = 1e-6, max_iter = int(1e6))
clf_svm = SVC(kernel="poly", degree=1, gamma="auto", **opts)
clf_svm.fit(Z_train, y_train)
predictions = clf_svm.predict(Z_test)
```

```python
print(f"{clf_svm.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
94.05%

              precision    recall  f1-score   support

           1       1.00      0.95      0.97        20
           2       1.00      0.95      0.97        19
           3       1.00      0.96      0.98        23
           4       0.87      0.93      0.90        14
           5       0.91      1.00      0.95        10
           6       0.88      1.00      0.94        22
           7       0.83      0.95      0.89        21
           8       1.00      0.95      0.97        20
           9       1.00      1.00      1.00        19
          10       0.94      1.00      0.97        16
          11       1.00      0.95      0.97        20
          12       1.00      1.00      1.00        12
          13       1.00      0.88      0.94        17
          14       1.00      0.83      0.91        24
          15       1.00      0.93      0.96        14
          16       1.00      0.86      0.93        22
          17       1.00      1.00      1.00        14
          18       0.95      0.86      0.90        22
          19       0.96      0.96      0.96        24
          20       0.91      0.95      0.93        21
          21       0.95      0.83      0.89        24
          22       1.00      0.95      0.98        21
          23       0.95      1.00      0.98        20
          24       1.00      0.93      0.97        15
          25       0.95      1.00      0.97        18
          26       1.00      1.00      1.00        20
          27       1.00      0.95      0.97        19
          28       1.00      0.95      0.98        22
          29       0.88      1.00      0.93        14
          30       0.95      0.91      0.93        23
          31       0.75      1.00      0.86        21
          32       0.92      1.00      0.96        24
          33       0.88      1.00      0.94        15
          34       0.90      0.82      0.86        22
          35       0.96      0.96      0.96        23
          36       0.62      0.93      0.74        14
          37       1.00      0.67      0.80        15
          38       1.00      1.00      1.00        19

    accuracy                           0.94       723
   macro avg       0.95      0.94      0.94       723
weighted avg       0.95      0.94      0.94       723
```

討論：

使用 kernel="linear"：

- 取 50 個主成分時，準確率為 89.90%。
- 取 100 個主成分時，準確率為 90.59%。
- 取 500 個主成分時，準確率為 91.56%。

使用 kernel="rbf"：

- 取 50 個主成分時，準確率為 72.20%。
- 取 100 個主成分時，準確率為 78.15%。
- 取 500 個主成分時，準確率為 82.43%。

使用 kernel="poly"：

- 取 50 個主成分時，準確率為 91.84%。
- 取 100 個主成分時，準確率為 93.08%。
- 取 500 個主成分時，準確率為 94.05%。

小結：

- 使用 kernel="poly"時，準確率最高；使用 kernel="linear"次之；使用 kernel="rbf"準確率最低。
- 取愈多主成分，準確率愈高。

(3)神經網路 (Neural Network)

(a)原始資料

1.使用 activation = 'logistic'且 hidden_layers = (30,)

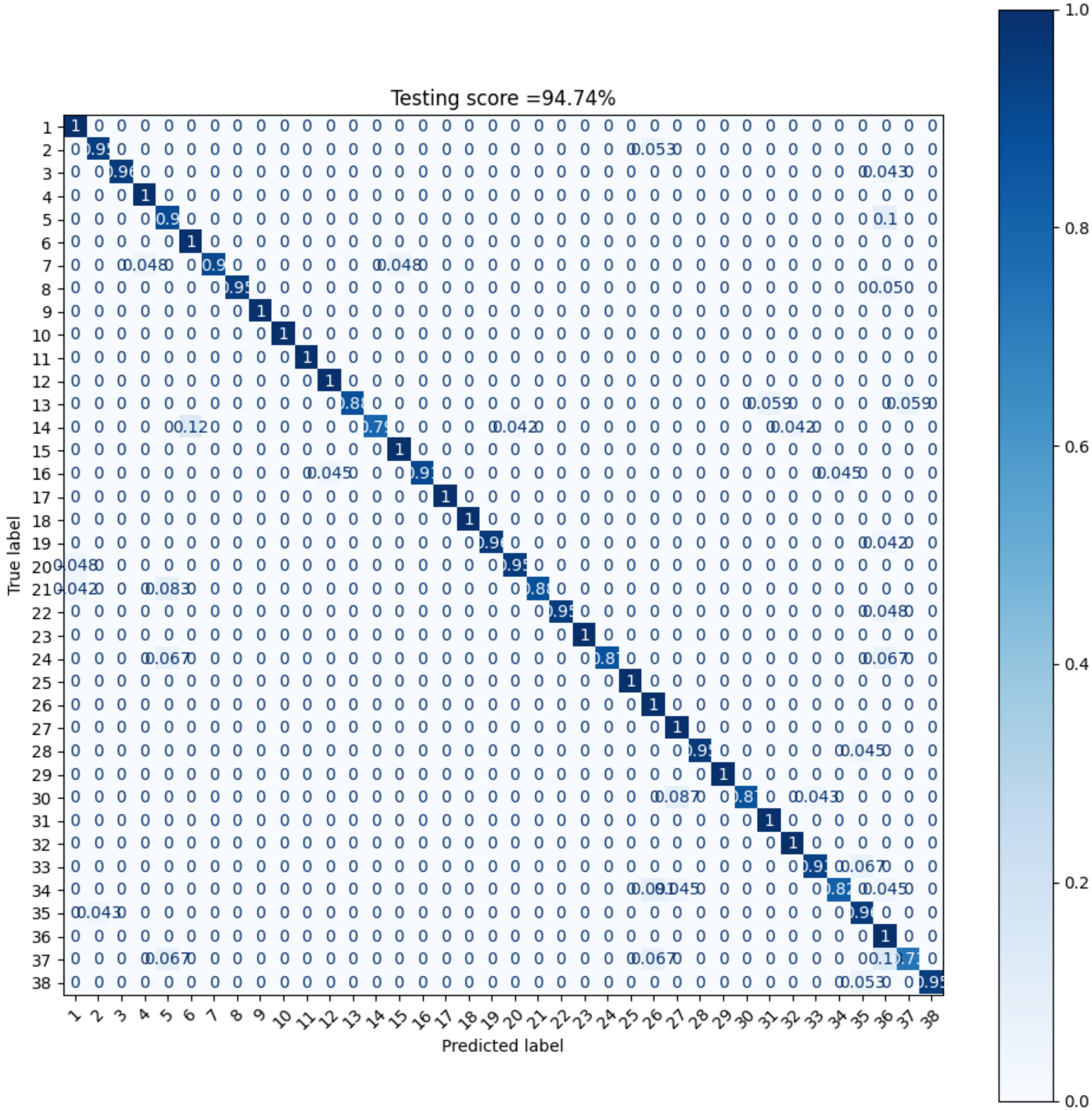
```
from sklearn.neural_network import MLPClassifier
# hidden_layers = (512,) # one hidden layer
# activation = ’relu’ # the default
hidden_layers = (30,)
activation = 'logistic'
opts = dict(hidden_layer_sizes = hidden_layers , verbose = False, \
activation = activation, tol = 1e-6, max_iter = int(1e6))
# solver = ’sgd’ # not efficient, need more tuning
# solver = ’lbfgs’ # not suitable here
solver = 'adam' # default solver
clf_MLP = MLPClassifier(solver = solver, **opts)
clf_MLP.fit(X_train_, y_train)
predictions = clf_MLP.predict(X_test_)
print(f"{accuracy_score(y_test, predictions):.2%}\n")
print(classification_report(y_test, predictions))
```

```
94.74%

              precision    recall  f1-score   support
```

```
           1       0.91      1.00      0.95        20
           2       0.95      0.95      0.95        19
           3       1.00      0.96      0.98        23
           4       0.93      1.00      0.97        14
           5       0.69      0.90      0.78        10
           6       0.88      1.00      0.94        22
           7       1.00      0.90      0.95        21
           8       1.00      0.95      0.97        20
           9       1.00      1.00      1.00        19
          10       1.00      1.00      1.00        16
          11       1.00      1.00      1.00        20
          12       0.92      1.00      0.96        12
          13       1.00      0.88      0.94        17
          14       1.00      0.79      0.88        24
          15       0.93      1.00      0.97        14
          16       1.00      0.91      0.95        22
          17       1.00      1.00      1.00        14
          18       1.00      1.00      1.00        22
          19       1.00      0.96      0.98        24
          20       0.95      0.95      0.95        21
          21       1.00      0.88      0.93        24
          22       1.00      0.95      0.98        21
          23       1.00      1.00      1.00        20
          24       1.00      0.87      0.93        15
          25       1.00      1.00      1.00        18
          26       0.83      1.00      0.91        20
          27       0.86      1.00      0.93        19
          28       1.00      0.95      0.98        22
          29       1.00      1.00      1.00        14
          30       1.00      0.87      0.93        23
          31       0.95      1.00      0.98        21
          32       0.96      1.00      0.98        24
          33       0.93      0.93      0.93        15
          34       0.95      0.82      0.88        22
          35       0.88      0.96      0.92        23
          36       0.61      1.00      0.76        14
          37       0.92      0.73      0.81        15
          38       1.00      0.95      0.97        19

    accuracy                           0.95       723
   macro avg       0.95      0.95      0.95       723
weighted avg       0.96      0.95      0.95       723
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
fig, ax = plt.subplots(1, 1, figsize=(12,12))
score = 100*clf_MLP.score(X_test_, y_test)
title = 'Testing score ={:.2f}%'.format(score)
disp = ConfusionMatrixDisplay.from_estimator(
```

```
clf_MLP,
X_test_,
y_test,
xticks_rotation=45, #'vertical',
# display_labels=class_names,
cmap=plt.cm.Blues,
normalize='true',
ax = ax
)
disp.ax_.set_title(title)
plt.show()
```



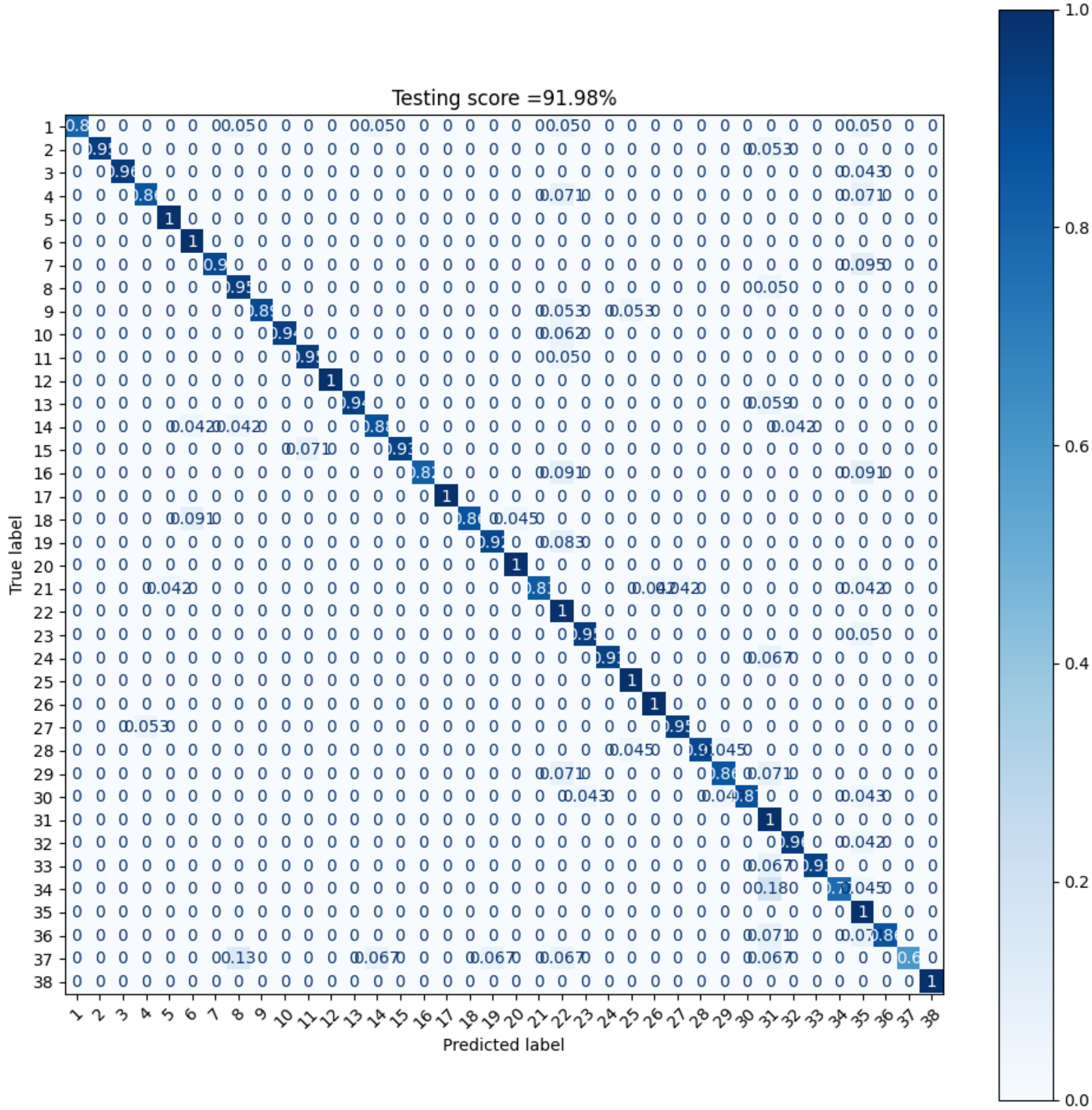
2.使用 activation = 'relu'且 hidden_layers = (512,)

```
from sklearn.neural_network import MLPClassifier
hidden_layers = (512,) # one hidden layer
activation = 'relu' # the default
# hidden_layers = (30,)
# activation = 'logistic'
opts = dict(hidden_layer_sizes = hidden_layers , verbose = False, \
activation = activation, tol = 1e-6, max_iter = int(1e6))
# solver = ’sgd’ # not efficient, need more tuning
# solver = ’lbfgs’ # not suitable here
solver = 'adam' # default solver
clf_MLP = MLPClassifier(solver = solver, **opts)
clf_MLP.fit(X_train_, y_train)
predictions = clf_MLP.predict(X_test_)
print(f"{accuracy_score(y_test, predictions):.2%}\n")
print(classification_report(y_test, predictions))
```

```
91.98%

              precision    recall  f1-score   support

           1       1.00      0.80      0.89        20
           2       1.00      0.95      0.97        19
           3       1.00      0.96      0.98        23
           4       0.92      0.86      0.89        14
           5       0.91      1.00      0.95        10
           6       0.88      1.00      0.94        22
           7       1.00      0.90      0.95        21
           8       0.83      0.95      0.88        20
           9       1.00      0.89      0.94        19
          10       1.00      0.94      0.97        16
          11       0.95      0.95      0.95        20
          12       1.00      1.00      1.00        12
          13       1.00      0.94      0.97        17
          14       0.91      0.88      0.89        24
          15       1.00      0.93      0.96        14
          16       1.00      0.82      0.90        22
          17       1.00      1.00      1.00        14
          18       1.00      0.86      0.93        22
          19       0.96      0.92      0.94        24
          20       0.95      1.00      0.98        21
          21       1.00      0.83      0.91        24
          22       0.66      1.00      0.79        21
          23       0.95      0.95      0.95        20
          24       1.00      0.93      0.97        15
          25       0.90      1.00      0.95        18
          26       0.95      1.00      0.98        20
          27       0.95      0.95      0.95        19
          28       1.00      0.91      0.95        22
```

```
          29       0.86      0.86      0.86        14
          30       1.00      0.87      0.93        23
          31       0.64      1.00      0.78        21
          32       0.96      0.96      0.96        24
          33       1.00      0.93      0.97        15
          34       1.00      0.77      0.87        22
          35       0.64      1.00      0.78        23
          36       1.00      0.86      0.92        14
          37       1.00      0.60      0.75        15
          38       1.00      1.00      1.00        19

    accuracy                           0.92       723
   macro avg       0.94      0.92      0.92       723
weighted avg       0.94      0.92      0.92       723
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
fig, ax = plt.subplots(1, 1, figsize=(12,12))
score = 100*clf_MLP.score(X_test_, y_test)
title = 'Testing score ={:.2f}%'.format(score)
disp = ConfusionMatrixDisplay.from_estimator(
clf_MLP,
X_test_,
y_test,
xticks_rotation=45, #’vertical’,
# display_labels=class_names,
cmap=plt.cm.Blues,
normalize='true',
ax = ax
)
disp.ax_.set_title(title)
plt.show()
```




討論：

- 使用 activation = 'logistic'且 hidden_layers = (30,)時，準確率為 94.74%。
- 使用 activation = 'relu'且 hidden_layers = (512,)時，準確率為 91.98%。
- 綜上所述，使用 activation = 'logistic'且 hidden_layers = (30,)之準確率較高。

(b)主成分資料

1.取 50 個主成分並使用 activation = 'logistic'且 hidden_layers = (30,)

```
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
```

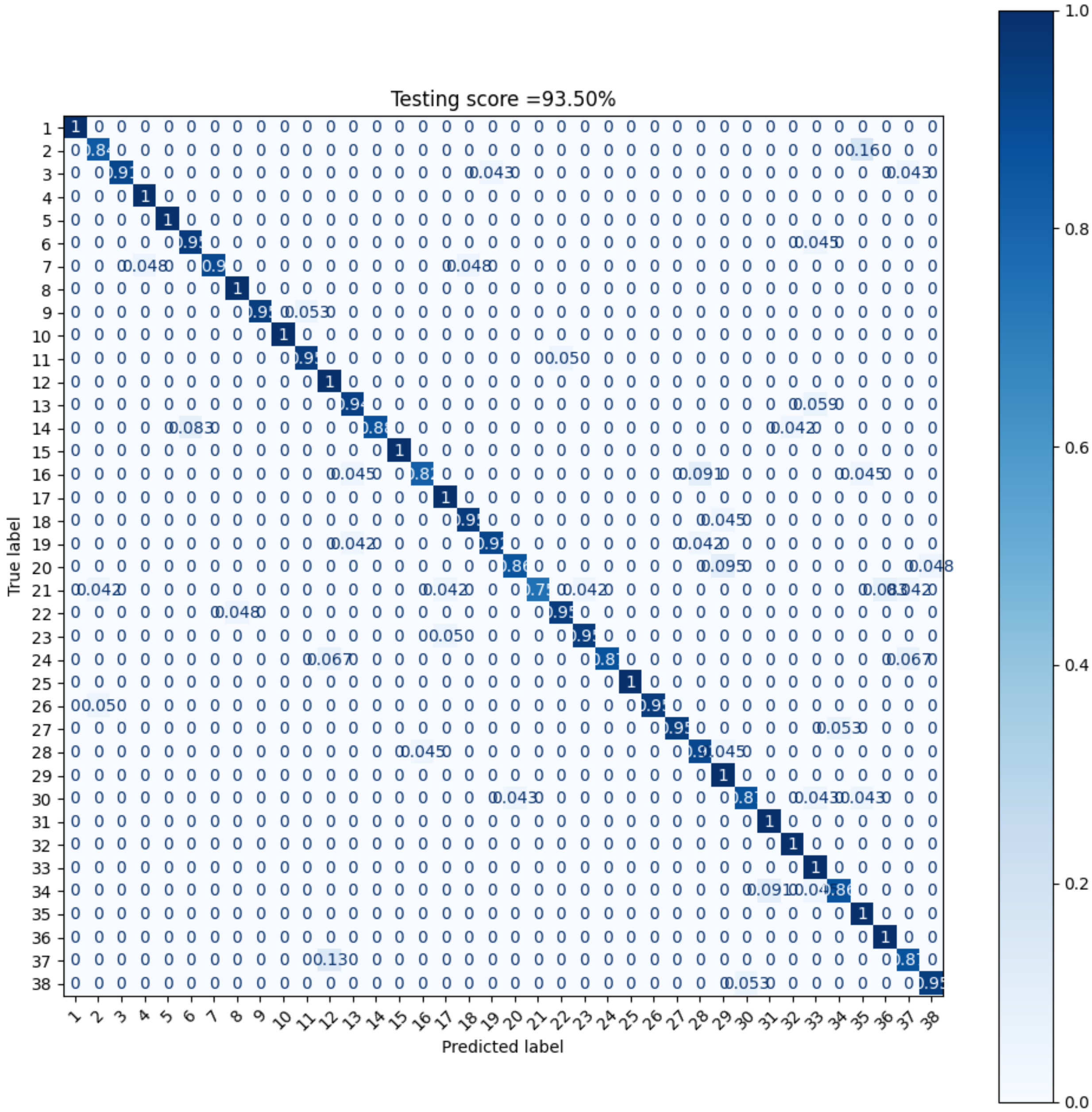
```
pca = PCA(n_components = 50).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)

hidden_layers = (30,)
activation = 'logistic'
opts = dict(hidden_layer_sizes = hidden_layers , verbose = False, \
activation = activation, tol = 1e-6, max_iter = int(1e6))
# solver = 'sgd' # not efficient, need more tuning
# solver = 'lbfgs' # not suitable here
solver = 'adam' # default solver
clf_MLP = MLPClassifier(solver = solver, **opts)
clf_MLP.fit(Z_train, y_train)
predictions = clf_MLP.predict(Z_test)
print(f"{clf_MLP.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
93.50%
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 1.00 | 1.00 | 20 |
| 2 | 0.89 | 0.84 | 0.86 | 19 |
| 3 | 1.00 | 0.91 | 0.95 | 23 |
| 4 | 0.93 | 1.00 | 0.97 | 14 |
| 5 | 1.00 | 1.00 | 1.00 | 10 |
| 6 | 0.91 | 0.95 | 0.93 | 22 |
| 7 | 1.00 | 0.90 | 0.95 | 21 |
| 8 | 0.95 | 1.00 | 0.98 | 20 |
| 9 | 1.00 | 0.95 | 0.97 | 19 |
| 10 | 1.00 | 1.00 | 1.00 | 16 |
| 11 | 0.95 | 0.95 | 0.95 | 20 |
| 12 | 0.80 | 1.00 | 0.89 | 12 |
| 13 | 0.89 | 0.94 | 0.91 | 17 |
| 14 | 1.00 | 0.88 | 0.93 | 24 |
| 15 | 1.00 | 1.00 | 1.00 | 14 |
| 16 | 0.95 | 0.82 | 0.88 | 22 |
| 17 | 0.88 | 1.00 | 0.93 | 14 |
| 18 | 0.95 | 0.95 | 0.95 | 22 |
| 19 | 0.96 | 0.92 | 0.94 | 24 |
| 20 | 0.95 | 0.86 | 0.90 | 21 |
| 21 | 1.00 | 0.75 | 0.86 | 24 |
| 22 | 0.95 | 0.95 | 0.95 | 21 |
| 23 | 0.95 | 0.95 | 0.95 | 20 |
| 24 | 1.00 | 0.87 | 0.93 | 15 |
| 25 | 1.00 | 1.00 | 1.00 | 18 |
| 26 | 1.00 | 0.95 | 0.97 | 20 |
| 27 | 1.00 | 0.95 | 0.97 | 19 |
| 28 | 0.87 | 0.91 | 0.89 | 22 |

```
          29       0.78      1.00      0.88        14
          30       0.95      0.87      0.91        23
          31       0.91      1.00      0.95        21
          32       0.96      1.00      0.98        24
          33       0.79      1.00      0.88        15
          34       0.95      0.86      0.90        22
          35       0.82      1.00      0.90        23
          36       0.88      1.00      0.93        14
          37       0.81      0.87      0.84        15
          38       0.95      0.95      0.95        19

    accuracy                           0.93       723
   macro avg       0.94      0.94      0.94       723
weighted avg       0.94      0.93      0.94       723
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
fig, ax = plt.subplots(1, 1, figsize=(12,12))
score = 100*clf_MLP.score(Z_test, y_test)
title = 'Testing score ={:.2f}%'.format(score)
disp = ConfusionMatrixDisplay.from_estimator(
clf_MLP,
Z_test,
y_test,
xticks_rotation=45, #’vertical’,
# display_labels=class_names,
cmap=plt.cm.Blues,
normalize='true',
ax = ax
)
disp.ax_.set_title(title)
plt.show()
```




2.取 100 個主成分並使用 activation = 'logistic'且 hidden_layers = (30,)

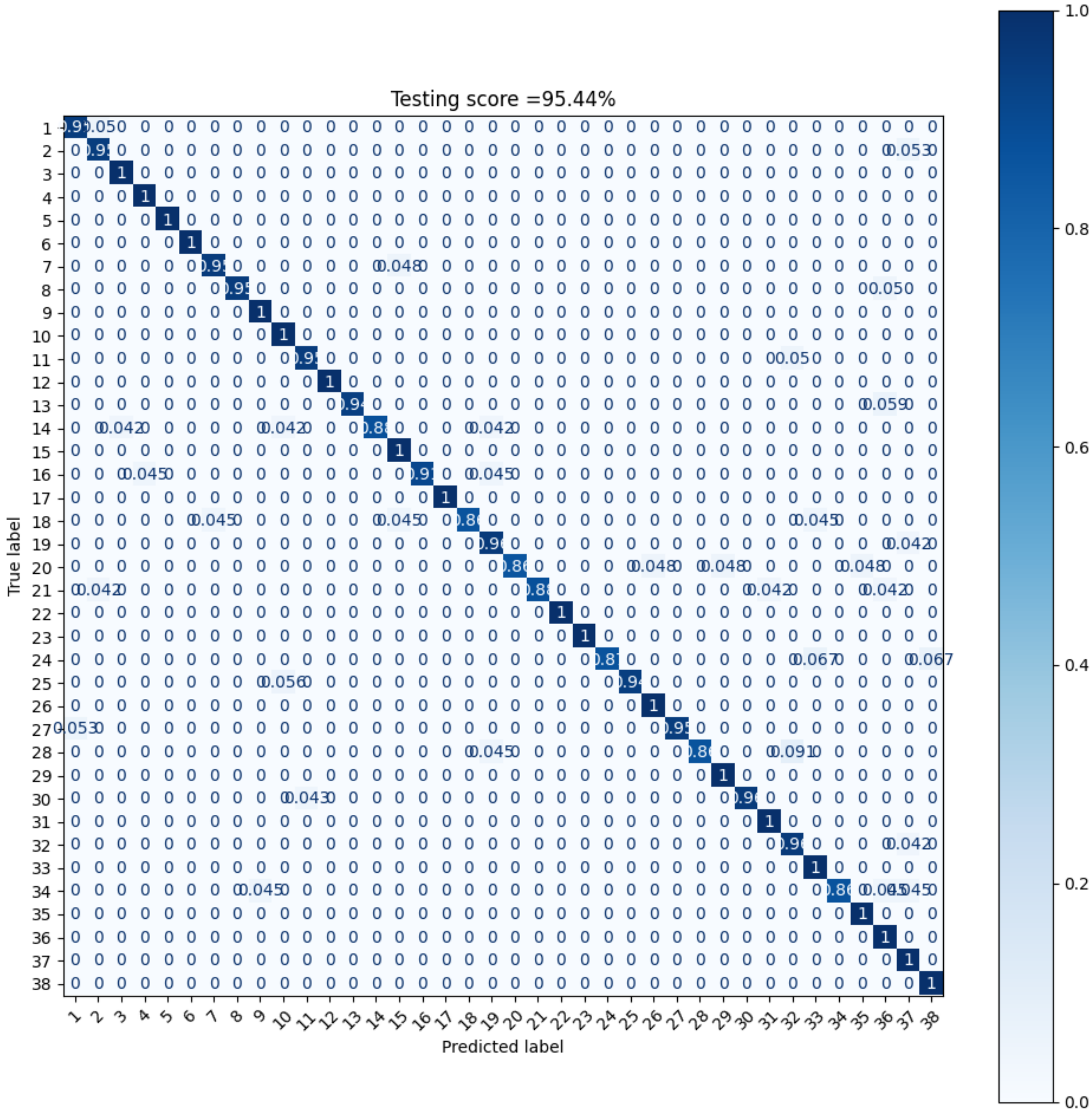
```
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier

pca = PCA(n_components = 100).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)

hidden_layers = (30,)
activation = 'logistic'
opts = dict(hidden_layer_sizes = hidden_layers , verbose = False, \
```

```python
activation = activation, tol = 1e-6, max_iter = int(1e6))
# solver = 'sgd' # not efficient, need more tuning
# solver = 'lbfgs' # not suitable here
solver = 'adam' # default solver
clf_MLP = MLPClassifier(solver = solver, **opts)
clf_MLP.fit(Z_train, y_train)
predictions = clf_MLP.predict(Z_test)
print(f"{clf_MLP.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
95.44%

              precision    recall  f1-score   support

           1       0.95      0.95      0.95        20
           2       0.90      0.95      0.92        19
           3       0.96      1.00      0.98        23
           4       0.93      1.00      0.97        14
           5       1.00      1.00      1.00        10
           6       1.00      1.00      1.00        22
           7       0.95      0.95      0.95        21
           8       1.00      0.95      0.97        20
           9       0.95      1.00      0.97        19
          10       0.89      1.00      0.94        16
          11       0.95      0.95      0.95        20
          12       1.00      1.00      1.00        12
          13       1.00      0.94      0.97        17
          14       1.00      0.88      0.93        24
          15       0.88      1.00      0.93        14
          16       1.00      0.91      0.95        22
          17       1.00      1.00      1.00        14
          18       1.00      0.86      0.93        22
          19       0.88      0.96      0.92        24
          20       1.00      0.86      0.92        21
          21       1.00      0.88      0.93        24
          22       1.00      1.00      1.00        21
          23       1.00      1.00      1.00        20
          24       1.00      0.87      0.93        15
          25       1.00      0.94      0.97        18
          26       0.95      1.00      0.98        20
          27       1.00      0.95      0.97        19
          28       1.00      0.86      0.93        22
          29       0.93      1.00      0.97        14
          30       1.00      0.96      0.98        23
          31       0.95      1.00      0.98        21
          32       0.88      0.96      0.92        24
          33       0.88      1.00      0.94        15
          34       1.00      0.86      0.93        22
          35       0.96      1.00      0.98        23
          36       0.78      1.00      0.88        14
```

```
          37       0.79      1.00      0.88        15
          38       0.95      1.00      0.97        19

    accuracy                           0.95       723
   macro avg       0.96      0.96      0.96       723
weighted avg       0.96      0.95      0.95       723
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
fig, ax = plt.subplots(1, 1, figsize=(12,12))
score = 100*clf_MLP.score(Z_test, y_test)
title = 'Testing score ={:.2f}%'.format(score)
disp = ConfusionMatrixDisplay.from_estimator(
clf_MLP,
Z_test,
y_test,
xticks_rotation=45, #’vertical’,
# display_labels=class_names,
cmap=plt.cm.Blues,
normalize='true',
ax = ax
)
disp.ax_.set_title(title)
plt.show()
```




3.取 500 個主成分並使用 activation = 'logistic'且 hidden_layers = (30,)

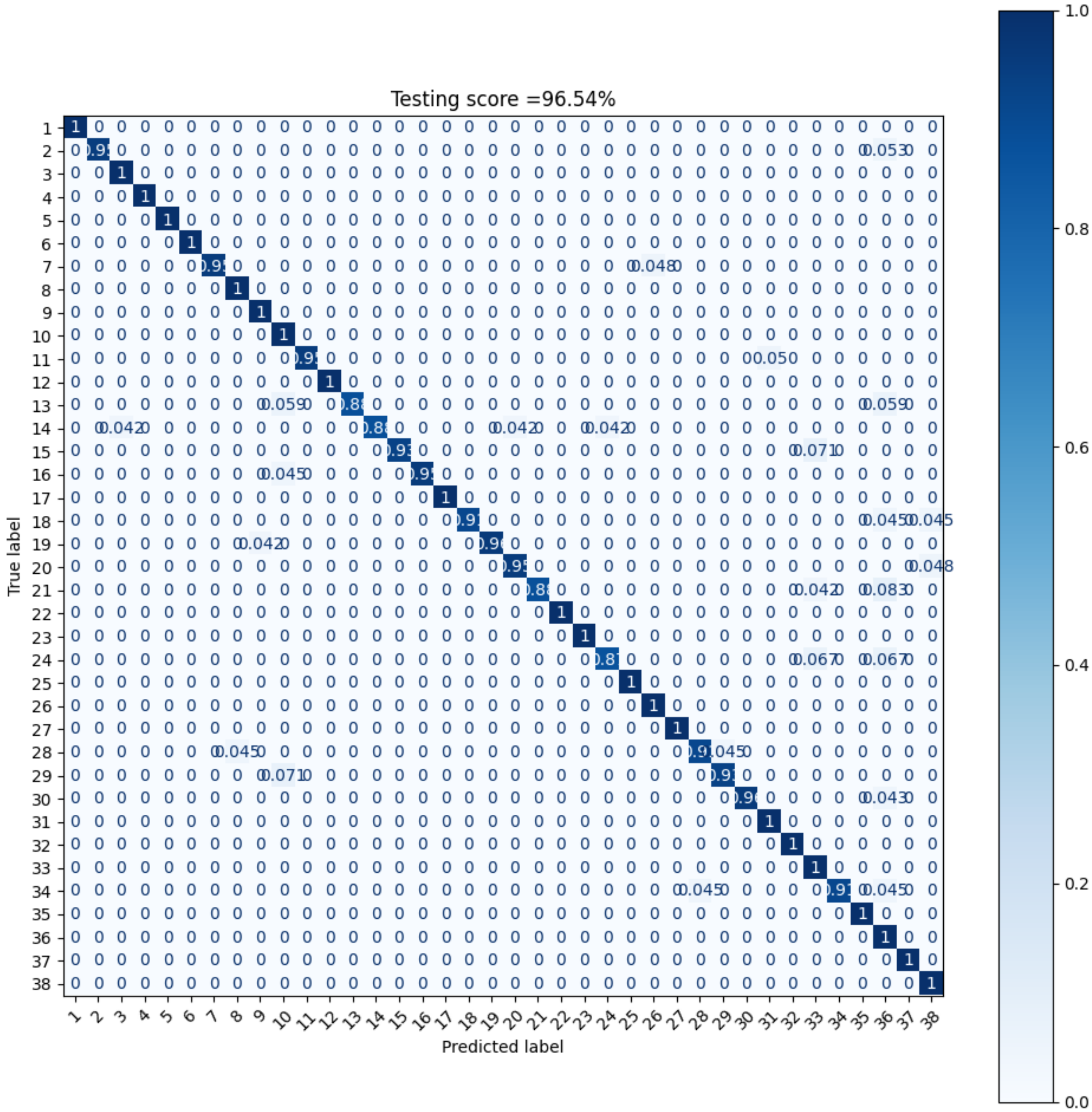
```
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier

pca = PCA(n_components = 500).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)

hidden_layers = (30,)
activation = 'logistic'
opts = dict(hidden_layer_sizes = hidden_layers , verbose = False, \
```

```
activation = activation, tol = 1e-6, max_iter = int(1e6))
# solver = 'sgd' # not efficient, need more tuning
# solver = 'lbfgs' # not suitable here
solver = 'adam' # default solver
clf_MLP = MLPClassifier(solver = solver, **opts)
clf_MLP.fit(Z_train, y_train)
predictions = clf_MLP.predict(Z_test)
print(f"{clf_MLP.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
96.54%

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        20
           2       1.00      0.95      0.97        19
           3       0.96      1.00      0.98        23
           4       1.00      1.00      1.00        14
           5       1.00      1.00      1.00        10
           6       1.00      1.00      1.00        22
           7       1.00      0.95      0.98        21
           8       0.95      1.00      0.98        20
           9       0.95      1.00      0.97        19
          10       0.84      1.00      0.91        16
          11       1.00      0.95      0.97        20
          12       1.00      1.00      1.00        12
          13       1.00      0.88      0.94        17
          14       1.00      0.88      0.93        24
          15       1.00      0.93      0.96        14
          16       1.00      0.95      0.98        22
          17       1.00      1.00      1.00        14
          18       1.00      0.91      0.95        22
          19       1.00      0.96      0.98        24
          20       0.95      0.95      0.95        21
          21       1.00      0.88      0.93        24
          22       1.00      1.00      1.00        21
          23       1.00      1.00      1.00        20
          24       0.93      0.87      0.90        15
          25       1.00      1.00      1.00        18
          26       0.95      1.00      0.98        20
          27       1.00      1.00      1.00        19
          28       0.95      0.91      0.93        22
          29       0.93      0.93      0.93        14
          30       1.00      0.96      0.98        23
          31       0.95      1.00      0.98        21
          32       1.00      1.00      1.00        24
          33       0.83      1.00      0.91        15
          34       1.00      0.91      0.95        22
          35       1.00      1.00      1.00        23
          36       0.64      1.00      0.78        14
```

```
          37       1.00      1.00      1.00        15
          38       0.90      1.00      0.95        19

    accuracy                           0.97       723
   macro avg       0.97      0.97      0.96       723
weighted avg       0.97      0.97      0.97       723
```

```python
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
fig, ax = plt.subplots(1, 1, figsize=(12,12))
score = 100*clf_MLP.score(Z_test, y_test)
title = 'Testing score ={:.2f}%'.format(score)
disp = ConfusionMatrixDisplay.from_estimator(
clf_MLP,
Z_test,
y_test,
xticks_rotation=45, #’vertical’,
# display_labels=class_names,
cmap=plt.cm.Blues,
normalize='true',
ax = ax
)
disp.ax_.set_title(title)
plt.show()
```




4.取 50 個主成分並使用使用 activation = 'relu'且 hidden_layers = (512,)

```
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier

pca = PCA(n_components = 50).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)

hidden_layers = (512,)
activation = 'relu'
opts = dict(hidden_layer_sizes = hidden_layers , verbose = False, \
```

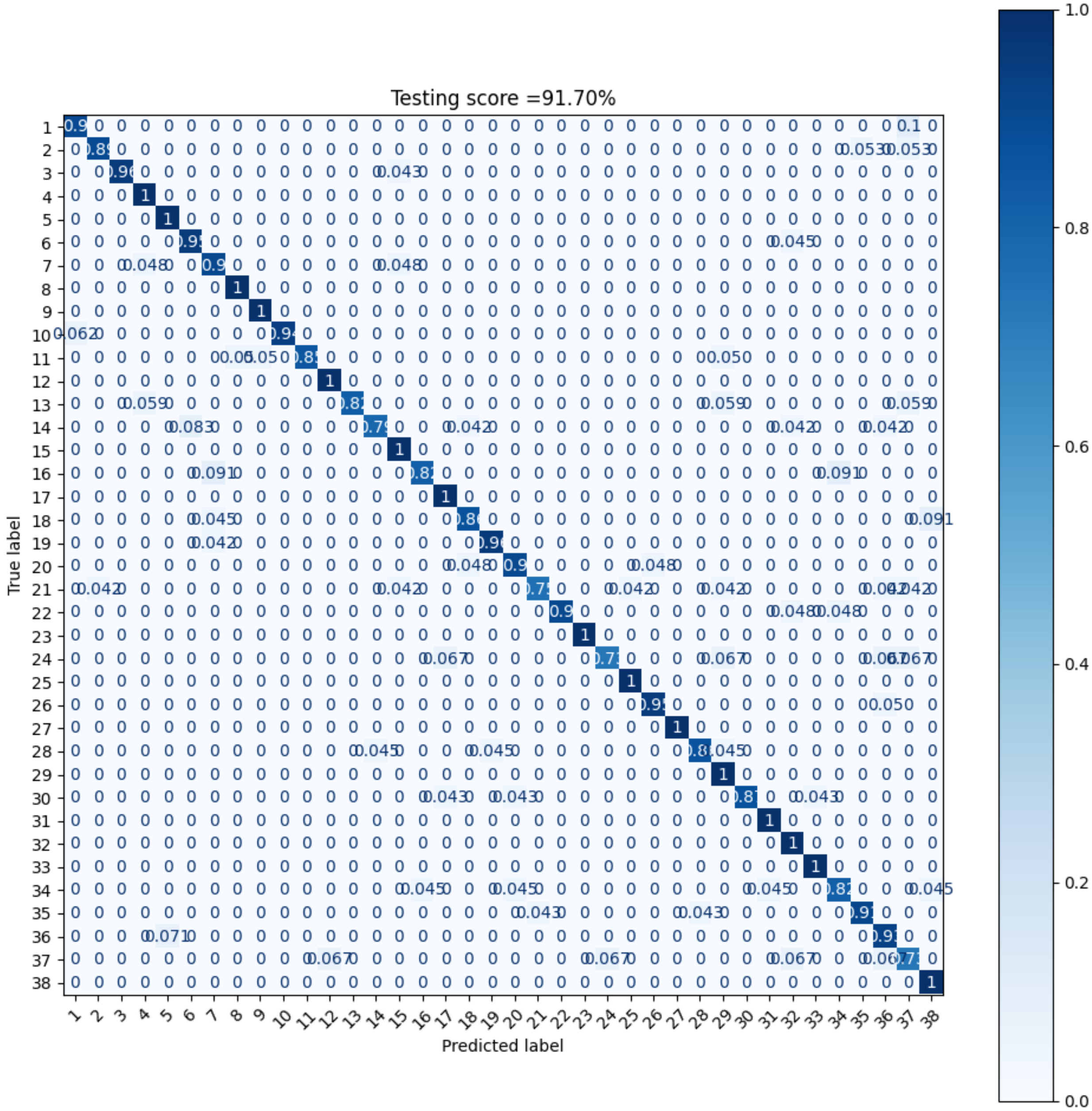
```
activation = activation, tol = 1e-6, max_iter = int(1e6))
# solver = 'sgd' # not efficient, need more tuning
# solver = 'lbfgs' # not suitable here
solver = 'adam' # default solver
clf_MLP = MLPClassifier(solver = solver, **opts)
clf_MLP.fit(Z_train, y_train)
predictions = clf_MLP.predict(Z_test)
print(f"{clf_MLP.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
91.70%
```

```
              precision    recall  f1-score   support

           1       0.95      0.90      0.92        20
           2       0.94      0.89      0.92        19
           3       1.00      0.96      0.98        23
           4       0.88      1.00      0.93        14
           5       0.91      1.00      0.95        10
           6       0.91      0.95      0.93        22
           7       0.83      0.90      0.86        21
           8       0.95      1.00      0.98        20
           9       0.95      1.00      0.97        19
          10       1.00      0.94      0.97        16
          11       1.00      0.85      0.92        20
          12       0.92      1.00      0.96        12
          13       1.00      0.82      0.90        17
          14       0.95      0.79      0.86        24
          15       0.82      1.00      0.90        14
          16       0.95      0.82      0.88        22
          17       0.88      1.00      0.93        14
          18       0.90      0.86      0.88        22
          19       0.96      0.96      0.96        24
          20       0.90      0.90      0.90        21
          21       0.95      0.75      0.84        24
          22       1.00      0.90      0.95        21
          23       1.00      1.00      1.00        20
          24       0.92      0.73      0.81        15
          25       0.95      1.00      0.97        18
          26       0.95      0.95      0.95        20
          27       1.00      1.00      1.00        19
          28       0.95      0.86      0.90        22
          29       0.74      1.00      0.85        14
          30       1.00      0.87      0.93        23
          31       0.95      1.00      0.98        21
          32       0.86      1.00      0.92        24
          33       0.94      1.00      0.97        15
          34       0.86      0.82      0.84        22
          35       0.95      0.91      0.93        23
          36       0.72      0.93      0.81        14
```

```
          37       0.65      0.73      0.69        15
          38       0.86      1.00      0.93        19

    accuracy                           0.92       723
   macro avg       0.92      0.92      0.92       723
weighted avg       0.92      0.92      0.92       723
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
fig, ax = plt.subplots(1, 1, figsize=(12,12))
score = 100*clf_MLP.score(Z_test, y_test)
title = 'Testing score ={:.2f}%'.format(score)
disp = ConfusionMatrixDisplay.from_estimator(
clf_MLP,
Z_test,
y_test,
xticks_rotation=45, #’vertical’,
# display_labels=class_names,
cmap=plt.cm.Blues,
normalize='true',
ax = ax
)
disp.ax_.set_title(title)
plt.show()
```




5.取 100 個主成分並使用使用 activation = 'relu'且 hidden_layers = (512,)

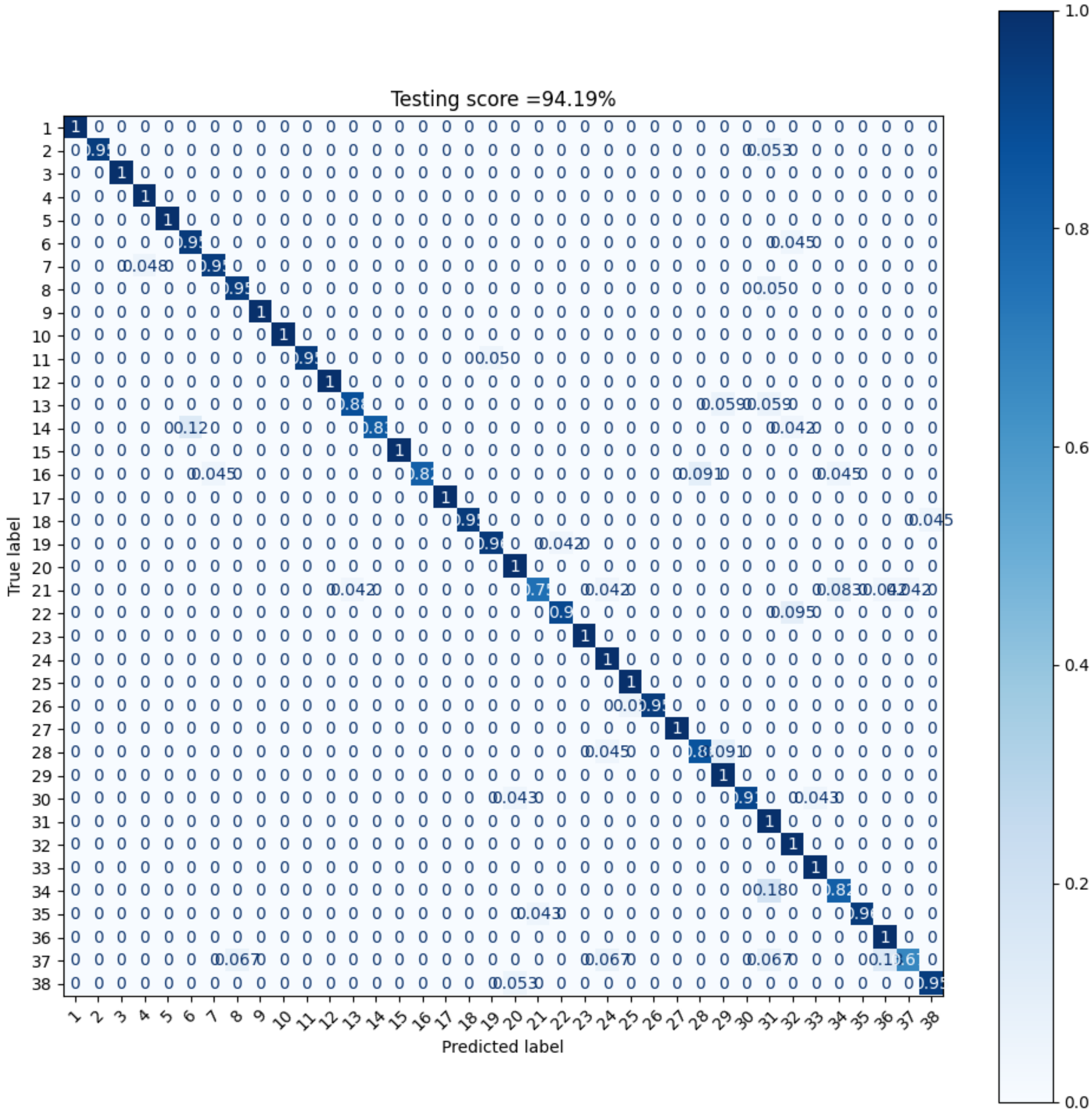
```
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier

pca = PCA(n_components = 100).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)

hidden_layers = (512,)
activation = 'relu'
opts = dict(hidden_layer_sizes = hidden_layers , verbose = False, \
```

```
activation = activation, tol = 1e-6, max_iter = int(1e6))
# solver = 'sgd' # not efficient, need more tuning
# solver = 'lbfgs' # not suitable here
solver = 'adam' # default solver
clf_MLP = MLPClassifier(solver = solver, **opts)
clf_MLP.fit(Z_train, y_train)
predictions = clf_MLP.predict(Z_test)
print(f"{clf_MLP.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
94.19%

              precision    recall  f1-score   support

           1       1.00      1.00      1.00        20
           2       1.00      0.95      0.97        19
           3       1.00      1.00      1.00        23
           4       0.93      1.00      0.97        14
           5       1.00      1.00      1.00        10
           6       0.88      0.95      0.91        22
           7       0.95      0.95      0.95        21
           8       0.95      0.95      0.95        20
           9       1.00      1.00      1.00        19
          10       1.00      1.00      1.00        16
          11       1.00      0.95      0.97        20
          12       1.00      1.00      1.00        12
          13       0.94      0.88      0.91        17
          14       1.00      0.83      0.91        24
          15       1.00      1.00      1.00        14
          16       1.00      0.82      0.90        22
          17       1.00      1.00      1.00        14
          18       1.00      0.95      0.98        22
          19       0.96      0.96      0.96        24
          20       0.91      1.00      0.95        21
          21       0.95      0.75      0.84        24
          22       0.95      0.90      0.93        21
          23       1.00      1.00      1.00        20
          24       0.83      1.00      0.91        15
          25       0.95      1.00      0.97        18
          26       1.00      0.95      0.97        20
          27       1.00      1.00      1.00        19
          28       0.90      0.86      0.88        22
          29       0.82      1.00      0.90        14
          30       1.00      0.91      0.95        23
          31       0.72      1.00      0.84        21
          32       0.86      1.00      0.92        24
          33       0.94      1.00      0.97        15
          34       0.86      0.82      0.84        22
          35       1.00      0.96      0.98        23
          36       0.82      1.00      0.90        14
```

```
          37       0.91      0.67      0.77        15
          38       0.95      0.95      0.95        19

    accuracy                           0.94       723
   macro avg       0.95      0.95      0.94       723
weighted avg       0.95      0.94      0.94       723
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
fig, ax = plt.subplots(1, 1, figsize=(12,12))
score = 100*clf_MLP.score(Z_test, y_test)
title = 'Testing score ={:.2f}%'.format(score)
disp = ConfusionMatrixDisplay.from_estimator(
clf_MLP,
Z_test,
y_test,
xticks_rotation=45, #’vertical’,
# display_labels=class_names,
cmap=plt.cm.Blues,
normalize='true',
ax = ax
)
disp.ax_.set_title(title)
plt.show()
```




6.取 500 個主成分並使用使用 activation = 'relu'且 hidden_layers = (512,)

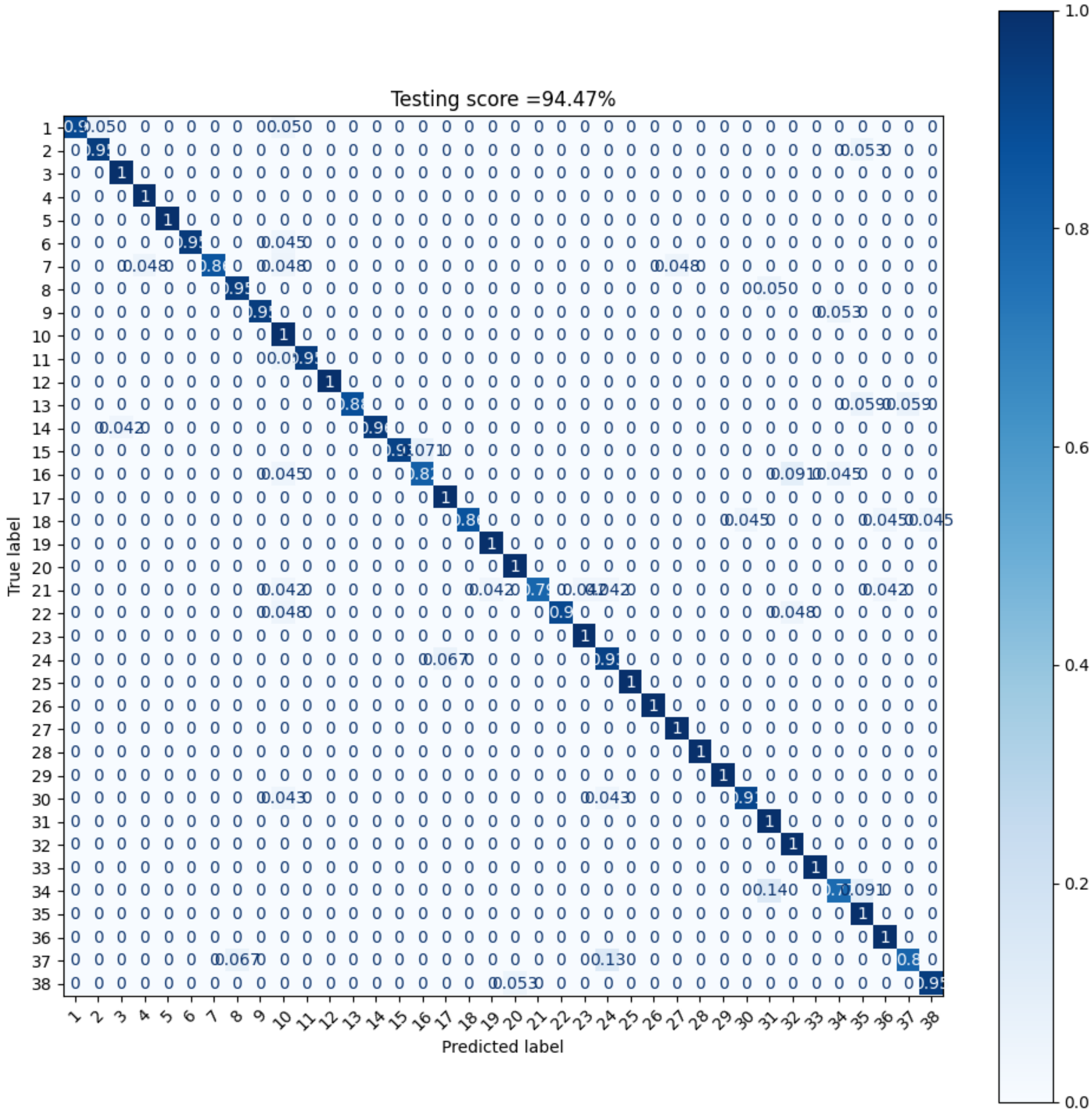
```
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier

pca = PCA(n_components = 500).fit(X_train_)
Z_train = pca.transform(X_train_)
Z_test = pca.transform(X_test_)

hidden_layers = (512,)
activation = 'relu'
opts = dict(hidden_layer_sizes = hidden_layers , verbose = False, \
```

```python
activation = activation, tol = 1e-6, max_iter = int(1e6))
# solver = 'sgd' # not efficient, need more tuning
# solver = 'lbfgs' # not suitable here
solver = 'adam' # default solver
clf_MLP = MLPClassifier(solver = solver, **opts)
clf_MLP.fit(Z_train, y_train)
predictions = clf_MLP.predict(Z_test)
print(f"{clf_MLP.score(Z_test, y_test):.2%}\n")
print(classification_report(y_test, predictions))
```

```
94.47%

              precision    recall  f1-score   support

           1       1.00      0.90      0.95        20
           2       0.95      0.95      0.95        19
           3       0.96      1.00      0.98        23
           4       0.93      1.00      0.97        14
           5       1.00      1.00      1.00        10
           6       1.00      0.95      0.98        22
           7       1.00      0.86      0.92        21
           8       0.95      0.95      0.95        20
           9       1.00      0.95      0.97        19
          10       0.67      1.00      0.80        16
          11       1.00      0.95      0.97        20
          12       1.00      1.00      1.00        12
          13       1.00      0.88      0.94        17
          14       1.00      0.96      0.98        24
          15       1.00      0.93      0.96        14
          16       0.95      0.82      0.88        22
          17       0.93      1.00      0.97        14
          18       1.00      0.86      0.93        22
          19       0.96      1.00      0.98        24
          20       0.95      1.00      0.98        21
          21       1.00      0.79      0.88        24
          22       1.00      0.90      0.95        21
          23       0.95      1.00      0.98        20
          24       0.78      0.93      0.85        15
          25       1.00      1.00      1.00        18
          26       1.00      1.00      1.00        20
          27       0.95      1.00      0.97        19
          28       1.00      1.00      1.00        22
          29       1.00      1.00      1.00        14
          30       0.95      0.91      0.93        23
          31       0.84      1.00      0.91        21
          32       0.89      1.00      0.94        24
          33       1.00      1.00      1.00        15
          34       0.89      0.77      0.83        22
          35       0.85      1.00      0.92        23
          36       0.88      1.00      0.93        14
```

```
          37       0.92      0.80      0.86        15
          38       0.95      0.95      0.95        19

    accuracy                           0.94       723
   macro avg       0.95      0.95      0.95       723
weighted avg       0.95      0.94      0.95       723
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
fig, ax = plt.subplots(1, 1, figsize=(12,12))
score = 100*clf_MLP.score(Z_test, y_test)
title = 'Testing score ={:.2f}%'.format(score)
disp = ConfusionMatrixDisplay.from_estimator(
clf_MLP,
Z_test,
y_test,
xticks_rotation=45, #’vertical’,
# display_labels=class_names,
cmap=plt.cm.Blues,
normalize='true',
ax = ax
)
disp.ax_.set_title(title)
plt.show()
```



討論：

使用 activation = 'logistic'且 hidden_layers = (30,)：

- 取 50 個主成分時，準確率為 93.50%。
- 取 100 個主成分時，準確率為 95.44%。
- 取 500 個主成分時，準確率為 96.54%。

使用 activation = 'relu'且 hidden_layers = (512,)：

- 取 50 個主成分時，準確率為 91.70%。
- 取 100 個主成分時，準確率為 94.19%。

- 取 500 個主成分時，準確率為 94.47%。

小結：

- 使用 activation = 'logistic'和 hidden_layers = (30,)時，準確率較高。
- 取愈多主成分，準確率愈高。

總結：

依照準確率比較：

- 多元羅吉斯回歸 (Multinomial Logistic Regression) (1)無論是何種演算法，原始資料的準確率較主成分資料的準確率高。 (2)在主成分資料中，無論是何種演算法，取愈多主成分，準確率愈高。 (3)在原始資料中，使用 liblinear 的演算法有最高的準確率 98.76%。

- 支援向量機 (Support Vector Machine) (1)大部分的 kernel，原始資料的準確率較主成分資料的準確率高，但當 kernel='poly'時卻相反。 (2)在主成分資料中，無論是何種 kernel，取愈多主成分，準確率愈高。 (3)取 500 個主成分且使用 kernel='poly'有最高的準確率 94.05%。

- 神經網路 (Neural Network) (1)無論是使用 activation = 'logistic'且 hidden_layers = (30,)或使用 activation = 'relu'且 hidden_layers = (512,)，取 50 個主成分時，原始資料的準確率較主成分資料的準確率高；但取 100 或 500 個主成分時，主成分資料的準確率較原始資料的準確率高。 (2)在主成分資料中，無論是何種 activation 和 hidden_layers，取愈多主成分，準確率愈高。 (3)取 500 個主成分並使用 activation = 'logistic'和 hidden_layers = (30,)有最高的準確率 96.54%。

依照執行時間比較：

- (1)無論是何種分類器，主成分資料的執行時間都較原始資料短。
- (2)無論是何種分類器，取愈多的主成分，執行時間會愈長。
- (3)原始資料中，多元羅吉斯回歸的執行時間較長(其中又以 newton-cg 的演算法最長)，支援向量機和神經網路差異不大。
- (4)主成分資料中，支援向量機的執行時間較短，多元羅吉斯回歸的執行時間較長(其中又以 newton-cg 的演算法最長)。

綜上所述：

- 我認為最佳分類器為神經網路中取 500 個主成分並使用 activation = 'logistic'和 hidden_layers = (30,)，因為它的準確率為所有分類器中第二高(96.54%)，且執行時間大約 1 分鐘，有很好的效率。
- 雖然多元羅吉斯回歸的原始資料中，使用 liblinear 的演算法會有最高的準確率 98.76%，但執行時間過長(約 92 分鐘)，效率不彰。