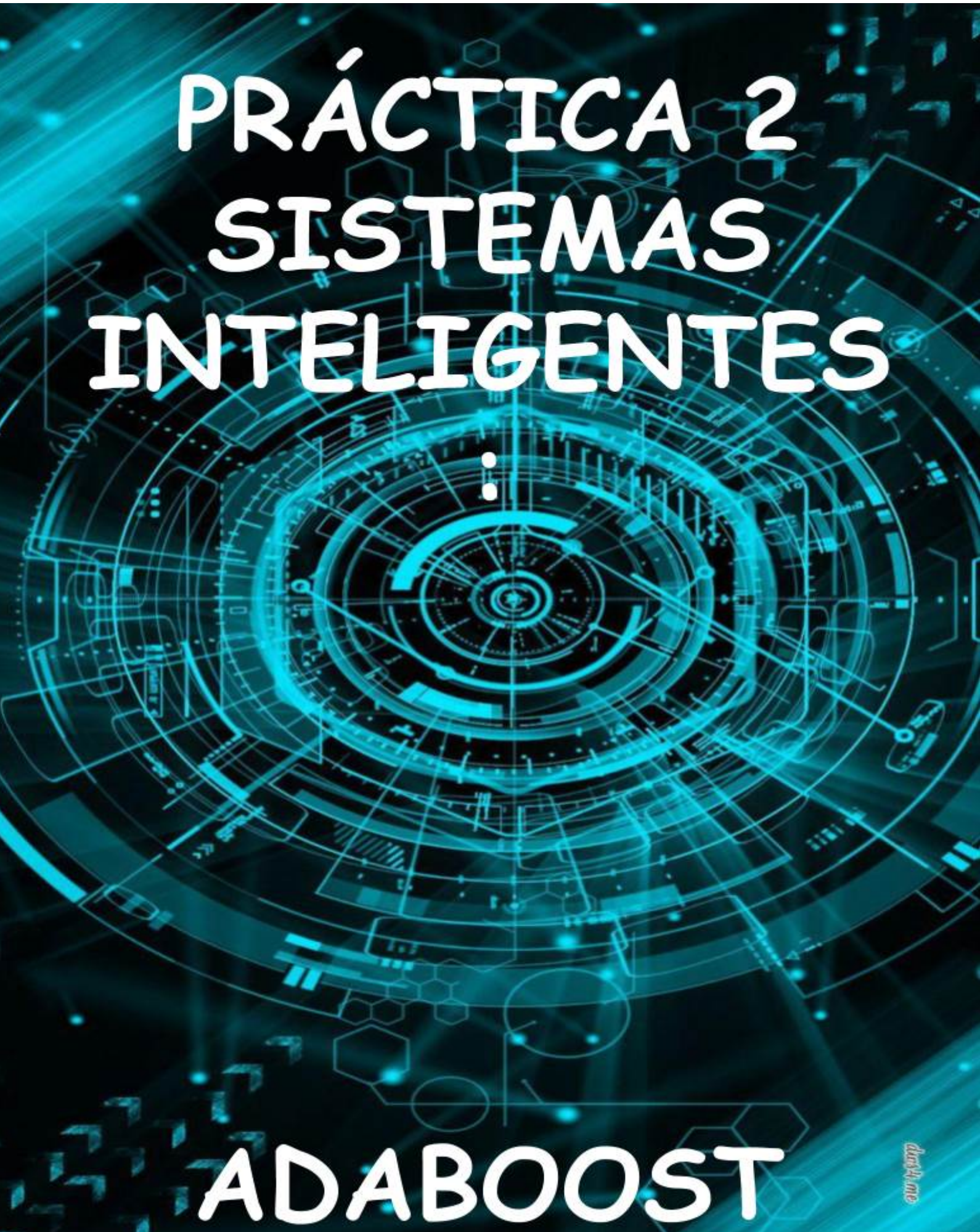
Nombre: Jose Miguel Gómez Lozano
DNI: 48833504L
Fecha: 22/12/2018



Índice

Introducción

En esta práctica, vamos a implementar un clasificador de imágenes, teniendo 8 tipos **(abrigos, bolsos, camisetas, pantalones, suéteres, vestidos, zapatillas, zapatos)**, el clasificador implementado debe ser capaz de decirnos si es o no es del tipo buscado.

Para esta práctica, partimos de una plantilla ofrecida por el profesorado con una *"base de datos"*, la cual contiene 8 carpetas con números desde 0 hasta 7, y cada carpeta contiene un tipo de imagen. Teniendo en cuenta esto, vamos a diseñar una clase *Enum* que llamaremos *ImagenType*, con todos los valores posibles (*Este paso **NO ES NECESARIO**, pero se usará para facilitar la implementación a la hora de identificar si es del mismo tipo o no).*

Tras *desarmar* el pseudocódigo ofrecido por la documentación de los profesores, hemos observado que podemos distinguir varios tipos significativos que serán clave en la implementación de nuestro algoritmo AdaBoost, así que vamos a ir destacando desde lo más específico, hasta lo más general del algoritmo conforme avance la explicación de la práctica.

Clases que componen al algoritmo AdaBoost:

{

Como punto de partida, explicaremos el tipo de dato que debe clasificar nuestro algoritmo, la clase **Imagen**.

Posteriormente, explicaremos la clase **Hyperplane** (Hiperplano), la cual se creará de forma completamente aleatoria, pudiendo ser más o menos eficiente y con una relación muy fuerte con los datos de **Imagen**.

A continuación, explicaremos la clase **ClasificadorDebil**, la cual es dependiente de **Hyperplane**.

Y, finalmente, explicaremos la clase **ClasificadorFuerte**, que depende de **ClasificadorDebil**, y se podría decir que, en su conjunto, es el algoritmo AdaBoost, pero ya explicaremos el *por qué* de esto.

}

Desglose del pseudocódigo

Por un lado, vamos a identificar lo más genérico del algoritmo AdaBoost:

Algorithm 1 Adaboost

procedure ADABOOST(X, Y)
  $D_1(i) = 1/N$ ▷ Indica como de difcil es de clasificar cada punto i
  for $t = 1 \rightarrow T$ **do** ▷ T es el nmero de clasificadores dbiles a usar
    Entrenar h_t teniendo en cuenta D_t
    Start
      for $k = 1 \rightarrow A$ **do** ▷ A = num. de pruebas aleatorias
        F_p = generaPlanoAlAzar()
        $\epsilon_t = P_{D_t}(h_t(x_i) \neq y_i) \rightarrow \epsilon_{t,k} = \sum_{i=1}^{N} D_t(i) \cdot (F_k(x) \neq y(x))$
      return $< F_p | \min(\epsilon_{t,k}) >$
    End
    Del h_t anterior obtener su valor de confianza $\alpha_t \in \mathbb{R}$
    Start
      $\alpha_t = 1/2 \log_2 \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
    End
    Actualizar D_{t+1}
    Start
      $D_{t+1} = \frac{D_t(i) \cdot e^{-\alpha_t \cdot y_i h_t(x_i)}}{Z_t}$
      $Z_t = \sum_i D_t(i)$
    End
  return $H(x) = sign(\sum_t \alpha_t \cdot h_t(x))$

Como podemos observar en la imágen anterior, hemos diferenciados dos franjas, por un lado la verde, que sería el proceso de construcción de un ***ClasificadorFuerte***, y por otro una franja azul, que corresponde a **aplicar dicho clasificador ya creado**.

Lo que identificamos en el algoritmo como **dificultad** para clasificar, lo hemos insertado como **peso** en la clase ***Imagen***, inicializando este a 1 como peso unitario de primera instancia, siempre y cuando no se use en un clasificador fuerte (en ese caso, se modificaría su valor).

El constructor de ***ClasificadorFuerte,*** primero inserta los **pesos** (dificultad para clasificar) a las imágenes, posteriormente, se generarán T clases de ***ClasificadorDebil*** (el cual, se encarga de los siguientes dos fragmentos de **START-END**). Y finalmente actualiza el valor de los pesos de las imágenes y **Zt**, que es un valor de normalización de valores para el peso de las imagenes.

NOTA: El peso de las imágenes está indicada en el pseudocódigo como Dt.

Algorithm 1 Adaboost

```
1:  procedure ADABOOST(X, Y)
2:      D_1(i) = 1/N            ▷ Indica como de difcil es de clasificar cada punto i
3:      for t = 1 → T do        ▷ T es el nmero de clasificadores dbiles a usar
4:          Entrenar h_t teniendo en cuenta D_t
5:          Start
6:              for k = 1 → A do        ▷ A = num. de pruebas aleatorias
7:                  F_p = generaPlanoAlAzar()
8:                  ε_t = P_{D_t}(h_t(x_i) ≠ y_i) → ε_{t,k} = Σ_{i=1}^{N} D_t(i) · (F_k(x) ≠ y(x))
                    return < F_p | min(ε_{t,k}) >
9:          End
10:         Del h_t anterior obtener su valor de confianza α_t ∈ ℝ
11:         Start
12:             α_t = 1/2 log_2 ((1 − ε_t)/ε_t)
13:         End
14:         Actualizar D_{t+1}
15:         Start
16:             D_{t+1} = D_t(i) · e^{−α_t · y_i h_t(x_i)} / Z_t
17:             Z_t = Σ_i D_t(i)
18:         End
        return H(x) = sign(Σ_t α_t · h_t(x))
```

Como podemos observar en la imágen anterior, podemos observar el comportamiento claro que pertenecería a la construcción de un ***ClasificadorDebil***, en el cual simplemente se generarían un número (en el pseudocódigo, el número *k*) de clases ***Hiperplano***, y se quedaría con el mejor de todos ellos.

Finalmente, a dicho clasificador le hemos añadido una variable llamada ***alfa*** de tipo double, esta representa el **α** del pseudocódigo, y ese fragmento **START-END** se realiza antes de acabar el constructor.

```
        return H(x) = sign(Σ_t α_t · h_t(x))
```

Para finalizar, destacar que esta última franja hace referencia al método ***Evaluar*** de ***ClasificadorFuerte***, el cual consiste en multiplicar la confianza que da cada ***ClasificadorDebil*** que lo compone, por el resultado de ***Evaluar*** que proporciona este último.

Si la suma de todos ellos es positiva, indica que la imagen pertenece al tipo indicado, si el valor es negativo, indica que no.

Imagen

La clase *Imagen* es la usada para, como su nombre indica, para representar las imágenes que se usarán en la práctica.

La clase es entregada por el profesorado, y está compuesta por 784 píxeles (28x28), y cada píxel es una escala de gris, por lo que el valor de cada píxel será de entre 0 y 255 (8 bits ó 1 byte por píxel).

Hemos añadido dos cosas a la clase, por un lado un atributo llamado *type* de tipo *ImagenType* que representa el tipo. Hemos añadido algunos métodos como *getType()*, *setType()* y *isType()* que nos serán de ayuda en la implementación de la práctica.

getType() y *setType()*: se van a encargar de recibir y devolver un tipo para la imagen.
isType(): este método recibe por parámetros un *ImagenType*, o bien un *int*, devolviendo un *boolean* que nos indicará si pertenece al mismo *ImagenType*.

También hemos añadido a la clase proporcionada un atributo llamado *imageIntData*, el cual es de tipo *int[]*, y contendrá los valores de los 784 píxeles [0-255]. Inicialmente teníamos solo *imageData* de tipo *byte[]*, pero nos manejaremos mejor trabajando con *int[]*, por eso su uso.

En resumen, esta es la clase *Imagen*, es una clase sencilla que se inicializa a través de un archivo leyendo sus bytes, por lo que no hay mucho más que explicar.

Hiperplano

La clase ***Hiperplano*** se genera a través de lo que podríamos llamar una "*Imagen*" aleatoria, ya que se genera con los datos ofrecidos por el profesorado en la documentación.

$$h^D: \sum_{i=1}^{D} (x_i p_i) - C = 0$$

Como podemos observar en la imagen de la derecha, ***h*** sería el hiperplano, el cual se crea a través de un vector (x_i) y una imagen (p_i).

La "imagen" se usará **SOLAMENTE** para generarlo, una vez generado, almacenamos el vector y el valor de ***C*** (*C* es la *Constante* del hiperplano, usada para identificar a qué zona/punto del hiperplano pertenece un punto).

Entonces, ¿qué tendrá nuestro hiperplano? Sencillo, un atributo llamado *vector* de tipo *double[]*, el cual contendrá un vector unitario entre [1,-1] con valores *reales*. Además, un atributo llamado *C* de tipo *double*.

Para el factor aleatorio, hacemos uso de la siguiente instrucción, que nos proporciona una alteración para cada ejecución:

```
Random random = new Random(System.currentTimeMillis());
```

Hiperplano consta de los siguientes atributos:

-*double[] vector*: Este es el vector que ha generado su constructor, usada para clasificar.

-*double C:* Es la constante que ha generado su constructor, usada para clasificar.

-*double error:* Es la cantidad de errores que ha encontrado el hiperplano generado.

Hiperplano consta de los siguientes métodos:

-*Evaluar(img):* Este método devolverá la diferencia como valor de tipo *double* sobre aplicar una *Imagen* pasada por parámetro, multiplicando cada componente (píxel o p_i en la función) de la imagen por cada componente del atributo *vector* del *Hiperplano* (x_i en la función), dando como resultado una supuesta *"C"*, esta *C* es la que va a identificar a nuestra imagen, y si dicho valor da positivo tras restarle el atributo *C* del Hiperplano, identificamos la imagen como una coincidencia (es decir, si buscamos bolsos, y le pasamos un bolso, y da un valor positivo, es correcto, a su vez, si buscamos bolsos, pasamos un pantalón, y da un valor negativo, la clasificación realizada por el hiperplano también es correcta).

-*Clasificar(img):*Recibe una imagen por parámetros y devuelve si el signo de la evaluación.

ClasificadorDebil

La clase ***ClasificadorDebil*** se encarga de, como ya indicamos en el desglose, de generar varias clases ***Hiperplano***, y de seleccionar la mejor de todas ellas.

El número de hiperplanos a generar es parametrizable, y siempre procurará quedarse con el mejor de todos ellos.

Aunque a groso modo, la clase ***ClasificadorDebil*** ya ha sido explicada, debido a que es muy simple, y consta de un constructor que hace casi todo su trabajo, en este apartado nos centraremos en explicar su estructura, ya que su simple funcionamiento ha sido explicado.

ClasificadorDebil consta de los siguientes atributos:

-Hiperplano best: Este es el mejor hiperplano que ha generado su constructor.

-double alfa: Es el valor de **α**, el cual también es generado en el constructor con *best*.

ClasificadorDebil consta de los siguientes métodos:

-ClasificadorDebil(n,images,type): Es su constructor, y recibe por parámetros el número de *Hiperplano* que deberá generar (n), también una batería de *Imagen* para el entrenamiento (images), y además, un *ImagenType* el cual indicará el tipo de imágenes que se están entrenando (type).

-Clasificar(h,img): Este método es usado para clasificar una imagen según uno de los hiperplanos que anda generando el constructor.

-Clasificar(img): Este método es usado para clasificar una imagen a través de best.

Como conclusión, se podría definir al ***ClasificadorDebil*** como un generador y selector de hiperplanos, quedándose **solo con uno de ellos**, con el más prometedor, guardándolo su atributo ***best***.

ClasificadorFuerte

La clase ***ClasificadorFuerte*** se encarga de generar un número de clases de tipo ***ClasificadorDebil*** y almacenarlos si son prometedores, que serán aquellos que acierten más de un *50%* de las veces.

ClasificadorFuerte consta de los siguientes atributos:

-ArrayList<ClasificadorDebil> clasificadoresDebiles: Este *ArrayList* contiene todos los clasificadores débiles que ha generado el constructor del clasificador fuerte, **siempre y cuando, hayan acertado al menos el 50%** de las veces.

ClasificadorFuerte consta de los siguientes métodos:

-ClasificadorFuerte(entreno,test,n,h,type): Este es el constructor de ***ClasificadorFuerte***, es el encargado de la parte señalada en el primer desglose del pseudocódigo, la franja verde.
Este método recibe por parámetros: una cantidad de imágenes en un *ArrayList* para entrenar, y otra cantidad para testado también en un *ArrayList*, además, dos valores de tipo *int* llamados *n* y *h*, que identifican la cantidad de clases ***ClasificadorDebil*** y la cantidad de clases ***Hiperplano*** que generará cada uno de los anteriores respectivamente. Como último parámetro, recibe un parámetro *type* de tipo *ImagenType*, que define el tipo de imágenes a clasificar que va orientado el algoritmo ***AdaBoost***.

-GetAciertos(imagenes,type): Este método de *ClasificadorFuerte* se hará cargo de devolver un valor de tipo *int* que se identifica con la cantidad de aciertos que realizó sobre un conjunto de imágenes de testado. Este método recibe dos parámetros: por un lado *images*, el cual es un conjunto *ArrayList<Imagen>*, y otro *type* de tipo *ImagenType*.

-Evaluar*(img):* Este método devuelve un valor de tipo *int*, que representa que si la suma de la multiplicación de clasificar en todos los clasificadores débiles que contiene el clasificador fuerte por el *alfa* de cada *ClasificadorDebil* es positiva se retorna un 1, en caso contrario un -1. Este método recibe por parámetro una variable *img* de tipo *Imagen*, la cual es la que se va a evaluar por todos los clasificadores débiles que contiene el clasificador fuerte.

-Clasificar*(img,type):* Este método se encarga de devolver un *boolean* que representa que si al aplicar *Evaluar* de *ClasificadorFuerte*, esto nos devuelve un 1, y además es del mismo *ImagenType*, entonces la clasificación es correcta, en el caso de que *Evaluar* nos devolviera -1, y además son de distinto *ImagenType*, entonces también es una clasificación correcta. En cualquier otro caso, la clasificación es incorrecta. Este método recibe por parámetros una clase *Imagen* con el nombre *img*, y *type* de tipo *ImagenType*.

Entrenamiento y testado

Para el entrenamiento y el testado se han realizado investigaciones y lectura de documentación, a parte se ha apoyado la decisión de los comentarios realizados por el profesorado, y por deducción se ha decidido hacer uso de una distribución de 80% y 20% de imágenes dedicadas a entrenamiento y testado respectivamente.

Se ha intentado realizar un análisis más práctico de las diferentes cantidades de hiperplanos y clasificadores débiles que puede contener un clasificador fuerte, pero por alguna razón (probablemente un error de implementación), la clasificación es casi siempre la misma.

No puedo detallar más sobre entrenamiento y testado a un nivel más detallado por los problemas con los que me he encontrado a lo largo de esta práctica, los cuales están más detallados en la página 12.

Cuestiones

¿Cuál es el número de clasificadores (T) que se han de generar para que un clasificador débil funcione?
-Para que **funcione** propiamente dicho, valdría con un solo clasificador clasificador (T), o también llamado *Hiperplano* según nuestra implementación realizada.

¿Cuánto has entrenado cada clasificador (A)? ¿Por qué? Muestra una gráfica que permita verificar lo que comentas.
-Para comenzar, he comprobado que cantidad de *Hiperplanos(T)* son idóneos para un buen *ClasificadorDebil(A)*, y se deduce que por un lado, cuantos más *Hiperplanos* haya, mayor será la probabilidad de un *ClasificadorDebil* eficaz, pero el tiempo de generar un *ClasificadorFuerte* es cuadrático cuando el número de *Clasificadores Débiles* e *Hiperplanos* que usará es muy alto, por lo que tiene un coste computacional elevadísimo. Un ejemplo: Si un *Clasificador Débil* se genera con 1000 *Hiperplanos*, será seguramente un **buen clasificador débil**, pero, si queremos hacer uso de 1000 *Clasificadores Débiles* dentro de nuestro *Clasificador Fuerte*, tendríamos que enfrentarnos a una generación de 1000*1000 Hiperplanos, lo cual tendría como consecuencia un coste computacional (y por ello temporal) elevadísimo.

• **¿Cómo afecta el número de clasificadores generados y su entrenamiento al tiempo empleado para el proceso de aprendizaje? ¿Qué importancia le darías? Justifica tu respuesta.**
-El número de clasificadores débiles tiende a ser más eficaz cuanto mayor es el número usado, pero el entrenamiento, como ya se comentó anteriormente, se usó un 80% para obtener el mejor entrenamiento sin un **sobreentrenamiento**, que produjera un clasificador **muy bueno para las imágenes con las que se entrenó, pero muy malo con las demás**. La importancia es enorme, ya que si se usara solo un clasificador débil dentro del clasificador fuerte, tendríamos un clasificador fuerte que trabajaría como uno débil prácticamente, lo cual no es muy eficaz, y si tuviéramos un sobreentrenamiento tendríamos también un clasificador *"basura"*.

• **¿Cómo has dividido los datos en conjunto de entrenamiento y test? ¿Para que es útil hacer esta división?**
-Los he dividido en dos *ArrayList<Imagen>*, por un lado un *ArrayList* llamado *entreno*, y por otro lado *test*, repartiendo un 80% en el primero y el 20% restante en el segundo. Para finalizar, he insertado a cada uno su *ImagenType* para facilitar que la implementación pudiera ser lo más genérica y flexible a la hora de cambiar de tipo de imagen a entrenar y testear.
Es útil por que no tiene sentido entrenar un clasificador para que trabaje **solo con las de entreno** (al menos no en el ámbito de la práctica a realizar).

• ¿Has observado si se produce sobre entrenamiento? Justifica tu respuesta con una gráfica en la que se compare el error de entrenamiento y el de test a lo largo de las ejecuciones.

-Desgraciadamente, como muestro en el apartado ***Conclusiones***, no he podido hacer un análisis empírico correcto de los datos, a pesar de haber dedicado una cantidad desmedida de tiempo a esta práctica, no logré averiguar de donde provenía la falla que causaba este fenómeno.

• Comenta detalladamente el funcionamiento de Adaboost teniendo en cuenta los errores de clasificación que obtienes para aprendizaje y test.

-AdaBoost es un algoritmo encargado de, mediante una serie de pasos, clasificar una imagen lo más eficaz posible, si su composición es eficaz también (número de hiperplanos, de clasificadores débiles).

• ¿Cómo has conseguido que Adaboost clasifique entre las 8 clases cuando solo tiene una salida binaria?

-He añadido una clase *Enum* a la que he llamado *ImagenType*, la cual he usado para identificar a que tipo pertenece cada una de las imágenes.

• ¿Qué clases se confunden más entre sí? Justifícalo mediante las gráficas que consideres.

-Clasificar con un clasificador débil, en primera instancia con los clasificadores (al menos a nivel personal) ha sido una tortura entender su funcionamiento en comparación con el clasificador fuerte.

Conclusiones y dificultades encontradas

Como recomendación para futuras prácticas, pongan ejemplos de análisis hechos con un clasificador débil de por ejemplo 1 y 5 hiperplanos, y además, de 1 clasificador fuerte de 5 hiperplanos y 5 clasificadores débiles, explicando que rangos suelen toman, es decir, no es lo mismo que se tomen un rango entre 0% y 100%, que entre 40% y 60%, esto a nivel personal, ha sido muy frustrante, y quizás en esa parte de la práctica sea donde he perdido el hilo.

El principal problema encontrado es que por alguna razón la cantidad de aciertos obtenida es casi constante, **casi siempre** acierta las **mismas imágenes**, lo cual me ha sumergido en una incertidumbre de la que no he logrado salir.

A continuación, muestro un pantallazo de que, aun entrenando el algoritmo con distintos tipos de imágenes, distinta cantidad de hiperplanos y de clasificadores débiles, no hay variedad notable al uso del clasificador fuerte.

```
Loaded 4001 images...
Imagenes usadas para Entrenamiento: 3205
Imagenes usadas para Testeo: 796

En este tester vamos a mostrar cuan eficaz es un clasificador fuerte según el número de hiperplanos y clasificadores débiles...
TYPE= SUDADERA => Con 10 hiperplanos y clasificadores débiles, el porcentaje medio de aciertos es: 87.8140703517588% con 699.0 aciertos.
TYPE= SUDADERA => Con 10 hiperplanos medio de aciertos en un clasificador DEBIL es: 41.33165829145729% con 329.0 aciertos.

TYPE= BOLSO => Con 20 hiperplanos y clasificadores débiles, el porcentaje medio de aciertos es: 87.8140703517588% con 699.0 aciertos.
TYPE= BOLSO => Con 20 hiperplanos medio de aciertos en un clasificador DEBIL es: 36.05527638190955% con 287.0 aciertos.

TYPE= CAMISETA => Con 30 hiperplanos y clasificadores débiles, el porcentaje medio de aciertos es: 87.8140703517588% con 699.0 aciertos.
TYPE= CAMISETA => Con 30 hiperplanos medio de aciertos en un clasificador DEBIL es: 87.8140703517588% con 699.0 aciertos.

TYPE= PANTALON => Con 40 hiperplanos y clasificadores débiles, el porcentaje medio de aciertos es: 87.8140703517588% con 699.0 aciertos.
TYPE= PANTALON => Con 40 hiperplanos medio de aciertos en un clasificador DEBIL es: 82.91457286432161% con 660.0 aciertos.

TYPE= JERSEY => Con 50 hiperplanos y clasificadores débiles, el porcentaje medio de aciertos es: 87.8140703517588% con 699.0 aciertos.
TYPE= JERSEY => Con 50 hiperplanos medio de aciertos en un clasificador DEBIL es: 87.68844221105527% con 698.0 aciertos.

TYPE= VESTIDO => Con 60 hiperplanos y clasificadores débiles, el porcentaje medio de aciertos es: 12.185929648241206% con 97.0 aciertos.
TYPE= VESTIDO => Con 60 hiperplanos medio de aciertos en un clasificador DEBIL es: 87.8140703517588% con 699.0 aciertos.

TYPE= ZAPATILLA => Con 70 hiperplanos y clasificadores débiles, el porcentaje medio de aciertos es: 87.8140703517588% con 699.0 aciertos.
TYPE= ZAPATILLA => Con 70 hiperplanos medio de aciertos en un clasificador DEBIL es: 68.34170854271356% con 544.0 aciertos.

TYPE= ZAPATO => Con 80 hiperplanos y clasificadores débiles, el porcentaje medio de aciertos es: 12.185929648241206% con 97.0 aciertos.
TYPE= ZAPATO => Con 80 hiperplanos medio de aciertos en un clasificador DEBIL es: 87.8140703517588% con 699.0 aciertos.

BUILD SUCCESSFUL (total time: 4 seconds)
```

Como podemos observar, el clasificador fuerte casi siempre acierta **699** imágenes de **796** (20%), obteniendo una aproximación de *87,81%* de aciertos (salvo alguna puntual vez que ronda el 12,19%), haciendo uso de un entrenamiento de una cantidad de **3205** (80%) de las imágenes de un total de 4001 imágenes (100%).

Como conclusión final, he entendido el funcionamiento de AdaBoost, he logrado un funcionamiento parcial del algoritmo, **pero**, es una pena que no logre entender por que pasa lo anteriormente descrito.

Bibliografía

La información necesaria para resolver dudas se ha obtenido gracias al profesorado, compañeros de clases, documentación ofrecida para la práctica y, adicionalmente, lo siguientes enlaces:

¿Qué es y cómo funciona AdaBoost?
Uso de la clase Random
Semilla para algoritmo aleatorio
Guardar y leer objetos (Clasificadores Fuertes) en un archivo con Java