SISTEMAS INTELIGENTES

Departamento de Ciencia de la Computación e Inteligencia Artificial

Universidad de Alicante **Curso 2018/2019**

v1.1

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 hiperplano
- Aplicar Adaboost al problema de clasificación de imágenes
- Realizar un análisis cuantitativo de la tasa de aciertos obtenida mediante este método de aprendizaje

Parte a)

1. Introducción

En la segunda práctica de la asignatura se va a desarrollar un sistema capaz de distinguir entre distintas imágenes. Como ejemplo imaginemos que debemos desarrollar una web que permite la venta de objetos online y deseamos que los usuarios puedan subir fotos de 8 tipos de objetos para poder realizar la venta. Únicamente deben subir una imagen y el sistema tiene que clasificar los objetos entre: *abrigos, bolsos, camisetas, pantalones, suéters, vestidos, zapatillas, zapatos*.

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. 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. El objetivo final sería construir un clasificador que, tras ser entrenado, pueda decirnos a que clase 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 de la base de datos. Se incluyen en total 4000 imágenes de los distintos tipos existentes. 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:

- ¿Qué incluye el fichero Main.java? Especifica que ejemplos básicos incluye y como se podrán utilizar después en la práctica
- Comprueba que DBLoader 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 DBLoader? ¿Cómo se puede acceder a una imagen en concreto? ¿Cómo sabemos a qué clase pertenece una imagen?
- ¿Cómo se muestra una imagen cargada previamente a partir de BDLoader?

1.3 Especificación de las imágenes de entrada

Como ya hemos dicho, las imágenes con las que vamos a trabajar corresponden a 8 tipos: ***abrigos, bolsos, camisetas, pantalones, suéteres, vestidos, zapatillas, zapatos*.** 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.

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.

- Verificar de que tipo y en que rango se mueven los píxeles de las imágenes. Lo necesitaréis a la hora de generar los clasificadores débiles. Podéis normalizarlos si queréis en otro tipo/rango (0..1), (0..255)…

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.

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 ecuación 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

Más concretamente, se deben crear las siguientes funciones (o parecidas) 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. 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.

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

3. Depuración del algoritmo AdaBoost para datos n-dimensionales

No olvidéis 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.

Debemos probar el algoritmo Adaboost para conseguir ajustar sus parámetros de entrada y así obtener el mejor resultado. Empezaremos probando exclusivamente Adaboost para clasificar *n* pantalones con respecto a un conjunto aleatorio de imágenes de otro tipo (también de tamaño *n*, con lo que el tamaño total del conjunto de entrenamiento será 2*n*). Asumir que *n* es número total de pantalones disponibles. Los parámetros a ajustar son:

- Porcentaje de ejemplos utilizados para el test.
- Número máximo de iteraciones de Adaboost
- Número de hiperplanos 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**. Documentar convenientemente las pruebas realizadas y las conclusiones extraídas.

4. Pruebas del algoritmo Adaboost para todas las clases y serialización

Debemos probar el algoritmo Adaboost para conseguir ajustar sus parámetros de entrada y así obtener el mejor resultado. El algoritmo Adaboost se ejecutará 8 veces (una por clase). Se deberá establecer como determinar la fusión de los resultados de los 8 clasificadores fuertes para obtener un único resultado (a que clase 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*) llamará a la clase Adaboost.java (que debéis crear vosotros) y recibirá los siguientes parámetros:

- `1) Adaboost –train <nombre_fichero.cf>`
  De esta manera se entrenará adaboost para todas las clases y posteriormente se almacenarán los clasificadores fuertes encontrados en el fichero indicado. Se deben emitir por pantalla los porcentajes de acierto para el conjunto de entrenamiento y test. El formato de almacenamiento es libre con las siguientes condiciones:
  - Se deben almacenar todos los clasificadores débiles en un único fichero
  - Se debe de poder cargar el fichero en la siguiente función, restaurando en ese caso los 8 clasificadores débiles almacenados en el fichero.

- `2) Adaboost –run <nombre_fichero.cf> <imagen_prueba>`
  Se cargarán los clasificadores fuertes y se ejecutarán sobre la imagen de prueba pasada por parámetro. Se imprimirá por la salida estándar el resultado de la clasificación. *Nota: el proceso de carga y uso de imágenes se utiliza en la plantilla adjunta para cargar la BD, podéis obtenerlo de allí.*

- **Por defecto, entregareis el proyecto para que use vuestro mejor clasificador (formado por los 8 clasificadores fuertes) sobre una imagen de vuestra elección** (opción 2, con los parámetros establecidos como corresponda). **Atención, ¡la práctica se corregirá con dicho clasificador!**

5. Documentación del clasificador

En base al trabajo realizado **se deben investigar varias cuestiones que tenéis que reflejar de manera clara y razonada en la memoria**. La memoria debe reflejar todas las labores y pruebas desarrolladas. De manera general, se deben contestar como mínimo las siguientes cuestiones:

- ¿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 comentas.
- ¿Cómo afecta el número de clasificadores generados y su entrenamiento 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 cuenta los errores de clasificación que obtienes para aprendizaje y test.

- ¿Cómo has conseguido que Adaboost clasifique entre las 8 clases cuando solo tiene una salida binaria?
- ¿Qué clases se confunden más entre sí? Justifícalo mediante las gráficas que consideres.

El apartado a) de la práctica se valorará de 0 a 10 puntos. Se propone al alumnado el uso de la plataforma Colaboratory (https://colab.research.google.com/) para poder obtener hasta dos puntos adicionales. Las tareas a desarrollar se describen en el apartado b:

Parte b)

6. Adaboost en Colaboratory

- Se propone el uso de la plataforma Colaboratory (https://colab.research.google.com/) para el desarrollo de una parte de la práctica. Esta parte es optativa y puede suponer hasta dos puntos adicionales.
- Colaboratory es una herramienta de investigación para la educación y la exploración del aprendizaje automático. Es un entorno de bloc de notas de Jupyter que se puede usar sin configuración. Colaboratory funciona con Python 2.7 y Python 3.6
- Se proporciona una plantilla de Notebook "P2_Adaboost.ipynb" que se debe cargar con la plataforma.
- Se debe realizar una comparación entre Adaboost utilizando umbrales (este código se da ya en la plantilla) con respecto a hiperplano (únicamente se debe implementar el clasificador débil de hiperplanos, al igual que se hizo en el apartado a de la práctica).
- Se deben rellenar distintas secciones de experimentación especificadas en el Notebook. Se recomienda utilizar representaciones gráficas siempre que sea posible.
- Además, se valorará la calidad de la documentación dentro del Notebook

7. 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.
- `/p2a:` Proyecto netbeans de Adaboost. Debe contener dentro del mismo, el mejor clasificador fuerte encontrado. Revisar que cumple los criterios especificados en la sesión 5 de este documento. **No se aceptará código java sin proyecto de netbeans asociado**.
- `/p2b:` Fichero Notebook de colaboratory en formato *.ipynb* con los datos, gráficas, documentación y pruebas reflejados dentro del mismo. Además, se debe incluir una versión en PDF del documento notebook (se puede generar fácilmente imprimiendo el Notebook).

- `/doc:` Documentación, en formato PDF (**no se aceptará otro formato**), contendrá solo la parte a).

8. Evaluación

- Se trata de una práctica individual, con lo que queremos destacar que **NO se puede compartir ningún tipo de código o documentación**. En caso contrario se aplicará la normativa anti copias de las EPS.
- La parte a) tendrá un peso de un 100% en la nota de la práctica. La parte b) permite subir hasta dos puntos sobre la nota obtenida en la parte a) hasta la valoración de un 10.
- La evaluación **se centrará fundamentalmente en la documentación entregada**. Concretamente la documentación tendrá un peso de un 60% con respecto a la implementación (40%).
- **Recomendamos por tanto activamente desarrollar un buen análisis y documentación de la práctica.**

9. Fecha de entrega

23 de diciembre de 2018 por el Moodle de la asignatura