Sistemas Inteligentes

Práctica 2 - Visión Artificial y Aprendizaje

Nombre: Francisco Javier Rico Pérez

Grupo: Martes - 15:00 / 17:00

Correo: fjrp8@alu.ua.es

23 de Diciembre de 2018

Índice

1. Introducción.

La práctica consiste en desarrollar un sistema capaz de distinguir entre distintas imágenes, concretamente entre distintas categorías: abrigos, bolsos, camisetas, pantalones, suéters, vestidos, zapatillas y por último zapatos.

Para esto voy a implementar el algoritmo Adaboost, explicado en clase de teoría. Este algoritmo propone entrenar iterativamente una serie de clasificadores base, de tal modo que cada nuevo clasificador preste mayor atención a los datos clasificados erróneamente por los clasificadores anteriores, y combinarlos de tal modo que se obtenga un clasificador con elevadas prestaciones. Para ello, durante una serie de iteraciones entrena un clasificador que implementa una función asignándole un peso de salida, y lo añade al conjunto de modo que la salida global del sistema se obtenga como combinación lineal ponderada de todos los clasificadores base.

2. Algoritmo Adaboost.

2.1 Explicación del pseudocódigo.

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 $H(x) = sign(\sum_t \alpha_t \cdot h_t(x))$

Lo primero es definir sus entradas, que en este caso son el conjunto de datos a utilizar junto con sus etiquetas para determinar de qué tipo son, determinar el número de iteraciones a realizar e inicializar la distribución de pesos otorgados a cada uno de los datos.

1. Inicializar Di = 1/N para i = 1 ... N
2. Para t = 1 .... T: Entrenar el clasificador débil utilizando la distribución Di
3. Obtener un valor de confianza para ht y calcular su error

4. Actualizar el la distribución Dt para i = 1 … N según los errores que hayan realizado en la etapa previa
5. Meta: Obtener un clasificador con error bajo
6. Devolvemos la hipótesis final H(x)

2.2 Explicación del Adaboost implementado.

```
public void AlgoritmoAdaboost(ArrayList<Imagen> entrenamiento, int cuantosDebiles, int numero){
    int res;
    double sumatorio = 0;
    double[] D = new double[entrenamiento.size()];
    inicializaPesos(D);

    for(int i = 0; i < 200; i++){
      ArrayList<ClasificadorDebil> debiles = new ArrayList<ClasificadorDebil>();
      ClasificadorDebil mejor = null;

        for(int j = 0; j < cuantosDebiles; j++){
            ClasificadorDebil ht = new ClasificadorDebil();
            ht.Entrenar(entrenamiento,D, numero);
            debiles.add(ht);
        }
        mejor = mejorClasificador(debiles);
        int []binario = new int[entrenamiento.size()];
        mejor.AplicaPixel(entrenamiento,binario);
        ActualizarDt(entrenamiento, mejor,D, binario, numero);
        cf.anyadeClasificador(mejor);
    }
}
```

Los argumentos pasados como parámetros son: El conjunto de imágenes de entrenamiento, el número de clasificadores débiles que deseamos generar para añadir a nuestro clasificador fuerte y por último la categoría que vamos a entrenar (tenemos en total 8 categorías por lo que formaremos 8 adaboost donde el 1 estará asociado a la primera categoría, así hasta la última).

En el método Adaboost tenemos nuestro vector de pesos D el cual inicialmente estará inicializado a 1/numTotal_imagenes, en mi caso he utilizado el 80% de las imágenes para entrenamiento y un 20% para test (tal y como me recomendaron en clase de prácticas).

Nuestro vector de pesos va a determinar cuál es el peso de cada imagen si nuestro clasificador falla, el peso de la imagen aumenta puesto que esto nos da señal de que la imagen es más complicada.

Para encontrar un buen clasificador débil a añadir al fuerte, generare 100 tal y como muestra el segundo bucle for, un clasificador débil trata de lo siguiente:

```
public class ClasificadorDebil{
    private int pixel;
    private int umbral;
    private int direccion;
    public double error;
    public double confianza;

    public ClasificadorDebil(){
        pixel = (int) (Math.random() * 784);
        umbral = (int) (Math.random() * 256) - 127;
        int valor = (int) (Math.random() * 2);
        if(valor == 0){
            direccion = -1;
        }
        else{
            direccion = 1;
        }
        error = 0.0;
        confianza = 0.0;
    }
```

Está formado por un pixel, un umbral y una dirección. Además tiene asociado un error y una confianza.

- El *umbral* tiene como rango de 0 a 255, aunque se restan 127 para que cuadre con la matriz de la imagen.
- La *dirección* nos sirve para realizar la clasificación.
- El número de *pixel* que nos sirve para clasificar dicho píxel en todas las imágenes.
- El *error* es la suma de los pesos en aquellas imágenes donde el clasificador no acierta.

- La *confianza* es el nivel de cuánto confiamos en ese clasificador, a menor error mayor confianza.

Todos estos valores serán generado de forma *aleatoria* para cada clasificador débil, por ello usamos el método random().

Una vez hemos generado el clasificado débil, realizamos el entrenamiento del mismo, para esto necesitamos el conjunto de entrenamiento, el vector de pesos D y el número de la categoría que vamos a entrenar.

```
public void Entrenar(ArrayList<Imagen> entrenamiento, double[] D, int numero){
  int [] binario = new int[entrenamiento.size()];

  for(int j = 0; j < entrenamiento.size(); j++){
      Imagen img = (Imagen) entrenamiento.get(j);

      byte imageData[] = img.getImageData();
      if(direccion == 1){
          if(imageData[pixel] < umbral){
            binario[j] = 1;
          }
          else{
            binario[j] = -1;
          }
      }
      else{
          if(imageData[pixel] >= umbral){
             binario[j] = 1;
          }
          else{
             binario[j] = -1;
          }
      }
  }
  int es;
  for(int i = 0; i < entrenamiento.size(); i++){
      if(numero == entrenamiento.get(i).getDigitoPertenece()){
          es = 1;
      }
      else{
          es = -1;
      }
      if(binario[i] == 1 && es == -1){
          this.error += D[i];
      }
      if(binario[i] == -1 && es == 1){
          this.error += D[i];
      }
  }
  double numerador = (double)(1 - error)/error;
  confianza = (double) 0.5 * Math.log(numerador);
}
```

Primero, creamos un vector binario con el tamaño del conjunto de entrenamiento. Recorremos todas las imágenes a través del bucle for extrayendo cada imagen que contiene dicho vector.

Una vez seleccionada esa imagen, cogemos su información (o bytes) y a través de la dirección determinaremos el valor en el vector binario (1 ó -1). Compararemos el valor que contiene dicha matriz de la imagen con nuestro umbral generado aleatoriamente tal y como se muestra.

Con otro for, volveos a recorrer de nuestro vector de imágenes. Previamente necesitamos filtrar a qué categoría pertenece cada imagen, es decir, si la categoría que queremos entrenar coincide con el dicha imagen seleccionada, marcamos a 1, si no a -1. Ahora como bien hemos explicado antes, se comprobará donde ha fallado nuestro clasificador en el entrenamiento y se sumará su error respectivamente, en el caso de fallo.

Una vez calculado su error a partir de este calculamos su confianza.

Se realizará este mismo proceso 100 veces. Tras esto, extraemos el mejor clasificador débil obtenido, con el siguiente método:

```
public ClasificadorDebil mejorClasificador(ArrayList<ClasificadorDebil> clasificadores){
    ClasificadorDebil mejor = null;

    for(ClasificadorDebil cd : clasificadores){
        if(mejor == null || cd.getError() < mejor.getError()){
            mejor = cd;
        }
    }
    return mejor;
}
```

El mejor clasificador débil obtenido se extrae recorriendo todos y escogiendo el que menor error contenga.

Al mejor clasificador débil obtenido, generare su vector binario para el conjunto de imágenes de entrenamiento, tal y como ya se ha mostrado anteriormente.

```
public void AplicaPixel(ArrayList<Imagen> vectorimg, int[] binario){

    for(int j = 0; j < vectorimg.size(); j++){
        Imagen img = (Imagen) vectorimg.get(j);

        byte imageData[] = img.getImageData();
        if(direccion == 1){
            if(imageData[pixel] < umbral){
              binario[j] = 1;
            }
            else{
              binario[j] = -1;
            }
        }
        else{
            if(imageData[pixel] >= umbral){
               binario[j] = 1;
            }
            else{
               binario[j] = -1;
            }
        }
    }
}
```

Observamos el pixel generado por ese clasificador en todas las imágenes y a partir de ahí bajo mi criterio obtenemos el vector binario.

Ahora será necesario actualizar la distribución D sobre el conjunto de entrenamiento.

```java
public void ActualizarDt(ArrayList<Imagen> entrenamiento, ClasificadorDebil mejor, double[] D, int[] binario, int numero){
    int res;

    for(int i = 0; i < D.length; i++){
        if(numero == entrenamiento.get(i).getDigitoPertenece()){
            res = 1;
        }
        else{
            res = -1;
        }

        if(binario[i] == -1 && res == 1){
            D[i] = (double) (D[i] * Math.pow(Math.E, mejor.getConfianza()));
        }
        else{
            if(binario[i] == 1 && res == -1){
                D[i] = (double) (D[i] * Math.pow(Math.E, mejor.getConfianza()));
            }
            else{
                D[i] = (double) (D[i] * Math.pow(Math.E, -1 * mejor.getConfianza()));
            }
        }
    }
    double Z = 0;
    for(int i = 0; i < D.length; i++){
        Z += D[i];
    }

    if(Z != 1){ //De ser 1 el resultado seria el mismo
        for(int i = 0; i < D.length; i++){
            D[i] = D[i]/Z;
        }
    }
}
```

Comenzamos recorriendo el vector de pesos, filtramos si las imágenes de entrenamiento corresponden a la categoría que deseamos entrenar. Lo comparamos con nuestro vector binario, de equivocarse, multiplicamos el peso de la imagen i por el número elevado a la confianza. En el caso de haber acertado, será la misma operación con la diferencia de que la confianza será negativa.

Para calcular Zt necesitamos tener este numerador calculado, al finalizar el bucle, comenzamos otro recorrido sumando cada peso de cada imagen. Finalmente si Z es distinto de 1 realizamos la división.

Una vez realizada la distribución añadimos nuestro mejor clasificador fuerte.

2.3 Explicación del Clasificador Fuerte.

```
public class ClasificadorFuerte{
    private ArrayList<ClasificadorDebil> cf;
    private double confianza;
    private double error;

    public ClasificadorFuerte(){
        cf = new ArrayList<ClasificadorDebil>();
    }
    public double getConfianza(){
        return confianza;
    }
    public double getError(){
        return error;
    }
    public ArrayList<ClasificadorDebil> getClasificadores(){
        return cf;
    }
    public void anyadeClasificador(ClasificadorDebil cl){
        cf.add(cl);
    }
}
```

Un clasificador fuerte está formado por muchos clasificadores débiles (tantos como nosotros queramos) en mi caso 200. Para ello utilizamos un ArrayList de Clasificadores Débiles. Cada uno tendrá asociado un error y una confianza.

2.4 Reparto de las imágenes.

```
public class CargaImagen {
    private ArrayList<Imagen> totales;
    private ArrayList<Imagen> entrenamiento;
    private ArrayList<Imagen> test;
    private int porcentaje;

    public CargaImagen(){
        totales = new ArrayList<>();
        entrenamiento = new ArrayList<>();
        test = new ArrayList<>();
    }
    public ArrayList<Imagen> getTotalImagenes(){
        return totales;
    }
    public ArrayList<Imagen> getImagenesEntr(){
        return entrenamiento;
    }
    public ArrayList<Imagen> getImagenesTest(){
        return test;
    }
```

Esta clase se encargará de almacenar las distintas imágenes y de organizarlas. Distingo entre 3 vectores, uno para almacenarlas todas, otro para las imágenes de entrenamiento y otro para las imágenes de test.

```
public void anyadeImagenes(DBLoader ml){
    for(int i = 0; i < 8; i++){
        ArrayList d_imgs = ml.getImageDatabaseForDigit(i);
        for(int j = 0; j < d_imgs.size(); j++){
            Imagen img = (Imagen) d_imgs.get(j);
            img.setDigitoPertenece(i);
            totales.add(img);
        }
    }
}
public void anyadeEntrenamiento(int porcent){
    porcentaje = porcent;
    double size1 = (4001.0*(porcent/100.0));
    int size2 = (int) size1;
    int valor;
    for(int i = 0; i < size2; i++){
        valor =  (int) (Math.random() * 4001);
        if(!entrenamiento.contains(totales.get(valor))){
            entrenamiento.add(totales.get(valor));
        }
        else{
            size2 = size2 + 1;
        }
    }
}
public void anyadeTest(int resto){
double size1 = (4001.0*(resto/100.0));
int size2 = (int) size1;
    int valor;
    for(int i = 0; i < size2; i++){
        valor = (int) (Math.random() * 4001);
        if(!entrenamiento.contains(totales.get(valor))){
            test.add(totales.get(valor));
        }
        else{
            size2 = size2 + 1;
        }
    }
}
```

Con el método anyadeImagenes, le pasamos la base de datos con las imágenes. Realizamos un bucle for donde vamos cargando las imágenes con la categoría asociada, con el segundo for iremos recorriendo las posiciones que hay en la carpeta de esa categoría. Usare una variable entera para tener controlado a qué categoría pertenece cada imagen, finalmente añadimos al vector e totales.

Con el método anyadeEntrenamiento pasamos el porcentaje que deseamos añadir para entrenamiento. Debemos de controlar que no se añaden

imágenes de forma repetida (puesto que sería realizar la misma prueba en la misma imagen x veces) para ello controlamos que el valor generado por random no lo contiene el vector.

El método anyadeTest realizamos el mismo proceso pero el % restante.

3. Ejecución y generación de los distintos Adaboost.

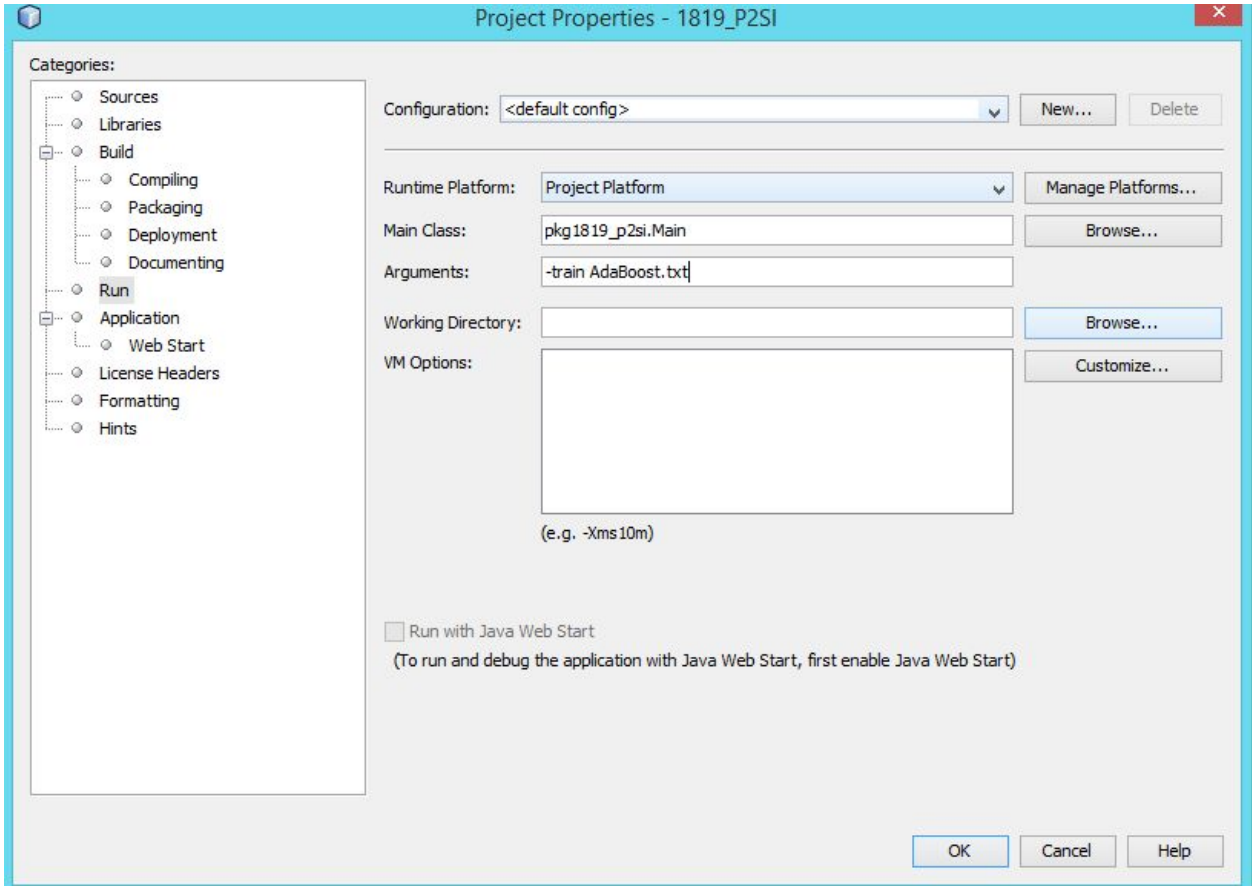
Todo esto lo realizamos a través de la clase Main y la clase Fichero.

Primero debemos de controlar los argumento como se indica en la práctica

1. Argumento: Adaboost -train <nombre_fichero.cf>
    a. De esta manera se entrenará adaboost para todas las categorías y el clasificador fuerte encontrado se almacenará en el fichero indicado. Se deben emitir por pantalla los porcentajes de acierto para el conjunto de entrenamiento y test.
2. Argumento: Adaboost -run <nombre_fichero.cf> <imagen_prueba>
    a. Se cargará un clasificador fuerte y se ejecutará sobre la imagen de prueba pasada por parámetro.

```
if(args.length >= 2){
    if(args[0].equals("-train")){
```

De esta forma controlamos la primera opción, como bien indica se entrenará el adaboost para todas las categorías y el clasificador fuerte se alamacenará en el fichero indicado, en mi caso AdaBoost.txt


Project Properties - 1819_P2SI
Categories:
Sources
Libraries
Build
Compiling
Packaging
Deployment
Documenting
Run
Application
Web Start
License Headers
Formatting
Hints
Configuration: <default config>
New...
Delete
Runtime Platform: Project Platform
Manage Platforms...
Main Class: pkg1819_p2si.Main
Browse...
Arguments: -train AdaBoost.txt
Working Directory:
Browse...
VM Options:
Customize...
(e.g. -Xms10m)
Run with Java Web Start
(To run and debug the application with Java Web Start, first enable Java Web Start)
OK
Cancel
Help

```
Main m = new Main();
    CargaImagen carga = new CargaImagen();
    carga.anyadeImagenes(ml);
    carga.anyadeEntrenamiento(80); //Escogemos por ejemplo un 80% para entrenar
    carga.anyadeTest(20); //Un 20% para test
    double contador = 0;

    if(args.length >= 2){
        if(args[0].equals("-train")){ //
            for(int i = 0; i < 8; i++){
                AdaBoost adb = new AdaBoost();
                adb.AlgoritmoAdaboost(carga.getImagenesEntr(), 100, i);
                contador = 0;

                for(int k = 0; k < carga.getImagenesEntr().size(); k++){
                    int coincide;
                        int res = adb.Clasifica(carga.getImagenesEntr().get(k));
                        if(i == carga.getImagenesEntr().get(k).getDigitoPertenece()){
                            coincide = 1;
                        }
                        else{
                            coincide = 0;
                        }
                        if(res == coincide){
                            contador++;
                        }
                }
                float resultado = (float) (((contador) / (carga.getImagenesEntr().size())) * 100);
```

Desde la base de datos cargamos las imágenes y las añadimos a nuestro vector de entrenamiento y test correspondiente (como ya indique, 80% entrenamiento y 20% test).

Para ello creamos y ejecutamos nuestro algoritmo adaboost, le pasamos las imágenes de entrenamiento, le indicamos que queremos 100 clasificadores débiles para seleccionar el mejor y “i” es la categoría que queremos entrenar.

Con el segundo for recorremos las imágenes de entrenamiento y clasificamos. Comparamos la categoría a la que pertenece la imagen con la que se está entrenando, si acierta se suma uno al contado para luego sacar la media de aciertos.

El método clasifica trata lo siguiente:

```
public int Clasifica(Imagen img){
    ClasificadorDebil d = null;
    int aplicado = 0;
    double confianza = 0.0;
    double suma = 0.0;

    for(int i = 0; i < cf.getClasificadores().size(); i++){
        d = cf.getClasificadores().get(i);
        aplicado = d.AplicaImagen(img);
        confianza = d.getConfianza();
        suma = suma + (aplicado * confianza);
    }
    if(suma > 0){
        return 1;
    }else if(suma < 0){
        return 0;
    }else{return 0;}
}
```

Recorremos nuestros clasificadores débiles que se hallan en el clasificador fuerte, le aplicamos el clasificador correspondiente para realizar el cálculo suma. Si este es mayor que 0 devolvemos 1 y si no devolveremos 0.

Donde AplicaImagen:

```
public int AplicaImagen(Imagen img){
    byte[] v_bytes = img.getImageData();
    int r = 0;

    if(direccion == 1){
        if(v_bytes[pixel] < umbral){
            r = 1;
        }
        else if(v_bytes[pixel] >= umbral){
            r = -1;
        }
    }
    else{
       if(v_bytes[pixel] < umbral){
            r = -1;
        }
        else if(v_bytes[pixel] >= umbral){
            r = 1;
        }
    }
    return r;
}
```

Bajo nuestro criterio y según el valor de ese píxel en la imagen asignamos 1 o -1

Con el resultados realizamos las media de lo que hemos acertado (extraemos la información del contador y realizamos una simple regla de 3).

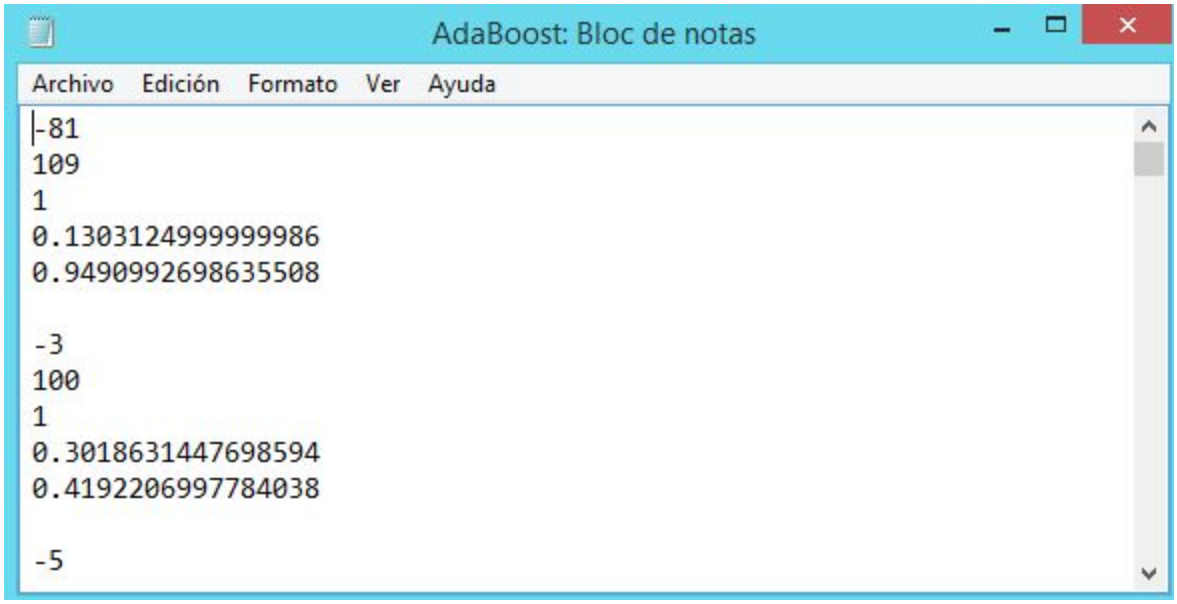
De nuevo, realizamos el mismo proceso con otro bucle for pero para las imágenes de test:

```
    float resultado = (float) (((contador) / (carga.getImagenesEntr().size())) * 100);
    System.out.print("Recuento de aciertos de entrenamiento para " + i + ":      ");
    System.out.println("Recuento de aciertos de test para: " + i + ":     ");
    System.out.print("     " + resultado +"%");
    contador = 0;
    for(int k = 0; k < carga.getImagenesTest().size(); k++){
        int coincide;
            int res = adb.Clasifica(carga.getImagenesTest().get(k));
            if(i == carga.getImagenesTest().get(k).getDigitoPertenece()){
                coincide = 1;
            }
            else{
                coincide = 0;
            }
            if(res == coincide){
                contador++;
            }
    }
    System.out.println("                                          " + (float) (((contador)/(carga
    m.anyade(adb);
}
Fichero.creaFichero(m, args[1]);
System.out.println("Fichero creado o sobreescrito!");
```

Escogemos de nuevo el resultado en % y creamos el fichero para almacenar ese ClasificadorFuerte en el archivo que hemos especificado como parámetro.

```java
public class Fichero {
    public static void creaFichero(Main m, String fich){
        try {
            FileWriter fichero = new FileWriter(fich);
            PrintWriter print = new PrintWriter(fich);
            int tamanyo = 0;
            for(int i = 0; i < 8; i++){
                tamanyo = m.getAda().get(i).getClasificadorFuerte().getClasificadores().size();
                for(int j = 0; j < tamanyo; j++){
                    ClasificadorDebil d = m.getAda().get(i).getClasificadorFuerte().getClasificadores().get(j);
                    print.println(d.getUmbral());
                    print.println(d.getPixel());
                    print.println(d.getDireccion());
                    print.println(d.getError());
                    print.println(d.getConfianza());
                    print.println();
                }
            }
        } catch (Exception ex) {
            System.out.println("Excepcion con el fichero" + ex );
        }
    }
```

A través de la clase Fichero creamos o simplemente sobreescribimos ese fichero si ya existe. La primera línea la formarán mis datos y los datos de los distintos clasificadores de la siguiente forma: umbral, pixel, dirección, error, confianza y un salto de línea y un nuevo clasificador.



El AdaBoost.txt se genera automáticamente en la carpeta del proyecto Netbeans.

```
if(args[0].equals("-run")){
        Fichero.leeFichero(m, args[1]);
        String directorio_imagen = "." + args[2];
        File file = new File(directorio_imagen);
        Imagen img1 = new Imagen(file.getAbsoluteFile());

        int digito_pertenece = 0;
        float r = m.AplicaFuertes(img1, m.getAda().get(0));

        for(int i = 1; i < m.getAda().size(); i++){
            float res = m.AplicaFuertes(img1, m.getAda().get(i));
            if(r < res){
                digito_pertenece = i;
                r = m.AplicaFuertes(img1, m.getAda().get(i));
            }
        }
        String cd = devolverElemento(digito_pertenece);
        System.out.println();
        System.out.println("La imagen es: " + cd);
    }
}
```

Ahora el segundo caso posible para especificar por argumentos. Primero comprobamos si el primer argumento es -run, seguido indicamos el fichero donde queremos que se carguen nuestros clasificadores y especificamos a qué imagen se la queremos aplicar. Lo primero será leer el fichero.

```
public static void leeFichero(Main m, String fich){
    try {
        String umbral, pixel, direccion, error, confianza;
        AdaBoost ada = null;
        FileReader reader = new FileReader(fich);
        BufferedReader buffer = new BufferedReader(reader);
        buffer.readLine();
        for(int i = 0; i < 8; i++){
            ada = new AdaBoost(); //10 adaboost
            ClasificadorDebil nuevo = null;
            for(int j = 0; j < 200; j++){
                nuevo = new ClasificadorDebil();
                umbral = buffer.readLine();
                pixel = buffer.readLine();
                direccion = buffer.readLine();
                error = buffer.readLine();
                confianza = buffer.readLine();
                buffer.readLine();
                nuevo.setUmbral(Integer.parseInt(umbral));
                nuevo.setPixel(Integer.parseInt(pixel));
                nuevo.setDireccion(Integer.parseInt(direccion));
                nuevo.setError(Double.parseDouble(error));
                nuevo.setConfianza(Double.parseDouble(confianza));
                ada.getClasificadorFuerte().anyadeClasificador(nuevo);
            }
            m.anyade(ada);
        }

    } catch (Exception ex) {

    }
}
```

Simplemente lo procesamos con FileReader y BufferedReader. Recogemos los datos leyendo las distintas líneas con readLine() y hacemos los setters correspondientes.

Finalmente vamos añadiendo a nuestro clasificador fuerte los distintos clasificadores y los 8 adaboost distintos al vector v_adaboost.

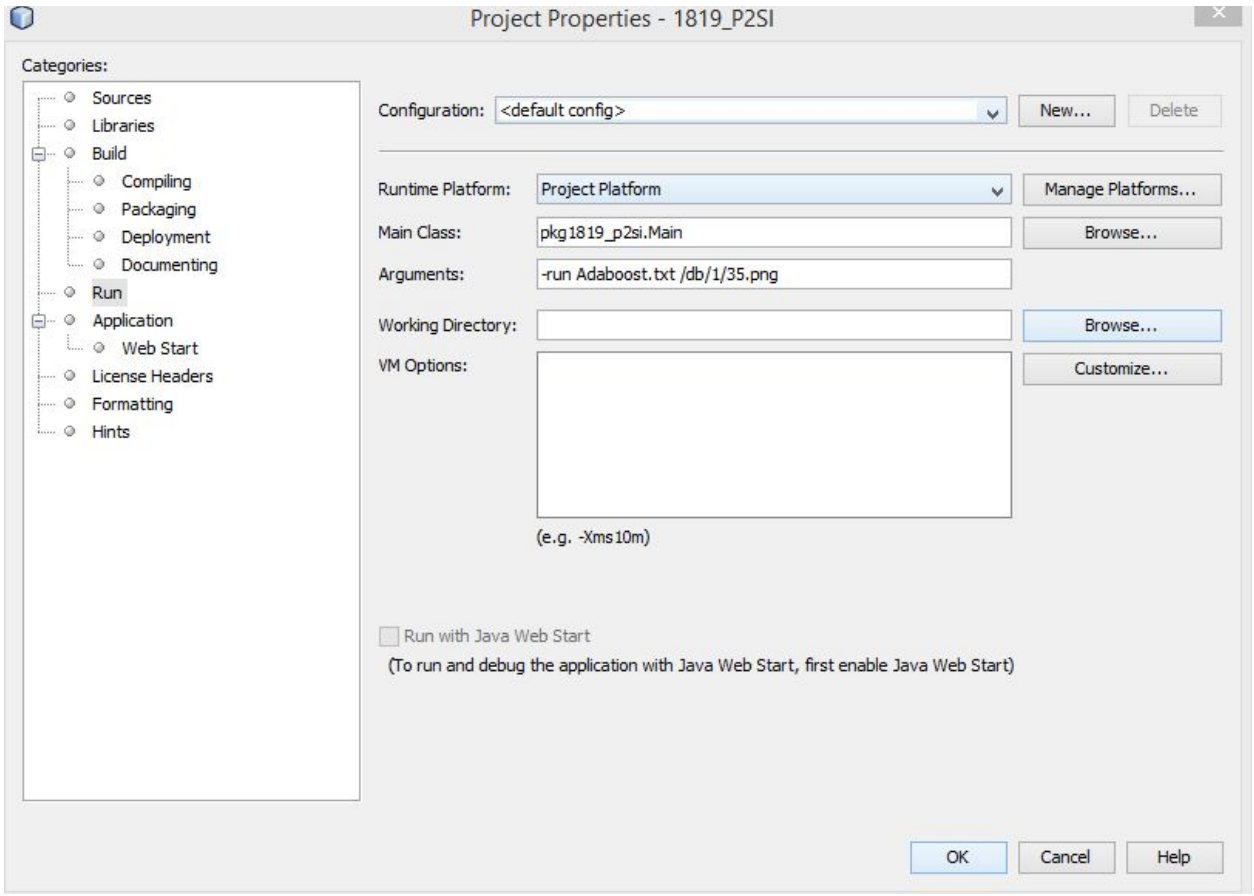
Por último aplicamos el clasificador tal y como se ha explicado anteriormente mostrando el resultado.

```java
public float AplicaFuertes(Imagen imagen, AdaBoost ada){
    float resultado = 0;

    for(int i = 0; i <  ada.getClasificadorFuerte().getClasificadores().size();i++){
        int aplicado = ada.getClasificadorFuerte().getClasificadores().get(i).AplicaImagen(imagen);
        double confianza = ada.getClasificadorFuerte().getClasificadores().get(i).getConfianza();
        resultado += (double)aplicado * confianza;
    }
    return resultado;
}
```

Salida para el primer modo:

```
Recuento de aciertos de entrenamiento para 0:      Recuento de aciertos de test para: 0:
     95.5%                                            93.875%

Recuento de aciertos de entrenamiento para 1:      Recuento de aciertos de test para: 1:
     98.875%                                          97.75%

Recuento de aciertos de entrenamiento para 2:      Recuento de aciertos de test para: 2:
     98.6875%                                         95.625%

Recuento de aciertos de entrenamiento para 3:      Recuento de aciertos de test para: 3:
     99.90625%                                        97.875%

Recuento de aciertos de entrenamiento para 4:      Recuento de aciertos de test para: 4:
     95.96875%                                        93.875%

Recuento de aciertos de entrenamiento para 5:      Recuento de aciertos de test para: 5:
     97.09375%                                        93.0%

Recuento de aciertos de entrenamiento para 6:      Recuento de aciertos de test para: 6:
     99.625%                                          98.375%

Recuento de aciertos de entrenamiento para 7:      Recuento de aciertos de test para: 7:
     99.15625%                                        97.0%
Fichero creado o sobreescrito!
```

Segundo caso:

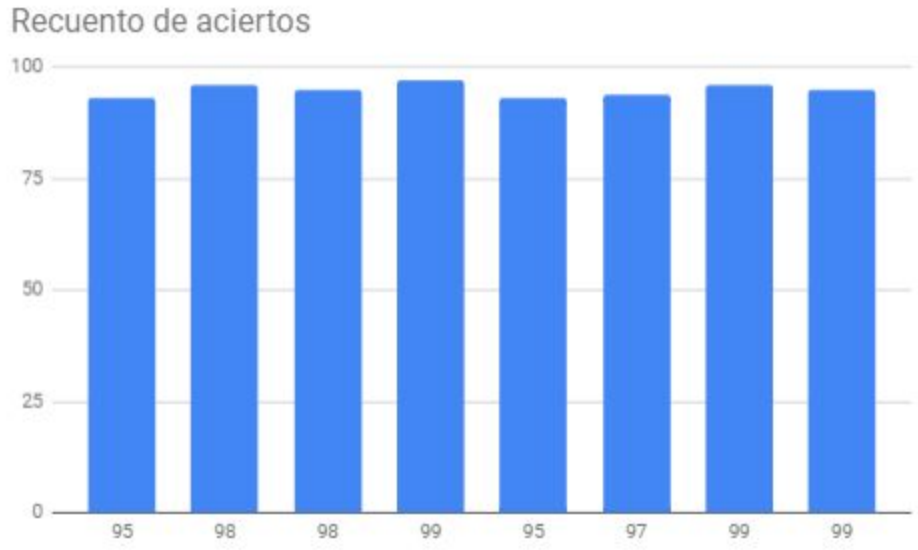



```
Image pixels: FF FF FF FF
La imagen es: bolso
```

4. Cuestiones a resolver

1. ¿Cuál es el número de clasificadores (T) que se han de generar para que un clasificador débil funcione? ¿Cuánto has entrenado cada clasificador (A)? ¿Por qué? Muestra una gráfica que permita verificar lo que contestas.

Mínimo hay que generar 100 clasificadores, se ha entrenado 100 veces, por que ha sido cuando he detectado que el algoritmo empezaba a mostrar soluciones correctas.



2. ¿Cómo afecta el número de clasificadores débiles generados y su entrenamiento al tiempo empleado para el proceso de aprendizaje? ¿Qué importancia le darías? Justifica tu respuesta.

Cuanto más clasificadores generamos, más coste computacional requerirá dado que el proceso de aprendizaje será más lento, pero a la vez podremos obtener mayor calidad de los clasificadores.

3. ¿Como has dividido los datos en conjunto de entrenamiento y test? ¿Para que es útil hacer esta división?

Es útil para tener una buena organización en el conjunto, observar qué funciona mejor y evitar sobre entrenamiento. Mis conjuntos están divididos de tal forma que hay un 80% de las imágenes para entrenamiento y un 20% para test.

4. ¿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.

A partir del 85% empieza a producirse. Esto ocurre puesto que el algoritmo empieza a quedarse ajustado a una características muy específicas de los datos de entrenamiento sin relación con el objetivo deseado.

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

El funcionamiento ha sido detallado durante el apartado 2, suelo obtener un 95% - 99% de acierto en la fase de aprendizaje y aproximadamente un 93% - 97% en la fase de test.

6. ¿Cómo has conseguido que Adaboost clasifique entre las 8 clases cuando solo tiene un salida binaria?

Ejecutando el Adaboost 8 veces.

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

He contemplado algún tipo de confusión en la categoría 5 y 7