Outline

- \( k \)-Nearest Neighbor
- Locally weighted regression
- Radial basis functions
- Case-based reasoning
- Lazy and eager learning

Nearest Neighbor

Key idea: just store all training examples \( (x_i, f(x_i)) \)

Need some distance measure between instances (e.g. Euclidean distance, Hamming distance)

Nearest neighbor:
- Given query instance \( x_q \), first locate nearest training example \( x_n \), then estimate \( f(x_q) = f(x_n) \)

\( k \)-Nearest neighbor:
- Given \( x_q \), take vote among its \( k \) nearest neighbors (if discrete-valued target function)
  - Let \( k \) not be divisible by number of possible labels
- Take mean of \( f \) values of \( k \) nearest neighbors if \( f \) real-valued
  \[
  f(x_q) = \frac{\sum_{i=1}^{k} f(x_i)}{k}
  \]

When To Consider Nearest Neighbor

- Instances are in \( \mathbb{R}^n \) (or, one can define some distance measure between instances; can use kernels)
- Less than 20 attributes per instance
  - To avoid curse of dimensionality, where many irrelevant attributes causes distance to be large, but distance is small if only relevant attributes used
  - Also, large number of attributes increases classification complexity
- Lots of training data

Advantages:
- Robust to noise
- Stable
- Training is very fast
- Learn complex target functions
- Don’t lose information

Disadvantages:
- Slow at query time (active research area: fast indexing and accessing algorithms)
- Easily fooled by irrelevant attributes

Nearest Neighbor’s Behavior in the Limit

Consider \( p(x) \) defines probability that instance \( x \) will be labeled positive (versus negative)

Nearest neighbor (\( k = 1 \)):
- As number of training examples \( \rightarrow \infty \), approaches Gibbs Algorithm
  Recall Gibbs has at most twice the expected error of Bayes optimal

\( k \)-Nearest neighbor:
- As number of training examples \( \rightarrow \infty \) and \( k \) gets large, approaches Bayes optimal (best possible with given hyp. space and prior information)
  Bayes optimal: if \( p(x) > 0.5 \) then predict \( + \), else \( - \)
Distance-Weighted k-NN

Might want weight nearer neighbors more heavily:

$$f(x_q) = \arg\max_{v \in V} \sum_{i=1}^{V} w_i \delta(v, f(x_i))$$

for discrete-valued ($\delta(v, f(x_i)) = 1$ if $v = f(x_i)$ and 0 otherwise), and

$$f(x_q) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

for continuous

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

and $d(x_q, x_i)$ is distance between $x_q$ and $x_i$.

Note now it makes sense to use all training examples instead of just $k$ (Shepard's method), but then get increased time to classify instances.

Curse of Dimensionality

Imagine instances described by 20 attributes, but only 2 are relevant to target function.

Curse of dimensionality: nearest neighbor is easily misled by high-dimensional $X$.

One approach:

- Stretch $j$th axis by weight $z_j$, where $z_1, \ldots, z_n$ chosen to minimize prediction error.
- Use cross-validation to automatically choose weights $z_1, \ldots, z_n$.
- Setting $z_j$ to zero eliminates this dimension altogether.

see [Moore and Lee, 1994]

RBF Networks (cont'd)

$$f(x) = w_0 + \sum_{i=1}^{k} w_i K_u(d(x, x_i))$$

(Note no weights from input to hidden layer)

One common choice for $K_u(d(x, x_i))$ is

$$K_u(d(x, x_i)) = \exp\left(-\frac{1}{2\sigma_u^2} d^2(x, x_i)\right),$$

i.e. Gaussian with mean at $x_u$ and variance $\sigma_u^2$, all features independent

[see bug on p. 239]

Locally Weighted Regression

Note k-NN forms local approximation to $f$ for each query point $x_q$. Why not form an explicit approximation $f(x)$ for region surrounding $x_q$?

- Fit linear, quadratic, etc. function to $k$ nearest neighbors.
- Produces “piecewise approximation” to $f$.
- Do this for each new query point $x_q$.

Several choices of error to minimize:

- Squared error over $k$ nearest neighbors:

$$E_1(x_q) = \frac{1}{2} \sum_{z \in k \text{ nearest } x_i} (f(z) - f(x_q))^2$$

- Distance-weighted squared error over all nbrs:

$$E_2(x_q) = \frac{1}{2} \sum_{z \in D} (f(z) - f(x_q))^2 K(d(x_q, z))$$

($K$ is decreasing in its argument)
- Combine $E_1$ and $E_2$.

Training Radial Basis Function Networks

1. Choose number of kernel functions (hidden units)

- If $= \text{number training instances}$, can fit training data exactly by placing one center per instance.
- Using fewer => more efficient, less chance of overfitting.

2. Choose center ($= \text{mean for Gaussian}$) $x_u$ of kernel function $K_u(d(x_u, x_i))$.

- Use all training instances if enough kernels avail.
- Use subset of training instances.
- Scatter uniformly throughout instance space.
- Can cluster data and assign one per cluster (helps answer step 1 also).
- Can use EM to find means of mixture of Gaussians.
- Can also use e.g. EM to find $\sigma_u$'s (for Gaussian).

3. Hold kernels fixed and train weights to fit linear function (output layer), e.g. GD or EG.
Case-Based Reasoning and CADET

Can apply instance-based learning even when \( X \) much more complex

Need different “distance” metric

Case-Based Reasoning is instance-based learning where instances have symbolic logic descriptions

\[
\text{((user-complaint error53-on-shutdown)}
\text{ (cpu-model PowerPC) (operating-system Windows)}
\text{ (memory 48meg)}
\text{ (installed-apps Excel Netscape VirusScan)}
\text{ (disk 1gig)}
\text{ (likely-cause ???))}
\]

CADET: 75 stored examples of mechanical devices, e.g. water faucets

- Training instance: (qualitative function, mech. structure)
- New query: desired function
- Target value: mechanical structure for this function

Distance metric: match qualitative function descriptions

Lazy and Eager Learning

Lazy: Wait for query before generalizing

- \( k \)-NN, locally weighted regression, case-based reasoning

Eager: Generalize before seeing query

- Radial basis function networks, ID3, backpropagation, naive Bayes

Does it matter?

- Computation time for training and generalization
- Eager learner must create global approximation, lazy learner can create many local approximations
- If they use same \( H \), lazy can represent more complex functions (e.g. consider \( H = \text{linear functions} \) since it considers the query instance \( x_q \) before generalizing, i.e. lazy produces a new hypothesis for each new \( x_q \)