AdD_a14_GRUPO_LOPEZ

March 31, 2024

1 Análisis de datos - Trabajo Integrador

Alumno: Milton Lopez

2 Introducción / dataset elegido

2.0.1 El dataset elegido para realizar el análisis es MNIST.

Preguntas a responder

- 1. ¿Se pueden encontrar heurísticas interesantes para clasificar los datos en función de sus valores?
- 2. ¿Es posible encontrar representaciones de baja dimensionalidad que nos permitan visualizar posibles grupos?

3 Análisis exploratorio inicial

3.1 Carga de datos y visualización inicial

[ ]:
```
# Importamos librerías

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import fetch_openml
```

[ ]:
```
# Cargamos el dataset
# Se eligió la versión de MNIST clásica, no la built-in de sklearn
digits = fetch_openml("mnist_784", version=1, as_frame=False)
X, y = digits.data, digits.target
```

[ ]:
```
# Primeras 5 filas
data = pd.DataFrame(X)
data["target"] = y

data.head()
```

[ ]:

	0	1	2	3	4	5	6	7	8	9	…	775	776	777	778	779	780	781	782	\
0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	

	783	target
0	0	5
1	0	0
2	0	4
3	0	1
4	0	9

[5 rows x 785 columns]

[ ]:
```
data.info()
```

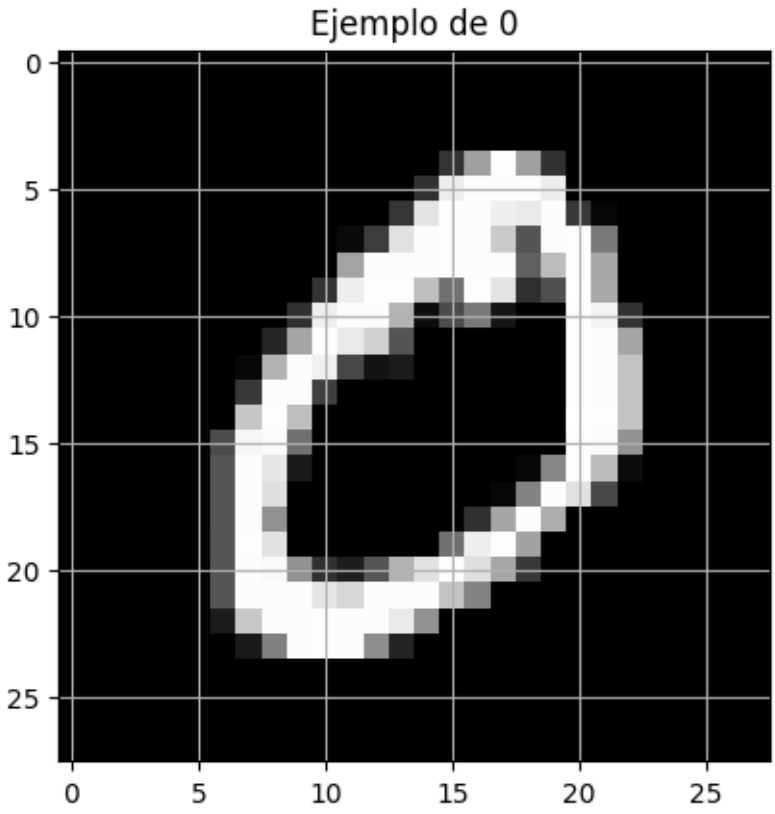
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Columns: 785 entries, 0 to target
dtypes: int64(784), object(1)
memory usage: 419.2+ MB
```

El dataframe indica que hay 70000 filas y 785 columnas en el dataset.

[ ]:
```
# Dimensiones del dataset
n_samples, n_features = X.reshape((len(X), -1)).shape
n_classes = len(np.unique(y))

print(f"Total de muestras (imágenes): {n_samples}")
print(f"Total de features (píxeles]) por muestra: {n_features}")
print(f"Número de clases (dígitos únicos): {n_classes}")
```

```
Total de muestras (imágenes): 70000
Total de features (píxeles]) por muestra: 784
Número de clases (dígitos únicos): 10
```

[ ]:
```
# Cargamos una muestra
plt.grid(True)
plt.title("Ejemplo de {}".format(y[1]))
plt.imshow(X[1, :].reshape((28, 28)), cmap="gray")
plt.show()
```



3.1.1 Definición del dataset:

Con estas primeras exploraciones, ya podemos definir y explicar el dataset MNIST con el que estamos trabajando.

Espacio ambiente: El espacio ambiente es el espacio donde existen los data points (imágenes). En el caso de MNIST, vimos que cada imagen se representa con un total de 784 features; una grilla de 28×28 píxeles donde cada píxel representa una feature.

Por lo tanto, el espacio ambiente es $\mathbb{R}^{784}$.

Data points (imágenes) y features (píxeles): Cada data point (variable) en MNIST es una imagen de 28×28 de un dígito escrito a mano. Un data point $x \in \mathbb{R}^{784}$ es un vector 784-dimensional, donde cada elemento del vector corresponde a una determinada intensidad del píxel.

Formalmente, un data point es $x^{(i)} \in \mathbb{R}^{784}$, donde $i = 1, 2, \ldots, m$, y m es la cantidad total de imágenes.

Vimos que la cantidad de imágenes m es 70000.

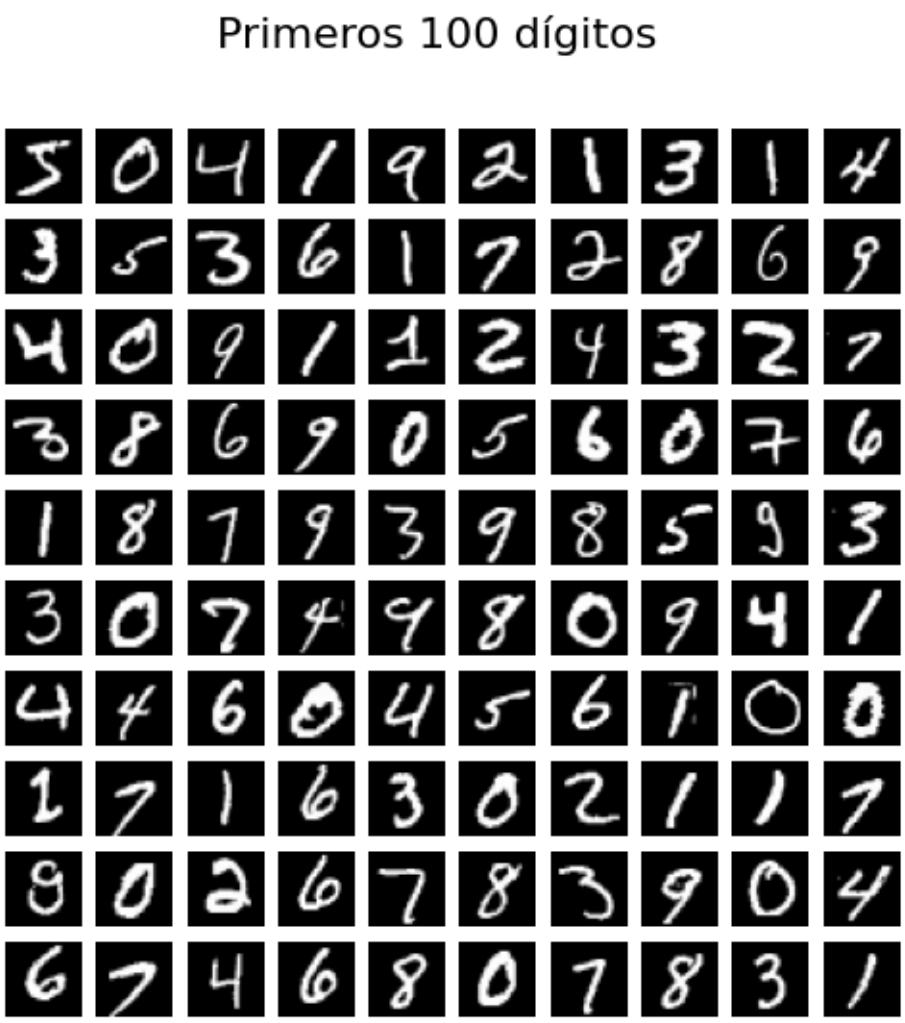
Labels / variable target: El conjunto de labels $Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ son los enteros que representan los dígitos del 0 al 9.

Dataset: Finalmente, el dataset $\mathcal{D}$ de dígitos MNIST puede representarse como un conjunto de pares (x, y) donde x es la imagen e y es el label/target (el dígito que representa la imagen).

$\mathcal{D} = \left\{ \left(x^{(i)}, y^{(i)} \right) \mid x^{(i)} \in \mathbb{R}^{784}, y^{(i)} \in \{0, 1, 2, \ldots, 9\} \right\}$.

3.1.2 Visualización de primeros dígitos

[ ]:
```
fig, axs = plt.subplots(nrows=10, ncols=10, figsize=(6, 6))
for idx, ax in enumerate(axs.ravel()):
    ax.imshow(X[idx].reshape((28, 28)), cmap="gray")
    ax.axis("off")
_ = fig.suptitle("Primeros 100 dígitos", fontsize=16)
```



3.1.3 Representación de las imágenes

Vamos a explorar una imagen y examinar tanto la representación numérica como la visual lado a lado.

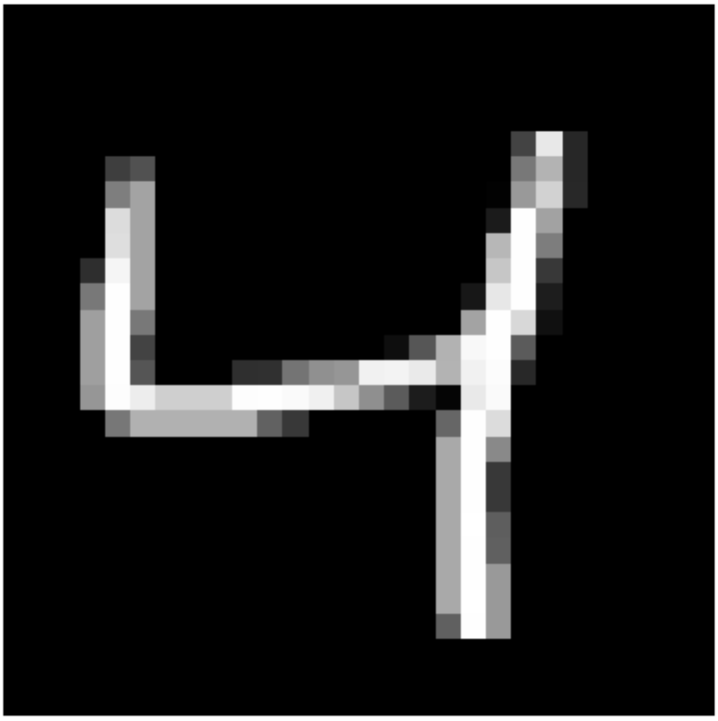
Esto es necesario ya que queremos saber:

- En qué escala está representada la intensidad de los píxeles.
- Cuál es el valor máximo y mínimo de intensidad de píxel.
- En que orden está la escala. Si la intensidad crece de oscuro a claro o viceversa.

```
[ ]: # Imagen del dataset

     images = X.reshape(-1, 28, 28)

     image_index = 2
     print(f"Imagen {2}:")
     print(images[image_index])
     plt.imshow(images[image_index], cmap="gray")
     plt.axis("off")
     plt.show()
```

```
Imagen 2:
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0  67 232  39   0   0   0   0   0]
 [  0   0   0   0  62  81   0   0   0   0   0   0   0   0   0   0   0   0
    0   0 120 180  39   0   0   0   0   0]
 [  0   0   0   0 126 163   0   0   0   0   0   0   0   0   0   0   0   0
    0   2 153 210  40   0   0   0   0   0]
 [  0   0   0   0 220 163   0   0   0   0   0   0   0   0   0   0   0   0
    0  27 254 162   0   0   0   0   0   0]
 [  0   0   0   0 222 163   0   0   0   0   0   0   0   0   0   0   0   0
    0 183 254 125   0   0   0   0   0   0]
 [  0   0   0  46 245 163   0   0   0   0   0   0   0   0   0   0   0   0
    0 198 254  56   0   0   0   0   0   0]
 [  0   0   0 120 254 163   0   0   0   0   0   0   0   0   0   0   0   0
   23 231 254  29   0   0   0   0   0   0]
 [  0   0   0 159 254 120   0   0   0   0   0   0   0   0   0   0   0   0
  163 254 216  16   0   0   0   0   0   0]
 [  0   0   0 159 254  67   0   0   0   0   0   0   0   0   0  14  86 178
```

```
 248 254  91   0   0   0   0   0   0   0]
[  0   0   0 159 254  85   0   0   0  47  49 116 144 150 241 243 234 179
 241 252  40   0   0   0   0   0   0   0]
[  0   0   0 150 253 237 207 207 207 253 254 250 240 198 143  91  28   5
 233 250   0   0   0   0   0   0   0   0]
[  0   0   0   0 119 177 177 177 177 177  98  56   0   0   0   0   0 102
 254 220   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 254 137   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 254  57   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 254  57   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 255  94   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 254  96   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 254 153   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 169
 255 153   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  96
 254 153   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]]
```



Podemos ver claramente que los valores de intensidad de los píxeles en escala de grises están en orden ascendente desde más oscuro (0) hasta más claro (255).

`0` representa un píxel totalmente negro.

`255` representa un píxel totalmente blanco.

3.2 Identificación de tipos de datos (categórico, ordinal, etc.)

3.2.1 Imágenes (features):

Como ya mencionamos, cada imagen es una matriz de 28x28 píxeles, y cada píxel representa la intensidad en escala de grises.

Los valores son de 0 a 255, donde 0 es negro y 255 es blanco. Estos datos son `numéricos` y se usan para clasificación, ya que las diferencias en las intensidades sirven para distinguir entre los diferentes dígitos.

3.2.2 Labels (objetivo):

Los labels son numéricos (0 a 9) y representan el dígito en cada imagen. Estos son datos `categóricos ordinales` porque tienen un orden claro (0 a 9). Sirven de dato informativo para la clasificación (lo que se quiere predecir).

El objetivo principal de trabajo con el dataset MNIST es clasificar cada imagen en una de las 10 categorías usando las intensidades de los píxeles.

3.3 Identificación de variables de entrada y salida

3.3.1 Variables de entrada (features):

Son las intensidades de los píxeles en las imágenes de los dígitos.

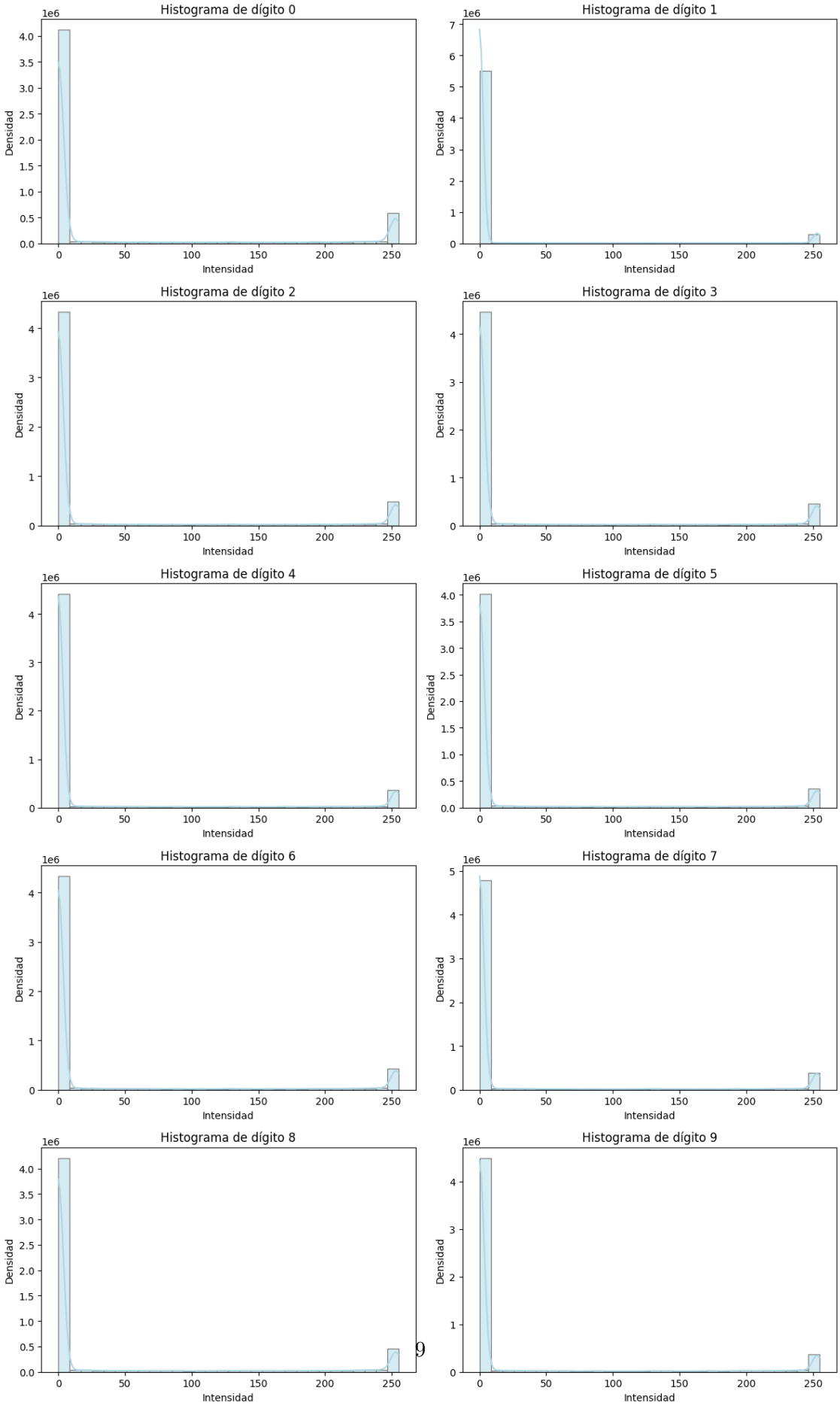
3.3.2 Variables de salida (objetivo):

Es el dígito que representa cada imagen (0-9).

3.3.3 Análisis de variables de entrada

Histogramas de píxeles de cada dígito

[ ]:
```
fig, axs = plt.subplots(5, 2, figsize=(12, 20))

# Ploteo de histograma
for digit in range(10):
    row = digit // 2
    col = digit % 2

    intensity_values = X[y == str(digit)].flatten()
    sns.histplot(
        intensity_values,
        bins=30,
        color="lightblue",
        edgecolor="gray",
        ax=axs[row, col],
        kde=True,
    )
    axs[row, col].set_title(f"Histograma de dígito {digit}")
    axs[row, col].set_xlabel("Intensidad")
    axs[row, col].set_ylabel("Densidad")

plt.tight_layout()
plt.show()
```

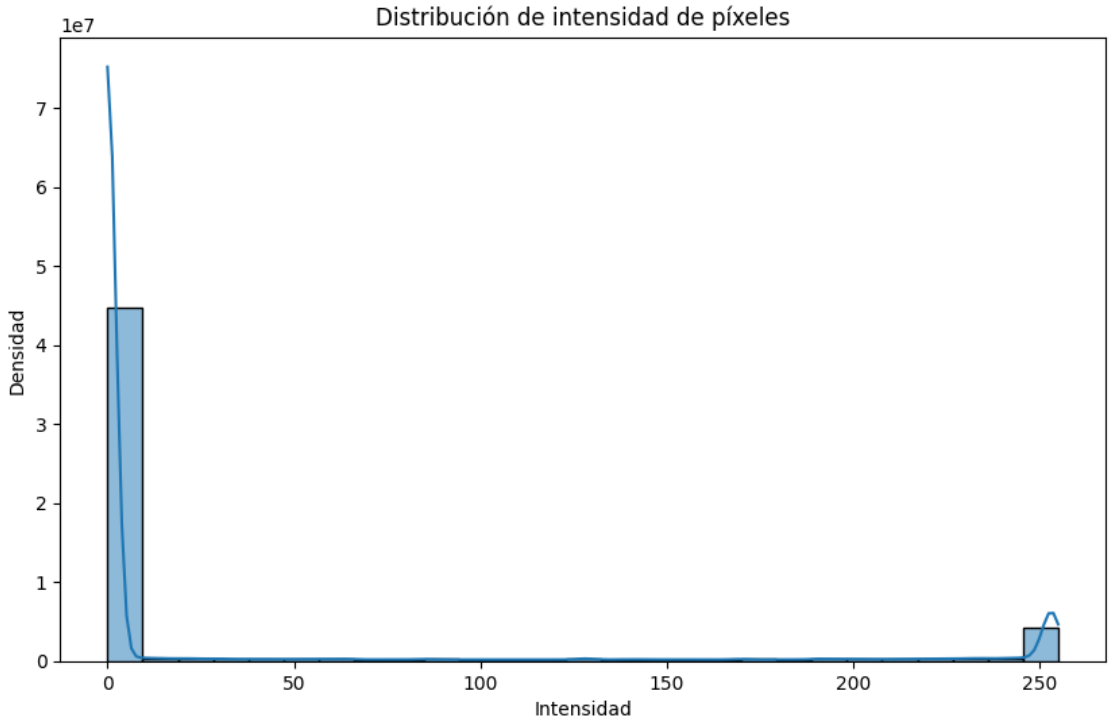

Histograma de dígito 0
Histograma de dígito 1
Histograma de dígito 2
Histograma de dígito 3
Histograma de dígito 4
Histograma de dígito 5
Histograma de dígito 6
Histograma de dígito 7
Histograma de dígito 8
Histograma de dígito 9
Densidad
Intensidad
1e6

Notamos que el dígito 1 contiene más pixeles negros que el resto de las distribuciones, y la oblicuidad hacia la derecha es más pesada. Es consistente con la intuición ya que el 1 contiene menos píxeles blancos (píxeles correspondientes al dígito dibujado) que el resto.

Distribución de intensidad de píxeles general - Momentos

[ ]:
```
from scipy.stats import skew, kurtosis

X_flattened = X.reshape((X.shape[0], -1))

# Cálculo de momentos de la distribución
mean_intensity = np.mean(X_flattened)
median_intensity = np.median(X_flattened)
std_intensity = np.std(X_flattened)
skewness = skew(X_flattened.ravel())
kurt = kurtosis(X_flattened.ravel(), fisher=False)

print(f"Intensidad media: {mean_intensity:.2f}")
print(f"Intensidad mediana: {median_intensity:.2f}")
print(f"Desviación estándar de intensidad: {std_intensity:.2f}")
print(f"Oblicuidad: {skewness:.2f}")
print(f"Curtosis: {kurt:.2f}")

plt.figure(figsize=(10, 6))
sns.histplot(data=X_flattened.ravel(), kde=True)
plt.title("Distribución de intensidad de píxeles")
plt.xlabel("Intensidad")
plt.ylabel("Densidad")
plt.show()
```

```
Intensidad media: 33.39
Intensidad mediana: 0.00
Desviación estándar de intensidad: 78.65
Oblicuidad: 2.15
Curtosis: 5.90
```



Análisis estadístico de la distribución Inmediatamente en el gráfico notamos que por la forma es una distribución bimodal (dos colas, una pesada y otra más liviana), `la mayoría de los píxeles tienen intensidades muy bajas`, cercanas a 0 (negro), con una proporción decreciente de píxeles a medida que aumenta la intensidad, y una cola más liviana en la intensidad más alta (blanco).

La cola derecha indica que existen píxeles con intensidades muy altas, que dado lo que sabemos del dataset MNIST, corresponden a las regiones blancas que representan los dígitos dibujados.

Dispersión: La intensidad media es `33.39`. Esto sugiere que el píxel promedio en el dataset es bastante oscuro. Consistente con los fondos negros que predominan en las imágenes.

La mediana de intensidad es `0`, que indica que más de la mitad de los valores de píxeles son negros.

La desviación estándar de `78.65` marca una dispersión considerable de las intensidades de píxeles alrededor de la media. Pero como estamos analizando una distribución asimétrica la desviación estándar no es el momento más adecuado para considerar.

Asimetría (oblicuidad y curtosis): El valor de oblicuidad de `2.15` indica una distribución sesgada a la derecha. Esto significa que hay más píxeles con valores de intensidad bajos y una cola larga en el extremo más alto de la escala de intensidad. Es decir, la frecuencia de píxeles blancos o claros (alta intensidad) es menor, pero hay suficientes como para extender la cola de la distribución.

Tanto visualmente como numéricamente con el valor de curtosis de `5.90` vemos una distribución leptocúrtica. La distribución tiene colas más pesadas y un pico más agudo que la distribución

normal. Hay una probabilidad más alta de valores extremos en comparación con una normal.

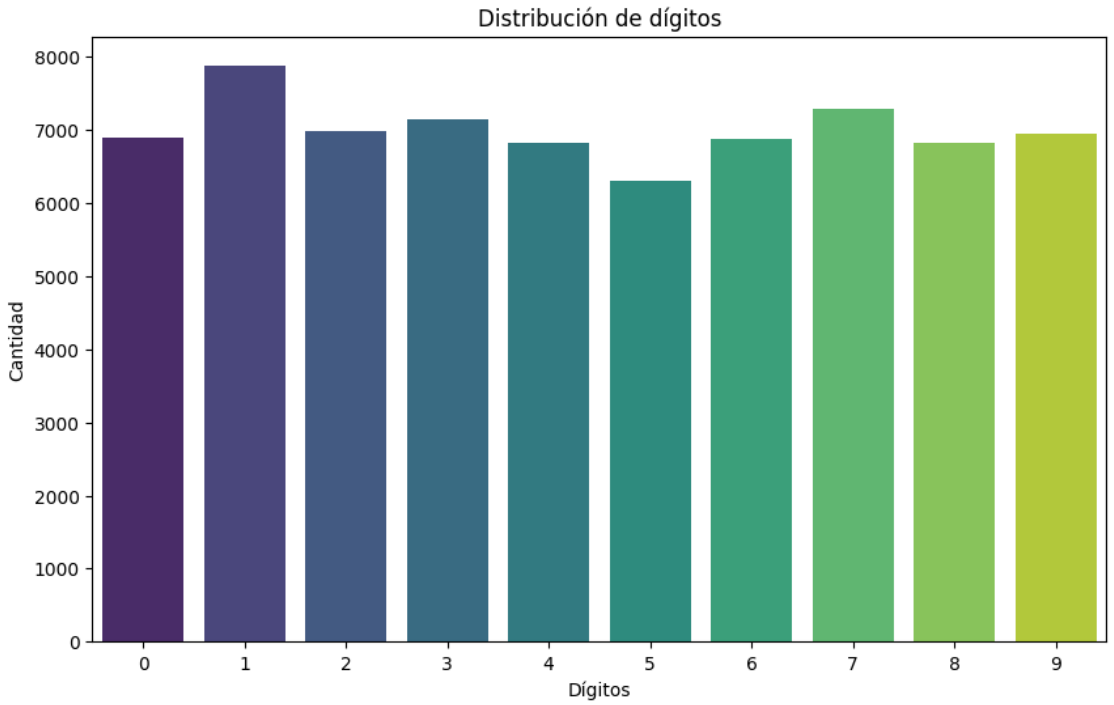
Observaciones: Una potencial extracción de features centrada en patrones blancos (edge detectors, etc.) puede ser efectiva para capturar información relevante de los dígitos, ya que los dígitos son blancos y los fondos negros.

3.3.4 Análisis de variables de salida

Balance de clases El balance de clases es importante para entrenar modelos que no resulten sesgados a una clase (dígito) específica.

Evaluamos la distribución de las clases ploteando la distribución de dígitos (cantidad de ocurrencias cada dígito único) en el dataset.

[ ]:
```
n_samples, n_features = X.reshape((len(X), -1)).shape
n_classes = len(np.unique(y))

# Cantidad de cada dígito
unique, counts = np.unique(y, return_counts=True)

plt.figure(figsize=(10, 6))
sns.barplot(x=unique, y=counts, palette="viridis", hue=unique, legend=False)
plt.xlabel("Dígitos")
plt.ylabel("Cantidad")
plt.title("Distribución de dígitos")
plt.xticks(unique)
plt.grid(axis="y", linestyle="", alpha=0.7)
plt.show()
```



La distribución de los dígitos en MNIST está relativamente balanceada.

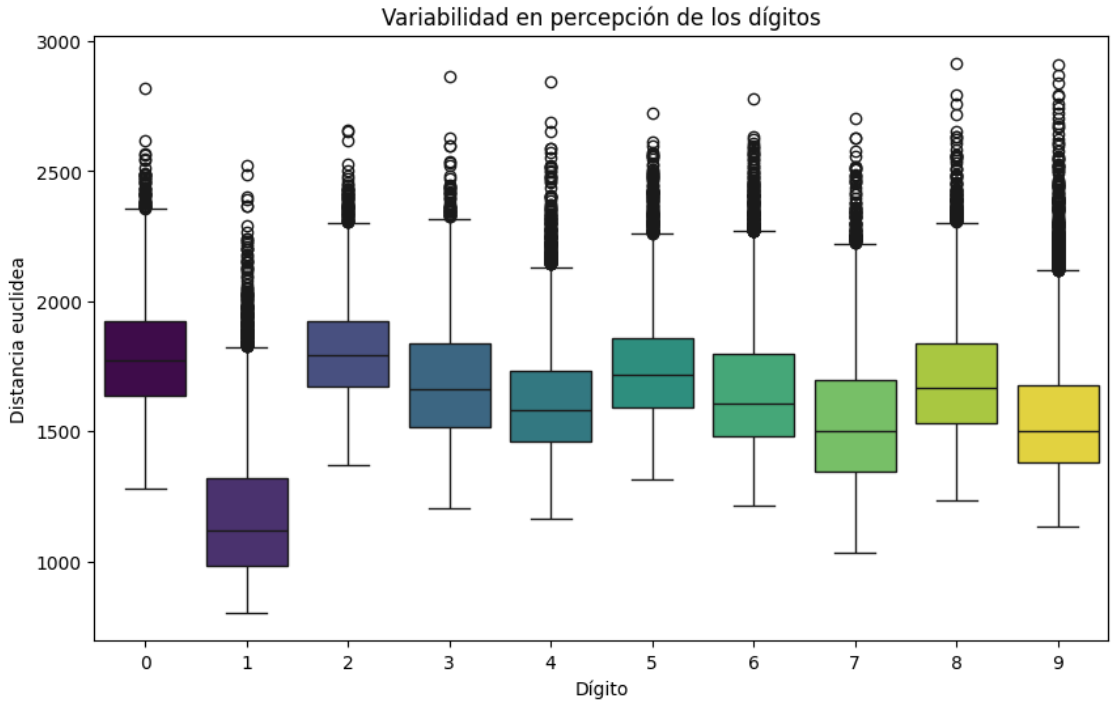
Cada clase (dígito) tiene alrededor de 7000 instancias, con el 1 siendo el de mayor cantidad. Esto indica una buena representación de todas las categorías. Si necesitamos entrenar modelos, este balanceo en el dataset reduce el riesgo de sesgo hacia clases más frecuentes.

Variabilidad y outliers en clasificación Un análisis interesante para realizar con respecto a la variable de salida (dígitos) es observar si en este dataset puede haber muchos outliers a la hora de clasificar. Por intuición, podemos pensar que algunos dígitos pueden ser más fáciles de clasificar que otros. Por ejemplo, un 1 pueden ser relativamente fácil de distinguir (todos dibujan estos dígitos de forma similar) en comparación con la diferencia entre un 5 y un 2, o un 3 y un 8.

Por esta razón vamos a explorar outliers para verificar qué dígitos pueden ser más difíciles de clasificar.

Para esto, se calcula la distancia euclidea de cada imagen al centroide (media) de su label.

[ ]:
```
mean_images = {}
for digit in range(10):
    mean_images[str(digit)] = np.mean(X[y == str(digit)], axis=0)

# Distancia euclidea de cada imagen a la media de su label
distances = []
for i in range(len(X)):
    digit = str(y[i])
    distance = np.sqrt(np.sum((X[i] - mean_images[digit]) ** 2))
    distances.append({"Dígito": int(digit), "Distancia": distance})

df = pd.DataFrame(distances)

plt.figure(figsize=(10, 6))
sns.boxplot(
    x="Dígito", y="Distancia", data=df, hue="Dígito", legend=False,␣
 ↪palette="viridis"
)
plt.xlabel("Dígito")
plt.ylabel("Distancia euclidea")
plt.title("Variabilidad en percepción de los dígitos")
plt.show()
```



Observaciones

- Todos los dígitos dibujados contienen ciertos outliers que pueden ser difíciles de clasificar.
- En la mayoría de los dígitos, no hay mucha variabilidad en la forma en la que están dibujados.
- Los 1 tienen las distancias más bajas: consistente con la intuición planteada anteriormente, no hay mucha variabilidad en la forma en que se dibuja este dígito.
- La mayor variabilidad parece estar en 0 y 2, aunque cada dígito tiene casos con distancias grandes a su instancia promedio.

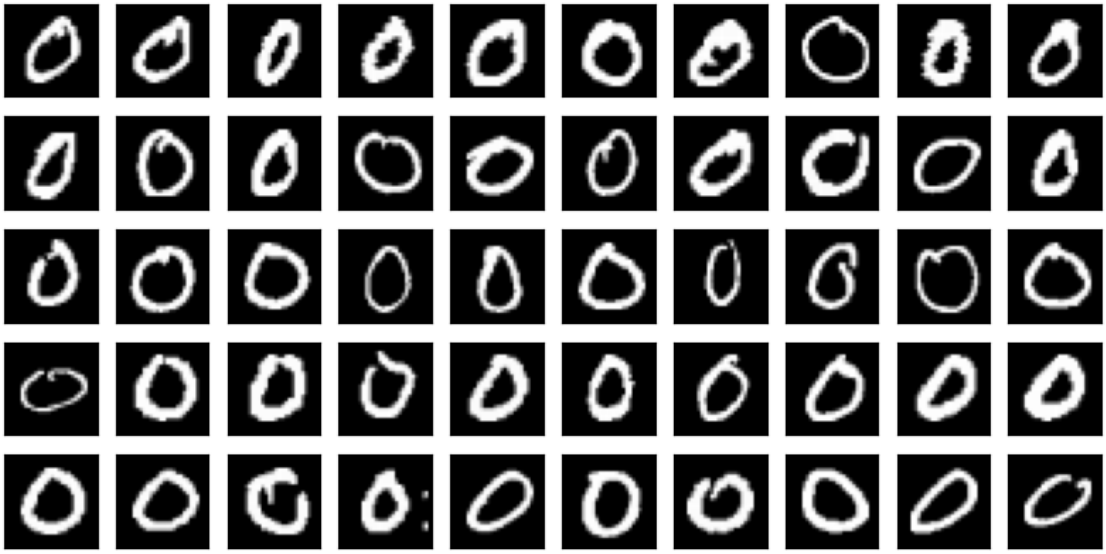
Vamos a visualizar algunos de estos dígitos dibujados.

[ ]:
```
def show_images_by_digit(digit_to_see):
    if digit_to_see in list(range(10)):
        indices = np.where(y == str(digit_to_see))[0]
        plt.figure(figsize=(10, 5))

        n_images = min(len(indices), 50)
        for digit_num in range(n_images):
            plt.subplot(5, 10, digit_num + 1)

            mat_data = X[indices[digit_num]].reshape(28, 28)
            plt.imshow(mat_data, cmap="gray")
            plt.xticks([])
            plt.yticks([])
        plt.show()
```

```
    else:
        print("Dígito debe estar entre 0 y 9.")
```

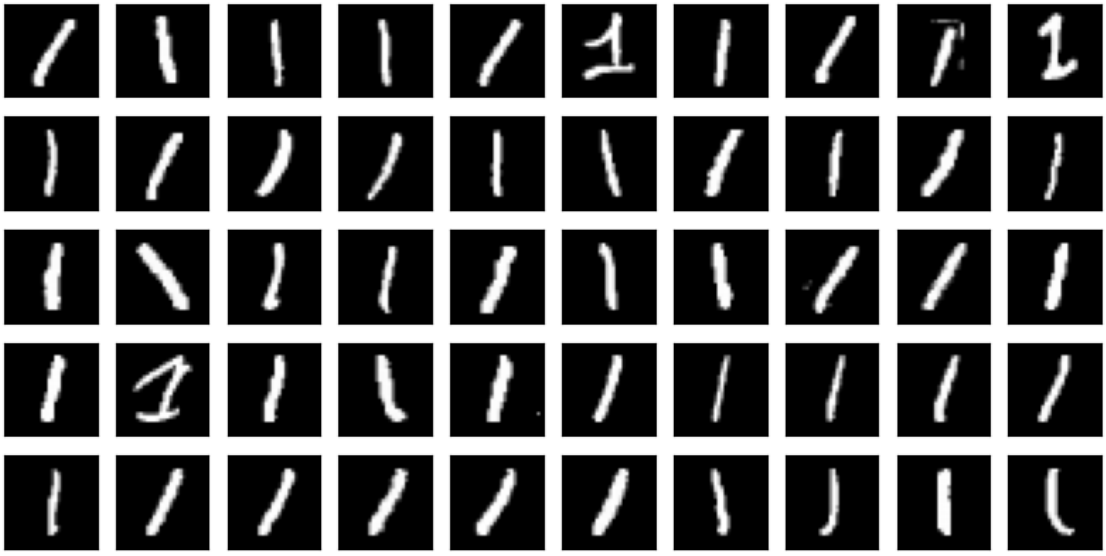
Variabilidad de 0 (ceros)

[ ]:
```
show_images_by_digit(0)
```



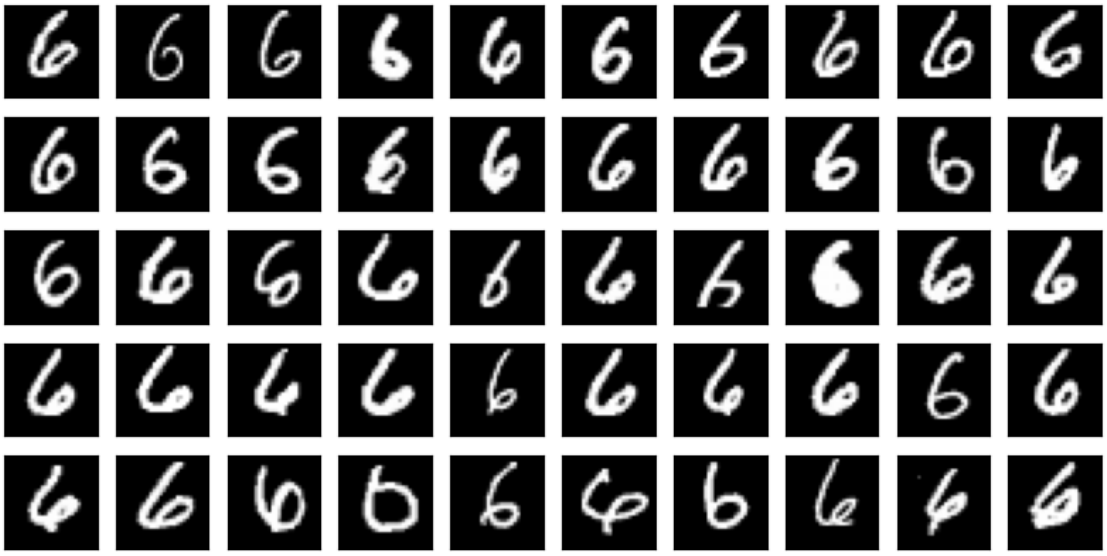
Variabilidad de 1 (unos)

[ ]:
```
show_images_by_digit(1)
```



Variabilidad de 6 (seis)

[ ]:
```
show_images_by_digit(6)
```



4 Limpieza y preparación de datos / ingeniería de features

4.1 Datos faltantes

[ ]:
```
data.head()
```

[ ]:

	0	1	2	3	4	5	6	7	8	9	…	775	776	777	778	779	780	781	782	\
0	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	

	783	target
0	0	5
1	0	0
2	0	4
3	0	1
4	0	9

[5 rows x 785 columns]

[ ]:
```
# Verificamos si hay valores faltantes en el dataset

missing_values = data.isnull().sum()
```

```
missing_values_summary = {
    "Datos faltantes": missing_values.sum(),
    "Datos faltantes for columna": missing_values[missing_values > 0],
}

missing_values_summary
```

```
[ ]: {'Datos faltantes': 0, 'Datos faltantes for columna': Series([], dtype: int64)}
```

En MNIST, todos los píxeles en las imágenes tienen valores del 0 al 255 asignados, con lo cual no es necesario aplicar técnicas de imputación para rellenar datos faltantes.

4.2 Codificación de variables

4.2.1 Variables de Entrada:

No es necesario aplicar una codificación adicional a las variables de entrada. Las imágenes ya están en formato numérico, donde cada píxel se representa con un valor numérico de intensidad. Lo que sí podría ser útil es normalizar los valores para asegurarse de que están en una escala común, dependiendo del algoritmo de ML que se vaya a entrenar.

4.2.2 Codificación de Variables de Salida:

Para los labels (los dígitos del 0 al 9) la necesidad de codificar también depende del algoritmo que se vaya a usar. Para entrenar algoritmos de ML clásico de clasificación multiclase no es necesario, mientras que para utilizar redes neuronales se suele aplicar one-hot encoding.

En MNIST, los dígitos ya están codificados como enteros (0-9). Esta representación numérica es adecuada para muchos algoritmos de ML, con lo cual no es necesario realizar ninguna codificación extra por el momento.

4.3 Relaciones entre variables de entrada

En esta sección vamos a analizar correlaciones entre los píxeles del dataset para identificar si hay ciertas regiones de las imágenes que estén comúnmente relacionadas.

Vamos a hacer énfasis en tratar de encontrar respuestas a las preguntas objetivo planteadas al inicio:

- 1. ¿Se pueden encontrar heurísticas interesantes para clasificar los datos en función de sus valores?
- 2. ¿Es posible encontrar representaciones de baja dimensionalidad que nos permitan visualizar posibles grupos?

Se van a realizar diferentes análisis y visualizaciones usando mecanismos de reducción de dimensionalidad como PCA (Principal Component Analysis) y técnicas de Manifold Learning (aprendizaje de variedades) para identificar combinaciones de píxeles (componentes principales) que capturen la mayor variabilidad en los datos.

4.3.1 Reducción de dimensionalidad

Si consideramos el dataset MNIST, ya mencionamos que consiste en puntos $x^{(i)} \in \mathbb{R}^{784}$. Cada punto siendo una imagen. Sin embargo, si intentáramos generar imágenes aleatorias muestreando uniformemente en este espacio de 784 dimensiones, la probabilidad de que obtengamos un dígito parecido a los de MNIST es muy baja.

Existe una dimensionalidad intrínseca [1] en el espacio ambiente de MNIST, y si bien los puntos están embebidos en un espacio de 784 dimensiones, viven en un subespacio mucho más chico.

Reducir la dimensionalidad nos va a permitir explorar estos subespacios para:

- Inspeccionar y encontrar posibles heurísticas en los datos (pregunta objetivo 1)
- Encontrar posibles clusters (pregunta objetivo 2)
- Intentar entrenar modelos más eficientes, que sólo usen puntos/features de este subespacio

Dependiendo de las suposiciones que hagamos sobre la naturaleza de estos subespacios, vamos a usar técnicas de reducción lineales y no lineales.

4.3.2 Reducción de dimensionalidad lineal - PCA

PCA tiene como objetivo encontrar las direcciones de máxima varianza de los datos en dimensiones altas y proyectar los datos en un nuevo subespacio lineal con menos dimensiones que el original. Los ejes ortogonales (componentes principales) del nuevo subespacio se pueden interpretar como las direcciones de máxima varianza de los datos.

Detalles matemáticos de PCA En PCA, se busca encontrar proyecciones $\tilde{x}_n$ de puntos x_n que sean lo más similares posible a los puntos originales, pero que tengan una dimensionalidad intrínseca menor que la dimensión ambiente (dimensión del dataset original).

Para un dataset $\mathcal{D} = \{x_1, \dots, x_N\}, x_n \in \mathbb{R}^D$ i.i.d. con media 0 y matriz de covarianza

$$S = \frac{1}{N} \sum_{n=1}^{N} x_n x_n^\top.$$

se asume que existe una representación comprimida de bajas dimensiones

$$z_n = B^\top x_n \in \mathbb{R}^M$$

de x_n, donde B una matriz de proyección

$$B := [b_1, \dots, b_M] \in \mathbb{R}^{D \times M}$$

Para encontrar un subespacio M-dimensional de $\mathbb{R}^D$ que retenga la mayor cantidad de información posible, se eligen las columnas de la matriz B como los M autovectores de la matriz de covarianza S que están asociados con los M autovalores más grandes.

La cantidad máxima de varianza que PCA puede capturar con los primeros componentes principales de M es

$$V_M = \sum_{m=1}^{M} \lambda_m$$

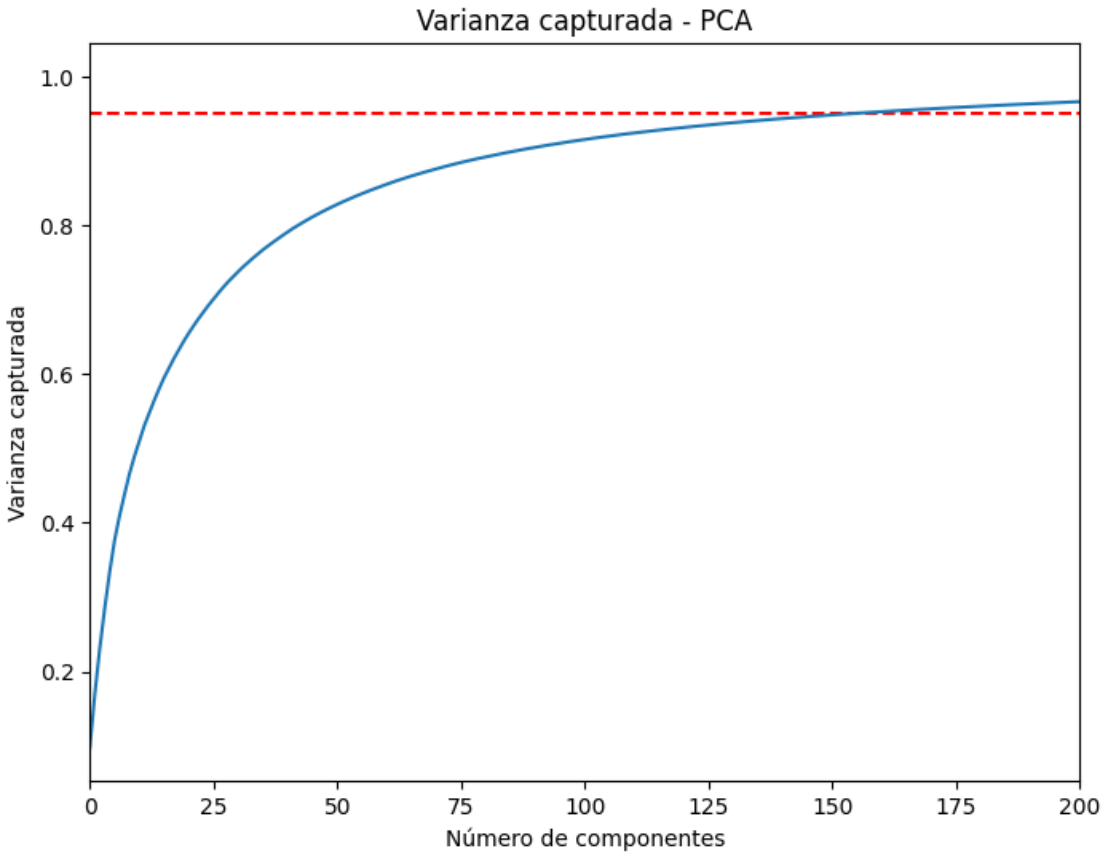
donde λ_m son los M autovalores más grandes de la matriz de covarianza S. La varianza perdida por la compresión de datos a través de PCA es

$$J_M := \sum_{j=M+1}^{D} \lambda_j = V_D - V_M.$$

También se puede definir la varianza relativa capturada como $\frac{V_M}{V_D}$, y la varianza relativa perdida por compresión como $1 - \frac{V_M}{V_D}$.

Varianza capturada por PCA Primero vamos a aplicar PCA sin reducir dimensionalidad, y sólo para visualizar la varianza capturada a medida que la cantidad de componentes aumenta.

[ ]:
```
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(X)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1

plt.figure(figsize=(8, 6))
# Línea en 95% de varianza
plt.plot((0, 800), (0.95, 0.95), "r--")
plt.plot(cumsum)
plt.xlim(0, 200)

plt.title("Varianza capturada - PCA")
plt.xlabel("Número de componentes")
plt.ylabel("Varianza capturada")
plt.show()

print("95% de varianza capturada con " + str(d) + " componentes principales")
```



`95% de varianza capturada con 154 componentes principales`

Después de aplicar PCA al dataset, vemos que, preservando el 95% de la varianza total, nos quedan 154 features, en lugar de los 784 píxeles por imagen originales.

El dataset `es de menos del 20% del tamaño original y solo se perdió el 5% de la varianza`.

En teoría, esto debería mejorar la performance de cualquier algoritmo de clasificación de ML. Vamos a volver a este punto cuando entrenemos los modelos.

Visualización de clusters con PCA Vamos a visualizar MNIST en 2 dimensiones. Los 2 ejes (componentes principales) que mayor varianza capturan.

[ ]:
```
import matplotlib.offsetbox
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA

# Helpers para plotear visualizaciones con PCA y t-SNE
```

```
def get_reduced_data(X_data, reduction_technique="t-SNE"):
    X_data_2D = X_data
    if X_data_2D.shape[-1] == 2:
        return X_data_2D
    if reduction_technique == "PCA":
        pca = PCA(n_components=2)
        X_data_2D = pca.fit_transform(X_data_2D)
    elif reduction_technique == "t-SNE":
        tsne = TSNE()
        X_data_2D = tsne.fit_transform(X_data_2D)
    return X_data_2D


def plot_embeddings(X_data, y_data, reduction_technique="t-SNE", min_distance=0.
 ↪03):
    np.random.seed(42)
    X_data_2D = get_reduced_data(X_data, reduction_technique)
    X_data_2D = (X_data_2D - X_data_2D.min()) / (X_data_2D.max() - X_data_2D.
 ↪min())

    fig = plt.figure(figsize=(10, 8))
    cmap = plt.cm.tab10
    plt.scatter(X_data_2D[:, 0], X_data_2D[:, 1], c=y_data, s=10, cmap=cmap)
    image_positions = np.array([[1.0, 1.0]])

    for index, position in enumerate(X_data_2D):
        dist = np.sum((position - image_positions) ** 2, axis=1)
        if np.min(dist) > min_distance:
            image_positions = np.r_[image_positions, [position]]
            image_shape = int(np.sqrt(X_data[index].shape[0]))
            image_data = X_data[index].reshape(image_shape, image_shape)
            imagebox = matplotlib.offsetbox.AnnotationBbox(
                matplotlib.offsetbox.OffsetImage(image_data, cmap="binary"),
                position,
                bboxprops={"edgecolor": tuple(cmap([y_data[index]])[0]), "lw":␣
 ↪2},
            )
            plt.gca().add_artist(imagebox)
    plt.title(f"Embedding - {reduction_technique}")
    return fig
```

[ ]:
```
# Elegimos una parte del dataset
X_p = X[:20000]
y_p = y[:20000]
y_p = y_p.astype(int)

fig = plot_embeddings(X_p, y_p, reduction_technique="PCA")
```

```
plt.show()
```

Podemos identificar algunos clusters que se diferencian más que otros.

En particular, los dígitos `0` y `1`.

Como habíamos visto en los box plots cuando analizamos la variabilidad, el `0` destacaba por ser el dígito que mayor variabilidad contenía (los dibujos variaban considerablemente entre instancias). En este gráfico, vemos lo mismo pero con otra representación. A lo largo del eje `x` (primer componente principal) el `0` parece ser el que más varía.

Por otra parte, el `1` tenía la menor variabilidad, y podemos verlo también acá. Es un cluster chico a lo largo del primer componente, aunque bien marcado.

Utilidad de PCA para visualizar clusters Por lo que vemos, incluso inspeccionando el dataset desde el punto de vista de los componentes que explican mejor la varianza, los datos MNIST parecen no formar grupos/clusters bien definidos salvo excepciones.

Como habíamos remarcado, PCA es una técnica que es mayoritariamente útil si los datos se encuentran aproximadamente en un subespacio lineal.

MNIST parece ser una estructura de altas dimensiones menos trivial, y este tipo de proyecciones lineales no van a ser suficientes para intentar encontrar grupos bien marcados.

`Sin embargo, vamos a volver a PCA más adelante para intentar analizar la importancia de las features.`

4.3.3 Reducción de dimensionalidad no lineal - Manifold learning

Analizando MNIST con PCA, vimos que la estructura de las imágenes en el dataset no parece formar un subespacio lineal. Existe una hipótesis en ML llamada *manifold hypothesis* [3] o hipótesis de variedades: La mayoría de los datasets de altas dimensiones se encuentran cerca de una variedad de dimensiones mucho más bajas. Para intentar aprender la geometría y la estructura de esta variedad se usan técnicas de reducción de dimensionalidad no lineal.

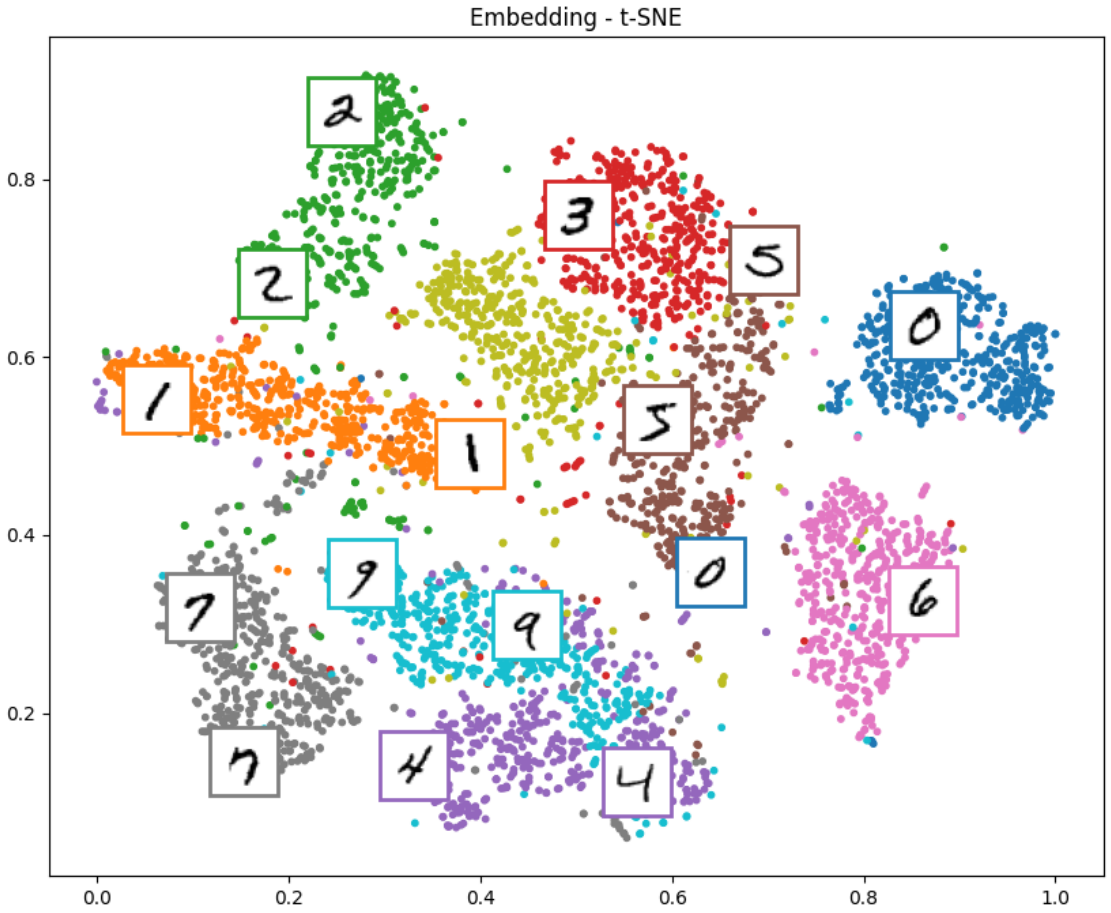
Los algoritmos de reducción de dimensionalidad no lineal modelan el manifold (variedad) no lineal en el que viven los puntos del dataset. Esto se denomina manifold learning o aprendizaje de variedades.

t-SNE Una técnica popular es *t-SNE* (t-distributed Stochastic Neighbor Embedding) [4].

t-SNE modela los datos en función de la distancias en el espacio de alta dimensiones del dataset original. Busca una distribución de probabilidad de distancias en el nuevo espacio de menos dimensiones que esté cerca de la distribución de probabilidad de distancias en el espacio original. De esta forma, aprende a embedear puntos en un espacio de dimensiones más bajas conservando las distancias del espacio original.

Aplicado a MNIST, t-SNE podría separar los diferentes dígitos en distintos clusters e incluso capturar subclusters que representen diferentes formas de escribir el mismo dígito, y nos puede descrbir mejor la estructura intrínseca de las imágenes (similitudes en el estilo del dibujo, variaciones, etc. que un método lineal como PCA no captura).

[ ]:
```
# Tomamos una parte del dataset

X_tsne = X[:5000]
y_psne = y[:5000]
y_psne = y_psne.astype(int)

fig = plot_embeddings(X_tsne, y_psne, reduction_technique="t-SNE")
plt.show()
```



Inmediatamente vemos que t-SNE forma clusters bien marcados y separados, al contrario que PCA.

Esto indica que para visualizar MNIST es mejor utilizar algoritmos de reducción de dimensionalidad no lineales que modelen el manifold donde viven los datos.

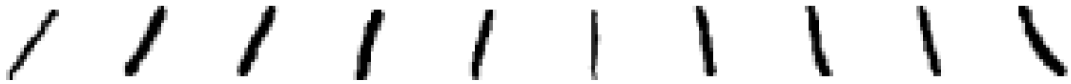
Como mencionamos, t-SNE parece capturar subclusters que representan diferentes formas de escribir el mismo dígito, como en el caso del `1`.

Vamos a ver esto en detalle.

[ ]:
```
# Seleccionar 1s
X_1 = X[y == "1"]
one_indices = [5563, 7316, 7695, 4467, 7160, 7065, 6433, 7430, 6166, 2980]
selected_images = X_1[one_indices]
```

[ ]:
```
fig, axes = plt.subplots(1, 10, figsize=(12, 2))

for i, ax in enumerate(axes):
    img = selected_images[i].reshape(28, 28)
    ax.imshow(img, cmap="binary")
```

```
    ax.axis("off")

plt.tight_layout()
plt.show()
```



Se puede ver que a medida que nos movemos hacia la derecha en el cluster formado por t-SNE, la inclinación de los 1 cambia hacia la izquierda. Esta es una de las variaciones intra-dígitos que pudo ser captada por t-SNE.

4.4 Importancia de features/variables

Antes de entrenar los modelos de ML, nos va a ser útil chequear e identificar variables/features de mayor importancia.

Para esto vamos a utilizar algunas técnicas de feature engineering para identificar qué píxeles son más importantes en la clasificación.

Volvamos a PCA y veamos qué pasa si tomamos las features (784 píxeles por imagen) y aplicamos PCA para descomponer los vectores de features en autovalores.

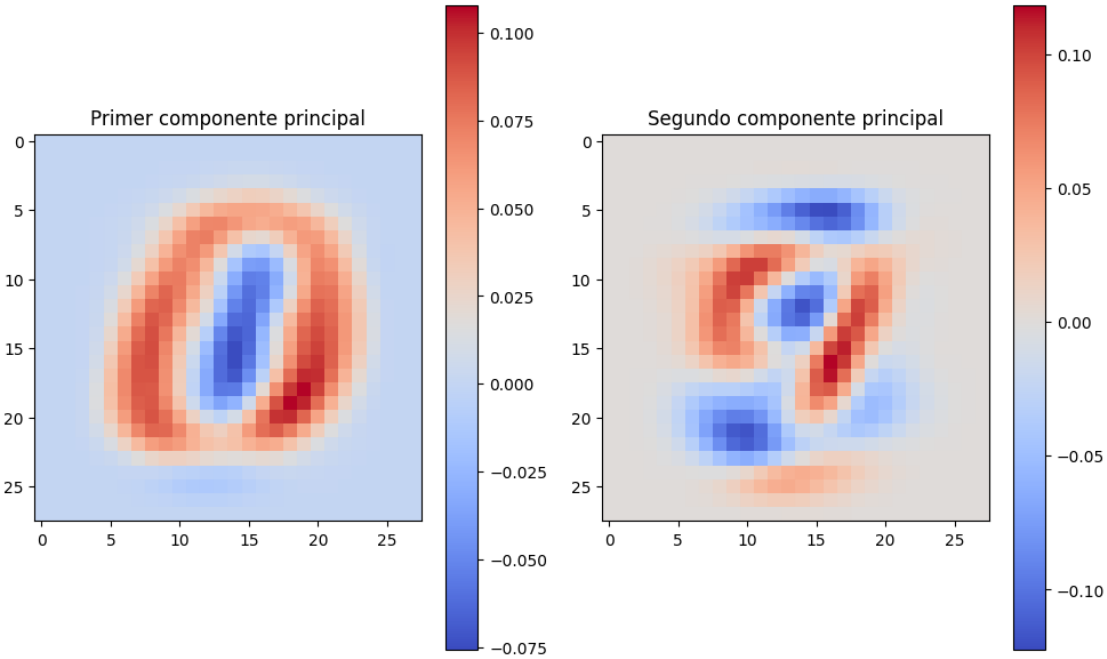
4.4.1 Eigendígitos (autodígitos)

Cuando utilizamos PCA anteriormente, ploteamos los dos componentes principales para visualizar los clusters que se forman projectados sobre los ejes que capturan la mayor varianza.

Ahora vamos a visualizar los componentes principales como imágenes.

[ ]:
```
# PCA con 2 componentes
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Componentes
first_component = pca.components_[0]
second_component = pca.components_[1]

# Reshapear componentes en forma de imagen 28x28
first_component_image = first_component.reshape(28, 28)
second_component_image = second_component.reshape(28, 28)

plt.figure(figsize=(10, 6))
plt.subplot(1, 2, 1)
plt.imshow(first_component_image, cmap="coolwarm", interpolation="none")
plt.title("Primer componente principal")
```

```
plt.colorbar()

plt.subplot(1, 2, 2)
plt.imshow(second_component_image, cmap="coolwarm", interpolation="none")
plt.title("Segundo componente principal")
plt.colorbar()

plt.tight_layout()
plt.show()
```



Cada imagen representa un componente principal y cada color de píxel indica el peso de ese píxel en el componente.

- Primer componente principal: Muestra las áreas de las imágenes de dígitos que más varían en el dataset. Los píxeles en rojo contribuyen positivamente, mientras que los de azul contribuyen negativamente a este componente. Este componente captura el mayor porcentaje de varianza del dataset.
- Segundo componente principal: Es ortogonal al primero y captura la mayor varianza del segundo. Las áreas roja y azul muestran dónde ocurre el segundo conjunto de variaciones en las imágenes.

Al igual que a las eigenfaces [2], a estos los podríamos llamar `eigendígitos` o `autodígitos`.

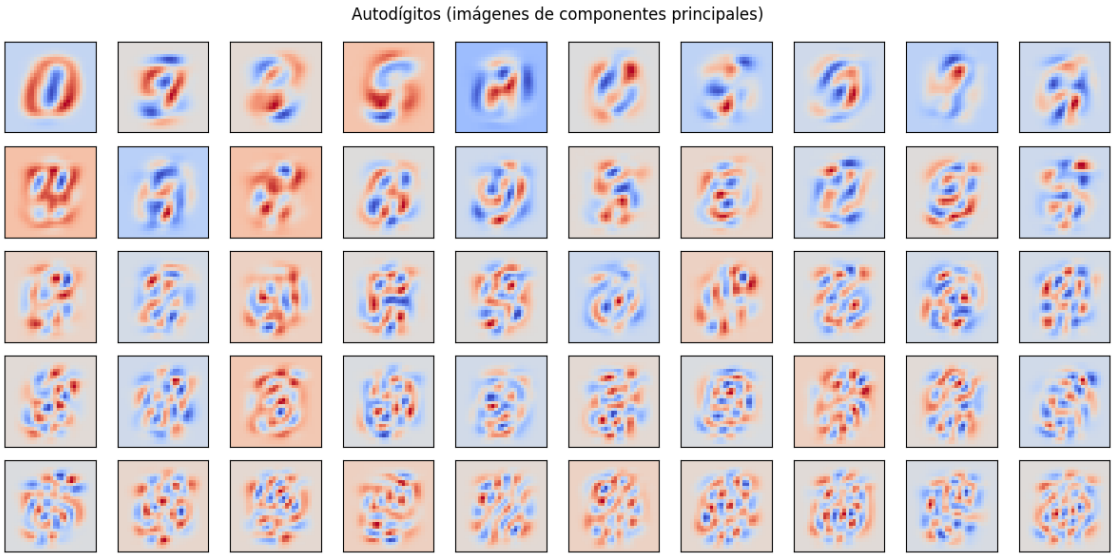
Importancia de píxeles Un píxel con un valor positivo alto (rojo) en un componente principal significa que el píxel, cuando tiene una alta intensidad (brillo) en la imagen original, contribuye de forma significativa a la varianza a lo largo de el componente.

Un píxel con un valor negativo alto (azul) significa que una intensidad alta en este píxel contribuye negativamente a la varianza a lo largo de el componente.

En el contexto de MNIST, estas visualizaciones nos permiten ver qué partes de las imágenes (qué píxeles) son más críticas e importantes para diferenciar entre los dígitos. Por ejemplo, si las áreas rojas en el primer componente están ubicadas donde un dígito en particular generalmente tiene features únicas (el centro del cero), indica que estas features son importantes para identificar ese dígito.

Primeros 50 componentes principales

[ ]:
```
n_components = 50
fig, axes = plt.subplots(
    5, 10, figsize=(12, 6), subplot_kw={"xticks": [], "yticks": []}
)
axes = axes.flatten()

pca_eigen = PCA(n_components)
pca_eigen_fit = pca_eigen.fit_transform(X)

for i in range(n_components):
    eigendigit = pca_eigen.components_[i].reshape(28, 28)
    axes[i].imshow(eigendigit, cmap="coolwarm", interpolation="none")

fig.suptitle(f"Autodígitos (imágenes de componentes principales)")
plt.tight_layout()
plt.show()
```



Las imágenes muestran cómo se ven los autovalores de la matriz de covarianza cuando se los reshapea como las imágenes originales del dataset.

Con estas 50 imágenes vemos cómo los componentes de PCA intentan capturar transformaciones lineales del dataet. Cada componente intenta capturar un determinado "aspecto" de las imágenes. Por ejemplo, la primera (y más importante) imagen aparentemente captura el dígito `0`, es decir, qué tan parecido a un `0` se ve el dígito.

Los primeros 10 componentes principales parecen conservar parte de la forma de los dígitos y después de eso, se convierten imágenes sin ningún dígito reconocible.

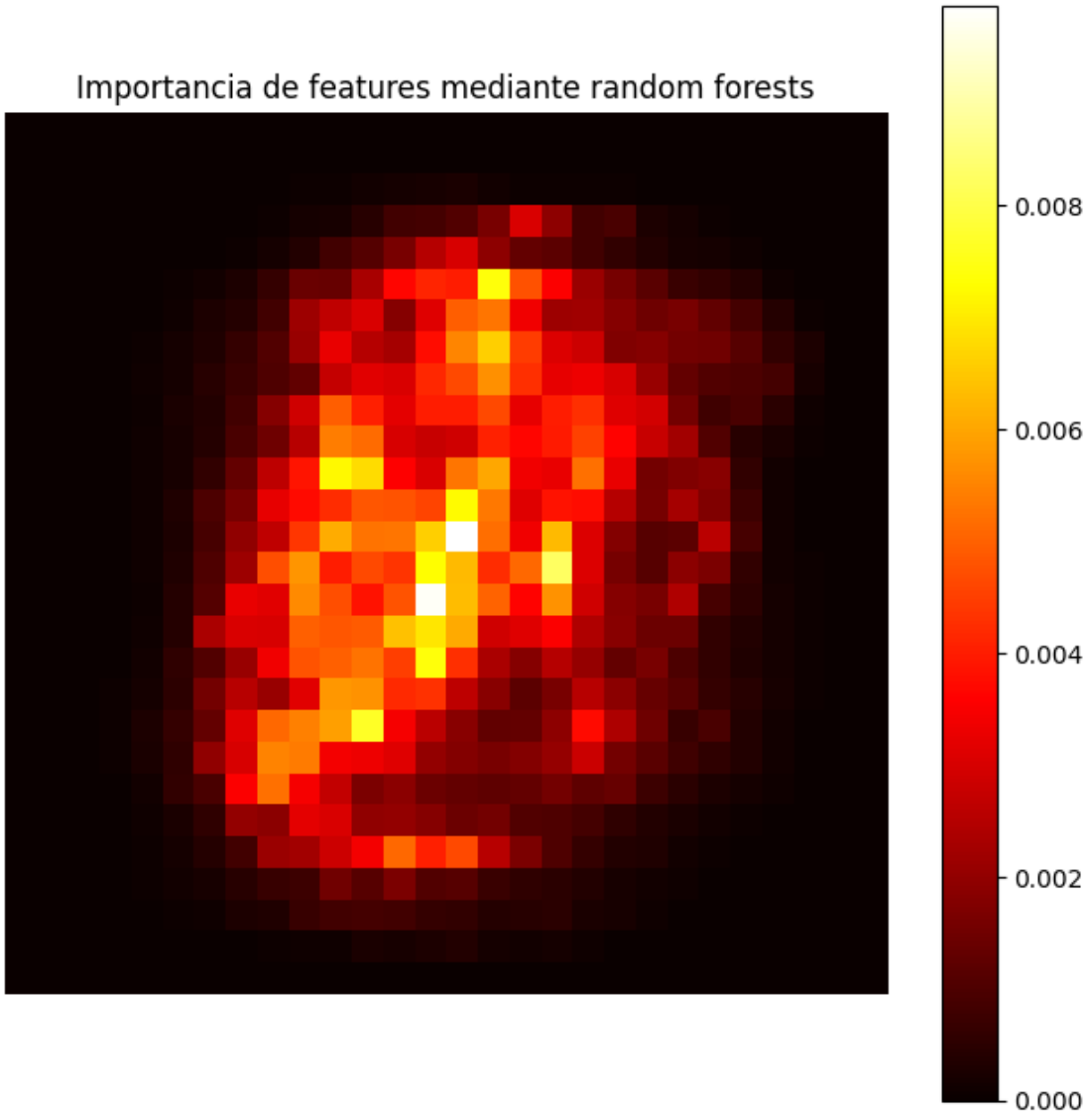
Importancia para entrenamiento de modelos Dado que los autodígitos son los componentes principales obtenidos con PCA, son combinaciones lineales de las features de los píxeles originales y pueden verse como templates que capturan features comunes e importantes de los dígitos. Cuando entrenemos los modelos, vamos a hacerlo con los autodígitos como inputs, lo que debería mejorar la performance.

4.4.2 Importancia de features mediante random forests

Otra manera de analizar qué píxeles son importantes a la hora de clasificar es fiteando random forests, dado que proporcionan esa información para cada feature después de entrenar el modelo.

Las importancias de las features se calcula como la media y la desviación estándar de la acumulación de impurezas dentro de cada árbol en el random forest.

[ ]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)

importances = rf.feature_importances_
importances_reshaped = importances.reshape(28, 28)

plt.figure(figsize=(8, 8))
plt.imshow(importances_reshaped, cmap="hot", interpolation="none")
plt.title("Importancia de features mediante random forests")
plt.colorbar()
plt.axis("off")
plt.show()
```



Vemos en el heatmap la importancia de cada píxel en la clasificación. Cuanto más brilloso (más hacia el rojo y el amarillo), más importante es el píxel que el random forest considera al hacer predicciones.

Este ploteo es bastante útil porque nos indica directamente qué partes de las imágenes (qué píxeles) son más críticas para que el modelo clasifique los dígitos de forma correcta. Los píxeles con mayor importancia (áreas más brillantes) son aquellos en los que el modelo se basa más para diferenciar entre los distintos dígitos.

Se puede ver que los píxeles cercanos a los bordes son los menos importantes. Esto lo podemos usar para reducir la cantidad de píxeles en cada imagen sin perder performance y accuracy cuando clasifiquemos.

5 Entrenamiento de modelos

Recurriendo a los modelos que conozca, defina una lista de modelos candidatos a entrenar (puede ser el mismo tipo de clasificador con distintos hiperparámetros). Nota: no se contemplará el desempeño del modelo elegido, sino las conclusiones que puedan establecerse a partir de la preparación previa de los datos. Ensaye distintas cadenas de procesamiento con las técnicas consideradas en la sección 3 (por ejemplo, distintas técnicas de imputación, selección de variables de entrada, codificación de variables categóricas, transformación, etc.).

Separación de datos Los pasos siguientes comprenden las etapas de preparación de datos y evaluación de resultados. Para ello, se debe particionar el dataset en entrenamiento y validación.

Evaluación de resultados ¿Qué puede concluir acerca de los modelos y preparaciones de datos ensayadas? Tener en cuenta como cada preparación afecta a los distintos modelos.

TBD!!

6 Referencias

[1] On the Intrinsic Dimensionality of Image Representations: https://arxiv.org/abs/1803.09672

[2] Low-dimensional procedure for the characterization of human faces: https://www.cs.bgu.ac.il/~ben-shahar/Teaching/Computational-Vision/Readings/1987-Sirovich_and_Kirby-Low_Dimensional_Procedure_for_the_Characterization_of_Human_Faces.pdf

[3] Manifold Hypothesis - Wikipedia: https://en.wikipedia.org/wiki/Manifold_hypothesis

[4] Visualizing Data using t-SNE: https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf