Nearest neighbors

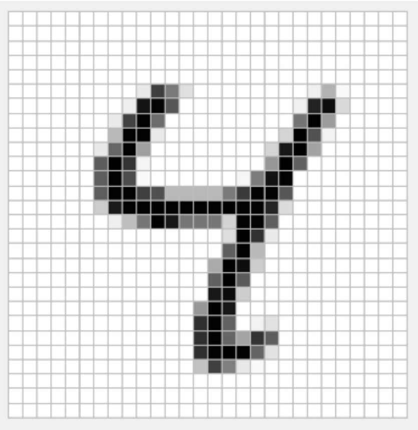
COMS 4771 Fall 2025

Digit recognition

Problem: Create a program that, given an image of a handwritten digit as input, returns the digit depicted in the image

Simplifying assumptions:

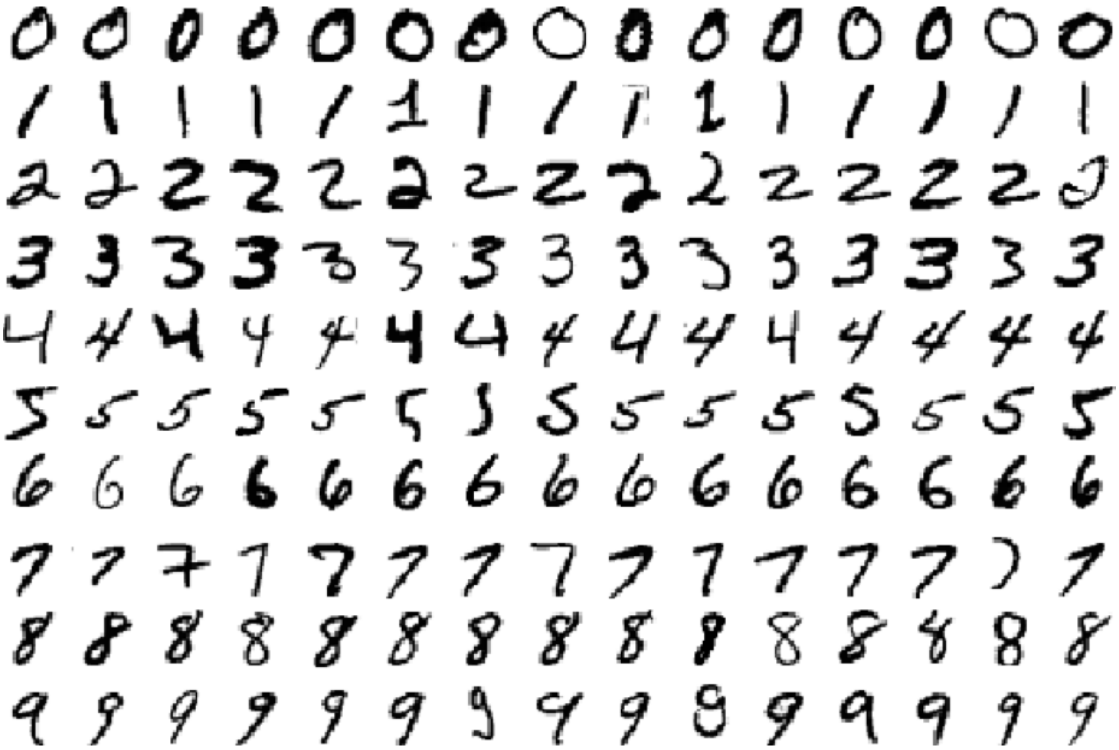
- The image depicts some digit (from $\{0, 1, \ldots, 9\}$)
- The depicted digit is (roughly) in the center of the image
- The image is a 28×28 pixel image (for a total of 784 pixels)
- Each pixel is grayscale; pixel intensity is an integer from $\{0, 1, \ldots, 255\}$



Machine learning approach to digit recognition:

- Don't explicitly write the image classifier by hand
- Collect a labeled dataset of images
  - Each image is an example of how someone might write a digit
  - Each image is annotated with a label—the digit depicted in the image
  - NIST has collected such a dataset with 60000 examples ("MNIST")[1]
- Provide the labeled dataset as input to a learning algorithm
- Learning algorithm returns an image classifier

[1] http://yann.lecun.com/exdb/mnist/



Nearest neighbors learning algorithm

Nearest Neighbors (NN) learning algorithm:

- Input: Labeled dataset $\mathcal{S}$
- Output: NN classifier for labeled dataset $\mathcal{S}$ (also a program!)

Notation:

- n: number of images in the dataset
- $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$: the n images
- $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$: the n corresponding labels
- Labeled dataset

$$\mathcal{S} = ((x^{(i)}, y^{(i)}))_{i=1}^{n} = ((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)}))$$

- (Sometimes x's and y's come separately: $(x^{(i)})_{i=1}^{n}$ and $(y^{(i)})_{i=1}^{n}$)

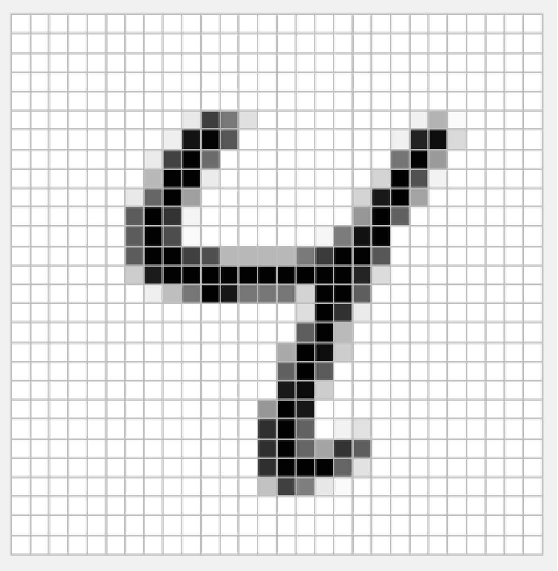
NN classifier for labeled dataset $\mathcal{S}$:

- Input: x
- Output: prediction of correct label of x
- Pseudocode:

Euclidean distance

$$D(x, z) = \|x - z\|$$

Image of digit as 784-vector: pixel intensities as features



Computational requirements of NN classifier:

- Memory

- Time

Why (or under what assumptions) should NN classifier work well?

```
import numpy as np

def learn(train_x, train_y):
  return (train_x, train_y)

def predict(params, test_x):
  x, y = params
  return y[np.argmin(np.sum(x**2, axis=1) - 2*test_x.dot(x.T),
   ↪  axis=1)]
```

If you want to strictly follow the idea that “learn” should return a function:

```
def learn(train_x, train_y):
  return lambda test_x: train_y[np.argmin(np.sum(train_x**2, axis=1)
   ↪  - 2*test_x.dot(train_x.T), axis=1)]
```

Evaluating a classifier

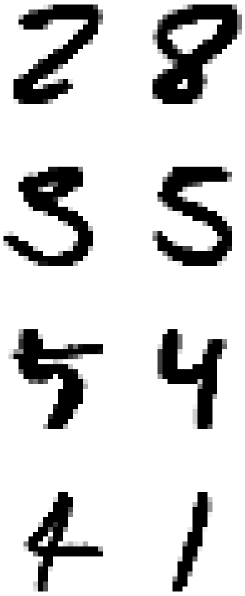
- Error rate on classifier f on labeled dataset:

- Training error rate (i.e., error rate on $\mathcal{S}$) of NN classifier:

NIST has provided **separate collection of 10000 labeled examples**, which we **did not provide to NN learning algorithm**

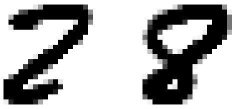
- We use it as test data
- Test error rate (i.e., error rate on test data) of NN classifier:

Test image, nearest neighbor in training data:

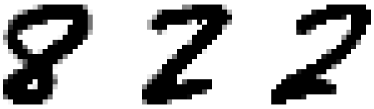


Upgrading NN: more neighbors

Test image, nearest neighbor in training data:



3 closest images in training data:



k-NN classifier for labeled dataset $\mathcal{S}$:

- Input: x
- Output: prediction of correct label of x
- Pseudocode:

hyperparameter k	1	3	5	7	9
test error rate	3.09%	2.95%	3.12%	3.06%	3.41%

Hyperparameter tuning (e.g., how to choose k?)

- Cross validation: use subset of training data to act as test data for purpose of evaluating different hyperparameter choices
- Pseudocode:

Upgrading NN: better distances

Other types of distances

- ℓ^p distance for d-vectors $x = (x_1, \dots, x_d)$

$$D_p(x, z) = \left(|x_1 - z_1|^p + \cdots + |x_d - z_d|^p\right)^{1/p}$$

Other types of distances

- "Edit distance" for strings (e.g., $x =$ "kitten")

$$D_{\text{edit}}(x, z) = \#\ \text{insertions/deletions/swaps needed to transform } x \text{ to } z$$

Digit recognition using NN classifier based on different distances

distance metric	ℓ^2	ℓ^3	"shape"
test error rate	3.09%	2.83%	$< 1\%$

Caution: many types of distances (e.g., ℓ^p distances) are sensitive to the quality of the numerical features

- 1000 extra irrelevant pixels with seemingly arbitrary intensity values

- Single irrelevant pixel with scale 1000 times that of regular pixels

“Curse of dimension”: weird effects in “high dimensional” feature spaces (e.g., space of all d-vectors for large d)

Question: How can we choose the distance function to use?