Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Tarea 4

Detección de Personas usando Adaboost y características tipo Haar.

Integrantes: Joaquín Zepeda
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla
Santiago de Chile

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Índice de Códigos

1. Introducción

La detección de personas y de rostros no es algo simple en el ámbito del procesamiento de imágenes, existen múltiples técnicas que buscan reconocer caras y personas las cuales son utilizadas en seguridad, universidades, lugares públicos, etc. Las características extraídas de la imagen son importantes para el reconocimiento de estas, por ejemplo cara, brazos, torso, etc. En esta tarea se implementara un extractor de características Haar los cuales corresponden a una secuencia de mascaras de diferentes tamaños los cuales realizan convoluciones con la imagen en diferentes sectores de esta, sumando asi los pixeles en regiones cubiertas por estas mascaras. Luego se implementará desde cero un clasificador Adabooost sin usar cascada para poder poner a prueba tanto la efectividad de las características de tipo Haar para clasificar como para probar la rapidez y efectividad del clasificador Adaboost. [1]
El objetivo de esta tarea es diseñar y construir desde cero un sistema de detección de personas que utilice características de tipo Haar y un clasificador Adaboost, esto con el fin de poner en practica los conceptos vistos en clases, poner en practica las habilidades de programación para resolver un problema real y poder apreciar en diferentes situaciones los resul- tados de estos métodos y de los distintos clasificadores. Para esto se utilizarán 3 tipos de imágenes: imágenes con personas, imágenes con autos e imágenes con sillas, las cuales se agruparan en 2 conjuntos, imágenes sin personas e imágenes con personas.
Esta tarea se desarrolla utilizando el lenguaje de programación Python y además se completan funciones en Cython, lo cual nos permite escribir código en C/C++ desde Python permitiendo tener un programa más rápido, a continuación se describe el procedimiento de la tarea y se muestran y analizan los resultados en la sección de Desarrollo para luego finalizar con las conclusiones.

2. Desarrollo

2.1. Preparación de Conjuntos de Entrenamiento y Prueba

1. Base de datos: Para las tareas de entrenamiento y prueba se utilizan las imágenes de la base de datos subida a U-Cursos (370 imágenes), la cual incluye 185 imágenes con personas y 185 imágenes sin personas (sillas y autos). Esta base de datos se separa en 80 % para entrenamiento y 20 % para prueba utilizando una semilla en la función train_test_split() para generar resultados repetibles. Esto se realiza con el parámetro random_state, además se mezcla el conjunto al utilizar el parámetro shuffle = True.

```
    # Conjuntos de train y test
#random_state = 42 semilla fija para resultados reproducibles.
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, shuffle = True
    ↪ , stratify = labels,random_state=42)

```

2. Imagen Integral: Una imagen integral corresponde a una matriz en la cual cada pixel se calcula a partir de la suma de los valores de los pixeles anteriores, esto se observa en la matriz del centro de la figura 1, esta corresponde a la imagen integral de la imagen representada por la matriz de la izquierda de esta figura. Sea *Img* la imagen original y *IImg* la imagen integral de esta, *IImg* se puede calcular de forma constructiva de la siguiente forma:

$$IImg[i][j] = Img[i][j] + IImg[i-1][j] + IImg[i][j-1] - IImg[i-1][j-1] \quad (1)$$

La ventaja de utilizar la imagen integral es que se puede determinar la suma en una determinada zona rectangular a partir de 2 sumas y 2 restas, lo cual es rápido y eficiente. Esto se puede observar en la figura 1, el la imagen de la derecha muestra un ejemplo de como se calculan los bloques,se determina la suma del rectángulo D de la siguiente forma:

$$D = esquina_{inferior\_der} + esquina_{superior\_izq} - esquina_{inferior\_izq} - esquina_{superior\_der} \quad (2)$$

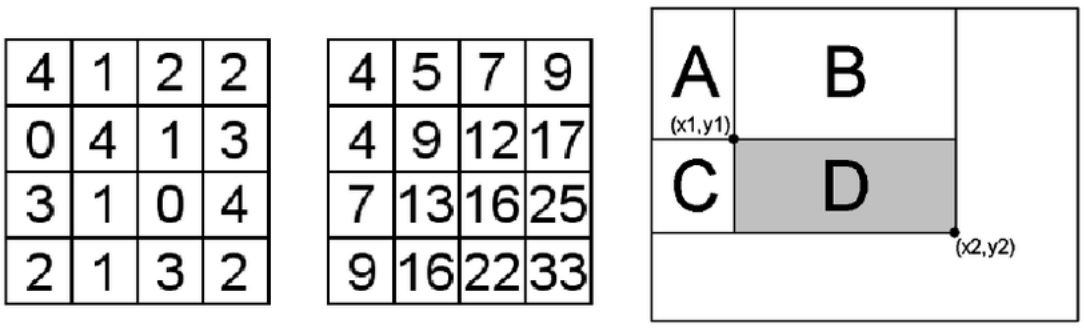


Figura 1: Ejemplo de imagen integral.

Esta función se muestra a continuación, para optimizar los tiempos de ejecución se utilizó

Cython.

```
%%cython

import numpy as np
cimport numpy as np

cpdef np.ndarray[np.float32_t,ndim=2] integral(np.ndarray[np.float32_t,ndim=2] img):
    """
    2. función en Cython que permita calcular la imagen integral, dada una imagen.
    """
    cdef int rows,cols,i,j
    cdef np.ndarray[np.float32_t, ndim=2] output=np.zeros([img.shape[0], img.shape[1]],
    dtype = np.float32)
    # tamano de la imagen
    rows = img.shape[0]
    cols = img.shape[1]
    #Como se van guardando los valores en la imagen integral, por construcción
    #se puede realizar con 4 operaciones (usando valores precomputados), de esta
    #manera se evita realizar sumas repetitivas.

    output[0][0] = img[0][0] #caso base
    for i in range(rows):
      for j in range(cols):
        #Casos borde
        if i==0:
            output[i][j] = output[i][j-1] + img[i][j]
        elif j==0:
            output[i][j] = output[i-1][j] + img[i][j]
        else:
            output[i][j] = img[i][j]+output[i-1][j]+output[i][j-1]-output[i-1][j-1]
    return output

```

Código 1: Función imagen integral Cython.

2.2. Características tipo Haar

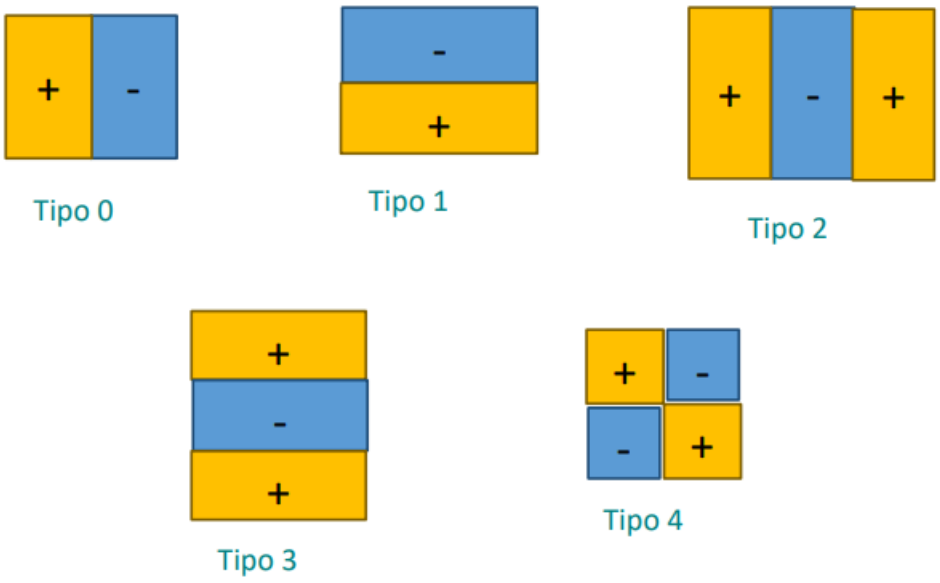


Figura 2: Máscaras Haar/rectangulares a ser usados en esta tarea. Las máscaras mostradas están asociadas a características con polaridad positiva. En el caso de la polaridad negativa, los signos asociados a las máscaras se invierten

3. Parámetros que determinan las mascaras de las características de tipo Haar:
   Estos se parametrizan como: (y1,x1,y2,x2,tipo,polaridad), donde:

   a. (y1,x1) es la esquina superior izquierda de la máscara.

   b. (y2,x2) es la esquina inferior derecha de la máscara.

   c. Tipo: número entre 0 y 4, que permite determinar el tipo de máscara.

   d. Polaridad: +1 (positiva) o –1 (negativa).

   Se utilizaron 3 valores para los anchos y 3 valores para los altos como se muestra en la tabla 1. Esto arroja 9 combinaciones de mascaras por cada tipo, las cuales se mueven dentro de la imagen, de pendiendo del tamaño de la mascara es el número de parámetros que tiene cada tipo en cada combinación. A partir de esto se genera una lista de parámetros de largo 3030. Cabe destacar que se cumple lo siguiente: $x2 = x1 + ancho, y2 = y1 + alto$.

Tabla 1: Tabla de anchos y largos de las mascaras por tipo.

	Tipo 0	**Tipo 1**	**Tipo 2**	**Tipo 3**	**Tipo 4**	**Tipo 5**
Anchos	**[4,8,12]**	**[4,8,12]**	**[3,6,9]**	**[3,6,9]**	**[4,8,12]**	**[4,8,12]**
Altos	**[4,8,12]**	**[4,8,12]**	**[3,6,9]**	**[3,6,9]**	**[4,8,12]**	**[4,8,12]**

```
1 def gen_parameters():
```

```
    L = []
    tipos = [   0,          1,      2,          3,      4]
    tipo2y3 = [3,6,9]
    anchos = [[4,8,12], [4,8,12], tipo2y3, tipo2y3,[4,8,12]]
    altos = [[4,8,12], [4,8,12], tipo2y3, tipo2y3,[4,8,12]]
    #generamos los parametros por cada tipo, sus 9 combinaciones
    #y además recorremos con una ventana deslizante de paso 3 con el fin
    #de determinar todas las mascaras que queden dentro de la imagen.
    for tipo in tipos:
          for y1 in range(0,24,3):
              for x1 in range(0,24,3):
                  for A in anchos[tipo]:
                      for B in altos[tipo]:
                          if y1+B<24 and x1+A<24:
                              #agregamos las mascaras en sus 2 polaridades
                              L.append([y1,x1,y1+B,x1+A,tipo,0])
                              L.append([y1,x1,y1+B,x1+A,tipo,1])
    return L
```

4. Extracción de características: Para la extracción de características se generó la función Haar(), la cual por cada imagen determina la imagen integral y luego dependiendo del tipo de la mascara se calcula el valor de la mascara aplicada en la imagen según las posiciones que indica la mascara y la polaridad, los diferentes tipos de mascaras se pueden observar en la figura 2. Por cada cuadrado de la mascara (zonas positivas y negativas) correspondiente se realizan 2 sumas y 2 restas como se menciono anteriormente en el item de imagen integral.

```
    def Haar(img_set,mascaras):
        """
        Función que, dado un conjunto de imágenes y parámetros de máscaras, calcule vectores de
     ↪  características.Dichos vectores deben contener todas las características tipo Haar
     ↪ determinadas en el punto anterior, para cada imagen. Por cada imagen, determinamos la
     ↪  imagen integral y luego dependiendo del tipo de la mascara se calcula el valor de la
     ↪ mascara aplicada en la imagen según las posiciones que indica la mascara."""
        feature_set = []
        for img in img_set:
            features = np.zeros(len(mascaras))
            #determinamos la imagen integral
            I = integral(img)
            for index,mask in enumerate(mascaras):
                #rescatamos los parametros de la mascara
                y1,x1,y2,x2,tipo,p = mask
                A = x2-x1 #ancho
                B = y2-y1 #alto
                if tipo==0:
                    d = x1+A//2
                    features[index]=p*(( I[y2][d]+I[y1][x1]-I[y2][x1]-I[y1][d] )
                                -( I[y2][x2] + I[y1][d]-I[y2][d]-I[y1][x2]))
                elif tipo==1:
                    d = y1+B//2
                    features[index]=p*(( I[y2][x2]+I[d][x1]-I[y1][x2]-I[d][x2] )
```

```
                        -( I[d][x2] + I[y1][x1]-I[d][x1]-I[y1][x2]))
            elif tipo==2:
                d1 = x1+A//3
                d2 = x1+2*A//3
                features[index]=p*(  (I[y2][d1]+I[y1][x1]-I[y1][d1]-I[y2][x1])
                                - ( I[y2][d2]+I[y1][d1]-I[y1][d2]-I[y2][d1])
                                + ( I[y2][x2] + I[y1][d2] - I[y2][d2]-I[y1][x2]))
            elif tipo==3:
                d1 = y1+B//3
                d2 = y1+2*B//3
                features[index]=p*( (I[d1][x2] + I[y1][x1] - I[d1][x1] - I[y1][x2])
                                -(I[d2][x2] + I[d1][x1] - I[d2][x1] - I[d1][x2])
                                +(I[y2][x2] + I[d2][x1] - I[y2][x1] - I[d2][x2] ))
            else:
                dy = y1+B//2
                dx = x1+A//2
                features[index]=p*( (I[dy][dx] + I[y1][x1] - I[dy][x1] - I[y1][dx])
                            - (  I[dy][x2] + I[y1][dx] - I[dy][dx] - I[y1][x2])
                            - (  I[y2][dx] + I[dy][x1] - I[y2][x1] - I[dy][dx])
                            + (  I[y2][x2] + I[dy][dx] - I[y2][dx] - I[dy][x2]))
        feature_set.append(features)
    return np.array(feature_set)

```

Código 2: Función que extrae las características Haar

2.3. 5. Algoritmo Adaboost

Todas las funciones del item 5. se implementan dentro de una clase Adaboost, la cual inicializa un clasificador y tiene la siguiente estructura:

```
class Adaboost:
    """
    Clasificador Adaboost, se genera a partir de clasificadores debiles.
    """
    def ___init___(self):
        #iniciamos los arreglos que funcionaran como contenedores
        self.at = np.array([])
        self.it = np.array([])
        self.ut = np.array([])
        self.T = 10 #valor por defecto

    def h(x,u):
        return (x>u)*2-1

    def choose_u(X_feature,y_label,w_vec):
        #para cada caracteristica determina el mejor umbral de clasificación.
        ...
    def fit(self,X_feature,y,T):
        #entrena el clasificador, guardando los valores de at, it y ut en cada iteración.
        ...
```

```
21      def predict(self,X):
22          #Predice la clase para las muestras en X.
23          ...
```

Código 3: Extracto resumen Clase Adaboost

Los métodos h(), choose_u(), fit() y predict() corresponden a los items 5.a, 5.b, 5.c y 5.d respectivamente los cuales se explican a detalle a continuación:

5.a Se implementa el clasificador débil h(x,u), en donde 'x' es el vector características y 'u' es un umbral. Su salida debe ser: $+1$ cuando $x > u$, -1 cuando $x < u$. En el caso en que x sea un arreglo, su salida debe tener la misma dimensionalidad que 'x', y debe contener $+1$ o -1 en cada elemento. Esta función se utilizará como base para construir los clasificadores débiles por cada característica 'x_i' asociada a i.

```
1     #clasificador debil
2     def h(x,u):
3         return (x>u)*2-1
4
```

5.b Se implementa una función que permite elegir el mejor umbral 'u' para cada clasificador débil, es decir, para cada característica se determina el mejor 'u'. Para realizar esto se determinan 10 valores de umbrales equidistantes entre el menor y mayor valor de la característica respectiva. Cabe destacar que el rango de los umbrales es este pues la idea es que el umbral divida el conjunto, para considerarlo como clasificador. Luego, se determinan 10 valores de r, determinados con la formula de la ecuación 3, para cada valor de umbral, eligiendo el valor de u asociado al clasificador débil de esa característica que predice etiquetas con mayor valor de r.

$$r = \sum_{k=0}^{N-1} w_k y_k h(x_{k,i}, u) \tag{3}$$

Con N el número de ejemplos de entrenamiento, k el índice de estos ejemplos. Como r está construido en base a los pesos, el label correcto y el clasificador, en caso de que se clasifique bien, es decir, que el y_k sea igual a $h(x_{k,i}, u)$, al multiplicarse valores del mismo signo se sumará a r, en cambio en caso de una clasificación errónea, se restará valor a r, por eso se busca el umbral que entregue un mayor valor de r.

```
1     def choose_u(X_feature,y_label,w_vec):
2         """
3         X_feauters corresponde a la caracteristica i, un vector 296.
4         y_label vector de 296.
5         """
6         r = []
7         U = []
8         #cada columna representa una caracteristica, por cada columna determinamos
9         #un valor de r
10        for k in range(X_feature.shape[1]):
11            r_list = []
12            u_aux = []
13            X_col = X_feature[:,k]
```

```
            inicio = np.min(X_col)
            fin =  np.max(X_col)
            #10 umbrales equidistantes entre el menor y el mayor valor
            u_list = np.linspace(inicio, fin, 10)
            for u in u_list:
                ri = np.sum(w_vec*y_label*h(X_col,u))
                r_list.append(ri)
                u_aux.append(u)
            U.append( u_aux[np.argmax(r_list)] )
            r.append(r_list[np.argmax(r_list)])
        return np.array(U),np.array(r)

```

5.c Entrenamiento del clasificador: para el entrenamiento del clasificador, primero se inicializa el vector de pesos con un valor de $1/N$ en cada ejemplo, con N el número de muestras de entrenamiento. Se eligen T clasificadores débiles los cuales se utilizan en cada iteración para actualizar los pesos mientras guarda los valores de at, it y ut en sus listas respectivas. En las ecuaciones 4 y 5 se muestran las formulas de α y de la actualización de los pesos ω en cada iteración respectivamente. Los pesos se actualizan con los valores de umbrales y alpha que entregan el mayor valor de r.

$$\alpha = \frac{1}{2} \ln \frac{1 + r_t}{1 - r_t} \tag{4}$$

$$\omega = \omega * e^{-\alpha_t y_i h_{x_i}} \tag{5}$$

```
    def fit(self,X_feature,y,T=10):
        #N muestras
        #inicializamos los pesos con 1/N
        at,it,ut,hf = [], [], [], []
        N = X_feature.shape[0]
        w = np.ones(N)*(1/N)

        for t in range(T):
            #w = w/np.sum(w) #normalizamos los pesos
            u,r = choose_u(X_feature,y,w)
            alpha = 0.5*np.log((1+r)/(1-r))
            #indice del máximo valor de r
            idxmax = np.argmax(r)
            rmax = np.max(r)
            at.append(0.5*np.log((1+rmax)/(1-rmax)))
            it.append(idxmax)
            ut.append(u[idxmax])

            #actualizamos los pesos
            w = w*np.exp(-alpha[idxmax]*y*h(X_feature[:,idxmax],u[idxmax]))
        #se guardan los valores
        self.at = np.array(at)
        self.it = np.array(it)
```

```
24          self.ut = np.array(ut)
25
```

5.d Clasificador fuerte: Se implementa el clasificador fuerte que se muestra en la ecuación 6 a partir de los parámetros de entrenamiento. Este método predice la clase de muestras para cada ejemplo en X.

$$H(x) = signo(\sum_{t=0}^{T-1} \alpha_t h(x_{i_t}, u_t)) \tag{6}$$

```
1       def predict(self,X):
2           """
3           x corresponde al vector de caracteristicas de una imagen o de un conjunto de imagenes
4           a clasificar, at los alfas calculados, it los indices de los alfas calculados y ut los
5           umbrales respectivos.
6
7           Predice la clase para las muestras en X.
8           Parameters
9           ----------
10          X : The data matrix for which we want to get the predictions.
11          El vector o matriz de caracteristicas
12
13          Returns
14          -------
15          y_pred : ndarray of shape (n_samples,)
16              Vector que contiene las clases predichas para cada ejemplo.
17
18          """
19          try:
20              y_pred = []
21              for x in X:
22                  y_pred.append(np.sign(np.sum(self.at*h(x[self.it],self.ut))))
23          except IndexError:
24              y_pred = np.sign(np.sum(self.at*h(X[self.it],self.ut)))
25          return np.array(y_pred)
26
```

2.4. Clasificación

A continuación se describe la clasificación, se muestran los resultados y se realiza el análisis respectivo. Los accuracy's en los conjuntos de entrenamiento y prueba se pueden observar en la tabla 2.

6. Se realizó dentro de la función Haar(), se determina la imagen integral y se determinan las características de la lista de parámetros generada.

```
1   parametros = gen_parameters()
2
```

Tabla 2: Accuracy en el conjunto de entrenamiento y en el conjunto de Test del clasificador fuerte determinado.

	T = 5	T = 10	T = 20
Tiempo de ejecución	3.45 segundos	6.78 segundos	13.72 segundos
Accuracy Train	85.14 %	87.84 %	90.88 %
Accuracy Test	86.49 %	89.19 %	90.54 %

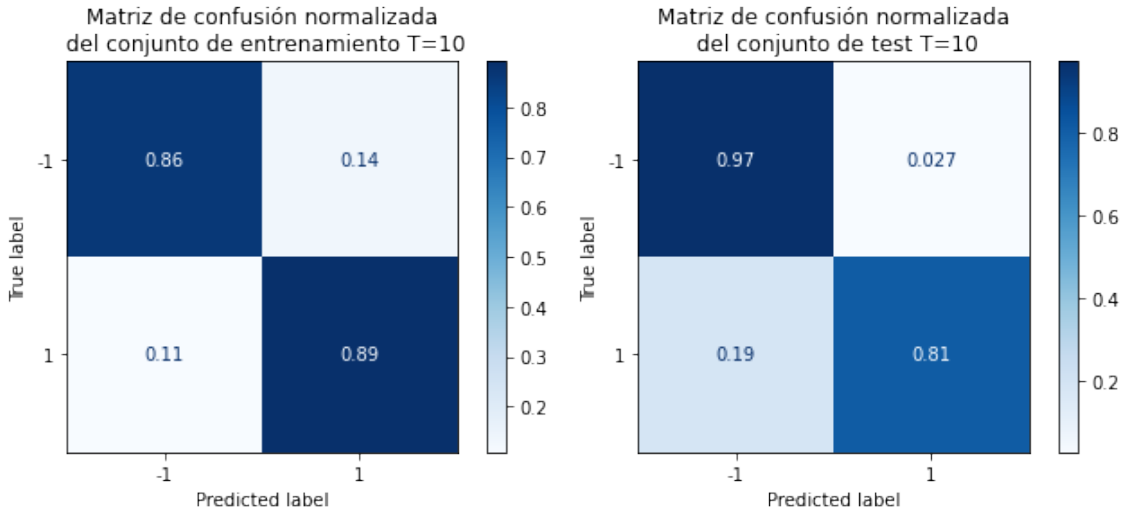
```
X_train_features = Haar(X_train,parametros)
X_test_features = Haar(X_test,parametros)
```

7. Se entrena el clasificador Adaboost considerando T=10, utilizando el conjunto de entrenamiento y el conjunto de prueba (Test), esto se realiza con el código a continuación.

```
clasificador = Adaboost() #inicializamos el clasificador
clasificador.fit(X_train_features,y_train,T=10)  #entrenamos
y_train_pred = clasificador.predict(X_train_features) #

```

Luego se utiliza la función ConfusionMatrixDisplay() para generar las matrices de confusión respectivas, las cuales se observan en la figura 3 y además se guarda el valor de su accuracy en la tabla 2. Sus resultados de Accuracy en test son buenos, llegando al 89 % de clasificaciones correctas, por otro lado el Accuracy en train nos indica que no hay overfitting, de todas maneras es algo que podría mejorar eventualmente con más iteraciones de entrenamiento. Además, en Test clasifica los verdaderos positivos con un alto porcentaje de clasificaciones correctas, es decir, las imágenes que no presentaban personas el clasificador las clasifico con su label (etiqueta) correcto con un 97 %. Por otro lado, los Verdaderos negativos tienen un menor porcentaje de clasificaciones correctas, es decir, al clasificador le es más fácil clasificar de forma correcta imágenes de no personas que imágenes de personas.



(a) (b)

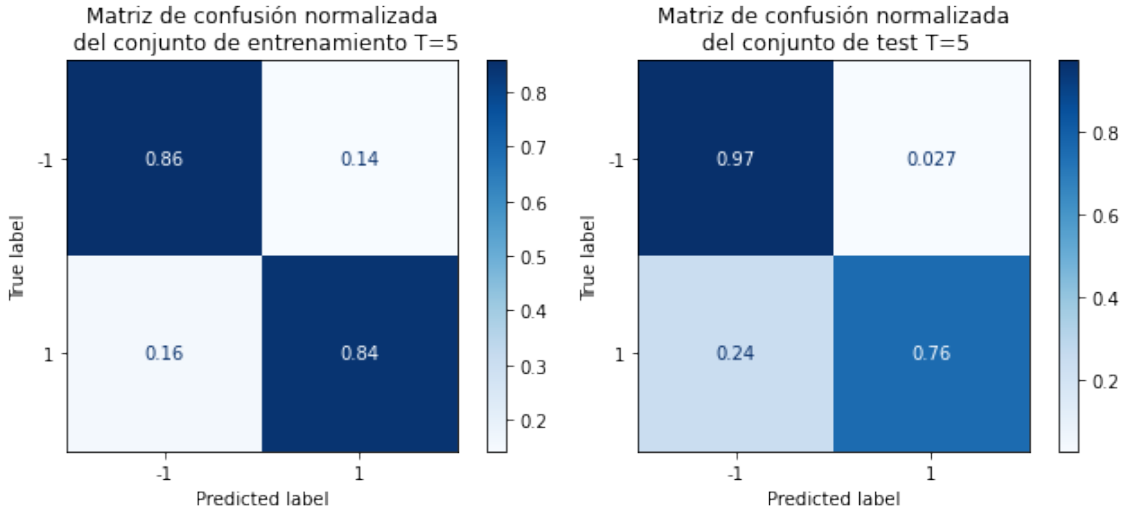
Figura 3: Matrices de confusión normalizadas. Utilizando T=10.

8. y 9. Para T=5, los resultados de clasificación son buenos, llegando tanto en Train como en Test a tener accuracy's sobre el 80 %. A pesar de esto la clasificación de verdaderos negativos, que en este caso seria clasificar personas como personas, está por debajo del 80 % de las veces, no siendo tan efectivo para esto que es el objetivo principal de la tarea. La matriz de confusión se puede observar en la figura 4.

   Por otro lado, para T=20 sus resultados de Accuracy en test son buenos, llegando al 90 % de clasificaciones correctas, por otro lado el Accuracy en train nos indica que no hay overfitting, de todas maneras es algo que podría mejorar eventualmente con más iteraciones de entrenamiento. Además, en Test clasifica los verdaderos positivos con un alto porcentaje de clasificaciones correctas, es decir, las imágenes que no presentaban personas el clasificador las clasifico con su label (etiqueta) correcto con un 97 %. De todas maneras el objetivo de la tarea es clasificar personas, por lo que es más relevante los verdaderos negativos que en este caso representan las clasificaciones correctas de las personas como personas. La matriz de confusión se puede observar en la figura 5.

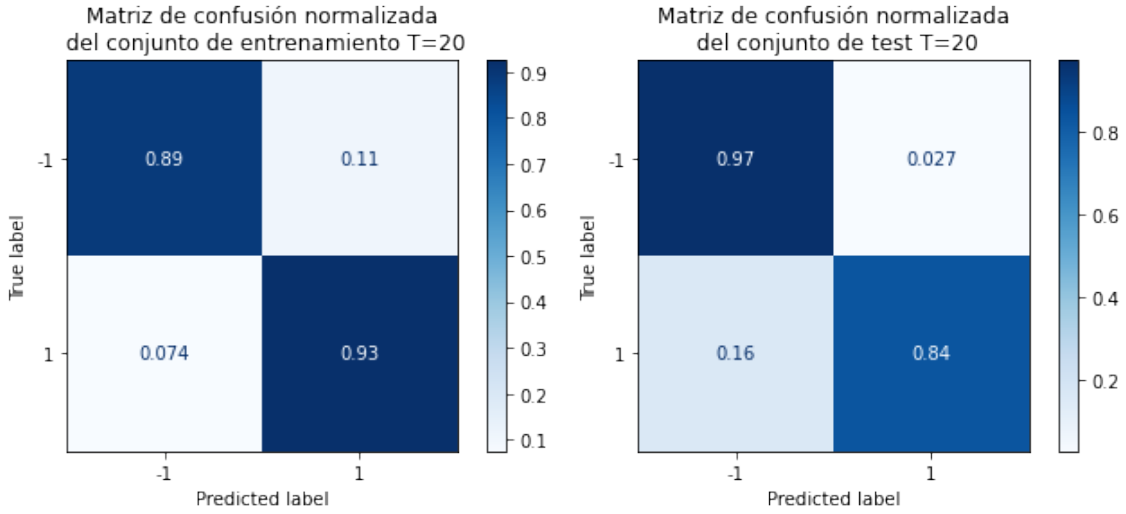
   Al comparar los 3 clasificadores, se puede observar que el que tiene mejores resultados corresponde al caso de T=20, de todas maneras no tiene una mejora significativa con respecto al clasificador de T=10, siendo este ultimo solo 1 % menor en accuracy en Test. Siendo el clasificador de T=10 más rápido, pues se demora la mitad aproximadamente que el clasificador T=20, creo que este clasificador sigue siendo una de las mejores opciones.
   De todas maneras la elección del mejor clasificador depende de la aplicación que se le quiera dar, en caso de querer darle énfasis a la velocidad de los clasificadores el clasificador de T=5 es la mejor opción pues se demora la mitad del clasificador T=10 y casi 5 veces menos que el clasificador de T=20, por lo que no hay un clasificador malo como tal, dependerá de lo que se busque realizar. Considerando el problema de la tarea, el cual es la detección de personas, el mejor clasificador para detectar personas corresponde al clasificador de T=20, pues la matriz

de confusión muestra que este es el que menos se equivoca al clasificar personas (mayor tasa de verdaderos negativos que en este caso representa a la clase persona).



(a) (b)

Figura 4: Matrices de confusión normalizadas. Utilizando T=5.



(a) (b)

Figura 5: Matrices de confusión normalizadas. Utilizando T=20.

10. Se seleccionan las primeras 5 mascaras que tienen mayores valores de r para visualizarlas en imágenes, en la figura 6 se puede observar el resultado de estas, presentándose solo 3 tipos de mascaras, 3 mascaras tipo 1, 3 mascaras tipo 0 y 1 mascara tipo 4. Si bien se pedían 5 mascaras se muestran 6 pues se evidencia mejor el desempeño. Como se puede observar en esta figura, las

mascaras en general encuadran caras y partes del cuerpo, en especial la mascara de tipo 4 en color azul encuadra el cuerpo de las personas, la mascara de tipo 1 de rosado oscuro encuadra caras o partes de ella y las mascaras de tipo 0 en rojo encuadra partes del cuerpo y zonas que no son relevantes de forma visual (detecta zonas donde no hay personas), esto puede ser una de las razones por la cual el desempeño del clasificador no es perfecto. Otro aspecto relevante fue que todas estas mascaras son de polaridad positiva, esto se puede deber al tipo de fotos de entrenamiento, por ejemplo en estás 3 imágenes la persona está orientada de forma similar. Tanto la imagen como los vértices de los cuadros fueron escalados, llevando la imagen a un tamaño de 192x192, los vértices fueron multiplicados por 192/24 siguiendo la regla de 3 simple, por otro lado el color de las imágenes dependen del tipo y de la polaridad para poder distinguir las diferentes mascaras, esto se muestra en el código encontrado al final de este item, los colores están en BGR.

(a) (b) (c)

Figura 6: Ejemplos de las mejores mascaras.En rojo se muestran las mascaras tipo 0, en rosado oscuro las mascaras de tipo 1 y en azul las mascaras de tipo 4 presentes en la figura, todas de polaridad positiva.

```
image = cv2.resize(img, (192,192))
s=0
for y1,x1,y2,x2,tipo,polaridad in  p[clasificador.it]:
    if s<=6:
        y1r = y1*192//24
        x1r = x1*192//24
        y2r = y2*192//24
        x2r = x2*192//24
        print(tipo,polaridad,(tipo*10,tipo*10,155+100*polaridad))
        #colores BGR
        image = cv2.rectangle(image, (x1r,y1r),(x2r,y2r),(int(60*(tipo)),int(10*(tipo)),int(150*
    ↪ polaridad+50*(1-tipo))),1)
    s+=1
cv2_imshow(image)
```

3. Conclusión

Luego del desarrollo de esta tarea, se logró posible diseñar y construir un sistema de detección de personas, que utilice características tipo Haar y un clasificador Adaboost desde cero, cumpliéndose el objetivo de esta, además los clasificadores desarrollados presentaron buenos resultados en el conjunto de prueba (cercanos al 90 %), evidenciando así la efectividad de las características tipo Haar y del algoritmo del clasificador Adaboost, que si bien es simple (en comparación a una red neuronal) funciona bien un gran porcentaje de las veces.Además fue posible comprender el algoritmo Adaboost y las características tipo Haar, sus ventajas y se pudo evidenciar su gran utilidad en el reconocimiento de personas. Estas herramientas son bastante útiles para nuestra formación como ingenieros pues se puede aplicar a un sin fin de proyectos. Se recomienda el uso de el clasificador Adaboost utilizando T=20 en caso de querer realizar un proyecto similar, pues fue el que presentó mejores resultados identificando personas. Cabe destacar que los resultados variaban dependiendo de la semilla aleatoria que se le asignaba a la función que separaba los conjuntos, lo cual es lago a considerar en caso de realizar un proyecto similar.Se pusieron en practica los conceptos y técnicas vistas en clases, programarlas en Python manteniendo el uso de librerías al mínimo, uso de Cython y el uso de funciones de Numpy lo cual permitió mejorar los tiempos de ejecución del programa. A pesar de esto algunas de las funciones realizadas no fueron optimizadas, por lo que el tiempo de ejecución del código, que si bien es bajo, puede mejorar. La mayor dificultad de esta tarea fue el de entender y lograr traspasar los algoritmos de Adaboost teóricos a código para desarrollar las funciones.

Referencias

[1] P. Viola and M. Jones, Rapid object detection using a boosted cascade of simple features,"Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517.

4. Anexos

```
# -*- coding: utf-8 -*-
"""Tarea4_imagenes.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/13eURgrzxIg5wa-p12DjdpVgVeN63R8R8

# Desarrollo por Joaquin Zepeda V.

Tarea 3 EL7008 - Detección de personas usando Adaboost y características de tipo Haar.

# Subimos las imagenes a google colab y luego las extraemos

Se debe subir la archivo imagenes_tarea4_2022.zip y luego se ejecuta todo.
"""

!unzip /content/imagenes_tarea4_2022.zip

from sklearn.model_selection import train_test_split
import numpy as np
import cython
import cv2

np.random.seed(42)

"""
1. leer las imágenes de la base de datos: 80 % train, 20 % test.  Para dividir los datos en
entrenamiento y prueba, se debe utilizar una semilla fija al usar train_test_split( )
Redimensionar las imagenes a 24x24.
Las etiquetas deben ser +1 para las imagenes con perosnas y -1 para las imagenes sin personas.
"""
import glob

def cargarDatos(nombre_carpeta,extension,clase):
    """
    Ejemplo de uso cargarDatos(car_side,"jpg",-1)
    """
    data = []
    path = glob.glob(f"{nombre_carpeta}//*.{extension}")
    if nombre_carpeta == "pedestrian":
        path.sort(key=lambda x: int((x.split(".")[0].split('/')[1]))) # ordena el conjunto de datos para
     ↪ tener resultados reproducibles
    else:
        path.sort(key=lambda x: int((x.split(".")[0].split('/')[1].split("_")[1]))) # ordena el conjunto de
     ↪  datos
    label = []
    for img in path:
```

```
47          #leemos la imagen en escala de grises
48          gray = cv2.imread(img,0)
49          gray32 = np.float32(gray)
50          out = cv2.resize(gray32, (24,24))
51          data.append(out)
52          label.append(clase)
53
54      return np.array(data),np.array(label)
55
56
57  #Carga todos los datos de la carpeta "car_side" y le asigna la clase -1
58  dataCar_side,CSlabel = cargarDatos("car_side","jpg",-1)
59  text_labels = ["Cars/Chairs","pedestrian"]
60
61  #Carga todos los datos de la carpeta "chair" y le asigna la clase -1
62  dataChair,CL = cargarDatos("chair","jpg",-1)
63
64  #Carga todos los datos de la carpeta "pedestrian" y le asigna la clase 1
65  dataPedestrian,PL = cargarDatos("pedestrian","png",1)
66
67  #data no normalizada
68  data = np.concatenate((dataCar_side,dataChair,dataPedestrian))
69  labels = np.concatenate((CSlabel,CL,PL))
70
71  # Conjuntos de train y test
72  #random_state = 42 semilla fija para resultados reproducibles.
73  X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, shuffle = True,
        ↪ stratify = labels,random_state=42)
74
75  print(X_train.shape)
76  print(X_test.shape)
77
78  # Commented out IPython magic to ensure Python compatibility.
79  #  %load_ext Cython
80
81  # Commented out IPython magic to ensure Python compatibility.
82  #  %%cython
83  #
84  # import numpy as np
85  # cimport numpy as np
86  #
87  # cpdef np.ndarray[np.float32_t,ndim=2] integral(np.ndarray[np.float32_t,ndim=2] img):
88  #     """
89  #     2. función en Cython que permita calcular la imagen integral, dada una imagen.
90  #     """
91  #     cdef int rows,cols,i,j
92  #     cdef np.ndarray[np.float32_t, ndim=2] output=np.zeros([img.shape[0], img.shape[1]], dtype =
        ↪ np.float32)
93  #     # tamano de la imagen
94  #     rows = img.shape[0]
95  #     cols = img.shape[1]
```

```
#     #Como se van guardando los valores en la imagen integral, por construcción
#     #se puede realizar con 4 operaciones (usando valores precomputados), de esta
#     #manera se evita realizar sumas repetitivas.
#
#     output[0][0] = img[0][0]
#     for i in range(rows):
#       for j in range(cols):
#           #Casos base
#           if i==0:
#               output[i][j] = output[i][j-1] + img[i][j]
#           elif j==0:
#               output[i][j] = output[i-1][j] + img[i][j]
#           else:
#               output[i][j] = img[i][j]+output[i-1][j]+output[i][j-1]-output[i-1][j-1]
#
#     return output

"""
3. Implementar una función que genere parámetros que permitan determinar las máscaras de
las características Haar. Las máscaras se deben parametrizar como: (y1, x1, y2, x2, tipo,
polaridad), donde:
a. (y1,x1) es la esquina superior izquierda de la máscara
b. (y2,x2) es la esquina inferior derecha de la máscara
c. Tipo: número entre 0 y 4, que permite determinar el tipo de máscara
d. Polaridad: +1 (positiva) o −1 (negativa).
Dado que en cada imagen las características de pueden calcular en muchas posibles
posiciones, se recomienda aplicarlas usando el concepto de ventana deslizante usando un
paso de tamaño 3. Además, se recomienda usar sólo 3 valores para los anchos de las
máscaras y 3 valores para los altos. En consecuencia, hay 9 posibles combinaciones de
anchos/altos. En el caso de las máscaras de tipo 0, 1 y 4, los anchos deben ser múltiplos de 4,
mientras que para las características 2 y 3, el ancho debe ser múltiplo de 3. Además, se debe
almacenar la polaridad de cada máscara, la cual puede ser +1 o -1.
"""
import numpy as np
def gen_parameters():
    L = []
    tipos = [   0,          1,      2,          3,      4]
    tipo2y3 = [3,6,9]
    anchos = [[4,8,12], [4,8,12], tipo2y3, tipo2y3,[4,8,12]]
    altos = [[4,8,12], [4,8,12], tipo2y3, tipo2y3,[4,8,12]]
    #generamos los parametros por cada tipo, sus 9 combinaciones
    #y además recorremos con una ventana deslizante de paso 3 con el fin
    #de determinar todas las mascaras que queden dentro de la imagen.
    for tipo in tipos:
            for y1 in range(0,24,3):
                for x1 in range(0,24,3):
                    for A in anchos[tipo]:
                        for B in altos[tipo]:
                            if y1+B<24 and x1+A<24:
                                #agregamos las mascaras en sus 2 polaridades
                                L.append([y1,x1,y1+B,x1+A,tipo,0])
```

```
                        L.append([y1,x1,y1+B,x1+A,tipo,1])

    return L


def Haar(img_set,mascaras):
    """
    4. Implementar una función que, dado un conjunto de imágenes y parámetros de máscaras,
    calcule vectores de características. Dichos vectores deben contener todas las características
    tipo Haar determinadas en el punto anterior, para cada imagen.

    Por cada imagen, determinamos la imagen integral y luego dependiendo del tipo de la mascara
    se calcula el valor de la mascara aplicada en la imagen según las posiciones que indica la mascara.
    """
    feature_set = []
    for img in img_set:
        features = np.zeros(len(mascaras))
        #determinamos la imagen integral
        I = integral(img)
        for index,mask in enumerate(mascaras):
            #rescatamos los parametros de la mascara
            y1,x1,y2,x2,tipo,p = mask
            A = x2-x1
            B = y2-y1

            if tipo==0:
                d = x1+A//2
                features[index]=p*(( I[y2][d]+I[y1][x1]-I[y2][x1]-I[y1][d] )
                            -( I[y2][x2] + I[y1][d]-I[y2][d]-I[y1][x2]))
            elif tipo==1:
                d = y1+B//2
                features[index]=p*(( I[y2][x2]+I[d][x1]-I[y1][x2]-I[d][x2] )
                             -( I[d][x2] + I[y1][x1]-I[d][x1]-I[y1][x2]))
            elif tipo==2:
                d1 = x1+A//3
                d2 = x1+2*A//3
                features[index]=p*(  (I[y2][d1]+I[y1][x1]-I[y1][d1]-I[y2][x1])
                              - ( I[y2][d2]+I[y1][d1]-I[y1][d2]-I[y2][d1])
                              + ( I[y2][x2] + I[y1][d2] - I[y2][d2] - I[y1][x2]))
            elif tipo==3:
                d1 = y1+B//3
                d2 = y1+2*B//3
                features[index]=p*( (I[d1][x2] + I[y1][x1] - I[d1][x1] - I[y1][x2])
                              -(I[d2][x2] + I[d1][x1] - I[d2][x1] - I[d1][x2])
                              +(I[y2][x2] + I[d2][x1] - I[y2][x1] - I[d2][x2] ))
            else:
                dy = y1+B//2
                dx = x1+A//2
                features[index]=p*( (I[dy][dx] + I[y1][x1] - I[dy][x1] - I[y1][dx])
                            - (  I[dy][x2] + I[y1][dx] - I[dy][dx] - I[y1][x2])
                            - (  I[y2][dx] + I[dy][x1] - I[y2][x1] - I[dy][dx])
```

```
                    + (  I[y2][x2] + I[dy][dx] - I[y2][dx] - I[dy][x2]))

        feature_set.append(features)
    return np.array(feature_set)

parametros = gen_parameters()

X_train_features = Haar(X_train,parametros)
X_test_features = Haar(X_test,parametros)

X_train_features.shape

len(parametros)

"""
5.a. Implementar una función (, ), Su salida debe ser: +1 cuando  >  , -1 cuando
 < . En el caso en que x sea un arreglo, su salida debe tener la misma
dimensionalidad que x, y debe contener +1 o -1 en cada elemento. Esta función se
usará como base para construir clasificadores débiles (, ) asociados a la característica número i.
"""
def h(x,u):
    return (x>u)*2-1

"""b. Implementar una función que permita elegir el mejor u para cada clasificador débil,
dada una matriz de características X, un vector de etiquetas y, y un vector de pesos
w. Para realizar este paso, se debe dividir el rango de cada característica i en 10
valores, y se debe encontrar el valor de u en el cual el clasificador débil asociado a
esa característica predice las etiquetas con el **mayor valor r**.

"""

def choose_u(X_feature,y_label,w_vec):
    """
    X_feauters corresponde a la caracteristica i, un vector 296.
    y_label vector de 296.
    """
    r = []
    U = []
    #cada columna representa una caracteristica, por cada columna determinamos
    #un valor de r
    for k in range(X_feature.shape[1]):
        r_list = []
        u_aux = []
        X_col = X_feature[:,k]
        inicio = np.min(X_col)
        fin =  np.max(X_col)
        #10 umbrales equidistantes entre el menor y el mayor valor
        u_list = np.linspace(inicio, fin, 10)
        for u in u_list:
            ri = np.sum(w_vec*y_label*h(X_col,u))
            r_list.append(ri)
```

```
249             u_aux.append(u)
250         U.append( u_aux[np.argmax(r_list)] )
251         r.append(r_list[np.argmax(r_list)])
252     return np.array(U),np.array(r)
253 
254 def clasificador_fit(X_feature,y,T):
255     #N muestras
256     #inicializamos los pesos con 1/N
257     at,it,ut,hf = [], [], [], []
258     N = X_feature.shape[0]
259     w = np.ones(N)*(1/N)
260 
261     for t in range(T):
262         w = w/np.sum(w) #normalizamos los pesos
263         u,r = choose_u(X_feature,y,w)
264         alpha = 0.5*np.log((1+r)/(1-r))
265         #indice del máximo valor de r
266         idxmax = np.argmax(r)
267         rmax = np.max(r)
268         at.append(0.5*np.log((1+rmax)/(1-rmax)))
269         it.append(idxmax)
270         ut.append(u[idxmax])
271 
272         #actualizamos los pesos
273         w = w*np.exp(-alpha[idxmax]*y*h(X_feature[:,idxmax],u[idxmax]))
274 
275     return np.array(at),np.array(it),np.array(ut)
276 
277 #Reuniendo todo en una clase Adaboost que representa al clasificador.
278 import numpy as np
279 
280 class Adaboost:
281     """
282     Clasificador Adaboost, se genera a partir de clasificadores debiles.
283     """
284     def ___init___(self):
285         #iniciamos los arreglos que funcionaran como contenedores
286         self.at = np.array([])
287         self.it = np.array([])
288         self.ut = np.array([])
289 
290     def h(x,u):
291         return (x>u)*2-1
292 
293     def choose_u(X_feature,y_label,w_vec):
294         """
295         X_feauters corresponde a la caracteristica i, un vector 296.
296         y_label vector de 296.
297         """
298         r = []
299         U = []
```

```
300         #cada columna representa una caracteristica, por cada columna determinamos
301         #un valor de r
302         for k in range(X_feature.shape[1]):
303             r_list = []
304             u_aux = []
305             X_col = X_feature[:,k]
306             inicio = np.min(X_col)
307             fin =  np.max(X_col)
308             #10 umbrales equidistantes entre el menor y el mayor valor
309             u_list = np.linspace(inicio, fin, 10)
310             for u in u_list:
311                 ri = np.sum(w_vec*y_label*h(X_col,u))
312                 r_list.append(ri)
313                 u_aux.append(u)
314             U.append( u_aux[np.argmax(r_list)] )
315             r.append(r_list[np.argmax(r_list)])
316         return np.array(U),np.array(r)
317
318     def fit(self,X_feature,y,T=10):
319         #N muestras
320         #inicializamos los pesos con 1/N
321         at,it,ut,hf = [], [], [], []
322         N = X_feature.shape[0]
323         w = np.ones(N)*(1/N)
324
325         for t in range(T):
326             #w = w/np.sum(w) #normalizamos los pesos
327             u,r = choose_u(X_feature,y,w)
328             alpha = 0.5*np.log((1+r)/(1-r))
329             #indice del máximo valor de r
330             idxmax = np.argmax(r)
331             rmax = np.max(r)
332             at.append(0.5*np.log((1+rmax)/(1-rmax)))
333             it.append(idxmax)
334             ut.append(u[idxmax])
335
336             #actualizamos los pesos
337             w = w*np.exp(-alpha[idxmax]*y*h(X_feature[:,idxmax],u[idxmax]))
338
339         self.at = np.array(at)
340         self.it = np.array(it)
341         self.ut = np.array(ut)
342
343     def predict(self,X):
344         """
345         x corresponde al vector de caracteristicas de una imagen o de un conjunto de imagenes
346         a clasificar, at los alfas calculados, it los indices de los alfas calculados y ut los
347         umbrales respectivos.
348
349         Predice la clase para las muestras en X.
350         Parameters
```

```
        ----------
        X : The data matrix for which we want to get the predictions.
        El vector o matriz de caracteristicas

        Returns
        -------
        y_pred : ndarray of shape (n_samples,)
            Vector que contiene las clases predichas para cada ejemplo.

        """
        try:
            y_pred = []
            for x in X:
                y_pred.append(np.sign(np.sum(self.at*h(x[self.it],self.ut))))
        except IndexError:
            y_pred = np.sign(np.sum(self.at*h(X[self.it],self.ut)))
        return np.array(y_pred)

clasificador = Adaboost() #inicializamos el clasificador
clasificador.fit(X_train_features,y_train,T=10)  #entrenamos
y_train_pred = clasificador.predict(X_train_features) #
y_test_pred = clasificador.predict(X_test_features)

# calculate accuracy
import matplotlib.pyplot as plt
from sklearn.metrics import  accuracy_score, confusion_matrix, recall_score
from sklearn.metrics import ConfusionMatrixDisplay

print("Train:")
accuracy = accuracy_score(y_train, y_train_pred)*100
recall = recall_score(y_train, y_train_pred, average='macro')*100
print("Classification accuracy is %2f" %accuracy," %")
print("Classification recall is %2f" %recall," %\n")

cm = confusion_matrix(y_train, y_train_pred,labels=[-1,1],normalize='true')
disp =ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[-1,1])
disp.plot(cmap='Blues')
plt.title("Matriz de confusión normalizada\n del conjunto de entrenamiento T=10")
plt.show()

print("Test:")
accuracy = accuracy_score(y_test, y_test_pred)*100
recall = recall_score(y_test, y_test_pred, average='macro')*100
print("Classification accuracy is %2f" %accuracy," %")
print("Classification recall is %2f" %recall," %\n")

cm = confusion_matrix(y_test, y_test_pred,labels=[-1,1],normalize='true')
disp =ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[-1,1])
disp.plot(cmap='Blues')
plt.title("Matriz de confusión normalizada\n del conjunto de test T=10")
plt.show()
```

```
402
403 import time
404 clasificador = Adaboost()
405
406 T = [5,10,20]
407 for t in T:
408     inicio = time.time()
409     clasificador.fit(X_train_features,y_train,T=t)
410     y_train_pred = clasificador.predict(X_train_features)
411     y_test_pred = clasificador.predict(X_test_features)
412     final = time.time()
413     print(f"Tiempo de ejecución para T={t}: {round(final-inicio,2)} segundos.")
414
415     print("Train:")
416     accuracy = accuracy_score(y_train, y_train_pred)*100
417     recall = recall_score(y_train, y_train_pred, average='macro')*100
418     print("Classification accuracy is %2f" %accuracy," %")
419     print("Classification recall is %2f" %recall," %\n")
420
421     cm = confusion_matrix(y_train, y_train_pred,labels=[-1,1],normalize='true')
422     disp =ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[-1,1])
423     disp.plot(cmap='Blues')
424     plt.title(f"Matriz de confusión normalizada\n del conjunto de entrenamiento T={t}")
425     plt.show()
426
427     print("Test:")
428     accuracy = accuracy_score(y_test, y_test_pred)*100
429     recall = recall_score(y_test, y_test_pred, average='macro')*100
430     print("Classification accuracy is %2f" %accuracy," %")
431     print("Classification recall is %2f" %recall," %\n")
432
433     cm = confusion_matrix(y_test, y_test_pred,labels=[-1,1],normalize='true')
434     disp =ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[-1,1])
435     disp.plot(cmap='Blues')
436     plt.title(f"Matriz de confusión normalizada\n del conjunto de test T={t}")
437     plt.show()
438
439 clasificador = Adaboost() #inicializamos el clasificador
440 clasificador.fit(X_train_features,y_train,T=10)  #entrenamos
441 y_train_pred = clasificador.predict(X_train_features) #
442 y_test_pred = clasificador.predict(X_test_features)
443
444 p= np.array(parametros)
445 p[clasificador.it]
446
447 import cv2
448 from google.colab.patches import cv2_imshow
449
450 img = cv2.imread('/content/pedestrian/1.png')
451 image = cv2.resize(img, (192,192))
452 s=0
```

```
for y1,x1,y2,x2,tipo,polaridad in  p[clasificador.it]:
    if s<=6:
        y1r = y1*192//24
        x1r = x1*192//24
        y2r = y2*192//24
        x2r = x2*192//24
        print(tipo,polaridad,(tipo*10,tipo*10,155+100*polaridad))
        #colores BGR
        image = cv2.rectangle(image, (x1r,y1r),(x2r,y2r),(int(60*(tipo)),int(10*(tipo)),int(150*
    ↪ polaridad+50*(1-tipo))),1)
    s+=1
cv2_imshow(image)

img = cv2.imread('/content/pedestrian/9.png')
image = cv2.resize(img, (192,192))
s=0
for y1,x1,y2,x2,tipo,polaridad in  p[clasificador.it]:
    if s<=6:
        y1r = y1*192//24
        x1r = x1*192//24
        y2r = y2*192//24
        x2r = x2*192//24
        print(tipo,polaridad,(tipo*10,tipo*10,155+100*polaridad))
        image = cv2.rectangle(image, (x1r,y1r),(x2r,y2r),(int(60*(tipo)),int(10*(tipo)),int(150*
    ↪ polaridad+50*(1-tipo))),1)
    s+=1
cv2_imshow(image)

img = cv2.imread('/content/pedestrian/6.png')
image = cv2.resize(img, (192,192))
s=0
print(len(p[clasificador.it]))
for y1,x1,y2,x2,tipo,polaridad in  p[clasificador.it]:
    if s<=6:
        y1r = y1*192//24
        x1r = x1*192//24
        y2r = y2*192//24
        x2r = x2*192//24
        print(tipo,polaridad,(60*(tipo),10*(tipo),150*polaridad+50*(1-tipo)))
        image = cv2.rectangle(image, (x1r,y1r),(x2r,y2r),(int(60*(tipo)),int(10*(tipo)),int(150*
    ↪ polaridad+50*(1-tipo))),1)
    s+=1
cv2_imshow(image)
```