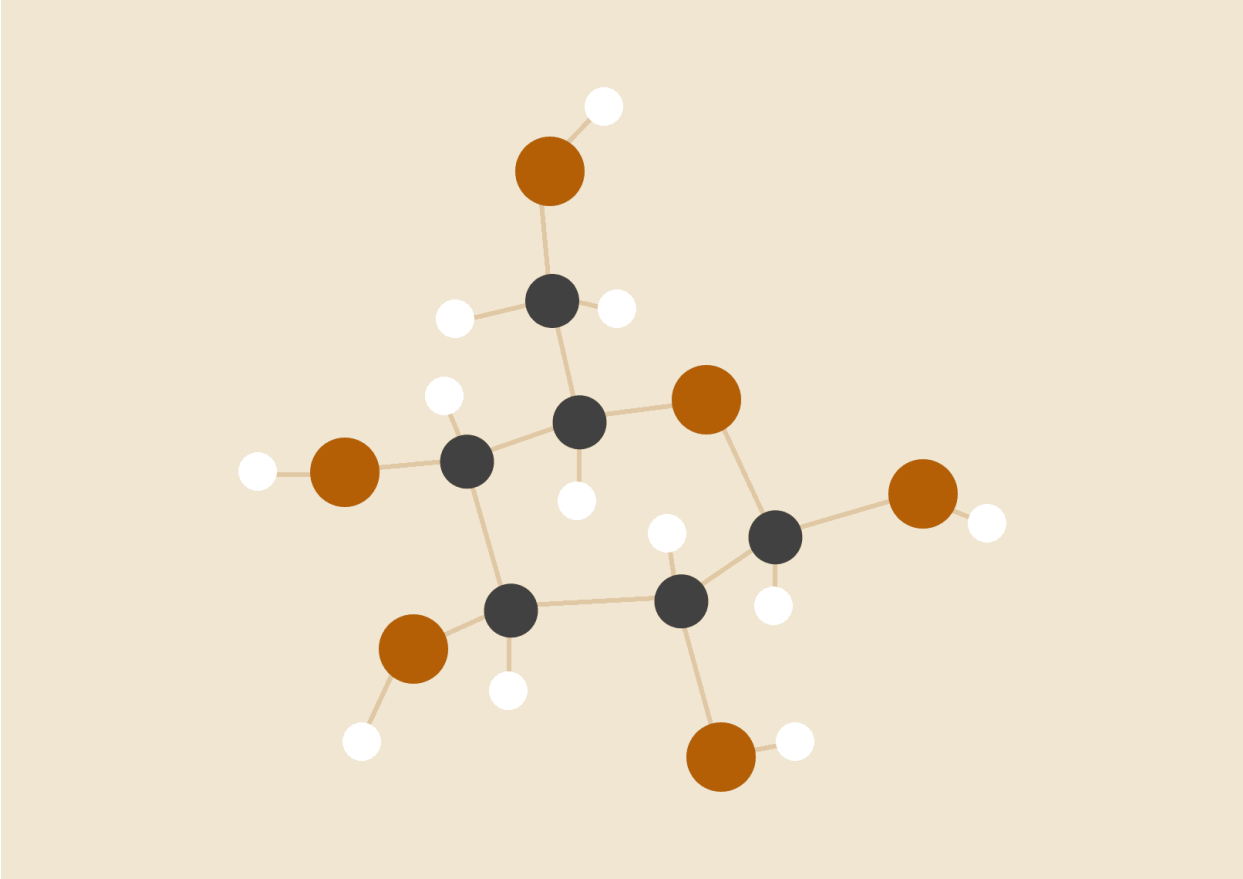
Laboratorio de Datos

TMNIST: imagenes y tipografias



MinaData

Diego Moros, Nicole Britos y Maria Obregon

12 de noviembre de 2024
Ciencia de datos

Introducción

El objetivo de este trabajo es analizar y comprender el conjunto de datos "TMNIST", que contiene dígitos escritos utilizando diferentes tipos de letra. Este conjunto de datos está inspirado en el conocido MNIST, pero en lugar de incluir dígitos escritos a mano, las imágenes están generadas a partir de diversas fuentes tipográficas de Google.

El análisis de este conjunto de datos busca comprender patrones en las imágenes tipográficas de dígitos, que permitirían avanzar en técnicas de clasificación aplicables a diversos contextos, como la detección automática de caracteres en distintos formatos. En primer lugar, realizaremos un análisis exploratorio para comprender las características principales del conjunto de datos. Estudiaremos los datos en busca de información relevante que nos permita identificar patrones y relaciones entre ellos. Para ello, aplicaremos diversos gráficos y técnicas de visualización que nos ayudarán a comprender la estructura de los datos y a identificar patrones clave.

Una vez que hayamos analizado la información, buscaremos establecer modelos predictivos que nos permitan identificar y justificar estos patrones. Emplearemos técnicas de clasificación, como el algoritmo KNN y árboles de decisión, que nos permitirán no solo diferenciar dígitos, sino también comprender qué características visuales resultan más relevantes en este contexto. Finalmente, nuestro objetivo es desarrollar un modelo que sea capaz de predecir correctamente los dígitos en función de las características de los píxeles, utilizando los patrones identificados en el análisis previo.

Análisis exploratorio

Visualización y Comprensión

¿Cómo son nuestros datos?

Antes de iniciar el análisis, es esencial comprender qué tipo de datos estamos utilizando y la información que nos proporciona el conjunto TMNIST. Como se observa en la “Tabla 1”, este conjunto de datos representa dígitos mediante diferentes tipografías y valores de píxeles. Cada fila consta de 786 elementos: 785 de ellos son valores cuantitativos (int64) y 1 es de tipo categórico (str). La primera columna indica el nombre de la fuente tipográfica empleada; la segunda columna muestra la etiqueta correspondiente al dígito (un número del 0 al 9); y desde la tercera hasta la columna 786 se encuentran los valores de píxeles en escala de grises (de 0 a 255), que forman una imagen de 28x28 píxeles.

names	labels	1	2	...	784
GrandHotel-Regular	2	0	0	...	0
EncodeSansCondensed-Medium	8	0	0	...	0
Varela-Regular	4	0	0	...	0
ArefRuqaa-Bold	3	0	0	...	0
...	...	...	...	...	...
BigShouldersStencilDisplay-Black	0	0	0	...	0
Gabriela-Regular	1	0	0	...	0
Ovo-Regular	6	0	0	...	0

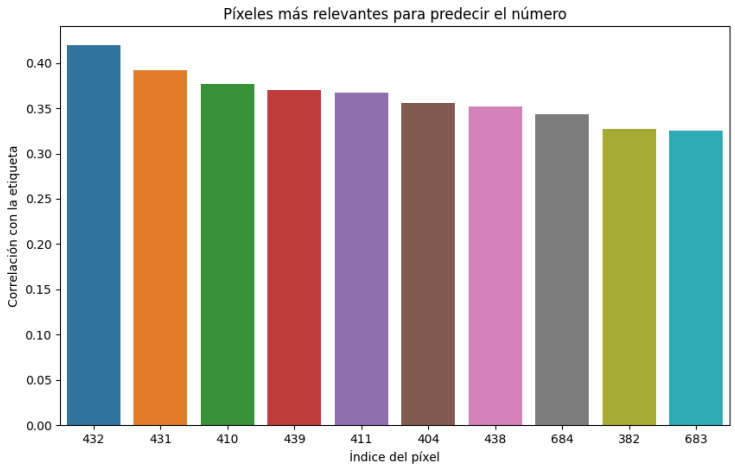
Tabla 1: TMNIST_Data.

¿Cómo se comportan nuestros datos?

Una vez comprendida la estructura de los datos, analizaremos su comportamiento. Dado que estamos tratando con imágenes, la exploración se complica debido a la alta dimensionalidad de los datos (784 atributos de píxeles por imagen). Por lo tanto, es crucial aplicar técnicas adecuadas para identificar relaciones entre los píxeles y su relevancia en la clasificación de los dígitos.

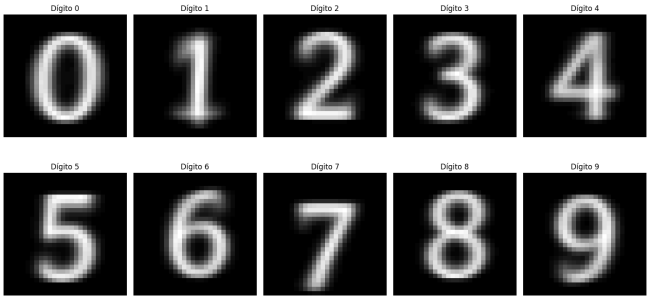
Inicialmente, evaluaremos la importancia relativa de cada píxel. Esto puede ayudarnos a reducir la cantidad de datos a considerar y a enfocarnos en aquellos píxeles que ofrecen mayor información sobre el dígito representado. Para ello, realizaremos un análisis de correlación entre los valores de píxeles y la etiqueta de la imagen (dígito), con el objetivo de identificar los atributos que tienen una mayor conexión con la clase.

Figura 1: Pixeles más relevantes.



Como se observa en la figura 1, algunos píxeles presentan una correlación más fuerte con la etiqueta de la imagen, lo que permite identificar las áreas de mayor relevancia para el modelo predictivo. Este análisis preliminar facilita enfocar el modelo en los atributos más informativos para la clasificación de los dígitos.

F***igura 2: Pixeles más relevantes.***



Complementando esta observación, en la “Figura 2” se muestra el promedio de los valores de píxeles para cada dígito, donde se nota que cada dígito tiende a concentrarse en zonas específicas de la imagen. Esto refuerza la idea de que ciertos píxeles son más determinantes a la hora de representar cada dígito, lo que sugiere la posibilidad de descartar algunos atributos menos relevantes en futuras simplificaciones del modelo.

Asimismo, observamos que algunos dígitos son más fáciles de diferenciar entre sí, debido a sus características visuales. En las figuras 3 y 4, por ejemplo, se puede ver que el dígito 1 se diferencia claramente del 3 debido a su composición más sencilla y menor uso de píxeles, mientras que la comparación entre el 3 y el 8 es menos clara, probablemente debido a que ambos dígitos ocupan una cantidad similar de píxeles y comparten ciertas similitudes en su estructura. Esto sugiere que la forma y densidad de los píxeles son factores importantes para distinguir entre ciertos dígitos.

Figura 3: Comparación visual (1 y 3).

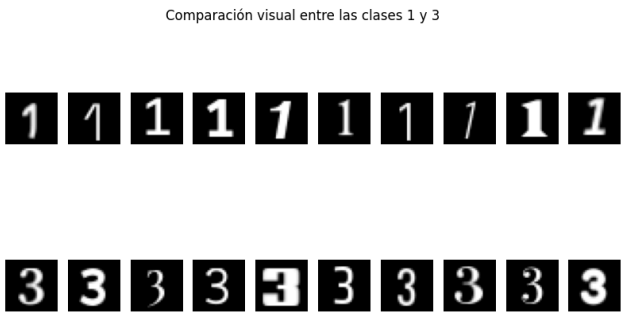
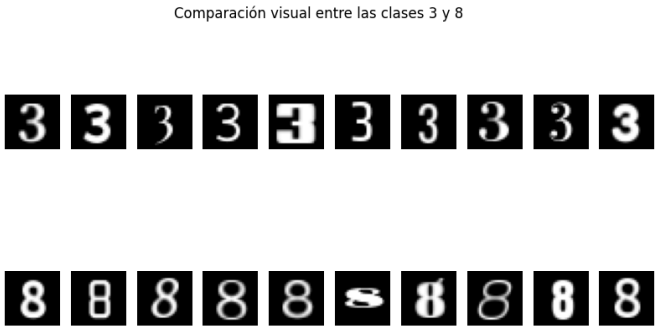
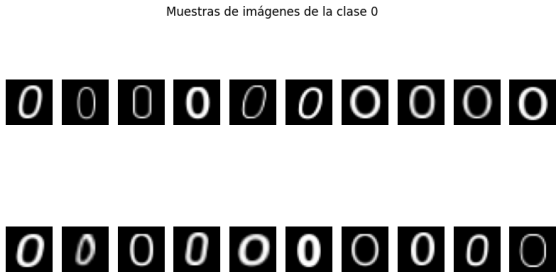


Figura 4: Comparación visual (3 y 8).



Para profundizar en el análisis, decidimos estudiar el comportamiento dentro de una misma etiqueta/clase, tomamos el dígito 0 como ejemplo y analizamos su comportamiento en distintas tipografías. En la figura 4, vemos cómo, a pesar de las variaciones en el estilo de cada fuente, la forma básica del 0 se mantiene reconocible. Esto nos sugiere que, aunque ciertos píxeles cambian entre tipografías, las características clave que identifican al dígito 0 siguen siendo consistentes.

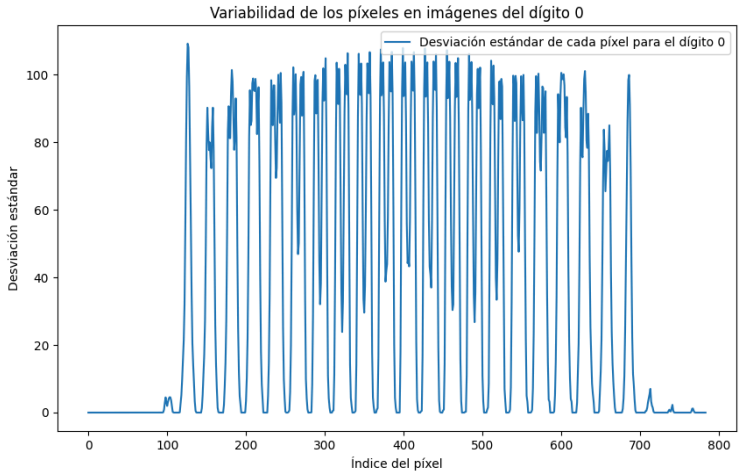
Figura 4: Comparación visual (0).



Luego, en la “Figura 5”, analizamos la variabilidad de los píxeles dentro de la clase del 0, identificando cuáles son más propensos al cambio y cuáles permanecen constantes. Este análisis nos ayuda a ver cómo algunas áreas son esenciales para preservar la percepción del dígito, incluso si algunos detalles cambian entre versiones. La estabilidad de estos "píxeles fijos" es importante para el reconocimiento, ya que permite que el dígito 0 sea identificado como tal, aun en presencia de variaciones tipográficas.

Este análisis nos muestra que, aunque las instancias del mismo dígito no son idénticas, presentan una alta similitud interna que permite su clasificación uniforme. La constancia de píxeles en áreas clave podría orientar un modelo predictivo, eliminando aquellos píxeles menos relevantes.

Figura 5: Variabilidad de los pixeles (0).



Modelado y predicción

Clasificación binaria

Una vez realizado el análisis exploratorio y extraídas algunas conclusiones, buscamos establecer un modelo que nos permita predecir alguno de los dígitos utilizando los datos de nuestro conjunto. Para ello, implementaremos una clasificación binaria, una técnica de aprendizaje supervisado cuyo objetivo es clasificar entre dos clases.

En este caso, filtraremos nuestro conjunto de datos para trabajar con dos categorías o valores, seleccionando los dígitos 1 y 0 como etiquetas. Posteriormente, procederemos a dividir los datos en dos subconjuntos: uno para entrenar (usando train_test_split) y otro para evaluar (utilizando test_size). La división será en una proporción de 70/30, donde el 70% de los datos se usará para entrenar el modelo y el 30% restante se destinará a comprobar la precisión de la predicción.

Para realizar la clasificación binaria, utilizaremos el modelo KNN (K-Vecinos Más Cercanos), ajustando distintos atributos y parámetros para que el modelo se adapte a nuestro problema y podamos evaluar su efectividad.

Las métricas que utilizaremos para evaluar el rendimiento del modelo incluyen:

- Exactitud: Proporción general de predicciones correctas.
- Precisión y exhaustividad: Métricas clave para evaluar los errores en ambas clases.

Probamos el modelo con **k=3**, una elección común que busca equilibrar simplicidad y precisión, aunque en la siguiente fase experimentaremos con otros valores de k.

Figura 6: Matriz de confusión de cada uno de los atributos evaluados

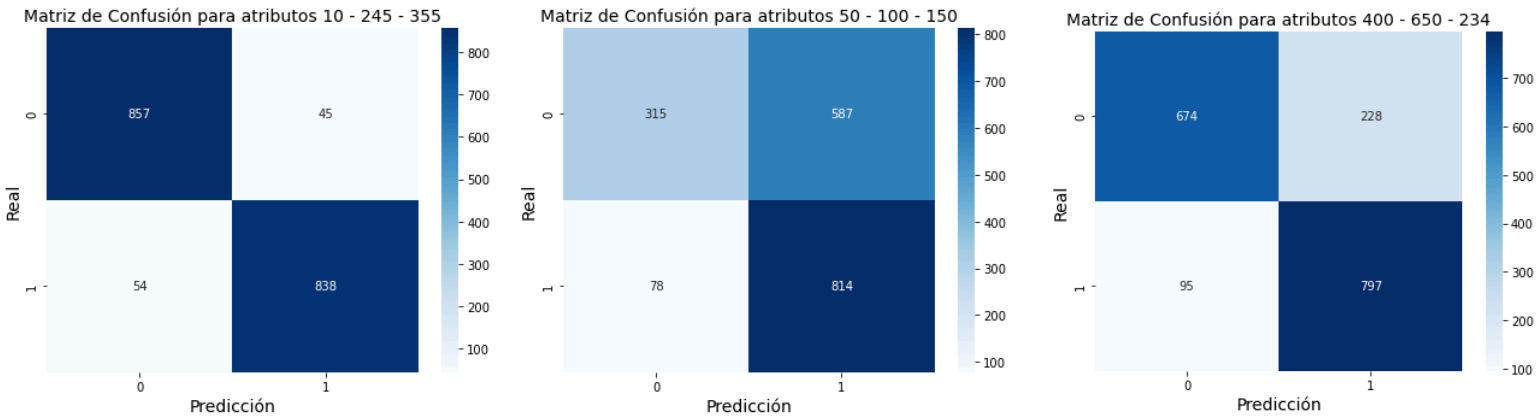


Figura 7: Rango de exactitud de KNN para cada subconjunto de atributos

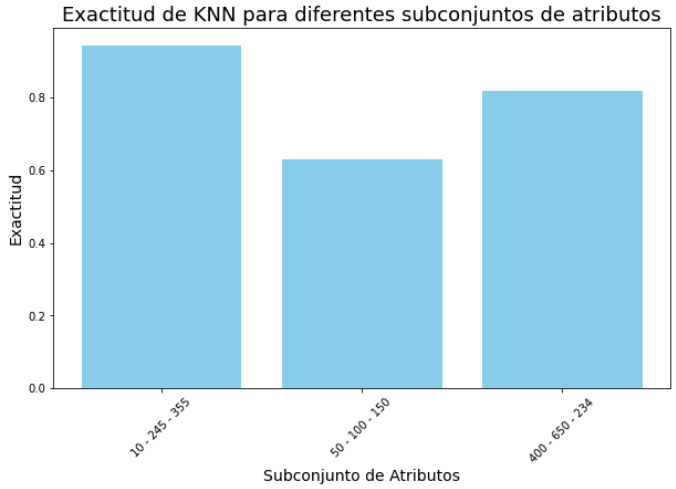
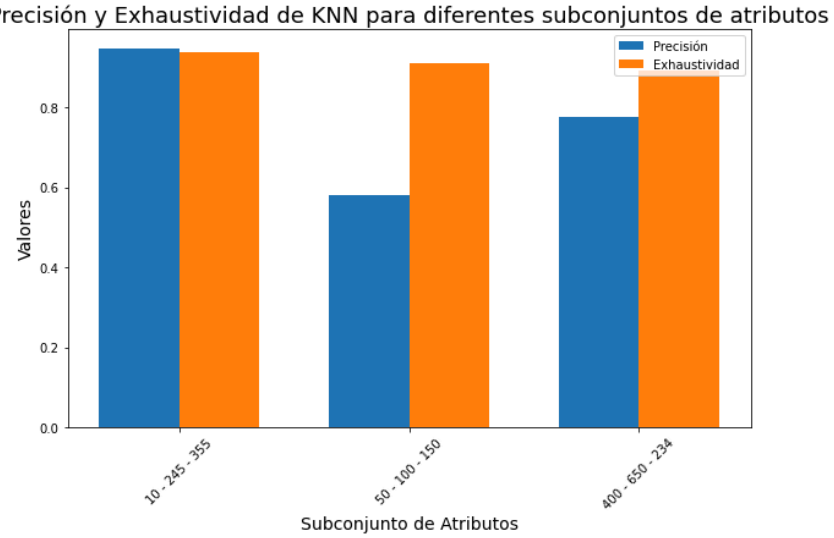


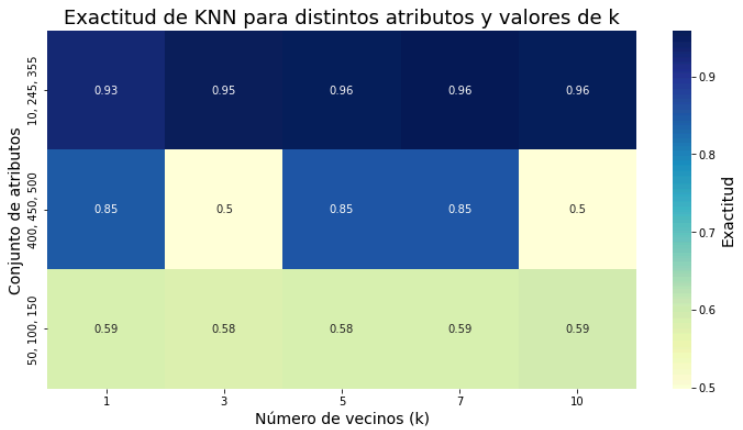
Figura 8: Rango de precisión y exhaustividad de KNN para cada subconjunto de atributos



Como se observa en las Figuras 6, 7 y 8, el modelo muestra una gran exactitud y precisión para cada uno de los subconjuntos mencionados. Esto sugiere que el modelo se ajusta bastante bien a nuestro problema, lo cual podemos verificar probando con diferentes valores de k para encontrar el equilibrio ideal.

- **Valores de k usados**: (1, 3, 5, 7, 10)

Figura 9: Rango de exactitud de KNN para distintos de atributos y para distintos valores de k



La combinación de atributos es clave en el rendimiento del modelo KNN. Los mejores resultados se lograron con los atributos `['10', '245', '355']`, independientemente del tamaño del subconjunto, y con valores de k relativamente bajos (1 o 3), como se observa en la Figura 9. Esto sugiere que, para este conjunto de datos específico, estos atributos y valores de k son óptimos, y que agregar más atributos o usar valores de k mayores no mejora el modelo. Estos resultados resaltan la importancia de seleccionar los atributos adecuados y ajustar k de manera correcta en los modelos KNN para maximizar la exactitud y evitar clasificaciones incorrectas.

Modelado y predicción

Clasificación multiclase

Para resolver el problema de clasificación de dígitos en el conjunto de datos TMNIST, se plantea un modelo de clasificación multiclase. Esta técnica permite asignar una de las diez posibles etiquetas (0 a 9) a cada imagen en función de sus características visuales.

El primer paso en la construcción del modelo es dividir los datos en dos subconjuntos: un conjunto de desarrollo (dev) y un conjunto de validación (held-out). Esta división permite entrenar el modelo en un conjunto y evaluarlo en otro, asegurando que la evaluación sea

objetiva y no esté influenciada por los datos utilizados durante el entrenamiento. Utilizamos la función ***train_test_split*** con la opción ***stratify,*** lo que preserva la distribución de cada clase en ambos subconjuntos, un paso crucial para evitar sesgos en la clasificación.

Para abordar la clasificación de los dígitos, utilizamos un modelo de **árbol de decisión**, que es una técnica de aprendizaje supervisado que clasifica instancias dividiendo los datos en decisiones binarias sucesivas. Este enfoque es útil en clasificación multiclase debido a su capacidad para generar reglas de decisión comprensibles y jerárquicas.

A fin de optimizar el rendimiento del árbol de decisión, probamos distintas configuraciones de hiper parámetros:

Profundidad máxima del árbol (***max_depth***), que controla el nivel de detalle en la clasificación y previene el sobreajuste cuando se limita entre valores bajos (1) y altos (10).

Muestras mínimas para dividir un nodo (***min_samples_split***), que evita que el modelo realice divisiones basadas en pocos datos, lo que puede resultar en reglas poco generalizables.

Muestras mínimas en cada hoja (***min_samples_leaf***), que establece el número mínimo de muestras en cada rama final del árbol para mantener la robustez del modelo.

Criterio de impureza (***criterion***), que determina la calidad de una división según la pureza de las clases resultantes. Se probaron los criterios ***gini*** y ***entropy*** para identificar el que mejor se adapta a nuestros datos.

Algunas de las permutaciones usadas fueron:

- max_depth_values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
- min_samples_split_values = [2, 5, 10]
- min_samples_leaf_values = [1, 2, 4]
- criterion_values = ['gini', 'entropy']

A cada uno se evaluó precision, recall, f1 y se generó la matriz de confusión correspondiente, como se puede ver en la “Tabla 2”

max_depth	nin_samples_spl	min_samples_lea	criterion	fold	precision	recall	f1	confusion_matrix
1	2	1	gini	1	0.9552845528	0.9832635983	0.9690721649	[[456 22 0 0 0 0 0 0 0 0] [8 470 0 0 0 0 0 0 0 0] [86 392 0 0 0 0 0 0 0 0] [390 89 0 0 0 0 0 0 0 0] [36 442 0 0 0 0 0 0 0 0] [433 46 0 0 0 0 0 0 0 0] [461 17 0 0 0 0 0 0 0 0] [38 441 0 0 0 0 0 0 0 0] [446 33 0 0 0 0 0 0 0 0] [109 369 0 0 0 0 0 0 0 0]]
...	...	...	...	...	...	...	...	...
10	10	4	entropy	5	0.9978021978	1	0.99889989	[[447 1 5 2 3 3 3 0 11 4] [0 454 8 2 3 0 2 8 0 1] [1 8 449 5 3 0 1 1 9 1] [2 5 6 426 1 17 4 1 10 6] [7 4 1 2 447 3 4 2 4 4] [2 2 0 12 0 448 8 0 3 4] [7 1 3 1 3 16 441 0 5 1] [1 13 0 7 4 1 1 447 3 2] [4 3 9 4 3 5 8 0 441 1] [5 1 2 11 13 9 5 6 1 426]]

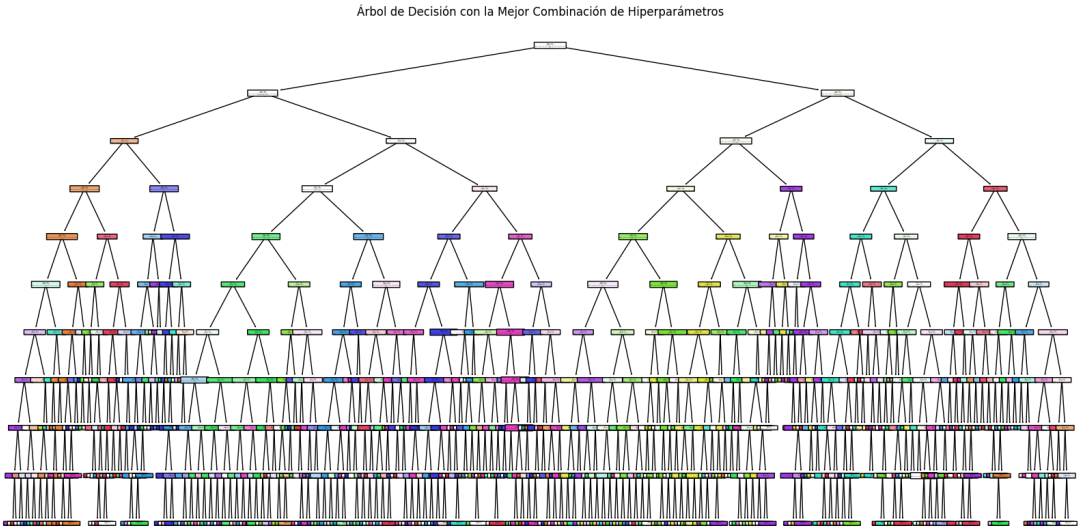
Tabla 2: resultados combinaciones.

Para elegir la mejor configuración de hiper parámetros, aplicamos validación cruzada con k-folding en el conjunto de desarrollo. Este método divide el conjunto dev en varias partes o “pliegues” y entrena el modelo en diferentes combinaciones de pliegues, permitiendo una evaluación más confiable del desempeño promedio. La validación cruzada asegura que nuestro modelo se evalúe en múltiples subconjuntos de datos, proporcionando una medida robusta del rendimiento del modelo en el conjunto de desarrollo y ayudando a seleccionar la mejor combinación de parámetros.

Al finalizar esta etapa, la mejor combinación de hiperparámetros con su árbol de decisión (Figura 10) fue:

- Profundidad máxima (**`max_depth`**): 10
- Mínimo número de muestras para dividir (**`min_samples_split`**): 2
- Mínimo número de muestras en cada hoja (**`min_samples_leaf`**): 2
- Criterio (**`criterion`**): **`gini`**

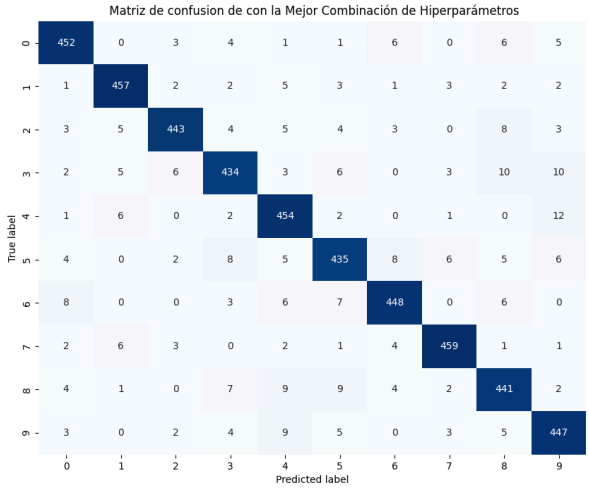
Figura 10: Árbol de decisión de la mejor combinación.



Usando la configuración indicada por el k-folding, el modelo nos arrojó los siguientes resultados en la validación cruzada y matriz de confusión (Figura 11):

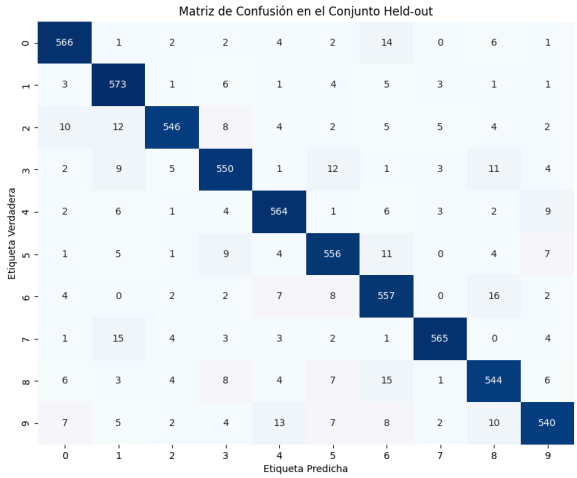
- Precisión: 0.999136
- Recall: 0.999135
- F1: 0.999135

Figura 11: Matriz de confusión con la mejor combinación.



Con la configuración óptima, el modelo se entrenó nuevamente en todo el conjunto dev y se evaluó en el conjunto held-out, logrando una exactitud del 92.99%. Este desempeño en un conjunto de datos no visto sugiere que el modelo generaliza bien, logrando reconocer correctamente la mayoría de los dígitos.

Figura 12: Matriz de confusión conjunto Held-Out.



El análisis de la matriz de confusión (figura 12) revela que el modelo es especialmente preciso en clases como los dígitos 1 y 4, pero también muestra algunos errores comunes, cómo clasificar el dígito 6 como un 8 en varias instancias. Estos errores reflejan la similitud visual entre ciertos dígitos y sugieren que mejoras futuras podrían incluir modelos más complejos, como redes neuronales, que pueden capturar características más detalladas en imágenes.