UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Integrantes:

- Maydelith Laura Zuñiga Cabrera - 202510437
- Joaquin Mathias Alejandro Llallire Meza - 202510406
- Joseph Paulo Geraldo Soto - 202510412
- Leonardo Julian Huanay Quiroz - 202510516

Nombre del estudiante	Porcentaje de aporte
Maydelith Laura Zuñiga Cabrera	30%
Joaquin Mathias Alejandro Llallire Meza	20%
Joseph Paulo Geraldo Soto	20%
Juan Manuel Moreno Moran	0%
Leonardo Julian Huanay Quiroz	30%

Actividades realizadas por integrantes:

Informe:

Marco teórico y Antecedentes: Joseph Paulo Geraldo Soto

Descripciones del proceso realizado por pregunta: Maydelith Laura Zuñiga Cabrera

Interpretación final: Joaquin Mathias Alejandro Llallire Meza y Leonardo Julian Huanay Quiroz

Conclusiones: Todos

Código:

Json: Leonardo Julian Huanay Quiroz

Parte a y b: Joaquin Mathias Alejandro Llallire Meza y Joseph Paulo Geraldo Soto

Parte c,d,e: Leonardo Julian Huanay Quiroz

Parte f y g: Maydelith Laura Zuñiga Cabrera

Antecedentes:

Computer vision es una área dentro de la inteligencia artificial que permite a las máquinas interpretar y comprender imágenes de manera similar a como lo hacen los seres humanos. A lo largo de las últimas décadas, esta tecnología ha experimentado avances significativos, desde los primeros intentos de reconocimiento de patrones hasta la integración de redes neuronales profundas que permiten la clasificación y el análisis de imágenes complejas. Hoy en día, es una herramienta clave en una amplia variedad de aplicaciones, como el reconocimiento de números dentro de un espacio y desarrollo de sistemas médicos, transformando industrias y mejorando la eficiencia en tareas analíticas.

Marco Teórico:

1. ¿Qué es scikit-learn?

   Es un conjunto de rutinas escritas en Python para hacer análisis predictivo, que incluyen clasificadores, algoritmos de clusterización, etc. Está basada en NumPy, SciPy y matplotlib, de forma que es fácil aprovechar el código que usan estas librerías (Universitat Oberta de Catalunya, 2024)

2. ¿Qué es el dataset "digits" disponible en el paquete scikit-learn?

   El dataset "digits" disponible en el paquete scikit-learn es un conjunto de datos utilizado para tareas de clasificación en machine learning. Este dataset contiene imágenes en escala de grises de dígitos escritos a mano, con el objetivo de predecir qué dígito (de 0 a 9) representa cada imagen (scikit-learn, 2020)

3. ¿Qué es cv2?

   Librería en Python que hace parte de OpenCV (Open Source Computer Vision Library), una biblioteca de código abierto para procesamiento de imágenes y visión por computadora. Esta librería proporciona una amplia gama de funciones que permiten realizar tareas de análisis y manipulación de imágenes y videos, como el procesamiento de imágenes en tiempo real, detección de objetos, reconocimiento facial, filtrado, transformaciones geométricas, y mucho más. (inesem, 2020)

4. ¿Qué es NumPy?

   Librería de Python fundamental para el cálculo numérico y la manipulación de grandes volúmenes de datos. Proporciona soporte para arrays multidimensionales (listas o matrices de más de una dimensión) y ofrece una serie de funciones matemáticas para realizar operaciones sobre estos arrays de manera eficiente. (NumPy, 2024)

5. ¿Qué es archivo CSV?

   Archivo de texto que almacena datos en formato de tabla, donde cada línea representa un registro y cada campo está separado por una coma. (Microsoft, 2025)

6. ¿Qué es el paquete Pandas?

   Pandas es una biblioteca de Python de código abierto ampliamente utilizada para el análisis de datos y el aprendizaje automático.(University of Virginia, 2024)

7. ¿Qué es OS?

   Es un módulo estándar en Python que proporciona una interfaz para interactuar con el sistema operativo. Permite realizar diversas operaciones relacionadas con el sistema de archivos, procesos y entorno del sistema operativo, como la manipulación de directorios y archivos, obtención de información sobre el sistema, y la ejecución de comandos del sistema operativo.(Python, 2022)

8. ¿Qué es openpyxl?

   Librería de Python utilizada para trabajar con archivos de Excel en el formato .xlsx (Excel 2010 y versiones posteriores). Permite leer, escribir y modificar hojas de cálculo de Excel sin necesidad de tener instalado Excel en el sistema.(Python, 2024)

9. KNN: Algoritmo de aprendizaje supervisado utilizado para tareas de clasificación y regresión. El principio básico de KNN es que la clasificación de un punto de datos se basa en la clase de sus K vecinos más cercanos en el espacio de características. Es decir, para predecir la etiqueta de un punto, el algoritmo busca los K puntos más cercanos a él en el conjunto de entrenamiento y asigna la clase más frecuente (en clasificación) o el valor promedio (en regresión) de esos vecinos. (IBM, 2023)

Visualización de imágenes promedio de dígitos:

Este código implementa un proceso que permite visualizar la imagen promedio de cada uno de los dígitos del conjunto de datos de dígitos de scikit-learn, utilizando el promedio de las imágenes de cada dígito. A continuación, se detallan los pasos realizados y cómo cumplen con los requisitos solicitados:

Carga del conjunto de datos "digits":

El código utiliza la función datasets.load_digits() de scikit-learn para cargar el conjunto de datos de dígitos, que contiene imágenes de dígitos del 0 al 9, junto con sus etiquetas.

Cálculo del promedio de las imágenes:

La función calcular_promedio() genera una matriz promedio para cada dígito (de 0 a 9).

Para esto:

- Se crean diccionarios para almacenar las imágenes correspondientes a cada dígito.
- Luego, se calcula el promedio de las imágenes de cada dígito utilizando np.mean() a lo largo de las imágenes asociadas a cada clase. Este promedio se almacena en el diccionario promedios.

Visualización del promedio de un dígito específico:

La función principal main() permite al usuario interactuar con el programa. Se le solicita al usuario que ingrese un número del 0 al 9. Si el número es válido, el programa imprime la matriz promedio de ese número, redondeada a dos decimales y con formato visual limpio.

El código también maneja entradas inválidas, pidiendo que el número esté en el rango de 0 a 9 y que sea un valor numérico.

Visualización de la matriz:

La matriz promedio de cada dígito se imprime de forma que se puede visualizar su estructura de 64 píxeles (8x8) con valores numéricos representando los niveles de

intensidad de los píxeles. Esta matriz proporciona una visión general de cómo se ven las imágenes promedio de los dígitos en el conjunto de datos.

Flujo de ejecución:

Al ejecutarse, el código permite al usuario solicitar la imagen promedio de cualquier número entre 0 y 9. Después de ingresar el número, el programa imprime la matriz promedio correspondiente.

Cálculo y exportación de matrices promedio de dígitos:

Este código organiza los números de 8x8 en un formato tabular y los exporta tanto en formato Excel como CSV. A continuación, se detallan los pasos realizados y cómo cumplen con los requisitos solicitados:

Carga del conjunto de datos "digits":

El código utiliza la función datasets.load_digits() de scikit-learn para cargar el conjunto de datos de dígitos, que incluye imágenes de dígitos del 0 al 9 junto con sus etiquetas correspondientes.

Cálculo del promedio de las imágenes:

Se crea un diccionario promedios donde se almacenan las imágenes promedio para cada dígito (0-9).

Para cada dígito, se seleccionan las imágenes correspondientes utilizando losDigitos.images[losDigitos.target == numero], y luego se calcula el promedio de esas imágenes utilizando np.mean() a lo largo del eje 0 (promediando las imágenes de cada dígito).

El promedio de cada imagen se redondea a dos decimales con np.round() y se guarda en el diccionario promedios.

Creación de un DataFrame con las matrices promedio:

Se inicializa un DataFrame vacío final_df.

Para cada dígito (0-9), se crea un DataFrame a partir de la matriz promedio de ese dígito. Las columnas del DataFrame se etiquetan como C0, C1, ..., C7

(representando las 8 columnas de la imagen de 8x8).

Se inserta una columna "Número" al principio del DataFrame, que indica el dígito correspondiente.

El DataFrame de cada dígito se concatena con el final_df utilizando pd.concat(), lo que va formando una tabla con las imágenes promedio de todos los dígitos.

Además, se añade una fila vacía como separador entre cada matriz de dígitos para mejorar la legibilidad del archivo.

Exportación a Excel y CSV:

Finalmente, el DataFrame final_df se exporta a un archivo Excel llamado "matrices_promedio_vertical.xlsx" y a un archivo CSV llamado "matrices_promedio_vertical.csv", ambos sin los índices, utilizando los métodos to_excel() y to_csv() de pandas.

Formato condicional:

Proceso para aplicar formato condicional a un archivo Excel seleccionado por el usuario, utilizando una escala de colores basada en los valores de las celdas. A continuación, se detallan los pasos realizados y cómo cumplen con los requisitos solicitados:

Carga de archivos Excel disponibles en el directorio:

El código obtiene todos los archivos con extensión .xlsx en el directorio actual utilizando os.listdir() y filtra los que son archivos Excel, almacenándolos en la lista files.

Selección del archivo Excel:

El usuario selecciona un archivo Excel de la lista proporcionada mediante un input. Si la selección es válida (índice dentro del rango), el archivo se carga utilizando openpyxl.load_workbook().

Definición de la regla de formato condicional:

Se define una regla de formato condicional utilizando ColorScaleRule, que aplica una escala de colores en función de los valores de las celdas: blanco para los valores mínimos, amarillo para los valores intermedios (percentil 50) y rojo para los valores máximos.

Aplicación del formato condicional:

La regla de formato condicional se aplica a un rango específico de celdas (de la columna 'B' a la 'I' para cada conjunto de 8 filas) en el archivo Excel. Si la celda de la columna 'A' está vacía, el programa omite esa fila.

Guardado del archivo modificado:

El archivo Excel con el formato condicional aplicado se guarda con un nuevo nombre, agregando el sufijo "_colored" al nombre original del archivo. El archivo se guarda utilizando el método save() de openpyxl.

Matrices de números:

0:

0	0.02	4.19	13.1	11.3	2.93	0.03	0
0	0.89	12.58	13.37	11.49	11.38	0.97	0
0	3.73	14.28	5.26	2.1	12.17	3.52	0
0	5.29	12.71	1.99	0.14	9.06	6.45	0
0	5.87	11.56	0.89	0.04	8.78	7.12	0
0	3.49	13.29	1.65	1.53	11.31	5.85	0
0	0.8	13.06	9.96	10.35	13.25	2.42	0
0	0.01	4.16	13.56	13.33	5.44	0.28	0

1:

1	0	0.01	2.46	9.21	10.41	6.08	0.99	0
1	0	0.09	4.07	12.76	13.95	8.45	1.2	0
1	0.01	1.1	7.05	14.86	14.16	7.39	0.7	0
1	0.01	2.13	9.08	14.29	13.86	6.25	0.37	0
1	0	1.15	6.83	11.87	13.69	5.55	0.36	0
1	0	0.43	5.18	10.29	13.41	5.88	0.35	0
1	0	0.12	4.75	10.95	13.58	7.69	2.05	0.64
1	0	0.01	2.24	9.14	13.03	8.58	3.05	1.49

2:

2	0	0.93	9.67	14.19	9.63	2.38	0.11	0
2	0.01	5.28	13.81	12.11	12.56	5.5	0.42	0
2	0.01	4.54	7.85	4.54	11.6	5.93	0.47	0
2	0	0.86	1.73	4.74	12.1	4.44	0.25	0
2	0	0.06	1.23	8.76	10.49	2.27	0.06	0
2	0	0.58	5.36	11.86	7.27	2.06	0.96	0.02
2	0.02	1.42	11.18	14.28	12.03	10.64	7.18	0.71
2	0.01	0.94	10.1	13.97	13.12	11.8	8.02	1.93

3:

3	0	0.64	8.39	14.17	14.22	7.48	0.79	0.01
3	0.01	4.21	12.66	8.99	11.28	11.99	2.11	0.02
3	0.01	2.22	3.7	3.11	12.03	9.32	0.81	0
3	0	0.3	1.46	8.94	14.27	5.61	0.08	0
3	0	0.06	1.05	5.64	12.05	11.28	2.2	0
3	0	0.43	1.4	0.97	4.41	12.14	6.32	0
3	0	0.87	7.11	6.23	8.26	13.02	5.93	0.07
3	0	0.5	9.32	14.65	13.97	8.67	1.41	0.07

4:

4	0	0	0.45	7.06	11.5	2.01	0.21	0.13
4	0	0.06	3.48	13.27	8.45	2.05	1.14	0.3
4	0	0.72	10.48	11.4	4.88	5.79	3.85	0.34
4	0.01	4.64	14.49	6.35	7.35	10.63	6.13	0.02
4	0	8.58	14.59	9.71	13.02	13.95	5.3	0
4	0.09	6.3	11.27	12.4	14.62	10.44	1.61	0
4	0.06	1.09	2.91	7.82	13.45	4.16	0.02	0
4	0	0.02	0.55	7.81	11.81	1.96	0	0

5:

5	0	0.97	9.98	13.04	13.9	12.18	4.14	0.04
5	0.01	3.9	14.8	12.03	8.24	6.23	2.12	0.03
5	0	5.59	14.35	5.7	1.93	0.58	0.05	0
5	0	5.37	14.32	12.04	8.97	4.16	0.38	0
5	0	1.94	7.65	8.43	8.84	7.64	1.81	0
5	0	0.25	1.26	3.67	7.39	8.18	2.38	0
5	0	0.83	5.77	8.19	10.84	7.32	1.34	0
5	0	0.96	10.66	14.74	9.36	2.53	0.2	0

6:

6	0	0	1.14	11.17	9.59	1.45	0.01	0
6	0	0.03	7.13	14.52	6.2	0.88	0.05	0
6	0	0.74	12.52	9.52	0.93	0.1	0.01	0
6	0	2.31	13.7	8.11	3.88	2.03	0.14	0
6	0	3.58	14.74	12.88	12.03	10.34	2.83	0
6	0	1.93	14.69	10.72	5.59	10.12	9.22	0.23
6	0	0.19	10.37	12.66	5.54	11.28	10.78	0.52
6	0	0	1.44	10.69	15.09	13.04	4.48	0.09

7:

7	0	0.17	5.1	13.06	14.25	11.03	5.19	1
7	0	1.09	10.53	11.71	11.17	12.44	5.36	0.61
7	0	0.82	4.83	2.32	7.09	11.26	3.18	0.12
7	0	0.61	4.35	6.25	12.13	12.34	5.06	0
7	0	1.49	9.07	13.44	14.77	11.37	4.41	0
7	0	1.12	5.45	11.74	10.85	3.88	0.61	0
7	0	0.1	3.11	12.36	5.87	0.18	0	0
7	0	0.12	6.39	11.66	2.21	0.01	0	0

8:

8	0	0.14	5.02	11.6	12.4	6.24	0.53	0
8	0.03	1.98	12.29	11.47	9.34	11.64	2.52	0
8	0.01	2.95	11.58	7.44	7.93	11.68	2.15	0
8	0	1.18	8.55	13.22	13.32	7.02	0.51	0
8	0	0.44	6.98	14.03	12.92	4.35	0.1	0
8	0	1.16	10.68	8.52	9.16	8.48	1.29	0
8	0	0.99	10.97	8.09	8.39	9.54	2.32	0.01
8	0	0.14	5.01	12.7	13.01	6.74	1.21	0.01

9:

9	0	0.14	5.68	11.83	11.26	5.94	1.57	0.12
9	0	2.43	12.67	9.59	10.13	11.36	2.61	0.12
9	0	3.66	12.5	5.65	8.32	14.11	3.21	0.04
9	0	1.96	10.42	12.19	13.23	14.06	3.84	0
9	0	0.17	2.94	5.17	5.09	11.64	4.83	0
9	0	0.16	0.52	0.61	2.46	9.74	5.85	0.02
9	0	0.66	6.12	4.96	5.81	10.53	5.19	0.11
9	0	0.09	5.73	12.04	13.14	8.89	2.09	0.06

Clasificación de dígitos utilizando K-Nearest Neighbors (KNN):

Este código implementa un proceso para clasificar imágenes de dígitos manuscritos

mediante el algoritmo K-Nearest Neighbors (KNN), utilizando el conjunto de datos digits de scikit-learn. A continuación, se detallan los pasos realizados y cómo cumplen con los requisitos solicitados:

Carga del conjunto de datos "digits":

El conjunto de datos digits de scikit-learn es cargado utilizando sklearn.datasets.load_digits(). Este conjunto contiene 1797 imágenes de dígitos del 0 al 9, cada una de 8x8 píxeles, con sus correspondientes etiquetas en el campo target.

Selección y carga de la imagen del dígito:

El código permite al usuario seleccionar una imagen del directorio datasets, el cual contiene archivos PNG creados en Paint representando dígitos manuscritos del 0 al 9. El usuario selecciona la imagen por su índice en el directorio mediante un input.

Preprocesamiento de la imagen seleccionada:

La imagen seleccionada es leída utilizando OpenCV (cv2.imread()) y convertida a escala de grises.

Se redimensiona la imagen a 8x8 píxeles utilizando cv2.resize() y se aplana en un vector unidimensional con .flatten().

La imagen es normalizada para que sus valores estén entre 0 y 16, a fin de hacerlos comparables con las imágenes del conjunto de datos original.

Clasificación utilizando K-Nearest Neighbors (KNN):

El modelo KNN es entrenado utilizando los datos del conjunto de entrenamiento digits.data y las etiquetas digits.target.

El número de vecinos n es 3.

El modelo busca en digits.data los n vecinos más cercanos al dígito ingresado osea calcula las distancias del dígito a sus vecinos y selecciona los tres vecinos con las distancias más pequeñas. Los resultados se almacenan en una lista llamada neighbours que como es una lista de una lista necesita llamarse como neighbours[0] para obtener los índices en donde se encuentran los vecinos del dígito.

digits["target"][i] usa esos índices para obtener las etiquetas (dígitos) correspondientes a esos vecinos y las almacena en la lista targets.

Después se imprime la lista target para mostrar a esos tres vecinos.

Clasificación por vecinos:

Carga del conjunto de datos "digits":

El conjunto de datos digits de scikit-learn es cargado utilizando sklearn.datasets.load_digits(). Este conjunto contiene 1797 imágenes de dígitos del 0 al 9, cada una de 8x8 píxeles, con sus correspondientes etiquetas en el campo target.

Selección y carga de la imagen del dígito:

El código permite al usuario seleccionar una imagen del directorio datasets (que debe contener archivos de imágenes de dígitos numerados del 0 al 9). El usuario selecciona la imagen por su índice en el directorio mediante un input.

Preprocesamiento de la imagen seleccionada:

La imagen seleccionada es leída utilizando OpenCV (cv2.imread()) y convertida a escala de grises

Se redimensiona la imagen a 8x8 píxeles utilizando cv2.resize() y se aplana en un vector unidimensional con .flatten().

La imagen es normalizada para que sus valores estén entre 0 y 16, a fin de hacerlos comparables con las imágenes del conjunto de datos original.

Clasificación utilizando K-Nearest Neighbors (KNN):

El modelo KNN es entrenado utilizando los datos del conjunto de entrenamiento digits.data y las etiquetas digits.target.

El número de vecinos n es inicialmente establecido en 3, pero el código permite incrementar este número si el usuario lo desea.

El modelo busca los n vecinos más cercanos al dígito ingresado y calcula las distancias a esos vecinos. Los resultados se almacenan en el diccionario dc, que

cuenta cuántas veces cada etiqueta (dígito) aparece entre los vecinos más cercanos.

Decisión sobre el número de vecinos:

Si todos los vecinos son distintos, el sistema pide al usuario si desea incrementar el número de vecinos para mejorar la clasificación. Si el usuario elige "Y", se incrementa el número de vecinos y se repite el proceso. Si el usuario elige "N", el programa detiene la búsqueda.

El código selecciona

Predicción y visualización de resultados:

print(f"Soy la inteligencia artificial, y he detectado que el dígito ingresado corresponde al número {predicted}.")

print("Vecinos más cercanos:", neighbours[0])

print("Distancias: ",distances)

El dígito predicho es el dígito (vecino) con más moda en la lista targets.

Se imprimen los vecinos más cercanos y las distancias de estos vecinos.

Generación de un DataFrame con las imágenes de los vecinos más cercanos:

Se crea un DataFrame final_df que contiene las imágenes de los vecinos más cercanos, donde cada fila es una imagen de 8x8 píxeles representada como una serie de valores en un DataFrame.

Se añaden las etiquetas de los dígitos correspondientes a cada vecino y se inserta un separador vacío entre las filas de los vecinos para mejorar la legibilidad.

Exportación a Excel:

El DataFrame final que contiene las imágenes de los vecinos más cercanos se exporta a un archivo Excel denominado "nearest_neightbours.xlsx", utilizando el método to_excel() de pandas.

Clasificación de dígitos utilizando K-Nearest Neighbors (KNN) con Promedios:

Este código realiza un proceso para clasificar una imagen de un dígito utilizando el algoritmo K-Nearest Neighbors (KNN), basándose en el cálculo de promedios de imágenes de cada dígito. A continuación, se detallan los pasos realizados y cómo cumplen con los requisitos solicitados:

Carga del conjunto de datos "digits":

El conjunto de datos digits de scikit-learn se carga usando sklearn.datasets.load_digits(). Este conjunto de datos contiene imágenes de dígitos (0-9) de 8x8 píxeles y sus respectivas etiquetas.

Cálculo de los promedios:

Se llama a la función calcular_promedio() del módulo sacar_promedios, que calcula los promedios de las imágenes para cada uno de los 10 dígitos. Esta función organiza las imágenes del conjunto de datos por su etiqueta y calcula el promedio de cada conjunto de imágenes correspondientes a cada dígito, almacenando los promedios en el diccionario avg.

Selección de la imagen a clasificar:

El código solicita al usuario que seleccione una imagen del directorio datasets. El usuario debe ingresar un número entre 0 y el número total de imágenes disponibles en el directorio, y el programa carga la imagen seleccionada como un archivo .png.

Preprocesamiento de la imagen seleccionada:

La imagen seleccionada se lee utilizando OpenCV (cv2.imread()) y se convierte a escala de grises.

Luego, la imagen se redimensiona a 8x8 píxeles utilizando cv2.resize(), se aplana en un vector unidimensional utilizando .flatten() y se normaliza para que los valores estén entre 0 y 16 (en comparación con las imágenes originales del conjunto de datos).

Clasificación utilizando K-Nearest Neighbors (KNN):

Se utiliza el modelo KNN de scikit-learn (KNeighborsClassifier) para clasificar la imagen seleccionada. El modelo es entrenado utilizando los promedios calculados (avg.values()) como características y las etiquetas correspondientes (avg.keys()).

Se realiza la predicción de la clase utilizando el método knn.predict(), y el dígito predicho se imprime en la consola, esta predicción es el resultado de calcular la distancia entre el dígito y los promedios de los otros 10 dígitos, luego se seleccionó la menor distancia entre puntos.

Obtención de vecinos más cercanos:

Después de la predicción, el código obtiene los vecinos más cercanos al dígito ingresado utilizando el método knn.kneighbors(). Las distancias y los índices de los vecinos más cercanos se imprimen para ayudar a visualizar cómo el modelo llegó a la predicción.

Visualización de la imagen de los vecinos más cercanos:

El código imprime la imagen correspondiente al vecino más cercano en el conjunto de datos, lo que permite ver una comparación visual de la imagen seleccionada con la imagen más parecida en el conjunto de entrenamiento.

Comparaciòn de métodos:

Para determinar cuál de los dos métodos es mejor en el contexto de clasificación automática de dígitos, se deben realizar experimentos comparativos utilizando ambos métodos. Por lo tanto, en la siguiente tabla podrá observarse el rendimiento de ambos métodos al intentar identificar el mismo dígito a partir de la misma imagen. Cabe recalcar que la misma imagen ha sido procesada dos veces por cada método.

Tabla:

	KNN (K-Nearest Neighbors)		KNN_Promedios	
n0	✔	✔	✔	✔
n1	✔	✔	✔	✔
n2	✔	✔	✘	✘
n3	✘	✘	✘	✘
n4	✔	✔	✔	✔
n5	✔	✔	✘	✘
n6	✘	✘	✘	✘
n7	✔	✔	✔	✔
n8	✔	✔	✘	✘
n9	✘	✘	✘	✘
n10	✘	✘	✘	✘
n11	✘	✔	✘	✘
n12	✔	✔	✘	✘
n13	✔	✔	✔	✔
n14	✘	✘	✔	✔
n15	✔	✔	✔	✔
n16	✔	✔	✔	✔
n17	✔	✘	✘	✘
n18	✔	✔	✔	✔
n19	✘	✘	✔	✔
n20	✔	✔	✔	✔
n21	✘	✘	✘	✘
n22	✘	✘	✘	✘
n23	✔	✔	✔	✔
n24	✔	✔	✘	✘

Tabla con los intentos y asertividad del número real:

KNN (K-Nearest Neighbors)			
		Primera	
		si	no
Segunda	si	15	1
	no	1	8

En base a la tabla anterior podemos decir que del 100% de los casos solo el 60% de los dígitos tipeados que se ingresaron fueron reconocidos correctamente en los dos intentos por el modelo KNN basado en la moda de los tres vecinos más cercanos ,

el 8% fue reconocido de forma inestable, pues en un caso en el primer intento el dígito fue reconocido correctamente y en el segundo intento incorrectamente y en otro caso ello fue al revés. Por otro lado, el 32% de los dígitos ingresados no fueron reconocidos por el modelo correctamente en los dos intentos.

KNN_Promedios			
		Primera	
		si	no
Segunda	si	12	0
	no	0	13

En base a la tabla anterior podemos decir que del 100% de los casos solo el 48% de los dígitos tipeados que se ingresaron fueron reconocidos correctamente en los dos intentos por el modelo KNN basado en los promedios de los 10 dígitos del dataset digits , no hubo algún caso en el que se reconociera de forma inestable el dígito, en otras palabras, en ningún proceso de reconocimiento se obtuvo que en el primer intento el dígito fue reconocido correctamente y en el segundo intento incorrectamente o al revés. Por otro lado, el 52% de los dígitos ingresados no fueron reconocidos por el modelo correctamente en los dos intentos.

En base al análisis realizado anteriormente se puede concluir que en eficiencia el primer modelo es mejor, ya que del total de 25 casos el 60% reconoció correctamente, mientras que el segundo modelo sólo reconoció correctamente el 48%. Por otro lado si tomamos en cuenta los casos particulares, podemos decir para los dígitos 2(archivo n2), 5(archivo n5), 8(archivo n8), 1(archivo n11), 4(archivo n12), 3(archivo n17) y 0(archivo n24) el primer modelo es mejor porque si acertó el número o al menos en uno de los intentos lo hizo, mientras que el segundo modelo no acertó al procesar esos dígitos. Sin embargo para los dígitos 6(archivo n14) y 9(archivo n19) el segundo modelo es mejor porque si acertó el número en ambos intentos, mientras que el primer modelo no acertó al procesar esos dígitos.

Aunque sabemos que el primer modelo es mejor, hay algo que aùn necesita ser comparado que la consistencia de los resultados entre ambos métodos de clasificación —KNN (K-Nearest Neighbors) y KNN_Promedios— podemos determinar cuál es la más consistente en sus respuestas en base a la matriz de confusión y las métricas de evaluación (Accuracy, Precision, Recall y F1 Score):

Modelo 1:

ACCURACY	0.92
PRECISION	0.9375
RECALL	0.9375
F1 SCORE	0.9375

Accuracy (Exactitud):

El modelo alcanzó una exactitud del 92%, lo que significa que acertó en el 92% de los casos al predecir correctamente tanto los positivos como los negativos. Sin embargo, cometió dos errores: un falso positivo (predijo "bien” en el primer intento y "mal" en el segundo) y un falso negativo (predijo "mal" en el primer intento y “bien” en el segundo).

Precision (Precisión):

La precisión de este modelo fue de 93.75%. Esto quiere decir que, de todas las veces que predijo un resultado positivo ("sí"), el 93.75% de las veces fue en ambos casos. Hubo un caso en el que predijo "sí", pero después "no", lo que representa un falso positivo.

Recall (Sensibilidad o Exhaustividad):

El recall también fue de 93.75%, lo que indica que el modelo logró identificar el 93.75% de los casos consistentemente.

F1 Score:

El F1 Score fue de 93.75%, lo cual representa un equilibrio sólido entre la precisión y el recall. Aunque no es perfecto, indica que el modelo tiene un rendimiento bastante confiable tanto para detectar los positivos como para evitar falsos positivos.

Modelo 2:

ACCURACY	1
PRECISION	1

RECALL 1

F1 SCORE 1

Accuracy (Exactitud):

Este modelo logró una exactitud perfecta del 100%. Esto significa que todas las predicciones que hizo fueron consistentes en respuesta. Cada caso positivo y negativo fue identificado de forma correcta.

Precision (Precisión):

La precisión fue del 100%, lo que implica que todas las veces que el modelo predijo un caso fue consistente con sus respuestas de inicio a fin.

Recall (Sensibilidad o Exhaustividad):

El recall también fue del 100%, lo que indica que el modelo detectó correctamente todos los casos que realmente eran positivos osea que desde el primer momento detectó de forma continua la misma respuesta.

F1 Score:

El F1 Score alcanzó el valor perfecto de 100%, lo cual refleja un equilibrio completo entre precisión y recall. Este resultado muestra que el modelo tiene un rendimiento ideal en respuestas, sin varianza de clasificación.

El modelo KNN_Promedios es superior en consistencia de respuestas al modelo 1.

Proyecto en Git-Hub:https://github.com/leonardohq-8519/icc-proyecto-2.git

Conclusiones:

Prompt para la Inteligencia Artificial:

Enviarle los documentos.py, los documentos.csv, imagen.png y documentos.xlsx.

Analiza el codigo del documentos.py adjuntos y los archivos .csv, y en base a ello brindame una conclusión general del objetivo que tenía cada código de cada archivo, su eficiencia para hacer la clasificaciòn de las imágenes y complementalo con una conclusión crítica en base a tus conocimientos sobre investigación en computación.

Conclusión (Leonardo Huanay):

En este proyecto se solicitó clasificar imágenes de dígitos, usando load_digits como conjunto de entrenamiento. Las imágenes fueron pasadas como matrices unidimensionales mediante la librería opencv para luego ser analizadas.

Para ello se usaron dos enfoques distintos. El primero usó todos los datos con la finalidad de realizar K-Nearest Neighbours a partir de este, se calcularon las distancias euclidianas a los 3 vecinos más cercanos y se designó una etiqueta en específico a la nueva imagen recibida por el código de acuerdo al valor más frecuente entre los encontrados por KNN. En el segundo enfoque se prefirió calcular una matriz promedio para cada una de las etiquetas, permitiendo un análisis de los datos más rápido, tomando únicamente de base para la clasificación del nuevo dato la menor distancia euclidiana con respecto a las 10 matrices generadas.

A diferencia de lo esperado en un inicio el modelo que terminó teniendo un menor rendimiento fue aquel que trabajaba mediante los promedios. Esto se debe a que al generar la media para cada uno de los 64 valores de las 10 etiquetas (del 0 al 9) se genera una "imagen borrosa" de las diferentes versiones de cada número, ocasionando para ciertos valores manchas que perjudican la clasificación de los nuevos dígitos. En conclusión, el archivo K_NN.py, pese a ser más lento, terminó clasificando de forma exitosa los dígitos en una mayor cantidad de ocasiones al manejar la variedad de estilos entre los datos efectivamente gracias a estar entrenado en base a todos los ejemplos y no imponer un prototipo único a cada etiqueta.

Conclusión IA - Gemini (Leonardo Huanay):

Conclusión General del Proyecto

El proyecto explora y demuestra con éxito dos métodos de clasificación basados en K-NN para el reconocimiento de dígitos. La primera aproximación (K_NN.py) es robusta pero computacionalmente más intensiva, ya que compara la entrada con cientos de ejemplos reales. La segunda (K_NN_Promedios.py) es extremadamente rápida y eficiente, al reducir el problema a una simple comparación con 10 prototipos.

La inclusión de scripts para visualizar los datos (promedios) y mejorar la presentación de los resultados (formato_condicional.py) demuestra una

comprensión completa del flujo de trabajo, desde el modelado hasta la interpretación y comunicación de los resultados.

Conclusión (Joseph Geraldo):

Este proyecto solicitó clasificar imagenes de números dibujados a mano mediante computer vision. Para ello se utilizó el load_digits y el algoritmo de KNN con 2 enfoques distintos:

Para el primero, se emplearon todos los datos para calcular luego las 3 distancias euclidianas más cercanas al nuevo dato introducido, asignando una etiqueta en base al valor más frecuente de los vecinos.

Para el segundo, se generó una matriz promedio para cada número, para así poder comparar de manera más “segura”, debido a que se disminuye la distancia euclidiana.

De manera contraria a la que se esperaba, el segundo enfoque resultó menos eficiente que el primero, debido a que esta matriz promedio resultaba en una “imagen confusa” por decirlo de algún modo. Esto concluye que, aunque el primer enfoque demore más tiempo, resulta más eficiente.

Conclusión IA - Deepseek (Joseph Geraldo):

Conclusión del Proyecto:

Este proyecto demostró la aplicación de dos enfoques basados en el algoritmo *K-Nearest Neighbors (KNN)* para clasificar dígitos manuscritos del conjunto *load_digits* de scikit-learn. El primer método utilizó todos los datos de entrenamiento para calcular distancias euclidianas y asignar etiquetas según los vecinos más cercanos, mientras que el segundo empleó matrices promedio de cada dígito para agilizar el proceso.

Resultados clave:

1. **Efectividad:** El enfoque tradicional de KNN, aunque más lento, mostró mayor precisión al considerar la diversidad de estilos en los datos originales.
2. **Limitaciones del método de promedios:** La generación de imágenes promedio

introdujo distorsiones ("borrosidad"), reduciendo la capacidad de clasificación en casos con variaciones significativas.

3. **Herramientas utilizadas:** Se aprovecharon librerías como OpenCV (cv2) para el procesamiento de imágenes, NumPy para cálculos matriciales, y pandas para la organización y exportación de resultados.

Conclusiones finales:

- **Robustez vs. velocidad:** El KNN estándar fue más preciso pero computacionalmente costoso, mientras que el método de promedios ofreció rapidez a expensas de precisión.
- **Recomendaciones:** Para mejorar el proyecto, se sugiere implementar una evaluación sistemática con métricas (accuracy, F1-score), optimizar el código para evitar redundancias, y explorar técnicas como *GridSearchCV* para ajustar hiperparámetros automáticamente.

En resumen, el proyecto validó la importancia de elegir estrategias según el contexto: priorizar precisión cuando sea crítica (KNN tradicional) o eficiencia en entornos con restricciones de tiempo (método de promedios). Además, destacó el valor de herramientas de visualización y preprocesamiento para interpretar resultados y mejorar modelos.

Conclusiones por Módulo

K_NN.py: Implementa un clasificador K-NN estándar con una lógica interesante para manejar la ambigüedad en las predicciones, haciéndolo interactivo.

K_NN_Promedios.py: Muestra una ingeniosa optimización del problema, transformándolo en una tarea de "template matching" o coincidencia de plantillas. Es una excelente demostración de cómo la ingeniería de características (crear los promedios) puede simplificar un modelo.

sacar_promedios.py y ver_promedios.py: Son módulos de apoyo fundamentales que encapsulan la lógica para crear y visualizar los "prototipos" de dígitos, que son el corazón del segundo enfoque.

formato_condicional.py: Es una utilidad valiosa que mejora la interpretabilidad de los

resultados, un paso a menudo olvidado en proyectos de ciencia de datos.

Oportunidades de Mejora

Evaluación Sistemática del Modelo: El proyecto actual clasifica imágenes individuales proporcionadas por el usuario. Para una evaluación rigurosa, se debería dividir el dataset digits en un conjunto de entrenamiento y uno de prueba (usando sklearn.model_selection.train_test_split). Luego, se podrían calcular métricas de rendimiento (como accuracy, precision, recall, F1-score) para ambos métodos y compararlos objetivamente.

Optimización del Código:

En K_NN.py, el knn.fit(data, labels) se ejecuta dentro del bucle while. Esto es ineficiente, ya que el modelo se "re-entrena" con los mismos datos en cada iteración. El fit debería realizarse una sola vez, antes de que comience el bucle.

El código de carga y preprocesamiento de imágenes está duplicado en K_NN.py y K_NN_Promedios.py. Podría refactorizarse en una única función para evitar la redundancia.

Búsqueda de Hiperparámetros: En K_NN.py, la elección de k es manual e interactiva. Se podría utilizar una técnica como GridSearchCV de Scikit-learn para encontrar automáticamente el valor óptimo de k que maximice el rendimiento del modelo en un conjunto de validación.

Robustez y Experiencia de Usuario:

El código asume que las imágenes de entrada están en una carpeta datasets y siguen un patrón de nomenclatura específico (n1.png, n2.png). Esto podría mejorarse permitiendo al usuario especificar una ruta de archivo completa o usando un selector de archivos gráfico.

El manejo de errores podría ser más específico (por ejemplo, diferenciar entre un ValueError si la entrada no es un número y un IndexError si el número está fuera de rango).

Conclusión (Joaquín Llallire):

Trabajar en este proyecto fue una experiencia bastante enriquecedora. Más allá de solo programar o seguir instrucciones, realmente pudimos entender cómo se aplican conceptos de inteligencia artificial y visión por computadora a problemas reales,

como reconocer un número a partir de una imagen.

Pudimos ver que hay distintas formas de enfrentar un mismo problema. Por ejemplo, un enfoque comparaba cada nueva imagen con todas las del dataset, y otro lo hacía solamente con promedios. El primero fue más preciso, pero tomaba más tiempo. El segundo fue más rápido, pero no tan exacto. Esa comparación nos hizo pensar que a veces no hay un único "mejor" modelo, sino que depende de qué es más importante: rapidez o precisión.

También aprendimos a usar herramientas que antes solo conocíamos de nombre, como OpenCV o Pandas, y logramos exportar nuestros resultados a Excel, aplicar formatos visuales y hasta visualizar los vecinos más cercanos a una imagen. Fue bastante satisfactorio ver cómo todo eso funcionaba junto.

En resumen, este proyecto nos sirvió no solo para aplicar lo aprendido, sino también para pensar como desarrolladores y analistas: buscando soluciones, comparando métodos y entendiendo los resultados más allá del código.

Conclusión (IA-Joaquín Llallire):

El proyecto demostró con éxito la aplicación de algoritmos de aprendizaje automático para el reconocimiento de dígitos escritos a mano utilizando el dataset load digits de la librería scikit-learn. Se exploraron dos enfoques basados en el algoritmo K-Nearest Neighbors (KNN): uno utilizando el conjunto completo de datos como referencia de entrenamiento y otro basado en la comparación contra promedios previamente calculados de cada dígito.

El primer enfoque, aunque computacionalmente más costoso, demostró una mayor capacidad de adaptación frente a la variabilidad de las imágenes, debido a que trabaja con la diversidad completa del dataset. El segundo enfoque, basado en promedios, ofreció un procesamiento más rápido y consistente, pero con menor precisión, al simplificar la información de cada clase a una sola representación promedio.

Los resultados mostraron que el enfoque tradicional con múltiples vecinos tiene una mejor tasa de acierto, pero el enfoque por promedios presenta una estabilidad destacable en sus predicciones. Ambos métodos se beneficiaron del uso de herramientas como NumPy, Pandas, OpenCV y openpyxl, las cuales permitieron manipular y visualizar los datos de forma eficiente.

En términos de rendimiento, la combinación de visualización, procesamiento de imágenes y evaluación cuantitativa aportó una comprensión clara sobre el comportamiento de cada modelo. Esta experiencia refleja un caso práctico en el que se equilibran velocidad y precisión según las necesidades del problema.

Conclusiòn Maydelith:

En conclusión, durante el presente informe se implementan dos distintos métodos de clasificación que si bien son fieles a la dataset digits del paquete scikit-learn, no clasifican de la misma manera los dígitos ingresados. El presente análisis del código muestra cómo puede ser màs eficiente el uso de seleccionar la moda de tres o màs vecinos cercanos al dígito escrito que identifica el dígito en base a la menor distancia euclidiana entre el dígito ingresado y los 10 promedios de los dataset digits. Cabe resaltar que aunque el primer método tiene 60% de casos acertados y el segundo método 48%, se ha mostrado en el informe que el segundo método es màs preciso y conciso en respuesta, pues en ambos intentos por caso sale el mismo resultado sea que lo reconozca o no, mientras que el primer método tiene dos casos en los que su respuesta es inconsistente. En general para la clasificación de dígitos manuscritos se recomienda usar el primer método, aunque se debe entrenar con màs casos para que tenga mayor consistencia en sus respuestas.

Conclusiòn Chat Gpt(Maydelith):

Este proyecto representa una aplicación efectiva de conceptos de aprendizaje supervisado y procesamiento de imágenes utilizando herramientas accesibles como scikit-learn, OpenCV y pandas. La implementación de dos enfoques distintos del algoritmo KNN permitió explorar no solo las capacidades del algoritmo en su forma tradicional, sino también cómo se puede optimizar su funcionamiento mediante el

uso de representaciones estadísticas como los promedios de clase.

El método tradicional de KNN, al basarse en todos los ejemplos disponibles del dataset, ofrece una mayor fidelidad en la predicción de los dígitos, ya que tiene la capacidad de adaptarse mejor a la variabilidad presente en los estilos de escritura. Este enfoque resulta especialmente útil en contextos donde la precisión es crítica, a pesar del mayor costo computacional.

Por otro lado, el método de KNN con promedios introduce una reducción significativa en la complejidad del modelo al convertir el problema en una tarea de comparación contra prototipos fijos. Esta estrategia sacrifica flexibilidad y capacidad de adaptación a cambio de velocidad y simplicidad, lo que puede ser valioso en sistemas embebidos, móviles o de recursos limitados.

Además del desarrollo algorítmico, se evidencia un buen manejo del pipeline de datos: desde la carga, visualización y transformación de datos hasta la exportación de resultados con formato condicional para facilitar su interpretación. Esto demuestra una comprensión sólida no solo de los algoritmos, sino también de la importancia del análisis y comunicación de resultados en proyectos de ciencia de datos.

Finalmente, este trabajo deja en evidencia que la elección del modelo no debe centrarse únicamente en la precisión, sino también considerar aspectos como la consistencia, la eficiencia computacional, la interpretabilidad del modelo y el contexto en el que se aplicará. El balance entre estos factores es lo que determina el verdadero valor de una solución de machine learning.

Recursos usados por las conclusiones generadas por Inteligencia Artificial que el grupo omitió:

1. Evaluación sistemática mediante métricas cuantitativas: Las IAs mencionan y recomiendan explícitamente el uso de métricas como accuracy, precision, recall y F1-score. Además, destacan su importancia para una comparación

objetiva entre métodos.

2. Justificación técnica del diseño de los algoritmos: Las IAs justifican por qué el modelo de KNN tradicional ofrece más fidelidad: al considerar todos los estilos de escritura. También explican que el modelo de promedios sacrifica precisión por eficiencia.
3. Detalle del preprocesamiento de datos: Las conclusiones IA explican el proceso de transformación: escalado, normalización, redimensionamiento a 8x8, flattening, etc. También destacan cómo estos pasos afectan directamente la calidad del modelo.
4. Crítica a la implementación técnica y propuesta de mejora: Las IAs identifican ineficiencias de código, como el fit() dentro del bucle o duplicación de funciones. Proponen mejoras reales: modularización, uso de GridSearchCV para elegir el valor óptimo de K, mejor manejo de errores y experiencia de usuario.
5. Análisis de consistencia y estabilidad del modelo: Las IAs identifican la consistencia de los resultados entre intentos (estabilidad del modelo). Incluso mencionan cómo un modelo puede tener mejor precisión, pero peor estabilidad, y recomiendan balancear ambos criterios.
6. Recomendaciones científicas concretas: Las IAs no solo comparan enfoques, sino que recomiendan integrar:
    - Técnicas alternativas como DBSCAN o GMM.
    - Embeddings más potentes que TF-IDF (como BERT, en contexto ampliado).
    - Refinamiento del proceso usando validación cruzada, control de hiperparámetros, y análisis interpretativo.

Fuentes bibliográficas:

INESEM. (s. f.). *OpenCV – Qué es, usos y aplicaciones en visión artificial*. In INESEM revista digital. Recuperado el 3 de julio de 2025, de https://www.inesem.es/revistadigital/informatica-y-tics/opencv/

IBM. (s. f.). ¿Qué es el algoritmo de k vecinos más cercanos (KNN)? *Think – IBM*. Recuperado el 3 de julio de 2025, de

https://www.ibm.com/mx-es/think/topics/knn#:~:text=del%20algoritmo%20KNN-,%C2%BFQu%C3%A9%20es%20el%20algoritmo%20KNN?,un%20punto%20de%20datos%20individual

NumPy Developers. (s. f.). *What Is NumPy?* En NumPy User Guide. Recuperado el 3 de julio de 2025, de https://numpy.org/devdocs/user/whatisnumpy.html

Openpyxl maintainers. (2024, 28 de junio). *openpyxl* (versión 3.1.5) [Paquete Python]. PyPI. https://pypi.org/project/openpyxl/

scikit-learn. (s. f.). *load_digits – Carga y devuelve el conjunto de datos de dígitos (clasificación)*. Documentación de sklearn trad. al español. Recuperado el 3 de julio 2025 qu4nt.github.io/sklearn-doc-es/modules/generated/sklearn.datasets.load_digits.html

UOC Data Science. (s. f.). *Preprocesamiento de datos con scikit-learn*. Espacio de recursos de ciencia de datos UOC. Recuperado el 3 de julio de 2025, de https://datascience.recursos.uoc.edu/es/preprocesamiento-de-datos-con-sklearn/