Práctica 2: clasificación de dígitos MNIST mediante AdaBoost

ELVI MIHAI SABAU SABAU | 51254875L

Introducción

Importar librerías

```
import numpy as np
import matplotlib.pyplot as plt
import random
from keras.datasets import mnist
```

Desde la libería de Keras podemos descargar la base datos MNIST

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-data
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
x_train.shape,y_train.shape
```

```
((60000, 28, 28), (60000,))
```

Guardar base de datos en un fichero

```
np.savez("mnist",x=x_train,y=y_train)
```

Cargar base de datos desde un fichero

```
npzfile = np.load("mnist.npz")
mnist_X = npzfile['x']
mnist_Y = npzfile['y']
mnist_X.shape
```

```
(60000, 28, 28)
```

Implementación AdaBoost

Funciones auxiliares, de normalización, y adaptación de los datasets.

```
# Funcion para imprimir graficas rapidamente
def plot_arrays(X, Y, title, xlabel, ylabel):
    plt.title(title)
    plt.plot(X, Y, "bo")
    plt.xlabel(ylabel)
    plt.ylabel(xlabel)
    plt.show()


# Funcion para normalizar el conjunto de imagenes de vector 2D a 1D
def normalizar_conjunto(X, dim_imagen):
  return X.reshape(len(X), dim_imagen)


# El conjunto de entrenamiento para una clase ( tipo de imágen ) debe ser: 50% imá
def generar_conjunto_entrenamiento(N, clase, x_train, y_train):

  # N: Total de imágenes del conjunto de entrenamiento, debe ser para para partir !
  if(N % 2 != 0):
    raise Exception("N no es Par")

  # Mitad de la clase, Mitad no clase.
  N = int(N/2)

  # Creamos una máscara para todo el conjunto de entrenamiento
  mask = y_train == clase

  # Partimos en 2 el dataset, los elementos que son de la clase, y los que no lo s
  arr_clase = x_train[mask]
  arr_noClase = x_train[~mask]

  # Concatenamos dichos arrays de imagenes, y aparte creamos Y, un array de 1 y -1
  X = np.concatenate((arr_clase[0:N], arr_noClase[0:N]))
  X = normalizar_conjunto(X, 28 * 28)

  Y = np.concatenate((np.ones(N), np.ones(N) * -1))

  return (X, Y)
```

Funciones necesarias para los clasificadores débiles

```
# Dimensión con la que vamos a trabajar. En nuestro caso 28*28
def generar_clasificador_debil(dim_imagen):
  ht = np.zeros(3)
  ht[0] = np.random.randint(0, dim_imagen)     # Num. Pixel
  ht[1] = random.randint(0, 255)               # Valor Umbral Grises
  ht[2] = np.sign(np.random.rand(1) * 2-1)     # Valor del ubral, miramos por enci
  return ht;
```

```
def aplicar_clasificador_debil(ht, X):

  # Por cada imágen de X, comprobamos si el pixel de dicha imágen tiene un valor ma
  #   si no lo es, en vez de descartarlo, le podemos dar la vuelta, y reusarlo.
  if(ht[2] == 1):
    mascara_resultados = X[:, int(ht[0])] > ht[1]
  else:
    mascara_resultados = X[:, int(ht[0])] <= ht[1]

  # Usamos int para pasar de true y false a 1 y 0
  resultados = np.int16(mascara_resultados)

  # Pasamos de 0 a -1
  resultados[np.where(resultados == 0)] = -1

  return resultados


def obtener_error(resultados, Y, D):

  # -1 si no es, 1 si es, y con int64 pasamos los true y falses resultantes a 1 y (
  errv = np.int64(resultados != Y)
  sumerr = np.float64(np.sum(errv * D))

  return [sumerr, errv]


def adaboost(X, Y, T, A):

  # Vector de error, se actualiza por cada clasificador debil, inicialmente uniforn
  D = np.ones(len(X), dtype=np.double) / len(X)

  # Datos para la gráfica.
  pixel = []
  error = []

  # Array donde guardaremos tuplas de [mejor_clasificador, confianza] = [ht, alpha
  H = []

  for t in range(T):

    # Menor suma de errores del clasificador debil y su array de errores.
    menor_sumerr = np.inf
    menor_errv = []

    # Mejor clasificador debil ( de A pruebas aleatorias, el que menor error tiene
    mejor_ht = None

    # Probamos A clasificadores debiles, y nos quedamos con el mejor de todos.
    for k in range(A):

      ht = generar_clasificador_debil(28 * 28)
      resultados_entrenamiento = aplicar_clasificador_debil(ht, X)
      [sumerr, errv] = obtener_error(resultados_entrenamiento, Y, D)
```

```
        if(sumerr < menor_sumerr):
          mejor_ht = ht
          menor_sumerr = sumerr
          menor_errv = errv

        """if(menor_sumerr > 0.45):
          A = A + 1"""

      # Calculamos la confianza.
      alpha = np.float64(0.5 * np.log2((1.0 - menor_sumerr) / menor_sumerr))

      # Arrays de clasificadores y sus confianzas.
      H.append([mejor_ht, alpha])

      # Recalculamos D y normalizamos.
      # print((alpha, menor_sumerr, np.sum(D)))
      Z = np.sum(D)
      exp = np.float64(np.exp(-alpha * -np.float64(menor_errv)))
      D = np.float64((D * exp) / Z)

      # Datos para gráfica.
      pixel.append(mejor_ht[0])
      error.append(menor_sumerr)

    # Un plot chulo de los mejores debiles
    plot_arrays(error, pixel, "Error de cada Clasificador por Pixel", "Grado de Erro

    print("Clasificadores de T:")
    print(H)

    return H
```

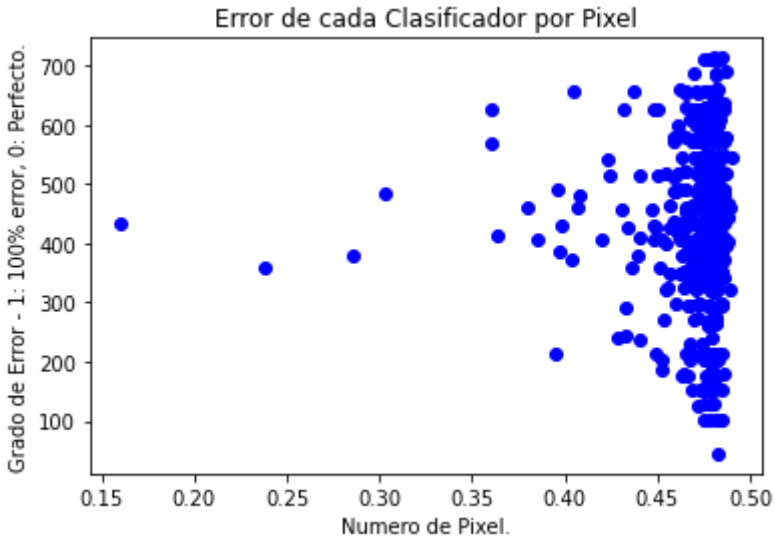
Definimos una funcion para aplicar el clasificador fuerte a un conjunto de imágnes.

```
def aplicar_clasificador_fuerte(H, X):
  certeza = []

  # Por cada clasificador debil ht del cojunto de fuertes h, lo aplicamos a todas
  for [ht, alpha] in H:

    # POR CADA IMAGEN SE PASA TODAS LAS ht
    resultados = aplicar_clasificador_debil(ht, X)

    if (len(certeza) == 0):
      certeza = alpha * resultados
    else:
      certeza = certeza + (alpha * resultados)

  f = np.sign(certeza)
  return f
```

Hito 1 - Entrenar para la clase "0"

Lanzar entrenamiento

```
# Cantidad de imágenes. Si pondemos demasiadas imagenes (+10000), puede que se de
N = 10000

# Clase ( tipo imágen en este caso de numero ) sobre el cual entrenar.
clase = 0

# X = Array de N imágenes por 748 slots (imágen aplanada)
# Y = Array de tags relacionados con cada imágen.
# Recibe, N: Numero de imágenes, clase: La clase sobre la cual entrenar, x_train, y
(X, Y) = generar_conjunto_entrenamiento(N, clase, x_train, y_train)

# Cantidad e clasificadores debiles que compondran un clasificador fuerte.
T = 500

# Cantidad de clasificadores debiles a probar para extraer el mejor de ellos.
A = 1000

H0 = adaboost(X, Y, T, A)
```




```
Clasificadores de T:
[[array([435.,  71.,  -1.]), 1.194014012418617], [array([358.,  20.,   1.])
```

Probamos el clasificador fuerte entrenado.

```
# N = Cantidad de numeros a clasificar.
N = 10000

# Normalizamos las imágenes.
X = normalizar_conjunto(x_test[0:N], 28 * 28)
```

```
# Comparamos entre el resultado del clasificador fuerte, y los tags de Y.
ceros_detectados = np.int16(aplicar_clasificador_fuerte(H0, X))
ceros_detectados_sumados = np.sum(np.int64(ceros_detectados[:] == 1))

ceros_reales = y_test[0:N]
ceros_reales_sumados = np.sum(np.int64(ceros_reales == clase))


print("Numeros detectados: " + str(ceros_detectados_sumados) + ", Numeros reales:
print(str((ceros_detectados_sumados / ceros_reales_sumados) * 100) + " % de certeza
print("----------------------")
print(list(zip(ceros_detectados, ceros_reales)))
```

```
Numeros detectados: 1209, Numeros reales: 980
123.36734693877551 % de certeza.
----------------------
[(-1, 7), (-1, 2), (-1, 1), (1, 0), (-1, 4), (-1, 1), (-1, 4), (-1, 9), (-1,
```

✓ 0 s completado a las 10:36