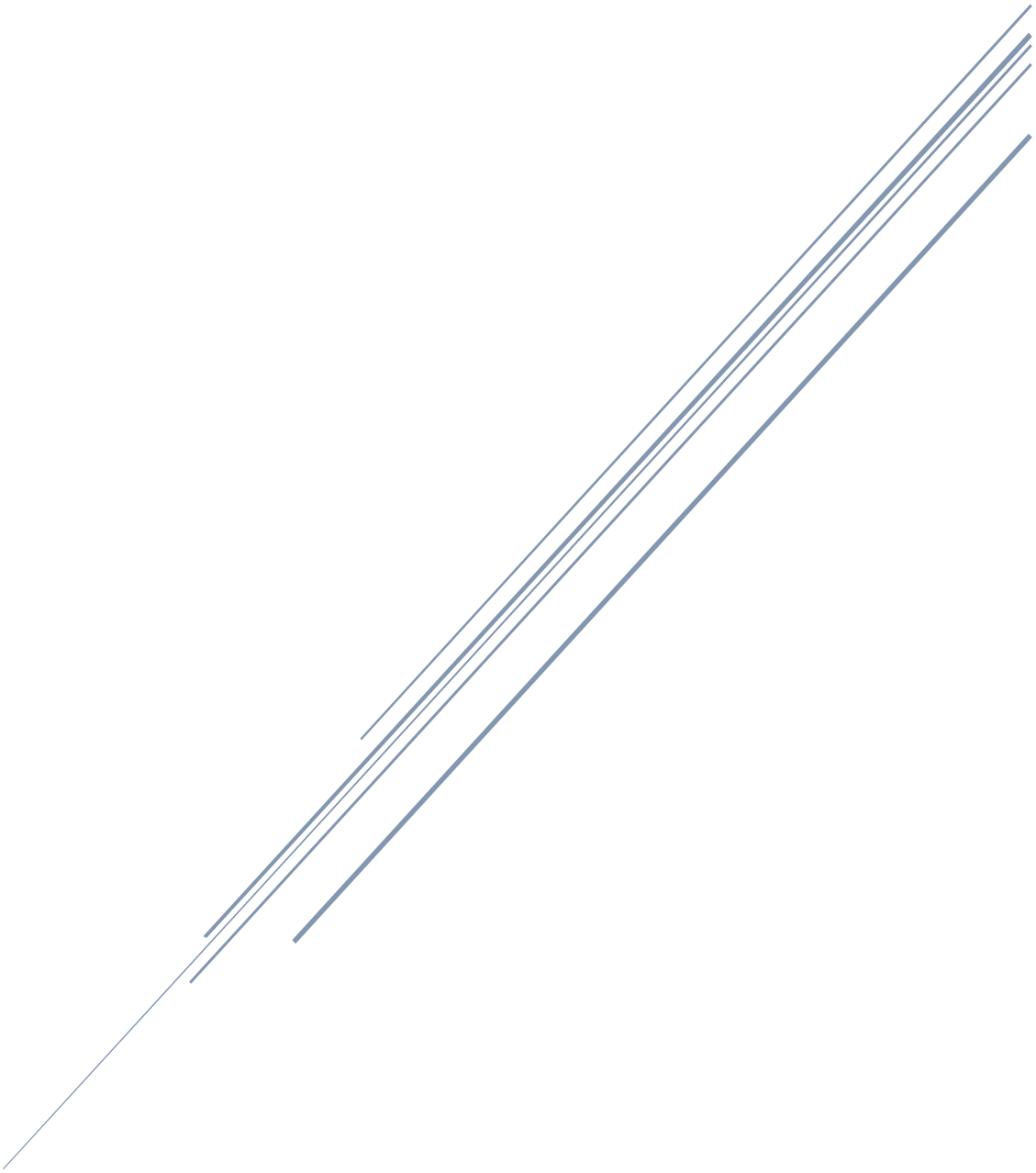
SISTEMAS INTELIGENTES



Raúl Beltrán Marco
23900664 F

ÍNDICE

Introducción

En esta práctica vamos a ver el funcionamiento y la aplicación del algoritmo **Adaboost,** así como su implementación en Java.

En esta práctica vamos a tener como **objetivo** crear un programa el cual sea capaz de reconocer una imagen elegida por el usuario; como **limitación** estas imágenes deberán pertenecer a una de las 10 clases de las cuales disponemos en nuestra base de datos: **Cifar10_2000,** además de que deben tener un tamaño de 32x32 píxeles.

Categoría	Objeto	Ejemplos
0	Aviones	
1	Automóviles	
2	Pájaros	
3	Gatos	
4	Ciervos	
5	Perros	
6	Ranas	
7	Caballos	
8	Barcos	
9	Camiones	

Para ello, haremos uso del algoritmo Adaboost con el cual crearemos tantos Clasificadores Fuertes como categorías de imágenes tengamos. Más adelante se explicarán los conceptos de Clasificador Fuerte y Clasificador Débil en sus respectivos apartados.

Es necesario indicar que este programa se rige por una serie de parámetros los cuales hemos declarado de forma global para poder utilizarlas a lo largo del programa.

```
private static final int T= 55;//Número de clasificadores dbiles a usar 50
private static final long A= 10000;//Número de pruebas aleatorias 5000
private static final int M= (32*32*3) - 1;//Dimensión de la imagen
private static final int P= 80;//Porcentaje de imagenes de avion
private static final int numClases= 10;
```

- ***T:*** Número máximo de clasificadores débiles que formarán un clasificador fuerte al final del Adaboost.
- ***A:*** Número máximo de pruebas aleatorias que se realizan para obtener un Clasificador Débil.
- ***M:*** Tamaño de las imágenes teniendo en cuenta la codificación RGB. (3 canales de color)
- ***P:*** Porcentaje de imágenes de un conjunto que se utilizarán para la fase de entrenamiento.
- ***numClases:*** Es el número total de categorías que tenemos en nuestra base de datos

Otro concepto importante para tener en cuenta antes de explicar el resto del programa es el significado de las variables de **Y, X** y **D**.

- **X:** Conjunto de imágenes, estas imágenes serán las que se utilicen para la creación de los Clasificadores Débiles y Fuertes. Su tamaño varía dependiendo del porcentaje de imágenes que se desee utilizar (P).
- **Y:** Conjunto de soluciones, este vector siempre debe tener el mismo tamaño que X, debido a que se utilizará para verificar si el Clasificador Débil o Fuerte ha acertado a la hora de clasificar la imagen.
- **D:** Conjunto de pesos, este vector siempre debe tener el mismo tamaño que X, este conjunto sólo se utiliza en la función Adaboost para indicar como de grave es el error que ha cometido el clasificador cuando falla, a medida que va avanzando el algoritmo los valores de este conjunto van cambiando.

Es necesario también aclarar las variables **Xtest** e **Ytest**, las cuales tienen el mismo significado y función que las variables X e Y, pero estas no serán utilizadas a la hora de realizar el test de los clasificadores Fuertes, además, X será un conjunto de imágenes que el Clasificador Fuerte nunca ha visto y, por tanto, podremos obtener un valor real de como de efectivo es dicho Clasificador Fuerte.

Dichos conjuntos se inicializan de la siguiente manera y serán modificados acordes al valor P que se haya elegido con la función **generaConuntos**.

```
for(int i=0; i < numClases; i++) {

    File carpeta = new File("./cifar10_2000/" + i);
    File[] lista= carpeta.listFiles();//Obtenemos el número
    int aux= lista.length * P/100;
    int resto= lista.length - aux;
    //Entrenamiento
    ArrayList<Imagen> X= new ArrayList<>();
    int[] Y =  new int[lista.length*2];
    //Test
    ArrayList<Imagen> Xtest= new ArrayList<>();
    int[] Ytest =  new int[lista.length*2];

    generaConjuntos(X, Y, Xtest, Ytest,aux, resto, i);
```

*El tamaño de Y se inicializa con valor de lista.length*2 para que no aparezca la excepción ArrayOutOfBoundException de java. (No se puede hacer un ArrayList<int>, así que no puede ser dinámico).

ClasificadorDebil.java

Esta clase se encarga de gestionar todo lo relacionado con el Clasificador Débil.

En esta práctica el Clasificador Débil se trata de un elemento que está principalmente compuesto por un **pixel**, un **umbral**, una **dirección** y una **confianza**.

- **Píxel**: Es un valor entre 0 y 3072. Indica que píxel de la imagen vamos a clasificar.
- **Umbral**: Es un valor entre 0 y 255. Indica el valor numérico que sirve para determinar si pertenece o no pertenece.
- **Dirección**: Sólo puede valer 1 o -1. A la hora de aplicar el Clasificador Débil nos indicará si debemos comprobar si el umbral es mayor o menor.
- **Confianza**: Es un valor el cuál indica la seguridad del propio Clasificador Débil de que la salida que ha devuelto

En mi código además le he añadido atributos como el **array de int prediccion** el cual se utilizará para almacenar los resultados que devuelve el Clasificador Débil al clasificar un conjunto de imágenes y dos variables **fallos** y **error**, las cuales almacenarán el número de veces que se equivoca el clasificador Débil y su error correspondiente.

Es importante destacar que siempre que se crea un Clasificador Débil estos 3 atributos son generados de forma **aleatoria** haciendo uso de la clase **Random** de Java de la siguiente manera:

```
public static ClasificadorDebil generarClasifAzar(int dimension, int size) {
    Random rng= new Random();
    int pxl= rng.nextInt(dimension);
    int umb= rng.nextInt(255);
    int direc= rng.nextInt(2);
    if(direc == 0)
        direc= -1;
    return new ClasificadorDebil(pxl, umb, direc, size);
}
```

Esta clase se basa en dos principales funciones: **aplicarClasifDebil** y **obtenerErrorClasif.**

En la función **aplicarClasifDebil**, solamente le pasaremos como parámetros el número de imágenes que va a clasificar para finalmente modificar el **vector de int: prediccion**. Cada elemento de este vector sólo puede valer **1** o **-1**, lo cual indica que **pertenece** o **no pertenece** al conjunto respectivamente, debido a que Adaboost sólo devuelve una salida binaria.

```
public void aplicarClasifDebil(ArrayList<Imagen> X) {

    for(int i=0; i < X.size(); i++) {
        if(this.direccion == 1) {//Pertenece a la clase
            if(Adaboost.Byte2Unsigned(X.get(i).getImageData()[this.pixel]) > this.umbral)
                this.prediccion[i]= 1;
            else
                this.prediccion[i]= -1;
        }
        else {//No pertenece a la clase
            if(Adaboost.Byte2Unsigned(X.get(i).getImageData()[this.pixel]) > this.umbral)
                this.prediccion[i]= -1;
            else
                this.prediccion[i]= 1;
        }
    }
}
```

Por cada imagen del conjunto, seguiremos los siguientes pasos para aplicar el Clasificador Débil:

1. Mirar la dirección del Clasificador Débil.
2. Comparar el umbral del Clasificador Débil, con el umbral de cierto pixel de la imagen, dicho pixel de la imagen deberá ser el mismo que el que está almacenado en la variable pixel del Clasificador Débil.

- Si **dirección** vale **1**, en el caso de que el umbral de la imagen sea **MAYOR** entonces el Clasificador Débil dice que la imagen **pertenece** (1), en caso contrario, **no pertenece**.
- Si **dirección** vale **-1**, en el caso de que el umbral de la imagen sea **MAYOR** entonces el Clasificador Débil dice que la imagen **no pertenece** (1), en caso contrario, **pertenece**.

En la función **obtenerErrorClasif,** le pasaremos como parámetros las imágenes que se han clasificado, así como las variables Y y D anteriormente explicadas.

En esta función básicamente haremos un sumatorio añadiendo el peso de cada imagen a medida que el clasificador vaya aumentando hasta finalmente almacenarlo en la variable error.

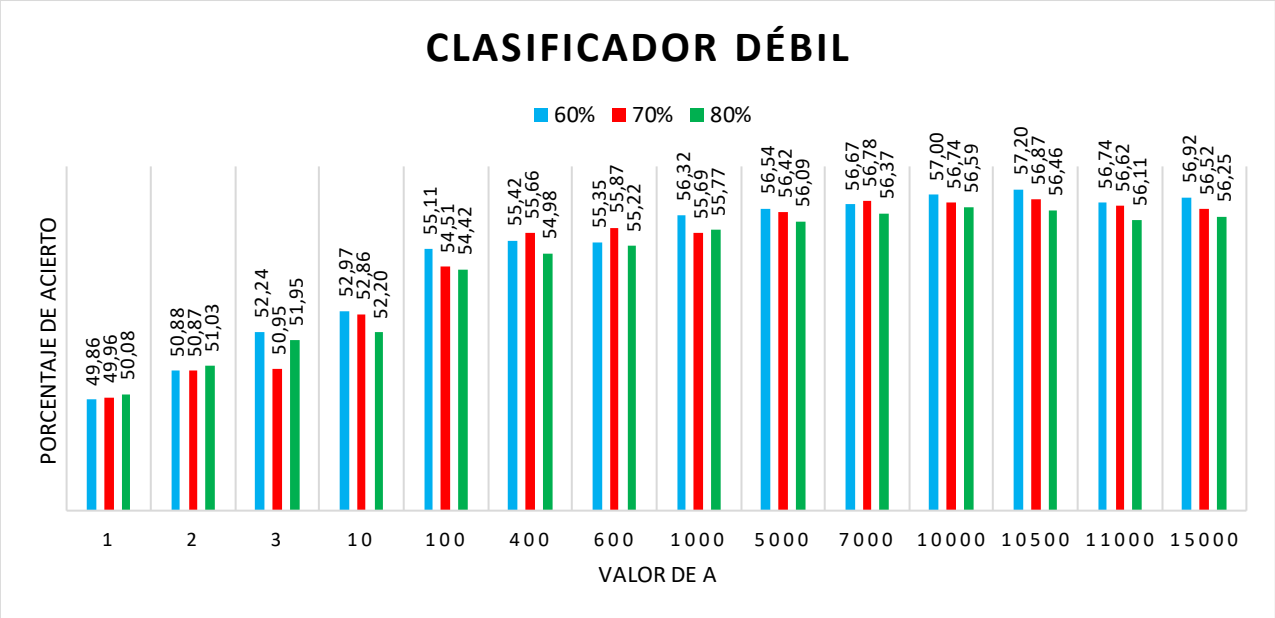
```
public void obtenerErrorClasif (ArrayList<Imagen> X, int[] Y, double[] D) {

    double aux= 0;
    int fallos=0;
    for(int i=0; i < X.size(); i++)
    {
        if(this.prediccion[i] != Y[i]) {//El Clasificador ha fallado
            aux += D[i];//Le sumamos el peso de esta imagen
            fallos++;
        }
    }
    this.error= aux;//Guardamos el error de este clasificador
    this.fallos= fallos;
}
```

Obtención de un clasificador débil

Antes de obtener un clasificador débil hay que saber qué es necesario para que un clasificador sea considerado clasificador débil y pueda llegar a formar parte de un clasificador fuerte.

Un **clasificador débil** es aquél que mediante un número arbitrario de pruebas aleatorias un clasificador consigue alcanzar aproximadamente o superar un 50% de acierto en su fase de **entrenamiento**.

En este programa el número arbitrario de pruebas aleatorias es controlado por una variable global declarada como **A**. El resultado de un clasificador débil también depende del **porcentaje de imágenes usadas en la fase de entrenamiento**, así que para demostrar cómo va evolucionando el porcentaje de acierto a medida que aumentamos el valor de A, he realizado el siguiente gráfico:



Este gráfico y los datos utilizados se encuentran en el Excel: "MejorValorA.xlxs"

Para la realización de este gráfico se ha generado un total de 30 clasificadores débiles probando distintos valores de A junto a distintos porcentajes de imágenes.

Como se puede observar no hace falta un valor excesivamente alto para lograr que un clasificador sea considerado un clasificador débil, también hay que tener en cuenta que se trata de **medias**, por tanto, se podría decir que cuando **A=10** asegura que todos los clasificadores superen o igualen el 50% de acierto.

Además, es necesario indicar que todos los porcentajes de imágenes consiguen un porcentaje de acierto de forma proporcional, haciendo que el **mejor valor para A** sea **aproximadamente 10.000,** también es preciso recalcar que, a partir de cierto valor de A, este no mejora; como es el caso de, por ejemplo:

Siendo el porcentaje de imágenes 70%, cuando A vale 10.000 hemos conseguido una media de 56,74% de acierto y luego cuando A vale 15.000 consigue incluso un porcentaje **peor** que el obtenido con un valor menor, 56,52% de acierto.

ClasificadorFuerte.java

Esta clase se encarga de gestionar todo lo relacionado con el Clasificador Fuerte.

Un Clasificador Fuerte es en esencia un **conjunto de Clasificadores Débiles,** por tanto, para sus dos constructores será necesario pasar como parámetro un ArrayList del objeto ClasificadorDebil.

Además, he creado una serie de variables las cuales ayudarán a la hora de utilizar dicho Clasificador Fuerte. Un **array de int** en el que guardamos los resultados de aplicar el Clasificador Fuerte tal y como hicimos anteriormente en la clase ClasificadorDebil.java, un **contador de fallos**, el cual será de gran ayuda a la hora de obtener el porcentaje de acierto del Clasificador Fuerte y, por último, una variable llamada **confianza** la cual nos servirá para el apartado "Test", esta variable almacenará el **porcentaje de acierto** que ha obtenido por el clasificador a la hora de clasificar el conjunto de imágenes elegidas para test (Xtest).

Esta clase se basa en dos principales funciones: **aplicarClasifFuerte** y **obtenerErrorClasif.**

La función **obtenerErrorClasif** es prácticamente igual que la mencionada en el apartado 2, la única diferencia es que no es necesario guardar una variable error que sea un sumatorio del peso de las imágenes que ha fallado, simplemente aumentaremos el valor de la variable fallos por cada fallo.

En la función **aplicarClasifFuerte,** le pasaremos el conjunto de imágenes que queremos que clasifique, esta función funciona diferente a la del Clasificador Débil.

Para cada una de las imágenes tendremos que obtener un valor el cual llamaremos **H**. Esta H es un **sumatorio** el cual se rige por la siguiente fórmula:

$$H = \sum_{i=1}^{n} \alpha_i \cdot checkImagen_i(x)$$

Sea **n** el número de Clasificadores Débiles que forman un Clasificador Fuerte, entonces lo que tendremos en cada iteración es **multiplicar** la **confianza** de un ClasificadorDebil, i, por el **resultado** obtenido de aplicar la función checkImagen*, el cual será **1 o -1**.

Después de obtener el sumatorio de H nos tendremos que fijar en su símbolo.

- **Si es positivo** → Entonces la imagen pertenece a su conjunto, se almacenará 1 en el array prediccion.
- **Si es negativo** → Entonces la imagen NO pertenece a su conjunto, se almacenará -1 en el array prediccion.

```
public void aplicarClasifFuerte(ArrayList<Imagen> X) {
    this.resize(X.size());
    for(int i=0; i < X.size(); i++) {
        double H= 0.0;
        for(int j=0; j < this.debiles.size(); j++)  //Es -1 o +1
            H += this.debiles.get(j).getConfianza()*this.debiles.get(j).checkImagen(X.get(i));
        if(H > 0)
            this.prediccion[i]= 1;//Es un avión
        else
            this.prediccion[i]= -1;//No es avión
    }
}
```

Como prediccion no es dinámico, para solucionar el problema he creado la función **resize**, la cual básicamente modifica el tamaño de prediccion acorde al tamaño del conjunto de imágenes (test o train).

*__checkImagen__ es una función creada a partir de esta función, tal y como su nombre indica su función es devolver 1 o -1, para ello se sigue la misma lógica que explicamos en la función aplicarClasifDebil pero en vez de guardar el valor lo devuelve directamente.

Conjunto de Entrenamiento y Test

Después de la realización de múltiples pruebas he llegado a la conclusión que el porcentaje ideal sería un porcentaje entre 70%-80% de imágenes para Entrenamiento y un 30%-20% de imágenes para Test.

En la mayoría de las pruebas que he realizado los mejores valores se suelen obtener con dichos porcentajes de imágenes.

Es fundamental hacer dicha división de imágenes para comprobar el correcto funcionamiento del reconocimiento de imágenes, ya que si se utilizan únicamente las mismas imágenes con las que se entrena podríamos no estar obteniendo resultados reales.

Estos conjuntos se generan con la función **generaEntrenamiento** y **generaTest,** estas dos funciones son muy similares: La primera añade de un cierto **conjunto, num imágenes** el cual varía dependiendo del valor de P y el número de imágenes de la categoría además de añadir un número equivalente de las otras categorías para que sea un 50-50 a la hora de que el clasificador clasifique las imágenes. La segunda, funciona de la misma manera con la diferencia de que el for que se encarga de poner **num imágenes**, ahora añade a partir de num hasta el final con el objetivo de que las imágenes usadas en test sean **distintas** al entrenamiento.

```
public static void generaEntrenamiento(ArrayList<Imagen> X, int[] Y, int num, int conjunto) {
    CIFAR10Loader ml = new CIFAR10Loader();
    ml.loadDBFromPath("./cifar10_2000");
    int indice= 0;
    File carpeta = new File("./cifar10_2000/" + conjunto);
    File[] lista= carpeta.listFiles();//Obtenemos el número de elementos del conjunto
    for(int i=0; i < numClases; i++) {
        ArrayList dlimgs = ml.getImageDatabaseForDigit(i);
        if(i == conjunto) {
            for(int j= 0; j < num; j++) {
                Imagen img = (Imagen) dlimgs.get(j);
                X.add(img);
                Y[indice]= 1;
                indice++;
            }
        }
        else {
            for(int j= 0; j < (lista.length - num)/(numClases - 1); j++) {
                Imagen img = (Imagen) dlimgs.get(j);
                X.add(img);
                Y[indice]= -1;
                indice++;
            }
        }
    }
}
```

```
public static void generaTest(ArrayList<Imagen> X, int[] Y, int num, int conjunto) {
    CIFAR10Loader ml = new CIFAR10Loader();
    ml.loadDBFromPath("./cifar10_2000");
    int indice= 0;
    File carpeta = new File("./cifar10_2000/" + conjunto);
    File[] lista= carpeta.listFiles();//Obtenemos el número de elementos del conjunto
    for(int i=0; i < numClases; i++) {
        ArrayList dlimgs = ml.getImageDatabaseForDigit(i);
        if(i == conjunto) {
            for(int j= num; j < lista.length; j++) {
                Imagen img = (Imagen) dlimgs.get(j);
                X.add(img);
                Y[indice]= 1;
                indice++;
            }
        }
        else {
            for(int j= 0; j < (lista.length - num)/(numClases - 1); j++) {
                Imagen img = (Imagen) dlimgs.get(j);
                X.add(img);
                Y[indice]= -1;
                indice++;
            }
        }
    }
}
```

Importancia del número de Clasificadores Débiles (T).

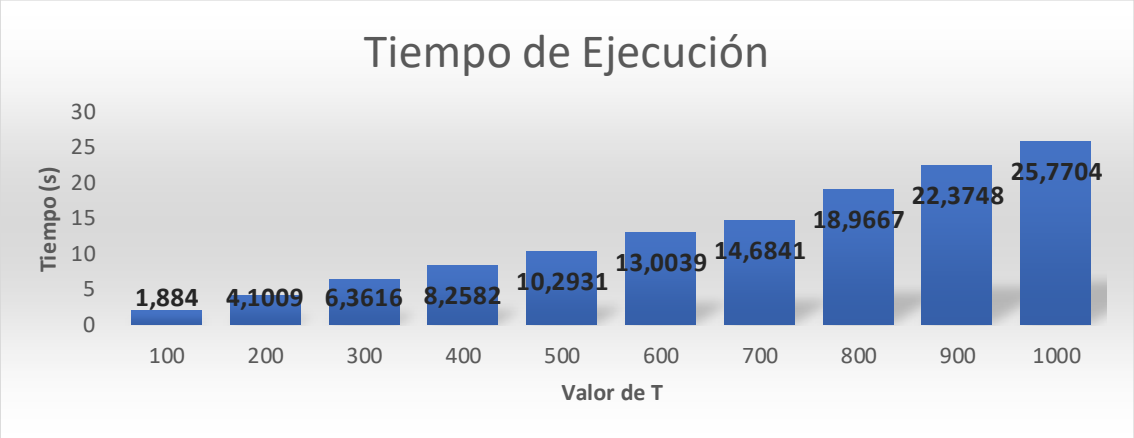
Una parte importante del algoritmo es el **número de clasificadores (T)** que se usarán en el proceso de aprendizaje y los cuales pasarán a formar parte del Clasificador Fuerte que genera el algoritmo.

Para ello haremos uso de las variables **start** y **end** para obtener cuánto tiempo tarda en segundos a medida que aumentamos T.

```
long start= System.currentTimeMillis();

ClasificadorFuerte clasificador= Adaboost(X, Y);

long end= System.currentTimeMillis();

double time = (double) ((end - start)/1000.0);
DecimalFormat df= new DecimalFormat("#.0000");
System.out.println(df.format(time));
```

Hemos realizado una prueba en la que se seleccionaban el **50% de las imágenes de cada conjunto de imágenes** para el entrenamiento, además de generar **10.000 pruebas aleatorias** T veces.

Después de ejecutar el programa con un cierto valor de T un total de 10 veces (El número de clasificadores fuertes que se pide en la práctica), hemos obtenido el tiempo promedio que tarda en generar un clasificador fuerte el cual está representado en la siguiente gráfica:



Este gráfico y los datos utilizados se encuentran en el Excel: "TiempoEjecución.xlxs"

Como podemos observar, a medida que aumentamos el valor de T, también lo hace el tiempo que tarda debido a que es T la variable que controla el bucle principal que se realiza en Adaboost.

El valor de T es el que más peso tiene en el programa, por encima del número de pruebas aleatorias y el porcentaje de imágenes que se vayan a utilizar para entrenamiento o para test. Además, también existe la posibilidad de que se genere **sobre entrenamiento**, por tanto, es preferible tener bien medido cuál es el perfecto valor que debe tomar T para obtener tanto como una respuesta rápida como para obtener un buen clasificador fuerte.

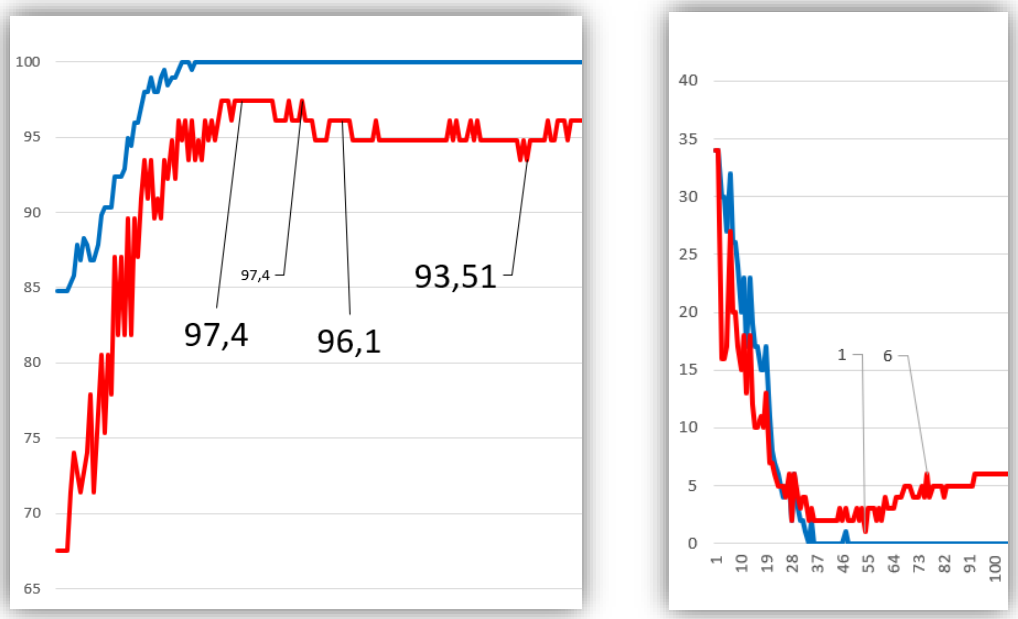
Sobre entrenamiento

Uno de los principales problemas que tiene Adaboost es el conocido como **sobre entrenamiento**, el cual ocurre cuando se realiza un **número excesivo de pruebas** sobre el Clasificador Fuerte.

Para demostrar que ocurre sobre entrenamiento, al final de cada iteración de Adaboost hemos creado un Clasificador Fuerte con los Clasificadores Débiles obtenidos hasta el momento y hemos ido almacenando dichos datos en una tabla.

Obteniendo la siguiente gráfica: **Color azul**, simbolizando el porcentaje de acierto y fallos obtenidos al aplicar el Clasificador Fuerte sobre el conjunto de **entrenamiento** y **color rojo**, simbolizando el porcentaje de acierto y fallos obtenidos al aplicar el Clasificador Fuerte sobre el conjunto de test.

Porcentaje de acierto **Número de fallos**



El resto de la gráfica y sus datos están en el Excel: DemostracionSobreEntrenamiento.xlxs

Como se puede observar al alcanzar un cierto número de Clasificadores Débiles generados Adaboost empieza a devolver Clasificadores Fuertes con un peor porcentaje de acierto y un mayor número de fallos sobre el conjunto de test.

Esto es debido al **Overfitting**, es decir, el Clasificador Fuerte en vez de aprender a reconocer imágenes lo que está haciendo es memorizar imágenes, de hecho, como se puede ver, en la parte de Entrenamiento está obteniendo un **100% de acierto** o un total de **0 fallos**, lo cual **nunca** debería ocurrir y es otro motivo por el cual podemos identificar que está ocurriendo sobre entrenamiento.

Por tanto, y para evitar este problema, hemos añadido esta parte al final de código de Adaboost, el cual termina el algoritmo a partir de un cierto número de Clasificadores Débiles generados comparando si el porcentaje de acierto baja, la cual se comentará en el apartado **Funcionamiento de Adaboost**.

Funcionamiento de Adaboost

Adaboost es un algoritmo que devuelve un Clasificador Fuerte el cual ha sido generado a base de Clasificadores Débiles.

```
public static ClasificadorFuerte Adaboost(ArrayList<Imagen> X, int[] Y, ArrayList<Imagen> Xtest, int[] Ytest) {
    ArrayList<ClasificadorDebil> ganadores= new ArrayList<>();
    ClasificadorFuerte best= new ClasificadorFuerte(ganadores);
    boolean first= true;
    double[] D= new double[X.size()];
    for(int i=0; i < X.size(); i++)
        D[i]= (double)1/X.size();
    for(int i=0; i < T; i++) {
        //Comienza entrenamiento
        ClasificadorDebil ganador= Entrenamiento(X, Y, D);
        double confianza= 0.5*Math.log((1 - ganador.getError())/ganador.getError())/Math.log(2);
        ganador.setConfianza(confianza);
        //Actualizar los pesos
        double Z= 0;
        double[] numeradores= new double[X.size()];
        for(int j=0; j < X.size(); j++) {
            numeradores[j]= D[j]*Math.pow(Math.E, (-1*ganador.getConfianza())*Y[j]*ganador.getPrediccion()[j]);
            Z+= numeradores[j];//Sumatorio de los numeradores/Constante de normalización
        }
        for(int j=0; j < X.size(); j++)
            D[j]= numeradores[j]/Z;
        ganadores.add(ganador);
        if(i >= 40) {
            ClasificadorFuerte aux= new ClasificadorFuerte(ganadores);
            aux.aplicarClasifFuerte(Xtest);
            aux.obtenerErrorClasif(Xtest, Ytest);
            double porcentajeTest= (((double)Xtest.size() - (double)aux.getFallos())/(double)Xtest.size()) *100;
            aux.setConfianza(porcentajeTest);
            if(first == true) {
                best= aux;
                first= false;
            }
            else if(aux.getConfianza() < best.getConfianza())
                    return best;//Devolvemos el actual antes de que empeore por el sobre entrenamiento.
        }

    }
    return new ClasificadorFuerte(ganadores);
}
```

Lo primero que realiza el algoritmo es establecer los pesos originales de las imágenes antes de empezar el bucle principal, siendo el **peso original** de una imagen **1 / Total de imágenes.**

Una vez establecido los pesos el programa entrenará T Clasificadores Débiles con los cuales formará un único Clasificador Fuerte el cual se especializará en reconocer una cierta categoría de imágenes.

Por cada Clasificador Débil será necesario calcular su confianza, la cual viene dada por la siguiente fórmula:

$$\textbf{confianza} = \frac{1}{2} \cdot \log_2\left(\frac{1 - \textbf{error}}{\textbf{error}}\right)$$

El **error** es el error que ha obtenido el Clasificador Débil **ganador,** que ha obtenido en su fase de entrenamiento. Dicha fase se encuentra en la función Entrenamiento, la cual genera **A** Clasificadores Débiles y finalmente se elige aquél clasificador que haya obtenido el error más pequeño y por tanto el mejor de todos los generados.

```
public static ClasificadorDebil Entrenamiento(ArrayList<Imagen> X, int[] Y, double[] D) {

    ArrayList<ClasificadorDebil> vector= new ArrayList<>();
    ClasificadorDebil ganador= null;
    int j=0;
    for(int i=0; i < A; i++) {
        ClasificadorDebil actual= ClasificadorDebil.generarClasifAzar(M, X.size());
        actual.aplicarClasifDebil(X);//Obtenemos su predicción
        actual.obtenerErrorClasif(X, Y, D);//Obtenemos el error
        vector.add(actual);//Guardamos el actual en el vector
    }
    double min= 0.0;
    for(int i=0; i < vector.size(); i++) {
        if(i == 0) {
            min= vector.get(i).getError();
            ganador= vector.get(i);
        }
        else if(vector.get(i).getError() < min) {
                min= vector.get(i).getError();
                ganador= vector.get(i);
        }
    }
    return ganador;
}
```

El siguiente paso, aparte de ya poder añadir el Clasificador Débil, ganador al ArrayList de Clasificadores Débiles, ganadores; ahora tendremos que actualizar los pesos de las imágenes, para ello necesitaremos dos bucles con los cuales recorrer todos los valores almacenados en D.

```
double Z= 0;
double[] numeradores= new double[X.size()];
for(int j=0; j < X.size(); j++) {
    numeradores[j]= D[j]*Math.pow(Math.E, (-1*ganador.getConfianza())*Y[j]*ganador.getPrediccion()[j]);
    Z+= numeradores[j];//Sumatorio de los numeradores/Constante de normalización
}
for(int j=0; j < X.size(); j++)
    D[j]= numeradores[j]/Z;
ganadores.add(ganador);
```

En el primer bucle lo utilizaremos para rellenar un array de double auxiliar que hemos creado el cual almacenará los numeradores de la fórmula que utilizaremos para recalcular los pesos además de obtener la variable **Z** (constante de normalización), la cual es un sumatorio de los numeradores creados.

Finalmente, aplicaremos la siguiente fórmula a todos los pesos de la imagen:

$$D_j = \frac{D_j \cdot e^{-k \cdot y \cdot h(x_j)}}{Z}$$

Siendo **k**, la confianza del mejor Clasificador Débil obtenido de Entrenamiento; **y** es la solución de la imagen j (vale 1 o -1) y **h(x_j)** es la clasificación que ha asignado el Clasificador Débil a la imagen j.

Por último, añadiremos una última condición con la cual podremos evitar el sobre entrenamiento, básicamente a partir de un cierto número de iteraciones, dicho número dependerá del valor de A y el porcentaje de imágenes asignados, con el cual iremos comparando un porcentaje de acierto obtenido de un Clasificador Fuerte con una cierta cantidad de Clasificadores Débiles con la siguiente iteración la cual tendrá más clasificadores, pero no por tanto un mejor resultado.

En caso de obtener un peor resultado detendremos el algoritmo de Adaboost y devolveremos el Clasificador Fuerte obtenido en la iteración anterior, siendo este el mejor resultado posible ya que si sigue iterando obtendremos overfitting y sobre entrenamiento.

```
if(i >= 38) {
    ClasificadorFuerte aux= new ClasificadorFuerte(ganadores);
    aux.aplicarClasifFuerte(Xtest);
    aux.obtenerErrorClasif(Xtest, Ytest);
    double porcentajeTest= (((double)Xtest.size() - (double)aux.getFallos())/(double)Xtest.size()) *100;
    aux.setConfianza(porcentajeTest);
    if(first == true) {
        best= aux;
        first= false;
    }
    else if(aux.getConfianza() < best.getConfianza())
            return best;//Devolvemos el actual antes de que empeore por el sobre entrenamiento.
}
```

El valor 38 lo he elegido, debido a que como se puede ver en el Excel FuncionamientoAdaboost.xlsx, la mayoría de los clasificadores Fuertes que utilizamos para cada categoría llegan al 98-99% de acierto en la fase de entrenamiento antes del valor 40, por tanto, para evitar el Overfitting y sobre entrenamiento utilizaremos ese valor.

Ficheros

En esta práctica también se exige que se almacenen los Clasificadores Fuertes generados en un fichero para luego utilizarlos en el apartado Test.

Para ello he creado las funciones **generaFichero** y **leerFichero**, empezaremos explicando esta primera.

```
public static void generaFichero(ArrayList<ClasificadorFuerte> clasificadores, String nombre) throws IOException {
    FileWriter fichero= new FileWriter(nombre);
    PrintWriter pw= new PrintWriter(fichero);
    String aux= "";

    for(int i=0; i < clasificadores.size(); i++) {//tantas líneas como clasificadores
        for(int j=0; j < clasificadores.get(i).getDebiles().size(); j++)//Guardamos sus clasificadores débiles
            aux += clasificadores.get(i).getDebiles().get(j).toString() + "|";
        aux+= ":" + clasificadores.get(i).getConfianza();
        pw.println(aux);
        aux= "";//limpiamos el buffer
    }
    if(fichero != null)
        fichero.close();//Al cerrar se escribe todo
}
```

En esta función básicamente hacemos uso de las clases FileWriter y PrintWriter de Java con las cuales podremos crear y escribir el fichero en el cual guardaremos todos los Clasificadores Débiles que forman un Clasificador Fuerte, además de añadir al final la confianza de este mismo.

El fichero tendrá tantas líneas como número de Clasificadores Fuertes haya generado nuestro programa, cada línea tendrá el siguiente formato:

pixel,umbral,direccion,confianza|pixel,umbral,direccion,confianza|…|:confianzaClasifFuerte

Como se puede observar, los Clasificadores Débiles están separados por el símbolo "|", además hemos tenido que crear un ToString() en la clase para facilitar la escritura del código.

Como detalle, el **String nombre** es un nombre el cuál ha elegido el usuario y automáticamente se guarda en la raíz del proyecto.

Respecto a leerFichero, simplemente utilizaremos las clases de FileReader y BufferedReader de Java con las cuales podremos abrir y leer el fichero con el formato que ya hemos mencionado, para almacenar los valores del fichero, haciendo uso de la función **charAt()** y un array auxiliar, como vemos hay un bucle j usaremos un switch para asignar los valores en su orden correspondiente.

```
public static void leerFichero(ArrayList<ClasificadorFuerte> clasificadores, String nombre) throws FileNotFoundException, IOException{
    File fichero= new File("./" + nombre);
    FileReader fr= new FileReader(fichero);
    BufferedReader br= new BufferedReader(fr);
    String linea;
    while((linea=br.readLine()) != null){
        ArrayList<ClasificadorDebil> debiles= new ArrayList<>();
        int i=0;
        String confianza= "";
        for(;i < linea.length();) {//Aumentaremos i a medida que leamos caracteres
            ClasificadorDebil debil= new ClasificadorDebil();
            for(int j=0; j < 4; j++) {
                String aux= "";
                while(linea.charAt(i) != ',' && linea.charAt(i) != '|') {
                    aux += linea.charAt(i);
                    i++;
                }
                i++;
                if(linea.charAt(i) == ':') {
                    i++;
                    while(i < linea.length()) {
                        confianza+= linea.charAt(i);
                        i++;
                    }
                }
```

Para pasar de string a int o double utilizaremos las funciones: **Integer.parseInt(string)** y **Double.parseDouble(string)**

Adaboost binario adaptado a 10 tipos de clases

Es cierto que Adaboost está limitado a devolver un Clasificador Fuerte el cual sólo puede indicar si la imagen que está clasificando pertenece o no pertenece (1 o -1) al conjunto para el cual ha sido asignado.

Para solucionar este problema, he decido crear un Clasificador Fuerte por cada categoría que tengamos en nuestra base de datos (CIFAR10_2000), es decir que aplicamos Adaboost 10 veces para luego guardar los **10 Clasificadores Fuertes** en un ArrayList, siendo el primer Clasificador Fuerte el encargado de reconocer los aviones (0), el segundo el encargado de reconocer automóviles (1), …

Es luego en el apartado Test en el cual utilizaremos los 10 Clasificadores Fuertes a la vez haciendo uso de la función auxiliar **clasificarImagen** la cual necesitará como parámetros el ArrayList ya mencionado, así como la ruta de la imagen la cual va a ser clasificada.

```
public static int clasificarImagen(ArrayList<ClasificadorFuerte> clasificadores, String path){
    File ruta= new File(path);
    Imagen img= new Imagen(ruta);
    int aciertos = 0;
    double[] prediccion= new double[numClases];
    for(int i=0; i < clasificadores.size(); i++) {
        prediccion[i]= clasificadores.get(i).checkImagen(img);
        if(prediccion[i] > 0.0)
            aciertos++;
    }
```

En esta parte del código simplemente aplicaremos todos los clasificadores haciendo uso de una nueva función **checkImagen**, la cual se encarga de clasificar una sola imagen y devuelve el valor de **H**, anteriormente explicado su funcionamiento en el apartado de **ClasificadorFuerte.java**, dependiendo si considera que la imagen pertenece o no a la categoría que reconoce el Clasificador Fuerte, entonces H valdrá positivo o negativo, cuando sea positivo aumentaremos el contador de aciertos.

Gracias al contador de aciertos que hemos utilizado podremos diferenciar los 3 posibles casos que nos podemos encontrar:

- Un clasificador devuelve 1 y el resto devuelve -1.
- Todos devuelven -1.
- Varios clasificadores devuelven 1 y hay que elegir una única opción.

Para resolver los casos más conflictivos será necesario mirar la **confianza** de cada Clasificador Fuerte, dicha confianza es el porcentaje de acierto que ha obtenido el Clasificador Fuerte al clasificar el conjunto de imágenes seleccionadas para test.

```
switch (aciertos) {
    case 1:
        //Caso ideal
        for(int i=0; i<prediccion.length; i++)
            if(prediccion[i] == 1)
                return i;
        break;
    case 0:
        //Peor caso
        double min= clasificadores.get(0).getConfianza();
        int result= 0;
        for(int i=1; i<clasificadores.size(); i++) {
            if(clasificadores.get(i).getConfianza() < min) {//peor porcentaje
                min= clasificadores.get(i).getConfianza();
                result= i;
            }
        }
        return result;
```

Cuando **aciertos es igual a 1** simplemente recorremos el array prediccion y devolveremos la posición donde se encuentre el 1, lo cual coincide con la categoría de la imagen clasificada.

Cuando **aciertos vale 0**, deberemos recorrer todos los Clasificadores Fuertes en busca de aquel que tenga la **MENOR CONFIANZA**, debido a que, a mayor confianza, más seguro está el clasificador que esa imagen no pertenece a esa categoría, por tanto, el clasificador con menor confianza seguramente se haya equivocado a la hora de decir que esa imagen no pertenece a la categoría que clasifica.

```
default:
    //Indecisión
    int primero= 0;
    double maximo= 0.0;
    int resultado= 0;
    for(int i=0; i<prediccion.length; i++) {
        if(primero == 0) {//Guardamos el primer max
            if(prediccion[i] > 0.0){
                maximo= prediccion[i];
                resultado= i;
                primero++;
            }
        }
        else {
            if(prediccion[i] > 0.0 && prediccion[i] > maximo){
                    maximo= prediccion[i];
                    resultado= i;
                }
            }
        }
    return resultado;
```

Cuando **aciertos vale más de 1**, deberemos recorrer todos los Clasificadores Fuertes que han devuelto 1 en busca de aquel que tenga la **MAYOR CONFIANZA**, debido a que, a mayor confianza, más seguro está el clasificador que esa imagen pertenece a esa categoría, por tanto, el clasificador con mayor confianza seguramente sea el que menos probabilidad a tenido de equivocarse.