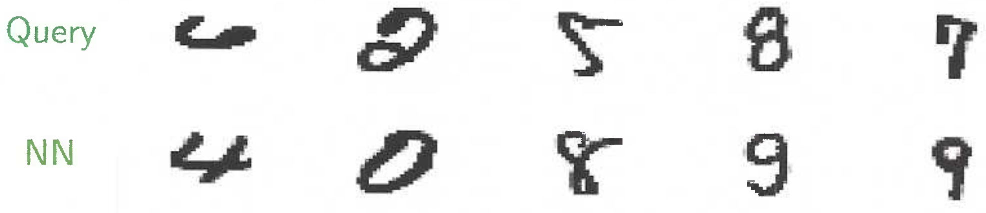
Improving the performance of nearest neighbor

Recall: nearest neighbor on MNIST

- Images of handwritten digits, represented as vectors in $\mathbb{R}^{784}$.
- Labels $0 - 9$
- Training set: 60,000 points; test set: 10,000 points

Test error of nearest neighbor using Euclidean distance: 3.09%

Examples of errors:



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.

if K=1, we choose 1 NN
K=3, we choose 3 NN

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**

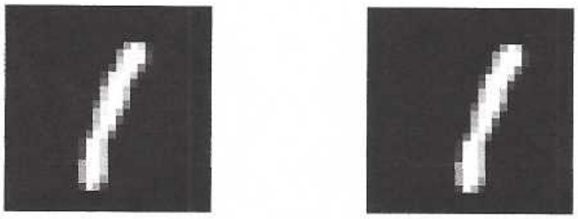
- Divide the training set into 10 equal pieces.
  Training set (call it S): 60,000 points
  Call the pieces $S_1, S_2, \ldots, S_{10}$: 6,000 points each.
- For each piece S_i:
  - Classify each point in S_i using k-NN with training set $S - S_i$
  - Let ϵ_i = fraction of S_i that is incorrectly classified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

10 equal pieces, 9 piece as training
1 piece as test

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

Another improvement: better distance functions

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



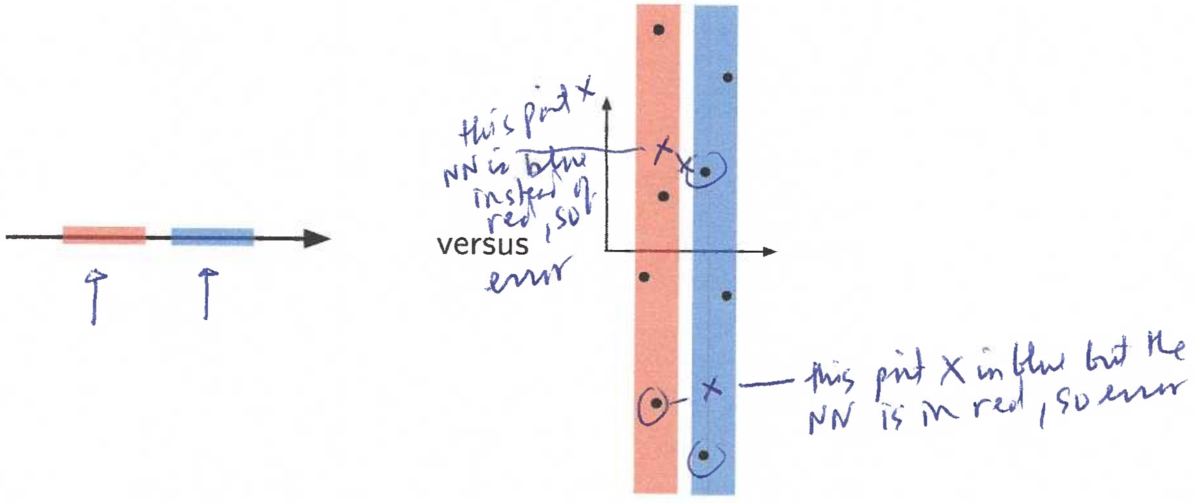
Much better idea: distance measures that are invariant under:

- 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!


versus
this pnt X NN is ~~btw~~ instead of red, so error
this pnt X in blue but the NN is in red, so error

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.

Nearest Neighbor for classification — Build model from data
Use model to make predictions

simplest & most flexible