Chapter 16: Demo PCA

In [1]:
```
from sklearn.datasets import fetch_openml
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
```

In [2]:
```
import datetime
x1 = datetime.datetime.now()
print(x1)
```

```
2020-10-14 15:38:52.152419
```

In [3]:
```
mnist = fetch_openml('mnist_784', version=1, cache=True ) # image 28x28 => 784 [
```

In [4]:
```
# mnist
```

In [5]:
```
mnist.data.shape
```

Out[5]:
```
(70000, 784)
```

In [6]:
```
mnist.target.shape
```

Out[6]:
```
(70000,)
```

In [7]:
```
# test_size: what proportion of original data is used for test set
train_img, test_img, train_lbl, test_lbl = train_test_split(
    mnist.data, mnist.target, test_size=1/7.0, random_state=0)
```

In [8]:
```
print(train_img.shape)
```

```
(60000, 784)
```

In [9]:
```
print(train_lbl.shape)
```

```
(60000,)
```

In [10]:
```
print(test_img.shape)
```

```
(10000, 784)
```

In [11]:
```
print(test_lbl.shape)
```

```
(10000,)
```

Standardizing the Data

Since PCA yields a feature subspace that maximizes the variance along the axes, it makes sense to standardize the data, especially, if it was measured on different scales.

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual feature do not more or less look like standard normally distributed data

Notebook going over the importance of feature Scaling: http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html#sphx-glr-auto-examples-preprocessing-plot-scaling-importance-py (http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html#sphx-glr-auto-examples-preprocessing-plot-scaling-importance-py)

In [12]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Fit on training set only.
scaler.fit(train_img)

# Apply transform to both the training set and the test set.
train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)
```

PCA to Speed up Machine Learning Algorithms (SVM)

Step 0: Import and use PCA. After PCA you will apply a machine learning algorithm of your choice to the transformed data

In [13]:
```
from sklearn.decomposition import PCA
```

In [14]:
```
# Make an instance of the Model
pca = PCA(.95)
```

In [15]:
```
# Fit PCA on training set. Note: you are fitting PCA on the training set only
pca.fit(train_img)
```

Out[15]:
```
PCA(copy=True, iterated_power='auto', n_components=0.95, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [16]:
```
pca.n_components_
```

Out[16]:
```
327
```

In [17]:
```
# Apply the mapping (transform) to both the training set and the test set.
train_img = pca.transform(train_img)
test_img = pca.transform(test_img)
```

Step 1: Import the model you want to use

In sklearn, all machine learning models are implemented as Python classes

In [18]:
```
from sklearn import svm
clf = svm.SVC(gamma=0.001, C=100) # các tham số cho mô hình hoạt động tốt hơn
```

Step 2: Training the model on the data, storing the information learned from the data

Model is learning the relationship between x (digits) and y (labels)

In [19]:
```
clf.fit(train_img,train_lbl)
```

Out[19]:
```
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [20]:
```
y_pred = clf.predict(test_img)
y_pred
```

Out[20]:
```
array(['0', '4', '1', ..., '1', '3', '0'], dtype=object)
```

Measuring Model Performance

Basically, how the model performs on new data (test set)

In [21]:
```
from sklearn.metrics import accuracy_score
print("Accuracy is ", accuracy_score(test_lbl,y_pred)*100,"%")
```

```
Accuracy is  97.38 %
```

In [22]:
```
score = clf.score(test_img, test_lbl)
print(score)
```

```
0.9738
```

Step 3: Predict the labels of new data (new images)

Uses the information the model learned during the model training process

In [23]:
```
new = clf.predict(test_img[0].reshape(1,-1))
new
```

Out[23]:
```
array(['0'], dtype=object)
```

In [24]:
```
x2 = datetime.datetime.now()
print(x2)
```

```
2020-10-14 15:44:41.782964
```

In [25]:
```
d = x2 - x1
print(d)
```

```
0:05:49.630545
```

In [26]:
```
import matplotlib.pyplot as plt
```

In [27]:
```
plt.figure(figsize=(8,6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

Out[27]:
```
Text(0, 0.5, 'Cumulative explained variance')
```