La mayoría de las aplicaciones en las que se utilizan SVM en visión por computadora requieren una herramienta más poderosa que un simple clasificador lineal. Esto se debe al hecho de que en estas tareas los datos de entrenamiento rara vez se pueden separar mediante un hiperplano.

El problema que presenta, por ejemplo, es la detección de rostros. Los datos de entrenamiento en este caso están compuestos por un conjunto de imágenes que son caras y otro conjunto de imágenes que no son caras ( cualquier otra cosa en el mundo excepto caras ). Estos datos de entrenamiento son demasiado complejos como para encontrar una representación de cada muestra ( vector de características ) que pueda hacer que todo el conjunto de caras sea linealmente separable del conjunto completo de no caras.

El nuevo modelo debe incluir tanto el antiguo requisito de encontrar el hiperplano que proporcione el mayor margen como el nuevo de generalizar correctamente los datos de entrenamiento sin permitir demasiados errores de clasificación.

from \_\_future\_\_ import print\_function  
import cv2 as cv  
import numpy as np  
import random as rng  
  
NTRAINING\_SAMPLES = 100 # Number of training samples per class  
FRAC\_LINEAR\_SEP = 0.9 # Fraction of samples which compose the linear separable part  
  
# Data for visual representation  
WIDTH = 512  
HEIGHT = 512  
I = np.zeros((HEIGHT, WIDTH, 3), dtype=np.uint8)  
  
# --------------------- 1. Set up training data randomly ---------------------------------------  
  
trainData = np.empty((2\*NTRAINING\_SAMPLES, 2), dtype=np.float32)  
labels = np.empty((2\*NTRAINING\_SAMPLES, 1), dtype=np.int32)  
rng.seed(100) # Random value generation class  
  
# Set up the linearly separable part of the training data  
#Los datos de entrenamiento de este ejercicio están formados por un conjunto de puntos 2D etiquetados  
#que pertenecen a una de dos clases diferentes. Para hacer que el ejercicio sea más atractivo,   
#los datos de entrenamiento se generan aleatoriamente utilizando funciones de densidad de probabilidad uniforme (PDF)  
  
#Hemos dividido la generación de los datos de entrenamiento en dos partes principales.  
#En la primera parte generamos datos para ambas clases que son linealmente separables.  
  
nLinearSamples = int(FRAC\_LINEAR\_SEP \* NTRAINING\_SAMPLES)  
trainClass = trainData[0:nLinearSamples,:]  
# The x coordinate of the points is in [0, 0.4)  
c = trainClass[:,0:1]  
c[:] = np.random.uniform(0.0, 0.4 \* WIDTH, c.shape)  
# The y coordinate of the points is in [0, 1)  
c = trainClass[:,1:2]  
c[:] = np.random.uniform(0.0, HEIGHT, c.shape)  
  
  
  
# Generate random points for the class 2  
trainClass = trainData[2\*NTRAINING\_SAMPLES-nLinearSamples:2\*NTRAINING\_SAMPLES,:]  
# The x coordinate of the points is in [0.6, 1]  
c = trainClass[:,0:1]  
c[:] = np.random.uniform(0.6\*WIDTH, WIDTH, c.shape)  
# The y coordinate of the points is in [0, 1)  
c = trainClass[:,1:2]  
c[:] = np.random.uniform(0.0, HEIGHT, c.shape)  
#------------------ Set up the non-linearly separable part of the training data ---------------  
  
#En la segunda parte creamos datos para ambas clases que no son separables linealmente, datos que se superponen.  
  
trainClass = trainData[nLinearSamples:2\*NTRAINING\_SAMPLES-nLinearSamples,:]  
  
# The x coordinate of the points is in [0.4, 0.6)  
c = trainClass[:,0:1]  
c[:] = np.random.uniform(0.4\*WIDTH, 0.6\*WIDTH, c.shape)  
  
# The y coordinate of the points is in [0, 1)  
c = trainClass[:,1:2]  
c[:] = np.random.uniform(0.0, HEIGHT, c.shape)  
  
#------------------------- Set up the labels for the classes ---------------------------------  
labels[0:NTRAINING\_SAMPLES,:] = 1 # Class 1  
labels[NTRAINING\_SAMPLES:2\*NTRAINING\_SAMPLES,:] = 2 # Class 2  
  
#------------------------ 2. Set up the support vector machines parameters --------------------  
#Set up Model  
print('Starting training process')  
svm = cv.ml.SVM\_create()  
svm.setType(cv.ml.SVM\_C\_SVC)  
svm.setC(0.1)  
svm.setKernel(cv.ml.SVM\_LINEAR)  
svm.setTermCriteria((cv.TERM\_CRITERIA\_MAX\_ITER, int(1e7), 1e-6))  
  
#------------------------ 3. Train the svm ----------------------------------------------------  
# Train model  
svm.train(trainData, cv.ml.ROW\_SAMPLE, labels)  
print('Finished training process')  
  
#------------------------ 4. Show the decision regions ----------------------------------------  
green = (0,100,0)  
blue = (100,0,0)  
for i in range(I.shape[0]):  
 for j in range(I.shape[1]):  
 sampleMat = np.matrix([[j,i]], dtype=np.float32)  
 response = svm.predict(sampleMat)[1]  
 if response == 1:  
 I[i,j] = green  
 elif response == 2:  
 I[i,j] = blue  
#----------------------- 5. Show the training data --------------------------------------------  
thick = -1  
# Class 1  
for i in range(NTRAINING\_SAMPLES):  
 px = trainData[i,0]  
 py = trainData[i,1]  
 cv.circle(I, (int(px), int(py)), 3, (0, 255, 0), thick)  
# Class 2  
for i in range(NTRAINING\_SAMPLES, 2\*NTRAINING\_SAMPLES):  
 px = trainData[i,0]  
 py = trainData[i,1]  
 cv.circle(I, (int(px), int(py)), 3, (255, 0, 0), thick)  
#------------------------- 6. Show support vectors --------------------------------------------  
thick = 2  
sv = svm.getUncompressedSupportVectors()  
for i in range(sv.shape[0]):  
 cv.circle(I, (int(sv[i,0]), int(sv[i,1])), 6, (128, 128, 128), thick)  
cv.imwrite('resources/result1.png', I) # save the Image  
cv.imshow('SVM for Non-Linear Training Data', I) # show it to the user  
cv.waitKey()

Starting training process  
Finished training process

27

The Kernel crashed while executing code in the current cell or a previous cell.   
  
Please review the code in the cell(s) to identify a possible cause of the failure.   
  
Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.   
  
View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.