

## Práctica 2: Visión artificial y aprendizaje

(Estas transparencias son sólo un resumen. Se completan y detallan en el enunciado de la práctica)

- Sesión 1: Introducción al problema a resolver y el entorno de trabajo.
- Sesión 2: Adaboost binario con clasificadores de umbral.
  - Tarea 1A (OBLIGATORIA): Implementación de Adaboost y DecisionStump.
  - Tarea 1B (OBLIGATORIA): Resultados del clasificador Adaboost.
- Sesión 3: Ajuste de parámetros T y A. Clasificador multiclase.
  - Tarea 1C (OBLIGATORIA): Ajuste óptimo de T y A.
  - Tarea 1D (OBLIGATORIA): Clasificador Adaboost multiclase.
  - Tarea 1E (OPTATIVA): Mejoras Creativas.
- Sesión 4: Clasificador multiclase Adaboost con Scikit-learn.
  - Tarea 2A (OBLIGATORIA): Clasificador Adaboost con scikit-learn.
  - Tarea 2B (OBLIGATORIA): Comparativa con implementación propia.
  - Tarea 2C (OPTATIVA): Sustituye el clasificador por árboles de decisión
- Sesión 5: Clasificador multiclase con Keras.
  - Tarea 2D (OBLIGATORIA): Clasificador MLP con Keras.
  - Tarea 2E (OPTATIVA): Clasificador CNN con Keras.
- Sesión 6: Comparativa final.
  - Tarea 2F (OBLIGATORIA): Comparativa de resultados.
- Sesión 7: Resolución de dudas y revisión.

### Sesión 1:

#### Introducción al problema a resolver y el entorno de trabajo.

- Técnicas de aprendizaje supervisado aplicadas al reconocimiento automático de números manuscritos, en nuestro caso las cifras del 0 al 9 de la base de datos MNIST (<http://yann.lecun.com/exdb/mnist/>).
- En la primera parte desarrollaremos en Python, programado desde cero, un clasificador multiclase adaboost para reconocer las cifras del 0 al 9.
- En la segunda parte utilizaremos la librería scikit-learn (<https://scikit-learn.org/>) para construir un clasificador multiclase adaboost para problema de la primera parte, y un multi-layer perceptron (MLP) para resolver el mismo problema utilizando la librería Keras (<https://keras.io/>) de TensorFlow.
- Librerías: Scikit-learn, Keras, Numpy (<https://numpy.org/>), Pandas (<https://pandas.pydata.org/>), Matplotlib (<https://matplotlib.org/>).

**Importante:** leer detenidamente en cada sesión el enunciado en la parte que le corresponde.

## Sesión 1:

- Esta práctica tiene dos hitos de entrega, ambos obligatorios.
- Hito 1 (primera parte de la práctica): hasta el 3 de diciembre de 2023 a las 23:55h.
- Hito 2 (entrega final): hasta el 24 de diciembre de 2023 a las 23:55h.

Deberás crear un módulo (una función) en tu archivo Python para cada tarea de la práctica, que llamará a todas las funciones implicadas en completar la tarea. Estos módulos se podrán llamar individualmente en la última sección de tu archivo Python, utilizando el condicional: `__name__ == "__main__"`:

```
def main():
    # otras..
    rend_1A = tareas_1A_y_1B_adaboost_binario(clase=5, T=10, A=10, verbose=True)
    # otras posteriors..

if __name__ == "__main__":
    main()
```

## Sesión 1:

Para **cargar la base de datos de MNIST** y familiarizarnos con la manipulación de esas imágenes:

```
from tensorflow import keras
import logging, os
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
```

# Ahora podemos cargar las imágenes de MNIST y ver en qué consisten:

```
(X_train, Y_train), (X_test, Y_test) = keras.datasets.mnist.load_data()
print(X_train.shape, X_train.dtype)
print(Y_train.shape, Y_train.dtype)
print(X_test.shape, X_test.dtype)
print(Y_test.shape, Y_test.dtype)
```

X\_train es un array de 60000 elementos que a su vez son arrays de 28x28 uint8 (unsigned int de 8 bits, números de 0 a 255), que son las imágenes de números manuscritos de 28x28 píxeles en escala de 256 grises.

Y\_train es otro array de 60000 elementos también uint8.

Y los arrays X\_test e Y\_test son iguales pero contienen 10000 observaciones cada uno.

Sesión 1:  
Podemos **mostrar en pantalla las imágenes** que hay en `X_train` e `X_test` utilizando Matplotlib:

```
import matplotlib.pyplot as plt
import numpy as np

def show_image(imagen, title):
    plt.figure()
    plt.suptitle(title)
    plt.imshow(imagen)
    plt.show()

for i in range(3):
    title = "Mostrando imagen X_train[" + str(i) + "]"
    title_prefix = title + " -- Y_train[" + str(i) + "]" + " = " + str(Y_train[i])
    show_image(X_train[i], title_prefix)
```

Sesión 1:  
Con **Numpy**, podemos acceder a un pixel (fila, columna) de todas las imágenes simultáneamente, contar cuántos valores distintos tiene, cambiar sus valores o pintar una gráfica con esos valores

```
def plot_X(X, title, fila, columna):
    plt.title(title)
    plt.plot(X)
    plt.xscale('linear')
    plt.yscale('linear')
    plt.show()
    #para mostrar el uso de plato

fila=3
columna=5

"""Extraes todos los valores de un píxel específico (ubicado en
la fila fila y columna columna) de todas las imágenes
del conjunto de entrenamiento X_train:"""
features_fila_col = X_train[:, fila, columna]
```

## Sesión 1:

```

""" Luego, calculas cuántos valores únicos existen
para ese píxel en particular a lo largo de todas las
imágenes. """
print(len(np.unique(features_fila_col)))

```

```

title = "Valores en (" + str(fila) + ", " + str(columna) + ")"
plot_X(features_fila_col, title, fila, columna)

```

# Esa gráfica no parece muy informativa. Podemos ordenar primero los valores, y usar escala logarítmica si queremos tener más resolución en el extremo izquierdo de la gráfica:

```
def plot_2D(X, title, xscale='linear'):
```

```

    plt.title(title)
    plt.plot(X)
    plt.xscale(xscale)
    plt.show()

```

```

features_sorted = np.sort(features_fila_col)
plot_2D(features_sorted, title, "log")

```

```

features_reversed = features_sorted[::-1]
plot_2D(features_reversed, title, "log")

```

## Sesión 1:

### Ejercicios de la sesión:

1. Leer detenidamente y probar todas las instrucciones de la Sesión 1.
2. Cargar datos de MNIST.
3. Visualizar datos de MNIST y su estructura y propiedades.
4. Resolver:
  - a. Contar la cantidad de seises en el conjunto de entrenamiento.
  - b. La cantidad de píxeles que no valen más de un cierto valor Z.

## Sesión 2: Adaboost binario con clasificadores de umbral

- Tarea 1A (OBLIGATORIA): implementa las clases Adaboost y DecisionStump
- Tarea 1B (OBLIGATORIA): Mostrar resultados de tu clasificador Adaboost

**Sigue las recomendaciones escritas en el enunciado de la práctica.**

## Sesión 2:

```
def load_MNIST_for_adaboost():  
  
    # Cargar los datos de entrenamiento y test tal y como nos los sirve keras (MNIST de Yann  
    Lecun)  
    (X_train, Y_train), (X_test, Y_test) = keras.datasets.mnist.load_data()  
  
    # Formatear imágenes a vectores de floats y normalizar  
    X_train = X_train.reshape((X_train.shape[0], 28*28)).astype("float32") / 255.0  
    X_test = X_test.reshape((X_test.shape[0], 28*28)).astype("float32") / 255.0  
    #X_train = X_train.astype("float32") / 255.0  
    #X_test = X_test.astype("float32") / 255.0  
  
    # Formatear las clases a enteros con signo para aceptar clase -1  
    Y_train = Y_train.astype("int8")  
    Y_test = Y_test.astype("int8")  
  
    return X_train, Y_train, X_test, Y_test
```

## Sesión 2:

### Algorithm 1 Adaboost

```

1: procedure ADABOOST( $X, Y$ )
2:    $D_1(i) = 1/N$   $\triangleright$  Indica como de difícil es de clasificar cada punto  $i$ 
3:   for  $t = 1 \rightarrow T$  do  $\triangleright T$  es el número de clasificadores débiles a usar
4:     Entrenar  $h_t$  teniendo en cuenta  $D_t$ 
5:     Start
6:       for  $k = 1 \rightarrow A$  do  $\triangleright A = \text{num. de pruebas aleatorias}$ 
7:          $F_p = \text{generaClasificadorDébilAlAzar}()$ 
8:          $\epsilon_t = P_{D_t}(h_t(x_i) \neq y_i) \rightarrow \epsilon_{t,k} = \sum_{i=1}^N D_t(i) \cdot (F_k(x) \neq y(x))$ 
9:         return  $< F_p | \min(\epsilon_{t,k}) >$ 
10:      End
11:      Del  $h_t$  anterior obtener su valor de confianza  $\alpha_t \in \mathbb{R}$ 
12:      Start
13:         $\alpha_t = 1/2 \log_2 \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$ 
14:      End
15:      Actualizar  $D_{t+1}$ 
16:      Start
17:         $D_{t+1} = \frac{D_t(i) \cdot e^{-\alpha_t \cdot y_i \cdot h_t(x_i)}}{Z_t}$ 
18:         $Z_t = \sum_i D_t(i)$ 
19:      End
20:      return  $H(x) = \text{sign}(\sum_t \alpha_t \cdot h_t(x))$ 

```

## Sesión 2:

### Programar una clase para nuestros clasificadores débiles

```

class Decisionstump:
    ## Constructor de clase, con número de características
    def __init__(self, n_features):
        # Seleccionar al azar una característica, un umbral y una polaridad.
        return self
    ## Método para obtener una predicción con el clasificador débil
    def predict(self, X):
        # Si la característica que comprueba este clasificador es mayor que el umbral y la polaridad es 1
        # o si es menor que el umbral y la polaridad es -1, devolver 1 (pertenece a la clase)
        # Si no, devolver -1 (no pertenece a la clase)

```

## Sesión 2:

La clase Adaboost y su constructor, su método de entrenamiento (enseñar al clasificador a clasificar correctamente) y su método de predicción (obtener el resultado de esa clasificación):

```

class Adaboost:
    ## Constructor de clase, con número de clasificadores e intentos por clasificador
    def __init__(self, T=5, A=20):
        # Dar valores a los parámetros del clasificador e iniciar la lista de clasificadores débiles vacía
        return self

    ## Método para entrenar un clasificador fuerte a partir de clasificadores débiles mediante Adaboost
    def fit(self, X, Y, verbose = False):
        # Obtener el número de observaciones y de características por observación de X
        # Iniciar pesos de las observaciones a 1/n_observaciones
        # Bucle de entrenamiento Adaboost: desde 1 hasta T repetir
        # Bucle de búsqueda de un buen clasificador débil: desde 1 hasta A repetir
        # Crear un nuevo clasificador débil aleatorio
        # Calcular predicciones de ese clasificador para todas las observaciones
        # Calcular el error: comparar predicciones con los valores deseados
        # y acumular los pesos de las observaciones mal clasificadas
        # Actualizar mejor clasificador hasta el momento: el que tenga menor error
        # Calcular el valor de alfa y las predicciones del mejor clasificador débil
        # Actualizar pesos de las observaciones en función de las predicciones, los valores deseados y alfa
        # Normalizar a 1 los pesos
        # Guardar el clasificador en la lista de clasificadores de Adaboost

    ## Método para obtener una predicción con el clasificador fuerte Adaboost
    def predict(self, X):
        # Calcular las predicciones de cada clasificador débil para cada input multiplicadas por su alfa
        # Sumar para cada input todas las predicciones ponderadas y decidir la clase en función del signo
  
```

## Sesión 2:

Crea en tu archivo una función con los parámetros clase (uno de los 10 dígitos), T y A, que cargue los datos de MNIST, entrene tu clasificador adaboost con el conjunto de entrenamiento X\_train, e imprima por pantalla las tasas de acierto (porcentaje de predicciones que coinciden con las correctas) con X\_train y con X\_test.

```
user@user:~/GInf-SI/Prácticas/P2$ python3.8 l_m_b.py
```

Entrenando clasificador Adaboost para el dígito 9, T=20, A=10

Entrenando clasificadores de umbral (con dimensión, umbral, dirección y error):

Añadido clasificador 1: 59, 0.4354, +1, 0.099150

Añadido clasificador 2: 274, 0.2253, -1, 0.456836

Añadido clasificador 3: 524, 0.2999, -1, 0.417796

...

Añadido clasificador 18: 632, 0.0881, -1, 0.444582

Añadido clasificador 19: 595, 0.4053, -1, 0.445673

Añadido clasificador 20: 140, 0.7672, +1, 0.422934

Tasas acierto (train, test) y tiempo: 89.88%, 89.73%, 0.089 s.

### Sesión 3: Ajuste de los parámetros de entrenamiento y clasificador multiclase a partir de clasificadores binarios

#### Tarea 1C (OBLIGATORIA): **Ajuste óptimo de T y A**

Utilizando Matplotlib, genera gráficas de curvas que permitan relacionar el tiempo de entrenamiento con la tasa de acierto para distintos valores de T y A...

#### Tarea 1D (OBLIGATORIA): **Clasificador multiclase**

En esta tarea te vas a apoyar en el clasificador binario que has implementado para componer un clasificador multiclase.

#### Tarea 1E (OPTATIVA): **Mejoras creativas**

La última tarea de esta parte de la práctica es un trabajo relativamente libre: mejora tu método adaboost para conseguir entrenamientos más eficaces (mejores tasas de acierto) y más rápidos.

**Sigue las recomendaciones escritas en el enunciado de la práctica.**

### Sesión 4: Clasificador multiclase adaboost usando scikit-learn

#### Tarea 2A (OBLIGATORIA): **Modela el clasificador adaboost con scikit-learn**

Documentarte en la web de scikit-learn sobre cómo utilizar la clase AdaBoostClassifier, e implementar una función en tu fichero de entrega para resolver el mismo problema que en la Tarea 1D, el clasificador multiclase.

#### Tarea 2B (OBLIGATORIA): **Compara tu versión de adaboost con la de scikit-learn y optimiza la configuración del clasificador débil por defecto**

#### Tarea 2C (OPTATIVA): **Sustituye el clasificador por árboles de decisión**

**Sigue las recomendaciones escritas en el enunciado de la práctica.**



## Sesión 5: Clasificador multiclase con redes neuronales usando Keras

### Tarea 2D (OBLIGATORIA): **Modela un clasificador MLP para MNIST con Keras**

En esta tarea vas a implementar un MLP usando la librería Keras. Puedes documentarte en la web de Keras sobre cómo programar en Python con muy pocas líneas de código un perceptrón multicapa.

### Tarea 2E (OPTATIVA): **Modela un clasificador mediante CNN para MNIST con Keras**

Para resolverla necesitarás documentarte sobre las CNN y buscar ejemplos de implementación utilizando Keras.

**Sigue las recomendaciones escritas en el enunciado de la práctica.**

## Sesión 6: Comparativa de técnicas

Tarea 2F (OBLIGATORIA): Realiza una comparativa de los modelos implementados

**Sigue las recomendaciones escritas en el enunciado de la práctica.**

## Sesión 7: Resolución de dudas