Temas Selectos de Física Computacional III: Introducción a la ciencia de datos. Clase 2: 11 Febrero 2025

Dr. Leonid Serkin

February 11, 2025

1 Conjunto de datos MNIST

El conjunto de datos MNIST es otro conjunto de datos particularmente famoso. Contiene varios miles de dígitos escritos a mano (del 0 al 9). Cada dígito está contenido en una imagen en escala de grises de 28×28 píxeles de 8 bits. Esto significa que cada dígito tiene 784 (28^2) píxeles, y cada píxel tiene un valor que varía de 0 (negro) a 255 (blanco). La Figura 1 muestra uno de esos dígitos.

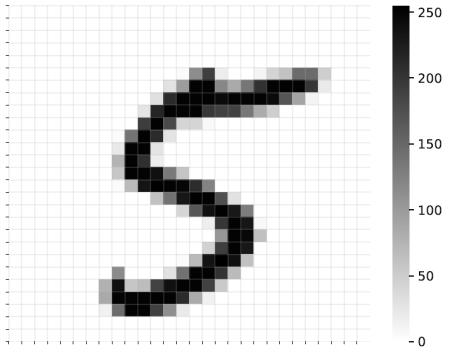

Figure 1: Uno de los dígitos del conjunto de datos MNIST.

El conjunto de datos que usaremos es fue preparado para esta clase y se llama:

`mnist_version_corta.csv`

En nuestro caso, MNIST está representado como un archivo CSV. Similar al conjunto de datos Iris, cada fila de los conjuntos de datos MNIST representa un dígito. Por simplicidad, este conjunto de datos contiene solo una pequeña fracción (10,000 dígitos de 70,000) del conjunto de datos MNIST real.

Para cada dígito, hay 785 valores disponibles: el primero es el valor numérico representado en la imagen (por ejemplo, para la Figura 1 sería 5). Las siguientes 784 columnas representan la imagen en escala de grises en orden por filas.

El conjunto de datos MNIST en formato CSV se puede leer con el mismo enfoque utilizado para Iris, teniendo en cuenta que, en este caso, la etiqueta del dígito (es decir, la primera columna) es un entero de 0 a 9, mientras que los 784 valores siguientes son enteros entre 0 y 255.

2 Ejercicios

1. Carga el conjunto de datos MNIST descargado previamente. Puedes hacer uso del módulo `csv` (u otro modulo si gustas). Abre el archivo y almacena todo en la variable `data` como una lista de listas (cada fila es una lista). Puedes usar:
   `len(data)`: Devuelve el número de filas.
   `len(data[0])`: Devuelve el número de columnas, siempre y cuando haya al menos una fila. Cuántas filas y columnas tiene el conjunto? Imprime las primeras 5 filas.

2. Ahora que ya sabemos cómo leer y manipular datos en formato CSV, vamos a aprender a visualizar las imágenes contenidas en el conjunto de datos MNIST. Recuerda que cada fila representa una imagen de un número escrito a mano, donde la primera columna es la etiqueta (el número representado) y las siguientes 784 columnas contienen los valores de los píxeles de la imagen (de 28x28 píxeles).
   Importa dos librerías clave:

   ```
   import matplotlib.pyplot as plt
   import numpy as np
   ```

   Toma la primera fila del conjunto de datos, que contiene la etiqueta y los valores de los píxeles.

   ```
   first_row = data[0]
   first_label = int(first_row[0])
   ```

   Convierte los valores de los píxeles en una lista de enteros y luego en un array de NumPy.

   ```
   first_image = [int(pixel) for pixel in first_row[1:]]
   first_image = np.array(first_image, dtype=np.uint8)
   ```

Ahora transforma el vector en una matriz de 28×28 y usa Matplotlib para graficarlo.

```
def show_digit(image_vector, label):
    image_matrix = image_vector.reshape(28, 28)
    plt.imshow(image_matrix, cmap="gray")
    plt.title(f"Dígito: {label}")
    plt.axis("off")
    plt.show()

show_digit(first_image, first_label)
```

3. Calcula las estadísticas de los valores de los píxeles, como la media y la desviación estándar, además de contar cuántas veces aparece cada dígito en el dataset. Extrae la información en listas separadas para las etiquetas y los valores de los píxeles. Usa `int(row[0])` para convertir la etiqueta en un número entero. Aplica una comprensión de listas para convertir los valores de píxeles en enteros.

   ```
   labels = []
   pixels = []

   for row in data:
       labels.append(int(row[0]))  # número de la imagen
       pixels.append([int(p) for p in row[1:]])  # píxeles como números
   ```

   Convertimos la lista de píxeles en un array de NumPy para facilitar los cálculos estadísticos.

   ```
   pixels_array = np.array(pixels)
   ```

   Usa el modulo de NumPy para calcular la media y la desviación estándar de los valores de los píxeles.

   ```
   pixel_mean = np.mean(pixels_array)  # Media
   pixel_std = np.std(pixels_array)  # Desviación estándar
   ```

   Ahora, cuenta cuántas imágenes hay de cada número en el conjunto. Grafica un histograma con `matplotlib` para visualizar la distribución de los dígitos en el dataset.

4. Inicializa una lista vacía que contendrá las listas de valores de píxeles, donde cada lista representará un dígito, y una lista vacía que contendrá las etiquetas (números del 0 al 9) correspondientes a cada dígito en el dataset.
   Puedes usar:

`cols.pop(0)`: Elimina y devuelve el primer elemento de la lista cols. Este primer elemento es la etiqueta (número del dígito) correspondiente a la imagen, que se almacena en la lista labels.

`labels.append(int(cols.pop(0)))`
Elimina y devuelve el primer elemento de `cols`, que es la etiqueta del dígito, lo convierte en un entero y lo agrega a la lista `labels`.

`map(int, cols)`: Convierte todos los valores restantes en cols (que son cadenas de texto) a enteros.

`dataset.append(list(map(int, cols)))`
Convierte los valores restantes en `cols` (los píxeles de la imagen) a enteros y los agrega a `dataset` como una lista.

5. Crea una función que, dado una posición $1 \leq k \leq 10000$, imprima el k-ésimo dígito del conjunto de datos (es decir, la k-ésima fila del archivo CSV) como una cuadrícula de 28×28 caracteres. Más específicamente, debes mapear cada rango de valores de píxeles a los siguientes caracteres:
$[0, 64) \rightarrow$ `" "`
$[64, 128) \rightarrow$ `"."`
$[128, 192) \rightarrow$ `"*"`
$[192, 256) \rightarrow$ `"#"`
Por ejemplo, debes mapear la secuencia `0, 72, 192, 138, 250` a la cadena `" .#*#"`. Para la Figura 1 (que es la imagen número 130 en el conjunto de datos), el resultado sería lo que ves en la Figura 2.

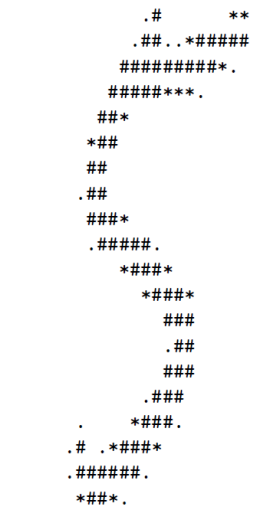

Figure 2: Imagen número 130 en el conjunto de datos

Para ello necesitas mapear cualquier valor de píxel entre 0 y 255 al carácter

que queremos mostrar.
Crea una función `caracter` utiliza un diccionario llamado `rangos` que mapea rangos de valores de píxeles (entre 0 y 255) a caracteres específicos. Estos caracteres son utilizados para representar diferentes niveles de intensidad de los píxeles, de modo que:

- Los valores de píxeles entre 0 y 63 se mapean al carácter `" "` (un espacio en blanco).
- Los valores de píxeles entre 64 y 127 se mapean al carácter `"."`.
- Los valores de píxeles entre 128 y 191 se mapean al carácter `"*"`.
- Los valores de píxeles entre 192 y 255 se mapean al carácter `"#"`.

La función recibe un valor de píxel como entrada. Luego, itera a través de los elementos del diccionario `ranges`, comparando el valor del píxel con los rangos definidos:

- Si el valor del píxel cae dentro de un rango particular, se devuelve el carácter asociado a ese rango.

Por ejemplo, si el valor del píxel es 70, caerá dentro del rango `(64, 128)`, por lo que la función devolverá el carácter `"."`. Esto permite una representación visual simplificada de imágenes en escala de grises mediante caracteres, donde la intensidad del píxel determina el carácter utilizado.

```
def caracter(pixel):
    rangos = {
        (0, 64): " ",
    }
    for (a, b), ch in rangos.items():
        if a <= pixel < b:
            return ch
```

Una vez que tenemos esta función, podemos mapear cualquier vector de 784 dimensiones a la secuencia de caracteres correspondiente.

Con esto, puedes iterar sobre las filas y sobre las columnas para imprimir la cuadrícula resultante.

Es importante notar que, cada imagen en MNIST tiene un tamaño de 28 píxeles de ancho por 28 píxeles de alto, lo que da un total de 784 píxeles. Por lo tanto, para imprimir el carácter en la fila i y la columna j, necesitaremos acceder al valor en la posición $28 \times i + j$ del vector de 784 dimensiones.

6. Hay 1,135 unos y 980 ceros en el conjunto de datos. Define una función que, dado un dígito del 0 al 9, cuente las ocurrencias entre todos los dígitos disponibles

```
def contar_pixeles_negros(conjunto, etiquetas, digito)
```

Para todos los ceros y unos por separado, cuenta cuántas veces cada uno de los 784 píxeles es negro (usa 128 como valor umbral). Puedes hacer esto construyendo dos listas, cada una de ellas con 784 elementos, que contienen respectivamente los conteos para los ceros y los unos. Entonces tus listas contienen el número de veces que el i-ésimo píxel fue negro para cada clase.

Genera los mapas de calor para cada una de estas listas. Por ejemplo en la Figura 3 puedes ver cómo se distribuyen los píxeles negros .

```
import seaborn as sns
sns.heatmap(np.reshape(lista, (28, 28)), cmap='binary')
plt.figure()
```

Para cada valor i, calcula `abs(primera_lista[i] - segunda_lista[i])`. El i con el valor más alto representa el píxel que mejor separa los dígitos "0" y "1" (es decir, el píxel que más a menudo es negro para una clase y blanco para la otra). ¿Dónde se encuentra este píxel dentro de la cuadrícula? ¿Por qué es así?

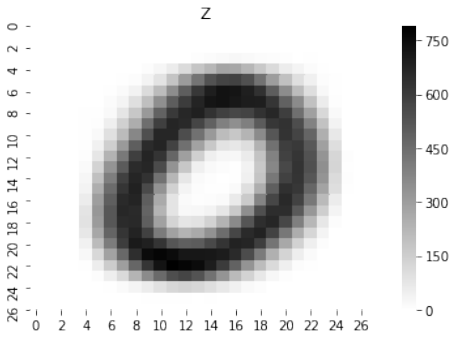

Figure 3: Mapa de píxeles para el 0.