Instance-Based Learning

- no explicit description of the target function
- generalization postponed until new instance has to be classified
- "lazy learning"
- local approximation of the target function
- High cost: training vs. classification!!!
- Case-base reasoning: more complex, symbolic representations for instances

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

Nearest neighbor:
- Given query instance $x_q$, first locate nearest training example $x_n$, then estimate $\hat{f}(x_q) \leftarrow f(x_n)$

$k$-Nearest neighbor:
- Given $x_q$, take vote among its $k$ nearest nbrs (if discrete-valued target function)
- take mean of $f$ values of $k$ nearest nbrs (if real-valued)

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$
Nearest neighbor

- Instances map to points in \( n \)-dimensional space \( \mathbb{R}^n \)
- Nearest neighbors defined by standard Euclidean distance given their feature vectors

\[
d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}
\]

When To Consider Nearest Neighbor

- Less than 20 attributes per instance
- Lots of training data

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

Disadvantages:
- Slow at query time
- Easily fooled by irrelevant attributes

Voronoi Diagram

3-Nearest Neighbors

query point \( q \)

nearest neighbor \( q \)
Behavior in the Limit

Consider \( p(x) \) defines probability that instance \( x \) will be labeled 1 (positive) versus 0 (negative).

Nearest neighbor:
- As number of training examples \( \to \infty \), approaches Gibbs Algorithm
  
  Gibbs: with probability \( p(x) \) predict 1, else 0

\( k \)-Nearest neighbor:
- As number of training examples \( \to \infty \) and \( k \) gets large, approaches Bayes optimal
  
  Bayes optimal: if \( p(x) > .5 \) then predict 1, else 0

Note Gibbs has at most twice the expected error of Bayes optimal
Nearest Neighbors (continuous)

Locally Weighted Regression (terminology)

- Regression means approximating a real-valued target function
- Residual is the error \( \hat{f}(x) - f(x) \) in approximating the target function
- Kernel function is the function of distance that is used to determine the weight of each training example
- In other words: the kernel function is the function \( K \) such that \( w_i = K(d(x_i, x_q)) \)

Distance-Weighted kNN

Might want weight nearer neighbors more heavily...

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

where

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

(or \( w_i = K(d(x_i, x_q)) \) in general 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
**Distance Weighted k-NN**

Give more weight to neighbors closer to the query point

\[ f^*(x_q) = \sum_{i=1}^{k} w_i f(x_i) / \sum_{i=1}^{k} w_i \]

where \( w_i = K(d(x_q, x_i)) \) and \( d(x_q, x_i) \) is the distance between \( x_q \) and \( x_i \).

Instead of only k-nearest neighbors, use all training examples (Shepard’s method)

---

**Distance Weighted Average**

- Weighting the data:
  \[ f^*(x_q) = \frac{\sum_i f(x_i)K(d(x_i, x_q))}{\sum_i K(d(x_i, x_q))} \]
- Relevance of a data point \((x_i, f(x_i))\) is measured by calculating the distance \(d(x_i, x_q)\) between the query \( x_q \) and the input vector \( x_i \).
- Weighting the error criterion:
  \[ E(x_q) = \sum_i (f^*(x_q) - f(x_i))^2 K(d(x_i, x_q)) \]

the best estimate \( f^*(x_q) \) will minimize the cost \( E(x_q) \), therefore

\[ \frac{\partial E(x_q)}{\partial f^*(x_q)} = 0 \]

---

**Kernel Functions**

**Distance Weighted NN**

\[ K(d(x_q, x_i)) = \frac{1}{d(x_q, x_i)^2} \]
Distance Weighted NN

\[ K(d(x_i,x_j)) = \frac{1}{(d_0 + d(x_i,x_j))^2} \]

Example: Mexican Hat

\[ f(x_1,x_2) = \sin(x_1)\sin(x_2)/x_1x_2 \]

Approximation

Residual
Locally Weighted Linear Regression

- Local linear function
  \[ f^*(x) = w_0 + \sum_n w_n x_n \]
- Error criterion
  \[ E = \sum_i (w_0 + \sum_n w_n x_n - f(x_i))^2 K(d(x_i, x_q)) \]
- Gradient descent
  \[ \Delta w_n = \sum_i (f^*(x_q) - f(x_i)) x_n K(d(x_i, x_q)) \]
- Least square solution
  \[ w = (KK^T)^{-1}Kf(X) \]

with \(K\) being the matrix of row vectors, \(K(d(x_i, x_q))\) is a vector whose \(i\)-th element is \(f(x_i)\)

Curse of Dimensionality

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

*Curse of dimensionality*: nearest nbr is easily mislead when 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 \(z_1, \ldots, z_n\)
- Note setting \(z_j\) to zero eliminates this dimension altogether (feature subset selection)

Radial Basis Function Networks

- Global approximation to target function, in terms of linear combination of local approximations
- Used, e.g., for image classification
- A different kind of neural network - similar to back-propagation neural network but activation function is Gaussian rather than sigmoid
- Closely related to distance-weighted regression, but “eager” instead of “lazy”

where \(a_i(x)\) are the attributes describing instance \(x\), and

\[ f(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x)) \]

One common choice for \(K_u(d(x_u, x))\) is

\[ K_u(d(x_u, x)) = e^{-\frac{1}{2\sigma^2}d^2(x_u, x)} \]
Radial Basis Function Networks

Training Radial Basis Function Networks

Q1: What center \( x_u \) to use for each kernel function \( K_u(d(x_u,x)) \)
- Scatter uniformly throughout instance space
- Use distribution of training instances (clustering)

Q2: How to train weights (assume here Gaussian \( K_u \))
- First choose mean and variance for each \( K_u \)
  - e.g., use EM
- Then hold \( K_u \) fixed, and train linear output layer
  - Efficient methods to fit linear function

Lazy and Eager Learning

Lazy: wait for query before generalizing
- \( k \)-NEAREST NEIGHBOR, Case based reasoning

Eager: generalize before seeing query
- Radial basis function networks, ID3, Backpropagation, Naive-Bayes, ...

Does it matter?
- Eager learner must create global approximation
- Lazy learner can create many local approximations
- If they use same \( H \), lazy can represent more complex fn's (e.g., consider \( H = \) linear functions)

Radial Basis Function Networks

\[
f(x) = w_0 + \sum_{i=1}^{k} w_i K_i(d(x_i,x))
\]

Lazy and Eager Learning