Informe de Trabajo Práctico 1

Aprendizaje Profundo con aplicación a Visión Artificial

Laura Zúñiga Osorio

16 de octubre de 2025

1. *k*-nearest neighbors. En este ejercicio, utilizamos la librería `pandas` para generar dos datasets de datos bidimensionales que les fueron asignadas distribuciones de clase particulares. En el primero, los datos de entrenamiento se forman en torno a un punto central `(n,n)` adicionando ruido aleatorio y designando la clase al entero `n`. Los datos de test se generan aumentando el ancho de la distribución en torno al punto. La clasificación se realizó implementando el método de k-vecinos cercanos barriendo los valores de k entre 1 y 27 en pasos de a 2. Los puntos indicados con una x señalan predicciones erróneas. Las distribuciones y resultados se muestran en la figura 1. Esto se repitió para una distribución de datos inspirada en la clasificación de cuadrantes del plano cartesiano, pero incluyendo una quinta clase adicional para los puntos cercanos a los ejes.

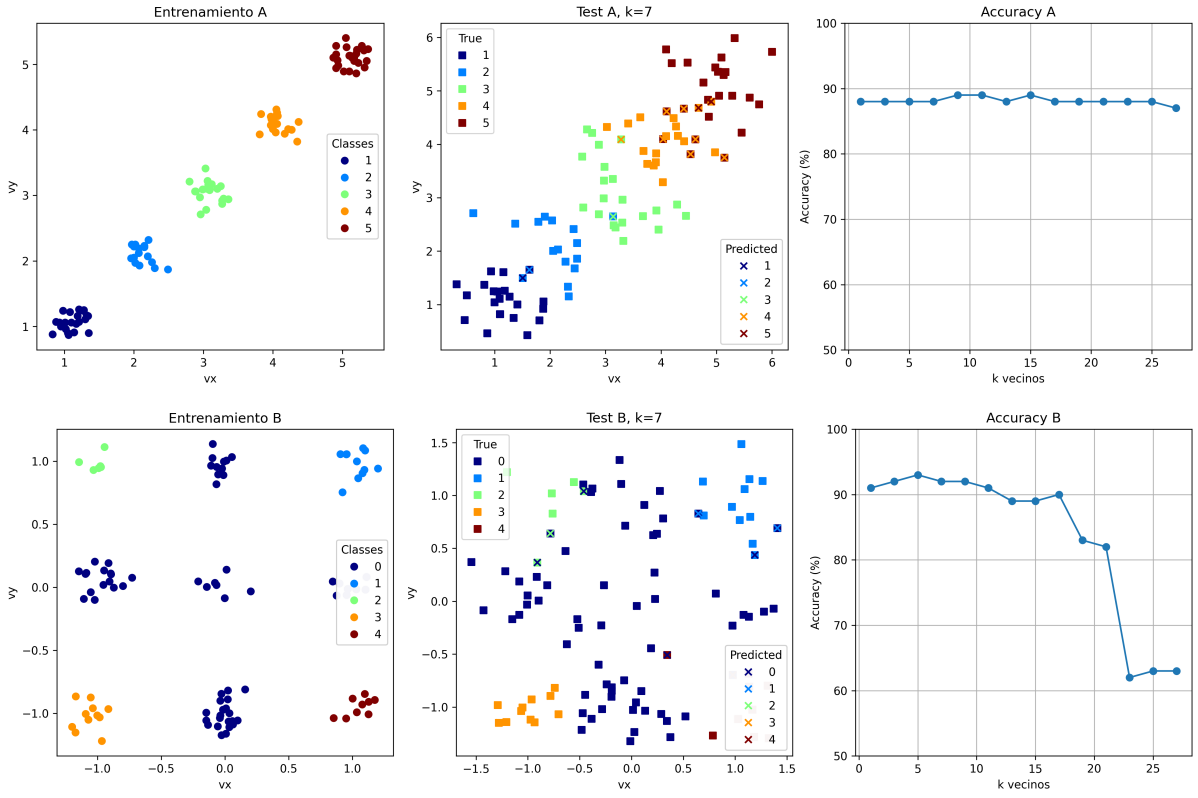

Figura 1: Clasificación por método de k-vecinos cercanos para dos distribuciones de datos diferentes.

Para generar un mapa que ilustre las fronteras de decisión, se graficaron 1000 puntos del espacio con su respectiva clase predicha, como se muestra en la figura 2.

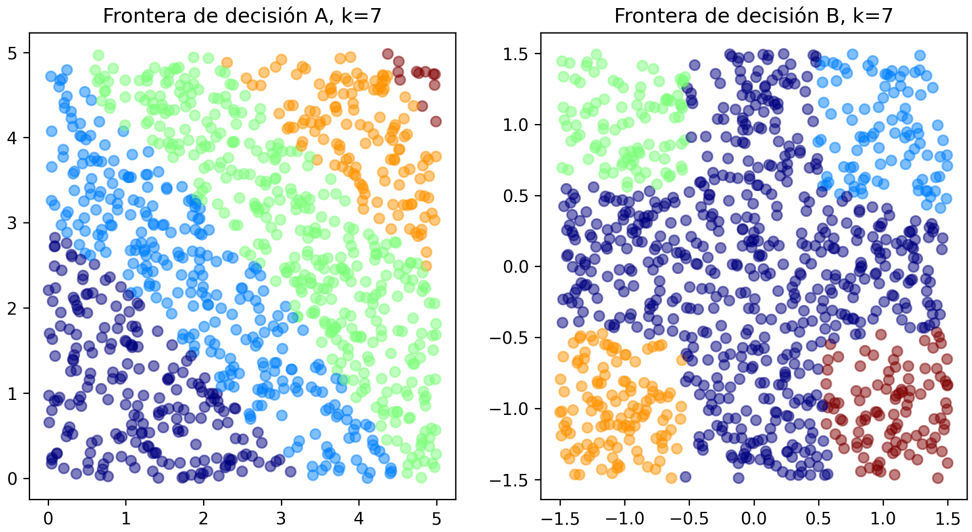

Figura 2: Zonas de decisión del clasificador para cada distribución de datos.

2. *k*-nearest neighborst para MNIST y CIFAR-10 . El mismo método de clasificación k-nn se implementó con los datasets públicos MNIST (imgs. de números manuscritos del 0 al 9) y CIFAR-10 (imágenes naturales separadas en 10 clases). Para cargar los datos, se hizo uso de la librería `torchvision.datasets` y la manipulación de sus dimensiones se hizo a nivel tensorial mediante la librería `torchvision.transforms`. Los 5 primeros elementos del conjunto de test se muestran para cada uno en la figura 3.

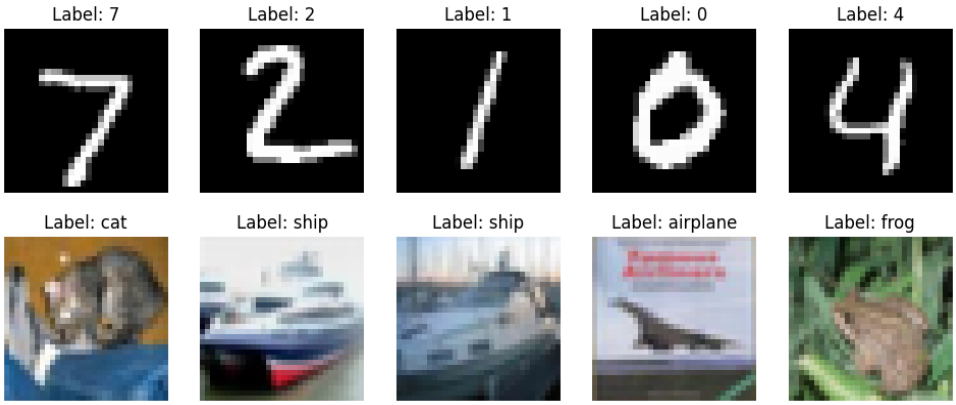

Figura 3: Ejemplos de imágenes del conjunto de test para MNIST (arriba) y CIFAR-10 (abajo).

Los gráficos de accuracy muestreando los 20 primeros datos de cada conjunto, según el número de vecinos, se muestran en la figura 4.

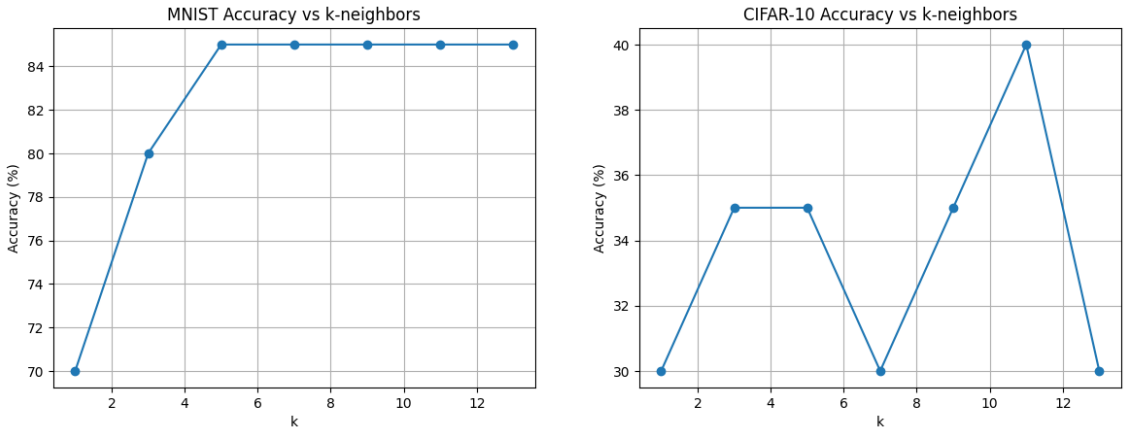

Figura 4: Resultados de accuracy como función de k para los conjuntos MNIST y CIFAR-10.

3. Implementación de SVM y Softmax Implementamos el método de clasificación lineal SVM y SoftMax utilizando regularización L2 y el paradigma de programación orientada a objetos. Para ello, ambas clases heredan de una clase base llamada `LinearClassifier`, que provee los métodos `fit`, `predict` y el cálculo del gradiente de la función de pérdida. Además, se implementó la métrica de *accuracy* para monitorear la precisión durante el entrenamiento. Evaluamos ambas implementaciones con los conjuntos de datos MNIST (figuras 5, 6) y CIFAR-10, utilizando el método de entrenamiento mini-batch (BGD). Finalmente, comparamos la exactitud de los métodos y graficamos las curvas de pérdida y precisión para distintas épocas, tanto en entrenamiento como en evaluación, analizando cuál resulta más adecuado para cada problema.

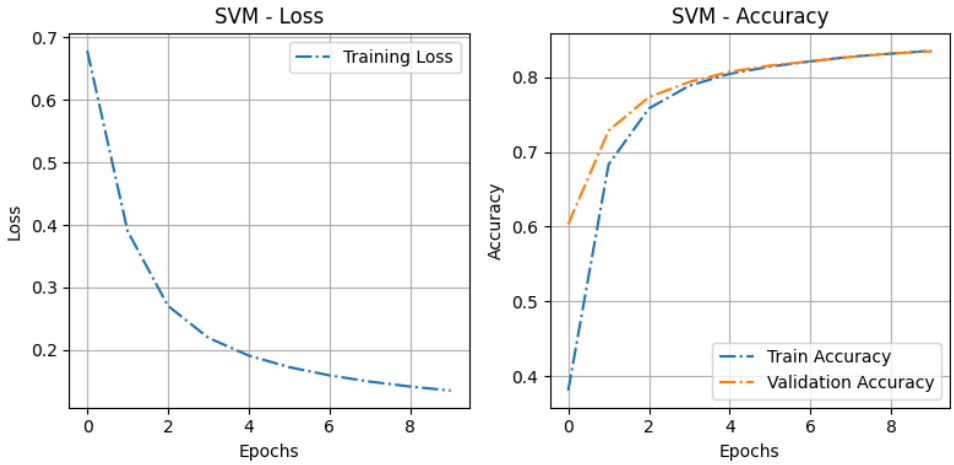

Figura 5: (MNIST) Métricas de entrenamiento para clasificador lineal. La clase heredada define la función de pérdida correspondiente al método *hinge loss*, propio de Support Vector Machine.

Los resultados después de 10 épocas del entrenamiento del conjunto CIFAR-10 se incluyen a continuación:

SVM. `Epoch 10: loss=0.5261, train acc=0.3488, val acc=0.3418`
SoftMax. `Epoch 10: loss=1.8987, train acc=0.3587, val acc=0.3450`

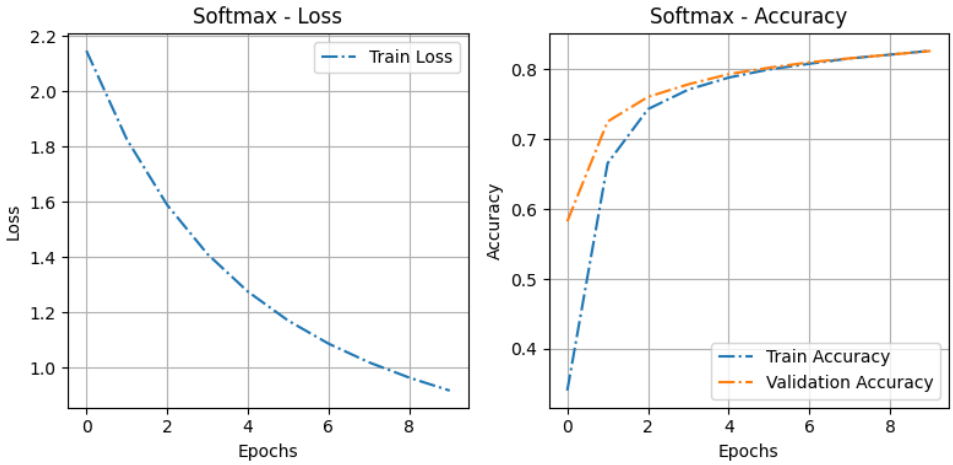

Figura 6: (MNIST) Métricas de entrenamiento para el método SoftMax.

Ambos métodos obtienen mejores resultados en el conjunto MNIST después de 10 épocas. Esto puede atribuirse a la naturaleza de los datos: MNIST contiene imágenes más simples y menos variables que CIFAR-10, lo que facilita la tarea de clasificación lineal.