Mnist Classification models

December 15, 2021

[1]:
```
import time
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml


#Metrics to evaluate models
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import multilabel_confusion_matrix

#Confusion matrix graphs
from sklearn.metrics import ConfusionMatrixDisplay

#Classifiers
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier

#Tools for PCA
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

#Cluster analysis
from sklearn.cluster import KMeans
```

[2]:
```
#Download Mnist dataset via fetch_openml from sklearn
mnist = fetch_openml('mnist_784', data_home='datasets/mnist')
```

[3]:
```
#Check dataset
mnist.data.shape
```

[3]:
```
(70000, 784)
```

[4]:
```
#Create array with labels and features
Z = np.c_[mnist.target, mnist.data.astype(int)]
#Sorted array
```

```
Z_sorted = sorted(Z, key=lambda z: z[0])

#Checked if array had been sorted
#Z_sorted
```

[5]:
```
#Create X and Y from data and labels
X = Z[:,1:]
Y = Z[:,0]
#Ensure shuffling of array is the same every time
shuffle_index = np.random.RandomState(seed=19).permutation(70000)
#Shuffle X and Y
X, Y = X[shuffle_index], Y[shuffle_index]
X
```

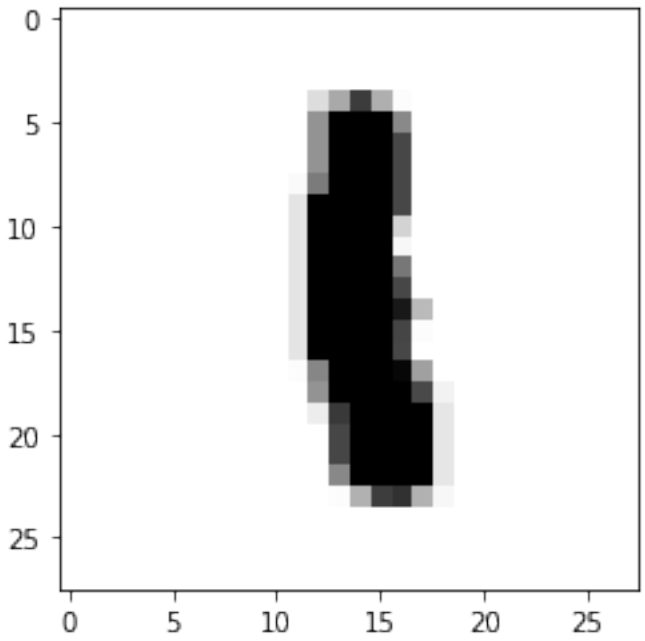
[5]:
```
array([[0, 0, 0, …, 0, 0, 0],
       [0, 0, 0, …, 0, 0, 0],
       [0, 0, 0, …, 0, 0, 0],
       …,
       [0, 0, 0, …, 0, 0, 0],
       [0, 0, 0, …, 0, 0, 0],
       [0, 0, 0, …, 0, 0, 0]], dtype=object)
```

[6]:
```
Y
```

[6]:
```
array(['6', '7', '4', …, '0', '0', '5'], dtype=object)
```

[7]:
```
some_digit_index = 1997
X = X.astype(int)
print(Y[some_digit_index])
img = plt.imshow(X[some_digit_index].reshape(28, 28), cmap='gray_r')
```

```
1
```



[8]:
```
#Training set
X_train = X[0:50000]
Y_train = Y[0:50000]

#Validation set
X_val = X[50000:60000]
Y_val = Y[50000:60000]

#Test set
X_test = X[60000:70000]
Y_test = Y[60000:70000]
```

[9]:
```
#Training set has 50000 images of digits and 784 features
X_train.shape
```

[9]:
```
(50000, 784)
```

[10]:
```
#Training random forest classifier
rdf = RandomForestClassifier(n_estimators=50, n_jobs=2, random_state=19)
t0 = time.time()
rdf.fit(X_train, Y_train)
t1 = time.time()
pred_train = rdf.predict(X_train[0:10000])
t2 = time.time()
```

```
print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
 ↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 10.216
Prediction time for train: 0.135
Youdens index for train: 1.000
```

[11]:
```
t3 = time.time()
pred_val = rdf.predict(X_val)
t4 = time.time()
print(f'Prediction time for validation: {t4-t3:3.3f}')
print(f'Youdens index for validation: {((metrics.balanced_accuracy_score(Y_val,␣
 ↪pred_val)*2)-1):.3f}')
```

```
Prediction time for validation: 0.152
Youdens index for validation: 0.933
```

[12]:
```
t5 = time.time()
pred_test = rdf.predict(X_test)
t6 = time.time()
print(f'Prediction time for test: {t6-t5:3.3f}')
print(f'Youdens index for test: {((metrics.balanced_accuracy_score(Y_test,␣
 ↪pred_test)*2)-1):.3f}')
```

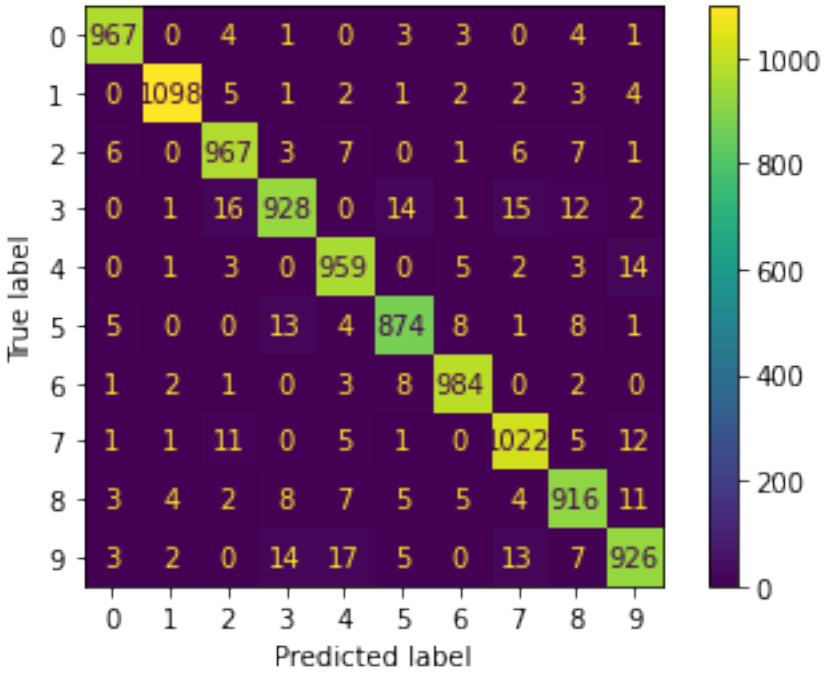
```
Prediction time for test: 0.154
Youdens index for test: 0.927
```

[13]:
```
#Random forest confusion matrix

cm = confusion_matrix(Y_test, pred_test)
cm
```

[13]:
```
array([[ 967,    0,    4,    1,    0,    3,    3,    0,    4,    1],
       [   0, 1098,    5,    1,    2,    1,    2,    2,    3,    4],
       [   6,    0,  967,    3,    7,    0,    1,    6,    7,    1],
       [   0,    1,   16,  928,    0,   14,    1,   15,   12,    2],
       [   0,    1,    3,    0,  959,    0,    5,    2,    3,   14],
       [   5,    0,    0,   13,    4,  874,    8,    1,    8,    1],
       [   1,    2,    1,    0,    3,    8,  984,    0,    2,    0],
       [   1,    1,   11,    0,    5,    1,    0, 1022,    5,   12],
       [   3,    4,    2,    8,    7,    5,    5,    4,  916,   11],
       [   3,    2,    0,   14,   17,    5,    0,   13,    7,  926]])
```

[14]:
```
#Create graph of Random forest confusion matrix

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rdf.classes_)
```

```
disp.plot()

plt.show()
```



[15]:
```
#Training naive bayes classifier

nb = GaussianNB()
t0 = time.time()
nb.fit(X_train, Y_train)
t1 = time.time()
pred_train = nb.predict(X_train[0:10000])
t2 = time.time()

print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
 ↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 0.738
Prediction time for train: 0.552
Youdens index for train: 0.093
```

[16]:
```
nb = GaussianNB(var_smoothing=0.086)
t0 = time.time()
nb.fit(X_train, Y_train)
```

```
t1 = time.time()
pred_train = nb.predict(X_train[0:10000])
t2 = time.time()

print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
 ↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 0.784
Prediction time for train: 0.561
Youdens index for train: 0.595
```

[17]:
```
t3 = time.time()
pred_val = nb.predict(X_val)
t4 = time.time()
print(f'Prediction time for validation: {t4-t3:3.3f}')
print(f'Youdens index for validation: {((metrics.balanced_accuracy_score(Y_val,␣
 ↪pred_val)*2)-1):.3f}')
```

```
Prediction time for validation: 0.548
Youdens index for validation: 0.593
```

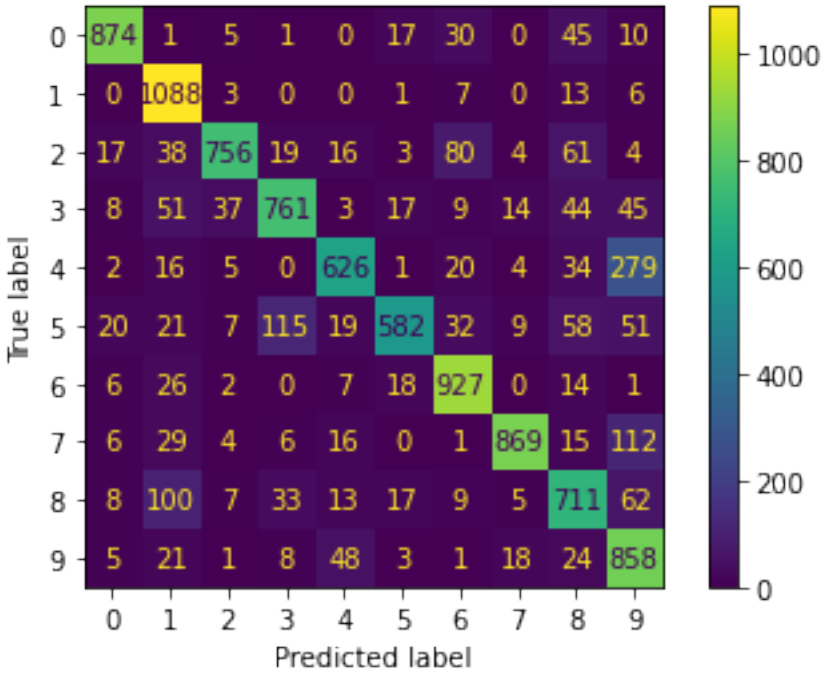
[18]:
```
t5 = time.time()
pred_test = nb.predict(X_test)
t6 = time.time()
print(f'Prediction time for test: {t6-t5:3.3f}')
print(f'Youdens index for test: {((metrics.balanced_accuracy_score(Y_test,␣
 ↪pred_test)*2)-1):.3f}')
```

```
Prediction time for test: 0.566
Youdens index for test: 0.603
```

[19]:
```
#Naive bayes confusion matrix

cm = confusion_matrix(Y_test, pred_test)
cm
```

[19]:
```
array([[ 874,    1,    5,    1,    0,   17,   30,    0,   45,   10],
       [   0, 1088,    3,    0,    0,    1,    7,    0,   13,    6],
       [  17,   38,  756,   19,   16,    3,   80,    4,   61,    4],
       [   8,   51,   37,  761,    3,   17,    9,   14,   44,   45],
       [   2,   16,    5,    0,  626,    1,   20,    4,   34,  279],
       [  20,   21,    7,  115,   19,  582,   32,    9,   58,   51],
       [   6,   26,    2,    0,    7,   18,  927,    0,   14,    1],
       [   6,   29,    4,    6,   16,    0,    1,  869,   15,  112],
       [   8,  100,    7,   33,   13,   17,    9,    5,  711,   62],
       [   5,   21,    1,    8,   48,    3,    1,   18,   24,  858]])
```

[20]:
```
#Create graph of Naive bayes confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=nb.classes_)

disp.plot()

plt.show()
```



[21]:
```
#Training KNeighborsClassifier

knc = KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='brute')
t0 = time.time()
knc.fit(X_train, Y_train)
t1 = time.time()
pred_train = knc.predict(X_train[0:10000])
t2 = time.time()

print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
 ↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 0.064
Prediction time for train: 16.929
Youdens index for train: 0.962
```

[22]:
```
t3 = time.time()
pred_val = knc.predict(X_val)
t4 = time.time()
print(f'Prediction time for validation: {t4-t3:3.3f}')
print(f'Youdens index for validation: {((metrics.balanced_accuracy_score(Y_val,␣
 ↪pred_val)*2)-1):.3f}')
```

```
Prediction time for validation: 16.747
Youdens index for validation: 0.942
```

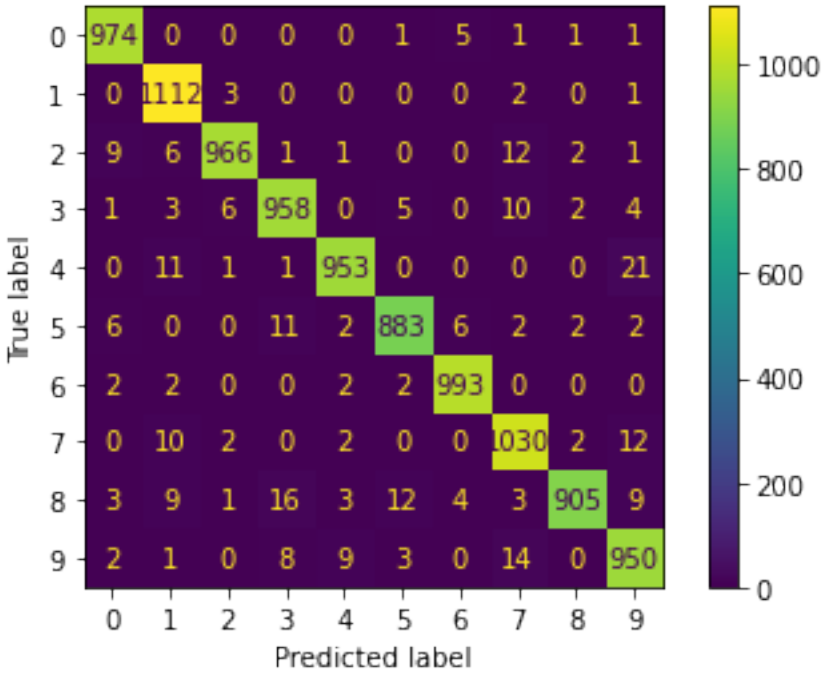
[23]:
```
t5 = time.time()
pred_test = knc.predict(X_test)
t6 = time.time()
print(f'Prediction time for test: {t6-t5:3.3f}')
print(f'Youdens index for test: {((metrics.balanced_accuracy_score(Y_test,␣
 ↪pred_test)*2)-1):.3f}')
```

```
Prediction time for test: 16.031
Youdens index for test: 0.944
```

[24]:
```
#KNeighborsClassifier confusion matrix

cm = confusion_matrix(Y_test, pred_test)
cm
```

[24]:
```
array([[ 974,    0,    0,    0,    0,    1,    5,    1,    1,    1],
       [   0, 1112,    3,    0,    0,    0,    0,    2,    0,    1],
       [   9,    6,  966,    1,    1,    0,    0,   12,    2,    1],
       [   1,    3,    6,  958,    0,    5,    0,   10,    2,    4],
       [   0,   11,    1,    1,  953,    0,    0,    0,    0,   21],
       [   6,    0,    0,   11,    2,  883,    6,    2,    2,    2],
       [   2,    2,    0,    0,    2,    2,  993,    0,    0,    0],
       [   0,   10,    2,    0,    2,    0,    0, 1030,    2,   12],
       [   3,    9,    1,   16,    3,   12,    4,    3,  905,    9],
       [   2,    1,    0,    8,    9,    3,    0,   14,    0,  950]])
```

[25]:
```
#Create graph of KNeighborsClassifier confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=knc.classes_)

disp.plot()

plt.show()
```



[26]:
```
#Training MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(75, 50), random_state=19)
t0 = time.time()
mlp.fit(X_train, Y_train)
t1 = time.time()
pred_train = mlp.predict(X_train[0:10000])
t2 = time.time()

print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 70.950
Prediction time for train: 0.070
Youdens index for train: 0.989
```

[27]:
```
t3 = time.time()
pred_val = mlp.predict(X_val)
t4 = time.time()
print(f'Prediction time for validation: {t4-t3:3.3f}')
print(f'Youdens index for validation: {((metrics.balanced_accuracy_score(Y_val,␣
↪pred_val)*2)-1):.3f}')
```

```
Prediction time for validation: 0.061
```

```
Youdens index for validation: 0.923
```

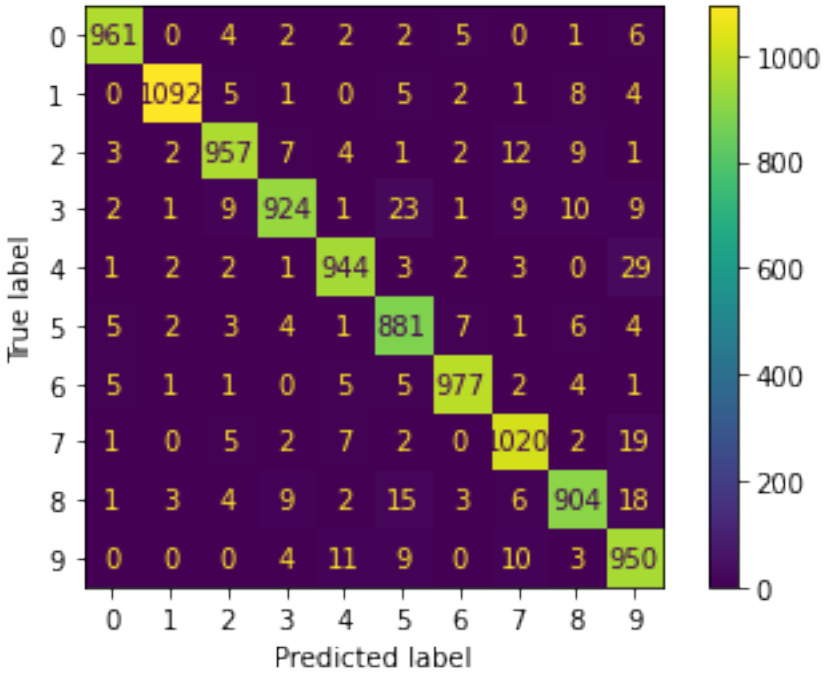
[28]:
```
t5 = time.time()
pred_test = mlp.predict(X_test)
t6 = time.time()
print(f'Prediction time for test: {t6-t5:3.3f}')
print(f'Youdens index for test: {((metrics.balanced_accuracy_score(Y_test,␣
 ↪pred_test)*2)-1):.3f}')
```

```
Prediction time for test: 0.075
Youdens index for test: 0.921
```

[29]:
```
#MLPClassifier confusion matrix

cm = confusion_matrix(Y_test, pred_test)
cm
```

[29]:
```
array([[ 961,    0,    4,    2,    2,    2,    5,    0,    1,    6],
       [   0, 1092,    5,    1,    0,    5,    2,    1,    8,    4],
       [   3,    2,  957,    7,    4,    1,    2,   12,    9,    1],
       [   2,    1,    9,  924,    1,   23,    1,    9,   10,    9],
       [   1,    2,    2,    1,  944,    3,    2,    3,    0,   29],
       [   5,    2,    3,    4,    1,  881,    7,    1,    6,    4],
       [   5,    1,    1,    0,    5,    5,  977,    2,    4,    1],
       [   1,    0,    5,    2,    7,    2,    0, 1020,    2,   19],
       [   1,    3,    4,    9,    2,   15,    3,    6,  904,   18],
       [   0,    0,    0,    4,   11,    9,    0,   10,    3,  950]])
```

[30]:
```
#Create graph of MLPClassifier confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=mlp.classes_)

disp.plot()

plt.show()
```



[31]:
```
#Cluster analysis

t0 = time.time()
kmeans = KMeans(n_clusters=10, n_init=10, random_state=19, max_iter=300,␣
↪algorithm='auto').fit(X_train, Y_train)
t1 = time.time()
pred_train = kmeans.predict(X_train[0:10000])
t2 = time.time()
pred_train = pred_train.astype(str)

print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 33.175
Prediction time for train: 0.128
Youdens index for train: -0.713
```

[32]:
```
t3 = time.time()
pred_val = kmeans.predict(X_val)
t4 = time.time()

pred_val = pred_val.astype(str)
```

```
print(f'Prediction time for validation: {t4-t3:3.3f}')
print(f'Youdens index for validation: {((metrics.balanced_accuracy_score(Y_val,␣
 ↪pred_val)*2)-1):.3f}')
```

```
Prediction time for validation: 0.060
Youdens index for validation: -0.726
```

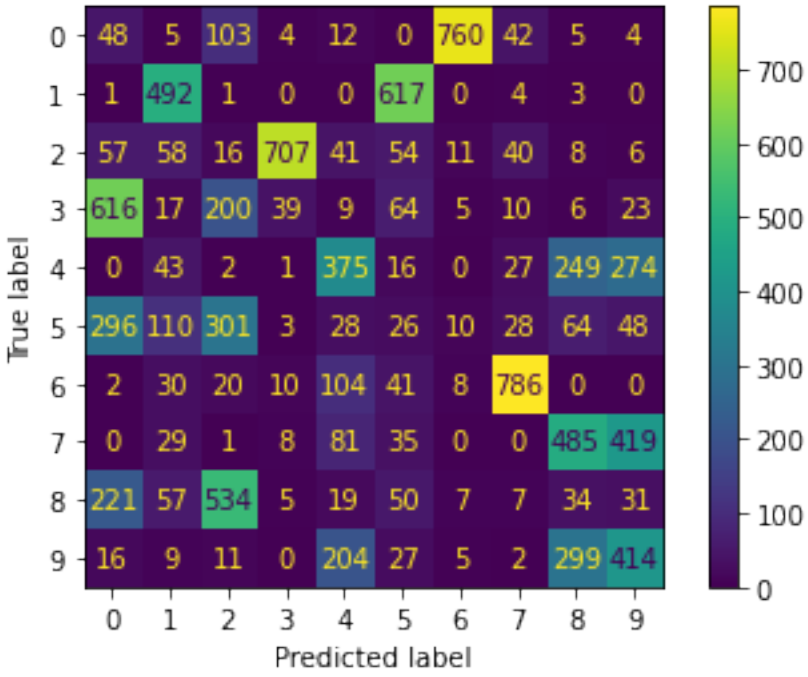
[33]:
```
t5 = time.time()
pred_test = kmeans.predict(X_test)
t6 = time.time()

pred_test = pred_test.astype(str)

print(f'Prediction time for test: {t6-t5:3.3f}')
print(f'Youdens index for test: {((metrics.balanced_accuracy_score(Y_test,␣
 ↪pred_test)*2)-1):.3f}')
```

```
Prediction time for test: 0.064
Youdens index for test: -0.717
```

[34]:
```
#Cluster analysis confusion matrix

cm = confusion_matrix(Y_test, pred_test)
cm
```

[34]:
```
array([[ 48,   5, 103,   4,  12,   0, 760,  42,   5,   4],
       [  1, 492,   1,   0,   0, 617,   0,   4,   3,   0],
       [ 57,  58,  16, 707,  41,  54,  11,  40,   8,   6],
       [616,  17, 200,  39,   9,  64,   5,  10,   6,  23],
       [  0,  43,   2,   1, 375,  16,   0,  27, 249, 274],
       [296, 110, 301,   3,  28,  26,  10,  28,  64,  48],
       [  2,  30,  20,  10, 104,  41,   8, 786,   0,   0],
       [  0,  29,   1,   8,  81,  35,   0,   0, 485, 419],
       [221,  57, 534,   5,  19,  50,   7,   7,  34,  31],
       [ 16,   9,  11,   0, 204,  27,   5,   2, 299, 414]])
```

[35]:
```
#Create graph of Cluster analysis confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)

disp.plot()

plt.show()
```



[36]:
```
#Prepare data for Principal component analysis

scaler = StandardScaler()

scaler.fit(X_train)

#Standardize train, val and test set
X_train = scaler.transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

[37]:
```
X_train[30000]
```

[37]:
```
array([ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
       -4.83977140e-03, -6.30410991e-03, -4.47218068e-03, -4.47218068e-03,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00, -5.29758889e-03, -9.12048405e-03, -1.23219199e-02,
       -1.66672826e-02, -2.11992688e-02, -2.47321873e-02, -2.96322779e-02,
       -3.20377898e-02, -3.25333942e-02, -3.52804833e-02, -3.22301660e-02,
```

```
-2.92673636e-02, -2.58725604e-02, -2.10494713e-02, -1.81656897e-02,
-1.60250518e-02, -9.80577533e-03, -7.61265508e-03, -4.79881857e-03,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  0.00000000e+00, -4.47218068e-03, -5.91032260e-03,
-7.58913573e-03, -1.30014478e-02, -1.86788399e-02, -2.77996324e-02,
-3.68610915e-02, -5.30189490e-02, -6.92364286e-02, -8.73292474e-02,
-1.03084461e-01, -1.17384328e-01, -1.28969433e-01, -1.37817911e-01,
-1.37640410e-01, -1.30166664e-01, -1.17634678e-01, -1.00618728e-01,
-7.80637686e-02, -5.42582789e-02, -3.74020408e-02, -2.33339389e-02,
-1.43430961e-02, -7.38431450e-03,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  0.00000000e+00, -4.47218068e-03, -8.36546550e-03,
-1.45423956e-02, -2.37254539e-02, -3.53889840e-02, -5.44689475e-02,
-7.94745808e-02, -1.11321867e-01, -1.41780875e-01, -1.72206942e-01,
-2.00405114e-01, -2.26696769e-01, -2.47797805e-01, -2.60052416e-01,
-2.60189410e-01, -2.45952082e-01, -2.20521935e-01, -1.85725394e-01,
-1.46907347e-01, -1.06747118e-01, -7.70518991e-02, -4.82576681e-02,
-3.02894152e-02, -1.62994073e-02, -4.47218068e-03,  0.00000000e+00,
 0.00000000e+00, -4.47218068e-03, -8.38080523e-03, -1.46275502e-02,
-2.44139811e-02, -4.76310048e-02, -7.79532627e-02, -1.16423497e-01,
-1.62672771e-01, -2.16351120e-01, -2.72341164e-01, -3.29997311e-01,
-3.93138621e-01, -4.52910624e-01, -4.99720604e-01, -5.23225596e-01,
-5.16598551e-01, -4.79649517e-01, -4.21192529e-01, -3.53457023e-01,
-2.80886407e-01, -2.14898271e-01, -1.57527147e-01, -1.11401939e-01,
-7.15733210e-02, -3.97588748e-02, -1.63743043e-02, -6.32128497e-03,
 0.00000000e+00, -4.47218068e-03, -1.29445531e-02, -2.19578141e-02,
-5.29135338e-02, -9.05856547e-02, -1.39610564e-01, -1.96595488e-01,
-2.65402273e-01, -3.43845685e-01, -4.27139165e-01, -5.20037047e-01,
-6.18110975e-01, -7.15373472e-01, -7.93677507e-01, -8.32911025e-01,
-8.23518387e-01, -7.62191543e-01, -6.64936033e-01, -5.54155378e-01,
-4.45226801e-01, -3.45418110e-01, -2.60930947e-01, -1.90506606e-01,
-1.31502848e-01, -7.46750989e-02, -3.23698070e-02, -7.86123196e-03,
 0.00000000e+00, -6.10181575e-03, -1.45597882e-02, -3.63673180e-02,
-8.07362131e-02, -1.33737497e-01, -1.98183859e-01, -2.76115602e-01,
-3.65300455e-01, -4.66674074e-01, -5.78700142e-01,  1.74750306e+00,
 1.52478912e+00,  1.32758832e+00,  1.20427290e+00,  1.16136020e+00,
 1.17816837e+00,  1.26790973e+00,  1.43860054e+00,  1.30126077e+00,
 3.77076709e-01, -4.51942888e-01, -3.38557689e-01, -2.47276458e-01,
-1.72813261e-01, -1.03543307e-01, -4.68036695e-02, -1.41042845e-02,
-4.47218068e-03, -1.22896784e-02, -2.63801797e-02, -6.04575407e-02,
-1.14846648e-01, -1.79384959e-01, -2.57457842e-01, -3.51301204e-01,
-2.84172887e-01,  5.35444557e-01,  1.56604537e+00,  1.47237693e+00,
 1.27901318e+00,  1.15538614e+00,  1.08677002e+00,  1.06688768e+00,
 1.08448373e+00,  1.14085537e+00,  1.25450442e+00,  1.46069833e+00,
 1.75963973e+00,  8.56115569e-01, -3.95769523e-01, -2.86108767e-01,
-1.95528109e-01, -1.20853499e-01, -5.58650249e-02, -1.50020229e-02,
-4.47218068e-03, -1.81217881e-02, -4.28395104e-02, -8.48394956e-02,
-1.39486728e-01, -2.10614792e-01, -2.98940444e-01, -4.08115898e-01,
```

```
 3.24811846e-01,  1.83170504e+00,  1.50437649e+00,  6.29635280e-01,
 5.12383808e-01,  4.80499384e-01, -4.12805998e-01, -7.45361453e-01,
-3.58231896e-02, -1.02770938e-01,  4.88635083e-01,  1.28454357e+00,
 1.67843365e+00,  1.72585921e+00, -4.21108004e-01, -2.96552496e-01,
-1.96777387e-01, -1.19691414e-01, -5.56155818e-02, -1.33532844e-02,
-5.15304043e-03, -2.16676325e-02, -5.15375204e-02, -9.26389563e-02,
-1.48835647e-01, -2.24518113e-01, -3.21346527e-01, -4.42423160e-01,
 1.24256165e+00,  1.65597707e+00,  1.38713620e+00, -4.59904346e-01,
-9.41250700e-01, -9.43923760e-01, -9.05945701e-01, -9.14808437e-01,
-9.54093617e-01, -1.00408342e+00, -1.00248882e+00,  1.09570804e+00,
 1.68333996e+00,  1.72432592e+00, -4.17737827e-01, -2.87984941e-01,
-1.80157082e-01, -1.05806060e-01, -4.86635004e-02, -1.60129028e-02,
-4.49303008e-03, -2.18008833e-02, -5.15341707e-02, -9.10919834e-02,
-1.45026288e-01, -2.24611751e-01, -3.30972462e-01, -4.63639925e-01,
 1.89694139e-01,  1.57564491e+00,  1.36906182e+00,  1.35171941e+00,
 3.59756934e-01, -5.40705402e-01, -7.69467002e-01, -8.00712236e-01,
-8.72240701e-01, -9.54049504e-01, -3.22313181e-01,  1.36168945e+00,
 1.76785933e+00,  1.82551216e+00, -3.95191439e-01, -2.70395442e-01,
-1.59572757e-01, -8.18643340e-02, -3.91186976e-02, -1.47227059e-02,
-6.78065629e-03, -2.03304129e-02, -4.62644155e-02, -8.04089635e-02,
-1.35309938e-01, -2.23624329e-01, -3.38805259e-01, -4.86065366e-01,
-3.25287908e-01,  1.47300128e+00,  1.39005356e+00,  1.46123701e+00,
 1.66458935e+00,  1.52975205e+00,  4.92014465e-01, -6.78255743e-01,
-9.01474300e-01, -9.81372078e-01, -1.80062558e-02,  1.51637004e+00,
 1.89923729e+00,  1.01564261e+00, -3.68353883e-01, -2.57828561e-01,
-1.50777853e-01, -6.17713559e-02, -2.83811209e-02, -1.02956665e-02,
-4.47218068e-03, -1.63490658e-02, -3.63845939e-02, -6.74571917e-02,
-1.26032243e-01, -2.25938000e-01, -3.54924248e-01, -5.12008661e-01,
-6.87281815e-01, -1.56763867e-01,  8.35371054e-01,  1.50913001e+00,
 1.68098740e+00,  1.68976961e+00,  1.50164951e+00,  1.24719967e+00,
 1.36159702e-01, -1.06078431e+00, -3.49463738e-02,  1.58324766e+00,
 2.04644833e+00,  4.51188110e-01, -3.54555324e-01, -2.55994797e-01,
-1.55881207e-01, -5.48052910e-02, -2.03250218e-02, -7.89439898e-03,
-4.47218068e-03, -1.13284096e-02, -2.55646035e-02, -5.70350512e-02,
-1.24101979e-01, -2.39223786e-01, -3.76485243e-01, -5.37388082e-01,
-7.07494115e-01, -8.39615585e-01, -8.51326066e-01, -2.26768455e-01,
 5.32594925e-01,  1.25049541e+00,  1.24140721e+00,  1.15242915e+00,
 1.11611903e+00,  1.07775336e+00,  1.26309736e+00,  1.64509632e+00,
 1.47548094e+00, -4.35729692e-01, -3.56327775e-01, -2.63793855e-01,
-1.66276489e-01, -5.82192530e-02, -1.90230487e-02, -6.15835948e-03,
-4.47218068e-03, -5.05562156e-03, -1.86522611e-02, -5.15211186e-02,
-1.29252516e-01, -2.59536382e-01, -3.97324777e-01, -5.53881234e-01,
-7.11166211e-01, -8.27825416e-01, -8.65089011e-01, -8.58704692e-01,
-9.02494087e-01, -5.07860066e-01,  2.92318847e-01,  7.46831174e-01,
 1.05144501e+00,  1.10819348e+00,  1.31378759e+00,  1.67228827e+00,
 1.31114705e-01, -4.71339405e-01, -3.68526257e-01, -2.73655760e-01,
-1.74639911e-01, -6.85926667e-02, -2.27618614e-02, -6.50308261e-03,
```

```
-5.22559188e-03,  0.00000000e+00, -1.72315830e-02, -5.28667822e-02,
-1.40304745e-01, -2.79825077e-01, -4.14426850e-01, -5.56442409e-01,
-6.93505364e-01, -7.91976242e-01, -8.39396238e-01, -8.66792684e-01,
-9.57470791e-01, -1.09693377e+00, -1.20821344e+00, -9.93041516e-01,
 2.83493292e-01,  1.16202588e+00,  1.38079317e+00,  1.47213333e+00,
-1.87842105e-01, -4.87087010e-01, -3.81382818e-01, -2.79755294e-01,
-1.76841180e-01, -7.75416110e-02, -2.84219981e-02, -9.05146322e-03,
-4.47218068e-03, -5.99912923e-03, -2.07382330e-02, -6.08726757e-02,
-1.56762364e-01, -2.98766808e-01, -4.23440073e-01, -5.47383357e-01,
-6.60467542e-01, -7.39955400e-01, -7.84673157e-01, -8.30354426e-01,
-9.19477930e-01, -1.02768626e+00, -1.12312056e+00, -7.24574107e-01,
 9.11396955e-01,  1.26672800e+00,  1.26808530e+00, -1.26322853e-01,
-6.15001158e-01, -4.94257528e-01, -3.80605989e-01, -2.74318684e-01,
-1.74339383e-01, -8.34142466e-02, -3.44366952e-02, -9.10219316e-03,
 0.00000000e+00, -7.01625051e-03, -2.41349207e-02, -7.26268650e-02,
-1.77093335e-01, -3.12663421e-01, -4.29351726e-01, -5.35808994e-01,
-6.21524055e-01, -6.81892808e-01, -7.21652423e-01, -7.64909945e-01,
-8.25441817e-01, -9.18722795e-01, -9.51712387e-01,  5.24321653e-01,
 1.25814653e+00,  1.34078537e+00,  5.26386445e-01, -7.41008292e-01,
-6.16130383e-01, -4.86376802e-01, -3.69056398e-01, -2.61937281e-01,
-1.67857853e-01, -8.78540133e-02, -3.61517963e-02, -1.07170309e-02,
-6.46968503e-03, -6.99670155e-03, -3.29168291e-02, -8.99889769e-02,
-1.97007269e-01, -3.26405112e-01, -4.39825172e-01, -5.35380023e-01,
-6.06952314e-01, -6.58515929e-01, -6.95693885e-01, -7.29151292e-01,
-7.79443492e-01, -8.51163526e-01,  1.95262737e-01,  1.26666707e+00,
 1.27555546e+00,  1.33262536e+00,  4.73249789e-01, -7.37382479e-01,
-5.98365057e-01, -4.64336421e-01, -3.49567831e-01, -2.46651030e-01,
-1.58183903e-01, -8.58299810e-02, -3.34069439e-02, -1.01046191e-02,
-4.47218068e-03, -1.03832517e-02, -3.87780708e-02, -1.01996406e-01,
-2.09593853e-01, -3.35970443e-01, -4.51895112e-01, -5.51933623e-01,
-6.31575487e-01, -6.92116804e-01, -7.37673380e-01, -7.70661420e-01,
-8.25583087e-01,  3.18359398e-01,  1.26131934e+00,  1.21324993e+00,
 1.23034163e+00,  7.51796850e-01, -7.89228326e-01, -7.05402002e-01,
-5.58350370e-01, -4.28618997e-01, -3.17942919e-01, -2.21862654e-01,
-1.41859564e-01, -7.89068221e-02, -3.11744792e-02, -8.14253194e-03,
 0.00000000e+00, -9.43975497e-03, -4.30589063e-02, -1.07594241e-01,
-2.05769345e-01, -3.28788483e-01, -4.53617715e-01, -5.68236541e-01,
-6.71779119e-01, -7.58471701e-01, -8.27749600e-01, -4.78227532e-01,
 1.07559492e+00,  1.20365702e+00,  1.14471048e+00,  1.04036189e+00,
 2.40158747e-01, -8.81553053e-01, -7.93291898e-01, -6.30267406e-01,
-4.89574391e-01, -3.69619220e-01, -2.70396349e-01, -1.88159093e-01,
-1.21547772e-01, -6.70583207e-02, -2.66022920e-02, -4.47218068e-03,
-4.47218068e-03, -8.73128518e-03, -4.18115943e-02, -9.93508070e-02,
-1.85539634e-01, -2.96915820e-01, -4.25154738e-01, -5.53290763e-01,
-6.80239686e-01, -8.00331874e-01, -5.21115912e-01,  1.06092844e+00,
 1.17388437e+00,  1.10805362e+00,  1.08603459e+00, -1.70089189e-01,
-1.00098012e+00, -8.37202976e-01, -6.69370525e-01, -5.22438778e-01,
```

```
-3.98476015e-01, -2.96545962e-01, -2.13230528e-01, -1.48685765e-01,
-9.58819271e-02, -4.94357438e-02, -2.03510920e-02, -4.47218068e-03,
-4.47218068e-03, -5.81328182e-03, -3.36322884e-02, -7.90261833e-02,
-1.46560596e-01, -2.40472648e-01, -3.57618938e-01, -4.83408140e-01,
-6.19006896e-01, -3.33566488e-01,  1.14920695e+00,  1.24015539e+00,
 1.16051956e+00,  9.80498105e-01, -2.72398933e-01, -7.98056218e-01,
-8.22162392e-01, -6.66992614e-01, -5.25075709e-01, -4.01799011e-01,
-3.01332206e-01, -2.20970633e-01, -1.58473618e-01, -1.08505048e-01,
-6.90146747e-02, -3.56214738e-02, -1.35251393e-02, -4.47218068e-03,
 0.00000000e+00, -4.47218068e-03, -2.33095270e-02, -5.23114790e-02,
-9.95201176e-02,  3.75652303e-01,  1.56052778e+00,  1.26676158e+00,
 1.70890063e+00,  1.72986700e+00,  1.68987421e+00,  1.47959879e+00,
 6.04638204e-01, -4.06868346e-01, -8.44474138e-01, -7.37034969e-01,
-6.13647378e-01, -4.92579153e-01, -3.80992098e-01, -2.86845335e-01,
-2.10483486e-01, -1.50952117e-01, -1.06865737e-01, -7.22040438e-02,
-4.27030849e-02, -2.00103923e-02, -1.06315936e-02,  0.00000000e+00,
 0.00000000e+00,  0.00000000e+00, -1.03992177e-02, -2.89797372e-02,
-5.90003710e-02,  5.65393143e+00,  9.32171517e+00,  6.03894450e+00,
 4.29527439e+00,  3.26180355e+00,  2.53906824e+00,  5.95612347e-01,
-4.56888542e-01, -5.65183353e-01, -5.30079682e-01, -4.70822173e-01,
-3.99176693e-01, -3.26764211e-01, -2.56040762e-01, -1.93102000e-01,
-1.42678978e-01, -9.99738293e-02, -6.96644120e-02, -4.66764888e-02,
-2.38697824e-02, -1.05815116e-02, -5.20652537e-03,  0.00000000e+00,
 0.00000000e+00,  0.00000000e+00, -7.30040852e-03, -1.08103850e-02,
-2.86123733e-02,  4.67978320e+00,  1.44776493e+01,  1.02314089e+01,
 7.15636603e+00,  3.67314718e+00,  8.34341812e-01, -2.96013152e-01,
-3.10239452e-01, -3.09147087e-01, -2.93110121e-01, -2.66399898e-01,
-2.36363390e-01, -2.02815025e-01, -1.62276290e-01, -1.25127986e-01,
-9.10086059e-02, -6.40925221e-02, -4.49774418e-02, -2.77188634e-02,
-1.57918467e-02, -6.21700643e-03, -4.47218068e-03,  0.00000000e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, -4.47218068e-03,
-1.45694563e-02, -3.25749888e-02, -5.25919643e-02, -7.72888697e-02,
-1.06717325e-01, -1.32626378e-01, -1.57124482e-01, -1.76633010e-01,
-1.84770115e-01, -1.83033625e-01, -1.72094713e-01, -1.53628407e-01,
-1.35924023e-01, -1.17050636e-01, -9.24390070e-02, -7.10341567e-02,
-5.16142170e-02, -3.70608527e-02, -2.28136454e-02, -1.21434281e-02,
-7.18867286e-03, -4.47218068e-03,  0.00000000e+00,  0.00000000e+00,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
-6.34282772e-03, -9.39969068e-03, -1.40191120e-02, -1.99480844e-02,
-2.47700962e-02, -3.08678853e-02, -4.10300336e-02, -4.42931981e-02,
-5.04289489e-02, -5.47949465e-02, -5.93315468e-02, -5.62753816e-02,
-5.10907728e-02, -4.19588006e-02, -3.35435370e-02, -2.43717474e-02,
-1.58399602e-02, -9.99476202e-03, -8.31351974e-03, -4.47218068e-03,
 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00])
```

[38]:
```
#PCA with 0.95 variance retained
pca = PCA(.95)
```

```
pca.fit(X_train)

X_train = pca.transform(X_train)
X_val = pca.transform(X_val)
X_test = pca.transform(X_test)
```

[39]:
```
X_train[30000]
```

[39]:
```
array([-1.17652676e+00, -2.91882918e+00, -5.86255886e+00, -6.29514941e+00,
       -1.33230702e+00,  5.06334359e+00, -4.79654365e+00,  3.34138538e+00,
       -2.07886477e+00,  2.77894626e+00, -7.10281135e+00,  4.20124064e+00,
        4.82642160e+00,  4.73933784e+00, -2.20994168e+00, -9.32995022e-01,
        3.76016208e+00, -4.08395376e+00,  4.36717868e+00,  3.35742921e+00,
        1.18356389e+00, -1.65859407e+00, -5.67231519e+00, -1.84240525e+00,
       -3.82119837e+00, -9.22646592e-01,  3.40417903e+00, -2.63054016e+00,
       -1.36811602e+00,  3.01599750e+00, -2.40828009e+00,  5.42084953e+00,
       -2.83695886e-01, -5.08245297e-01,  2.07432621e+00,  3.51503244e+00,
       -1.84580715e+00, -2.03561335e+00,  2.57752864e+00, -7.92538642e-02,
       -7.72439103e-01, -6.17169394e-01, -1.70230532e+00, -4.54784456e-01,
        1.03212890e+00, -8.50657186e-01, -1.66415179e+00, -1.67563840e+00,
       -4.79847815e-01, -9.01788934e-01,  4.53690834e-01, -2.19440493e+00,
       -1.59634311e+00,  4.29439990e-01,  1.99437057e+00,  1.29687408e+00,
       -1.02402240e+00, -4.16104576e-01, -3.17857909e-02,  9.39489431e-02,
        5.49849885e-01, -2.90615239e+00,  3.43148213e-01, -1.07230559e+00,
       -3.79460105e+00, -1.69003062e+00, -1.00975601e+00,  6.34969131e-01,
       -4.31261208e-01, -1.27013915e+00,  7.03951048e-01,  1.38830572e+00,
       -7.67028429e-02, -9.50301235e-02,  1.64023874e+00, -5.38243179e-01,
       -2.84332328e+00, -8.28422803e-01,  2.19110012e+00,  5.34705213e-01,
       -1.33150644e+00, -1.99936290e+00,  1.62053579e+00, -1.02282390e-01,
        6.70147064e-02,  5.78449252e-01, -2.27839503e+00,  1.44631330e-01,
       -1.58506977e-01, -5.67679558e-01,  5.77654090e-01, -1.30485521e+00,
        2.09587394e+00,  2.44850038e+00,  7.03385976e-02,  1.28610353e+00,
       -7.11561492e-02, -1.91542306e-01, -2.38982575e+00, -1.83110634e+00,
       -7.39166568e-01,  1.57844946e-01, -7.54686161e-01,  2.49015700e-01,
        2.44131315e+00,  1.77653004e+00,  2.41409256e-02,  5.39554282e-01,
       -1.55752840e+00, -4.04916311e-01,  1.31830992e+00,  1.47256597e+00,
        1.06028075e+00,  2.37646728e+00, -5.86387995e-01,  1.74297519e+00,
       -1.03834027e+00, -2.29104444e-01, -4.84429179e-01, -8.45097836e-01,
       -1.32038170e+00,  1.41297271e+00,  8.91940566e-01, -1.65588557e+00,
       -8.08613583e-01,  1.63806540e+00,  2.88999882e+00, -2.53270547e-01,
       -9.44178899e-01,  5.37324035e-01,  7.17320571e-02, -3.73166286e-01,
        1.17366255e+00, -7.52467847e-02, -1.24661897e+00,  4.48851844e-01,
       -5.29247283e-01, -9.31427227e-01, -1.27004118e+00,  9.43104183e-01,
        7.27304818e-01, -7.74485371e-02, -8.88322406e-01,  4.03632362e-01,
        1.40935972e-01, -1.20457443e-01, -2.47247673e-01, -6.14480788e-01,
        8.07417713e-01, -1.26545886e+00,  6.54158580e-01, -2.49849571e-01,
```

```
 1.96232587e+00, -1.62371950e+00,  1.69723804e-01,  4.39053513e-01,
-2.90322019e-01, -1.01950789e+00,  1.53221239e+00,  3.11921086e-01,
-2.68472442e-01, -1.71450807e-02, -1.01912583e+00,  8.64962702e-01,
 2.74664118e-01,  3.31638163e-01, -5.21218659e-01,  8.85917055e-01,
-7.27828745e-01,  1.07965555e+00, -1.09811554e-01,  4.73432829e-01,
 1.49836637e-01,  5.58116479e-01,  7.81708112e-01,  5.82546394e-01,
-8.82969224e-04, -2.39787494e+00, -4.04635923e+00, -1.60659234e+00,
 9.11010430e-02,  3.24050906e+00, -1.01669129e+00, -1.85262235e+00,
 1.37239615e+00,  1.53038012e-01,  2.08418091e+00,  2.21768242e-01,
-5.37318177e-01,  2.16852118e-01, -2.45814482e-01,  2.07669270e+00,
-2.68459781e-01, -4.85261576e-01,  6.26950674e-01, -2.24136885e+00,
-1.57677798e-01, -4.43169169e-01,  8.53679500e-01,  3.05651346e-01,
 1.29400051e-01, -3.46541685e-01, -1.98112461e-01, -9.42602281e-02,
-1.50388611e+00, -2.12681173e-01,  6.28277185e-01, -6.68012528e-01,
 2.24340646e-01,  9.49049701e-02, -9.83077117e-01, -7.43173765e-01,
-1.48536173e+00, -3.16404978e+00,  2.05323394e+00, -1.24086553e+00,
-5.61882054e-01, -3.92394141e-01, -2.99250659e-01,  1.26578080e+00,
 6.73820104e-01, -5.93411263e-01, -9.88628304e-01,  5.33615620e-01,
 7.52134214e-01, -1.62844435e+00, -6.21644911e-01, -1.13334783e+00,
-8.57089534e-01,  1.08796548e+00, -3.61008654e-01,  1.84803542e+00,
 7.54621542e-01, -6.87164060e-01,  2.93966483e-01,  4.99032848e-01,
 1.22476158e+00,  1.18729131e+00,  2.04613320e+00, -1.93889957e+00,
-1.58705164e-01, -7.56240248e-01,  6.67539247e-01, -8.26647893e-01,
-5.85233096e-01,  3.32075861e-01,  3.15811965e-01, -2.54826021e+00,
-2.24728140e+00,  5.39243030e-01, -6.10897950e-01,  7.46033314e-01,
-4.49194562e-01,  5.48818679e-01, -3.69439789e-01,  1.44180581e-01,
 5.22288293e-01, -4.03906344e-01,  7.23824086e-01,  1.75739735e+00,
 4.06621277e-01, -1.11859637e+00, -1.78173602e+00, -5.21053737e-01,
-4.09914160e-01, -1.24364960e+00,  8.39747643e-01,  6.54132228e-01,
-9.03172933e-01, -3.68480145e-01, -1.50706588e-01,  6.54260812e-01,
-6.78765042e-01,  4.91560464e-01, -8.55461094e-01,  6.55674839e-01,
-1.13424664e+00, -1.20909361e-01, -9.79425559e-01, -1.55735672e-01,
-2.19345978e-03,  4.21199570e-01, -4.20269472e-01, -4.37797947e-02,
-6.11469290e-01,  1.91837688e-01, -4.00356727e-01, -8.42766468e-01,
-2.22553604e-01,  7.67949420e-01,  1.51365524e+00,  1.38854998e+00,
-4.90378000e-01,  1.19806364e+00,  1.19261359e+00,  8.57035034e-01,
-4.20577800e-01, -2.51735198e-01, -1.03927977e+00,  1.41241385e-01,
-4.26809425e-01,  8.82038002e-01, -1.65381357e-01,  2.53904960e-01,
-2.47037199e-01,  1.49546673e-01,  1.17745870e+00,  6.57779027e-01,
-1.13575986e+00,  5.60417542e-01,  2.32330689e-01,  3.49629807e-02,
 4.85376610e-03,  6.79431742e-02,  3.99233824e-01, -1.34710797e-01,
 2.27411015e-01,  2.43074804e-01, -6.46867241e-02,  2.82487137e-01,
-2.05658237e-01,  8.57795496e-01,  2.13691218e-01,  2.70191087e-01,
 1.92077502e-01, -1.81793861e-01, -3.99313424e-01,  3.18830005e-02])
```

[40]:
```
X_train.shape
```

[40]: (50000, 328)

[41]:
```
#Training PCA applied random forest classifier
rdf = RandomForestClassifier(n_estimators=50, n_jobs=2, random_state=19)
t0 = time.time()
rdf.fit(X_train, Y_train)
t1 = time.time()
pred_train = rdf.predict(X_train[0:10000])
t2 = time.time()


print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
 ↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 28.319
Prediction time for train: 0.102
Youdens index for train: 1.000
```

[42]:
```
t3 = time.time()
pred_val = rdf.predict(X_val)
t4 = time.time()
print(f'Prediction time for validation: {t4-t3:3.3f}')
print(f'Youdens index for validation: {((metrics.balanced_accuracy_score(Y_val,␣
 ↪pred_val)*2)-1):.3f}')
```

```
Prediction time for validation: 0.103
Youdens index for validation: 0.862
```

[43]:
```
t5 = time.time()
pred_test = rdf.predict(X_test)
t6 = time.time()
print(f'Prediction time for test: {t6-t5:3.3f}')
print(f'Youdens index for test: {((metrics.balanced_accuracy_score(Y_test,␣
 ↪pred_test)*2)-1):.3f}')
```

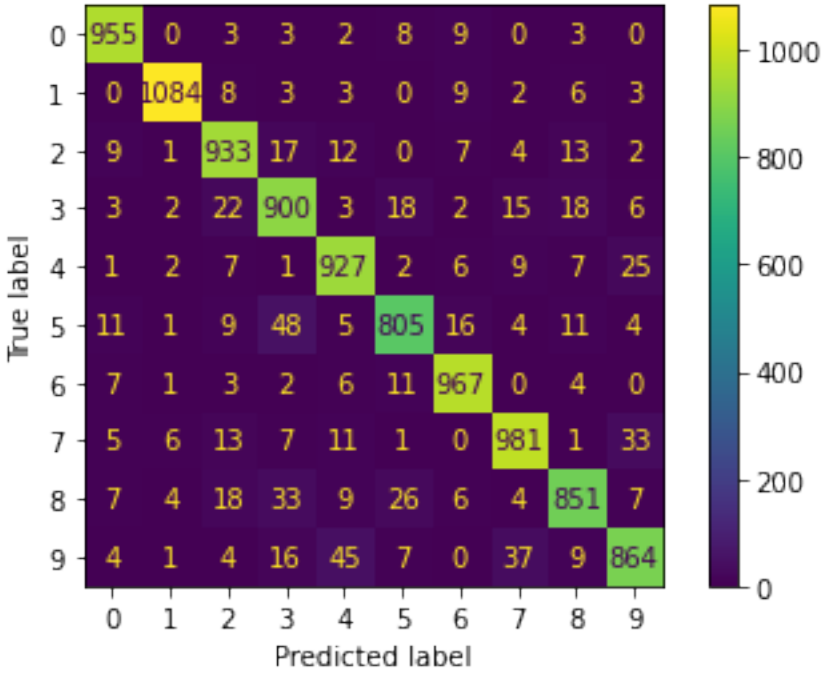
```
Prediction time for test: 0.129
Youdens index for test: 0.851
```

[44]:
```
#PCA applied Random forest confusion matrix

cm = confusion_matrix(Y_test, pred_test)
cm
```

[44]:
```
array([[ 955,    0,    3,    3,    2,    8,    9,    0,    3,    0],
       [   0, 1084,    8,    3,    3,    0,    9,    2,    6,    3],
       [   9,    1,  933,   17,   12,    0,    7,    4,   13,    2],
       [   3,    2,   22,  900,    3,   18,    2,   15,   18,    6],
```

```
[   1,    2,    7,    1,  927,    2,    6,    9,    7,   25],
[  11,    1,    9,   48,    5,  805,   16,    4,   11,    4],
[   7,    1,    3,    2,    6,   11,  967,    0,    4,    0],
[   5,    6,   13,    7,   11,    1,    0,  981,    1,   33],
[   7,    4,   18,   33,    9,   26,    6,    4,  851,    7],
[   4,    1,    4,   16,   45,    7,    0,   37,    9,  864]])
```

[45]:
```
#Create graph of PCA applied Random forest confusion matrix

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rdf.classes_)

disp.plot()

plt.show()
```



[46]:
```
#Training PCA applied naive bayes classifier

nb = GaussianNB()
t0 = time.time()
nb.fit(X_train, Y_train)
t1 = time.time()
pred_train = nb.predict(X_train[0:10000])
t2 = time.time()
```

```
print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 0.330
Prediction time for train: 0.223
Youdens index for train: -0.123
```

[47]:
```
#Training PCA applied naive bayes classifier

nb = GaussianNB(var_smoothing=0.086)
t0 = time.time()
nb.fit(X_train, Y_train)
t1 = time.time()
pred_train = nb.predict(X_train[0:10000])
t2 = time.time()

print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 0.326
Prediction time for train: 0.234
Youdens index for train: -0.285
```

[48]:
```
t3 = time.time()
pred_val = nb.predict(X_val)
t4 = time.time()
print(f'Prediction time for validation: {t4-t3:3.3f}')
print(f'Youdens index for validation: {((metrics.balanced_accuracy_score(Y_val,␣
↪pred_val)*2)-1):.3f}')
```

```
Prediction time for validation: 0.219
Youdens index for validation: -0.286
```

[49]:
```
t5 = time.time()
pred_test = nb.predict(X_test)
t6 = time.time()
print(f'Prediction time for test: {t6-t5:3.3f}')
print(f'Youdens index for test: {((metrics.balanced_accuracy_score(Y_test,␣
↪pred_test)*2)-1):.3f}')
```

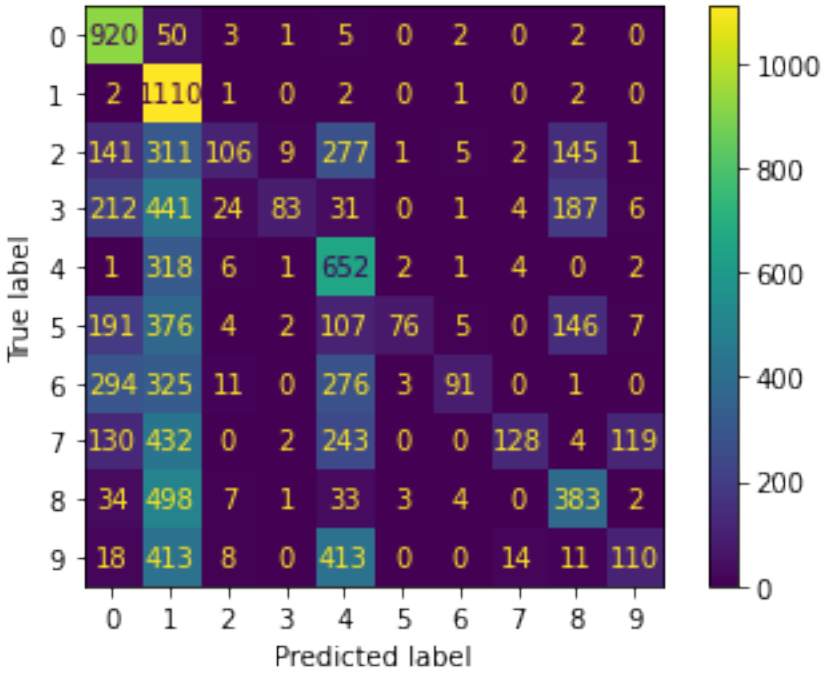
```
Prediction time for test: 0.219
Youdens index for test: -0.283
```

[50]:
```
#PCA applied Naive bayes confusion matrix
```

```
cm = confusion_matrix(Y_test, pred_test)
cm
```

[50]:
```
array([[ 920,   50,    3,    1,    5,    0,    2,    0,    2,    0],
       [   2, 1110,    1,    0,    2,    0,    1,    0,    2,    0],
       [ 141,  311,  106,    9,  277,    1,    5,    2,  145,    1],
       [ 212,  441,   24,   83,   31,    0,    1,    4,  187,    6],
       [   1,  318,    6,    1,  652,    2,    1,    4,    0,    2],
       [ 191,  376,    4,    2,  107,   76,    5,    0,  146,    7],
       [ 294,  325,   11,    0,  276,    3,   91,    0,    1,    0],
       [ 130,  432,    0,    2,  243,    0,    0,  128,    4,  119],
       [  34,  498,    7,    1,   33,    3,    4,    0,  383,    2],
       [  18,  413,    8,    0,  413,    0,    0,   14,   11,  110]])
```

[51]:
```
#Create graph of PCA applied Naive bayes confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=nb.classes_)

disp.plot()

plt.show()
```



[52]:
```
#Training PCA applied KNeighborsClassifier

knc = KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='brute')
```

```
t0 = time.time()
knc.fit(X_train, Y_train)
t1 = time.time()
pred_train = knc.predict(X_train[0:10000])
t2 = time.time()

print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
 ↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 0.064
Prediction time for train: 12.814
Youdens index for train: 0.929
```

[53]:
```
t3 = time.time()
pred_val = knc.predict(X_val)
t4 = time.time()
print(f'Prediction time for validation: {t4-t3:3.3f}')
print(f'Youdens index for validation: {((metrics.balanced_accuracy_score(Y_val,␣
 ↪pred_val)*2)-1):.3f}')
```

```
Prediction time for validation: 11.885
Youdens index for validation: 0.898
```

[54]:
```
t5 = time.time()
pred_test = knc.predict(X_test)
t6 = time.time()
print(f'Prediction time for test: {t6-t5:3.3f}')
print(f'Youdens index for test: {((metrics.balanced_accuracy_score(Y_test,␣
 ↪pred_test)*2)-1):.3f}')
```

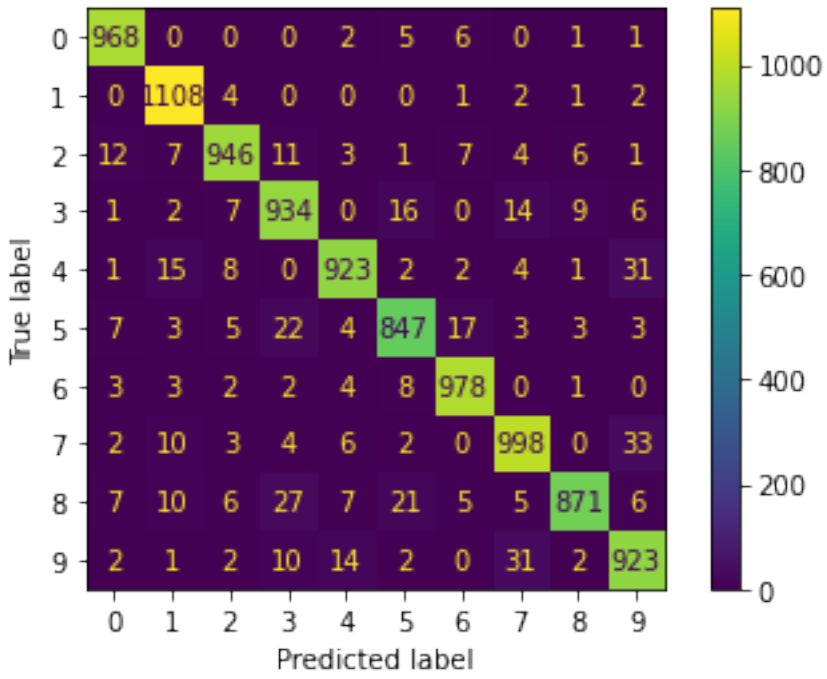
```
Prediction time for test: 12.524
Youdens index for test: 0.898
```

[55]:
```
#PCA applied KNeighborsClassifier confusion matrix

cm = confusion_matrix(Y_test, pred_test)
cm
```

[55]:
```
array([[ 968,    0,    0,    0,    2,    5,    6,    0,    1,    1],
       [   0, 1108,    4,    0,    0,    0,    1,    2,    1,    2],
       [  12,    7,  946,   11,    3,    1,    7,    4,    6,    1],
       [   1,    2,    7,  934,    0,   16,    0,   14,    9,    6],
       [   1,   15,    8,    0,  923,    2,    2,    4,    1,   31],
       [   7,    3,    5,   22,    4,  847,   17,    3,    3,    3],
       [   3,    3,    2,    2,    4,    8,  978,    0,    1,    0],
       [   2,   10,    3,    4,    6,    2,    0,  998,    0,   33],
       [   7,   10,    6,   27,    7,   21,    5,    5,  871,    6],
```

```
       [   2,    1,    2,   10,   14,    2,    0,   31,    2,  923]])
```

[56]:
```
#Create graph of PCA applied KNeighborsClassifier confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=knc.classes_)

disp.plot()

plt.show()
```



[57]:
```
#Training PCA applied MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(75, 50), random_state=19)
t0 = time.time()
mlp.fit(X_train, Y_train)
t1 = time.time()
pred_train = mlp.predict(X_train[0:10000])
t2 = time.time()

print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
 ↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 18.930
Prediction time for train: 0.025
```

```
Youdens index for train: 1.000
```

[58]:
```
t3 = time.time()
pred_val = mlp.predict(X_val)
t4 = time.time()
print(f'Prediction time for validation: {t4-t3:3.3f}')
print(f'Youdens index for validation: {((metrics.balanced_accuracy_score(Y_val,␣
 ↪pred_val)*2)-1):.3f}')
```

```
Prediction time for validation: 0.024
Youdens index for validation: 0.937
```

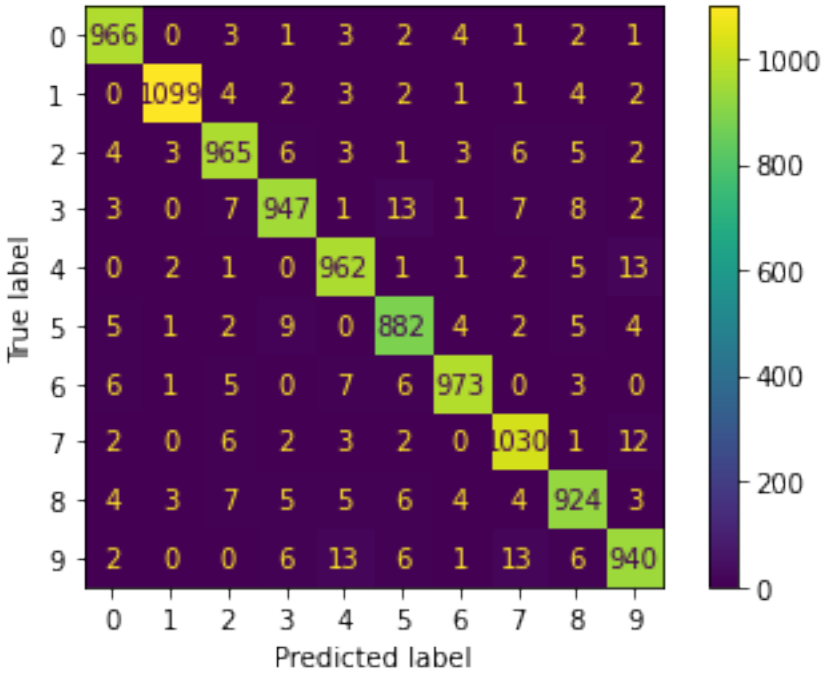
[59]:
```
t5 = time.time()
pred_test = mlp.predict(X_test)
t6 = time.time()
print(f'Prediction time for test: {t6-t5:3.3f}')
print(f'Youdens index for test: {((metrics.balanced_accuracy_score(Y_test,␣
 ↪pred_test)*2)-1):.3f}')
```

```
Prediction time for test: 0.031
Youdens index for test: 0.937
```

[60]:
```
#PCA applied MLPClassifier confusion matrix

cm = confusion_matrix(Y_test, pred_test)
cm
```

[60]:
```
array([[ 966,    0,    3,    1,    3,    2,    4,    1,    2,    1],
       [   0, 1099,    4,    2,    3,    2,    1,    1,    4,    2],
       [   4,    3,  965,    6,    3,    1,    3,    6,    5,    2],
       [   3,    0,    7,  947,    1,   13,    1,    7,    8,    2],
       [   0,    2,    1,    0,  962,    1,    1,    2,    5,   13],
       [   5,    1,    2,    9,    0,  882,    4,    2,    5,    4],
       [   6,    1,    5,    0,    7,    6,  973,    0,    3,    0],
       [   2,    0,    6,    2,    3,    2,    0, 1030,    1,   12],
       [   4,    3,    7,    5,    5,    6,    4,    4,  924,    3],
       [   2,    0,    0,    6,   13,    6,    1,   13,    6,  940]])
```

[61]:
```
#Create graph of PCA applied MLPClassifier confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=mlp.classes_)

disp.plot()

plt.show()
```



[62]:
```
#PCA applied Cluster analysis

t0 = time.time()
kmeans = KMeans(n_clusters=10, n_init=10, random_state=19, max_iter=300,␣
↪algorithm='auto').fit(X_train, Y_train)
t1 = time.time()
pred_train = kmeans.predict(X_train[0:10000])
t2 = time.time()
pred_train = pred_train.astype(str)

print(f'Training time: {t1-t0:3.3f}')
print(f'Prediction time for train: {t2-t1:3.3f}')
print(f'Youdens index for train: {((metrics.balanced_accuracy_score(Y_train[0:
↪10000], pred_train)*2)-1):.3f}')
```

```
Training time: 24.884
Prediction time for train: 0.015
Youdens index for train: -0.817
```

[63]:
```
t3 = time.time()
pred_val = kmeans.predict(X_val)
t4 = time.time()

pred_val = pred_val.astype(str)
```

```
print(f'Prediction time for validation: {t4-t3:3.3f}')
print(f'Youdens index for validation: {((metrics.balanced_accuracy_score(Y_val,␣
 ↪pred_val)*2)-1):.3f}')
```

```
Prediction time for validation: 0.015
Youdens index for validation: -0.813
```

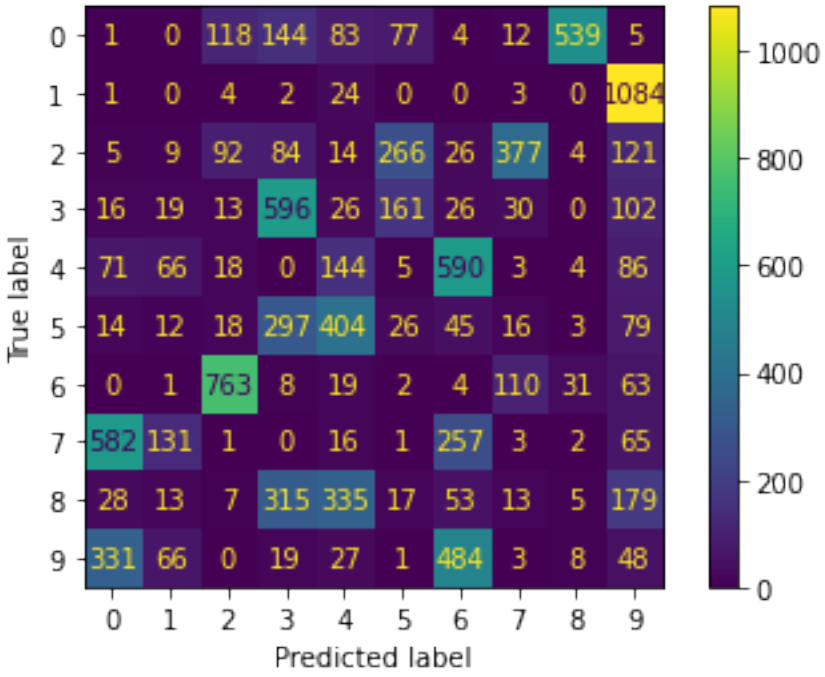
[64]:
```
t5 = time.time()
pred_test = kmeans.predict(X_test)
t6 = time.time()

pred_test = pred_test.astype(str)

print(f'Prediction time for test: {t6-t5:3.3f}')
print(f'Youdens index for test: {((metrics.balanced_accuracy_score(Y_test,␣
 ↪pred_test)*2)-1):.3f}')
```

```
Prediction time for test: 0.017
Youdens index for test: -0.814
```

[65]:
```
#PCA applied Cluster analysis confusion matrix

cm = confusion_matrix(Y_test, pred_test)
cm
```

[65]:
```
array([[   1,    0,  118,  144,   83,   77,    4,   12,  539,    5],
       [   1,    0,    4,    2,   24,    0,    0,    3,    0, 1084],
       [   5,    9,   92,   84,   14,  266,   26,  377,    4,  121],
       [  16,   19,   13,  596,   26,  161,   26,   30,    0,  102],
       [  71,   66,   18,    0,  144,    5,  590,    3,    4,   86],
       [  14,   12,   18,  297,  404,   26,   45,   16,    3,   79],
       [   0,    1,  763,    8,   19,    2,    4,  110,   31,   63],
       [ 582,  131,    1,    0,   16,    1,  257,    3,    2,   65],
       [  28,   13,    7,  315,  335,   17,   53,   13,    5,  179],
       [ 331,   66,    0,   19,   27,    1,  484,    3,    8,   48]])
```

[66]:
```
#Create graph of PCA applied Cluster analysis confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)

disp.plot()

plt.show()
```


True label
Predicted label
0 1 0 118 144 83 77 4 12 539 5
1 1 0 4 2 24 0 0 3 0 1084
2 5 9 92 84 14 266 26 377 4 121
3 16 19 13 596 26 161 26 30 0 102
4 71 66 18 0 144 5 590 3 4 86
5 14 12 18 297 404 26 45 16 3 79
6 0 1 763 8 19 2 4 110 31 63
7 582 131 1 0 16 1 257 3 2 65
8 28 13 7 315 335 17 53 13 5 179
9 331 66 0 19 27 1 484 3 8 48
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0