SISTEMAS INTELIGENTES

Departamento de Ciencia de la Computación e Inteligencia Artificial

Universidad de Alicante **Curso 2017/2018**

v1.2

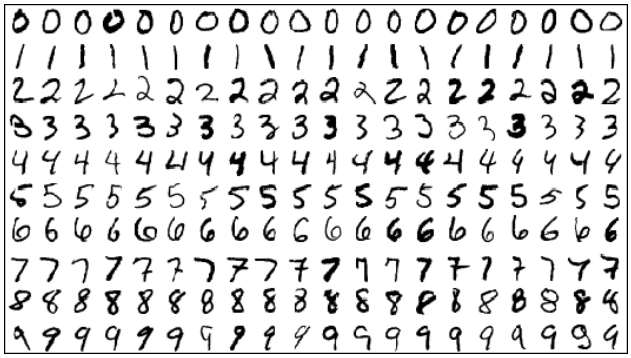
Práctica 2. Visión Artificial y Aprendizaje

Objetivos:

- Comprender el funcionamiento del aprendizaje automático supervisado y en concreto el método de Boosting.
- Entender el concepto de imagen y píxel, y cómo podemos aprender a distinguir entre imágenes a partir de la información que aparece en ellas.
- Entender el papel de un clasificador débil en métodos de Boosting.
- Implementar el algoritmo AdaBoost así como un clasificador débil de umbral
- Aplicar AdaBoost al problema de clasificación de dígitos manuscritos
- Realizar un análisis cuantitativo de la tasa de aciertos obtenida mediante este método de aprendizaje

Sesión 1. Introducción

En la segunda práctica de la asignatura se va a desarrollar un sistema capaz de distinguir entre dígitos manuscritos. Para ello se va a implementar un sistema de aprendizaje automático supervisado. La entrada al sistema consistirá en un conjunto de imágenes etiquetadas según la clase a la que pertenezcan (los dígitos de 0 a 9). El objetivo de esta práctica es aprender un clasificador en base a este conjunto de entrada que permita clasificar sin problemas imágenes pertenecientes a estas clases, aunque no se hayan visto anteriormente. La base de datos con la que vamos a trabajar será un subconjunto de la base de datos MNIST (http://yann.lecun.com/exdb/mnist/) . El objetivo final sería construir un clasificador que, tras ser entrenado, pueda decirnos a que dígito manuscrito corresponde una imagen.



1.1 Netbeans y Java

Del mismo modo que en la primera práctica, para el desarrollo de esta segunda se utilizará NetBeans y el lenguaje Java. NetBeans es un entorno de desarrollo integrado, hecho principalmente para el lenguaje de programación Java. También es un proyecto de código abierto, por lo tanto, lo podemos descargar libremente desde su página oficial: *http://netbeans.org/*

1.2 Inicio del proyecto

En la página moodle de la asignatura encontraréis una plantilla de proyecto para netbeans. Básicamente, define una clase imagen, que proporcionará los métodos requeridos para cargar y acceder a nivel de pixel a una imagen. A partir de esa clase imagen se construye un cargador automático del conjunto MNIST (que esté disponible en la ruta *1718_P2SI/mnist1000)*. Como indica su nombre se incluyen los 1000 primeros dígitos de este conjunto (en la BD original hay 60000). Además, se proporciona una clase de ayuda para que podáis visualizar las imágenes que carguéis.

Para empezar la toma de contacto con la plantilla de ayuda, contestar las siguientes preguntas:

- Familiarízate con la base de datos MNIST y entiende su finalidad. Puedes echar un vistazo a la página oficial (http://yann.lecun.com/exdb/mnist/).
- ¿Qué incluye el fichero Main-java? Especifica que ejemplos básicos incluye y como se podrán utilizar después en la práctica
- Comprueba que MNISTLoader carga correctamente las imágenes de la base de datos. En caso contrario revisa que la ruta de acceso sea la correcta. ¿Cuál es el formado de base de datos que especifica MNISTLoader? ¿Cómo se puede acceder a una imagen en concreto? ¿Cómo sabemos que dígito representa una imagen?
- ¿Cómo se muestra una imagen cargada previamente a partir de MNISTLoader?

1.3 Especificación de las imágenes de entrada

Como ya hemos dicho, las imágenes con las que vamos a trabajar corresponden a varios tipos: 10 tipos, dígitos de 0 al 9. Las imágenes están almacenadas en escala de grises. Cada imagen va a estar representada por un vector característico. El número de componentes de este vector vendrá determinado por el número de píxeles que componen la imagen y el valor de cada componente se corresponderá con el valor de gris de cada píxel.

Sesión 2. Adaboost

2.1 Clasificadores débiles

Teniendo en cuenta que cada imagen se especifica por un vector característico del mismo tamaño que la imagen, en el caso de las imágenes que se os han proporcionado, el tamaño de este vector es de 28x28 = 784 componentes. Podemos establecer entonces que las imágenes, representadas por su vector característico, se van a encontrar distribuidas en un espacio de 784

dimensiones. El objetivo final de la práctica es encontrar un clasificador robusto que divida este espacio de manera que se distinga la parte del espacio en la que está un dígito concreto.

Para obtener este clasificador robusto, vamos a comenzar por crear clasificadores no tan robustos, pero que al menos nos sirvan para realizar una clasificación. Estos clasificadores se conocen como clasificadores débiles.

La forma más sencilla de clasificar objetos distribuidos en un espacio es dividir el espacio en dos partes y especificar que los objetos que quedan a un lado se van a clasificar según una clase y los objetos que quedan al otro lado según la otra clase. Esto es posible realizarlo de manera muy sencilla utilizando un umbral. Los umbrales pueden ser positivos o negativos y especifican para un determinado pixel (solo uno) el rango aceptable para el mismo.

Para obtener un clasificador débil correcto hemos de realizar un proceso de "aprendizaje". La idea es generar aleatoriamente **c** umbrales y utilizarlos para clasificar el conjunto de aprendizaje. Finalmente, nos quedaremos con el umbral que mejor clasifique, es decir, aquel que obtenga menor tasa de error para el conjunto dado.

Seguimos con la implementación del clasificador débil. Uno de los parámetros que tendremos que ajustar es el número de clasificadores que se van a generar aleatoriamente cada vez que entrenemos un clasificador débil. Este valor ha de ser establecido empíricamente mediante pruebas. La condición para un clasificador débil es que ha de mejorar cualquier clasificación arbitraria.

Más concretamente, se deben crear las siguientes funciones para trabajar con cualquier tipo de clasificador débil en Adaboost:

- *clasificador = generarClasificadorAzar(dimensión de los datos)*
  genera un clasificador débil, dentro del espacio de búsqueda, al azar. Requiere saber la dimensión de los puntos con los que se trabajará. Para el caso de umbrales un clasificador contendrá el pixel al que afecta, la dirección del umbral y el valor del umbral. Se utilizará dentro de Adaboost.

- *resultado_clasificacion = aplicarClasificadorDebil(clasificador,datos)*
  aplica un clasificador débil a los puntos. Devuelve un vector booleano con la clasificación. Se utilizará para aplicar el clasificador fuerte después de haberlo aprendido con Adaboost.

- *obtenerErrorClasificador(clasificador,datos,D)*
  obtiene el error de clasificación a partir del vector D (ver siguiente apartado) y la clasificación correcta de los datos. Se utilizará dentro de Adaboost.

2.2 Diseño inicial del algoritmo AdaBoost

Vamos a continuar con el diseño y la implementación del método de Boosting. El algoritmo que vamos a implementar se llama AdaBoost. Como clasificador débil utilizaremos el desarrollado en las sesiones anteriores.

Alg Adaboost
Inicializamos distribución de pesos $D_1(i) = 1/N$ sobre el conjunto de entrenamiento
para t=1 hasta T
Entrenar clasificador débil ht a partir de Dt
Calcular el valor de confianza para ht: $\alpha_t = ½ \text{ Ln}(1-\varepsilon_t/\varepsilon_t)$
donde $\varepsilon_t = Pr_{Dt}[h_t(x_i)!=y_i]$
Actualizar distribución D sobre el conjunto de entrenamiento:

$$D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

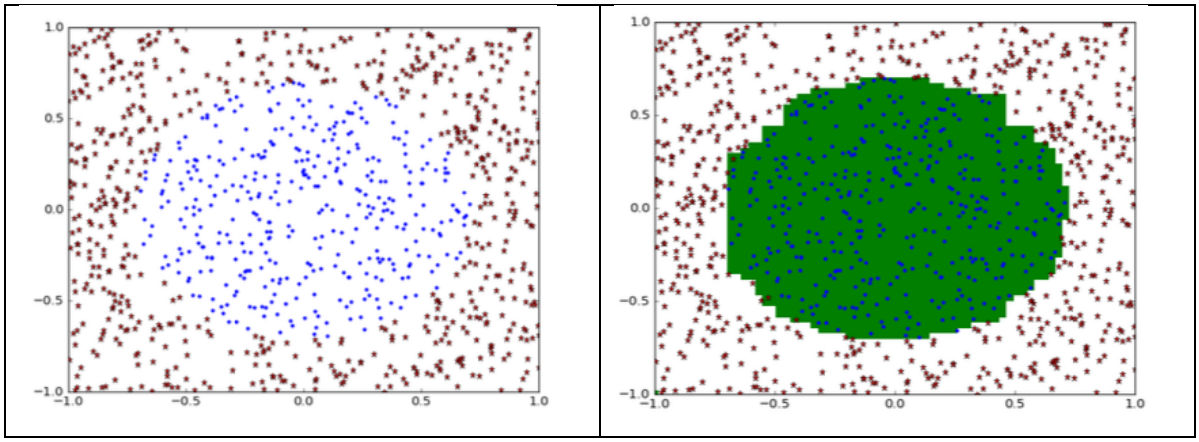
Actualizar el clasificador fuerte:

$$H(x) = sign(f(x)) = sign\left(\sum_{i=1}^{T} \alpha_t h_t(x)\right)$$

si $Pr_{Dt}[H(x_i)!=y_i] = 0$
devolver H
fpara
devolver H
falg

Sesión 3. Finalización de Adaboost. Pruebas utilizando datos 2D

Se propone de manera opcional, para facilitar la depuración del algoritmo, generar un conjunto de datos 2D etiquetados y proceder a su clasificación mediante Adaboost. El algoritmo genérico n-dimensional no debería variar con respecto al bidimensional pero este último facilita visualizar que está realizando el algoritmo. A continuación, se muestra un ejemplo de puntos etiquetados y un ejemplo de un clasificador fuerte obtenido mediante Adaboost utilizando umbrales:



Sesión 4. Implementación y depuración del algoritmo AdaBoost para datos n-dimensionales

Continuamos en las siguientes sesiones con la implementación del algoritmo AdaBoost. Nos centraremos en comprobar que todas las partes del algoritmo se ejecutan correctamente, prestando especial atención al entrenamiento de clasificadores débiles utilizando D (no deberíamos obtener dos veces el mismo clasificador ya que D cambia en cada iteración), el correcto cálculo de ε_t y α_t y, sobre todo, que el vector D se actualiza y se normaliza correctamente.

Sesión 5. Pruebas del algoritmo Adaboost n-dimensional para clasificar únicamente entre 0 y el resto de dígitos

Debemos probar el algoritmo AdaBoost para conseguir ajustar sus parámetros de entrada y así obtener el mejor resultado. Empezaremos probando exclusivamente Adaboost para clasificar el dígito 0 con respecto a todos los demás. Los parámetros a ajustar son:

- Porcentaje de ejemplos utilizados para el test.
- Número máximo de iteraciones de AdaBoost
- Número de umbrales que se van a generar al entrenar un clasificador débil

Todo ello debe ir orientado a obtener el mejor porcentaje de clasificación posible evitando el **sobre-entrenamiento**.

Sesión 6. Pruebas del algoritmo Adaboost para todos los dígitos y serialización

Debemos probar el algoritmo AdaBoost para conseguir ajustar sus parámetros de entrada y así obtener el mejor resultado. El algoritmo Adaboost se ejecutara 10 veces (una por dígito). Se deberá establecer como determinar la fusión de los resultados de los 10 clasificadores fuertes para obtener un único resultado (a que dígito pertenece la imagen).

Además, se debe implementar tanto el guardado del mejor clasificador que hayáis encontrado como la carga de clasificadores. La clase principal del proyecto (que contendrá el método *main*) se llamará Adaboost.java y podrá recibir los siguientes parámetros:

- `1) Adaboost –t <fichero_almacenamiento_clasificador_fuerte>`
  De esta manera se entrenará adaboost para todos los dígitos 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) Adaboost <fichero_origen_clasificador_fuerte> <imagen_prueba>`
  Se cargará un clasificador fuerte y se ejecutará sobre la imagen de prueba pasada por parámetro.
  Nota: el proceso de carga y uso de imágenes se utiliza en la plantilla adjunta para cargar la BD MNIST, podéis obtenerlo de allí.

- Por defecto, entregareis el proyecto para que use vuestro mejor clasificador sobre una imagen de vuestra elección (opción 2, con los parámetros establecidos como corresponda)

Sesión 7. Documentación del clasificador

En base al trabajo realizado en las sesiones anteriores, en este apartado de la práctica se deben investigar varias cosas que tenéis que reflejar de manera clara y razonada en la memoria:

- ¿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.
- ¿Cómo afecta el número de clasificadores generados al tiempo empleado para el proceso de aprendizaje? ¿Qué importancia le darías? Justifica tu respuesta.
- ¿Cómo has dividido los datos en conjunto de entrenamiento y test? ¿Para que es útil hacer esta división?
- ¿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.
- Comenta detalladamente el funcionamiento de AdaBoost teniendo en que tasa media de fallos obtienes para aprendizaje y test.
- ¿Cómo has conseguido que Adaboost clasifique entre los 10 dígitos cuando solo tiene una salida binaria?

Parte optativa. Probando otro tipo de clasificadores débiles

Prueba a implementar un hiperplano para que haga la misma tarea de clasificación que la que se realiza con los umbrales. Para generar los umbrales solo se utiliza el valor de un pixel, sin embargo, los hiperplanos deben relacionar todos los pixels de la imagen. Por ejemplo, para clasificar un punto 2D con clasificadores de umbral cada clasificador prueba si una coordenada está dentro o fuera del umbral. Deseamos crear una línea que pueda clasificar los puntos 2D. La línea relaciona todas las coordenadas del punto.

Lo que se pretende por tanto es dividir el espacio en dos partes y especificar que los objetos que quedan a un lado se van a clasificar según una clase y los objetos que quedan al otro lado según la otra clase. Esto es posible realizarlo de manera sencilla con un hiperplano. Decimos que en un espacio de dimensión D, el hiperplano es el objeto geométrico plano que divide ese espacio en dos. Así, el hiperplano en D=2 es una recta, en D=3 es un plano, etc. De manera general, definimos un hiperplano en dimensión D cómo los puntos (P=$\{p_1, p_2, \ldots, p_d\}$) del espacio D-dimensional que cumplen:

$$h^D: \sum_{i=1}^{D} (x_i p_i) - C = 0$$

Así, todos los puntos del espacio D que al ser sustituidos en la equación anterior nos devuelven un valor negativo estarán al lado negativo del espacio según el hiperplano y, al contrario, los que nos devuelven un valor positivo están al otro lado del espacio. Por lo que un hiperplano es un candidato ideal para crear un clasificador simple.

- Se debe utilizar vuestro algoritmo Adaboost con clasificadores débiles de tipo hiperplano

- Comparar los resultados con respecto al clasificador de umbral y determinar si vale la plena utilizar este tipo de clasificadores en cuanto a tiempo requerido de Adaboost y mejora de resultados.

Recuerda que no cambiará nada de Adaboost, solo hay que reescribir las siguientes funciones:

- *clasificador = generarClasificadorAzar(dimensión de los datos)*
- *resultado_clasificacion = aplicarClasificadorDebil(clasificador,datos)*
- *obtenerErrorClasificador(clasificador,datos,D)*

Formato de entrega

Un fichero comprimido con el siguiente contenido. **Utilizad vuestro nombre y apellidos como nombre de fichero**:

- Leeme.txt: Cualquier cosa que se quiera hacer notar sobre el funcionamiento de la práctica o la entrega.
- /Practica2SI : Proyecto netbeans de AdaBoost. **Debe contener dentro del mismo, el mejor clasificador fuerte encontrado. Revisar que cumple los criterios especificados en la sesión 6 de este documento.**
- /Practica2SIOptativa: Proyecto netbeans con la parte optativa de la práctica.
- /Doc: Documentación, en formato PDF, de la parte obligatoria de la práctica. Debe contener dentro de esta misma carpeta pero en otro fichero, la documentación en PDF de la parte optativa, si se ha realizado.

Fecha de entrega

Viernes 22 de diciembre a las 23:55 por el Moodle de la asignatura