Machine learning lecture slides

COMS 4771 Fall 2020

Nearest neighbor classification

Outline

- Optical character recognition (OCR) example
- Nearest neighbor rule
- Error rate, test error rate
- k-nearest neighbor rule
- Hyperparameter tuning via cross-validation
- Distance functions, features
- Computational issues

Example: OCR for digits

- Goal: Automatically label images of handwritten digits
- Possible labels are $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Start with a large collection of already-labeled images
  - $D := \{(x_1, y_1), \ldots, (x_n, y_n)\}$
  - x_i is the i-th image; $y_i \in \{0, 1, \ldots, 9\}$ is the corresponding label.
  - The National Institute for Standards and Technology (NIST) has amassed such a data set.

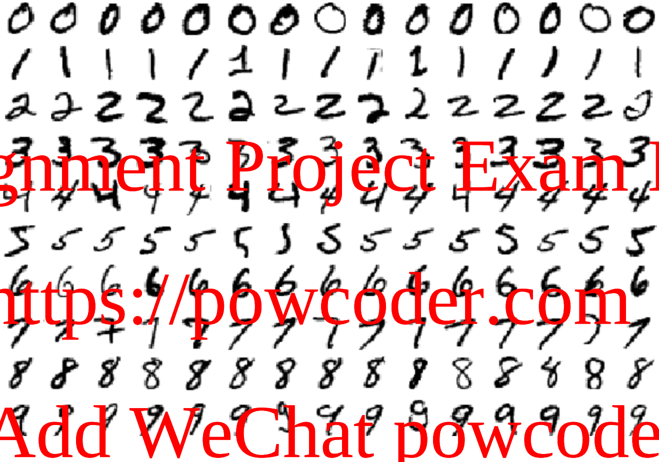

Figure 1: Some images of handwritten digits from MNIST data set

Nearest neighbor (NN) classifier

- *Nearest neighbor (NN) classifier* NN_D:
  - Represented using collection of labeled examples $D := ((x_1, y_1), \ldots, (x_n, y_n))$, plus a snippet of code
- Input: x
  - Find x_i in D that is "closest" to x (the *nearest neighbor*)
  - (Break ties in some arbitrary fixed way)
  - Return y_i

- Treat (grayscale) images as vectors in Euclidean space $\mathbb{R}^d$
  - $d = 28^2 = 784$
  - Generalizes physical 3-dimensional space
- Each point $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ is a vector of d real numbers
  - $\|x - z\|_2 = \sqrt{\sum_{j=1}^{d} (x_j - z_j)^2}$
  - Also called ℓ_2 distance
  - WARNING: Here, x_j refers to the j-th coordinate of x

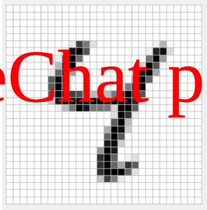

Figure 2: Grayscale pixel representation of an image of a handwritten "4"

Naïve distance between images of handwritten digits (2)

- Why use this for images?

- Why not use this for images?

- What is the core prediction problem?

- What *features* (i.e., predictive variables) are available?
- Will these features be available at time of prediction?

- Is there enough information ("training data") to learn the relationship between the features and label?

- What are the modeling assumptions?
- Is high-accuracy prediction a useful goal for the application?

Error rate

- *Error rate* (on a collection of labeled examples S)
  - Fraction of labeled examples in S that have incorrect label prediction from $\hat{f}$
  - Written as $\mathrm{err}(\hat{f}, S)$
  - (Often, the word "rate" is omitted)
- OCR via NN:

$$\mathrm{err}(\mathrm{NN}_{\mathcal{D}}, \mathcal{D}) =$$

Test error rate (1)

- Better evaluation: *test error rate*
  - Train/test split, $S \cap T = \emptyset$
    - S is *training data*, T is *test data*
  - Classifier $\hat{f}$ only based on S
  - *Training error rate*: $\mathrm{err}(\hat{f}, S)$
  - *Test error rate*: $\mathrm{err}(\hat{f}, T)$
- On OCR data: test error rate is $\mathrm{err}(\mathrm{NN}_S, T) = 3.09\%$
  - Is this good?
    - What is the test error rate of uniformly random predictions?
    - What is the test error rate of a constant prediction?

Test error rate (2)

- Why is test error rate meaningful?

- What are the drawbacks of evaluation via test error rate?

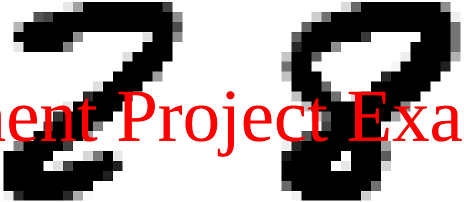

Figure 3: A test example and its nearest neighbor in training data (2, 8)

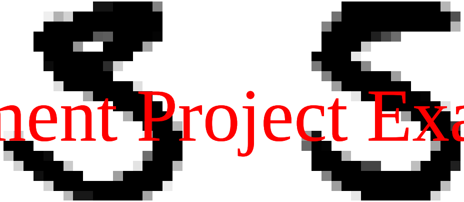

Figure 4: A test example and its nearest neighbor in training data (3, 5)

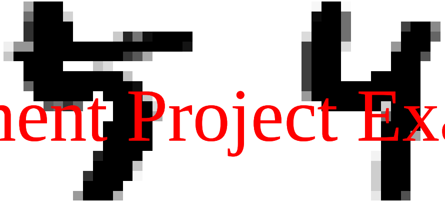

Figure 5: A test example and its nearest neighbor in training data (5, 4)

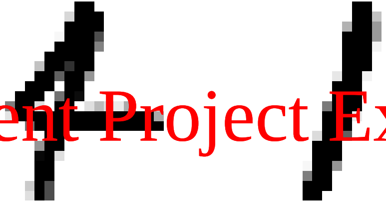

Figure 6: A test example and its nearest neighbor in training data (4, 1)

More on the modeling assumptions

- Modeling assumption: Nearby images are more likely to have the same label than different labels.
  - This is an assumption about the choice of distance function
  - In our OCR example, this is an assumption about the choice of features

Diagnostics

- What are the kinds of errors made by NN_S?

Figure 7: A test example and its nearest neighbor in training data (2, 8)

Figure 8: Three nearest neighbors of the test example (8,2,2)

Upgrade: k-NN

- *k-nearest neighbor (k-NN) classifier* $\mathrm{NN}_{k,D}$
- Input: x
  - Find the k nearest neighbors of x in D
  - Return the plurality of the corresponding labels
- As before, break ties in some arbitrary fixed way

Typical effect of k

- Smaller k: smaller training error rate
- Larger k: higher training error rate, but predictions more "stable" due to voting
- On OCR data: lowest test error rate achieved at $k = 3$

k	1	3	5	7	9
$\mathrm{err}(\mathrm{NN}_{k,S}; T)$	0.0309	0.0295	0.0312	0.0306	0.0341

Hyperparameter tuning

- k is a *hyperparameter* of k-NN
- How to choose hyperparameters?
  - Bad idea: Choosing k that yields lowest training error rate (degenerate choice: $k = 1$)
  - Better idea: Simulate train/test split on the training data
- Hold-out validation
  - Randomly split S into A and B
  - Compute *validation error rate* for all $k \in \{1, 3, 5, 7, 9\}$:

$$V_k := \mathrm{err}(\mathrm{NN}_{k,A}, B)$$

  - Let $\hat{k}$ be the value of k for which V_k is smallest
  - Classifier to use is $\mathrm{NN}_{\hat{k},S}$

Upgrade: Distance functions (1)

- Specialize to input types
  - Edit distance for strings
  - Shape distance for images
  - Time warping distance for audio waveforms

Upgrade: Distance functions (2)

- Generic distances for vectors of real numbers
  - ℓ_p distances

$$\|x - z\|_p = \left(\sum_{j=1}^{d} |x_j - z_j|^p \right)^{1/p} .$$

  - What are the unit balls for these distances (in $\mathbb{R}^2$)?

- Distance functions for images of handwritten digits

distance	ℓ_2	ℓ_3	tangent	shape
test error rate	0.0309	0.0283	0.0110	0.0063

Features

- When using numerical *features* (arranged in a vector from $\mathbb{R}^d$):
  - Scale of features matters
  - Noisy features can ruin NN
    - E.g., consider what happens in OCR example if you have another 10000 additional features that are pure "noise"
    - Or a single pure noise feature whose scale is $10000\times$ the nominal scale of pixel values

- "Curse of dimension"
  - Weird effects in $\mathbb{R}^d$ for large d
  - Can find $2^{\Omega(d)}$ points that are approximately equidistant

- Brute force search: $\Theta(dn)$ time for each prediction (using Euclidean distance in $\mathbb{R}^d$)
- Clever data structures: "improve" to $2^d \log(n)$ time

- Approximate nearest neighbors: sub-linear time to get "approximate" answers
  - E.g., find point among the top-1% of closest points?