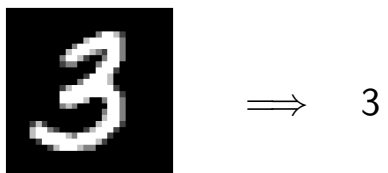


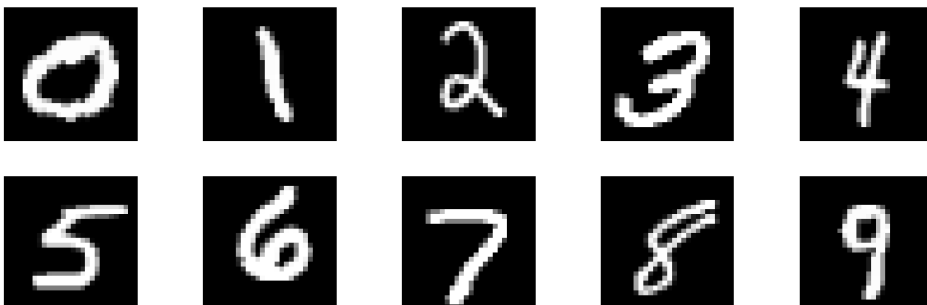
Nearest neighbor classification

The problem we'll solve today

Given an image of a handwritten digit, say which digit it is.



Some more examples:



The machine learning approach

Assemble a data set:



The MNIST data set of handwritten digits:

- **Training set** of 60,000 images and their labels.
- **Test set** of 10,000 images and their labels.

And let the machine figure out the underlying patterns.

Nearest neighbor classification

Training images $x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(60000)}$

Labels $y^{(1)}, y^{(2)}, y^{(3)}, \dots, y^{(60000)}$ are numbers in the range 0 – 9

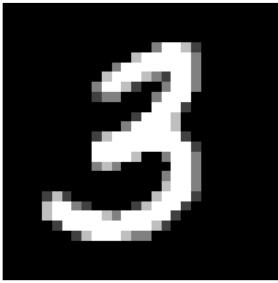


How to **classify** a new image x ?

- Find its nearest neighbor amongst the $x^{(i)}$
- Return $y^{(i)}$

The data space

How to measure the distance between images?



MNIST images:

- Size 28×28 (total: 784 pixels)
- Each pixel is grayscale: 0-255

Stretch each image into a vector with 784 coordinates:



- Data space $\mathcal{X} = \mathbb{R}^{784}$
- Label space $\mathcal{Y} = \{0, 1, \dots, 9\}$

The distance function

Remember Euclidean distance in two dimensions?

$$z = \underset{\bullet}{(3, 5)}$$

$$x = \underset{\bullet}{(1, 2)}$$

Euclidean distance in higher dimension

Euclidean distance between 784-dimensional vectors x, z is

$$\|x - z\| = \sqrt{\sum_{i=1}^{784} (x_i - z_i)^2}$$

Here x_i is the i th coordinate of x .

Nearest neighbor classification

Training images $x^{(1)}, \dots, x^{(60000)}$, labels $y^{(1)}, \dots, y^{(60000)}$



To classify a new image x :

- Find its nearest neighbor amongst the $x^{(i)}$ using **Euclidean distance in \mathbb{R}^{784}**
- Return $y^{(i)}$

How accurate is this classifier?

Accuracy of nearest neighbor on MNIST

Training set of 60,000 points.

- What is the error rate on training points? **Zero**.
In general, **training error** is an overly optimistic predictor of future performance.
- A better gauge: separate test set of 10,000 points.
Test error = fraction of test points incorrectly classified.
- What test error would we expect for a *random classifier*?
(One that picks a label 0 – 9 at random?) **90%**.
- Test error of nearest neighbor: **3.09%**.

Examples of errors

Test set of 10,000 points:

- 309 are misclassified
- Error rate 3.09%

Examples of errors:

Query					
NN					

Ideas for improvement: (1) *k*-NN (2) better distance function.

K-nearest neighbor classification

To classify a new point:

- Find the k nearest neighbors in the training set.
- Return the most common label amongst them.

MNIST:

k	1	3	5	7	9	11
Test error (%)	3.09	2.94	3.13	3.10	3.43	3.34

In real life, there's no test set. How to decide which k is best?

Cross-validation

How to estimate the error of k -NN for a particular k ?

10-fold cross-validation

- Divide the training set into 10 equal pieces.
Training set (call it S): 60,000 points
Call the pieces S_1, S_2, \dots, S_{10} : 6,000 points each.
- For each piece S_i :
 - Classify each point in S_i using k -NN with training set $S - S_i$
 - Let ϵ_i = fraction of S_i that is incorrectly classified
- Take the average of these 10 numbers:

$$\text{estimated error with } k\text{-NN} = \frac{\epsilon_1 + \dots + \epsilon_{10}}{10}$$

Another improvement: better distance functions

The Euclidean (ℓ_2) distance between these two images is very high!



Much better idea: distance measures that are invariant under:

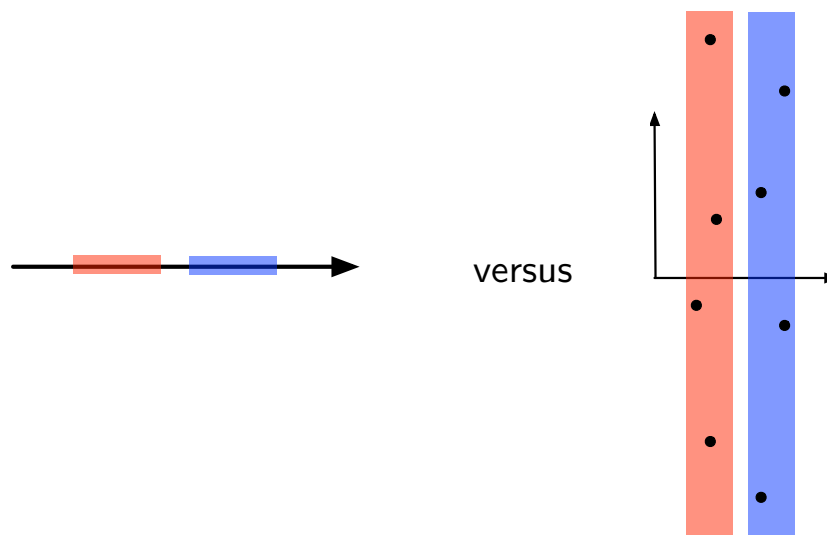
- Small translations and rotations. e.g. *tangent distance*.
- A broader family of natural deformations. e.g. *shape context*.

Test error rates:

ℓ_2	tangent distance	shape context
3.09	1.10	0.63

Related problem: feature selection

Feature selection/reweighting is part of picking a distance function.
And, one noisy feature can wreak havoc with nearest neighbor!



Algorithmic issue: speeding up NN search

Naive search takes time $O(n)$ for training set of size n : slow!

Luckily there are data structures for speeding up nearest neighbor search, like:

- ① Locality sensitive hashing
- ② Ball trees
- ③ K -d trees

These are part of standard Python libraries for NN, and help a lot.

Postscript:
Useful distance functions for machine learning

Measuring distance in \mathbb{R}^m

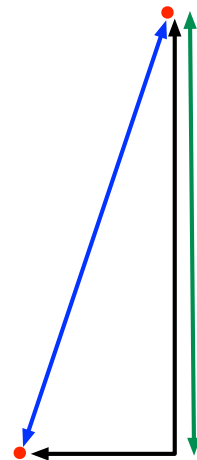
Usual choice: **Euclidean distance**:

$$\|x - z\|_2 = \sqrt{\sum_{i=1}^m (x_i - z_i)^2}.$$

For $p \geq 1$, here is ℓ_p **distance**:

$$\|x - z\|_p = \left(\sum_{i=1}^m |x_i - z_i|^p \right)^{1/p}$$

- $p = 2$: Euclidean distance
- ℓ_1 distance: $\|x - z\|_1 = \sum_{i=1}^m |x_i - z_i|$
- ℓ_∞ distance: $\|x - z\|_\infty = \max_i |x_i - z_i|$



Example 1

Consider the all-ones vector $(1, 1, \dots, 1)$ in \mathbb{R}^d .
What are its ℓ_2 , ℓ_1 , and ℓ_∞ length?

Example 2

In \mathbb{R}^2 , draw all points with:

- ① ℓ_2 length 1
- ② ℓ_1 length 1
- ③ ℓ_∞ length 1

Metric spaces

Let \mathcal{X} be the space in which data lie.

A distance function $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a **metric** if it satisfies these properties:

- $d(x, y) \geq 0$ (nonnegativity)
- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

Example 1

$\mathcal{X} = \mathbb{R}^m$ and $d(x, y) = \|x - y\|_p$

Check:

- $d(x, y) \geq 0$ (nonnegativity)
- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

Example 2

$\mathcal{X} = \{\text{strings over some alphabet}\}$ and $d = \text{edit distance}$

Check:

- $d(x, y) \geq 0$ (nonnegativity)
- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

A non-metric distance function

Let p, q be probability distributions on some set \mathcal{X} .

The **Kullback-Leibler divergence** or **relative entropy** between p, q is:

$$d(p, q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}.$$