Sistemas Inteligentes

Práctica 2

Visión Artificial y Aprendizaje

Alejandro Jesús Aliaga Hyder
48765284-V

Grupo 6 prácticas
Viernes 09:00 - 11:00

Índice

1 Introducción ……P.3
2 AdaBoost ……..P.3
3 Clasificadores ……..P.4-5
3.1 Débil
3.2 Fuerte
4 **Cantidad de clasificadores usados ……P.6**
5 **Tiempo de aprendizaje ……...P.6-7**
6 Conjuntos de aprendizaje y testeo ……P.7-8
6.1 ¿Cómo has realizado la división?
6.2 ¿Para qué es útil esta división?
7 **Sobreentrenamiento ……P.9**
8 **Clasificando los dígitos ……P.9**
9 **Conclusión……..P.10**

1. Introducción

En esta segunda práctica de la asignatura, vamos a desarrollar un sistema capaz de distinguir dígitos manuscritos. Se ha implementado un sistema de aprendizaje automatizado supervisado. Para lograr esto se hará uso del algoritmo AdaBoost. Antes de responder a las preguntas propuestas en la documentación de la práctica explicaré en que consiste AdaBoost.

La salida obtenida por el algoritmo AdaBoost está formada por una serie de clasificadores más sencillos cuyo objetivo es obtener un error algo menor que una clasificación aleatoria. El conjunto de estos se llama clasificador Fuerte. Realizaremos un clasificador fuerte para cada dígito a clasificar, el conjunto de los clasificadores fuertes correspondientes a cada dígito formarán el clasificador encargado de distinguir entre las imágenes pasadas.

Para finalizar la práctica, realizaremos una serie de pruebas al clasificador obtenido. Podremos saber el porcentaje de aciertos y como podemos mejorar este.

2. AdaBoost

En este apartado previo a la explicación del código y resultados obtenidos voy a explicar en que consistes el algoritmo AdaBoost.

Hay una serie de algoritmo denominados “Boosting”, son algoritmos de aprendizaje automático que reducen el sesgo de error mediante aprendizaje supervisado. Estos están basados en el cuestión de formar un clasificador robusto (distingue las clases en el espacio) mediante otro clasificadores poco robustos llamados débiles.

Dada una cantidad de imágenes, un clasificador puede aprender de ellas y así clasificar automáticamente las futuras imágenes. Estos clasificadores (débiles) son construidos a partir de una características de los objetos, en este caso será un píxel de las imágenes.

AdaBoost propone entrenar de forma iterativa una serie de clasificadores base, al que llamamos conjunto de entrenamiento, de tal modo que cada nuevo clasificador preste mayor atención a los datos clasificados erróneamente. Para esto itera cada nuevo clasificar en función del número de pruebas que queramos realizar a cada clasificador, asignándole un peso.

Para conseguir que cada nuevo clasificador preste mayor atención a los datos más erróneos cada clasificador débil va a tener una característica que almacena el error. Esta trata de una función que pondera la importancia de cada error en función de la confianza que tengamos en el durante el entrenamiento.

3. Clasificadores

A continuación se van a explicar más en profundidad los clasificadores entrenados mediante el adaBoost y las funciones que los componen.

3.1 Débil

Objetos que representan una clasificación poco fiable de las clases. La forma en que clasifican el espacio es dividir el espacio de la imagen (un vector con los píxels que conforman la imagen) en dos partes y especificar que los objetos que quedan en un lado se van a clasificar según la clase y los objeto de una clase en el otro lado.

La forma que he usado para poder distinguir los lados en los que se divide el espacio, es generar de forma aleatoria un número que es positivo o negativo. En caso que la dirección sea positiva la clasificación correcta se encontrará a la izquierda, al contrario, cuando es negativo. Este es el código que representa el umbral.

```
boolean resultado = false;

if ( _direccion == 1 )
{
    if ( _umbral < imagen.getImageData()[_pixel] )
        resultado = true;
}
else if ( _umbral >= imagen.getImageData()[_pixel] )
        resultado = true;

return resultado;
```

Esto será usado por el clasificador débil para aplicarlo a un conjunto de imágenes de entrenamiento. Para cada imagen empleada para entrenar el débil, se usa el código anterior. La función aplicarClasificadorDebil queda de la siguiente forma:

```
ArrayList<Boolean> clasificacion = new ArrayList();

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

    aux = h (img);

    clasificacion.add(aux);
}
return clasificacion;
```

Para concluir con la explicación voy a enseñar la forma en que obtengo el error y la confianza a partir de este.

Una vez obtenida la clasificación que da el débil sobre unas imágenes de entrenamiento. Esta es comparada con la clasificación real de la imágenes. Para ello comparamos ambas listas y en caso de obtener una mala clasificación por parte del débil tendremos un error, este es ponderado en función del peso de cada imagen (que se explicará más adelante). Una vez obtenido el error del débil, podemos obtener la confianza de este en función del error. El código que representa el cálculo del error y de la confianza es el siguiente:

```
ArrayList clasificacion =  this.aplicarClasificadorDebil ( aprendizaje );

for ( int i = 0; i < clasificacion.size(); ++i )
{
    if ( !clasificacion.get(i).equals(correcto.get(i)) )
        _error += aprendizaje.get(i).getPeso();
}
_confianza = 0.5f * (float)Math.log10(1.0f - _error ) / _error;
```

3.2 Fuerte

Una clasificador fuerte es aquel formado por con seria de clasificadores débiles previamente seleccionados en adaBoost. Para obtener la clasificación sobre una imagen dada tenemos que calcular el sumatorio de las confianzas.

Para ello se obtendrá la h de cada clasificador débil sobre la imagen, si el débil decide que la imagen pasada está correctamente clasificada, sumaremos la confianza de ese clasificador, en caso en contrario restaremos la confianza.

El código que representa el cálculo de la clasificación dada por el fuerte de un dígito es la siguiente:

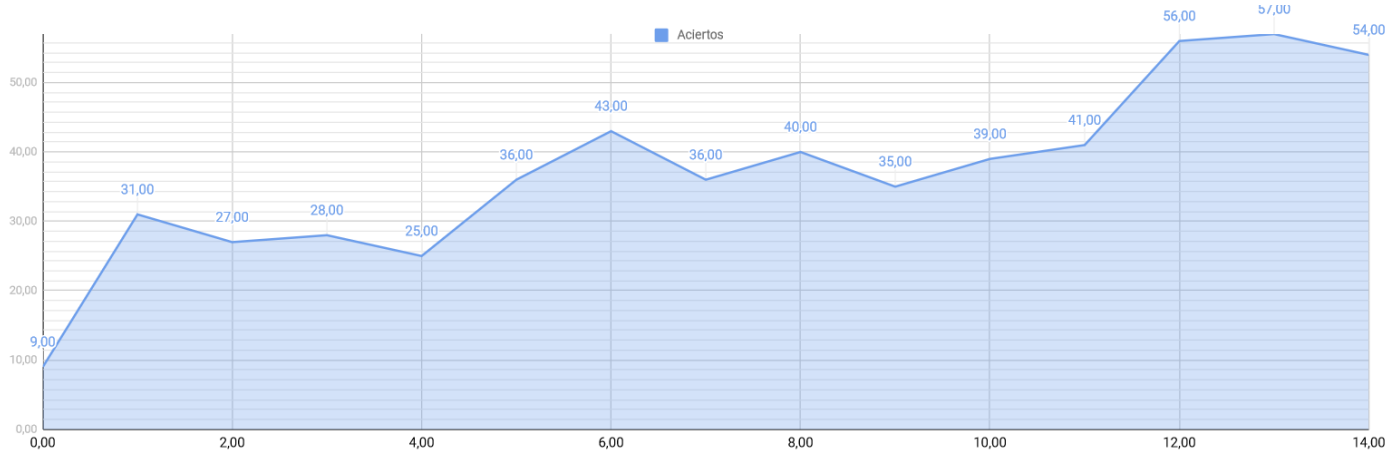
```
float H = 0.0f;
boolean h;
float conf;

for ( int i = 0; i < _debiles.size(); ++i )
{
    h = _debiles.get(i).h(imagen);
    conf = _debiles.get(i).getConfianza();
    if ( h )
        H += conf;
    else
        H -= conf;
}
return H;
```

4. Cantidad de clasificadores usados

Enunciado original.
¿Cuál es el número de clasificadores que se han de generar para que un clasificador débil funcione? Muestra una gráfica que permita verificar lo que comentas

A continuación voy a mostrar como varía el porcentaje de aciertos según aumentamos la cantidad de clasificadores débiles que forman uno fuerte. Para estas pruebas el porcentaje de entrenamiento será 72 y el número de pruebas para obtener cada clasificador débil 1008.



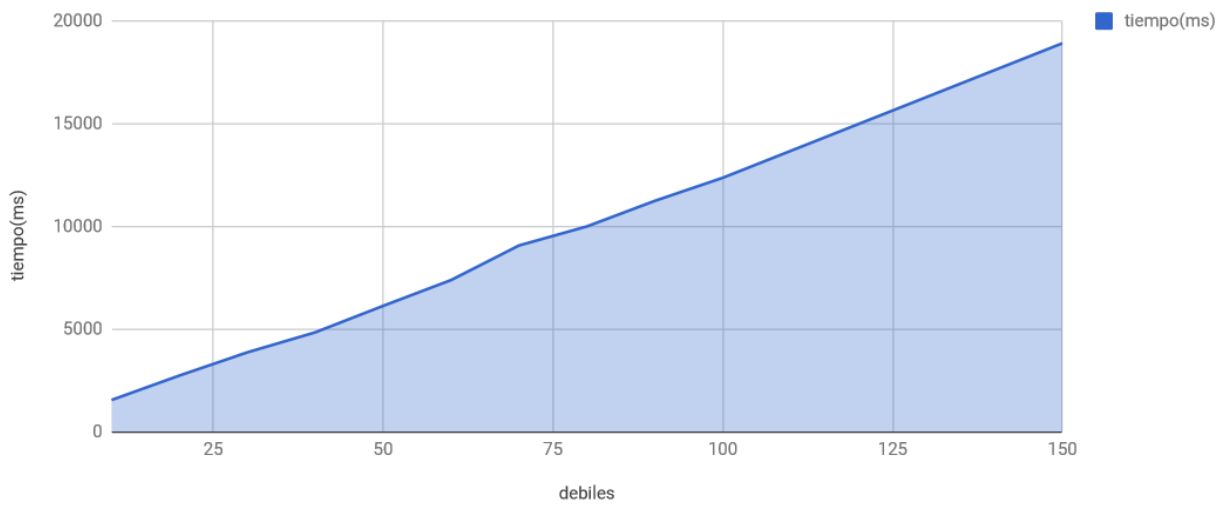
Entendiendo porque funciona el clasificador obtener un porcentaje de aciertos mayor al azar, a partir de 14 débiles se obtiene de forma estable un porcentaje mayor al 50%

5. Tiempo de aprendizaje

Enunciado original.
¿Cómo afecta el número de clasificadores generados al tiempo empleado para el proceso de aprendizaje? ¿Qué importancia le darías? Justifica tu respuesta.

Por cada clasificador debil que añadimos, tenemos que hacer 1000 pruebas. Por tanto cuando mayor número de débiles se incrementa muchísimo la cantidad de operaciones.

La siguiente gráfica muestra los tiempos de ejecución según aumentamos la cantida de clasificadores débiles.



6. Conjunto de aprendizaje y testeo

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

6.1 ¿Cómo has realizado la división?

Las imágenes las he separado en conjuntos a partir de un porcentaje previamente definido. Como los dígitos tienen distinta cantidad de imágenes por capeta, he de obtener para cada dígito la cantidad de imágenes de su carpeta, y una vez obtenida le aplico el porcentaje.

Tras realizar este proceso en las imágenes de cada dígito la suma de estás concluirá el porcentaje de imágenes del total de entrenamiento.

Para no tener problemas a la hora de despreciar decimales cuando calculo el porcentaje de imágenes para un dígito, este lo guardo para posteriormente añadir la cantidad de imágenes que componen la suma del error.

En cuanto a las imágenes de testeo, tan solo serán las imágenes restantes.

Esta es una parte del código, donde separo las imágenes y voy acumulando el error.

```
ArrayList imgs = _ml.getImageDatabaseForDigit(i);
digitos = imgs.size();

separarImagenes =  _porcentajeEntrenamiento * digitos / 100;
sde = (float)digitos * _porcentajeEntrenamiento / 100.0f;
sumaError += sde - (int)sde;

tipo = 0;
for ( int j = 0; j < digitos; ++j )
{
    Imagen img = (Imagen) imgs.get(j);

    if ( j <  separarImagenes )
    {
        _aprendizaje.add ( img );
        tipo++;
    }

    else
    {
        _testeo.add ( img );
        _correctoTesteo.add(i);
    }

}
```

6.2 ¿Para qué es útil esta división?

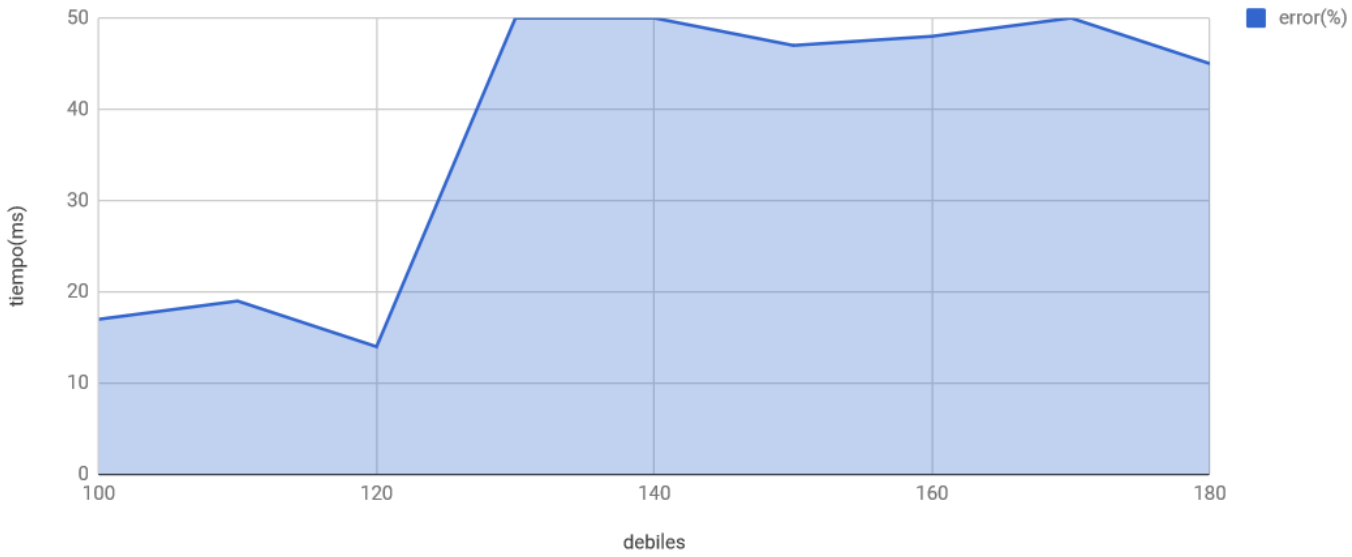
AdaBoost es un algoritmo supervisado, este necesita de entrenamiento previo de los clasificadores para obtener uno fuerte. Por ello es necesario un conjunto de imágenes de las que sepamos a la clase que pertenecen.

Del conjunto de imágenes no todas ellas son necesarias para entrenar el fuerte, ya que podemos causar sobreentrenamiento, si el conjunto es separado, podemos controlar la cantidad de imágenes a entrenar para evitarlo y además tendremos unas imágenes para poder testear el fuerte.

7. Sobreentrenamiento

Enunciado original.
¿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.

En la siguiente gráfica podemos observar como aumenta el porcentaje de error a medida que aumentamos la cantidad de clasificadores débiles que forman un clasificador fuerte. En mi práctica a partir de los 120 clasificadores se puede percibir una gran aumento en el error.



8. Clasificando los dígitos

Enunciado original.
¿Cómo has conseguido que Adaboost clasifique entre los 10 dígitos cuando solo tiene una salida binaria?

Para cada imagen de testeo, se calcula el valor de la H de los clasificadores fuertes de cada dígito. Aquel clasificador que obtenga el valor más alto de H será aquel que más confianza tenga en que está bien clasificado. Si los clasificadores están implementados correctamente, cuando les llegue una imagen no correspondiente a ellos devolverán una H muy baja. De esta forma se pueden distinguir los dígitos.

9. Conclusión

Durante la realización de la práctica he podido comprobar la importancia del boosting y de sus aplicaciones. Al analizar varios aspectos de este una vez implementado, me he dado cuenta que podía mejorar la tasa de aciertos llevando al límite las características del algoritmo que sin perjudicar la salida.