

# Laboratorio de Datos

## Trabajo Práctico 02:

### Clasificación de Kuzushiji-MNIST

---

9 de Noviembre de 2025

#### Grupo 1

Integrante	LU	Correo electrónico
Hildt, Lautaro	1166/24	<a href="mailto:lautarohildt43@gmail.com">lautarohildt43@gmail.com</a>
Fisanotti, Juan Bautista	1146/24	<a href="mailto:bautifisanotti@gmail.com">bautifisanotti@gmail.com</a>



**Facultad de Ciencias Exactas y Naturales**  
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

<http://www.exactas.uba.ar>

## Resumen

En este trabajo, realizamos un análisis del dataset de caracteres japoneses Kuzushiji-MNIST. Primero, hicimos un análisis exploratorio (EDA) para entender las características de las 10 clases, usando técnicas como mapas de calor de variabilidad y matrices de distancia euclidiana. Luego, abordamos el problema de clasificación binaria (Clases 4 vs. 5) usando un modelo K-Nearest Neighbors (KNN). Comparamos sistemáticamente distintas estrategias de selección de atributos y optimizamos los hiperparámetros N (atributos) y k (vecinos), logrando un Accuracy máximo de 0.932. Finalmente, para el problema multiclase (10 clases), usamos K-Fold Cross Validation para optimizar un Árbol de Decisión, analizando la profundidad y min\_samples\_split. El modelo final fue evaluado en un conjunto *held-out*, donde analizamos sus errores en detalle a través de la matriz de confusión.

## Introducción

La digitalización de textos históricos es un desafío enorme en el campo del *machine learning*. Particularmente, la escritura cursiva japonesa, conocida como "Kuzushiji", fue usada por siglos pero hoy es ilegible para la mayoría de los japoneses modernos. Reconocer estos caracteres es un problema de visión por computadora mucho más complejo que el de los dígitos manuscritos (MNIST).

Para impulsar la investigación en esta área, Clanuwat, et al. (2018) [1] crearon el dataset Kuzushiji-MNIST (KMnist), que usamos en este trabajo. Este dataset busca ser un reemplazo directo del MNIST, pero aumentando significativamente la dificultad.

El objetivo de este informe es replicar un flujo de trabajo de ciencia de datos sobre este dataset, explorando sus características y construyendo modelos de clasificación para resolver tanto un problema binario como uno multiclase.

## Objetivos

El objetivo principal de este trabajo es el desarrollo de modelos de *machine learning* para abordar el problema de clasificación de caracteres Kuzushiji.

Específicamente, buscamos investigar cómo afectan los distintos algoritmos, técnicas de evaluación e hiperparámetros al desarrollo y rendimiento de estos modelos. Para ello, nos enfocamos en:

- Explorar la estructura de los datos (EDA) para guiar la construcción de los modelos.
- Comparar el impacto de distintas estrategias de selección de atributos (inteligentes vs. aleatorias) en un modelo KNN.
- Analizar la sensibilidad de los modelos a distintos hiperparámetros (como k en KNN, max\_depth y min\_samples\_split en Árboles de Decisión).
- Implementar y comparar técnicas de validación (split simple y K-Fold Cross Validation) para la selección de un modelo robusto.
- Evaluar el rendimiento del modelo multiclase final en un conjunto de datos *held-out* para entender sus fortalezas y debilidades específicas.

## Metodología

En esta sección describiremos el "cómo" de nuestro trabajo: el dataset, las herramientas y las técnicas de validación que usamos.

## Dataset

Usamos el dataset completo kuzushiji\_full.csv, que consiste en 70,000 imágenes. Cada imagen es de 28x28 píxeles (784 atributos) y está etiquetada con una de las 10 clases (del 0 al 9), que representan 10 caracteres Hiragana diferentes.

## Herramientas

Todo el análisis fue realizado en Python. Las librerías principales fueron:

- **Pandas:** Para la carga y manipulación de los datos.
- **Matplotlib y Seaborn:** Para la generación de todos los gráficos.
- **Scikit-learn:** Para implementar todos los modelos (KNN, Árboles de Decisión), las técnicas de validación (K-Fold, train/test split) y el escalado de datos (StandardScaler).

## Modelos y Técnicas

**K-Nearest Neighbors (KNN):** Para el problema binario, usamos KNN. Este modelo clasifica un punto nuevo basándose en la "opinión" de sus k vecinos más cercanos. Por eso, fue crucial "escalar" los atributos con StandardScaler, para que los píxeles con mayor brillo no dominaran la decisión.

**Árboles de Decisión:** Para el problema multiclase, usamos Árboles de Decisión. Estos modelos aprenden una serie de "preguntas" (ej: ¿el píxel 50 es mayor a 0.5?) para dividir los datos. Sus hiperparámetros clave son max\_depth (qué tan profundo puede ser) y min\_samples\_split (cuántas muestras necesita para hacer una pregunta).

## Técnicas de Validación:

- **Train-Test Split:** Para ambos problemas, separamos los datos en entrenamiento y prueba. Es fundamental usar stratify=y para asegurarnos de que en ambos conjuntos haya una proporción balanceada de clases.
- **K-Fold Cross Validation (K=5):** Para la optimización del Árbol de Decisión, usamos K-Fold. En lugar de usar un solo set de validación (que puede dar resultados "suertudos"), K-Fold divide el set de desarrollo en 5 partes. Entrena 5 veces, usando cada parte como validación una vez, y nos da el *promedio* de rendimiento. Esto nos da una estimación mucho más robusta.
- **GridSearchCV:** Para automatizar la búsqueda de hiperparámetros (como la profundidad, el criterio y el split) junto con K-Fold, utilizamos la función GridSearchCV de Scikit-learn. Esta herramienta se encarga de probar sistemáticamente todas las combinaciones de hiperparámetros que le pasamos en una grilla. Para cada combinación, realiza la validación cruzada (K-Fold), calcula la métrica de rendimiento promedio, y finalmente nos devuelve la mejor combinación de parámetros encontrada.

## Métricas de Evaluación

Para este trabajo, elegimos el Accuracy (porcentaje de aciertos) como nuestra métrica principal. La razón de esta elección es que nuestros conjuntos de datos (tanto el binario de Clases 4/5 como el multiclase) estaban perfectamente balanceados.

En problemas con clases desbalanceadas, el Accuracy puede ser engañoso, ya que un modelo podría tener un 90% de aciertos simplemente "adivinando" siempre la clase mayoritaria. Pero en un set balanceado como el nuestro, el Accuracy es una métrica muy clara y directa.

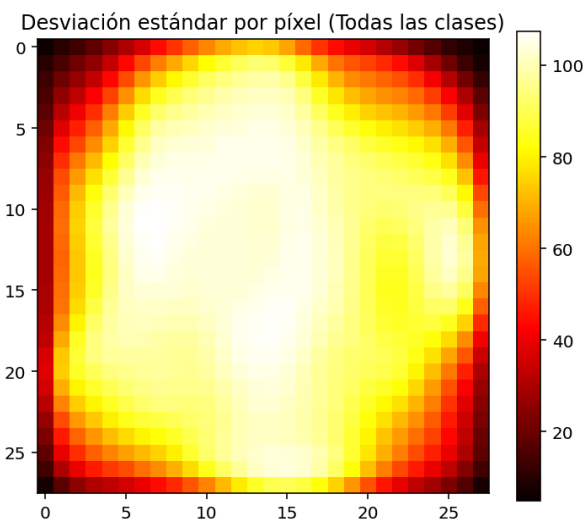
## Análisis Exploratorio

Arrancamos con un análisis exploratorio para entender la información del dataset. Lo primero que hicimos fue analizar la cantidad de datos y clases.

Vimos que el dataset completo consiste en 70,000 imágenes en total. Estas están divididas en 10 clases (etiquetadas del 0 al 9). Un punto clave que encontramos es que el dataset está perfectamente balanceado: cada una de las 10 clases tiene exactamente 7,000 imágenes, lo cual es un escenario ideal para entrenar nuestros modelos y nos permitió usar el Accuracy como métrica principal.

## Atributos Relevantes

Para identificar atributos (píxeles) relevantes, generamos un mapa de calor de la desviación estándar de todos los píxeles.

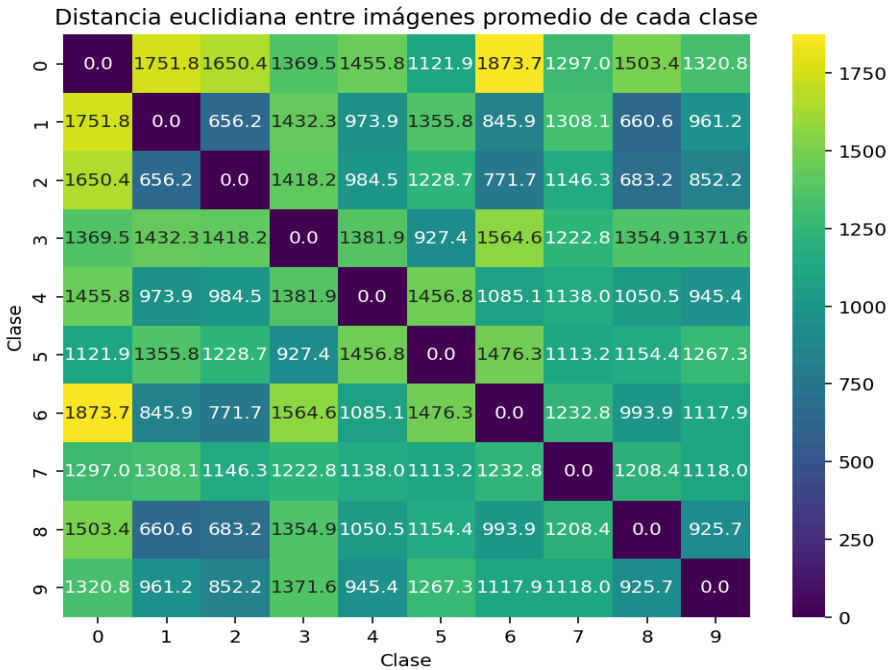


- Los atributos relevantes son los píxeles de la región central (en amarillo/blanco), ya que tienen una alta variabilidad, indicando que ahí es donde se dibujan los caracteres .
- Los atributos no relevantes son los píxeles de los bordes, que tienen una variabilidad muy baja .

Por lo tanto, creemos que se pueden descartar atributos: los píxeles de los bordes no aportan información útil para la clasificación y podrían eliminarse para reducir la dimensionalidad del problema.

## Similitud entre Clases

Para determinar qué tan parecidas son las clases entre sí, calculamos la "imagen promedio" de cada una de las 10 clases y medimos la distancia euclidiana entre ellas.



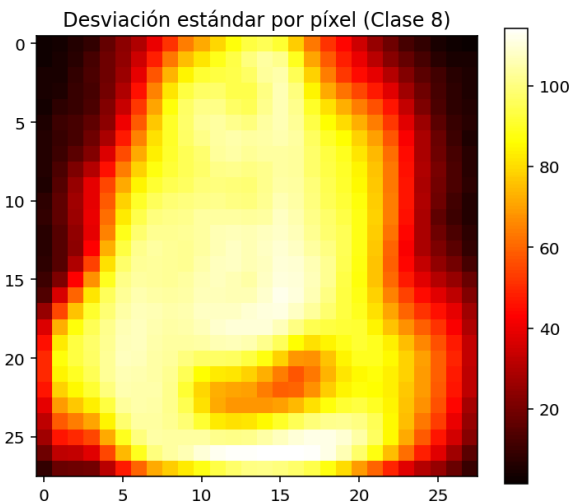
El heatmap nos muestra que hay clases muy cercanas y otras muy lejanas. Por ejemplo:

- La Clase 2 y la Clase 1 son muy cercanas (Distancia 656.2).
- La Clase 2 y la Clase 6 también son muy cercanas (Distancia 771.7).

Esto sugiere, como primera hipótesis, que al modelo le costará más o menos lo mismo diferenciar la Clase 2 de la 1 y la 2 de la 6. (Como veremos en la sección de interpretación de los hallazgos).

### Similitudes dentro de una misma clase

Para responder qué tan similares son las imágenes dentro de una misma clase, analizamos la desviación estándar de los atributos de por ejemplo la Clase 8.

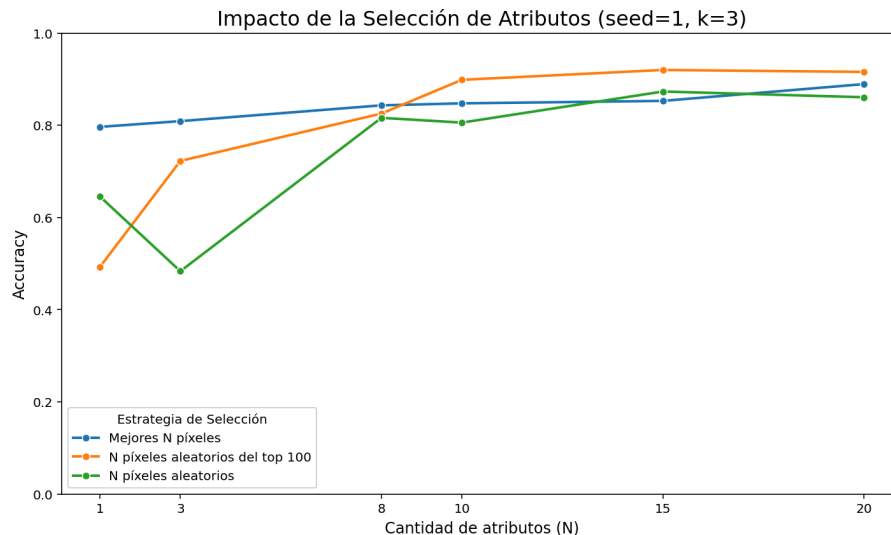
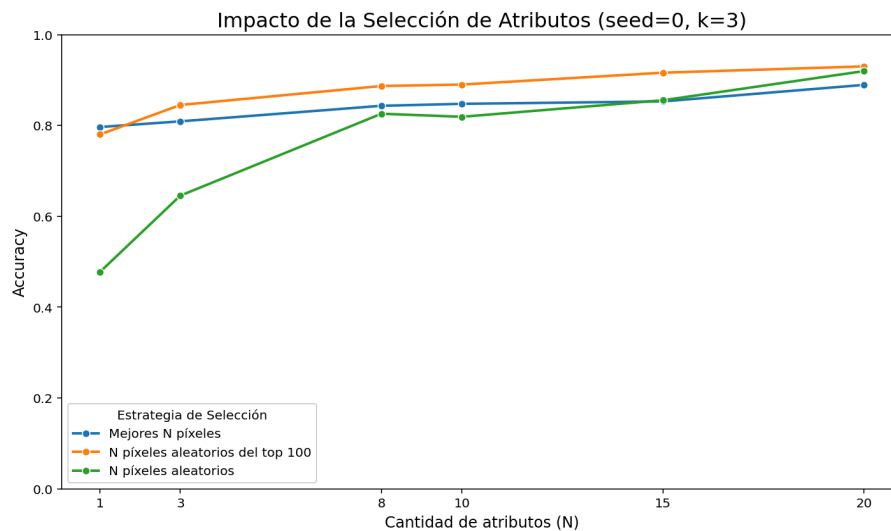


Este gráfico nos muestra que, aunque la forma general del caracter se mantiene, hay una desviación estándar muy alta en casi todos los trazos .

La conclusión que sacamos es que las imágenes de la Clase 8 no son todas similares entre sí. Existe una gran variabilidad en cómo los distintos individuos escribieron este caracter, lo que probablemente dificultará la tarea de clasificación del modelo.

### Impacto de la Selección de Atributos (Problema Binario)

Para investigar la importancia de la selección de atributos, aislamos un problema binario (Clases 4 vs. 5) y usamos un modelo KNN (con  $k=3$ ). Comparamos 3 estrategias de selección y repetimos el experimento con dos conjuntos de atributos aleatorios distintos (seed=0 y seed=1) para ver la estabilidad.



De estos dos gráficos, sacamos una conclusión fundamental sobre el balance entre robustez y potencial:

- **La selección de atributos es clave:** Las estrategias "inteligentes" (azul y naranja) le ganan por mucho a la "Aleatoria" (verde), especialmente con pocos atributos.

➤ **Análisis de Robustez y Potencial:**

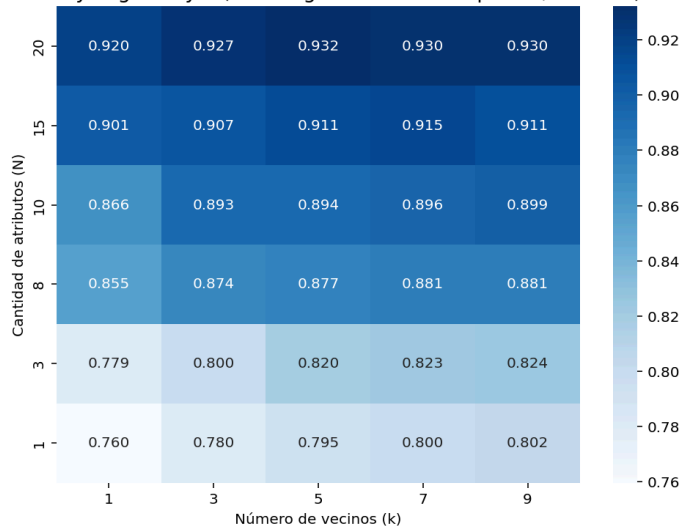
- La estrategia "Mejores N píxeles" (azul) es robusta. Es determinista y nos da un 80% de Accuracy con un solo píxel, sin importar la semilla.
- La estrategia "N píxeles aleatorios del top 100" (naranja) es inestable: con seed=1 empieza muy mal (Accuracy < 50%), pero con seed=0 es la mejor estrategia.

Esto nos dice que la estrategia "naranja" tiene el potencial de ser la mejor, pero es sensible a la suerte.

**Optimización de N y k (Problema Binario)**

Decidimos tomar la estrategia con mayor potencial ("N aleatorios del top 100") y optimizarla. Fijamos el conjunto de atributos aleatorios (el que nos dio los mejores resultados) y corrimos un experimento para encontrar la mejor combinación de N (atributos) y k (vecinos).

Accuracy según N y k (Estrategia 'Aleatorios Top 100', seed=0)



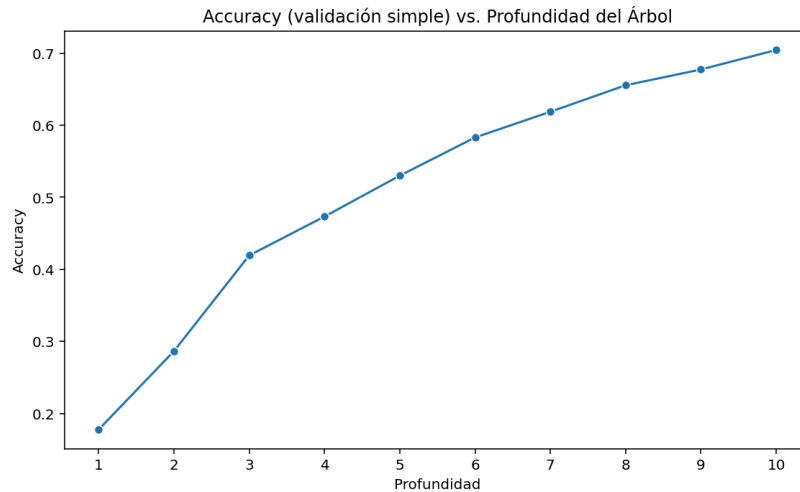
Este heatmap nos dio la respuesta final. El punto de Accuracy máximo fue 0.932%, logrado con N=20 atributos y k=5vecinos.

Es importante notar que, como este análisis se hizo siguiendo la consigna (sin K-Fold Cross Validation), este Accuracy de 0.932 se obtuvo sobre un **único conjunto de prueba**. Es posible que esta configuración de hiperparámetros esté "sobreajustada" a ese split particular, y que el valor sea optimista. Un método como K-Fold (que usamos en la siguiente sección) nos daría una estimación más robusta.

**Optimización de Hiperparámetros (Problema Multiclase)**

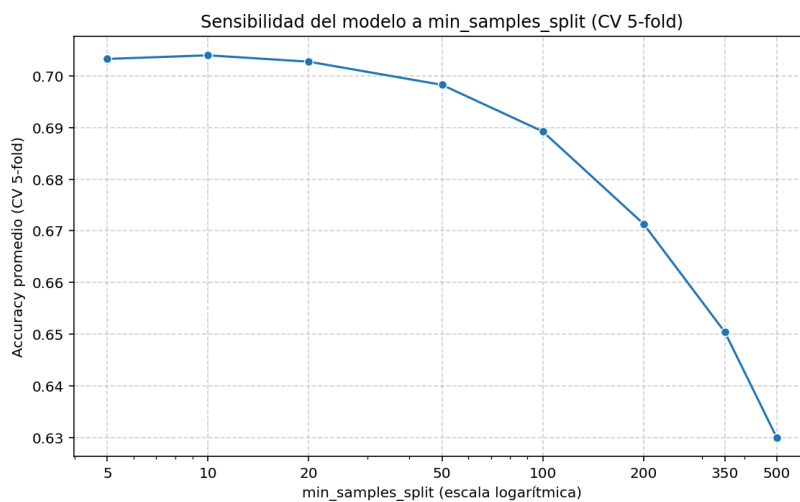
Para el problema multiclase (10 clases), cambiamos a un modelo de Árbol de Decisión y nos enfocamos en el proceso de optimización de hiperparámetros usando K-Fold Cross Validation (K=5).

Primero, hicimos un análisis mediante validación simple de cómo afecta la max\_depth (profundidad).



El gráfico muestra una tendencia clara: a medida que el árbol es más profundo, mejor se vuelve (al menos hasta 10), pasando de menos de 0.2% a más de 0.7% de Accuracy.

Luego, hicimos un análisis más robusto usando K-Fold para encontrar el `min_samples_split` óptimo, fijando `max_depth=10` (que vimos que era bueno).



Este gráfico fue muy revelador. Nos mostró que:

- El rendimiento óptimo está en valores bajos. El pico de Accuracy se da con `min_samples_split=10`.
- Si el valor es muy alto (ej: 100, 200, 500), el rendimiento colapsa. Esto pasa porque forzamos al árbol a ser tan simple (requiriendo 500 muestras para hacer una pregunta) que no puede aprender los patrones, lo que se denomina *underfitting*.

Con estos análisis, ya teníamos los mejores hiperparámetros para la estructura del árbol (`max_depth=10` y `min_samples_split=10`). Para completar la optimización, corrimos el GridSearchCV principal para ver si el criterio ('gini' o 'entropy') hacía alguna diferencia.

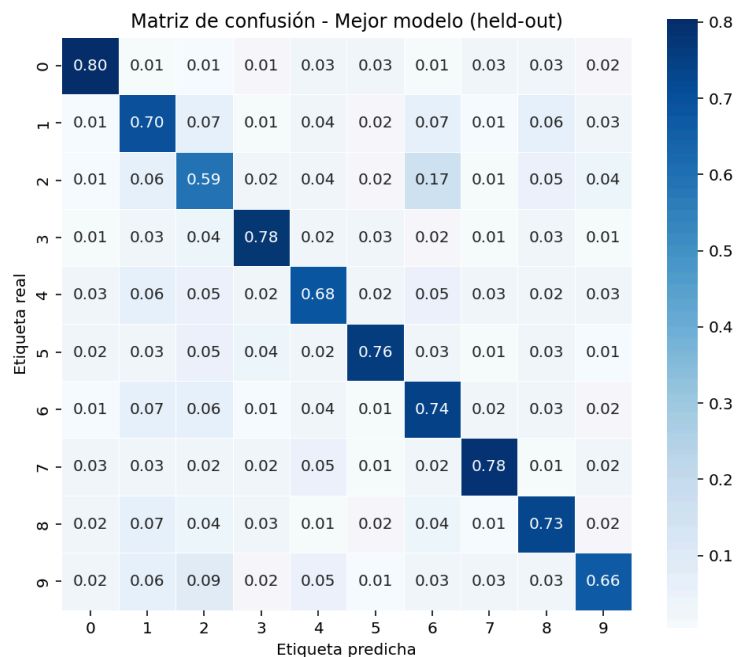
Tal como sospechábamos al ver los gráficos que generamos (que eran casi idénticos), la diferencia de rendimiento entre 'gini' y 'entropy' fue insignificante.

La función GridSearchCV seleccionó automáticamente 'entropy' como el criterio óptimo, ya que superó a 'gini' por un margen muy pequeño en el score promedio. Por lo tanto, confirmamos que nuestros parámetros ganadores eran (max\_depth=10, min\_samples\_split=10, criterion='entropy').

### Evaluación del Modelo Final (Held-Out)

Con los mejores parámetros (max\_depth=10, min\_samples\_split=10, criterion='entropy') seleccionados usando K-Fold, entrenamos un modelo final usando todo el set de desarrollo.

Luego, medimos su rendimiento en el 20% de datos que habíamos guardado y *nunca antes tocado*: el conjunto *held-out*.



La matriz de confusión nos da el resumen completo de los aciertos y errores. La diagonal principal muestra los aciertos.

De esta matriz podemos ver que:

- Clases como la 0 (80%), 3 (78%), 5 (76%), 7 (78%) y 8 (73%) tienen un rendimiento excelente.
- Otras clases tienen problemas. La Clase 2 es la peor, con un acierto de solo 59% , lo que significa que el modelo falla en encontrarla más del 40% de las veces.
- Los números fuera de la diagonal nos muestran *con qué* se confunde: el error más grande del modelo es que un 17% de las veces que ve una Clase 2, la predice erróneamente como una Clase 6.

### Interpretación de los hallazgos

El hallazgo más interesante de este trabajo vino cuando cruzamos los resultados de nuestro análisis exploratorio con los resultados del modelo final.

El EDA (Distancia Euclidiana) mostraba que la Clase 1 (dist. 656.2) y la Clase 6 (dist. 771.7) eran ambas muy parecidas a la Clase 2.

Pero la Matriz de Confusión demostró que el modelo se confunde un 17% de las veces con la Clase 6 , y solo un 6% con la Clase 1

¿Qué aprendimos de esto? Aprendimos que un análisis basado en la "imagen promedio" (como la distancia euclidiana) es demasiado simple y puede ser engañoso. Las imágenes promedio ocultan la variabilidad interna de cada clase.

## Conclusiones

A lo largo de este trabajo, cumplimos los objetivos propuestos de investigar cómo distintos algoritmos y técnicas de optimización afectan el rendimiento de los modelos.

1. En el Análisis Exploratorio: Vimos que, si bien el EDA nos dio una intuición inicial sobre el dataset, esta intuición (basada en promedios) probó ser limitada y engañosa, como demostró la matriz de confusión final.
2. En el Problema Binario (KNN): Demostramos que la selección de atributos es fundamental. También descubrimos el balance entre una estrategia robusta ("Mejores N") y una potencialmente óptima pero inestable("Aleatoria del top 100"). Al optimizar esta última, encontramos un modelo excelente con un Accuracy final de 0.932% (con N=20 y k=5).
3. En el Problema Multiclase (Árbol de Decisión): Nos enfocamos en optimizar el modelo. Usando K-Fold, determinamos que los mejores hiperparámetros eran max\_depth=10 y min\_samples\_split=10. El modelo final, entrenado con estos parámetros, alcanzó un Accuracy de 0.722% en el conjunto *held-out* (de validación final).

El análisis final de la matriz de confusión nos mostró que, si bien el rendimiento del modelo es bueno, tiene problemas específicos con clases como la 2 y la 6, un punto que la distancia euclidiana no logró predecir.

## Bibliografía

[1] Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., & Ha, D. (2018). *Deep learning for classical japanese literature*. (arXiv:1812.01718).