Computer Vision
Degree in Information Technology
2º Semester 2021/2022

Worksheet 8

Goals:

- Facial recognition with image classification methods

Exercises

Part I – Face Recognition using SVM and PCA

Faces are high dimensionality data consisting of several pixels. Data in high dimensionality is difficult to process and cannot be visualized using simple techniques like scatterplots for 2-dimensional data.

What we will do is to use PCA to reduce the high dimensionality of data and then feed it into the SVM classifier to classify the pictures.

Imports and Numpy arrays analysis

1. Import relevant libraries and modules:

```
import pylab as pl
import numpy as np
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA as RandomizedPCA
from sklearn.svm import SVC
```

2. Next, we will download the data into the disk and load it as NumPy arrays using fetch_lfw_people from sklearn.datasets:

```
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
```

3. The lfw dataset consists of a database of face photographs designed for studying the problem of unconstrained face recognition.
   The data set contains more than 13,000 images of faces collected from the web. Each face has been labelled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set.
   The images are in gray scale.

   a. Next, we will introspect the image arrays to find the shape of the pictures. We will use the NumPy shape attribute that returns a tuple with each index having the number of corresponding elements.

```
n_samples, h, w = lfw_people.images.shape
np.random.seed(42)
```

   b. As can be seen from the variable explorer, we have 1288 samples (pictures) with a height of 50 px and a width of 37 px (50 x 37 = 1850 features)

   c. We will use the data arrays in the lfw_people utils. Bunch directly and store it in X. We will use this data in our further processing.

```
X = lfw_people.data
n_features = X.shape[1]
```

   d. The data in X has 1288 samples and 1850 features for each sample.

Target & Target names -Labels

1. For image classification, we need to define the labels which are the id of the person to whom the picture belongs. Here, y represents the target which is the label of each picture. The label is further defined by the target_names variable which consists of the 7 names of the people to be identified.

```
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]
```

2. target is a 1288x1 NumPy array that contains the values 0–6 of the name the 1288 pictures correspond to. So, if the id 0 has a target value of 5, it would refer to “Hugo Chavez”. Do some prints to understand the structure.

3. Hence, y is the target in numerical form, target_names is any of the target/labels by the name, and n_classes is the variable that stores the number of classes we have, in our case, we have 7:
    a. Ariel Sharon
    b. Colin Powell
    c. Donald Rumsfeld
    d. George W Bush
    e. Gerhard Schröder
    f. Hugo Chavez
    g. Tony Blair

```
print("Total dataset size:")
print("n_samples: ", n_samples)
print("n_features: ", n_features)
print("n_classes: ", n_classes)
```

So, we have 1288 samples (pictures), a total of 1850 features for each sample (50px x 37px) and 7 classes (the people).

Splitting into Train/Test

1. train_test_split module from sklearn.model_selection splits the data (X-features and y-labels) into training data and testing data, with 25 % of the data used for testing and the remaining 75 % to train the model.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)
```

2. Analyse the content od X_train, X_test, y_train and y_test.

Dimensionality Reduction using PCA

1. PCA from sklearn.decomposition selects the top components to train the model. We have already imported PCA as RandomizedPCA in the first block of code.
   In our case, we have a total of 966 features in the training set X_train , and we will reduce them to 50 using PCA (dimensionality reduction).

```
n_components = 50
pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)
```

2. Now, it's time to transform on both X_train and X_test to reduce the dimensionality.

```
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

3. Analyse the content of X_train_pca and X_test_pca

Training the SVM Classifier

1. At this point, the data is ready for classification. As classification algorithm, we will apply Support Vector Machines.
   First, it is applied GridSearchCV, a library function that is an approach to hyperparameter tuning. It will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid, and return the best estimator in clf.best_estimator. The parameters are given in the param_grid:

```
print("Fitting the classifier to the training set")
param_grid = {
         'C': [1e3, 5e3, 1e4, 5e4, 1e5],
          'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1],
          }
model = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'),
param_grid)
model = model.fit(X_train_pca, y_train)
print("Best estimator found by grid search:")
print(model.best_estimator_)
```

Predictions

1. Let us now predict the people's names on the test data. We will use the classifier which we found from GridSearchCV and which we had already fit on the training data.

```
print("Predicting the people names on the testing set")
y_pred = model.predict(X_test_pca)
```

Classification Report & Confusion Matrix

1. Once the predictions are made, let us print the Classification Report which will display the precision, recall, F1-score, and support scores for the model. This gives a deeper intuition of the classifier's behaviour.

```
print(classification_report(y_test, y_pred, target_names=target_names))
```

Plotting

Finally, we will plot both the portraits of the people as well as the eigenfaces! We will define 2 functions: title to plot the result of the prediction on a portion of the test set, and plot_gallery to evaluate the prediction by plotting them:

```
def title(y_pred, y_test, target_names, i):
    pred_name = target_names[y_pred[i]].rsplit(' ', 1)[-1]
    true_name = target_names[y_test[i]].rsplit(' ', 1)[-1]
    return 'predicted: %s\ntrue:      %s' % (pred_name, true_name)
def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90,
hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())
```

Let us now plot the results of the prediction on a portion of the test set:

```
prediction_titles = [title(y_pred, y_test, target_names, i)
                         for i in range(y_pred.shape[0])]
plot_gallery(X_test, prediction_titles, h, w)
```

Lastly, let us plot the accuracy of the PCA + SVM Model for Facial Recognition:

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print(score)
```

Part II – Animal Classification using SVM and PCA

Using the Animal dataset build a model which can classify cats and dogs. Use the Images of case 1 to 4 to test your model. Include this implementation in your practical work.