Informe de Clustering sobre el Conjunto de Datos MNIST

Gaizka Carmona Molina

21 de septiembre de 2025

1. Descripción de los Datos

El conjunto de datos utilizado es **MNIST**, compuesto por imágenes de dígitos escritos a mano.

- Número de instancias: `60.000 train, 10.000 test`
- Número de atributos: `784 píxeles por imagen (28x28 en escala de grises)`
- Tipo de atributos: valores enteros en el rango $[0, 255]$, representando la intensidad de cada píxel.
- Distribución de clases: 10 dígitos (`0--9`), con una distribución aproximadamente equilibrada.

Análisis exploratorio

En las Figuras 1 y 2 se muestran dos representaciones gráficas:

1. Un *barplot* con la cantidad de instancias por clase. En azul, la cantidad de instancias en el set *train*, en naranja, la cantidad en *test*. Arriba, la suma de ambas.
2. Una proyección PCA(2) con 3.000 muestras aleatorias, coloreadas según su clase verdadera, tanto para *train* como para *test*.

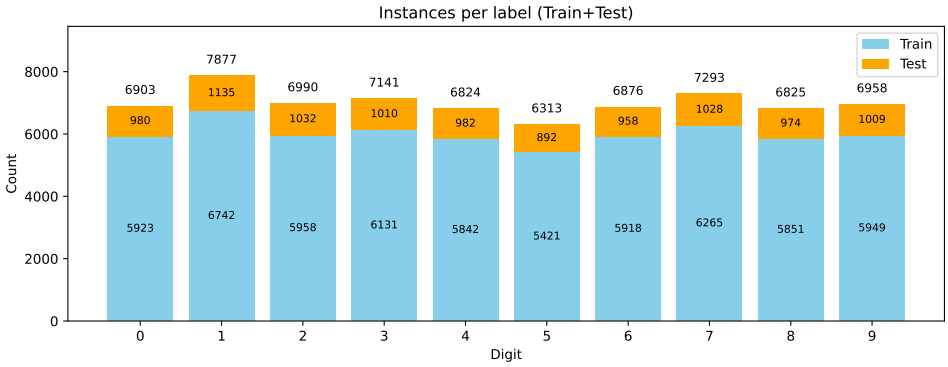

Figura 1: Distribución de instancias por clase (train y test).

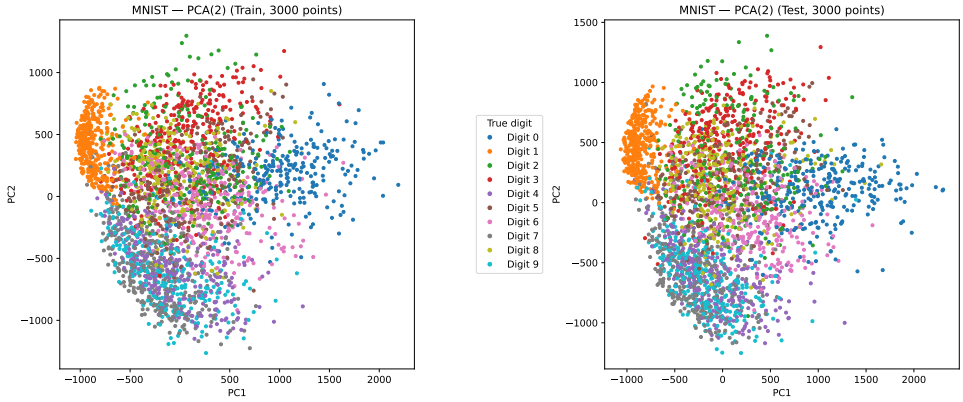

Figura 2: Proyección PCA(2) de las instancias (train y test).

2. Algoritmo de Clustering

Se ha empleado el algoritmo **K-Means**, un método particional que busca minimizar la suma de distancias cuadráticas de cada instancia al centroide de su clúster. Otros algoritmos fueron probados, tanto clustering particional o **Agglomerative Clustering** cómo basado en densidad o **DBSCAN**. Sin embargo, el objetivo principal autoimpuesto de la tarea —la reducción del coste computacional del análisis— no era compatible con estos métodos, por lo que fueron descartados.

Formulación matemática

Dado un conjunto de datos $X = \{x_1, x_2, \ldots, x_n\}$, con $x_i \in \mathbb{R}^d$, el algoritmo K-Means busca encontrar un conjunto de K centroides $C = \{c_1, \ldots, c_K\}$ que minimicen:

$$J = \sum_{i=1}^{n} \min_{j \in \{1,\ldots,K\}} \|x_i - c_j\|^2$$

donde:

- Cada instancia se asigna al cluster con centroide más cercano.
- Los centroides se actualizan como la media de las instancias en su cluster.
- El proceso se repite hasta la convergencia de los centroides, es decir, hasta que el desplazamiento máximo entre dos iteraciones consecutivas sea menor que un umbral de tolerancia. En *scikit-learn*, esto se define cómo:

$$\max_i \|c_i^{(t)} - c_i^{(t-1)}\| < \text{tol}$$

donde $c_i^{(t)}$ representa la posición del centroide i en la iteración t, y tol es un parámetro de tolerancia definido automáticamente como 1e-4 en la implementación de *scikit-learn*.

Se han probado distintos valores de K (`10, 15, 18, 25, 50, 100, 200`) y se ha evaluado tanto en `train` como en `test`, con el objetivo de verificar la constancia de los resultados.

Por otro lado, en una tarea paralela, se ha comparado, utilizando $K = 10$, varias variantes de $PCA(n)$, junto con la versión `raw` del algoritmo.

3. Resultados y Análisis Crítico

Métricas de Evaluación

Se han empleado varias métricas de calidad de clustering:

- **class-to-cluster accuracy** (junto con alineación mediante método húngaro).
- **Silhouette** (cohesión vs separación de clusters, rango $[-1, 1]$).
- **Calinski-Harabasz Index** (separabilidad entre clusters, mayor es mejor).
- **Davies-Bouldin Index** (baja compacidad entre clusters, menor es mejor).

Resultados obtenidos

Distintos valores de K

En la Tabla 1 se resumen los resultados para los distintos valores de K[1].

K	Train			Test		
	Silhouette	CH	DB	Silhouette	CH	DB
10	0.0581	2296.6	2.8360	0.0606	391.4	2.8566
15	0.0617	1852.5	2.7920	0.0639	316.6	2.7961
18	0.0629	1670.9	2.7574	0.0661	284.2	2.7564
50	0.0578	873.8	2.7506	0.0569	148.9	2.7432
100	0.0512	535.2	2.7393	0.0495	91.4	2.7385
200	0.0459	322.4	2.6566	0.0424	55.2	2.6246

Tabla 1: Comparativa de métricas para distintos K (mayor Sil/CH mejor; menor DB mejor).

En la Figura 3 se muestran las matrices de confusión obtenidas para distintos valores de K, los mismos usados para obtener las métricas de la Tabla 1.

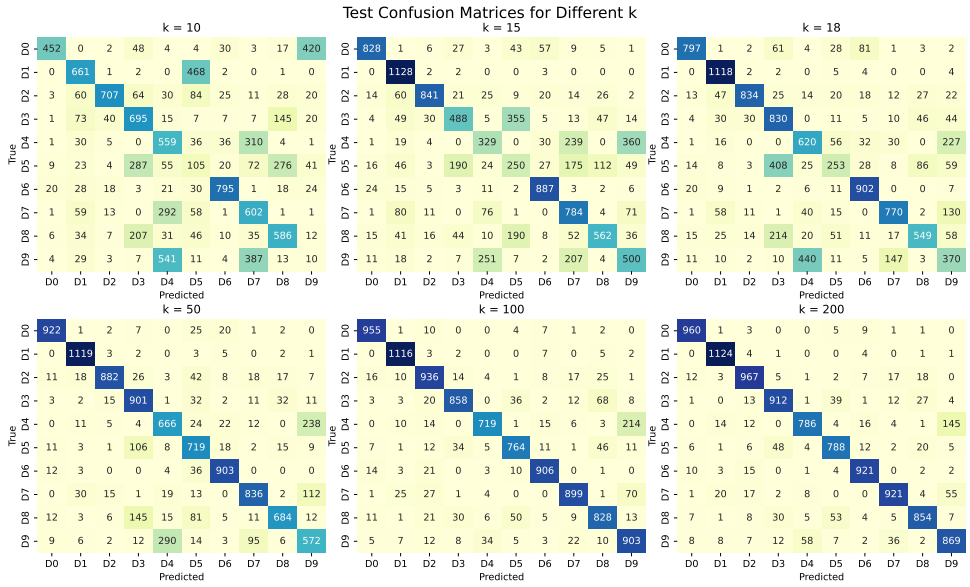

Figura 3: Matrices de confusión (de test) para los distintos valores de K.

[1]La semilla utilizada en todos los cálculos con aleatoriedad fue 18.

raw vs. PCA(n)

En la Tabla 2 se resumen los resultados obtenidos al comparar distintas versiones de $PCA(n)$ con el modelo `raw`, sabiendo que todos los cálculos han sido hechos con $K = 10^2$.

Variant	Train				Test			
	Acc	Silhouette	CH	DB	Acc	Silhouette	CH	DB
Raw(784)	0.5134	0.0581	2296.6	2.8360	0.5172	0.0606	391.4	2.8566
PCA(50)	0.5233	0.1002	2940.1	2.4615	0.5186	0.0989	494.6	2.5155
PCA(100)	0.5242	0.0707	2590.9	2.6500	0.5322	0.0730	441.0	2.6446
PCA(200)	0.5318	0.0633	2404.5	2.7162	0.5437	0.0663	412.7	2.7043
PCA(500)	0.5337	0.0588	2298.4	2.7736	0.5459	0.0617	394.4	2.7645

Tabla 2: Comparativa de métricas por variante (mayor Silhouette/CH mejor; menor DB mejor).

En las Figuras 4 y 5 se pueden observar el diagrama de puntos y la matriz de confusión obtenida para los casos *raw* y para $PCA(500)$, aunque el resto de combinaciones se encuentran junto con la entrega de esta documentación.

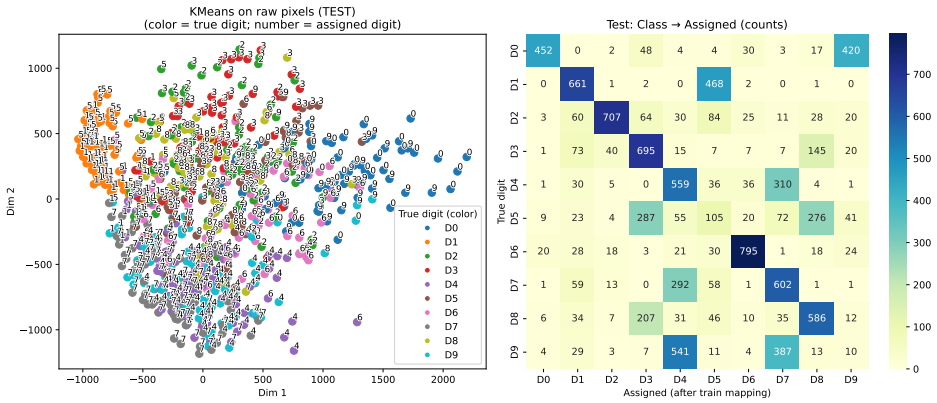

Figura 4: *scatter plot* y *heatmap* para el modelo `raw`

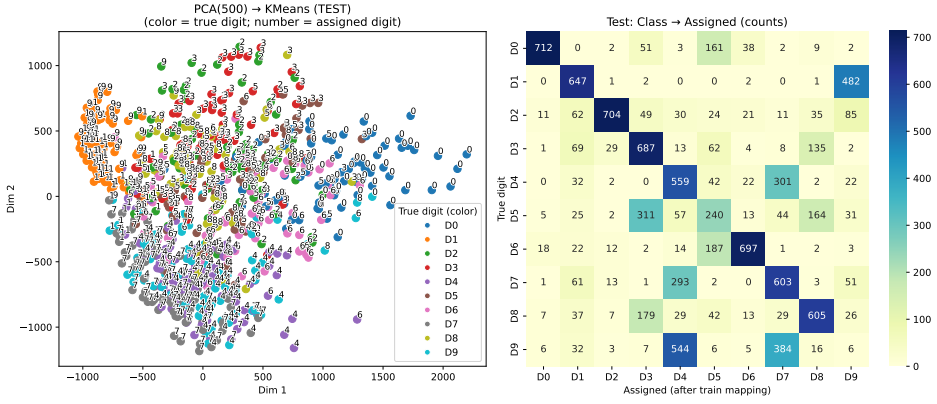

Figura 5: *scatter plot* y *heatmap* para el modelo `PCA(500)`

[2]La semilla utilizada en todos los cálculos con aleatoriedad fue, nuevamente, 18.

Análisis Crítico

- El algoritmo K-Means presenta limitaciones al trabajar con datos no lineales y de alta dimensión.
- Se observa que al aumentar K, la métrica de class-to-cluster accuracy siempre tiende a la mejora, aunque esto no siempre implica mejor separación de clases reales.
- La Silhouette y el índice de Davies-Bouldin permiten observar que un exceso de clusters degrada la cohesión.
- **Limitación importante:** K-Means asume clusters convexos y esféricos, lo que no refleja bien la estructura real de los dígitos manuscritos.

Conclusión

En general, K-Means proporciona una segmentación razonable de los dígitos, pero muestra claras limitaciones en cuanto a la separación de clases similares (ej: dígitos 4 y 9). Como trabajo futuro, se podría comparar con algoritmos jerárquicos o basados en densidad (Agglomerative Clustering y DBSCAN).

Sin embargo, tal y como ha sido mencionado al inicio, para esta tarea, una de las limitaciones autoimpuestas ha sido mantener la carga computacional baja, de manera que se pudiesen hacer muchas pruebas en poco tiempo con *hardware* limitado y sin tener acceso a potentes servidores de alto consumo.