Face Recognition

In [34]:
```
import numpy as np
import pandas as pd

#Visualization
import matplotlib.pyplot as plt

#Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn import metrics

#System
import os
print(os.listdir("../input"))
```

```
['olivetti_faces_target.npy', 'olivetti_faces.npy']
```

In [35]:
```
import warnings
warnings.filterwarnings('ignore')
print("Warnings ignored!!")
```

```
Warnings ignored!!
```

In this task, face recognition was conducted using facial images. The steps for face recognition are as follows:

Principal components of facial images were obtained through PCA. An appropriate number of principal components was determined. Accuracy scores were obtained for three different classification models. Cross-validation accuracy scores were obtained for the three different classification models. Parameter optimization was performed on the best model.

In [36]:
```
data=np.load("../input/olivetti_faces.npy")
target=np.load("../input/olivetti_faces_target.npy")
```

Let's verify above information

In [37]:
```
print("There are {} images in the dataset".format(len(data)))
print("There are {} unique targets in the dataset".format(len(np.unique(target))))
print("Size of each image is {}x{}".format(data.shape[1],data.shape[2]))
print("Pixel values were scaled to [0,1] interval. e.g:{}".format(data[0][0,:4]))
```

```
There are 400 images in the dataset
There are 40 unique targets in the dataset
Size of each image is 64x64
Pixel values were scaled to [0,1] interval. e.g:[0.30991736 0.3677686  0.41735536 0.4
4214877]
```

In [38]:
```
print("unique target number:",np.unique(target))
```

```
unique target number: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39]
```

Show 48 Disticnt People in the Olivetti Dataset

In [39]:
```
def show_40_distinct_people(images, unique_ids):
    #Creating 4X10 subplots in  18x9 figure size
    fig, axarr=plt.subplots(nrows=4, ncols=10, figsize=(18, 9))
    #For easy iteration flattened 4X10 subplots matrix to 40 array
    axarr=axarr.flatten()

    #iterating over user ids
    for unique_id in unique_ids:
        image_index=unique_id*10
        axarr[unique_id].imshow(images[image_index], cmap='gray')
        axarr[unique_id].set_xticks([])
        axarr[unique_id].set_yticks([])
        axarr[unique_id].set_title("face id:{}".format(unique_id))
    plt.suptitle("There are 40 distinct people in the dataset")
```

In [40]:
```
show_40_distinct_people(data, np.unique(target))
```

As seen in the photo gallery above, the data set has 40 different person-owned, facial images.

Show 10 Face Images of Selected Target

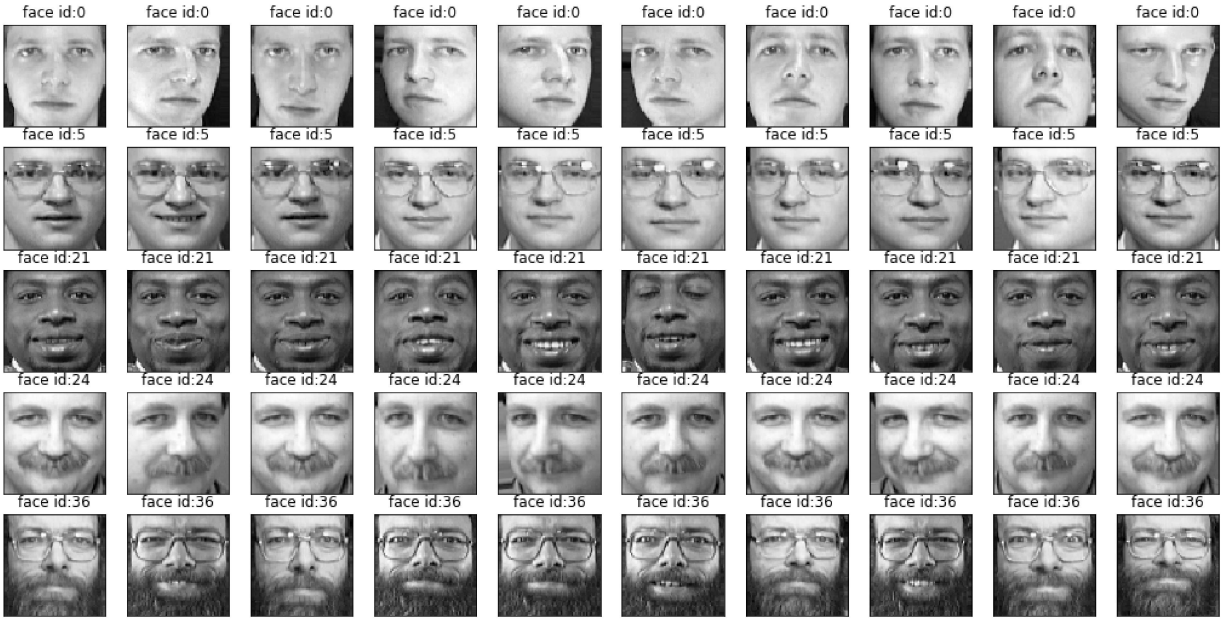
In [41]:
```
def show_10_faces_of_n_subject(images, subject_ids):
    cols=10# each subject has 10 distinct face images
    rows=(len(subject_ids)*10)/cols #
    rows=int(rows)

    fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(18,9))
```

```
    #axarr=axarr.flatten()

    for i, subject_id in enumerate(subject_ids):
        for j in range(cols):
            image_index=subject_id*10 + j
            axarr[i,j].imshow(images[image_index], cmap="gray")
            axarr[i,j].set_xticks([])
            axarr[i,j].set_yticks([])
            axarr[i,j].set_title("face id:{}".format(subject_id))
```

In [42]:
```
#You can playaround subject_ids to see other people faces
show_10_faces_of_n_subject(images=data, subject_ids=[0,5, 21, 24, 36])
```



Each face of a subject has different characteristic in context of varying lighting, facial express and facial detail(glasses, beard)

Machine Learning Model fo Face Recognition

Machine learning models can work on vectors. Since the image data is in the matrix form, it must be converted to a vector.

In [43]:
```
#We reshape images for machine learnig  model
X=data.reshape((data.shape[0],data.shape[1]*data.shape[2]))
print("X shape:",X.shape)
```

```
X shape: (400, 4096)
```

Split data and target into train and test datasets

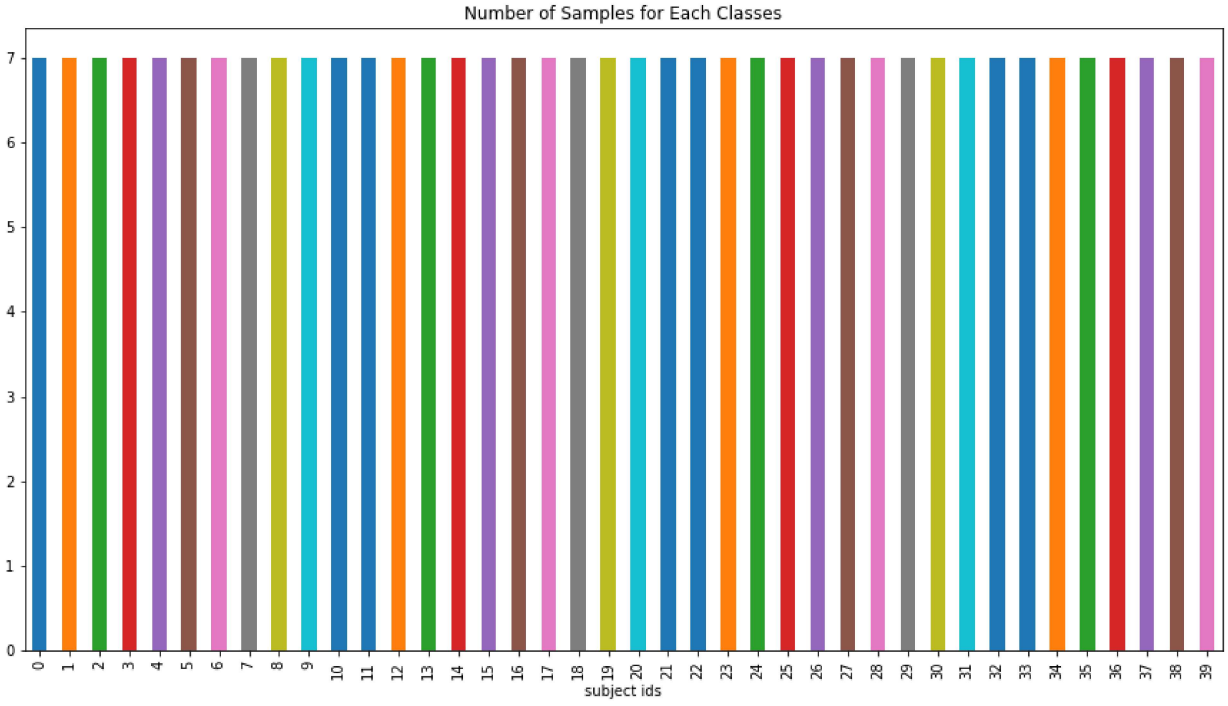
The dataset contains 10 face images for each subject. Out of these images, 70 percent will be allocated for training and 30 percent for testing. The stratify feature ensures an equal number of training and test images for each subject, resulting in 7 training images and 3 test images for every subject. You have the flexibility to adjust the training and test rates as needed.

In [44]:
```
X_train, X_test, y_train, y_test=train_test_split(X, target, test_size=0.3, stratify=t
print("X_train shape:",X_train.shape)
print("y_train shape:{}".format(y_train.shape))
```

```
X_train shape: (280, 4096)
y_train shape:(280,)
```

In [45]:
```
y_frame=pd.DataFrame()
y_frame['subject ids']=y_train
y_frame.groupby(['subject ids']).size().plot.bar(figsize=(15,8),title="Number of Sampl
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f640be05f60>
```



Principle Component Analysis

Machine learning methods are generally categorized into two groups: supervised learning and unsupervised learning. In supervised learning, the dataset is typically divided into two main parts: 'data' and 'output'. The 'data' section contains the sample values, while the 'output' section holds the class (for classification) or the target value (for regression). On the other hand, in unsupervised learning, the dataset contains only the data section.

Unsupervised learning is generally categorized into two methods: data transformation and clustering. In this study, data transformation will be conducted using unsupervised learning techniques. These methods facilitate easier data interpretation by both computers and humans.

One of the most common applications of unsupervised transformation is reducing data size, which involves reducing the dimension of the data.

Principal Component Analysis (PCA) is a method used to represent data in a more compact form. According to this method, data is transformed into new components, and the size of the data is decreased by selecting the most essential components.

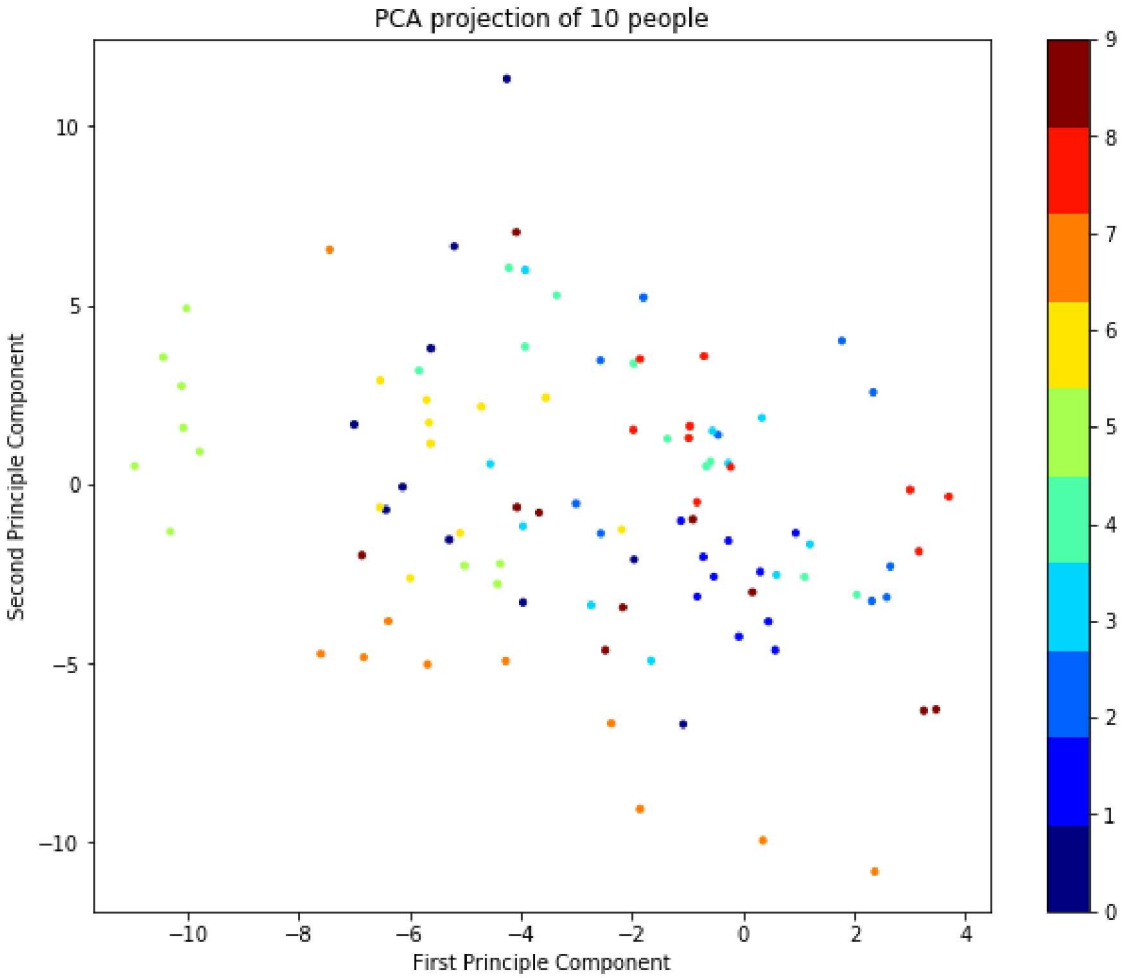
In [46]:
```
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pca.fit(X)
X_pca=pca.transform(X)
```

In [47]:
```
number_of_people=10
index_range=number_of_people*10
fig=plt.figure(figsize=(10,8))
ax=fig.add_subplot(1,1,1)
scatter=ax.scatter(X_pca[:index_range,0],
            X_pca[:index_range,1],
            c=target[:index_range],
            s=10,
           cmap=plt.get_cmap('jet', number_of_people)
          )

ax.set_xlabel("First Principle Component")
ax.set_ylabel("Second Principle Component")
ax.set_title("PCA projection of {} people".format(number_of_people))

fig.colorbar(scatter)
```

Out[47]:
```
<matplotlib.colorbar.Colorbar at 0x7f640752ffd0>
```



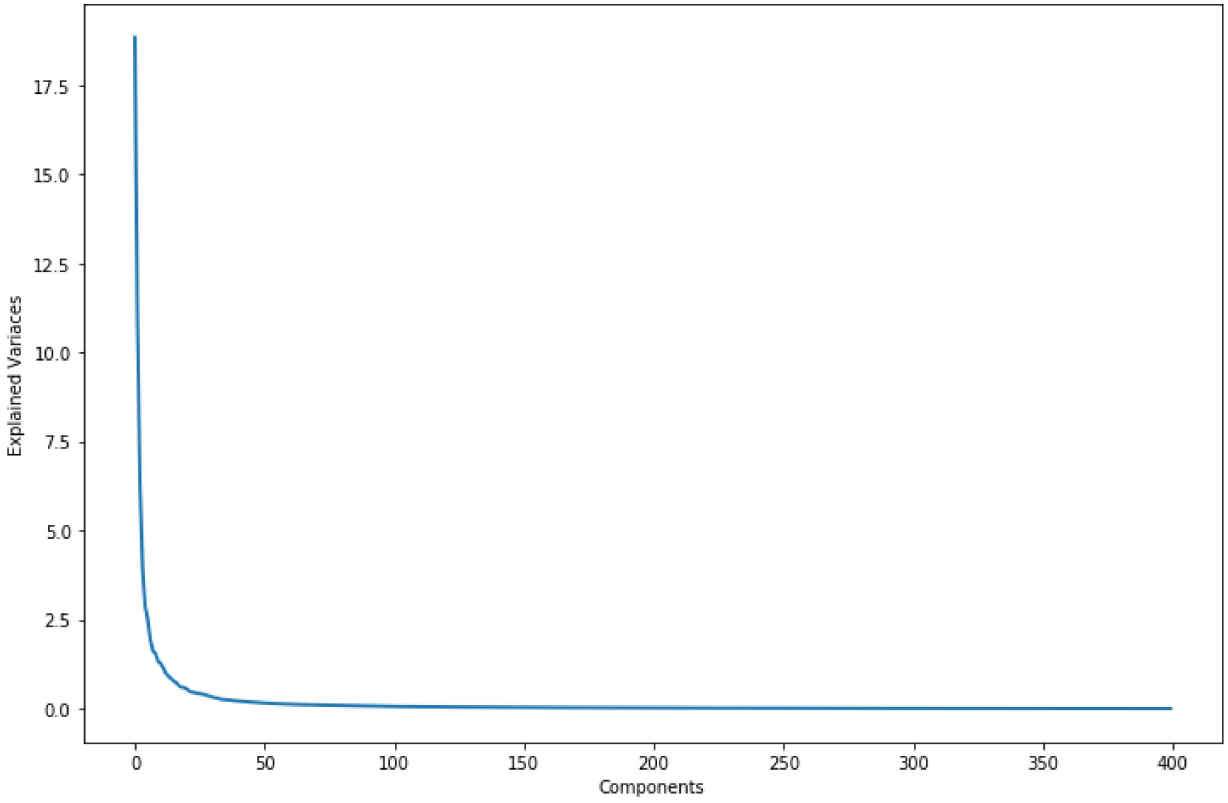
Finding Optimum Number of Principle Component

In [48]:
```
pca=PCA()
pca.fit(X)

plt.figure(1, figsize=(12,8))

plt.plot(pca.explained_variance_, linewidth=2)

plt.xlabel('Components')
plt.ylabel('Explained Variaces')
plt.show()
```



In the figure above, it can be seen that 90 and more PCA components represent the same data.

Now let's make the classification process using 90 PCA components.

In [49]:
```
n_components=90
```

In [50]:
```
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X_train)
```

Out[50]:
```
PCA(copy=True, iterated_power='auto', n_components=90, random_state=None,
  svd_solver='auto', tol=0.0, whiten=True)
```

Show Average Face

In [51]:
```
fig,ax=plt.subplots(1,1,figsize=(8,8))
ax.imshow(pca.mean_.reshape((64,64)), cmap="gray")
ax.set_xticks([])
ax.set_yticks([])
ax.set_title('Average Face')
```

Out[51]:
```
Text(0.5,1,'Average Face')
```

Show Eigen Faces

In [52]:
```
number_of_eigenfaces=len(pca.components_)
eigen_faces=pca.components_.reshape((number_of_eigenfaces, data.shape[1], data.shape[2

cols=10
rows=int(number_of_eigenfaces/cols)
fig, axarr=plt.subplots(nrows=rows, ncols=cols, figsize=(15,15))
axarr=axarr.flatten()
for i in range(number_of_eigenfaces):
    axarr[i].imshow(eigen_faces[i],cmap="gray")
    axarr[i].set_xticks([])
    axarr[i].set_yticks([])
    axarr[i].set_title("eigen id:{}".format(i))
plt.suptitle("All Eigen Faces".format(10*"=", 10*"="))
```

Out[52]:
```
Text(0.5,0.98,'All Eigen Faces')
```

Classification Results

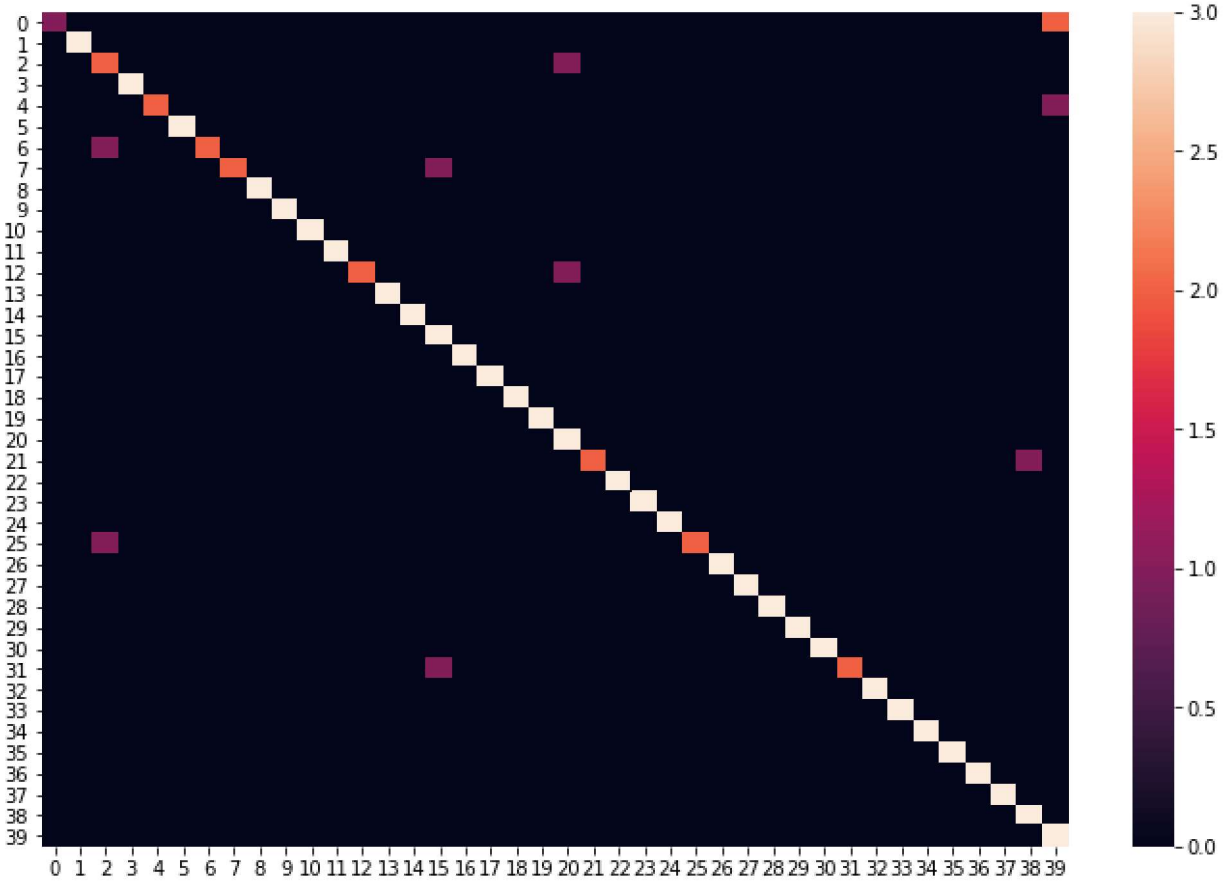
In [53]:
```
X_train_pca=pca.transform(X_train)
X_test_pca=pca.transform(X_test)
```

In [54]:
```
clf = SVC()
clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
print("accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
```

```
accuracy score:0.92
```

In [55]:
```
import seaborn as sns
plt.figure(1, figsize=(12,8))
sns.heatmap(metrics.confusion_matrix(y_test, y_pred))
```

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x7f640515b860>



In [56]:
```
print(metrics.classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	0.33	0.50	3
1	1.00	1.00	1.00	3
2	0.50	0.67	0.57	3
3	1.00	1.00	1.00	3
4	1.00	0.67	0.80	3
5	1.00	1.00	1.00	3
6	1.00	0.67	0.80	3
7	1.00	0.67	0.80	3
8	1.00	1.00	1.00	3
9	1.00	1.00	1.00	3
10	1.00	1.00	1.00	3
11	1.00	1.00	1.00	3
12	1.00	0.67	0.80	3
13	1.00	1.00	1.00	3
14	1.00	1.00	1.00	3
15	0.60	1.00	0.75	3
16	1.00	1.00	1.00	3
17	1.00	1.00	1.00	3
18	1.00	1.00	1.00	3
19	1.00	1.00	1.00	3
20	0.60	1.00	0.75	3
21	1.00	0.67	0.80	3
22	1.00	1.00	1.00	3
23	1.00	1.00	1.00	3
24	1.00	1.00	1.00	3
25	1.00	0.67	0.80	3
26	1.00	1.00	1.00	3
27	1.00	1.00	1.00	3
28	1.00	1.00	1.00	3
29	1.00	1.00	1.00	3
30	1.00	1.00	1.00	3
31	1.00	0.67	0.80	3
32	1.00	1.00	1.00	3
33	1.00	1.00	1.00	3
34	1.00	1.00	1.00	3
35	1.00	1.00	1.00	3
36	1.00	1.00	1.00	3
37	1.00	1.00	1.00	3
38	0.75	1.00	0.86	3
39	0.50	1.00	0.67	3
micro avg	0.92	0.92	0.92	120
macro avg	0.95	0.92	0.92	120
weighted avg	0.95	0.92	0.92	120

More Results

In [57]:
```
models=[]
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(("LR",LogisticRegression()))
models.append(("NB",GaussianNB()))
models.append(("KNN",KNeighborsClassifier(n_neighbors=5)))
models.append(("DT",DecisionTreeClassifier()))
models.append(("SVM",SVC()))
```

```
for name, model in models:

    clf=model

    clf.fit(X_train_pca, y_train)

    y_pred=clf.predict(X_test_pca)
    print(10*"=","{} Result".format(name).upper(),10*"=")
    print("Accuracy score:{:0.2f}".format(metrics.accuracy_score(y_test, y_pred)))
    print()
```

```
========== LDA RESULT ==========
Accuracy score:0.93

========== LR RESULT ==========
Accuracy score:0.93

========== NB RESULT ==========
Accuracy score:0.87

========== KNN RESULT ==========
Accuracy score:0.70

========== DT RESULT ==========
Accuracy score:0.67

========== SVM RESULT ==========
Accuracy score:0.92
```

According to the above results, Linear Discriminant Analysis and Logistic Regression seems to have the best performances.

Validated Results

In [58]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X)
X_pca=pca.transform(X)
for name, model in models:
    kfold=KFold(n_splits=5, shuffle=True, random_state=0)

    cv_scores=cross_val_score(model, X_pca, target, cv=kfold)
    print("{} mean cross validations score:{:.2f}".format(name, cv_scores.mean()))
```

```
LDA mean cross validations score:0.97
LR mean cross validations score:0.94
NB mean cross validations score:0.79
KNN mean cross validations score:0.69
DT mean cross validations score:0.48
SVM mean cross validations score:0.87
```

According to the cross validation scores Linear Discriminant Analysis and Logistic Regression still have best performance

In [59]:
```
lr=LinearDiscriminantAnalysis()
lr.fit(X_train_pca, y_train)
```

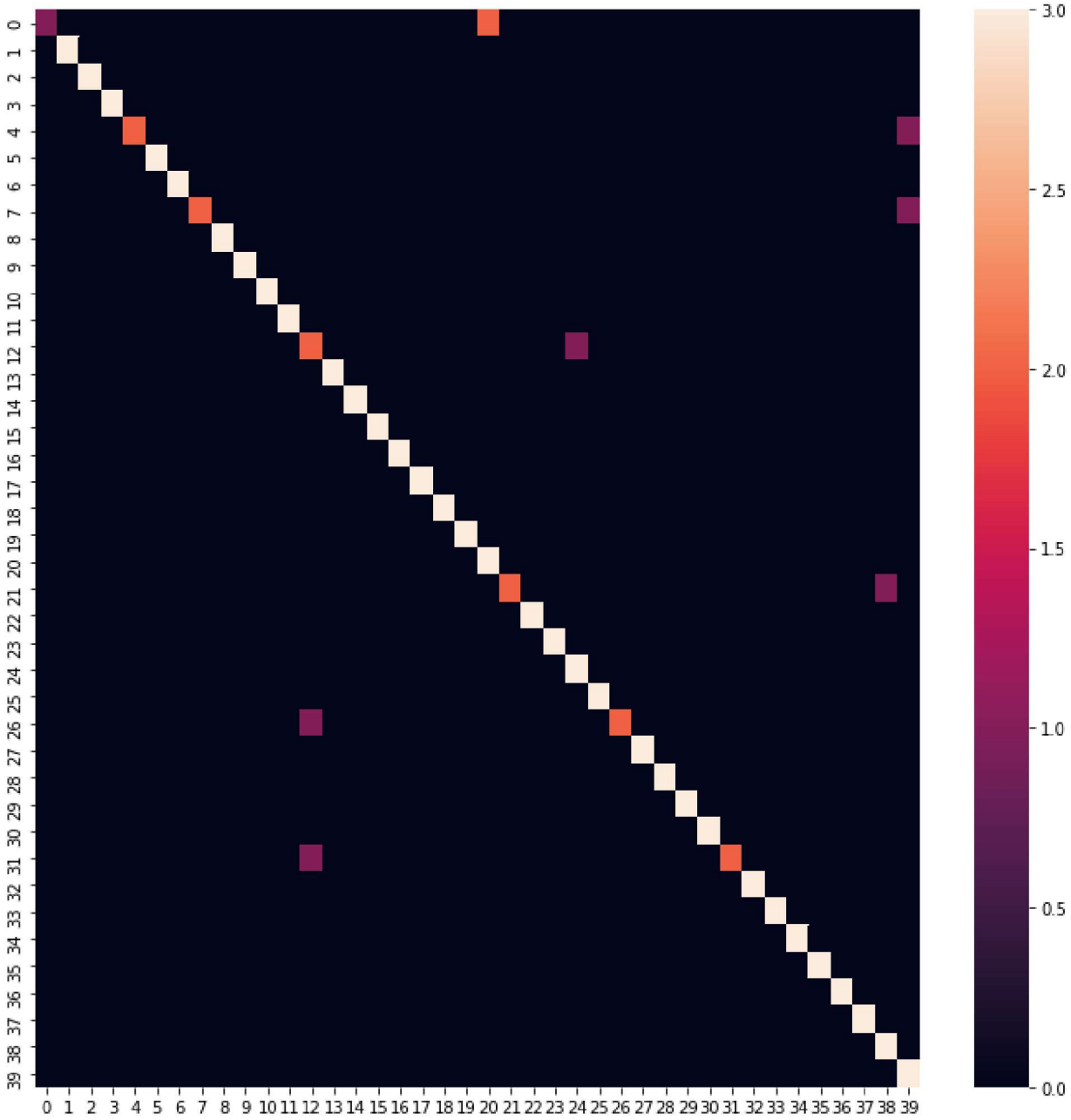
```
y_pred=lr.predict(X_test_pca)
print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
```

```
Accuracy score:0.93
```

In [60]:
```
cm=metrics.confusion_matrix(y_test, y_pred)

plt.subplots(1, figsize=(12,12))
sns.heatmap(cm)
```

Out[60]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6404bfc6a0>
```



In [61]:
```
print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pre
```

```
Classification Results:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         3
           3       1.00      1.00      1.00         3
           4       1.00      0.67      0.80         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         3
           7       1.00      0.67      0.80         3
           8       1.00      1.00      1.00         3
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         3
          12       0.50      0.67      0.57         3
          13       1.00      1.00      1.00         3
          14       1.00      1.00      1.00         3
          15       1.00      1.00      1.00         3
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         3
          18       1.00      1.00      1.00         3
          19       1.00      1.00      1.00         3
          20       0.60      1.00      0.75         3
          21       1.00      0.67      0.80         3
          22       1.00      1.00      1.00         3
          23       1.00      1.00      1.00         3
          24       0.75      1.00      0.86         3
          25       1.00      1.00      1.00         3
          26       1.00      0.67      0.80         3
          27       1.00      1.00      1.00         3
          28       1.00      1.00      1.00         3
          29       1.00      1.00      1.00         3
          30       1.00      1.00      1.00         3
          31       1.00      0.67      0.80         3
          32       1.00      1.00      1.00         3
          33       1.00      1.00      1.00         3
          34       1.00      1.00      1.00         3
          35       1.00      1.00      1.00         3
          36       1.00      1.00      1.00         3
          37       1.00      1.00      1.00         3
          38       0.75      1.00      0.86         3
          39       0.60      1.00      0.75         3

   micro avg       0.93      0.93      0.93       120
   macro avg       0.96      0.93      0.93       120
weighted avg       0.95      0.93      0.93       120
```

More Validated Results: Leave One Out cross-validation

The Olivetti dataset comprises 10 face images for each subject, which is a relatively small number for training and testing machine learning models. When dealing with limited examples per class, a recommended cross-validation method for more robust assessment of machine learning models is the 'Leave One Out' (LOO) cross-validation approach.

In the LOO methodology, only one sample per class is utilized for testing, while the remaining samples are allocated for training. This process iterates until each sample has been employed

for testing, ensuring a comprehensive evaluation of the model's performance.

In [62]:
```
from sklearn.model_selection import LeaveOneOut
loo_cv=LeaveOneOut()
clf=LogisticRegression()
cv_scores=cross_val_score(clf,
                          X_pca,
                          target,
                          cv=loo_cv)
print("{} Leave One Out cross-validation mean accuracy score:{:.2f}".format(clf.__clas
                                                                            cv_scores.
```

```
LogisticRegression Leave One Out cross-validation mean accuracy score:0.96
```

In [63]:
```
from sklearn.model_selection import LeaveOneOut
loo_cv=LeaveOneOut()
clf=LinearDiscriminantAnalysis()
cv_scores=cross_val_score(clf,
                          X_pca,
                          target,
                          cv=loo_cv)
print("{} Leave One Out cross-validation mean accuracy score:{:.2f}".format(clf.__clas
                                                                            cv_scores.
```

```
LinearDiscriminantAnalysis Leave One Out cross-validation mean accuracy score:0.98
```

Hyperparameter Tunning: GridSearcCV

To enhance the generalization performance of our model, we can employ GridSearchCV. This technique allows us to fine-tune the hyperparameters of the Logistic Regression classifier, thereby optimizing its performance and robustness.

In [64]:
```
from sklearn.model_selection import GridSearchCV
```

In [65]:
```
from sklearn.model_selection import LeaveOneOut

#This process takes long time. You can use parameter:{'C': 1.0, 'penalty': 'l2'}
#Grid search cross validation score:0.93
"""
params={'penalty':['l1', 'l2'],
                'C':np.logspace(0, 4, 10)
                }
clf=LogisticRegression()
#kfold=KFold(n_splits=3, shuffle=True, random_state=0)
loo_cv=LeaveOneOut()
gridSearchCV=GridSearchCV(clf, params, cv=loo_cv)
gridSearchCV.fit(X_train_pca, y_train)
print("Grid search fitted..")
print(gridSearchCV.best_params_)
print(gridSearchCV.best_score_)
print("grid search cross validation score:{:.2f}".format(gridSearchCV.score(X_test_pca
"""
```

Out[65]:
```
'\nparams={\'penalty\':[\'l1\', \'l2\'],\n               \'C\':np.logspace(0, 4, 10)
\n               }\nclf=LogisticRegression()\n#kfold=KFold(n_splits=3, shuffle=True,
random_state=0)\nloo_cv=LeaveOneOut()\ngridSearchCV=GridSearchCV(clf, params, cv=loo_
cv)\ngridSearchCV.fit(X_train_pca, y_train)\nprint("Grid search fitted..")\nprint(gri
dSearchCV.best_params_)\nprint(gridSearchCV.best_score_)\nprint("grid search cross va
lidation score:{:.2f}".format(gridSearchCV.score(X_test_pca, y_test)))\n'
```

In [66]:
```
lr=LogisticRegression(C=1.0, penalty="l2")
lr.fit(X_train_pca, y_train)
print("lr score:{:.2f}".format(lr.score(X_test_pca, y_test)))
```

```
lr score:0.93
```

Precision-Recall-ROC Curves

The precision-recall curves are primarily designed for binary classification tasks. However, in the Olivetti dataset, there exist 40 distinct classes, which deviates from the typical binary classification scenario. Nevertheless, it's reassuring to note that with the capabilities of sklearn, we can still visualize and analyze precision-recall metrics even in multi-label settings.

In [67]:
```
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier

Target=label_binarize(target, classes=range(40))
print(Target.shape)
print(Target[0])

n_classes=Target.shape[1]
```

```
(400, 40)
[1 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 0
 0 0 0]
```

In [68]:
```
X_train_multiclass, X_test_multiclass, y_train_multiclass, y_test_multiclass=train_tes
                                                                                      r
```

In [69]:
```
pca=PCA(n_components=n_components, whiten=True)
pca.fit(X_train_multiclass)

X_train_multiclass_pca=pca.transform(X_train_multiclass)
X_test_multiclass_pca=pca.transform(X_test_multiclass)
```

In [70]:
```
oneRestClassifier=OneVsRestClassifier(lr)

oneRestClassifier.fit(X_train_multiclass_pca, y_train_multiclass)
y_score=oneRestClassifier.decision_function(X_test_multiclass_pca)
```

In [71]:
```
# For each class
precision = dict()
recall = dict()
average_precision = dict()
for i in range(n_classes):
    precision[i], recall[i], _ = metrics.precision_recall_curve(y_test_multiclass[:, i
                                                                 y_score[:, i])
    average_precision[i] = metrics.average_precision_score(y_test_multiclass[:, i], y_
```

```
# A "micro-average": quantifying score on all classes jointly
precision["micro"], recall["micro"], _ = metrics.precision_recall_curve(y_test_multicl
    y_score.ravel())
average_precision["micro"] = metrics.average_precision_score(y_test_multiclass, y_scor
                                                             average="micro")
print('Average precision score, micro-averaged over all classes: {0:0.2f}'
      .format(average_precision["micro"]))
```

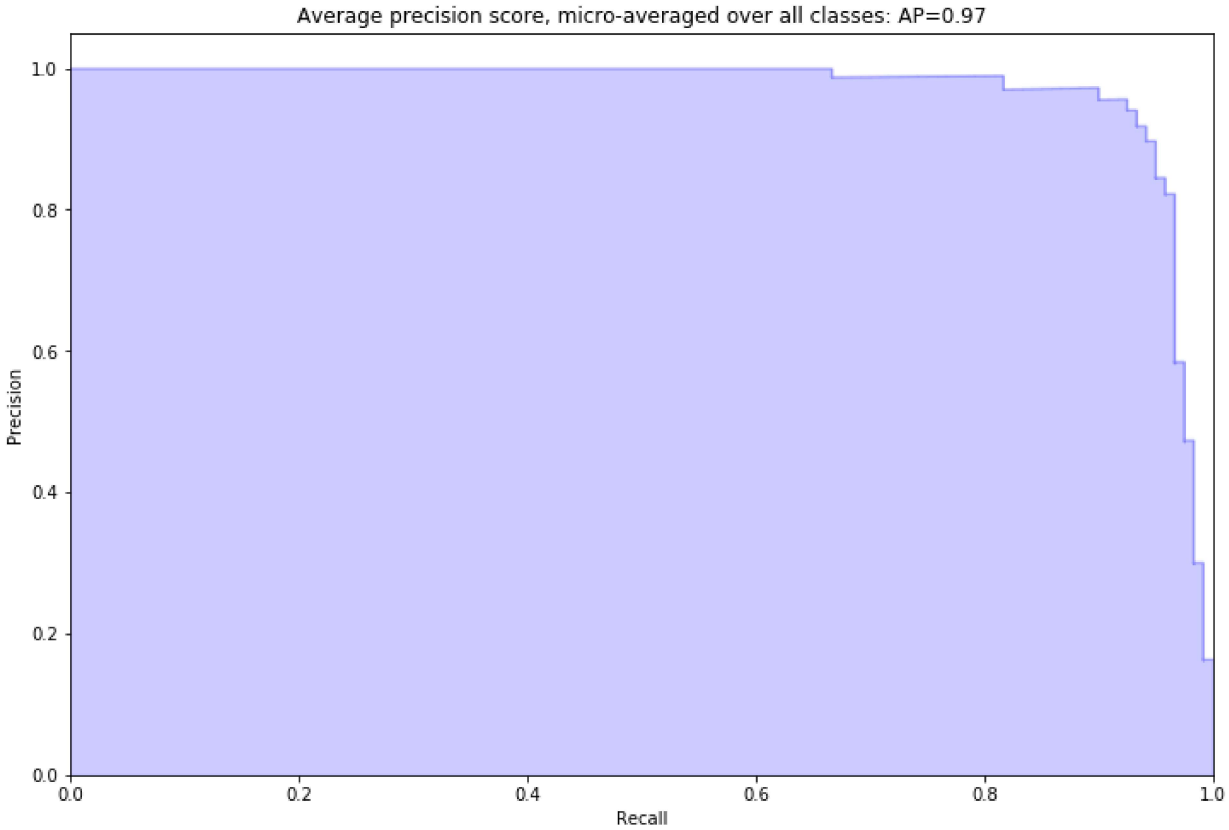
```
Average precision score, micro-averaged over all classes: 0.97
```

In [72]:
```
from sklearn.utils.fixes import signature

step_kwargs = ({'step': 'post'}
               if 'step' in signature(plt.fill_between).parameters
               else {})
plt.figure(1, figsize=(12,8))
plt.step(recall['micro'], precision['micro'], color='b', alpha=0.2,
         where='post')
plt.fill_between(recall["micro"], precision["micro"], alpha=0.2, color='b',
                 **step_kwargs)

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title(
    'Average precision score, micro-averaged over all classes: AP={0:0.2f}'
    .format(average_precision["micro"]))
```

Out[72]:
```
Text(0.5,1,'Average precision score, micro-averaged over all classes: AP=0.97')
```



Linear Discriminant Analysis İle Boyut Azaltma

In [73]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [74]:
```
lda = LinearDiscriminantAnalysis(n_components=n_components)
X_train_lda = lda.fit(X_train, y_train).transform(X_train)
X_test_lda=lda.transform(X_test)
```

In [75]:
```
lr=LogisticRegression(C=1.0, penalty="l2")
lr.fit(X_train_lda,y_train)
y_pred=lr.predict(X_test_lda)
```

In [76]:
```
print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pre
```

```
Accuracy score:0.94
Classification Results:
              precision    recall  f1-score   support

           0       0.50      0.33      0.40         3
           1       1.00      1.00      1.00         3
           2       0.60      1.00      0.75         3
           3       1.00      1.00      1.00         3
           4       1.00      1.00      1.00         3
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00         3
           7       1.00      0.67      0.80         3
           8       0.75      1.00      0.86         3
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         3
          12       1.00      0.67      0.80         3
          13       1.00      1.00      1.00         3
          14       1.00      1.00      1.00         3
          15       1.00      1.00      1.00         3
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         3
          18       1.00      1.00      1.00         3
          19       1.00      1.00      1.00         3
          20       1.00      1.00      1.00         3
          21       1.00      0.67      0.80         3
          22       1.00      1.00      1.00         3
          23       0.75      1.00      0.86         3
          24       1.00      1.00      1.00         3
          25       1.00      0.67      0.80         3
          26       1.00      0.67      0.80         3
          27       1.00      1.00      1.00         3
          28       1.00      1.00      1.00         3
          29       1.00      1.00      1.00         3
          30       1.00      1.00      1.00         3
          31       1.00      1.00      1.00         3
          32       1.00      1.00      1.00         3
          33       1.00      1.00      1.00         3
          34       1.00      1.00      1.00         3
          35       1.00      1.00      1.00         3
          36       1.00      1.00      1.00         3
          37       1.00      1.00      1.00         3
          38       0.75      1.00      0.86         3
          39       0.75      1.00      0.86         3

   micro avg       0.94      0.94      0.94       120
   macro avg       0.95      0.94      0.94       120
weighted avg       0.95      0.94      0.94       120
```

Machine Learning Automated Workflow: Pipeline

Machine learning applications typically follow a standardized workflow when dealing with datasets. To streamline this process, sklearn provides the Pipeline object, a powerful tool for automating this workflow. The Pipeline facilitates the implementation of standardized workflows for various machine learning operations, including scaling, feature extraction, and modeling.

One of the key advantages of using the Pipeline is its ability to ensure consistency throughout the entire dataset. By encapsulating multiple pre-processing steps and the model itself, the Pipeline guarantees that these operations are uniformly applied to both the training and test data. This uniform application is crucial for maintaining the integrity and consistency of the machine learning process across the dataset.

In [77]:
```
from sklearn.pipeline import Pipeline
```

In [78]:
```
work_flows_std = list()
work_flows_std.append(('lda', LinearDiscriminantAnalysis(n_components=n_components)))
work_flows_std.append(('logReg', LogisticRegression(C=1.0, penalty="l2")))
model_std = Pipeline(work_flows_std)
model_std.fit(X_train, y_train)
y_pred=model_std.predict(X_test)
```

In [79]:
```
print("Accuracy score:{:.2f}".format(metrics.accuracy_score(y_test, y_pred)))
print("Classification Results:\n{}".format(metrics.classification_report(y_test, y_pre
```

Accuracy score:0.94
Classification Results:

	precision	recall	f1-score	support
0	0.50	0.33	0.40	3
1	1.00	1.00	1.00	3
2	0.60	1.00	0.75	3
3	1.00	1.00	1.00	3
4	1.00	1.00	1.00	3
5	1.00	1.00	1.00	3
6	1.00	1.00	1.00	3
7	1.00	0.67	0.80	3
8	0.75	1.00	0.86	3
9	1.00	1.00	1.00	3
10	1.00	1.00	1.00	3
11	1.00	1.00	1.00	3
12	1.00	0.67	0.80	3
13	1.00	1.00	1.00	3
14	1.00	1.00	1.00	3
15	1.00	1.00	1.00	3
16	1.00	1.00	1.00	3
17	1.00	1.00	1.00	3
18	1.00	1.00	1.00	3
19	1.00	1.00	1.00	3
20	1.00	1.00	1.00	3
21	1.00	0.67	0.80	3
22	1.00	1.00	1.00	3
23	0.75	1.00	0.86	3
24	1.00	1.00	1.00	3
25	1.00	0.67	0.80	3
26	1.00	0.67	0.80	3
27	1.00	1.00	1.00	3
28	1.00	1.00	1.00	3
29	1.00	1.00	1.00	3
30	1.00	1.00	1.00	3
31	1.00	1.00	1.00	3
32	1.00	1.00	1.00	3
33	1.00	1.00	1.00	3
34	1.00	1.00	1.00	3
35	1.00	1.00	1.00	3
36	1.00	1.00	1.00	3
37	1.00	1.00	1.00	3
38	0.75	1.00	0.86	3
39	0.75	1.00	0.86	3
micro avg	0.94	0.94	0.94	120
macro avg	0.95	0.94	0.94	120
weighted avg	0.95	0.94	0.94	120