Departamento de Ciencia de la Computación e Inteligencia Artificial
Universidad de Alicante

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 y familiarización con los proyectos de Thonny y Colab

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 entrenar 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 conjunto de entrenamiento con el que vamos a trabajar será 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 qué dígito manuscrito corresponde una imagen.

1.1 Thonny, Colab y Python

Del mismo modo que en la primera práctica, para el desarrollo de esta segunda se utilizará el lenguaje Python y como herramientas de desarrollo tendremos Thonny (https://thonny.org) o Colab de Google (https://colab.research.google.com). Thonny es un IDE (Integrated Development Environment) con los elementos básicos para crear aplicaciones en el lenguaje Python, es posible depurar y observar las variables en ejecución de forma sencilla. Por otro lado, Colab es un entorno de desarrollo que permite programar en Python directamente desde nuestro navegador sin necesidad de instalar ningún complemento ni librería, directamente con nuestra cuenta de GCloud de la UA podremos acceder y crear un nuevo fichero (Jupyter Notebook) para la práctica.

1.2 Inicio del proyecto

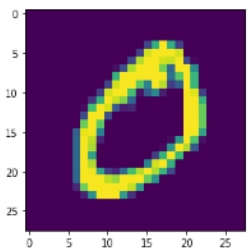
En esta segunda práctica, es importante destacar que a diferencia de la primera, **realizaremos tanto la memoria como la implementación en un solo fichero Notebook de Colab.** En la página moodle de la asignatura encontraréis una carpeta con los ficheros Python necesarios para comenzar la práctica con Thonny y un notebook de Colab. El proyecto Thonny nos servirá como apoyo en caso de ser necesario depurar nuestro código.

La carpeta destinada a Thonny consta de un fichero **main.py** desde donde se ejecutará la práctica, un fichero **adaboost.py** que utilizaremos para implementar el algoritmo AdaBoost, un fichero **clasificador_debil.py** que contiene la implementación de un clasificador débil y su generación y un fichero de utilidades llamado **utils.py** que podremos utilizar para funciones adicionales como mostrar imágenes o construir gráficas. También encontraréis la base de datos MNIST en un fichero llamado **mnist.npz**.

El notebook de Jupyter contiene la misma estructura que el proyecto para Thonny, pero en lugar de tener ficheros independientes tenemos funciones y la base de datos se descarga de forma dinámica.

1.3 Especificación de las imágenes de entrada

Como ya hemos comentado, 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.



Tareas a realizar en esta primera sesión:

- Familiarizarse con los entornos de Thonny y Colab. Ejecutar el código y observar los resultados.
- Familiarizarse con la base de datos MNIST y entender su finalidad. Puedes echar un vistazo a la página oficial (http://yann.lecun.com/exdb/mnist/).
- Comprobar la estructura de las tuplas **mnist_X** y **mnist_Y** para entender cómo se están cargando las imágenes y sus correspondientes etiquetas.
- Probar mostrar en pantalla diferentes imágenes del array y sus etiquetas.

Sesión 2: Algoritmo Adaboost. Clasificadores débiles

Dentro de los algoritmos de aprendizaje automático, el término boosting hace referencia a un tipo de algoritmos que tratan de encontrar una hipótesis fuerte a partir de un conjunto de hipótesis simples y débiles. En concreto, AdaBoost, proviene de la contracción en inglés de Adaptive Boosting, su objetivo es concretar un clasificador fuerte como combinación de varios clasificadores débiles. Lo que le proporciona la característica de adaptativo es porque este algoritmo propone entrenar un conjunto de clasificadores débiles de manera iterativa, de modo que cada nuevo clasificador débil se enfoca en los datos que fueron clasificados erróneamente, así el algoritmo se va adaptando en cada paso y, de esta manera, consigue un mejor resultado.

Debemos tener en cuenta que el algoritmo AdaBoost con el cual vamos a trabajar es un algoritmo binario, es decir va a trabajar con dos tipos de datos los cuales representamos como +1 o -1 indicando si “pertenece a la clase” o “no pertenece a la clase”.

De momento nos vamos a centrar en una única clase para ver cómo funciona AdaBoost. A continuación, tenéis disponible el pseudocódigo de este algoritmo, donde podemos ver que la entrada **X** representa el vector de imágenes de entrenamiento e **Y** las etiquetas correspondientes a cada imagen:

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 = generaAlAzar()
        $\epsilon_t = P_{D_t}(h_t(x_i) \neq y_i) \rightarrow \epsilon_{t,k} = \sum_{i=1}^{N} D_t(i) \cdot (F_k(x) \neq y(x))$
      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 $H(x) = sign(\sum_t \alpha_t \cdot h_t(x))$

2.1 Clasificadores débiles

Como se ha comentado anteriormente, AdaBoost trata de encontrar un clasificador fuerte eficiente a partir de un conjunto de clasificadores débiles simples. 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, de manera que los valores que queden a un lado del umbral se clasificarán de un clase y los que queden al otro lado del umbral de la otra clase.

En este apartado nos vamos a centrar en la parte del código del algoritmo que calcula y aplica estos clasificadores débiles.

6: **for** $k = 1 \rightarrow A$ **do** ▷ A = num. de pruebas aleatorias
7: $F_p = \text{generaAlAzar}()$
8: $\epsilon_t = P_{D_t}(h_t(x_i) \neq y_i) \rightarrow \epsilon_{t,k} = \sum_{i=1}^{N} D_t(i) \cdot (F_k(x) \neq y(x))$
return $< F_p | \min(\epsilon_{t,k}) >$

Nuestros clasificadores débiles van a tener en cuenta un único píxel de la imagen y basándose en ese píxel y un umbral deben clasificar la imagen como si “pertenece a la clase” o “no pertenece a la clase”. Así, nuestros clasificadores débiles van a estar compuestos por:

- Un **píxel**, que nos dirá qué píxel es el que vamos a utilizar para clasificar.
- Un **umbral**, que nos dirá qué valor numérico es el que vamos a utilizar para clasificar.
- Un **dirección**, que nos dirá si debemos comprobar si el umbral es mayor o menor.

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.

Como podéis ver en el pseudocódigo del algoritmo, estas funciones se utilizan dentro de un bucle **for**. Esto es así porque la idea es generar aleatoriamente **A** clasificadores débiles, clasificar el conjunto de aprendizaje y calcular el error de ese clasificador débil. Y, finalmente, nos quedaremos con el clasificador débil que mejor clasifique, es decir, aquel que obtenga la menor tasa de error para el conjunto dado. Así que uno de los parámetros que tendremos que ajustar es el número de clasificadores débiles que se van a generar aleatoriamente cada vez que creemos un clasificador débil, es decir ese valor **A** que determina el número de veces que se ejecuta el bucle. 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:

- `generar_clasificador_debil(dimensión_de_los_datos)`
  genera y devuelve un clasificador débil, dentro del espacio de búsqueda, al azar. Requiere saber la dimensión de los objetos con los que se trabajará, en nuestro caso estamos trabajando con una dimensión de 784 píxeles. Los clasificadores débiles los podemos representar como una tupla o un vector de tres componentes, por ejemplo, podríamos tener:

  (pixel, umbral, dirección) = (589, 100, -1)

  [pixel, umbral, dirección] = [402, 23, 1]

- `aplicar_clasificador_debil(clasificadorDebil, imagen)`
  aplica un clasificador débil a una imagen y devuelve un valor booleano con el resultado de la clasificación. En esta función recibiremos el clasificador débil en el formato mencionado anteriormente y el vector de componentes de la imagen. Se utilizará para aplicar el clasificador fuerte después de haberlo aprendido con Adaboost:

  aplicar_clasificador_debil([589, 100, -1], [134, 3, 245, …, 34])

- `obtener_error(clasificador, X, Y, D)`
  obtiene el error de clasificación a partir del vector D (ver siguiente apartado) y la clasificación correcta de los datos. El resultado de esta función es el error calculado del clasificador dado. Nuevamente, utilizaremos un clasificador débil como entrada a esta función además de un vector con las imágenes **X**, las etiquetas correspondientes **Y** y el vector **D**. La entrada podría tener la siguiente forma:

  obtener_error([589, 100, -1], [[134, …, 3], [32, …, 34]], [4.23, …, 0.23])

Tareas sesión 2:

- Implementar las tres funciones necesarias para trabajar con los clasificadores débiles.
- Crear el bucle que genera A clasificadores débiles y escoge el que tenga menor tasa de error.

Sesión 3: Algoritmo Adaboost. Clasificador fuerte para el dígito 0 (Hito 1)

Una vez ya tenemos implementado la generación de los conjuntos de datos de entrenamiento **X** e **Y** y del código necesario para generar clasificadores débiles continuaremos con la implementación del algoritmo AdaBoost para poder clasificar una única clase, la clase 0.

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.

Además, deberemos realizar las primeras pruebas para determinar los valores adecuados de **A** y **T**, que determinan el número de veces que se ejecuta cada uno de los bucles. En concreto **A** determina cuántos clasificadores débiles se van a generar al azar para quedar con el que menor tasa de error tenga y T determina cuántos clasificadores débiles van a formar el clasificador fuerte. **Estas pruebas deben quedar reflejadas en la documentación mediante diferentes gráficas de modo que justifiquen las decisiones tomadas para optimizar los resultados del clasificador**, por ejemplo, podríamos tener una gráfica que correlacione los valores de **A** y de **T** con la tasa de aciertos del clasificador o bien una gráfica en la que se pueda observar cómo cambia la precisión respecto al número de imágenes que son utilizadas para entrenamiento y test.

Tareas sesión 3:

- Implementar completamente el algoritmo AdaBoost para que sea capaz de clasificar la categoría 0.
- Analizar los resultados obtenidos ante diferentes valores de A y T.

Sesión 4: Algoritmo Adaboost. Clasificador fuerte de todas las clases

Una vez ya tenemos el algoritmo AdaBoost capaz de generar un clasificador fuerte para una clase, debemos hacer que nuestra práctica sea capaz de distinguir entre **10 clases diferentes**. Para esto debemos generar un clasificador fuerte para cada clase, es decir, deberemos llamar a AdaBoost para cada una de nuestras clases. Al final, obtendremos **10 clasificadores fuertes**, uno para cada clase.

Cuando tengamos todos los clasificadores fuertes, podremos comprobar la tasa de acierto de estos clasificadores tanto para los conjuntos de entrenamiento como de test. **Los resultados deben ser mostrados en el Notebook.**

Tareas sesión 4:

- Implementar completamente el algoritmo AdaBoost para que sea capaz de clasificar todas las categorías.

Sesión 5 y 6: Documentación y pruebas. Hito final

Para este hito final debemos tener realizadas diferentes pruebas (especial atención a las gráficas) y terminar la documentación que se deberá haber ido elaborando de manera continua durante todo el desarrollo de la práctica.

La documentación **es la parte más importante de la práctica (60%)** y en ella se debe realizar un análisis minucioso de la misma reflejando de manera clara y razonada, al menos, los siguientes puntos:

- Explica brevemente cómo has adaptado la base datos MNIST al algoritmo AdaBoost.
- Comenta detalladamente el funcionamiento de AdaBoost teniendo en cuenta que tasa media de fallos obtienes para aprendizaje y test. Correlaciona los porcentajes de acierto y de fallo con los valores de A y de T.
- ¿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 qué 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.
- ¿Cómo has conseguido que Adaboost clasifique entre los 10 dígitos cuando solo tiene una salida binaria?

Entrega de la práctica

La fecha límite de entrega de la práctica es el **23 de Diciembre de 2021 a las 23:55h**. La entrega se realizará a través de Moodle.

Formato para la entrega final

La entrega debe consistir en un fichero comprimido zip con:

- Fichero Notebook de Colab exportado en formato **ipynb** (“Archivo->Descargar->Descargar .ipynb”)
- Fichero Notebook de Colab exportado en **pdf**.

¡¡¡AVISO IMPORTANTE!!!

No cumplir cualquiera de las normas de formato/entrega puede suponer un suspenso en la práctica.

Recordad que **las prácticas son INDIVIDUALES** y NO se pueden hacer en parejas o grupos.

Cualquier código copiado supondrá un suspenso de la práctica para todas las personas implicadas en la copia y, como indica el Reglamento para la Evaluación de Aprendizajes de la Universidad de Alicante (BOUA 9/12/2015) y el documento de Actuación ante copia en pruebas de evaluación de la EPS, **se informará a la dirección de la Escuela Politécnica Superior para la toma de medidas oportunas**.

Plan de entregas por hitos

Durante el periodo de elaboración de la práctica se realizarán dos hitos de entrega. Es obligatorio cumplir las fechas de las entregas correspondientes:

- Hito 1: Implementación del algoritmo AdaBoost para una clase.
    - **Fecha: 5 de diciembre**
    - **La no entrega del hito 1 en la fecha prevista supone una penalización del 20%.**
- Hito 2 (entrega final): práctica completa y documentación.
    - **Fecha: 23 de diciembre**

La nota de la práctica sufrirá una penalización de dos puntos si no se cumple rigurosamente con los requisitos de la entrega (tanto en la estructura de los ficheros entregados como en la salida que debe generar la práctica).