Luis Márquez Carpintero

53978593P

Universitat d'Alacant
Universidad de Alicante

Sistemas Inteligentes

CURSO 2017-2018

[Desarrollar un sistema capaz de distinguir entre dígitos manuscritos. Mediante un sistema de aprendizaje automático supervisado (Adaboost).]

Práctica 2. Aprendizaje supervisado.

Contenido

Introducción

Para el funcionamiento de la práctica hemos debido realizar la implementación de un algoritmo de aprendizaje automático supervisado, *Adaboost*.

Gracias a este algoritmo debemos de ser capaces de identificar cada uno de los números manuscritos de la base de datos, para ello, de nuestra BBDD crearemos dos grupos, el primero que nos servirá para clasificar las imágenes (tamaño de un 80% aprox.), y el segundo -conjunto de test- con un tamaño del 20% restante, lo emplearemos para ver si nuestro *Adaboost* funciona correctamente y clasifica las imágenes de *test* según lo esperado.

En este documento realizaremos y detallaremos el funcionamiento del código que he implementado, la explicación del algoritmo, así como algunos aspectos problemáticos a tener en cuenta en la correcta implementación y funcionamiento del ya mencionado algoritmo.

Antes de empezar querría destacar que para la implementación hemos debido descargar el proyecto -de Netbeans- facilitado en el Moodle de la asignatura. Esta plantilla de Netbeans, ya viene con una clase *Imagen* creada y con la configuración necesaria para acceder a la BBDD, así como, un ejemplo para poder acceder a las imágenes de la base de datos.

La base de datos es fundamental, ya que, debe ser bastante voluminosa, esto nos permite destinar una mayor cantidad de imágenes al entrenamiento y obtener un resultado más fiable, al mismo tiempo, nos facilita detectar nuevos patrones para cada uno de los dígitos.

En nuestro caso concreto en el fichero "*Main.java",* renombrado *"Adaboost.java"*, hemos añadido todas nuestras funciones, incluido nuestra función *Adaboost*, encargada de llamar al resto de funciones del *Main.java.* Aunque, bien es cierto que hemos modificado la clase *Imagen* para insertar los campos *"Imagen"* y *"peso",* el primero contiene el digito al que se asocia la imagen, y el segundo el peso asignado a dicha imagen.

Como nos indica el enunciado de la práctica esta tiene tres formas de ejecución:

1.- `Adaboost –t <fichero_almacenamiento_clasificador_fuerte>`, realizaremos el entrenamiento hasta hallar con todos los clasificadores débiles y este lo guardaremos en un fichero pasado por parámetro.

2.- *Adaboost <fichero_origen_clasificador_fuerte> <número_imagen_prueba>* leeremos los clasificadores débiles del fichero pasados por parámetro y lo ejecutaremos sobre una imagen de la BBDD.

3.- Por defecto, Interpretamos del enunciado que ejecutaremos sobre todas nuestras imágenes de test los clasificadores fuertes tras el entrenamiento y además mostraremos el número de fallos y el número de aciertos por consola.

Pseudocódigo implementado

El Pseudocódigo implementado que a continuación se muestra, ha sido ligeramente modificado, partiendo del facilitado en el enunciado de la práctica, para la implementación del *Adaboost*, este ha sido adaptado para que se ajuste con más precisión a lo implementado en mi práctica, basándose este a su vez en lo especificado en el las transparencias de teoría, así como, en el enunciado.

He añadido comentarios en zonas de código con la intención de explicar y detallar lo que hace cada parte del pseudocódigo implementado.

```
// numActual -> Numero con el que trabajra nuestro adaboost, del 0 al 9
// Porcentaje -> Porcentaje de elementos para el conjunto de entrenamiento
// T -> Numero de clasificadoes debiles que tendrá nuestro adaboost
// imgTest -> Número identificativo de la imagen en el conunto de test.
// N -> Número identificativo de la imagen en el conunto de entrenamiento.
// D[] -> Peso de cada una de las imagenes.
Alg Adaboost (numActual, Porcentaje, T, imgTest)
      //Inicialmente el peso de cada una de las imagenes de es Dt=1/N
      para i=1 hasta N// N imgs entrenamiento
            D[i] = 1/N
      fpara
      para t=1 hasta T //Para cada clasificador debil
            /*Genero muchos clasificadores debiles hasta encontrar
            uno con el error menor a 0,5 y 1000 iteraciones como poco*/
            para errormin>0.5 y i<100
                  crearClasificadorDebil();
                  /*Generar humbral pixel y valor del umbral aleatorio y compararlo por
                  cada img de entrenamiento obtener el error */
                  Si el errorClasificador< ErrorMasPequeño
                        ErrorMasPequeño = errorClasificador
                        guardarValoresRandomClasificador
                        εt = Suma pesos imágenes mal clasificadas
                        valoresAleatorios = nuevosValoresAleatorios
```

```
                fsi
            fpara

            //calcular confianza
            ht = αt = ½ Ln(1-εt/εt)
            para cada imagen
            //Actualizar distribución D sobre el conjunto de entrenamiento:
                Dt'[i] = Dt[i]*e^(-at+yi*ht) / Zt
            fpara
            para cada imagen
            //Actualizados los pesos calculamos Z:
                Z = ∑_i Dt[i]_i
            Fpara
            para cada imagen
            //Ajustamos de nuevo los pesos de las imagenes:
                Dt'[i] = Dt'[i] / Zt
            Fpara
        fpara
        //Aplicamos valores clasificadores debiles a la imagen seleccionada
        AplicarValoresTestImg(1)
        devolver ∑_i ∝ [cfi] * h[i]
falg
```

Funcionamiento

- División de los datos en conjunto de entrenamiento y test

Para que el algoritmo *Adaboost* funcione, como algoritmo supervisado que es, deberemos dividir el conjunto de datos en dos subconjuntos, el subconjunto de entrenamiento y el de test.

- **Conjunto para entrenamiento**: Usado para entrenar los clasificadores débiles y obtener una serie de umbrales que se adapte correctamente a los patrones en los pixeles de dicho número.

- **Conjunto para tets**: Una vez entrenado desearemos conocer el grado de fallos que obtenemos con nuestro, *Adaboost*, ahí es donde aplicamos el conjunto de entrenamiento.

El conjunto de datos contiene cerca de 1000 imágenes, nosotros realizamos la división en un porcentaje pasado por parámetro a nuestro *Adaboost* (especificado en la sesión5 del enunciado). Es recomendable destinar significativamente más imágenes al conjunto de entrenamiento que al test, alrededor de un 70-30, o un 80-20 son buenas cifras.

En nuestro código lo tenemos en un 80-20, pero dicho porcentaje se puede modificar cambiando uno de los parámetros que recibe nuestro *Adaboost*.

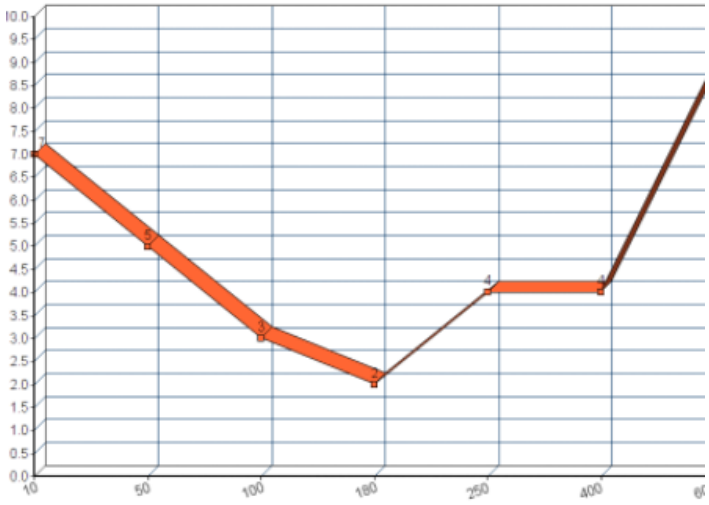
La selección de imágenes para cada subconjunto se ha implementado de la siguiente forma, escogemos un 80% de las imágenes de cada digito y las añadimos a una colección de entrenamiento, mientras que para el 20% restante las añadimos a la colección de test.

Al dividir nuestras imágenes en estos dos conjuntos conseguimos que tras ajustar y pasar la fase de entrenamiento con éxito podamos enfrentarnos a imágenes nunca vistas, consiguiendo una especie de simulacro de lo que sucedería con imágenes nuevas, haciéndonos una idea de si funciona bien o necesitamos modificar nuestro algoritmo. Si estas imágenes de test, son superadas la gran mayoría de las veces, quiere decir que hemos realizado un buen entrenamiento, y que el algoritmo funciona como se espera, de lo contrario, o bien, hemos implementado mal el algoritmo *Adaboost* o debemos reajustar los parámetros, como la cantidad de clasificadores débiles, o el número de imágenes empleadas para entrenamiento.

- Número de clasificadores para que un clasificador débil funcione

El número de clasificadores débiles que se han de generar para que un clasificador fuerte funcione correctamente varía en función de cómo hayamos implementado el algoritmo y como de eficaz sea. Esto debemos evaluarlo y sopesarlo nosotros mismos, en nuestro caso pensamos que con cerca de 180 clasificadores débiles es una cantidad adecuada como para poder identificar los dígitos correctamente sin producir sobre entrenamiento.

En la siguiente gráfica, podemos observar como con un número distinto de clasificadores débiles (t), el número de fallos va aumentando progresivamente hasta alcanzar un pico en el gráfico.



Para conseguir los datos del gráfico nos ha bastado con ir realizando múltiples ejecuciones, siendo estos los datos obtenidos:

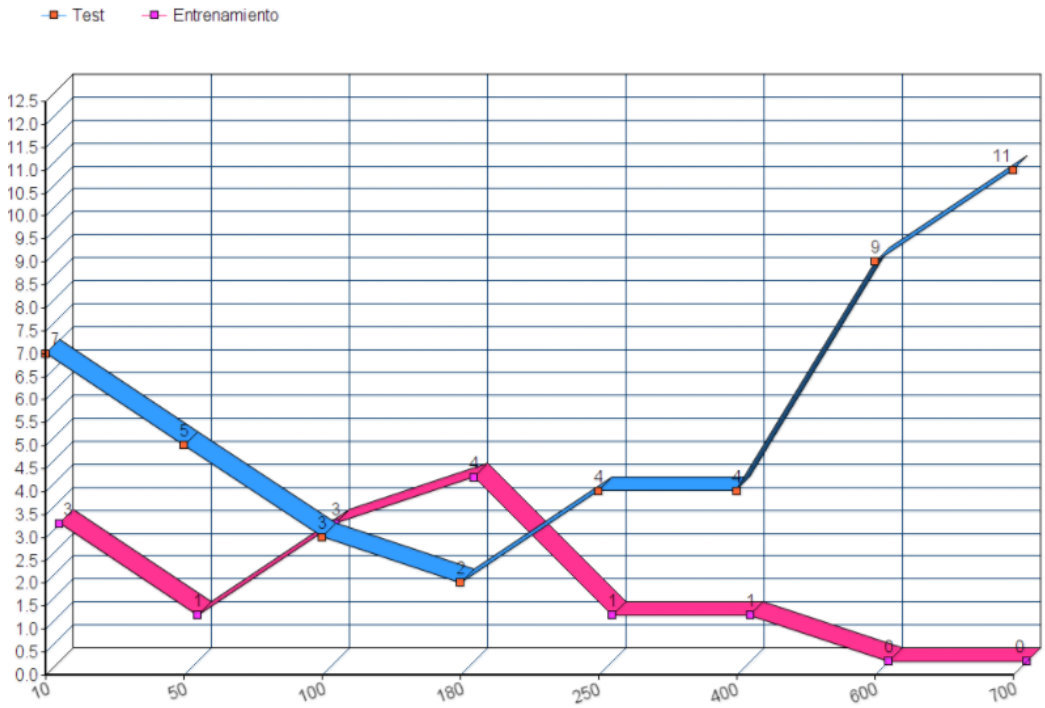
Número de clasificadores débiles		**Errores en la fase de tests**
Para T = 10	→	7 errores del conjunto de test
Para T = 50	→	5 errores del conjunto de test
Para T = 100	→	3 errores del conjunto de test
Para T =180	→	2 errores del conjunto de test
Para T = 250	→	4 errores del conjunto de test
Para T = 400	→	4 errores del conjunto de test
Para T = 600	→	9 errores del conjunto de test

Este error que observamos, o bien, se debe a una falta de entrenamiento (los valores a la izquierda del valor óptimo de la gráfica -180-), o bien, a un sobre entrenamiento del clasificador débil, esto se produce cuando hemos realizado más de 180 clasificadores débiles o más. Este error, producido por el exceso de clasificadores débiles -sobre entrenamiento-, el sobre entrenamiento se produce por crear demasiados clasificadores débiles con el fin de disminuir el error en el conjunto de entrenamiento. Es decir, el sobre entrenamiento es un resultado tan *"a la medida"* del conjunto de entrenamiento -con unos fantásticos resultados-, pero que al llegar al conjunto de test falla en exceso por culpa de este sobre entrenamiento.

- ## Observación sobre si se produce sobre entrenamiento

Como mencionábamos, en nuestro *Adaboost* hemos llegado a encontrar un momento en el que atribuimos una serie de errores inesperados al sobre entrenamiento, no obstante, modificando la cantidad de clasificadores débiles (T), hemos conseguido reparar nuestro algoritmo para hallar un funcionamiento óptimo.

En las siguientes gráficas se puede observar el comportamiento de nuestro programa con los errores típicos del sobre entrenamiento.



Gráfica Nº2

Como podemos apreciar en la gráfica a partir de los 400 clasificadores débiles empezamos a notar una diferencia más notoria entre los resultados del test y los de las imágenes de entrenamiento. Podemos diferenciar la gráfica en tres zonas, la primera con pocos clasificadores, que nos genera un volumen de fallos mayor, la segunda donde se encontrarían los valores que yo consideraría óptimos cercanos al 180, y la ultima zona cerca de los 400 clasificadores débiles en los que apreciamos que el error en las imágenes de test aumenta considerablemente, mientras que en las de entrenamiento su error se aproxima al 0 la gran mayoría de las veces. Como conclusión en este apartado podríamos remarcar la gran importancia de no sobreentrenar las imágenes (a pesar de que en el conjunto de entrenamiento mejoren los resultados), y que conforme más aumente la brecha entre el conjunto de entrenamiento y entre el conjunto de test, mayor es el sobre entrenamiento que le estamos aplicando a nuestro Adaboost.

- Importancia del número de clasificadores generados con respecto al tiempo

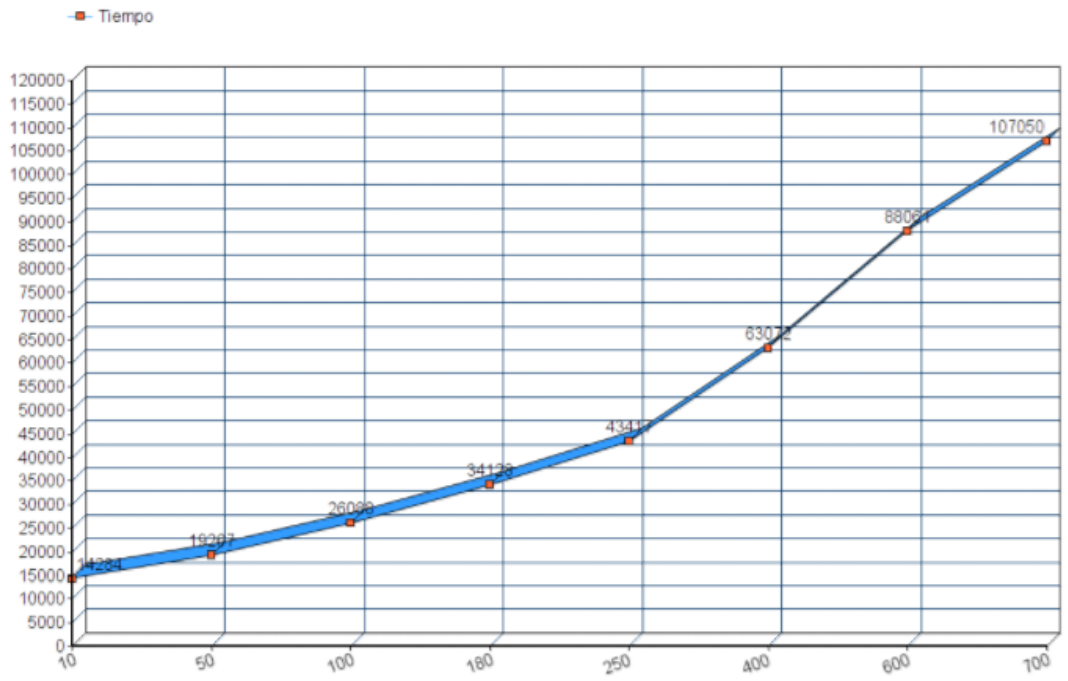
Gracias a las trazas colocadas sabemos que para una imagen (10 *Adabooost's*) nuestro algoritmo tarda aproximadamente 1234 milisegundos, más o menos una media de 12,8 ms por cada clasificador fuerte y 0,05ms por cada clasificador débil, éste es un tiempo poco significativo, no obstante visto desde lo que tardamos con 10 *Adaboost* hay que multiplicarlo por el número de imágenes del test, bien es cierto que tenemos la condición de parada más rápida para el clasificador débil (en conseguir un error menor a 0.5 parar de buscar), aun siendo el tiempo relativamente bajo no podemos prolongar todo lo necesario hasta obtener el mejor resultado, deberemos obtener una relación (calidad resultado)/(tiempo de computo) por ello, por ejemplo, nuestro clasificador débil para una vez encontrado un error menor a 0.5. De este modo conseguimos que por lo menos el peor de los resultados sea menor a 0.5, evitando iteraciones innecesarias que no deben porque mejorar el valor ya encontrado.

```
Loaded digit 6
Loaded digit 7
Loaded digit 8
Loaded digit 9
Loaded 1000 images...
La img nº403 que en realidad es un 4:
La imagen es un 4 y en realidad es un 4
Tiempo ejecucion 10 Adboost 250 clasificadores debiles por Adaboost: 1234
BUILD SUCCESSFUL (total time: 1 second)
```

Imagen1: Traza para saber el tiempo de ejecución 10 CF x 250 CD, 1234 miliseconds

Es probable que individualmente pueda parecer un tiempo pequeño pero que, tras ser aplicado a cada imagen del test por cada uno de los diez clasificadores fuertes, se convierte en un tiempo nada despreciable, que conviene optimizar.

En esta gráfica podemos observar como cambia el tiempo en función de la cantidad de clasificadores débiles a utilizar, hemos extraído los datos añadiendo una traza del tiempo de ejecución y realizando múltiples pruebas.



Observamos como según aumentamos el número de clasificadores débiles por cada clasificador fuerte, el tiempo de ejecución se hace considerablemente más alto.

Por todo ello, considero que el tiempo en el calculo tiene una importancia elevada ya que sobre todo si poseemos un conjunto de entrenamiento mayor al que disponemos en esta práctica o deseamos disponer de más de 10 clasificadores fuertes (por ejemplo, todas las letras del abecedario) podemos aumentar el tiempo de calculo considerablemente.

Además, tampoco debemos perder de vista la aplicación real de este algoritmo, y es que en muchos casos pretendemos obtener en tiempo casi real los números ya identificados, para ser enviados inmediatamente a una aplicación informática. En casos como, por ejemplo, algunas funciones de Google Translate y One Note que incorporan un mecanismo en el que transcriben los caracteres escritos manualmente inmediatamente.

En definitiva, debemos ser conscientes de lo que supone mejorar el algoritmo acosta de algo de tiempo, debemos ser nosotros mismos los encargados de evaluar y medir que aplicación práctica le vamos a dar al algoritmo, y si merece la pena realizar una mayor cantidad de clasificadores débiles a pesar de perder algo de tiempo.

- ## Funcionamiento del AdaBoost

Nuestro *Adaboost*, concretamente el de nuestro código, está fuertemente inspirado en el pseudocódigo de la práctica, aunque no se trata de una implementación idéntica del algoritmo.

Paso por paso el funcionamiento de nuestro *Adaboost* es el siguiente. Primero, cargamos las imágenes de la base de datos digito a digito, de cada digito nos quedamos con un porcentaje adecuado y las vamos añadiendo a un conjunto de entrenamiento, el porcentaje restante al conjunto de test, a continuación, asignamos a cada imagen un peso ($\frac{1}{N\ entrenamiento}$). El porcentaje destinado a cada fase se recibe por parámetro (nosotros hemos indicado 80).

Hecho esto vamos a realizar una serie de clasificadores débiles (T), por cada clasificador débil vamos a:

1. **Generar un umbral**, dicho valor corresponde a un color de gris de 0 a 255 -en nuestro código de -127 a 127- debido a que almacenamos en que lo vamos a comparar con un tipo byte (-127, 127).

2. **Generar una dirección,** diremos si el color del pixel es mayor (+1) o menor (-1) que el valor generado para el umbral -punto anterior-.

3. **Generar pixel**, este pixel indica la posición en la imagen en la que tomaremos el valor de gris, en nuestra práctica generamos dos valores aleatorios, el del pixel que corresponde con el ancho y el del alto de la imagen, estos valores los multiplicaremos entre sí para obtener el número de a que pixel global queremos acceder.

Generados estos valores aleatorios debemos acceder al pixel generado aleatoriamente y ver si la tonalidad de gris es superior (dirección del umbral 1), o inferior (dirección umbral -1), en el caso de entrenar con imágenes que no sean del digito al que corresponde el algoritmo fuerte, buscaremos el caso contrario, es decir “un fallo” en el entrenamiento es un éxito en la aplicación y viceversa.

Cada vez que no se acertemos con los valores aleatorios generados incrementaremos nuestro error, siendo este la suma de los pesos de las imágenes mal clasificadas.

Para cada clasificador débil buscamos que por lo menos acierte más del 50% de las veces, es decir, que su error sea menos a 0,5 para ello generamos continuamente clasificadores débiles (con valores al azar), hasta hallar uno con un error inferior a ese 0,5. Encontrado este clasificador débil producido, insisto, de forma aleatoria. Almacenaremos los datos generados aleatoriamente (umbral, dirección del umbral, pixel) del clasificador.

Encontrado los datos del clasificador débil cuyo error no supere el 50% para dicho digito, debemos de generar su confianza (α) que se realiza mediante la fórmula $\alpha = \frac{1}{2}\ln(\frac{1-\epsilon t}{\epsilon t})$ siendo $\epsilon\ t$, el error calculado en el paso anterior.

Conociendo el valor de α ya podemos actualizar los pesos de las imágenes (inicialmente inicializados a $1/N$), para actualizarlo realizaremos imagen a imagen la siguiente formula, donde Dt es el peso de la imagen a actualizar.

Para actualizar dichos pesos vamos a dividirlo en tres pasos:

Paso 1: Actualizamos los pesos, donde e es, at, yt, ht,

$$D_t' = D_t * e^{-\alpha t + yt + ht}$$

Paso 2: Calculamos Z, donde Z es el sumatorio de os nuevos pesos de todas las imágenes, Z es la constante de normalización y sirve para que posteriormente al dividir, la suma de los pesos no de mayor que uno.

$$Z = \sum_i D_t$$

Paso 3: A los pesos nuevos generados en el primer paso lo dividimos entre Z, como hemos mencionado su finalidad es que la suma de pesos de todas las imágenes no sea mayor que uno.

$$D_t' = \frac{D_t}{Z}$$

Podemos verificar y comprobar que hemos hecho bies estos pasos si al sumar los nuevos pesos de las imágenes de uno:

```
La suma de los pesos es 0.9999997
```

Llegado el momento, va siendo hora de realizar nuestra fase de tests, cogeremos una imagen del conjunto de tests y le aplicaremos todos los clasificadores débiles generados hasta el momento (por eso los guardamos una vez dan un valor aceptable, inf. 0,5).

Con la lista de resultados obtenida de los clasificadores débiles le aplicamos la siguiente formula, que corresponde con la salida de nuestro clasificador fuerte, Adaboost.

$$H(x) = sign(\sum_t \alpha t \cdot ht(x))$$

Como observamos en la formula, obtenemos nuestra como consecuencia de las sumas del producto de la confianza y hs obtenidas por cada clasificador débil. De esto solo nos interesa el signo, ya que, el valor de H, es

nuestro veredicto, siendo positivo cuando dice que la imagen corresponde al número asociado a nuestro clasificador fuerte.

Como apreciamos deberemos tener un clasificador fuerte por cada uno de los dígitos a clasificar, debido a que la salida del algoritmo (*H*) es binaria y existe la posibilidad de los falsos positivos, deberemos idear una forma de quedarnos con el correcto, a continuación, detallaré el que he empleada para la práctica.

- ## Método para clasificar los 10 dígitos cuando solo tienen una salida binaria

Conseguir que una imagen pase los diez *Adaboost* correctamente en teoría no debería suponer ningún problema, pues solo se da el caso en el que 9 *Adaboosts* den menos uno, y el Adaboost restante +1, clasificando el dígito rápidamente como perteneciente al número de ese *Adaboost*.

En los casos prácticos, puesto que no es un algoritmo perfecto, da falsos negativos y falsos positivos, ante los falsos negativos, si en ese momento algún otro clasificador fuerte da un falso positivo, no podremos hacer nada. No obstante, cuando nos da falsos positivos miraremos la confianza de cada clasificador y seleccionaremos como bueno aquel que mayor confianza nos genere.

Barajé, varias opciones para hallar el método para seleccionar aquel clasificador perteneciente al número, las dos opciones que más fuerza tomaron han sido.

1. Repetir los *Adaboost* que den positivos hasta que solo uno de ellos de positivo, hemos descartado esta opción porque aumenta considerablemente el tiempo de computo.
2. Almacenar la confianza generada a cada clasificador fuerte y hacer más caso a aquella cuya confianza sea mayor.

Por motivos de tiempo de computo, como ya se ha mencionado antes, hemos decidido quedarnos con la segunda opción.

Conclusiones

En esta práctica hemos debido de realizar la realización del algoritmo adaboost para poder clasificar correctamente una serie de imágenes, ésta práctica tiene una clara aplicación real que es la de poder identificar los números (o incluso dado el caso letras) escritas manualmente.

Adaboost es un algoritmo muy útil que nos permite identificar figuras en la imagen, además nos vamos desenvolviendo en términos como lo es el entrenamiento y los test, nos ha resultado difícil realizar determinadas comprobaciones y reparar ciertos bugs, debido a que no sabemos el valor aproximado que ronda algunas variables. También nos ha complicado la situación de que se ha resubido el enunciado ampliando algunas partes, y en consecuencia, nos ha costado más de lo que hubiese sido habitual implementar las ampliaciones en nuestro código, eso sin mencionar el poco tiempo que le he podido dedicar a mejorar y comprobar el funcionamiento.

Resulta una práctica, bajo mi punto de vista, algo complicada de implementar, ya que no solo no sabemos el rango aproximado de ciertas variables, sino que el hecho de que nos den algunas formulas en las que debemos confiar para un buen funcionamiento, dificulta la ejecución.

La parte más costosa de la realización de la práctica, para mi ha sido la de entender y asimilar el funcionamiento del pseudocódigo y del algoritmo Adaboost así como los conceptos de teoría, ya que en la programación no hemos encontrado ninguna dificultad.