Nonparametric Density Estimation

Matthew Stone
CS 520, Spring 2000
Lecture 6

Our Learning Problem, Again

- **Use training data to estimate unknown probabilities and probability density functions**
- **So far, we have depended on describing the functions in a known parametric form**
- **Today, we relax that assumption**

Let's Start with an Obvious Idea

- **Nearest-neighbor classification**
  Algorithm:
  - Start with n points of training data:
    $$\mathscr{D}^n = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$$
  - Given test point **x**
  - Find training point **x′** closest to **x**
  - Assign **x** the same category as **x′**

How Well Does This Work?

- **Hard to say unless you have a lot of data**
  - But suppose data is no object

- **Label of x′ is class** ω'**; true label of x is** ω
- **Correct answer if** $\omega' = \omega$**: What's** $P(\omega' = \omega)$**?**
  - $p(\omega|\mathbf{x}')$ in general
  - $p(\omega|\mathbf{x})$ as **x′** becomes closer to **x**

How Well Does This Work?
continued

- **Probability of error at x is therefore:**

$$1-\sum_{i=1}^{c} P(\omega_i \mid \mathbf{x})P(\omega_i \mid \mathbf{x})$$

  – i.e., wrong in all cases except those where **x** and **x′** happen to agree.

- **In principle, best you could do is:**

$$1-P(\omega_i \mid \mathbf{x})$$

  – i.e., guess most likely

How Well Does This Work?
some perspective

- **Anyone who's anyone gets 95% accuracy**
  – When Bayes error is 5%

$$1-P(\omega_i \mid \mathbf{x})$$

  – Limit nearest-neighbor error is ~9%

$$1-\sum_{i=1}^{c} P(\omega_i \mid \mathbf{x})P(\omega_i \mid \mathbf{x})$$

    - Could be better, if distributions are favorable
    - Could be worse, because you don't have infinite data

How Well Does This Work?

some perspective

- **Surprisingly good (since it's so easy)**
- **But it may not be enough for your task**
  - Classifying sequences
    - At 7 elements, Bayes could get 2/3 right
    - Nearest neighbor is just getting 1/2 right

A Possible Improvement

- **K-Nearest Neighbor Classification**
  - Start with n points of training data:

$$\mathscr{D}^n = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$$

  - Given test point **x**
  - Find k training points X closest to **x**
  - Assign **x** the most frequent category of X

K-Nearest Neighbor

- **Good points:**
  - More likely data can overcome rare events
    - In nearest neighbor, each rare data point translates into a ball of likely mistakes
    - In 3-nearest neighbor, you need two rare data points together to get a ball of likely mistakes
    - Can get better and better the more points vote

K-Nearest Neighbor

- **Bad points:**
  - Need tons more data
    - Only if voters are close to **x** does vote provide good density information about **x**
    - Only by considering lots of voters do you converge on an accurate likelihood for **x**

Returning to Density Estimation

- **Haven't we changed the problem?**
  - – K-nearest neighbor is a classifier
  - – Maximum likelihood builds a distribution
- **Want to get a distribution for KNN**
  - – Compare approaches
  - – Mix KNN and other info probabilistically

Returning to Density Estimation

- **Basic nearest neighbor idea works**
  - – To find $p(\mathbf{x},\omega_i)$
    - Place a cell of volume V around **x**
    - Capture k samples, of which i are in ω_i
    - Calculate

$$p(\mathbf{x},\omega_i) = \frac{i/k}{V}$$

$$p(\mathbf{x}\,|\,\omega_i) = \frac{p(\mathbf{x},\omega_i)}{P(\omega_i)}$$

Returning to Density Estimation

- **Basic nearest neighbor idea works**
  - Well, almost…
    - Real probabilities should integrate to one (Although you don't always need real ω_i probabilities to build discriminant functions)
    - Volume V varies as a function of $\mathbf{x}$ so you may have trouble across the whole space

$$p(\mathbf{x}, \omega_i) = \frac{i/k}{V}$$

Sample-based Density Estimation

- **We'll now consider a close variant of KNN that represents the density more conveniently**

Parzen Windows

Parzen Windows

- **Treat each sample as contributing a small Gaussian density that peaks around it and drops off quickly**
  - Use parameter h (dummy for variance σ) to control drop off
  - Density around **u** is:

$$\varphi(\mathbf{x};\mathbf{u}) = \frac{1}{\sqrt{2\pi}h}\exp\frac{-(\mathbf{x}-\mathbf{u})^{\mathrm{T}}(\mathbf{x}-\mathbf{u})}{2\mathrm{h}^2}$$

Parzen Windows

- **Overall density for data**

$$\mathscr{D}^n = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$$

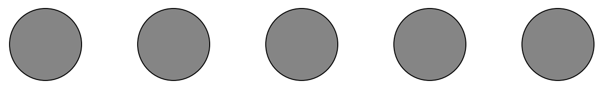
- **is**

$$p_n(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\varphi(\mathbf{x};\mathbf{x}_i)$$

Cute Implementation

- **3 Layer Network:**
- **Layer One:**

Inputs

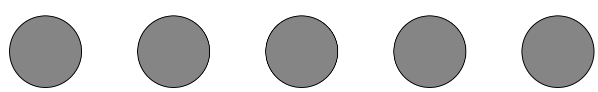


  - Each node gets a feature of the pattern that you're classifying
  - Pattern is normalized to have unit length

Cute Implementation

- **3 Layer Network:**
- **Layer Two:**

Patterns



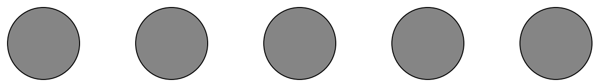
  - Each node gets a normalized training vector **w**; on input **x** it computes

$$z = \mathbf{w}^{T}\mathbf{x}$$

Cute Implementation

- **3 Layer Network:**
- **Layer Two:**

Patterns



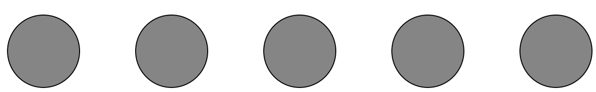
– The node will output likelihood component

$$e^{(z-1)^2/\sigma^2}$$

Cute Implementation

- **3 Layer Network:**
- **Layer Three:**

Categories



– One node per class

– Sums input from pattern nodes for training data in the class

$$e^{(z-1)^2/\sigma^2}$$

Kind of “Neural Network”

- **Easy to train**
  - Add a new pattern node for each sample
- **Easy to interpret probabilistically**
  - Approximates arbitrary input distributions (using samples)
  - Outputs Bayes optimal classification given its assumed distribution of inputs