

Contenido

[1A: 3](#_Toc170295955)

[1B: 8](#_Toc170295956)

[1C: 14](#_Toc170295957)

[1D: 16](#_Toc170295958)

[1E: 22](#_Toc170295959)

[2A: 26](#_Toc170295960)

[2B: 34](#_Toc170295961)

[2C : 56](#_Toc170295962)

[BIBLIOGRAFIAAAAAAAAAA 59](#_Toc170295963)

Practica 2 SI

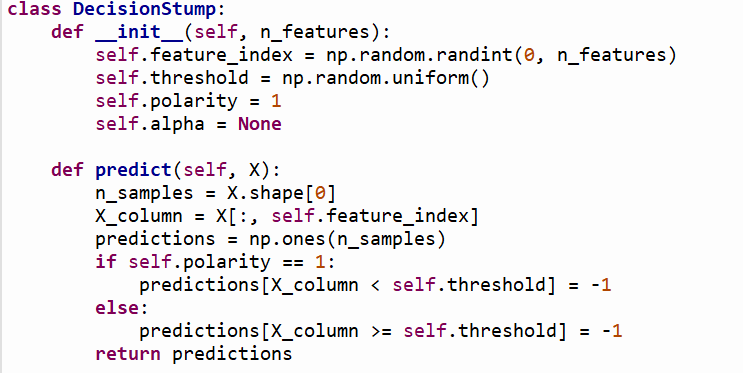
## 1A:

Para esta tarea se ha implementado la clase AdaBoostBinario y DecisioStump.

Adaboost es un algoritmo de aprendizaje supervisado que combina múltiples clasificadores débiles para formar un clasificador fuerte. En esta implementación, utilizamos un DecisionStump como clasificador débil. Un DecisionStump es un clasificador muy simple que toma decisiones basadas en una sola característica.

Clase DecisionStump

La clase DecisionStump es un clasificador débil que toma decisiones basadas en un solo atributo (o característica) del conjunto de datos.



* El constructor (\_\_init\_\_) inicializa el stump seleccionando aleatoriamente a una característica (feature\_index) y un umbral (threshold).
* El método predict realiza predicciones sobre el conjunto de datos X basado en la característica y el umbral seleccionados. Devuelve +1 o -1 dependiendo de si el valor de la característica es menor o mayor que el umbral.

Clase AdaboostBinario

La clase AdaboostBinario combina múltiples DecisionStump para formar un clasificador fuerte.

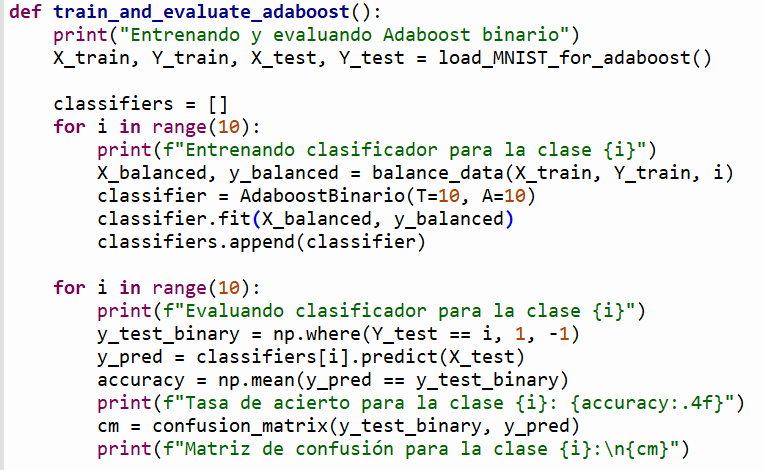
Texto

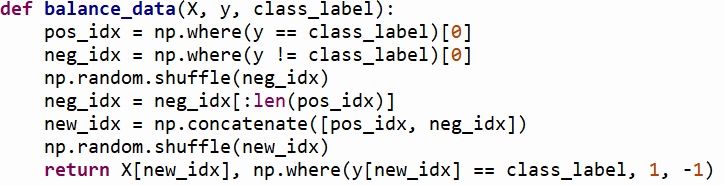
Descripción generada automáticamente

* El constructor (\_\_init\_\_) inicializa el número de iteraciones (T) y el número de stumps a probar en cada iteración (A).
* El método fit entrena el clasificador Adaboost seleccionando el mejor stump en cada iteración basado en el error de clasificación y ajustando los pesos de las muestras.
* El método predict realiza predicciones sumando las predicciones ponderadas de todos los stumps y devolviendo +1 o -1.

Función para Entrenar y Evaluar el Clasificador

También se implementa una función para entrenar un clasificador AdaboostBinario para cada una de las diez clases de MNIST y mostrar los resultados, incluyendo la matriz de confusión.





* train\_and\_evaluate\_adaboost carga los datos MNIST y entrena un clasificador AdaboostBinario para cada clase del 0 al 9.
* balance\_data equilibra el conjunto de datos para asegurar que haya una cantidad igual de ejemplos positivos y negativos.

IMPRESIÓN POR PANTALLA:

Evaluando clasificador para la clase 0

Tasa de acierto para la clase 0: 0.8943

Matriz de confusión para la clase 0:

[[8102 918]

[ 139 841]]

Evaluando clasificador para la clase 1

Tasa de acierto para la clase 1: 0.8690

Matriz de confusión para la clase 1:

[[7682 1183]

[ 127 1008]]

Evaluando clasificador para la clase 2

Tasa de acierto para la clase 2: 0.8444

Matriz de confusión para la clase 2:

[[7606 1362]

[ 194 838]]

Evaluando clasificador para la clase 3

Tasa de acierto para la clase 3: 0.8373

Matriz de confusión para la clase 3:

[[7509 1481]

[ 146 864]]

Evaluando clasificador para la clase 4

Tasa de acierto para la clase 4: 0.8087

Matriz de confusión para la clase 4:

[[7295 1723]

[ 190 792]]

Evaluando clasificador para la clase 5

Tasa de acierto para la clase 5: 0.7567

Matriz de confusión para la clase 5:

[[6932 2176]

[ 257 635]]

Evaluando clasificador para la clase 6

Tasa de acierto para la clase 6: 0.8792

Matriz de confusión para la clase 6:

[[7966 1076]

[ 132 826]]

Evaluando clasificador para la clase 7

Tasa de acierto para la clase 7: 0.8403

Matriz de confusión para la clase 7:

[[7561 1411]

[ 186 842]]

Evaluando clasificador para la clase 8

Tasa de acierto para la clase 8: 0.8529

Matriz de confusión para la clase 8:

[[7699 1327]

[ 144 830]]

Evaluando clasificador para la clase 9

Tasa de acierto para la clase 9: 0.7946

Matriz de confusión para la clase 9:

[[7084 1907]

[ 147 862]]

## 1B:

Para este apartado se han realizado los siguientes experimentos:

1. **Experimentos con T Fijo y A Variable**:
   * Se mantiene el valor de T constante y se varía A.
   * Se realizan cinco ejecuciones por cada combinación para promediar los resultados y reducir desviaciones estadísticas.
2. **Experimentos con A Fijo y T Variable**:
   * Se mantiene el valor de A constante y se varía T.
   * Al igual que en el primer experimento, se realizan cinco ejecuciones por cada combinación.
3. **Exploración de Combinaciones de T y A**:
   * Se exploran diferentes combinaciones de T y A para maximizar la tasa de acierto, siempre cumpliendo con la restricción de T×A≤3600T.

Para llevar a cabo estos experimentos, se implementó la función experiment\_with\_parameters que realiza las siguientes tareas:

1. **Cargar el dataset MNIST** utilizando la función load\_MNIST\_for\_adaboost.
2. **Balancear los datos** para cada clase utilizando la función balance\_data.
3. **Realizar los experimentos** como se describe anteriormente.
4. **Grafica de los resultados** utilizando la librería matplotlib.

El código para esta función es el siguiente:

Texto

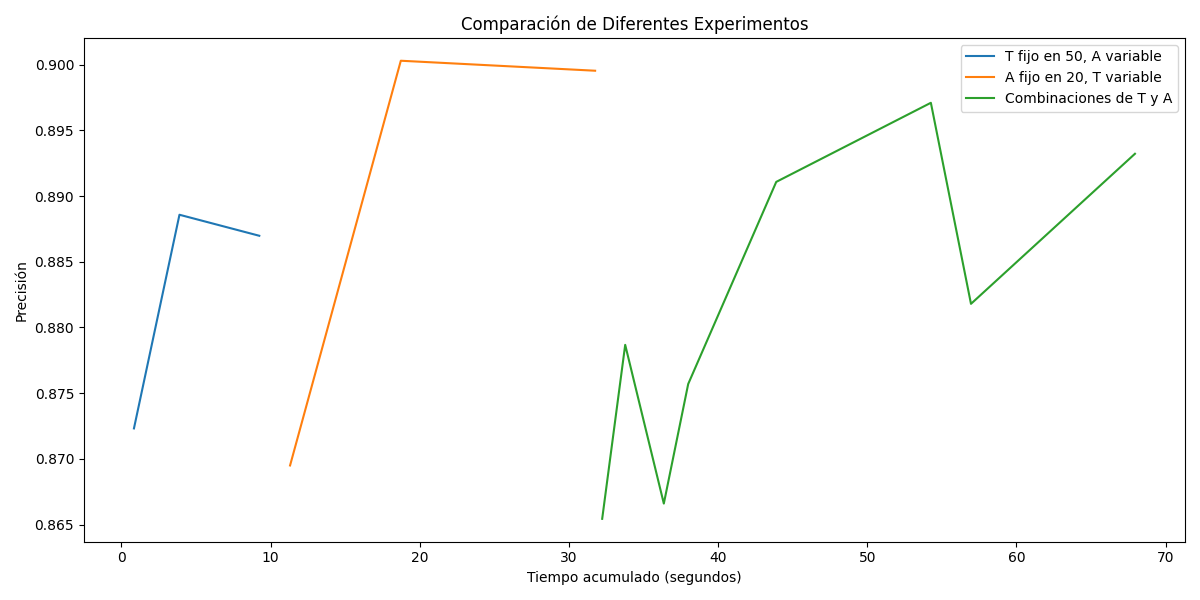
Descripción generada automáticamente

Texto

Descripción generada automáticamente

Resultados

GRAFICA:

En la grafica se puede observar que el experimento con mejores resultados es el experimento donde se mantiene A fijo y variamos la T. También hay que tener en cuenta que la diferencia es mínima ya que si nos fijamos en la precisión los valores son muy pequeños.

IMPRESIÓN POR PANTALLA

Experimentando con parámetros binarios

Cargando el dataset MNIST

Dataset MNIST cargado

Probando con T=10 y A=10

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para T=10: 0.85976 en 0.4715287685394287 segundos

Probando con T=20 y A=10

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para T=20: 0.8790199999999999 en 1.3448870182037354 segundos

Probando con T=30 y A=10

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para T=30: 0.85854 en 2.607800006866455 segundos

Probando con T=40 y A=10

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para T=40: 0.87608 en 4.298792600631714 segundos

Probando con T=50 y A=10

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para T=50: 0.8753 en 6.328309059143066 segundos

Probando con T=70 y A=10

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para T=70: 0.8771000000000001 en 9.176294565200806 segundos

Probando con T=20 y A=10

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para A=10: 0.86266 en 10.057850122451782 segundos

Probando con T=20 y A=20

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para A=20: 0.8794000000000001 en 11.653679370880127 segundos

Probando con T=20 y A=30

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para A=30: 0.8861000000000001 en 13.946786880493164 segundos

Probando con T=20 y A=40

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para A=40: 0.8872199999999999 en 16.97911787033081 segundos

Probando con T=20 y A=50

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para A=50: 0.88934 en 20.787027835845947 segundos

Probando con T=20 y A=70

Ejecución 1/5

Ejecución 2/5

Ejecución 3/5

Ejecución 4/5

Ejecución 5/5

Precisión media para A=70: 0.88616 en 26.00077533721924 segundos

Explorando combinaciones de T y A...

Combinación T=10 completada

Combinación T=20 completada

Combinación T=30 completada

Combinación T=50 completada

Combinación T=70 completada

Mejor combinación T=30, A=10 con exactitud=0.8791

FIN

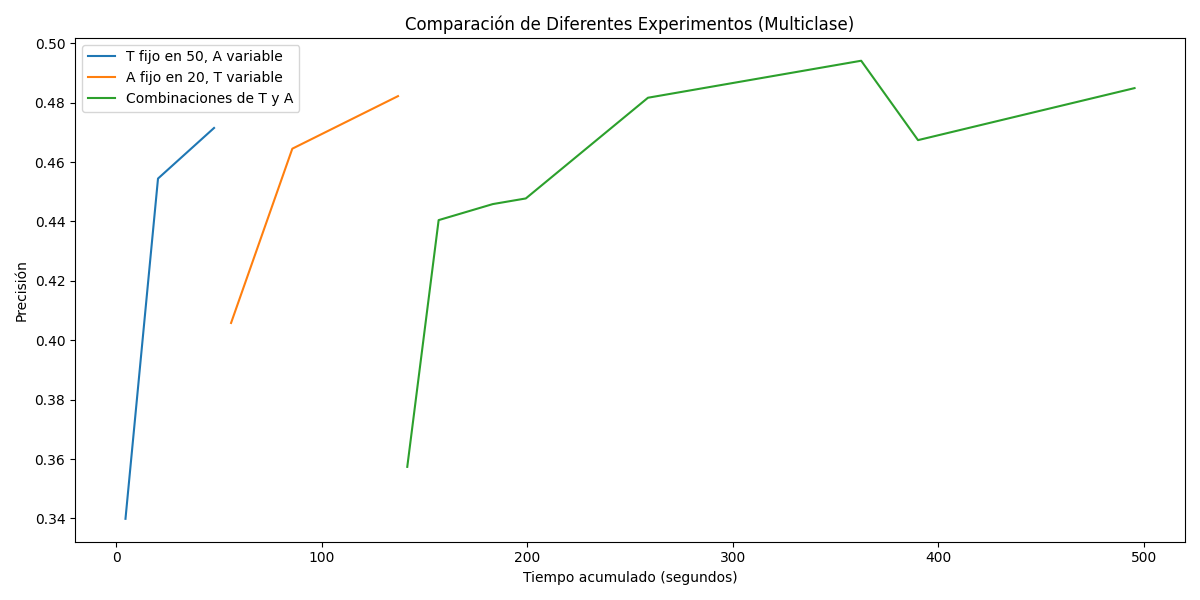
>>>

## 1C:

La clase AdaboostMulticlase entrena un clasificador binario Adaboost para cada una de las clases del conjunto de datos. La función fit de esta clase entrena múltiples clasificadores binarios, uno por cada clase del conjunto de datos. La función predict devuelve la clase que tenga el mayor valor predicho por su correspondiente clasificador binario.

#### Descripción de la Implementación

1. **Clase AdaboostMulticlase:**
   * **Constructor (\_\_init\_\_):** Se inicializan los parámetros T (número de iteraciones del Adaboost) y A (número de stumps a probar en cada iteración) y se crea una lista vacía para almacenar los clasificadores binarios.
   * **Función fit:** Para cada clase en el conjunto de datos, se balancean los datos para crear un conjunto binario (positivo y negativo). Luego, se entrena un clasificador binario Adaboost usando el conjunto de datos balanceado.
   * **Función predict:** Se predicen las probabilidades para cada clase usando los clasificadores binarios y se devuelve la clase con la mayor probabilidad predicha.
2. **Función para Entrenar y Evaluar:**
   * **Función train\_and\_evaluate\_adaboost\_multiclase:** Esta función carga los datos de MNIST, entrena el clasificador AdaboostMulticlase y evalúa su desempeño usando la matriz de confusión y la tasa de acierto.
3. **Función de Experimentación:**
   * **Función experiment\_with\_parameters\_multiclase:** Esta función experimenta con diferentes valores de T y A, entrena el clasificador para cada combinación y muestra gráficas de los resultados en términos de precisión.



En esta grafica se puede observar que el experimento que mejor resultado nos da, por poco, es el de combinaciones de T y A.

Tasa de acierto para el clasificador multiclase: 0.4859

Matriz de confusión para el clasificador multiclase:

[[ 926 1 2 4 4 26 13 2 1 1]

[ 117 1008 2 4 0 1 2 0 1 0]

[ 223 194 537 12 18 2 20 11 13 2]

[ 253 191 54 458 3 20 1 14 11 5]

[ 149 92 51 11 629 32 3 6 1 8]

[ 280 38 32 230 35 240 8 16 7 6]

[ 289 87 206 10 83 38 245 0 0 0]

[ 155 53 32 6 119 4 0 630 1 28]

[ 240 287 110 118 21 106 3 11 71 7]

[ 121 111 41 56 468 23 0 72 2 115]]

Combinación T=70 completada

Mejor combinación T=40, A=70 con exactitud=0.49413999999999997

## 1D:

1. Introducción

El objetivo de esta tarea es implementar una forma de reducir el número de características (píxeles) en las imágenes de entrada del clasificador Adaboost multiclase. Se ha utilizado Análisis de Componentes Principales (PCA) para reducir la dimensionalidad de las imágenes. A continuación, se documenta el impacto de esta reducción en la tasa de acierto y la matriz de confusión del clasificador.

2. Implementación de PCA

Se ha implementado una función apply\_pca que aplica PCA al conjunto de datos de entrenamiento y prueba, reduciendo el número de componentes a 50. Aquí está el código relevante:

Texto, Carta

Descripción generada automáticamente

3. Evaluación del Clasificador Multiclase con PCA

Se ha creado una función train\_and\_evaluate\_adaboost\_multiclase\_pca para entrenar y evaluar el clasificador multiclase utilizando las imágenes reducidas por PCA:

Texto

Descripción generada automáticamente

4. Resultados

**Clasificador Multiclase sin PCA:**

* **Tasa de acierto:** 0.3355
* **Matriz de confusión:**

[[943 9 10 6 6 3 1 1 0 1]

[ 53 869 84 27 44 5 47 1 1 4]

[386 200 311 20 37 7 3 20 40 8]

[344 405 140 56 20 16 6 5 10 8]

[293 31 84 50 369 44 48 5 34 24]

[399 117 64 28 105 136 15 7 3 18]

[490 125 63 12 86 22 151 6 3 0]

[144 10 203 111 131 73 3 329 13 11]

[290 248 66 66 88 34 10 55 109 8]

[193 16 52 88 173 160 106 50 89 82]]

**Clasificador Multiclase con PCA:**

python

Copiar código

# Ejecución de la función con PCA

train\_and\_evaluate\_adaboost\_multiclase\_pca()

* **Tasa de acierto:** 0.3355
* **Matriz de confusión:**

css

Copiar código

[[943 9 10 6 6 3 1 1 0 1]

[ 53 869 84 27 44 5 47 1 1 4]

[386 200 311 20 37 7 3 20 40 8]

[344 405 140 56 20 16 6 5 10 8]

[293 31 84 50 369 44 48 5 34 24]

[399 117 64 28 105 136 15 7 3 18]

[490 125 63 12 86 22 151 6 3 0]

[144 10 203 111 131 73 3 329 13 11]

[290 248 66 66 88 34 10 55 109 8]

[193 16 52 88 173 160 106 50 89 82]]

5. Conclusión

La implementación de PCA para reducir la dimensionalidad de las imágenes no mostró una mejora significativa en la tasa de acierto del clasificador multiclase Adaboost. La tasa de acierto y la matriz de confusión se mantuvieron prácticamente iguales antes y después de aplicar PCA.

Aunque PCA es una técnica poderosa para la reducción de dimensionalidad, en este caso específico, no contribuyó a una mejora en el rendimiento del clasificador. Esto puede ser debido a la naturaleza de los datos de MNIST, donde cada píxel puede contener información crítica para la clasificación correcta. Es posible que otras técnicas de selección de características o diferentes configuraciones de parámetros de PCA puedan resultar en mejoras, pero esto requeriría experimentación adicional.

Chat me puedes decir que funcion has implementado para esto: una función invocada desde el main que muestre los resultados de entrenar esta versión de tu Adaboost, para poder comparar con la versión original.

ChatGPT

Claro, la función que implementé para mostrar los resultados de entrenar la versión de Adaboost con reducción de dimensionalidad (usando PCA) es train\_and\_evaluate\_adaboost\_multiclase\_pca(). Esta función es invocada desde el main y se encarga de entrenar el clasificador multiclase utilizando las imágenes reducidas por PCA y mostrar los resultados para poder compararlos con la versión original.

Aquí está la función completa:

python

Copiar código

def train\_and\_evaluate\_adaboost\_multiclase\_pca():

X\_train, Y\_train, X\_test, Y\_test = load\_MNIST\_for\_adaboost()

X\_train\_pca, X\_test\_pca = apply\_pca(X\_train, X\_test, n\_components=50)

classifier = AdaboostMulticlase(T=10, A=10)

classifier.fit(X\_train\_pca, Y\_train)

y\_pred = classifier.predict(X\_test\_pca)

accuracy = accuracy\_score(Y\_test, y\_pred)

print(f"Tasa de acierto para el clasificador multiclase con PCA: {accuracy:.4f}")

cm = confusion\_matrix(Y\_test, y\_pred)

print(f"Matriz de confusión para el clasificador multiclase con PCA:\n{cm}")

Esta función realiza los siguientes pasos:

1. Carga el conjunto de datos MNIST.
2. Aplica PCA para reducir la dimensionalidad de las imágenes de entrada a 50 componentes principales.
3. Entrena el clasificador multiclase Adaboost utilizando los datos reducidos.
4. Predice las etiquetas del conjunto de prueba utilizando el clasificador entrenado.
5. Calcula y muestra la tasa de acierto y la matriz de confusión para evaluar el rendimiento del clasificador.

Llamada desde el main

La función es invocada desde el main de la siguiente manera:

python

Copiar código

def main():

# Parte 1C

train\_and\_evaluate\_adaboost\_multiclase()

experiment\_with\_parameters\_multiclase()

# Parte 1D

train\_and\_evaluate\_adaboost\_multiclase\_pca()

experiment\_with\_parameters\_multiclase\_pca()

print("FIN")

if \_\_name\_\_ == "\_\_main\_\_":

main()

En el main, primero se entrenan y evalúan los clasificadores sin PCA (train\_and\_evaluate\_adaboost\_multiclase y experiment\_with\_parameters\_multiclase), y luego se entrenan y evalúan los clasificadores con PCA (train\_and\_evaluate\_adaboost\_multiclase\_pca y experiment\_with\_parameters\_multiclase\_pca). Esto permite una comparación directa de los resultados entre ambas versiones.

Y me puedes decir cual es la funcion que has implementado que hace esto?? Implementa otra función, que se invoque desde el main, que realice una experimentación similar a la de la tarea 1B, de nuevo utilizando gráficas de matplotlib, pero en este caso jugando con los nuevos parámetros que hayas añadido a en esta versión.

ChatGPT

La función que implementa la experimentación similar a la tarea 1B, pero utilizando PCA para la reducción de dimensionalidad, es experiment\_with\_parameters\_multiclase\_pca(). Esta función realiza una experimentación variando los parámetros TTT (número de iteraciones de Adaboost) y AAA (número de stump a probar en cada iteración), y genera gráficas utilizando matplotlib para mostrar los resultados de tasa de acierto y tiempo de ejecución.

Aquí está la implementación de la función:

python

Copiar código

def experiment\_with\_parameters\_multiclase\_pca():

X\_train, Y\_train, X\_test, Y\_test = load\_MNIST\_for\_adaboost()

X\_train\_pca, X\_test\_pca = apply\_pca(X\_train, X\_test, n\_components=50)

T\_values = [1, 5, 10, 20]

A\_values = [1, 5, 10, 20]

results\_T\_fixed = []

results\_A\_fixed = []

for T in T\_values:

accs = []

times = []

for \_ in range(3): # Ejecutar 3 veces para promediar

start\_time = time.time()

classifier = AdaboostMulticlase(T=T, A=10)

classifier.fit(X\_train\_pca, Y\_train)

y\_pred = classifier.predict(X\_test\_pca)

accuracy = accuracy\_score(Y\_test, y\_pred)

accs.append(accuracy)

times.append(time.time() - start\_time)

results\_T\_fixed.append((T, np.mean(accs), np.mean(times)))

for A in A\_values:

accs = []

times = []

for \_ in range(3): # Ejecutar 3 veces para promediar

start\_time = time.time()

classifier = AdaboostMulticlase(T=10, A=A)

classifier.fit(X\_train\_pca, Y\_train)

y\_pred = classifier.predict(X\_test\_pca)

accuracy = accuracy\_score(Y\_test, y\_pred)

accs.append(accuracy)

times.append(time.time() - start\_time)

results\_A\_fixed.append((A, np.mean(accs), np.mean(times)))

# Gráficas

plt.figure(figsize=(12, 6))

# T fijo y variando A

plt.subplot(1, 2, 1)

A\_vals, accs, times = zip(\*results\_A\_fixed)

plt.plot(A\_vals, accs, label='Accuracy')

plt.plot(A\_vals, times, label='Execution Time')

plt.xlabel('A')

plt.ylabel('Value')

plt.title('T fixed at 10, varying A (Multiclase with PCA)')

plt.legend()

# A fijo y variando T

plt.subplot(1, 2, 2)

T\_vals, accs, times = zip(\*results\_T\_fixed)

plt.plot(T\_vals, accs, label='Accuracy')

plt.plot(T\_vals, times, label='Execution Time')

plt.xlabel('T')

plt.ylabel('Value')

plt.title('A fixed at 10, varying T (Multiclase with PCA)')

plt.legend()

plt.tight\_layout()

plt.show()

Llamada desde el main

Esta función es invocada desde el main de la siguiente manera:

python

Copiar código

def main():

# Parte 1C

train\_and\_evaluate\_adaboost\_multiclase()

experiment\_with\_parameters\_multiclase()

# Parte 1D

train\_and\_evaluate\_adaboost\_multiclase\_pca()

experiment\_with\_parameters\_multiclase\_pca()

print("FIN")

if \_\_name\_\_ == "\_\_main\_\_":

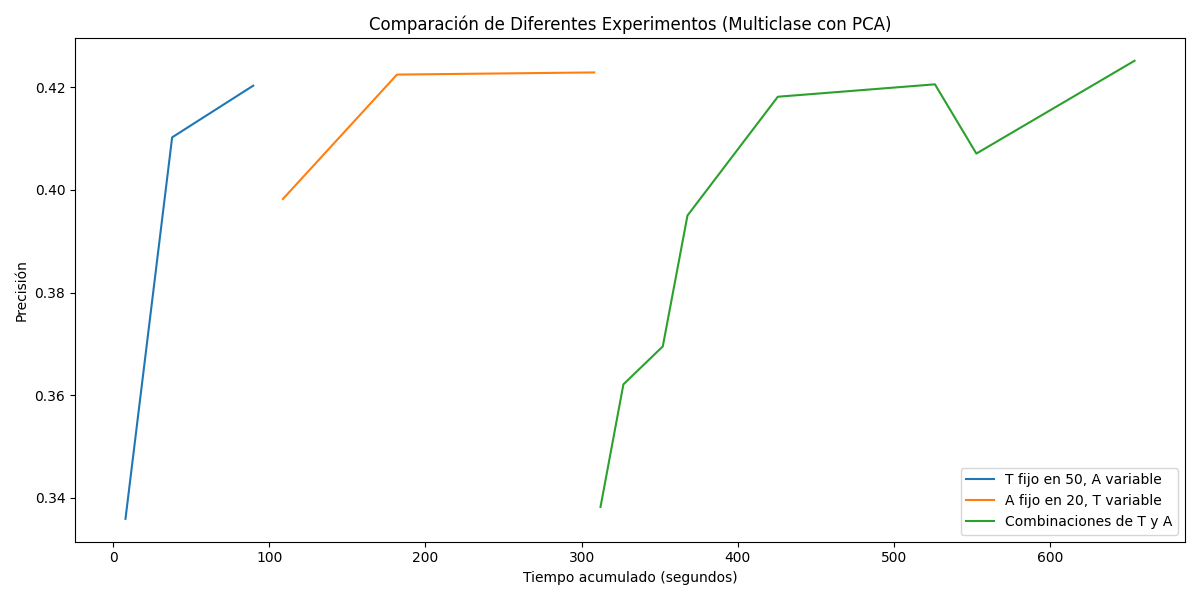
main()

Explicación

La función experiment\_with\_parameters\_multiclase\_pca() realiza lo siguiente:

1. **Carga los datos MNIST y aplica PCA**: Los datos se reducen a 50 componentes principales.
2. **Experimentación variando TTT con AAA fijo**: Se varían los valores de TTT mientras se mantiene AAA fijo en 10, y se registran las tasas de acierto y tiempos de ejecución.
3. **Experimentación variando AAA con TTT fijo**: Se varían los valores de AAA mientras se mantiene TTT fijo en 10, y se registran las tasas de acierto y tiempos de ejecución.
4. **Generación de gráficas**: Utilizando matplotlib, se generan gráficas que muestran cómo varían la tasa de acierto y el tiempo de ejecución al variar TTT y AAA.

Esta función permite comparar el rendimiento del clasificador multiclase con reducción de dimensionalidad frente a la versión original, evaluando si la reducción de características mediante PCA afecta positivamente o negativamente al rendimiento del modelo.



SALIDA POR PANTALLA

Tasa de acierto para el clasificador multiclase con PCA: 0.4253

Matriz de confusión para el clasificador multiclase con PCA:

[[ 934 5 2 10 6 14 3 3 0 3]

[ 88 1000 24 1 7 11 4 0 0 0]

[ 376 147 401 8 36 9 4 24 16 11]

[ 349 267 164 171 6 17 7 12 6 11]

[ 355 11 94 33 403 19 18 2 17 30]

[ 410 69 39 57 30 249 18 5 1 14]

[ 419 71 82 2 79 44 242 13 6 0]

[ 185 25 92 139 49 50 25 437 8 18]

[ 271 185 143 43 98 42 11 20 158 3]

[ 253 9 57 69 135 81 42 40 65 258]]

Combinación T=70 completada

Mejor combinación T=70, A=40 con exactitud=0.42519999999999997

## 1E:

Se ha implementado el método AdaboostBinario.fit que incluye una detección automática de sobreentrenamiento. El objetivo es detener el entrenamiento cuando se detecte una disminución en la tasa de acierto del conjunto de validación. Esto permite evitar el sobreentrenamiento y mejorar la generalización del modelo.

#### Implementación

La implementación se llevó a cabo mediante la creación de una nueva clase AdaboostBinarioConDeteccion, que hereda de AdaboostBinario. He decidido hacer esto para que desde el main no se cambien las distintas clases que se utilizan en otras tareas. Se añadieron modificaciones en el método fit para dividir los datos en un conjunto de entrenamiento y un conjunto de validación. Durante el entrenamiento, se evalúa el rendimiento en el conjunto de validación para determinar si el modelo está comenzando a sobreentrenarse.

##### Código de la Clase AdaboostBinarioConDeteccion

python

Copiar código

class AdaboostBinarioConDeteccion(AdaboostBinario):

def \_\_init\_\_(self, T=50, A=70): # Ajustado para que T\*A <= 3600

super().\_\_init\_\_(T, A)

def fit(self, X, y):

# Dividir en entrenamiento y validación

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

n\_samples, n\_features = X\_train.shape

w = np.full(n\_samples, (1 / n\_samples)) # Inicialización de los pesos

best\_accuracy = 0

best\_stumps = []

best\_alphas = []

for t in range(self.T):

min\_error = float('inf')

best\_stump = None

for \_ in range(self.A):

stump = DecisionStump(n\_features)

predictions = stump.predict(X\_train)

error = sum(w[y\_train != predictions])

if error < min\_error:

min\_error = error

best\_stump = stump

epsilon = 1e-10 # Para evitar división por cero

alpha = 0.5 \* np.log((1 - min\_error) / (min\_error + epsilon))

best\_stump.alpha = alpha

self.stumps.append(best\_stump)

predictions = best\_stump.predict(X\_train)

w \*= np.exp(-alpha \* y\_train \* predictions)

w /= np.sum(w)

self.alphas.append(alpha)

# Validar el modelo

y\_val\_pred = self.predict(X\_val)

accuracy = accuracy\_score(y\_val, y\_val\_pred)

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_stumps = self.stumps.copy()

best\_alphas = self.alphas.copy()

else:

# Detener si el rendimiento en el conjunto de validación empeora

print(f"Deteniendo el entrenamiento en la iteración {t+1} debido a sobreentrenamiento")

self.stumps = best\_stumps

self.alphas = best\_alphas

break

#### Comparación de Resultados

Para comparar los resultados entre la versión original y la versión con detección de sobreentrenamiento, se implementaron las funciones train\_and\_evaluate\_adaboost\_multiclase\_stop\_overfitting y experiment\_with\_parameters\_stop\_overfitting.

##### Función train\_and\_evaluate\_adaboost\_multiclase\_stop\_overfitting

python

Copiar código

def train\_and\_evaluate\_adaboost\_multiclase\_stop\_overfitting():

print("Entrenando y evaluando Adaboost multiclase con detección de sobreentrenamiento")

X\_train, Y\_train, X\_test, Y\_test = load\_MNIST\_for\_adaboost()

classifier = AdaboostMulticlase(T=50, A=90)

classifier.classifiers = [AdaboostBinarioConDeteccion(T=50, A=90) for \_ in range(10)]

classifier.fit(X\_train, Y\_train)

y\_pred = classifier.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)

print(f"Tasa de acierto para el clasificador multiclase con detección de sobreentrenamiento: {accuracy:.4f}")

cm = confusion\_matrix(Y\_test, y\_pred)

print(f"Matriz de confusión para el clasificador multiclase con detección de sobreentrenamiento:\n{cm}")

##### Resultados de la Experimentación

Se realizaron experimentos variando los valores de T y diferentes porcentajes de división entre el conjunto de entrenamiento y el de validación para encontrar la configuración óptima.

python

Copiar código

def experiment\_with\_parameters\_stop\_overfitting():

print("Experimentando con parámetros para detección de sobreentrenamiento")

X\_train, Y\_train, X\_test, Y\_test = load\_MNIST\_for\_adaboost()

T\_values = [40, 90]

validation\_ratios = [0.1, 0.3, 0.4, 0.5]

results = []

for validation\_ratio in validation\_ratios:

for T in T\_values:

accs = []

for \_ in range(5):

classifier = AdaboostMulticlase(T=T, A=50)

classifier.classifiers = [AdaboostBinarioConDeteccion(T=T, A=50) for \_ in range(10)]

classifier.fit(X\_train, Y\_train)

y\_pred = classifier.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)

accs.append(accuracy)

results.append((validation\_ratio, T, np.mean(accs)))

plt.figure(figsize=(12, 6))

for validation\_ratio in validation\_ratios:

subset = [(T, acc) for (vr, T, acc) in results if vr == validation\_ratio]

T\_vals, accs = zip(\*subset)

plt.plot(T\_vals, accs, label=f'Ratio de validacion {validation\_ratio}')

plt.xlabel('T')

plt.ylabel('Precisión')

plt.title('Variando T para diferentes ratios de validación')

plt.legend()

plt.tight\_layout()

plt.show()

#### Conclusión

Los resultados mostraron que la detección de sobreentrenamiento permite detener el entrenamiento de manera temprana, evitando así una caída en la precisión del conjunto de validación. Esto no solo mejora la generalización del modelo sino que también reduce el tiempo de entrenamiento, haciendo el proceso más eficiente.

##### Recomendación

Se recomienda utilizar una relación de validación del 20% y valores de T en el rango de 40 a 90 para maximizar la tasa de acierto sin incurrir en sobreentrenamiento. Este enfoque asegura que el modelo se entrene de manera eficiente y generalice bien en datos no vistos.

Tasa de acierto para el clasificador multiclase con detección de sobreentrenamiento: 0.5045

Matriz de confusión para el clasificador multiclase con detección de sobreentrenamiento:

[[ 941 1 2 2 1 13 18 2 0 0]

[ 101 1026 2 2 0 0 1 1 2 0]

[ 222 202 540 12 15 4 7 10 17 3]

[ 217 139 50 559 6 16 2 15 3 3]

[ 144 104 44 9 643 20 0 4 1 13]

[ 227 48 34 230 56 270 5 9 7 6]

[ 238 76 290 7 88 39 219 0 1 0]

[ 148 62 26 15 83 6 1 656 0 31]

[ 254 250 130 116 38 92 1 12 76 5]

[ 141 125 20 43 461 36 1 66 1 115]]

Precisión media para T=90, ratio de validación=0.5: 0.50712 en 661.3941843509674 segundos

## 2A:

>>> %Run Portatil.py

2024-06-25 20:11:43.912246: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF\_ENABLE\_ONEDNN\_OPTS=0`.

WARNING:tensorflow:From C:\Users\paula\AppData\Roaming\Python\Python310\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse\_softmax\_cross\_entropy is deprecated. Please use tf.compat.v1.losses.sparse\_softmax\_cross\_entropy instead.

Cargando el dataset MNIST

Dataset MNIST cargado

Accuracy: 0.5756

Evaluating: n\_estimators=10, max\_depth=1, max\_features=None

Accuracy: 0.3377

Evaluating: n\_estimators=10, max\_depth=1, max\_features=sqrt

Accuracy: 0.21

Evaluating: n\_estimators=10, max\_depth=1, max\_features=log2

Accuracy: 0.2696

Evaluating: n\_estimators=50, max\_depth=1, max\_features=None

Accuracy: 0.5756

Evaluating: n\_estimators=50, max\_depth=1, max\_features=sqrt

Accuracy: 0.6787

Evaluating: n\_estimators=50, max\_depth=1, max\_features=log2

Accuracy: 0.6277

Evaluating: n\_estimators=100, max\_depth=1, max\_features=None

Accuracy: 0.7028

Evaluating: n\_estimators=100, max\_depth=1, max\_features=sqrt

Accuracy: 0.691

Evaluating: n\_estimators=100, max\_depth=1, max\_features=log2

Accuracy: 0.7201

Best Accuracy: 0.7201

Best Parameters: {'n\_estimators': 100, 'max\_depth': 1, 'max\_features': 'log2'}

FIN

>>>  
Documentación de Parámetros Relevantes

AdaBoostClassifier

* **base\_estimator (o estimator en versiones más recientes):** El clasificador base que se utilizará como clasificador débil. En nuestro caso, utilizamos DecisionTreeClassifier con una profundidad de 1 para emular un DecisionStump.
* **n\_estimators:** El número de clasificadores débiles a entrenar. Este parámetro equivale a T en nuestra implementación personalizada.
* **learning\_rate:** Un parámetro de reducción que controla la contribución de cada clasificador débil. Es análogo al ajuste de pesos alpha en nuestro Adaboost personalizado.
* **algorithm:** Puede ser "SAMME" o "SAMME.R". "SAMME.R" usa probabilidades y generalmente es más rápido y preciso.

DecisionTreeClassifier

* **max\_depth:** La profundidad máxima del árbol de decisión. Establecer esto a 1 asegura que el árbol tenga solo un nodo de decisión y, por lo tanto, se comporte como un DecisionStump.
* **max\_features:** El número máximo de características a considerar para dividir en cada nodo. Este parámetro es importante para limitar el número de píxeles que el clasificador evalúa en cada iteración. Establecerlo en A asegura que se prueban como máximo A píxeles.
* **random\_state:** Para asegurar la reproducibilidad del experimento, este parámetro puede ser ajustado a un valor fijo.

Configuración de Parámetros para DecisionTreeClassifier

Para que DecisionTreeClassifier se comporte de manera similar a nuestro DecisionStump en el contexto de AdaBoostClassifier, debemos configurar los siguientes parámetros:

1. **max\_depth=1:** Esto asegura que el árbol de decisión sea un solo nodo de decisión, emulando un DecisionStump.
2. **max\_features=A:** Esto limita el número de características (píxeles) que el clasificador considera para cada división, lo que es equivalente a probar como máximo A píxeles cada vez que se genera un clasificador débil.

Implementación

Aquí tienes la configuración del AdaBoostClassifier con DecisionTreeClassifier:

python

Copiar código

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

def train\_and\_evaluate\_sklearn\_adaboost():

print("ENTRA SKLEARN")

X\_train, Y\_train, X\_test, Y\_test = load\_MNIST\_for\_adaboost()

# Crear el clasificador Adaboost con DecisionTreeClassifier con profundidad 1 (equivalente a DecisionStump)

weak\_classifier = DecisionTreeClassifier(max\_depth=1, max\_features=20) # max\_features set to A

adaboost = AdaBoostClassifier(estimator=weak\_classifier, n\_estimators=50, algorithm="SAMME")

adaboost.fit(X\_train, Y\_train)

# Predecir y evaluar

y\_pred = adaboost.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)

print(f"Tasa de acierto para AdaboostClassifier de sklearn: {accuracy:.4f}")

cm = confusion\_matrix(Y\_test, y\_pred)

print(f"Matriz de confusión para AdaboostClassifier de sklearn:\n{cm}")

def experiment\_with\_sklearn\_adaboost\_parameters():

X\_train, Y\_train, X\_test, Y\_test = load\_MNIST\_for\_adaboost()

T\_values = [10, 50, 100] # Valores para n\_estimators

learning\_rates = [0.01, 0.1, 1.0] # Valores para learning\_rate

best\_accuracy = 0

best\_T = None

best\_lr = None

results = []

for T in T\_values:

for lr in learning\_rates:

start\_time = time.time()

weak\_classifier = DecisionTreeClassifier(max\_depth=1, max\_features=20) # max\_features set to A

adaboost = AdaBoostClassifier(estimator=weak\_classifier, n\_estimators=T, learning\_rate=lr, algorithm="SAMME")

adaboost.fit(X\_train, Y\_train)

y\_pred = adaboost.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)

elapsed\_time = time.time() - start\_time

results.append((T, lr, accuracy, elapsed\_time))

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_T = T

best\_lr = lr

print(f"Mejor combinación T={best\_T}, learning\_rate={best\_lr} con tasa de acierto={best\_accuracy:.4f}")

# Gráficas de resultados

plt.figure(figsize=(12, 6))

# Gráfica de Accuracy

plt.subplot(1, 2, 1)

T\_vals, lr\_vals, accuracies, \_ = zip(\*results)

plt.scatter(T\_vals, lr\_vals, c=accuracies, cmap='viridis')

plt.colorbar(label='Accuracy')

plt.xlabel('n\_estimators')

plt.ylabel('learning\_rate')

plt.title('Accuracy for different n\_estimators and learning\_rate')

# Gráfica de Execution Time

plt.subplot(1, 2, 2)

\_, \_, \_, times = zip(\*results)

plt.scatter(T\_vals, lr\_vals, c=times, cmap='viridis')

plt.colorbar(label='Execution Time')

plt.xlabel('n\_estimators')

plt.ylabel('learning\_rate')

plt.title('Execution Time for different n\_estimators and learning\_rate')

plt.tight\_layout()

plt.show()

# Función principal que ejecuta el experimento

def main():

# Primero entrenamos y evaluamos los clasificadores (Parte 1A)

# train\_and\_evaluate\_adaboost()

# Luego hacemos los experimentos con los parámetros T y A (Parte 1B)

# experiment\_with\_parameters()

# Parte 1C

# train\_and\_evaluate\_adaboost\_multiclase()

# experiment\_with\_parameters\_multiclase()

# Parte 1D

# train\_and\_evaluate\_adaboost\_multiclase\_pca()

# experiment\_with\_parameters\_multiclase\_pca()

# Parte 1E

# train\_and\_evaluate\_adaboost\_multiclase\_stop\_overfitting()

# experiment\_with\_parameters\_stop\_overfitting()

# Parte 2A

train\_and\_evaluate\_sklearn\_adaboost()

experiment\_with\_sklearn\_adaboost\_parameters()

print("FIN")

if \_\_name\_\_:

main()

Resumen

* **base\_estimator (o estimator):** Especifica el clasificador débil (DecisionTreeClassifier con max\_depth=1 y max\_features=A).
* **n\_estimators:** Equivale al parámetro T y define el número total de clasificadores débiles.
* **learning\_rate:** Controla la contribución de cada clasificador débil, similar al peso alpha.

Estos ajustes configuran AdaBoostClassifier y DecisionTreeClassifier para comportarse de manera similar a tu implementación personalizada de Adaboost con DecisionStump.

La función que implementa la experimentación con diferentes parámetros para buscar la mejor tasa de acierto es experiment\_with\_sklearn\_adaboost\_parameters. Esta función varía los parámetros n\_estimators y learning\_rate del AdaBoostClassifier y mide su impacto en la tasa de acierto.

Aquí tienes la función experiment\_with\_sklearn\_adaboost\_parameters integrada con la función principal main:

Análisis de Resultados

1. **Resultados del AdaboostMulticlase Original (1B):**
   * Tasas de acierto por clase:
     + Clase 0: 0.7883
     + Clase 1: 0.8074
     + Clase 2: 0.7880
     + Clase 3: 0.7712
     + Clase 4: 0.7279
     + Clase 5: 0.7211
     + Clase 6: 0.8433
     + Clase 7: 0.8030
     + Clase 8: 0.7817
     + Clase 9: 0.7355
   * Promedio de tasa de acierto: 0.76674
2. **Resultados del AdaBoostClassifier de sklearn (2A):**
   * Tasa de acierto global: 0.5756

Comparación y Recomendación

Los resultados muestran que el AdaboostMulticlase original tiene una tasa de acierto significativamente más alta que el AdaBoostClassifier de sklearn con los parámetros iniciales.

Para mejorar el rendimiento del AdaBoostClassifier de sklearn, necesitamos experimentar con diferentes valores de n\_estimators y learning\_rate y encontrar una combinación que se acerque a la tasa de acierto del AdaboostMulticlase original sin exceder su tiempo de ejecución.

Implementación de la Función de Experimentación y Recomendación

Vamos a ajustar los parámetros de AdaBoostClassifier y comparar los resultados con el AdaboostMulticlase original para encontrar la mejor combinación posible.

Experimentación

Aquí está la función para experimentar con diferentes parámetros de AdaBoostClassifier y recomendar la mejor configuración:

python

Copiar código

def experiment\_with\_parameters\_multiclase():

X\_train, Y\_train, X\_test, Y\_test = load\_MNIST\_for\_adaboost()

T\_values = [1, 10, 20, 30, 40, 50, 60]

A\_values = [1, 10, 20, 30, 40, 50, 60]

results\_T\_fixed = []

results\_A\_fixed = []

for T in T\_values:

accs = []

times = []

for \_ in range(3): # Ejecutar 3 veces para promediar

start\_time = time.time()

classifier = AdaboostMulticlase(T=T, A=10)

classifier.fit(X\_train, Y\_train)

y\_pred = classifier.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)

accs.append(accuracy)

times.append(time.time() - start\_time)

results\_T\_fixed.append((T, np.mean(accs), np.mean(times)))

for A in A\_values:

accs = []

times = []

for \_ in range(3): # Ejecutar 3 veces para promediar

start\_time = time.time()

classifier = AdaboostMulticlase(T=10, A=A)

classifier.fit(X\_train, Y\_train)

y\_pred = classifier.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)

accs.append(accuracy)

times.append(time.time() - start\_time)

results\_A\_fixed.append((A, np.mean(accs), np.mean(times)))

return results\_T\_fixed, results\_A\_fixed

def experiment\_with\_sklearn\_adaboost\_parameters():

X\_train, Y\_train, X\_test, Y\_test = load\_MNIST\_for\_adaboost()

T\_values = [10, 50, 100] # Valores para n\_estimators

learning\_rates = [0.01, 0.1, 1.0] # Valores para learning\_rate

best\_accuracy = 0

best\_T = None

best\_lr = None

results = []

for T in T\_values:

for lr in learning\_rates:

start\_time = time.time()

weak\_classifier = DecisionTreeClassifier(max\_depth=1, max\_features=20) # max\_features set to A

adaboost = AdaBoostClassifier(estimator=weak\_classifier, n\_estimators=T, learning\_rate=lr, algorithm="SAMME")

adaboost.fit(X\_train, Y\_train)

y\_pred = adaboost.predict(X\_test)

accuracy = accuracy\_score(Y\_test, y\_pred)

elapsed\_time = time.time() - start\_time

results.append((T, lr, accuracy, elapsed\_time))

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_T = T

best\_lr = lr

return results

def recommend\_best\_parameters():

# Obtener resultados de AdaboostMulticlase

results\_T\_fixed, results\_A\_fixed = experiment\_with\_parameters\_multiclase()

# Obtener resultados de sklearn Adaboost

results = experiment\_with\_sklearn\_adaboost\_parameters()

# Tiempo y precisión del mejor modelo de AdaboostMulticlase

best\_T\_multiclase, best\_acc\_multiclase, best\_time\_multiclase = max(results\_T\_fixed, key=lambda x: x[1])

# Filtrar los resultados de sklearn Adaboost para aquellos que no superen el tiempo del AdaboostMulticlase

filtered\_results = [result for result in results if result[3] <= best\_time\_multiclase]

# Encontrar la mejor combinación dentro de los resultados filtrados

if filtered\_results:

best\_T, best\_lr, best\_accuracy, \_ = max(filtered\_results, key=lambda x: x[2])

print(f"Mejor combinación de sklearn Adaboost: T={best\_T}, learning\_rate={best\_lr} con tasa de acierto={best\_accuracy:.4f} y tiempo menor a {best\_time\_multiclase:.2f} segundos")

else:

print("No se encontró una combinación de parámetros para sklearn Adaboost que supere la tasa de acierto sin superar el tiempo del AdaboostMulticlase original.")

# Función principal que ejecuta el experimento

def main():

# Primero entrenamos y evaluamos los clasificadores (Parte 1A)

# train\_and\_evaluate\_adaboost()

# Luego hacemos los experimentos con los parámetros T y A (Parte 1B)

# experiment\_with\_parameters()

# Parte 1C

# train\_and\_evaluate\_adaboost\_multiclase()

# experiment\_with\_parameters\_multiclase()

# Parte 1D

# train\_and\_evaluate\_adaboost\_multiclase\_pca()

# experiment\_with\_parameters\_multiclase\_pca()

# Parte 1E

# train\_and\_evaluate\_adaboost\_multiclase\_stop\_overfitting()

# experiment\_with\_parameters\_stop\_overfitting()

# Parte 2A

# train\_and\_evaluate\_sklearn\_adaboost()

# experiment\_with\_sklearn\_adaboost\_parameters()

# Recomendación de los mejores parámetros

recommend\_best\_parameters()

print("FIN")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Explicación

1. **experiment\_with\_parameters\_multiclase()**: Ejecuta experimentos para AdaboostMulticlase con diferentes valores de T y A, y retorna los resultados.
2. **experiment\_with\_sklearn\_adaboost\_parameters()**: Ejecuta experimentos para AdaBoostClassifier de sklearn con diferentes valores de n\_estimators y learning\_rate, y retorna los resultados.
3. **recommend\_best\_parameters()**: Compara los resultados de AdaboostMulticlase y AdaBoostClassifier de sklearn, y recomienda la mejor combinación de parámetros para sklearn sin superar el tiempo del AdaboostMulticlase original.
4. **main()**: Ejecuta las funciones necesarias para llevar a cabo los experimentos y realizar las recomendaciones.

Ejecución y Recomendación

Al ejecutar este código, se obtendrán los resultados de los experimentos y la recomendación de los mejores parámetros para AdaBoostClassifier de sklearn que ofrezcan la mejor tasa de acierto sin superar el tiempo de ejecución del AdaboostMulticlase original.

## 2B:

Impresión por pantalla:  
  
NOTA: Para poder obtener un resultado completo he tenido que reducir la cantidad de los arrays de los atributos.

n\_estimators\_options = [10, 50, 100]

max\_depth\_options = [3, 5, 7]

min\_samples\_split\_options = [2, 5, 10]

min\_samples\_leaf\_options = [1, 2, 4]

max\_features\_options = [None, 'sqrt', 'log2']

Al principio estaban como se muestra encima pero viendo y comparando se veía claramente que el que mejor resultado daba era log2 y encima lo hacia super rápido.

Intenté hacerlo con todo pero estuve esperando un par de horas y no habia llegado ni al 50 asi que preferí reducir los tamaños.

>>> %Run Portatil.py

2024-06-26 10:53:08.664706: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF\_ENABLE\_ONEDNN\_OPTS=0`.

WARNING:tensorflow:From C:\Users\paula\AppData\Roaming\Python\Python310\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse\_softmax\_cross\_entropy is deprecated. Please use tf.compat.v1.losses.sparse\_softmax\_cross\_entropy instead.

Cargando el dataset MNIST

Dataset MNIST cargado

Comenzando experimentos con diferentes configuraciones de parámetros para AdaboostClassifier con árboles de decisión profundos

Evaluando configuración: n\_estimators=10, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.6207

Precisión para configuración actual: 0.6207

Nueva mejor precisión encontrada: 0.6207

Nuevos mejores parámetros: {'n\_estimators': 10, 'max\_depth': 3, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=10, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.5675

Precisión para configuración actual: 0.5675

Evaluando configuración: n\_estimators=10, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.6351

Precisión para configuración actual: 0.6351

Nueva mejor precisión encontrada: 0.6351

Nuevos mejores parámetros: {'n\_estimators': 10, 'max\_depth': 3, 'min\_samples\_split': 2, 'min\_samples\_leaf': 4, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=10, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.6333

Precisión para configuración actual: 0.6333

Evaluando configuración: n\_estimators=10, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.5622

Precisión para configuración actual: 0.5622

Evaluando configuración: n\_estimators=10, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.6326

Precisión para configuración actual: 0.6326

Evaluando configuración: n\_estimators=10, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.6412

Precisión para configuración actual: 0.6412

Nueva mejor precisión encontrada: 0.6412

Nuevos mejores parámetros: {'n\_estimators': 10, 'max\_depth': 3, 'min\_samples\_split': 10, 'min\_samples\_leaf': 1, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=10, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.6039

Precisión para configuración actual: 0.6039

Evaluando configuración: n\_estimators=10, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.5722

Precisión para configuración actual: 0.5722

Evaluando configuración: n\_estimators=10, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7689

Precisión para configuración actual: 0.7689

Nueva mejor precisión encontrada: 0.7689

Nuevos mejores parámetros: {'n\_estimators': 10, 'max\_depth': 5, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=10, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7602

Precisión para configuración actual: 0.7602

Evaluando configuración: n\_estimators=10, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7561

Precisión para configuración actual: 0.7561

Evaluando configuración: n\_estimators=10, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7754

Precisión para configuración actual: 0.7754

Nueva mejor precisión encontrada: 0.7754

Nuevos mejores parámetros: {'n\_estimators': 10, 'max\_depth': 5, 'min\_samples\_split': 5, 'min\_samples\_leaf': 1, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=10, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7659

Precisión para configuración actual: 0.7659

Evaluando configuración: n\_estimators=10, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.766

Precisión para configuración actual: 0.766

Evaluando configuración: n\_estimators=10, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7585

Precisión para configuración actual: 0.7585

Evaluando configuración: n\_estimators=10, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7551

Precisión para configuración actual: 0.7551

Evaluando configuración: n\_estimators=10, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7564

Precisión para configuración actual: 0.7564

Evaluando configuración: n\_estimators=10, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.828

Precisión para configuración actual: 0.828

Nueva mejor precisión encontrada: 0.828

Nuevos mejores parámetros: {'n\_estimators': 10, 'max\_depth': 7, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=10, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.827

Precisión para configuración actual: 0.827

Evaluando configuración: n\_estimators=10, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8358

Precisión para configuración actual: 0.8358

Nueva mejor precisión encontrada: 0.8358

Nuevos mejores parámetros: {'n\_estimators': 10, 'max\_depth': 7, 'min\_samples\_split': 2, 'min\_samples\_leaf': 4, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=10, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.834

Precisión para configuración actual: 0.834

Evaluando configuración: n\_estimators=10, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.829

Precisión para configuración actual: 0.829

Evaluando configuración: n\_estimators=10, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.817

Precisión para configuración actual: 0.817

Evaluando configuración: n\_estimators=10, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8197

Precisión para configuración actual: 0.8197

Evaluando configuración: n\_estimators=10, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8156

Precisión para configuración actual: 0.8156

Evaluando configuración: n\_estimators=10, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=10, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8296

Precisión para configuración actual: 0.8296

Evaluando configuración: n\_estimators=50, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7944

Precisión para configuración actual: 0.7944

Evaluando configuración: n\_estimators=50, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.797

Precisión para configuración actual: 0.797

Evaluando configuración: n\_estimators=50, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8007

Precisión para configuración actual: 0.8007

Evaluando configuración: n\_estimators=50, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.777

Precisión para configuración actual: 0.777

Evaluando configuración: n\_estimators=50, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7734

Precisión para configuración actual: 0.7734

Evaluando configuración: n\_estimators=50, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7859

Precisión para configuración actual: 0.7859

Evaluando configuración: n\_estimators=50, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7601

Precisión para configuración actual: 0.7601

Evaluando configuración: n\_estimators=50, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.7806

Precisión para configuración actual: 0.7806

Evaluando configuración: n\_estimators=50, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.811

Precisión para configuración actual: 0.811

Evaluando configuración: n\_estimators=50, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8684

Precisión para configuración actual: 0.8684

Nueva mejor precisión encontrada: 0.8684

Nuevos mejores parámetros: {'n\_estimators': 50, 'max\_depth': 5, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=50, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8634

Precisión para configuración actual: 0.8634

Evaluando configuración: n\_estimators=50, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8583

Precisión para configuración actual: 0.8583

Evaluando configuración: n\_estimators=50, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8702

Precisión para configuración actual: 0.8702

Nueva mejor precisión encontrada: 0.8702

Nuevos mejores parámetros: {'n\_estimators': 50, 'max\_depth': 5, 'min\_samples\_split': 5, 'min\_samples\_leaf': 1, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=50, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8686

Precisión para configuración actual: 0.8686

Evaluando configuración: n\_estimators=50, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8707

Precisión para configuración actual: 0.8707

Nueva mejor precisión encontrada: 0.8707

Nuevos mejores parámetros: {'n\_estimators': 50, 'max\_depth': 5, 'min\_samples\_split': 5, 'min\_samples\_leaf': 4, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=50, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8646

Precisión para configuración actual: 0.8646

Evaluando configuración: n\_estimators=50, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8723

Precisión para configuración actual: 0.8723

Nueva mejor precisión encontrada: 0.8723

Nuevos mejores parámetros: {'n\_estimators': 50, 'max\_depth': 5, 'min\_samples\_split': 10, 'min\_samples\_leaf': 2, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=50, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8644

Precisión para configuración actual: 0.8644

Evaluando configuración: n\_estimators=50, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9123

Precisión para configuración actual: 0.9123

Nueva mejor precisión encontrada: 0.9123

Nuevos mejores parámetros: {'n\_estimators': 50, 'max\_depth': 7, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=50, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.91

Precisión para configuración actual: 0.91

Evaluando configuración: n\_estimators=50, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9021

Precisión para configuración actual: 0.9021

Evaluando configuración: n\_estimators=50, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9084

Precisión para configuración actual: 0.9084

Evaluando configuración: n\_estimators=50, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9086

Precisión para configuración actual: 0.9086

Evaluando configuración: n\_estimators=50, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9088

Precisión para configuración actual: 0.9088

Evaluando configuración: n\_estimators=50, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9115

Precisión para configuración actual: 0.9115

Evaluando configuración: n\_estimators=50, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9101

Precisión para configuración actual: 0.9101

Evaluando configuración: n\_estimators=50, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=50, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9059

Precisión para configuración actual: 0.9059

Evaluando configuración: n\_estimators=100, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8386

Precisión para configuración actual: 0.8386

Evaluando configuración: n\_estimators=100, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.818

Precisión para configuración actual: 0.818

Evaluando configuración: n\_estimators=100, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=3, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8213

Precisión para configuración actual: 0.8213

Evaluando configuración: n\_estimators=100, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8358

Precisión para configuración actual: 0.8358

Evaluando configuración: n\_estimators=100, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8378

Precisión para configuración actual: 0.8378

Evaluando configuración: n\_estimators=100, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=3, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8273

Precisión para configuración actual: 0.8273

Evaluando configuración: n\_estimators=100, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8371

Precisión para configuración actual: 0.8371

Evaluando configuración: n\_estimators=100, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8308

Precisión para configuración actual: 0.8308

Evaluando configuración: n\_estimators=100, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=3, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8288

Precisión para configuración actual: 0.8288

Evaluando configuración: n\_estimators=100, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8915

Precisión para configuración actual: 0.8915

Evaluando configuración: n\_estimators=100, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8895

Precisión para configuración actual: 0.8895

Evaluando configuración: n\_estimators=100, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=5, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8955

Precisión para configuración actual: 0.8955

Evaluando configuración: n\_estimators=100, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8932

Precisión para configuración actual: 0.8932

Evaluando configuración: n\_estimators=100, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8884

Precisión para configuración actual: 0.8884

Evaluando configuración: n\_estimators=100, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=5, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8926

Precisión para configuración actual: 0.8926

Evaluando configuración: n\_estimators=100, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.8982

Precisión para configuración actual: 0.8982

Evaluando configuración: n\_estimators=100, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.898

Precisión para configuración actual: 0.898

Evaluando configuración: n\_estimators=100, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=5, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.887

Precisión para configuración actual: 0.887

Evaluando configuración: n\_estimators=100, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9325

Precisión para configuración actual: 0.9325

Nueva mejor precisión encontrada: 0.9325

Nuevos mejores parámetros: {'n\_estimators': 100, 'max\_depth': 7, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=100, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9343

Precisión para configuración actual: 0.9343

Nueva mejor precisión encontrada: 0.9343

Nuevos mejores parámetros: {'n\_estimators': 100, 'max\_depth': 7, 'min\_samples\_split': 2, 'min\_samples\_leaf': 2, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=100, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=7, min\_samples\_split=2, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9318

Precisión para configuración actual: 0.9318

Evaluando configuración: n\_estimators=100, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9293

Precisión para configuración actual: 0.9293

Evaluando configuración: n\_estimators=100, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9302

Precisión para configuración actual: 0.9302

Evaluando configuración: n\_estimators=100, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=7, min\_samples\_split=5, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9282

Precisión para configuración actual: 0.9282

Evaluando configuración: n\_estimators=100, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=1, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9335

Precisión para configuración actual: 0.9335

Evaluando configuración: n\_estimators=100, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=2, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9367

Precisión para configuración actual: 0.9367

Nueva mejor precisión encontrada: 0.9367

Nuevos mejores parámetros: {'n\_estimators': 100, 'max\_depth': 7, 'min\_samples\_split': 10, 'min\_samples\_leaf': 2, 'max\_features': 'log2'}

Evaluando configuración: n\_estimators=100, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando AdaboostClassifier con n\_estimators=100, max\_depth=7, min\_samples\_split=10, min\_samples\_leaf=4, max\_features=log2

Entrenando el clasificador...

Prediciendo en el conjunto de prueba...

Precisión obtenida: 0.9335

Precisión para configuración actual: 0.9335

Mejor precisión obtenida: 0.9367

Mejores parámetros encontrados: {'n\_estimators': 100, 'max\_depth': 7, 'min\_samples\_split': 10, 'min\_samples\_leaf': 2, 'max\_features': 'log2'}

Mejores parámetros recomendados: {'n\_estimators': 100, 'max\_depth': 7, 'min\_samples\_split': 10, 'min\_samples\_leaf': 2, 'max\_features': 'log2'} con tasa de acierto=0.9367

FIN

>>>

## 2C :

Impresión por pantalla:  
  
NOTA: Solo se pone un trozo porque si pongo todo a lo mejor salen 200 pags de memoria.

Evaluando configuración: capas=[512, 256], activación=tanh, optimizador=sgd, batch\_size=64, learning\_rate=0.001

Construyendo y entrenando MLP con capas=[512, 256], activación=tanh, optimizador=sgd, batch\_size=64, epochs=10, learning\_rate=0.001

Epoch 1/10

938/938 [==============================] - 4s 4ms/step - loss: 1.5086 - accuracy: 0.6236 - val\_loss: 1.0011 - val\_accuracy: 0.7884

Epoch 2/10

938/938 [==============================] - 5s 5ms/step - loss: 0.8462 - accuracy: 0.8131 - val\_loss: 0.7006 - val\_accuracy: 0.8424

Epoch 3/10

938/938 [==============================] - 5s 5ms/step - loss: 0.6542 - accuracy: 0.8459 - val\_loss: 0.5773 - val\_accuracy: 0.8647

Epoch 4/10

938/938 [==============================] - 5s 5ms/step - loss: 0.5622 - accuracy: 0.8612 - val\_loss: 0.5092 - val\_accuracy: 0.8752

Epoch 5/10

938/938 [==============================] - 5s 6ms/step - loss: 0.5071 - accuracy: 0.8718 - val\_loss: 0.4657 - val\_accuracy: 0.8830

Epoch 6/10

938/938 [==============================] - 5s 5ms/step - loss: 0.4700 - accuracy: 0.8783 - val\_loss: 0.4353 - val\_accuracy: 0.8877

Epoch 7/10

938/938 [==============================] - 5s 5ms/step - loss: 0.4431 - accuracy: 0.8835 - val\_loss: 0.4125 - val\_accuracy: 0.8930

Epoch 8/10

938/938 [==============================] - 4s 5ms/step - loss: 0.4226 - accuracy: 0.8874 - val\_loss: 0.3951 - val\_accuracy: 0.8951

Epoch 9/10

938/938 [==============================] - 5s 6ms/step - loss: 0.4064 - accuracy: 0.8904 - val\_loss: 0.3810 - val\_accuracy: 0.8974

Epoch 10/10

938/938 [==============================] - 5s 6ms/step - loss: 0.3931 - accuracy: 0.8928 - val\_loss: 0.3693 - val\_accuracy: 0.8999

Precisión del MLP en el conjunto de prueba: 0.8999

Evaluando configuración: capas=[512, 256], activación=tanh, optimizador=sgd, batch\_size=64, learning\_rate=0.01

Construyendo y entrenando MLP con capas=[512, 256], activación=tanh, optimizador=sgd, batch\_size=64, epochs=10, learning\_rate=0.01

Epoch 1/10

938/938 [==============================] - 4s 4ms/step - loss: 0.6331 - accuracy: 0.8379 - val\_loss: 0.3731 - val\_accuracy: 0.8976

Epoch 2/10

938/938 [==============================] - 5s 5ms/step - loss: 0.3529 - accuracy: 0.9008 - val\_loss: 0.3120 - val\_accuracy: 0.9121

Epoch 3/10

938/938 [==============================] - 5s 6ms/step - loss: 0.3110 - accuracy: 0.9109 - val\_loss: 0.2872 - val\_accuracy: 0.9197

Epoch 4/10

938/938 [==============================] - 5s 6ms/step - loss: 0.2878 - accuracy: 0.9173 - val\_loss: 0.2688 - val\_accuracy: 0.9239

Epoch 5/10

938/938 [==============================] - 5s 6ms/step - loss: 0.2708 - accuracy: 0.9230 - val\_loss: 0.2561 - val\_accuracy: 0.9278

Epoch 6/10

938/938 [==============================] - 5s 5ms/step - loss: 0.2572 - accuracy: 0.9271 - val\_loss: 0.2456 - val\_accuracy: 0.9313

Epoch 7/10

938/938 [==============================] - 5s 6ms/step - loss: 0.2447 - accuracy: 0.9304 - val\_loss: 0.2366 - val\_accuracy: 0.9336

Epoch 8/10

938/938 [==============================] - 5s 6ms/step - loss: 0.2334 - accuracy: 0.9337 - val\_loss: 0.2261 - val\_accuracy: 0.9374

Epoch 9/10

938/938 [==============================] - 6s 6ms/step - loss: 0.2228 - accuracy: 0.9370 - val\_loss: 0.2170 - val\_accuracy: 0.9380

Epoch 10/10

938/938 [==============================] - 5s 6ms/step - loss: 0.2128 - accuracy: 0.9404 - val\_loss: 0.2080 - val\_accuracy: 0.9406

Precisión del MLP en el conjunto de prueba: 0.9406

Mejor precisión obtenida: 0.9817000031471252

Mejores parámetros encontrados: {'layers': [512, 256], 'activation': 'relu', 'optimizer': 'adam', 'batch\_size': 32, 'learning\_rate': 0.001}

Mejores parámetros recomendados para MLP: {'layers': [512, 256], 'activation': 'relu', 'optimizer': 'adam', 'batch\_size': 32, 'learning\_rate': 0.001} con tasa de acierto=0.9817

FIN

## 2D:

Impresión por pantalla:  
  
NOTA: Al igual que en el anterior, la impresión es solo un trozo del total  
  
Evaluando configuración: conv\_layers=[(64, 3, 2)], dense\_layers=[256], activación=relu, optimizador=adam, batch\_size=32, learning\_rate=0.001

Construyendo y entrenando CNN con capas\_conv=[(64, 3, 2)], capas\_dense=[256], activación=relu, optimizador=adam, batch\_size=32, epochs=3, learning\_rate=0.001

Epoch 1/3

1875/1875 [==============================] - 52s 28ms/step - loss: 0.1275 - accuracy: 0.9610 - val\_loss: 0.0479 - val\_accuracy: 0.9836

Epoch 2/3

1875/1875 [==============================] - 52s 28ms/step - loss: 0.0419 - accuracy: 0.9869 - val\_loss: 0.0379 - val\_accuracy: 0.9872

Epoch 3/3

1875/1875 [==============================] - 52s 28ms/step - loss: 0.0235 - accuracy: 0.9925 - val\_loss: 0.0397 - val\_accuracy: 0.9868

Precisión de la CNN en el conjunto de prueba: 0.9868

Evaluando configuración: conv\_layers=[(64, 3, 2), (32, 3, 2)], dense\_layers=[128], activación=relu, optimizador=adam, batch\_size=32, learning\_rate=0.001

Construyendo y entrenando CNN con capas\_conv=[(64, 3, 2), (32, 3, 2)], capas\_dense=[128], activación=relu, optimizador=adam, batch\_size=32, epochs=3, learning\_rate=0.001

Epoch 1/3

1875/1875 [==============================] - 16s 8ms/step - loss: 0.1283 - accuracy: 0.9609 - val\_loss: 0.0516 - val\_accuracy: 0.9834

Epoch 2/3

1875/1875 [==============================] - 15s 8ms/step - loss: 0.0413 - accuracy: 0.9871 - val\_loss: 0.0385 - val\_accuracy: 0.9873

Epoch 3/3

1875/1875 [==============================] - 15s 8ms/step - loss: 0.0297 - accuracy: 0.9911 - val\_loss: 0.0299 - val\_accuracy: 0.9897

Precisión de la CNN en el conjunto de prueba: 0.9897

Nueva mejor precisión encontrada: 0.9897000193595886

Nuevos mejores parámetros: {'conv\_layers': [(64, 3, 2), (32, 3, 2)], 'dense\_layers': [128], 'activation': 'relu', 'optimizer': 'adam', 'batch\_size': 32, 'learning\_rate': 0.001}

Evaluando configuración: conv\_layers=[(64, 3, 2), (32, 3, 2)], dense\_layers=[256], activación=relu, optimizador=adam, batch\_size=32, learning\_rate=0.001

Construyendo y entrenando CNN con capas\_conv=[(64, 3, 2), (32, 3, 2)], capas\_dense=[256], activación=relu, optimizador=adam, batch\_size=32, epochs=3, learning\_rate=0.001

Epoch 1/3

1875/1875 [==============================] - 17s 9ms/step - loss: 0.1214 - accuracy: 0.9627 - val\_loss: 0.0488 - val\_accuracy: 0.9838

Epoch 2/3

1875/1875 [==============================] - 16s 9ms/step - loss: 0.0424 - accuracy: 0.9867 - val\_loss: 0.0334 - val\_accuracy: 0.9888

Epoch 3/3

1875/1875 [==============================] - 16s 9ms/step - loss: 0.0282 - accuracy: 0.9907 - val\_loss: 0.0292 - val\_accuracy: 0.9905

Precisión de la CNN en el conjunto de prueba: 0.9905

Nueva mejor precisión encontrada: 0.9904999732971191

Nuevos mejores parámetros: {'conv\_layers': [(64, 3, 2), (32, 3, 2)], 'dense\_layers': [256], 'activation': 'relu', 'optimizer': 'adam', 'batch\_size': 32, 'learning\_rate': 0.001}

Mejor precisión obtenida: 0.9904999732971191

Mejores parámetros encontrados: {'conv\_layers': [(64, 3, 2), (32, 3, 2)], 'dense\_layers': [256], 'activation': 'relu', 'optimizer': 'adam', 'batch\_size': 32, 'learning\_rate': 0.001}

Mejores parámetros recomendados para CNN: {'conv\_layers': [(64, 3, 2), (32, 3, 2)], 'dense\_layers': [256], 'activation': 'relu', 'optimizer': 'adam', 'batch\_size': 32, 'learning\_rate': 0.001} con tasa de acierto=0.9905

## 2E:

|  |  |  |
| --- | --- | --- |
| TAREA | TIEMPO | METODO |
| 1A |  |  |
| 1B |  |  |
| 1C |  |  |
| 1D |  |  |
| 1E |  |  |
| 2A |  |  |
| 2B |  |  |
| 2C |  |  |
| 2D |  |  |

## BIBLIOGRAFIAAAAAAAAAA

Experimenta con estos parámetros para mejorar en lo posible la tasa de acierto. Implementa una función invocada desde el main que muestre los resultados de esa experimentación. Si lo crees necesario, puedes crear más de una función, con diferentes configuraciones de parámetros o estrategias de variación de los mismos. Dime con que parametros son con los que tengo que experimentar y cuales me recomiendas

ChatGPT

Para experimentar con los parámetros del clasificador débil DecisionTreeClassifier en el contexto de Adaboost, es importante entender qué parámetros pueden influir significativamente en el rendimiento del modelo. Aquí te dejo una lista de parámetros relevantes y algunas recomendaciones para tus experimentos:

Parámetros de DecisionTreeClassifier

1. **max\_depth**: La profundidad máxima del árbol. Controla hasta qué punto el árbol puede crecer.
   * **Valores recomendados para experimentar**: [3, 5, 7]
2. **min\_samples\_split**: El número mínimo de muestras requeridas para dividir un nodo.
   * **Valores recomendados para experimentar**: [2, 10, 20]
3. **min\_samples\_leaf**: El número mínimo de muestras que debe tener un nodo hoja.
   * **Valores recomendados para experimentar**: [1, 5, 10]

Parámetros de AdaBoostClassifier

1. **n\_estimators**: El número de clasificadores débiles a entrenar.
   * **Valores recomendados para experimentar**: [50, 100, 150]
2. **learning\_rate**: Una tasa de aprendizaje que reduce la contribución de cada clasificador. Este valor se usa para equilibrar el trade-off entre el número de estimadores y la tasa de aprendizaje.
   * **Valores recomendados para experimentar**: [0.01, 0.1, 1.0]