Part 01: Image Recognition using KNN

Lab: Image Recognition with KNN
Objective:
In this lab, the goal is to understand and apply the K-Nearest Neighbors (KNN) model for classifying images. We will use the MNIST dataset, a collection of handwritten digit images, to train and test our model. We will also examine the impact of added noise to the images on the model's performance.

Dataset: MNIST
The MNIST dataset (Modified National Institute of Standards and Technology) contains 70,000 images of handwritten digits (0 to 9), divided as follows:

- 60,000 images for training.
- 10,000 images for testing.

The images are in grayscale and have a size of 28x28 pixels. Each image represents a handwritten digit, and the objective is to predict the digit it represents.

The dataset is organized into two parts:

1. **Input Data (X):** The pixels of the images, in the form of vectors with a dimension of 784 (28x28).
2. **Labels (Y):** The labels corresponding to each image, representing the handwritten digits from 0 to 9.

In [1]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml("mnist_784", version=1)
```

```
C:\Users\elazh\anaconda3\lib\site-packages\sklearn\datasets\_openml.py:93
2: FutureWarning: The default value of `parser` will change from `'liac-ar
ff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence this warni
ng. Therefore, an `ImportError` will be raised from 1.4 if the dataset is
dense and pandas is not installed. Note that the pandas parser may return
different data types. See the Notes Section in fetch_openml's API doc for
details.
  warn(
```

In [2]:
```
x = mnist["data"]
```

In [3]:
```
x.head()
```

Out[3]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775
0	0.0	0.0	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	0.0	...	0.0
2	0.0	0.0	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	0.0	...	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0

5 rows × 784 columns

In [4]:
```
y = mnist["target"]
```

In [5]:
```
y.head()
```

Out[5]:
```
0    5
1    0
2    4
3    1
4    9
Name: class, dtype: category
Categories (10, object): ['0', '1', '2', '3', ..., '6', '7', '8', '9']
```

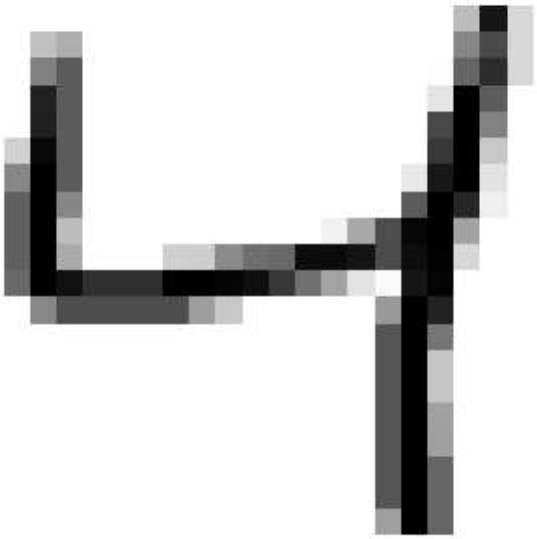
In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [7]:
```
examp = x.iloc[2]
examp
```

Out[7]:
```
pixel1      0.0
pixel2      0.0
pixel3      0.0
pixel4      0.0
pixel5      0.0
           ...
pixel780    0.0
pixel781    0.0
pixel782    0.0
pixel783    0.0
pixel784    0.0
Name: 2, Length: 784, dtype: float64
```

In [8]:
```
examp_image= np.reshape(np.ravel(examp), (28, 28))
```

In [9]:
```
plt.imshow(examp_image,cmap="binary")
plt.axis("off")
plt.show()
```



In [10]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

In [11]:
```
x_train, x_test, y_train, y_test = x[:60000], x[60000:], y[:60000], y[60000
```

In [12]:
```
knn = KNeighborsClassifier(n_neighbors=3)
```

In [13]:
```
knn.fit(x_train, y_train)
```

Out[13]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [14]:
```
y_pred = knn.predict(x_test)
y_pred
```

Out[14]:
```
array(['7', '2', '1', ..., '4', '5', '6'], dtype=object)
```

In [15]:
```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

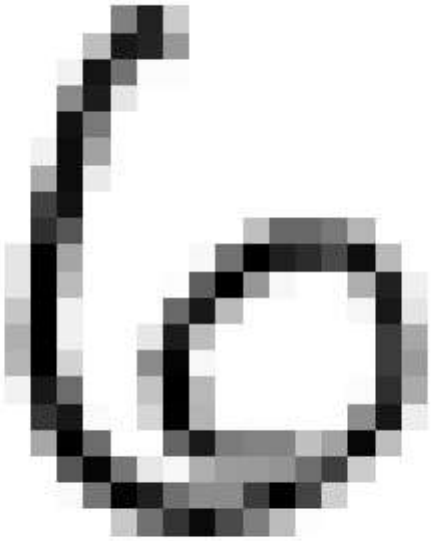
Out[15]:
```
0.9705
```

In [16]:
```
examp2 = x_test.iloc[100]
examp2
```

Out[16]:
```
pixel1      0.0
pixel2      0.0
pixel3      0.0
pixel4      0.0
pixel5      0.0
            ...
pixel780    0.0
pixel781    0.0
pixel782    0.0
pixel783    0.0
pixel784    0.0
Name: 60100, Length: 784, dtype: float64
```

In [17]:
```
examp2_image= np.reshape(np.ravel(examp2), (28, 28))
```

In [18]:
```
plt.imshow(examp2_image,cmap="binary")
plt.axis("off")
plt.show()
```



In [19]:
```
y_pred[100]
```

Out[19]:
```
'6'
```

In [20]:
```
from sklearn.metrics import accuracy_score, classification_report, confusic
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9705

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.96      1.00      0.98      1135
           2       0.98      0.97      0.97      1032
           3       0.96      0.97      0.96      1010
           4       0.98      0.97      0.97       982
           5       0.97      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.99      0.94      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000


Confusion Matrix:
 [[ 974    1    1    0    0    1    2    1    0    0]
 [   0 1133    2    0    0    0    0    0    0    0]
 [  10    9  996    2    0    0    0   13    2    0]
 [   0    2    4  976    1   13    1    7    3    3]
 [   1    6    0    0  950    0    4    2    0   19]
 [   6    1    0   11    2  859    5    1    3    4]
 [   5    3    0    0    3    3  944    0    0    0]
 [   0   21    5    0    1    0    0  991    0   10]
 [   8    2    4   16    8   11    3    4  914    4]
 [   4    5    2    8    9    2    1    8    2  968]]
```

Part 02: Image Enhancement using KNN

In this part of the lab, we are exploring the potential of the K-Nearest Neighbors (KNN) algorithm to enhance noisy images. The task is to introduce noise into the MNIST dataset, train a KNN model on the noisy images, and evaluate whether KNN can effectively remove the noise. The following steps are involved:

Add random noise to the MNIST images. Train the KNN model using the noisy images as input and the clean images (original MNIST data) as the target output. Test the model on new noisy images to assess if it can "denoise" them and restore the original features. Compare the model's performance by evaluating how well it predicts the clean version of the noisy images. Your goal is to implement this task and observe how well KNN performs in terms of denoising. Does the model learn to remove noise, and how effective is it?

Please complete the lab and send your final code along with any observations or results to me via email at dijaguelmim@gmail.com (mailto:dijaguelmim@gmail.com). The object

In [ ]: