

Documentación Adaboost

15.06.2020

- Adaboost.
- Implementación.
- Estructura del programa.
 - Main
 - Adaboost
 - Clasificador Fuerte
 - Clasificador Débil
- Consola.

Adaboost.

El algoritmo que vamos a implementar en esta práctica es el algoritmo Adaboost. Este algoritmo nos proporciona un nuevo clasificador 'robusto' a partir de un conjunto de clasificadores 'débiles'. Dicho de otra manera consigue un clasificador eficiente a partir de muchos que no son nada eficientes.

¿Como funciona? La salida del algoritmo es el resultado de aplicar cada uno de los clasificadores a un vector con valores que pueden variar entre -1 y 1. Esto se llama cálculo de coeficientes de ponderación, y se realiza de manera progresiva. Hay que primero entrenar los clasificadores y después testarlos. Durante este proceso de entrenamiento y testeo se asignan penalizaciones, asignándoles a cada uno un valor de error y un valor de confianza. El cálculo del valor de error también nos ayuda a calcular los pesos de los elementos del vector. Más adelante enseñaré como lo he implementado.

Implementación.

Vamos a implementar el algoritmo Adaboost para nuestra práctica, que consiste en conseguir que el programa sea capaz de decir qué número está 'leyendo'.

La entrada al programa viene de la base de datos MNIST Loader, que nos va a proporcionar clasificados en carpetas una serie de imágenes que contienen números 'escritos a mano'. Una vez cargados estos datos, se procede a separar en elementos que van a servir para entrenar, para testear y para comprobar cuantos aciertos tienen nuestros clasificadores.

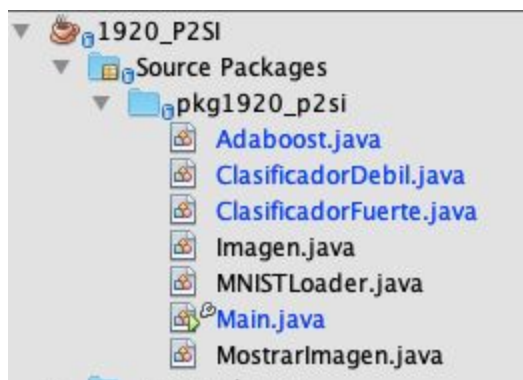
Entonces vamos a generar, para cada dígito, un clasificador robusto que contendrá tantos clasificadores robustos como queramos. Y les haremos pruebas para saber cuales son los que tenemos que tener en cuenta. Para conseguir mejores clasificadores tendremos que hacer pruebas con el porcentaje de elementos que queremos entrenar y con el número de elementos que queremos hacer pruebas, y cuantos clasificadores (débiles) queremos, cuantos más tengamos más acertada será la predicción, pero no queremos tener más de los necesarios, para no cargar demasiado el programa.

Una vez obtengamos nuestro clasificador fuerte, para poder usarlo en otra ocasión sin necesidad de generarlo de nuevo, lo guardaremos en un archivo '.txt'. Se dentro de la carpeta del programa, con el nombre 'clasificadorFuerte.txt'.

Por lo tanto hay que destacar que el programa tendrá dos opciones de funcionamiento:

1. Argumento "-t" : Con este argumento accedemos a la primera parte del programa, donde se crea el clasificador fuerte y se guarda.
2. Argumento "imagen": En esta parte del programa ya tenemos nuestro clasificadorFuerte hecho, solo tenemos que leerlo del fichero de texto y cargarlo en el programa. A continuación lo utilizamos para decir que dígito estamos viendo en la imagen pasada como parámetro.

Estructura del Programa.



A parte de la plantilla que ya se nos proporcionaba, he añadido 3 clases :

- Adaboost.java
- Clasificador Fuerte.java
- Clasificador Débil.java

Voy a explicar las partes del código que son más importantes para demostrar como he planteado esta práctica. La implementación de Adaboost para clasificar dígitos.

1. Main

```
public class Main{

    public static int NUM_PORCENTAJE = 50;
    public static int NUM_CLASIFICADORES = 10;
    public static int NUM_PRUEBAS = 10;

    public static int ByteUnsigned(byte b) {...3 lines }

    public static void guardarClasificador(ArrayList<ClasificadorFuerte> fuertes, String fileName) throws IOException {...50 lines }

    public static ArrayList<ClasificadorFuerte> leerClasificadorFuerte(String fileName) throws IOException {...60 lines }

    public static void main(String[] args) throws IOException {...81 lines }
```

Las variables estáticas están en Main.java y nos van a servir para variables que vamos a utilizar en otras partes del programa, que además es fundamental que las calibremos para conseguir mejores resultados.

NUM_PORCENTAJE : Decide el porcentaje de los elementos de entrada se dedicarán a entrenamiento. Por ejemplo si es igual a 50, significa que la mitad de datos de entrada sirven para entrenar y la otra mitad sirve para testear.

NUM_CLASIFICADORES : La cantidad de clasificadores débiles que se va a generar por clasificador fuerte, para cada dígito.

NUM_PRUEBAS : Pruebas para conseguir datos como la ponderación de pesos, el error y la confianza de cada clasificador.

Después tenemos dos funciones que con el nombre nos indican que hacen “guardarClasificador” y “leerClasificadorFuerte”.

Dentro de main vamos a ver partes del código importantes para el desarrollo.

```

Adaboost adaboost = new Adaboost(NUM_PORCENTAJE, NUM_CLASIFICADORES, NUM_PRUEBAS);

if( args[0].equals("-t") ){
    System.out.println("\n\n 3. ENTRENADO ARGUMENTO : '-t' \n");

    init = System.currentTimeMillis();
    ClasificadoresFuentes = adaboost.getClasificadoresFuentes();
    guardarClasificador(ClasificadoresFuentes, "clasificadorFuerte.txt");
    numAciertos = adaboost.aplicarClasificadorFuerte( adaboost.getTest(), ClasificadoresFuentes);
}

```

Esta es la opción de entrenamiento. Las 4 funciones que harán el trabajos son:

“Adaboost” : La primera es la construcción del objeto Adaboost, con los 3 valores de entrada, que comentamos anteriormente. De esta manera podremos acceder a sus funcionalidades y utilizarlas.

“adaboost.getClasificadorFuerte” : Nos interesa buscar el clasificador más fuerte que se pueda encontrar dados los valores de entrada anteriores. Y cargando los datos de la base de datos.(veremos más adelante dentro de esta función).

“guardarClasificador” : Ya solo tenemos que guardar el clasificador que la línea anterior de código nos ha proporcionado.

“aplicarClasificadorFuerte” : Ponemos en funcionamiento nuestro clasificador, junto con una colección de elementos que nos sirvan para testear nuestro recién hecho clasificador. Devuelve cuantas veces hemos acertado, este dato nos servirá para saber qué clasificador es mejor o peor. Y de esta manera cambiar los valores estáticos del Main.java.

Algo que no es tan importante, pero me parece una buena herramienta es “System.currentTimeMillis()” que nos ayuda a calcular el tiempo en que tarda nuestro algoritmo en procesar estas funciones.

```

}else{
    |
    int digitoEvaluar = 1;
    ClasificadoresFuentes = leerClasificadorFuerte("clasificadores.txt");
    imagenes = adaboost.ml.getImageDatabaseForDigit(digitoEvaluar);
    numAciertos = adaboost.aplicarClasificadorFuerte( imagenes, ClasificadoresFuentes);
}

```

La siguiente parte del programa es fácil de explicar. Primero se carga el clasificador del fichero de texto, se carga la imagen que se desea evaluar y por último se aplica este clasificador y se guarda si ha acertado o no.

Adaboost.

Empezamos mostrando sus propiedades. En la inicialización las tres propiedades de tipo int se inicializan con el valor que tengan los métodos estáticos.

```
package pkg1920_p2si;
import java.util.*;
import java.io.*;
/**
 *
 * @author stalynalejandro
 */
public class Adaboost {

    private int porcentaje;
    private int debiles;
    private int pruebas;

    private ArrayList test = new ArrayList();
    private ArrayList<ArrayList> entrenamiento = new ArrayList();
    private ArrayList<ArrayList> correcto = new ArrayList();

    //Array auxiliar
    private ArrayList tipo = new ArrayList();

    public MNISTLoader ml = new MNISTLoader();
```

Los ArrayList es donde se van a clasificar los datos de entrada.

Entrenamiento y correcto son bidimensionales por que cada dígito tiene sus propias colecciones de elementos (imágenes).

Tenemos el objeto MNISTLoader para cargar los datos desde esta clase.

Vamos a ver las funciones de esta clase.

```
//Constructor
Adaboost(int porcentaje, int debiles, int pruebas){...14 lines }

private void clasificarImagenes(){...87 lines }

public ArrayList<ClasificadorFuerte> getClasificadoresFuentes(){...17 lines }

public int aplicarClasificadorFuerte( ArrayList test, ArrayList<ClasificadorFuerte> fuertes ){...47 lines }

public ClasificadorFuerte aplicarAdaboost( ArrayList<Imagen> entrenamiento, ArrayList correctos, int digito){...100 lines }

public ArrayList getEntrenamiento(){
    return this.entrenamiento;
}

public ArrayList getCorrecto(){
    return this.correcto;
}

public ArrayList getTest(){
    return this.test;
}
```

El constructor inicializa sus valores, con sus argumentos de entrada pero también llama a la función clasificar imágenes para que las añada a los arrayList correspondientes para su uso en el futuro.

En la función “aplicarAdaboost” es donde implementamos el algoritmo que se nos da en el enunciado de la práctica, desde esta función llamamos a todas las funciones necesarias de la clase “ClasificadorFuerte.java” que necesitamos.

Clasificador Fuerte.

Son las propiedades del clasificador fuerte, donde el error es la suma de todas las imágenes donde el clasificador falla. Y cuanto menor sea el error, mayor será la confianza.

La función “getRandomBetween” nos servirá para generar números aleatorios dentro de un rango. Los argumentos que se pasan son el inicio y final del rango que se da. Otro dato importante es que el tamaño del ArrayList de clasificadores débiles viene determinado por la variable estática NUM_CLASIFICADORES.

```
package pkg1920_p2si;
import java.util.*;

/**
 * @author stalynalejandro
 */
public class ClasificadorFuerte {

    public float error;        //Suma de pesos en todas las imágenes
    public float confianza;    //Cuanto menor sea el error, mayor será

    private ArrayList<ClasificadorDebil> clasificadores;

    //Util
    private int getRandomBetween(int min, int max){
        return ((int) (Math.random() * ((max - min) + 1)) + min);
    }

    //Init
    public ClasificadorFuerte(){
        clasificadores = new ArrayList();
        this.confianza = 0.0f;
        this.error = 0.0f;
    }
}
```

Entre otras funciones dentro de esta clase, aquí están las más importantes que vienen explicadas en el enunciado de la práctica. Vemos como se utiliza la función de antes “getRandomBetween” para generar los clasificadores débiles.

```
//Genera un clasificador débil dada una dimensión.
public ClasificadorDebil generarClasificadorAzar(int DIM){

    int pixel      = getRandomBetween(0, DIM);
    int umbral     = getRandomBetween(0, 255);
    int direccion  = getRandomBetween(0, 1);

    return new ClasificadorDebil(pixel, umbral, direccion, 0, 0);
}

public ArrayList aplicarClasificadorDebil( ClasificadorDebil debil, ArrayList<Imagen> entrenamiento ){

    ArrayList<Boolean> resultado = new ArrayList();

    for( int i = 0; i < entrenamiento.size(); i++ ){
        Imagen imagen = (Imagen) entrenamiento.get(i);
        resultado.add(hipotesis(debil, imagen));
    }

    return resultado;
}

//Obtiene el error del clasificador débil
public float ObtenerErrorClasificadorDebil(ClasificadorDebil debil, ArrayList<Imagen> entrenamiento, ArrayList<Integer> correcto){

    float error = 0.0f;

    ArrayList resultadoDebil = this.aplicarClasificadorDebil(debil, entrenamiento);

    for( int i = 0; i < entrenamiento.size(); i++ ){

        if(resultadoDebil.get(i).equals(correcto.get(i))){

            error += entrenamiento.get(i).getPeso();

        }

    }

}
```

Clasificador Débil.

Las propiedades del clasificador débil. La idea detrás de generar clasificadores débiles es simplemente intentar generar unos clasificadores que sean un poco mejor que directamente clasificar de manera aleatoria. Por eso en esta clase no hay demasiado código que explicar pero veremos las funciones de las que más se hace uso aquí.

```

/**
 *
 * @author stalynalejandro
 */
public class ClasificadorDebil {

    private int pixel;
    private int umbral;           //[0..255] => [blanco..negro]
    private int direccion;        //Nos sirve para clasificar el pixel.

    private float error;
    private float confianza;

    public ClasificadorDebil(){
        this.pixel      = 0;
        this.umbral      = 0;
        this.direccion   = 0;
        this.error       = 0;
        this.confianza   = 0.0f;
    }

    public ClasificadorDebil(int pixel, int umbral, int direccion, int error, float confianza){
        this.pixel      = pixel;
        this.umbral      = umbral;
        this.direccion   = direccion;
        this.error       = error;
        this.confianza   = confianza;
    }
}

```

Esta función nos sirve para establecer el error y la confianza que cada Clasificador Débil tiene, para después procesar esta información, en el algoritmo Adaboost.

```

,
public void setError( float error ){

    this.error = error;

    this.setConfianza( 0.5f * (float)Math.log10(1.0f - this.error) / this.error);

}

```

Una función muy importante es la hipótesis que se crea, es la que permite que sea un poco mejor que clasificar aleatoriamente es la función “hipótesis”. Y de esta manera utiliza el umbral para esta clasificación.

```

,
public boolean hipotesis( Imagen imagen ){

    boolean boo = false;

    if( this.direccion == 1 ){

        if( this.umbral < imagen.getImageData()[this.pixel] ){

            boo = true;

        }

    }

    else{

        if( this.umbral >= imagen.getImageData()[this.pixel] ){

            boo = true;

        }

    }

    return boo;

}

```


Consola.

Vamos a ver una prueba. En la siguiente imagen se ve lo que los pasos que suceden cuando se ejecuta el programa. En esta caso vamos a estar entrenando un clasificador para solo un dígito el 0. En la práctica esta por defecto entrenar a los 10 dígitos [0..9].

```
.....ADABOOST.....
```

```
Porcentaje de entrenamiento : 50
Número de clasificadores    : 10
Número de pruebas          : 10
```

1. CARGANDO DÍGITOS DE LA BASE DE DATOS...

```
Loaded digit 0
Loaded digit 1
Loaded digit 2
Loaded digit 3
Loaded digit 4
Loaded digit 5
Loaded digit 6
Loaded digit 7
Loaded digit 8
Loaded digit 9
Loaded 1000 images...
```

2. CLASIFICANDO IMÁGENES...

```
Array Test : 503
Array Entrenamiento : 497
Array Correcto (Dígitos) : 10
    Correcto (0) : 48
    Correcto (1) : 58
    Correcto (2) : 49
    Correcto (3) : 46
    .....
```

3. ENTRENADO ARGUMENTO : '-t'

```
Entrenando para dígito 0...
Peso Inicial : 0.020833334
```

```
Digito : 0
Entrenamiento: 48
Clasificados : 48 : [false, false, false, false, false, false,
Correcto : 48 : [true, true, true, true, true, true, true,
```

```
Digito : 0
Entrenamiento: 48
Clasificados : 48 : [true, true, true, true, true, true, true,
Correcto : 48 : [true, true, true, true, true, true, true,
```

```
Digito : 0
Entrenamiento: 48
Clasificados : 48 : [false, false, false, false, false, false,
Correcto : 48 : [true, true, true, true, true, true, true,
```

```
Digito : 0
Entrenamiento: 48
Clasificados : 48 : [false, false, false, false, false, false,
Correcto : 48 : [true, true, true, true, true, true, true,
```

```
Digito : 0
Entrenamiento: 48
Clasificados : 48 : [true, false, true, true, true, true, true,
Correcto : 48 : [true, true, true, true, true, true, true,
```

```
Digito : 0
Entrenamiento: 48
Clasificados : 48 : [true, true, true, true, true, true, true,
Correcto : 48 : [true, true, true, true, true, true, true,
```

```
Número de Clasificadores Fuertes Generados : 1
```

```
mejor : 502
```

```
Clasificadores Fuertes guardados en 'clasificadorFuerte.txt'
```

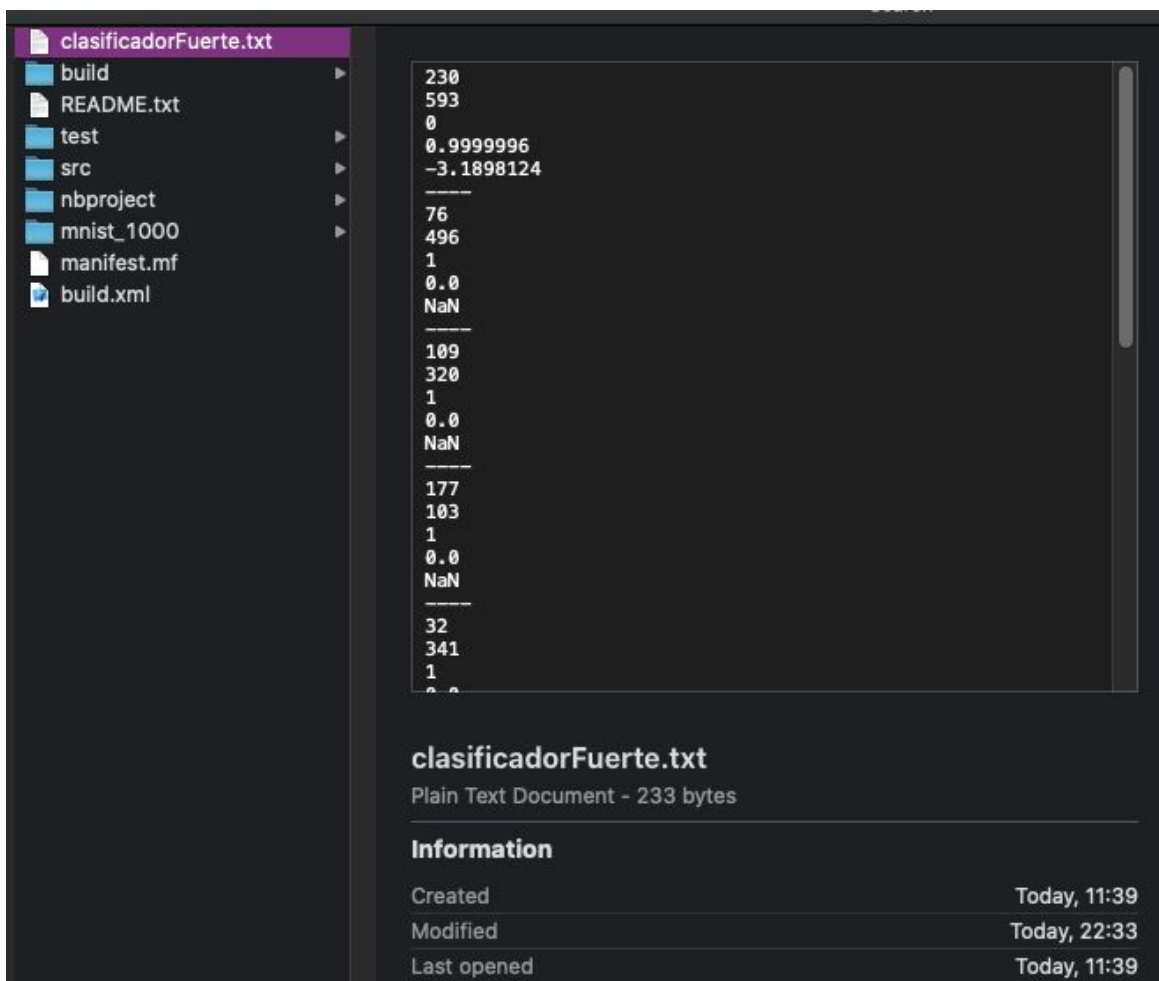
```
** Resultados imagenes de testeo:
```

```
Aciertos: 1
Fallos: 502
```

```
La tarea ha requerido 32 ms
```

```
BUILD SUCCESSFUL (total time: 3 seconds)
```


Y esto es lo que se ha guardado en 'clasificadorFuerte.txt'



The screenshot shows a code editor with a dark theme. On the left, a file explorer sidebar lists the following files and folders: 'clasificadorFuerte.txt' (selected), 'build', 'README.txt', 'test', 'src', 'nbproject', 'mnist_1000', 'manifest.mf', and 'build.xml'. The main editor area displays the content of 'clasificadorFuerte.txt', which consists of several lines of text, some of which are separated by horizontal lines. The text includes numerical values, floating-point numbers, and 'NaN'.

```
230
593
0
0.9999996
-3.1898124
-----
76
496
1
0.0
NaN
-----
109
320
1
0.0
NaN
-----
177
103
1
0.0
NaN
-----
32
341
1
0.0
```

Below the editor area, the file name 'clasificadorFuerte.txt' is displayed, followed by the description 'Plain Text Document - 233 bytes'. An 'Information' section at the bottom provides metadata:

| Information | |
|-------------|--------------|
| Created | Today, 11:39 |
| Modified | Today, 22:33 |
| Last opened | Today, 11:39 |