Nearest neighbor classification

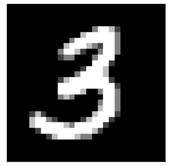
DSE 220

Outline

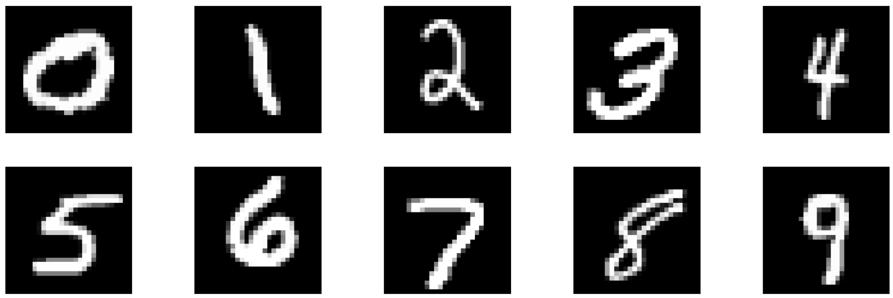
1. Nearest neighbor classification
2. k-nearest neighbor
3. Choosing the features and distance function

The problem we'll solve today

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

 $\Longrightarrow$ 3

More examples:



The machine learning approach

Assemble a data set:

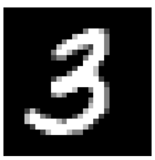
The MNIST data set of handwritten digits:

- **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 $x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(60000)}$
Labels $y^{(1)}, y^{(2)}, y^{(3)}, \ldots, y^{(60000)}$ are numbers in the range $0 - 9$

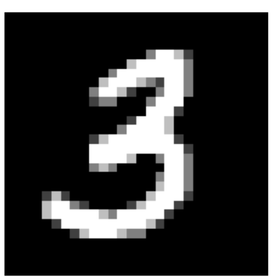


How to **classify** a new image x?

- 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 $\mathcal{Y} = \{0, 1, \ldots, 9\}$

The distance function

Remember Euclidean distance in two dimensions?

$z = (3, 5)$

$x = (1, 2)$

Euclidean distance in higher dimension

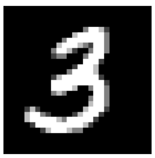
Euclidean distance between 784-dimensional vectors x, z is

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

Here x_i is the ith coordinate of x.

Nearest neighbor classification

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



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:

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

Outline

1. Nearest neighbor classification
2. k-nearest neighbor
3. Choosing the features and distance function

K-nearest neighbor classification

Classify a point using the labels of its k nearest neighbors among the training points.

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?

1. Hold-out set.
    - Let S be the training set.
    - Choose a subset $V \subset S$ as a *validation set*.
    - What fraction of V is misclassified by the k-nearest neighbors in $S \setminus V$?

2. Leave-one-out cross-validation.
    - For each point $x \in S$, find the k-nearest neighbors in $S \setminus \{x\}$.
    - What fraction are misclassified?

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 $S_1, S_2, \ldots, S_{10}$: 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:

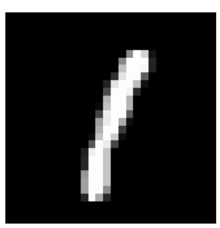
$$\text{estimated error with } k\text{-NN} \;=\; \frac{\epsilon_1 + \cdots + \epsilon_{10}}{10}$$

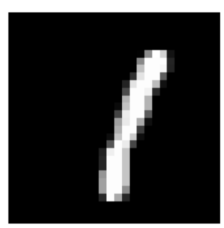
Outline

1. Nearest neighbor classification
2. *k*-nearest neighbor
3. Choosing the features and distance function

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:

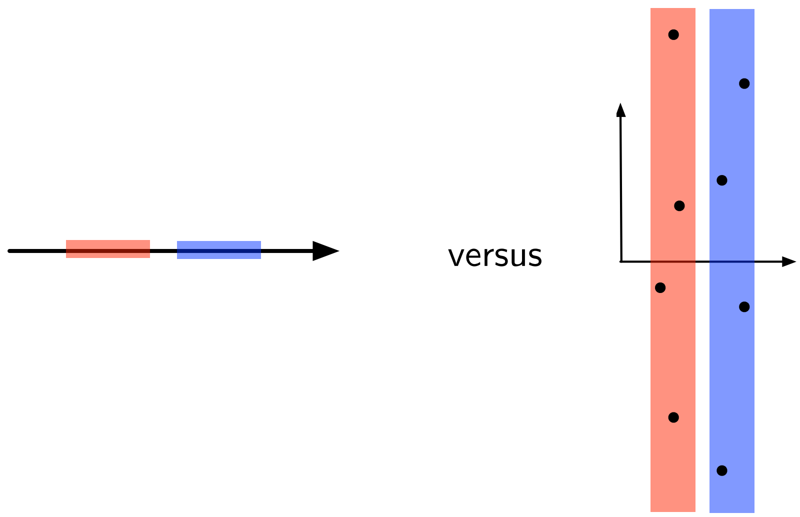
- Small translations and rotations. e.g. *tangent distance*.
- A broader family of natural deformations. e.g. *shape context*.

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!

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.

Example: k-d trees for NN search

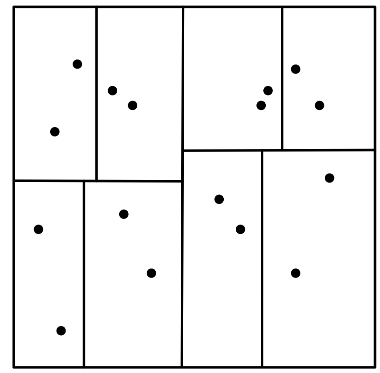
A hierarchical, rectilinear spatial partition.

For data set $S \subset \mathbb{R}^d$:

- Pick a coordinate $1 \leq i \leq d$.
- Compute $v = \text{median}(\{x_i : x \in S\})$.
- Split S into two halves:

$$S_L = \{x \in S : x_i < v\}$$
$$S_R = \{x \in S : x_i \geq v\}$$

- Recurse on S_L, S_R



Two types of search, given a query $q \in \mathbb{R}^d$:

- *Defeatist search*: Route q to a leaf cell and return the NN in that cell. This might not be the true NN.
- *Comprehensive search*: Grow the search region to other cells that cannot be ruled out using the triangle inequality.