Universität Augsburg University

Instance Based Learning

Instance-Based Learning

- No explicit description of the target function is constructed
- Instead: Simply the training examples are stored and used to perform classification
- Generalization is postponed until a new instance must be classified
- Each time a new query instance is encountered, its relationship to the previously stored examples is examined in order to assign a target function value to the new instance.

Outline

- *k*-Nearest Neighbor
- Locally weighted regression
- Radial basis functions
- Lazy and eager learning
  - ↳ Because processing is delayed until a new instance must be classified

1. k-NN Learning (1)

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

Classification:

Nearest neighbor (NN):

– Given query instance x_q, first locate nearest training example x_n, then estimate

$$\hat{f}(x_q) \leftarrow f(x_n)$$

Nearest = e.g. Euclidean distance in $\mathbb{R}^d$

$$n = arg \min_{i \in \{1,\ldots,N\}} \|x_q - x_i\|_{L2}$$

with

$$\|x_q - x_i\|_{L2} = d(x_q, x_i) = \sqrt{(x_q - x_i)^T (x_q - x_i)}$$

… Classification:

k-Nearest neighbor (k-NN):

Given x_q,

– take vote among its *k* nearest neighbors (if discrete-valued target function)

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

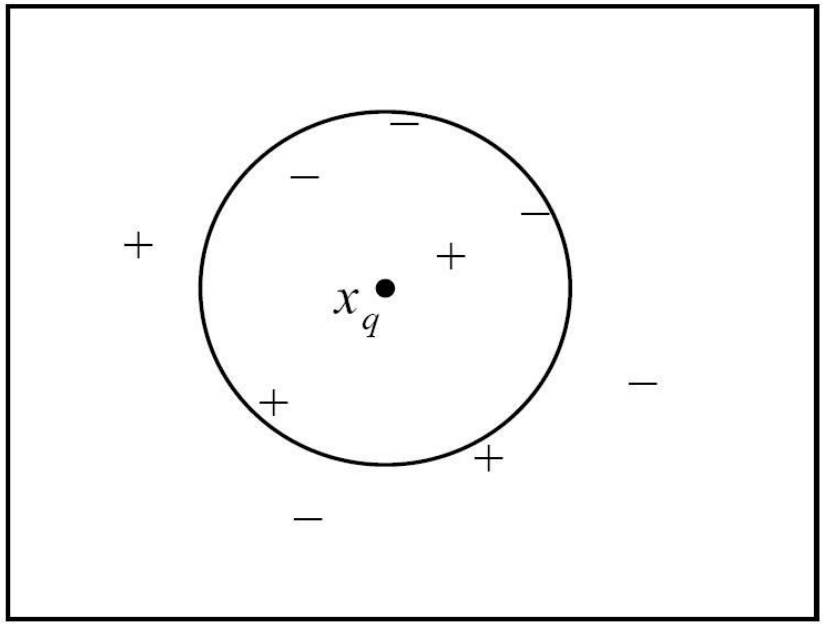
with

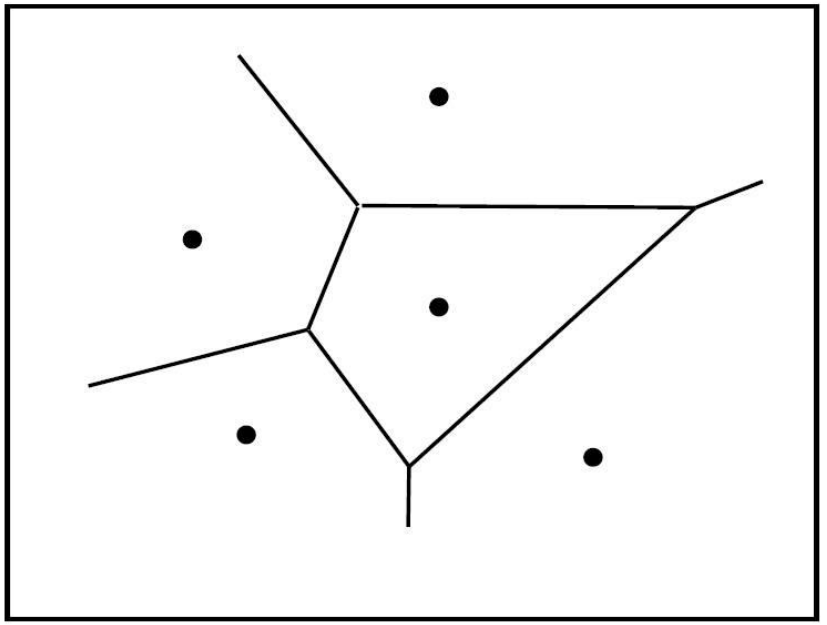
$$\delta(a,b) = \begin{cases} 1 & if\ a == b \\ 0 & else \end{cases}$$

– take mean of *f* values of *k* nearest neighbors (if real-valued target function)

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

Voronoi Diagram





1-NN ➔ x_q = ?
5-NN ➔ x_q = ?

1-NN decision surface (Convex polygons) ➔ Shows for every possible instance its classification (given *h*)

When To Consider Nearest Neighbor

- Instances map to points in $\mathbb{R}^d$
- Less than 20 attributes per instance (→ Curse of dimensionality; for $d \to \infty$ everything is far away)
- 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

Cannot learn the relevant attributes

Example:

Learning Problem:

$$\boldsymbol{x} \in \mathbb{R}^{100},\ y = f(\boldsymbol{x}) \coloneqq x_5$$

K-NN cannot learn that only one component is relevant. It will always use <u>all</u> 100 components to determine the k nearest neighbors of a query $\boldsymbol{x}$.

Distance-Weighted *k*-NN (1)

Might want weight nearer neighbors more heavily...

Discrete-valued target function:

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

real-valued target function:

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

where

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

and $d(x_q, x_i)$ is distance between x_q and x_i. Requires special handling for $d(x_q, x_i) = 0$. If $d(x_q, xi) = 0$ then $\hat{f}(x_q) = f(x_i)$ if there is only one such x_i, else vote among all such $x_i s$.

Now it makes sense to use ***all*** training examples instead of just k

➔ called Shepard's method (proposed 1968)

Inductive Bias:

The classification of an instance x_q will be most similar to the classification of other instances that are nearby in Euclidean space.

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 jth 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

see [Moore and Lee, 1994]

NOTE: (Pruned) Decision trees do not have that problem, because only one feature is selected at each node.

2. Locally Weighted Regression

Regression := approximating a real-valued target function

Classification := approximating a discrete-valued target function

Residual := $\hat{f}(x) - f(x)$ = error in approximating the target function

Kernel function := distance function used to determine weights of each training example:

$$w_i \equiv K(d(x_i, x_q))$$

Locally Weighted Regression

Note *k*-NN forms local approximation to f for each query point x_q

Why not form an explicit approximation $\hat{f}(x)$ for region surrounding x_q

- Fit linear function to *k* nearest neighbors: $\hat{f}(x) = \vec{\mathbf{w}} \cdot \vec{\mathbf{x}}$
- Fit quadratic, ...
- Produces “piecewise approximation” to *f*

Several choices of error to minimize:

- Squared error over *k* nearest neighbors $E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} \left(f(x) - \hat{f}(x)\right)^2$

- Distance-weighted squared error over all nbrs

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} \left(f(x) - \hat{f}(x)\right)^2 K\left(d(x_q, x)\right)$$

- Combine E_1 und E_2