*Bruno Sancho Deltell - 20519135F*

Universidad de Alicante Convocatoria C2

# Práctica 2: Visión artificial y aprendizaje

## *ÍNDICE*

## *Tarea 1A (OBLIGATORIA): implementa las clases Adaboost y DecisionStump:*

Aclaración:

> *El Adaboost se ha implementado siguiendo las directrices marcadas por "Departamento de Ciencia* de *la Computación e Inteligencia Artificial - UA"*

La **clase Adaboost** tiene los siguientes atributos:

> **T**: número de clasificadores débiles que se generan cada vez (para luego coger el mejor)
> **A**: número de veces que se generan esos T clasificadores débiles
> **Classifiers**: lista que contendrá los mejores clasificadores débiles seleccionados
> **Accuracy**: contendrá la precisión del clasificador fuerte una vez este sea probado con X_test

Y los siguientes **métodos**:

> **set_accuracy**: es el setter de accuracy al que se llamará tras probar el adaboost
> **fit:** es la función de entrenamiento. Se encarga de iterar sobre T y A realizando llamadas a "DecisionStump" que representa un clasificador débil. En la función fit, los valores de alpha, error y w (pesos) se calculan tal y como se detalla en la teoría de la asignatura (no lo repetiré para no extenderme).
> **predict:** itera sobre los mejores clasificadores débiles seleccionados (classifiers) y devuelve (en el caso de clasificar binario, sign = True) el signo de la suma de lo devuelto por cada clasificador multiplicado por su w.

La **clase DecisionStump** tiene los siguientes atributos:

> **feature_index**: representa el índice de la característica (píxeles) seleccionada al azar.
> **threshold:** Representa el umbral seleccionado al azar. Se utilizará para comparar con el valor de la característica correspondiente.
> **polarity:** Representa la polaridad de la decisión. Puede ser 1 o -1. Si la polaridad es 1, significa que si la característica es mayor que el umbral, se clasifica como 1; si la polaridad es -1, significa que si la característica es menor que el umbral, se clasifica como 1.

Y el siguiente **método:**

> **predict**: coge un conjunto de datos (X) y realiza predicciones en base a la polaridad y el umbral (tal y como se detalla en la teoría, y también en comentarios en el código)

## *Tarea 1B (OBLIGATORIA): Mostrar resultados de tu clasificador adaboost:*

Con la el parámetro Verbose = True la salida sería tal que así:

Entrenando clasificador Adaboost para el dígito 3, T=50, A=50
Entrenando clasificadores de umbral (con dimensión, umbral, dirección y error):
Mejor clasificador 0: 178, 0.8059, 1, 0.343663
Mejor clasificador 1: 632, 0.1066, 1, 0.367559
Mejor clasificador 2: 351, 0.9456, 1, 0.339842
…
Mejor clasificador 47: 591, 0.3793, 1, 0.483811
Mejor clasificador 48: 335, 0.2310, -1, 0.479444
Mejor clasificador 49: 173, 0.7471, 1, 0.484694
Tiempo de entrenamiento: 0.2861020565032959 segundos
Tasas acierto (train, test): 81.1939%, 81.4700%

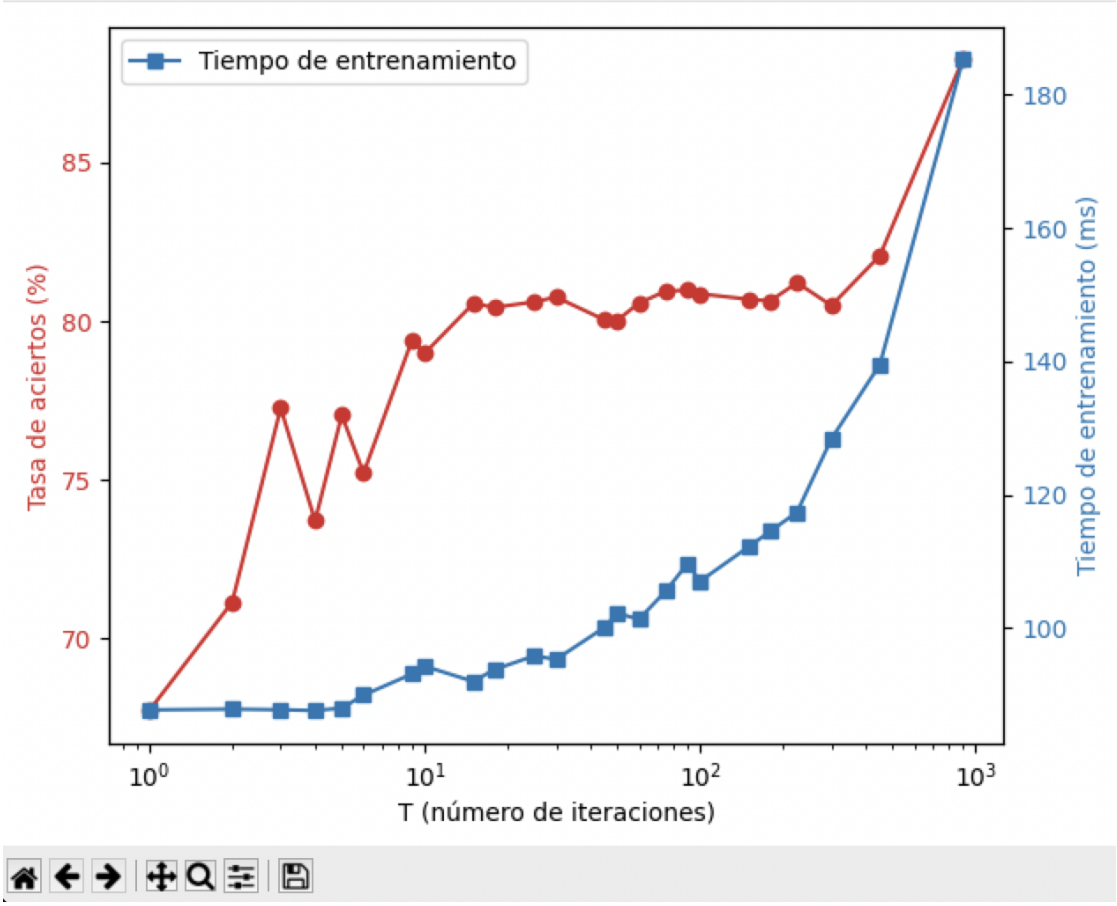
Lo único remarcable en esta tarea es la forma de cargar los datos de entrenamiento (X_train). Estos se cargan de manera balanceada, es decir, 50% pertenecen a la clase a entrenar (3 en este caso) y el otro 50% estaría conformado por el resto de clases de manera aleatoria. Esto con el fin de que el entrenamiento sea lo más fidedigno posible.
La tasa de acierto (accuracy) se calculará con datos aleatorios X_test.

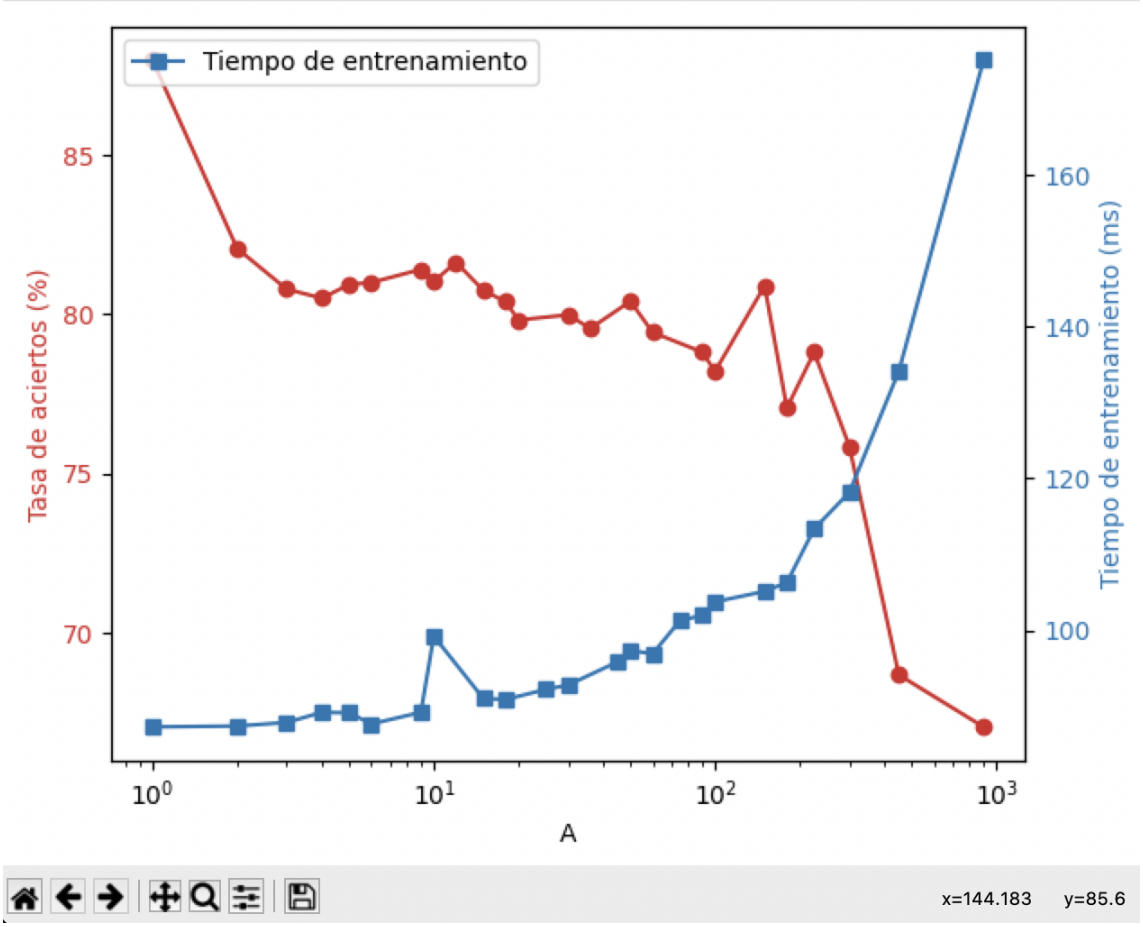
## *Tarea 1C (OBLIGATORIA): Ajuste óptimo de T y A:*

Para el cálculo de los valores óptimos de T y A se ha definido el vector TA, que contiene todos los números enteros cuyo producto es 900.
Para ayudar a determinar estos valores se han creado gráficas que comparan tiempo y precisión para cada combinación de T y A.
Teniendo en cuenta la aleatoriedad a la hora de entrenar estos clasificadores y con el fin de suavizar las gráficas, se han entrenado y ponderado los resultados de 10 clasificadores para cada par T,A.

Ejemplo:

Entrenando clasificadores Adaboost para el dígito 3, T=1, A=900
Tasa de aciertos (test, tiempo): 60.29% y 88.9899730682373 ms
Tasa de aciertos (test, tiempo): 70.49% y 87.07618713378906 ms
Tasa de aciertos (test, tiempo): 68.82000000000001% y 88.31095695495605 ms
Tasa de aciertos (test, tiempo): 66.49000000000001% y 87.7678394317627 ms
Tasa de aciertos (test, tiempo): 66.75999999999999% y 85.20221710205078 ms
Tasa de aciertos (test, tiempo): 78.36% y 84.21516418457031 ms
Tasa de aciertos (test, tiempo): 60.5% y 91.11928939819336 ms
Tasa de aciertos (test, tiempo): 71.2% y 87.99195289611816 ms
Tasa de aciertos (test, tiempo): 68.44% y 90.10195732116699 ms
Tasa de aciertos (test, tiempo): 65.9% y 86.53497695922852 ms


Tiempo de entrenamiento
Tasa de aciertos (%)
Tiempo de entrenamiento (ms)
T (número de iteraciones)
85
80
75
70
180
160
140
120
100
$10^0$
$10^1$
$10^2$
$10^3$


Tiempo de entrenamiento
Tasa de aciertos (%)
Tiempo de entrenamiento (ms)
A
85
80
75
70
160
140
120
100
$10^0$
$10^1$
$10^2$
$10^3$
x=144.183
y=85.6

Siendo T y A:
TA = [(1,900),(2,450),(3,300), (4,225), (5,180), (6,150), (9,100), (10,90), (15,60), (18,50), (25,36), (30,30), (45,20), (50,18), (60,15), (75,12), (90,10), (100,9), (150,6), (180,5), (225,4), (300,3), (450,2), (900,1)]

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

La **clase AdaboostMulticlass** tiene los mismos atributos que la clase Adaboost, la diferencia radica en que la variable classifiers está compuesta por los 10 clasificadores fuertes.

Los **métodos** de esta clase son los siguientes:

**fit:** está se encarga de generar 10 clasificadores binarios (entrenados cada uno con sus respectivos datos balanceados) y almacenarlos en la lista classifiers.

**predict_1_image:** itera sobre classifiers llamando a la función predict (sign = False) y coge el mayor índice de los valores devueltos por cada clasificador.

**calculate_accuracy:** hace lo mismo que el método anterior pero para todo X_test. De esta manera puede calcular cuan preciso es el Adaboost MultiClase generado.

Ejemplo de ejecución(llamando a fitm después a calculate_accuracy y por último a predict_1_image):

Classifier 0 has an accuracy of: 87.32061455343576
Classifier 1 has an accuracy of: 87.27380599228715
Classifier 2 has an accuracy of: 82.48573346760658
Classifier 3 has an accuracy of: 81.60985157396836
Classifier 4 has an accuracy of: 79.52755905511812
Classifier 5 has an accuracy of: 76.52647113078768
Classifier 6 has an accuracy of: 85.6285907401149
Classifier 7 has an accuracy of: 84.75658419792498
Classifier 8 has an accuracy of: 83.60109383011451
Classifier 9 has an accuracy of: 78.87880316019499
Porcentaje final de acierto: 73.1900%
Recuento de imágenes acertadas por clase:
Clase 0: 810.0 imágenes acertadas
Clase 1: 1065.0 imágenes acertadas
Clase 2: 657.0 imágenes acertadas
Clase 3: 693.0 imágenes acertadas
Clase 4: 719.0 imágenes acertadas
Clase 5: 508.0 imágenes acertadas
Clase 6: 689.0 imágenes acertadas
Clase 7: 771.0 imágenes acertadas
Clase 8: 647.0 imágenes acertadas
Clase 9: 599.0 imágenes acertadas
#Muestra la Imagen
La predicción del Multiclase para la imagen es: 1

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

La idea consiste en que un clasificador débil no pueda observar una característica (píxel) que tenga el mismo valor en la gran mayoría de las imágenes (99%,98%...). Esto debido a que la ganancia de información sobre la imagen que estas características aportan es prácticamente nula.

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

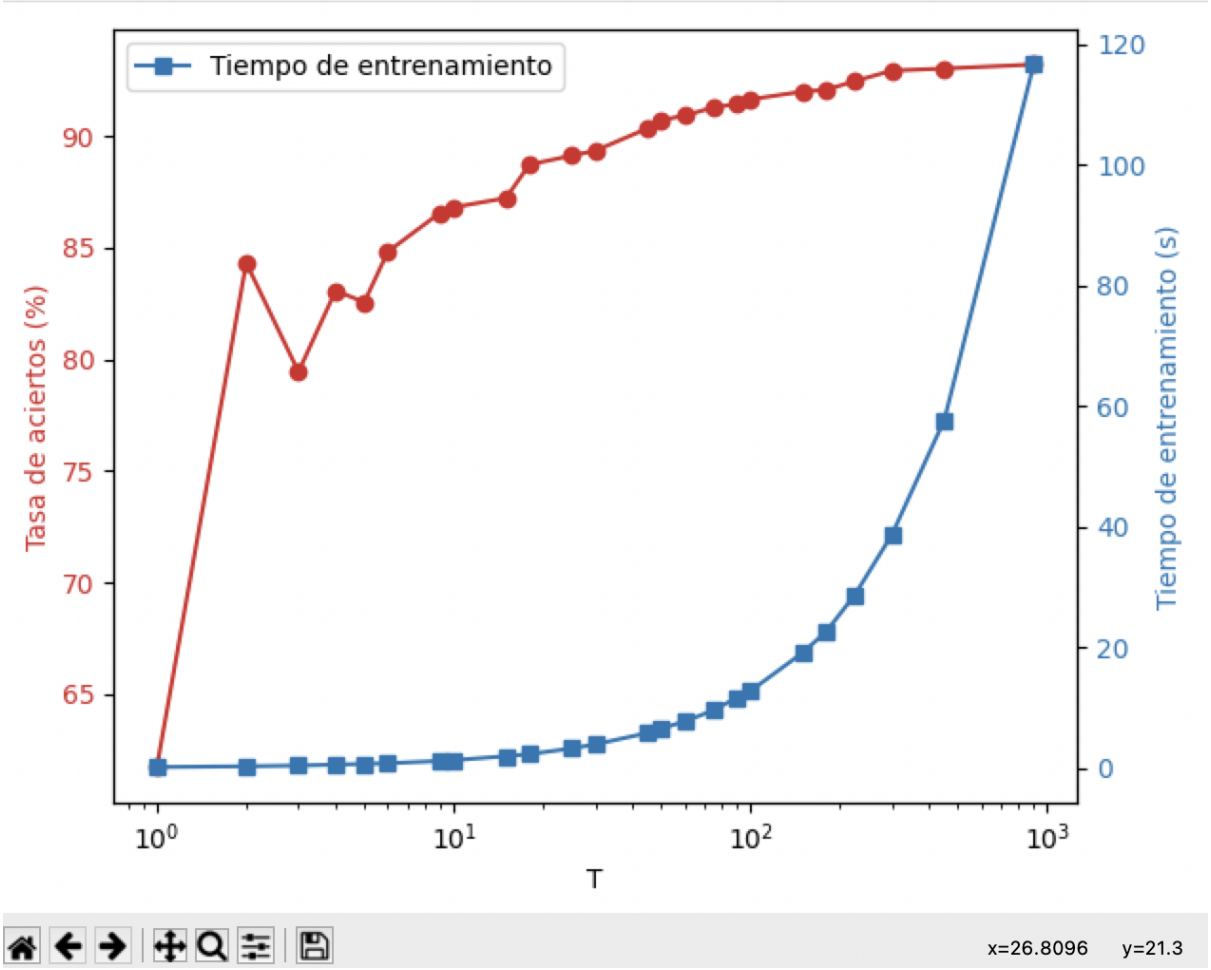
Esta tarea es muy simple, tan solo se crea una instancia de "AdaBoostClassifier" (sin ningún parámetro) de la librería scikit-learn y tras ello se entrena y prueba con datos cargados de forma aleatoria.

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

Para empezar había que determinar la equivalencia entre nuestros T y A, y los parámetros de scikit-learn equivalentes.
La equivalencia de T es **n_estimators,** que representa el número de clasificadores débiles (por defecto, árboles de decisión) que se estrenarán en el proceso de boosting.
Para A **no existe equivalencia directa**, he indagado en la página "https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier" y lo más parecido podría ser **max_leaf_nodes,** pero este está más relacionado con la creación de árboles menos profundos que puede ser útil para prevenir el sobreentrenamiento.

Por ello, para la experimentación se ha decidido probar modificando solo T:

Ejemplo de ejecución:
Entrenando clasificador Adaboost de scikit-learn para el dígito 3, T=1
Tasa de aciertos (test, tiempo):64.38000000000001% y 0.13557815551757812
Entrenando clasificador Adaboost de scikit-learn para el dígito 3, T=2
Tasa de aciertos (test, tiempo):85.34% y 0.258087158203125
…
Entrenando clasificador Adaboost de scikit-learn para el dígito 3, T=450
Tasa de aciertos (test, tiempo):93.17% y 56.851085901260376
Entrenando clasificador Adaboost de scikit-learn para el dígito 3, T=900
Tasa de aciertos (test, tiempo):93.32000000000001% y 115.55954098701477




#Llama a la Tarea 1C e imprime lo consiguiente

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

El código implementado en esta tarea es simple:

1-Cargar Datos: como hasta ahora (sin balancear)
2-Configurar Clasificador Ensemble: Define un clasificador Decision Tree con una **profundidad máxima de 3** mediante DecisionTreeClassifier. Luego, configura un clasificador AdaBoost utilizando este Decision Tree como base.
3-Entrenamiento del Modelo: Entrena el clasificador AdaBoost en el conjunto de entrenamiento MNIST y mide el tiempo de entrenamiento utilizando fit() y calculando la diferencia entre el tiempo inicial y final.
4-Realizar Predicciones: Utiliza el modelo entrenado para realizar predicciones en el conjunto de prueba.
5-Evaluar Precisión: Calcula y muestra la precisión del modelo en el conjunto de prueba usando la función de precisión y muestra el tiempo total de entrenamiento y precisión alcanzada.

**Accuracy on AdaBoost with Decision Tree: 82.000 %**
**Training Time: 94.938 seconds**

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

Se ha creado un clasificador MPL secuencial y se ha experimentado con los siguientes parámetros con el fin de obtener los valores óptimos:

| Parámetro | Variaciones Estudiadas | Efectos Observados |
|---|---|---|
| Número de capas ocultas | 1, 2, 3 | Aumenta la capacidad de aprendizaje, pero conlleva mayor tiempo de cómputo. |
| Neuronas por capa oculta | 32, 64, 128 | Mayor número de neuronas permite capturar patrones más complejos, pero aumenta la complejidad del modelo. |
| Learning Rate | 0.001, 0.01, 0.1 | Ajustar la tasa de aprendizaje para mejorar la convergencia y rendimiento. |
| Batch Size | 16, 32, 64 | Variar el tamaño del lote afecta la velocidad de entrenamiento. |
| Validation Split | 0.1, 0.2, 0.3 | Proporción de datos de validación afecta el ajuste del modelo. |

Tras realizar pruebas con los valores, la combinación que ha presentado un mejor rendimiento es Número de capas ocultas = 3, Neuronas por capa oculta = 128, learning rate:=0.001, Batch Size = 64 y Validation Split=0.3.

Resultando en:
Accuracy on MLP: 97.330%
Training Time: 11.44 seconds

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

**Características de las CNN:**
-Capas Convolucionales: Las capas convolucionales permiten aprender patrones espaciales en la imagen, como bordes y texturas, utilizando filtros que se desplazan sobre la entrada.

-Capas de Pooling: Las capas de pooling reducen la dimensionalidad de las representaciones espaciales, preservando las características más importantes y mejorando la eficiencia computacional.

-Jerarquía de Características: Las CNN aprenden jerarquías de características, permitiendo la detección de patrones complejos a partir de combinaciones más simples.

-Invariancia a la Traslación: Las CNN son invariables a pequeñas traslaciones en la entrada, lo que las hace robustas a las variaciones en la posición de los objetos.

-Parámetros Compartidos: Las capas convolucionales comparten parámetros, lo que reduce el número de parámetros entrenables y facilita el aprendizaje de patrones invariantes.

**Ventajas sobre MLP:**
En cuanto a las ventajas sobre el modelo MPL se encuentran:
-Mejor rendimiento en imágenes debido a la captura de patrones locales.
-Menor cantidad de parámetros entrenables.
-Invariancia espacial a través de las capas convolucionales.
-Mejora en la eficiencia y generalización.

En cuanto a la **experimentación con el modelo CNN**:

| Capas Convolucionales (n_hid_lyrs) | Neuronas por Capa Densa (n_nrns_lyr) | Batch Size | Learning Rate | epochs | Tiempo de Entrenamiento (s) | Precisión (%) |
|---|---|---|---|---|---|---|
| 1 | 32 | 32 | 0.001 | 10 | 151.821 | 98,64% |
| 2 | 128 | 64 | 0.01 | 15 | 1965.65 | 98.96% |
| 3 | 256 | 128 | 0.001 | 20 | 11967.98 | 99.74% |

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

No se han vuelto a implementar llamadas a cada una de las gráficas, se han usado los métodos previamente creados para analizar y extraer las conclusiones finales. Estas conclusiones son:

Los modelos de redes neuronales son más apropiados (aunque un poco más difíciles de implementar / configurar ) para el problema de clasificación de números escritos a mano. Tanto el tiempo como la precisión son mayores en estos. En cuanto a cual de estos modelos se debería utilizar, personalmente elegiría **CNN,** a pesar de sus tiempos de entrenamiento mayores (mucho mayores que los de MPL pero no excesivos), brinda una precisión un tanto mayor.

## *REFERENCIAS*

**Página oficial de scikit-learn:**
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

**Departamento de Ciencia de la Computación e Inteligencia Artificial UA:**
Se ha usado el material proporcionado y las implementaciones son como se detallan(puede que no todo sean estándares).