El Análisis de Componentes Principales (PCA) es un procedimiento estadístico que extrae las características más importantes de un conjunto de datos.

En un conjunto de puntos 2D. Cada dimensión corresponde a una característica que le interesa. Aquí algunos podrían argumentar que los puntos están establecidos en un orden aleatorio. Sin embargo, si observa mejor, verá que hay un patrón lineal.

Un punto clave del PCA es la Reducción de Dimensionalidad. La reducción de dimensionalidad es el proceso de reducir el número de dimensiones del conjunto de datos dado

PCA nos permite encontrar la dirección en la que nuestros datos varían más. De hecho, el resultado de ejecutar PCA en el conjunto de puntos del diagrama consta de 2 vectores llamados vectores propios , que son los componentes principales del conjunto de datos.

from \_\_future\_\_ import print\_function  
from \_\_future\_\_ import division  
import cv2 as cv  
import numpy as np  
import argparse  
from math import atan2, cos, sin, sqrt, pi  
  
#Visualizar resultado  
#El resultado final se visualiza a través de la siguiente funcion, donde los componentes principales se dibujan en líneas  
#y cada vector propio se multiplica por su valor propio y se traduce a la posición media.  
def drawAxis(img, p\_, q\_, colour, scale):  
 p = list(p\_)  
 q = list(q\_)  
   
 angle = atan2(p[1] - q[1], p[0] - q[0]) # angle in radians  
 hypotenuse = sqrt((p[1] - q[1]) \* (p[1] - q[1]) + (p[0] - q[0]) \* (p[0] - q[0]))  
 # Here we lengthen the arrow by a factor of scale  
 q[0] = p[0] - scale \* hypotenuse \* cos(angle)  
 q[1] = p[1] - scale \* hypotenuse \* sin(angle)  
 cv.line(img, (int(p[0]), int(p[1])), (int(q[0]), int(q[1])), colour, 1, cv.LINE\_AA)  
 # create the arrow hooks  
 p[0] = q[0] + 9 \* cos(angle + pi / 4)  
 p[1] = q[1] + 9 \* sin(angle + pi / 4)  
 cv.line(img, (int(p[0]), int(p[1])), (int(q[0]), int(q[1])), colour, 1, cv.LINE\_AA)  
 p[0] = q[0] + 9 \* cos(angle - pi / 4)  
 p[1] = q[1] + 9 \* sin(angle - pi / 4)  
 cv.line(img, (int(p[0]), int(p[1])), (int(q[0]), int(q[1])), colour, 1, cv.LINE\_AA)  
   
  
#Extraer orientación  
#La orientación se extrae mediante la llamada a la siguiente funcion, que realiza todo el procedimiento PCA.   
def getOrientation(pts, img):  
   
 sz = len(pts)  
 data\_pts = np.empty((sz, 2), dtype=np.float64)  
 for i in range(data\_pts.shape[0]):  
 data\_pts[i,0] = pts[i,0,0]  
 data\_pts[i,1] = pts[i,0,1]  
 # Perform PCA analysis  
 mean = np.empty((0))  
 mean, eigenvectors, eigenvalues = cv.PCACompute2(data\_pts, mean)  
 # Store the center of the object  
 cntr = (int(mean[0,0]), int(mean[0,1]))  
   
   
 cv.circle(img, cntr, 3, (255, 0, 255), 2)  
 p1 = (cntr[0] + 0.02 \* eigenvectors[0,0] \* eigenvalues[0,0], cntr[1] + 0.02 \* eigenvectors[0,1] \* eigenvalues[0,0])  
 p2 = (cntr[0] - 0.02 \* eigenvectors[1,0] \* eigenvalues[1,0], cntr[1] - 0.02 \* eigenvectors[1,1] \* eigenvalues[1,0])  
 drawAxis(img, cntr, p1, (0, 255, 0), 1)  
 drawAxis(img, cntr, p2, (255, 255, 0), 5)  
 angle = atan2(eigenvectors[0,1], eigenvectors[0,0]) # orientation in radians  
   
 return angle  
#Primero, los datos deben organizarse en una matriz con tamaño nx 2, donde n es el número de puntos de datos que tenemos.  
#Entonces podemos realizar ese análisis PCA. La media calculada (es decir, el centro de masa) se almacena en la variable cntr y los vectores propios   
#y valores propios se almacenan en los vectores correspondientes.  
  
  
  
#Leer imagen y convertirla a binario.  
#Aquí aplicamos los procedimientos de preprocesamiento necesarios para poder detectar los objetos de interés  
src = cv.imread(cv.samples.findFile("resources/pca\_test1.jpg"))  
cv.imshow('src', src)  
# Convert image to grayscale  
gray = cv.cvtColor(src, cv.COLOR\_BGR2GRAY)  
# Convert image to binary  
\_, bw = cv.threshold(gray, 50, 255, cv.THRESH\_BINARY | cv.THRESH\_OTSU)  
  
#Extraer objetos de interés  
#Luego busque y filtre contornos por tamaño y obtenga la orientación de los restantes.  
contours, \_ = cv.findContours(bw, cv.RETR\_LIST, cv.CHAIN\_APPROX\_NONE)  
  
  
for i, c in enumerate(contours):  
 # Calculate the area of each contour  
 area = cv.contourArea(c)  
 # Ignore contours that are too small or too large  
 if area < 1e2 or 1e5 < area:  
 continue  
 # Draw each contour only for visualisation purposes  
 cv.drawContours(src, contours, i, (0, 0, 255), 2)  
 # Find the orientation of each shape  
 getOrientation(c, src)  
cv.imshow('output', src)  
cv.waitKey()