Kernels

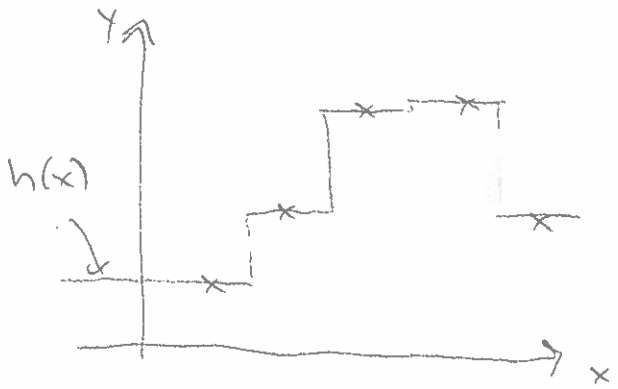
Background: local learning R&N 18.8
aka instance-based "
memory-based "
non-parametric "

Nearest-neighbor method

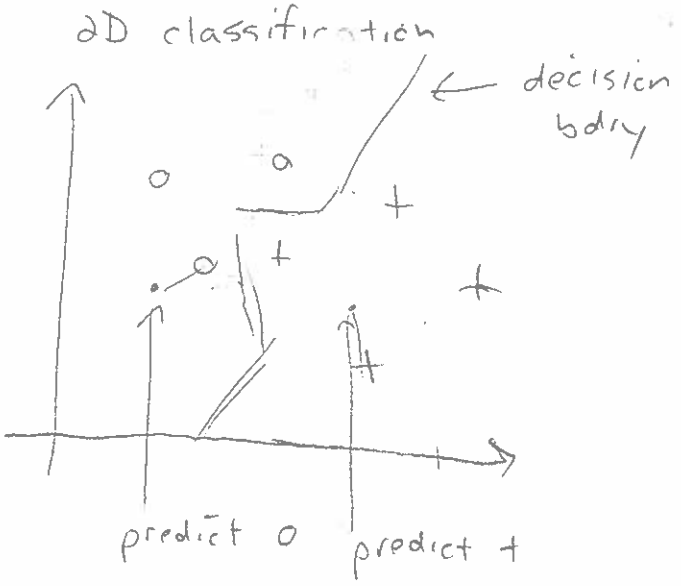
- Training examples $(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)$
- No training phase
- On input $\vec{x}$, predict output of closest training example to $\vec{x}$:

$$\begin{cases} j = \underset{i}{\operatorname{argmin}} \|x - x_i\|^2 \\ h(x) = y_j \end{cases}$$

E.g. 1D regression

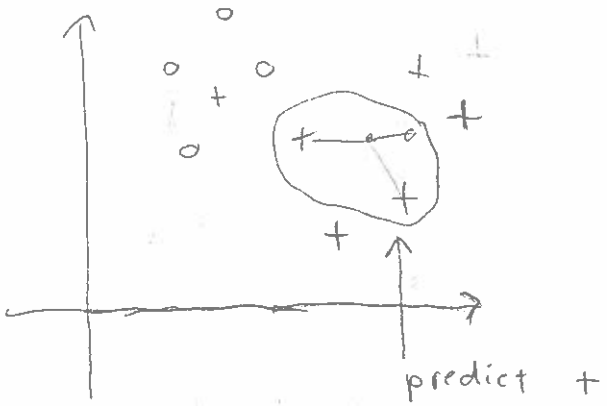


2D classification



Improvements

#1) Use K nearest neighbors instead of one (avoid overfitting); see R&N Fig. 18.26)



e.g. K=3

Take majority vote of 3 closest neighbors

Formally: on input $\vec{x}$

- Let $j_1, j_2, \ldots, j_k$ be set of k closest examples, (smallest distance $\|\vec{x} - \vec{x}_j\|$)

$$h(\vec{x}) = \frac{1}{k} \sum_{p=1}^{k} y_{j_p}$$ ← average over neighbors

- Classification

$h(\vec{x})$ = fraction of K neighbors with $y = 1$

predict $\begin{cases} 1 & h(x) \geq \frac{1}{2} \\ 0 & h(x) < \frac{1}{2} \end{cases}$

(Select K odd to avoid ties)

#2) Kernel regression (classification)

: KNN downsides:
- all neighbors weighted equally
- how to choose k?

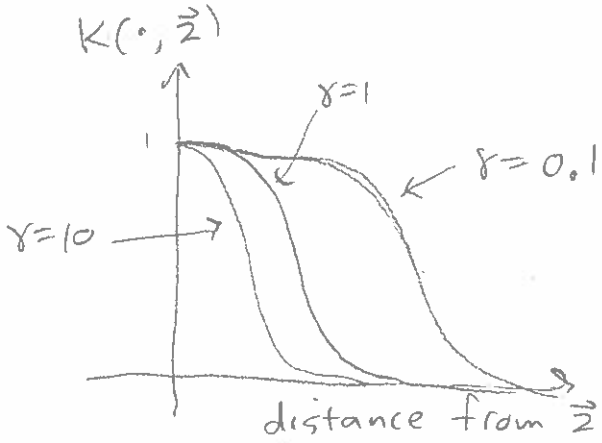
Idea: use all neighbors, weight by distance, using kernel function

$K(\vec{x}, \vec{z})$ ≡ similarity between $\vec{x}$ + $\vec{z}$

(later: view as a dot product)

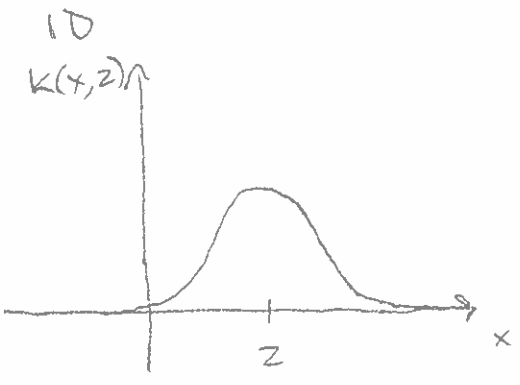
E.g. Gaussian kernel

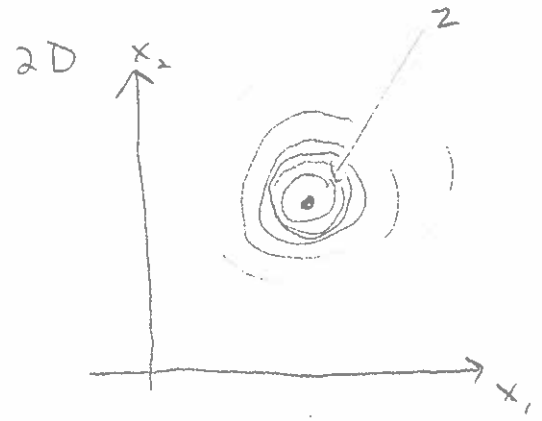
$$K(\vec{x}, \vec{z}) = \exp(-\gamma \|x - z\|^2)$$



- closer points are most similar
- exponential dropoff with distance

Similarity to a fixed point $\vec{z}$





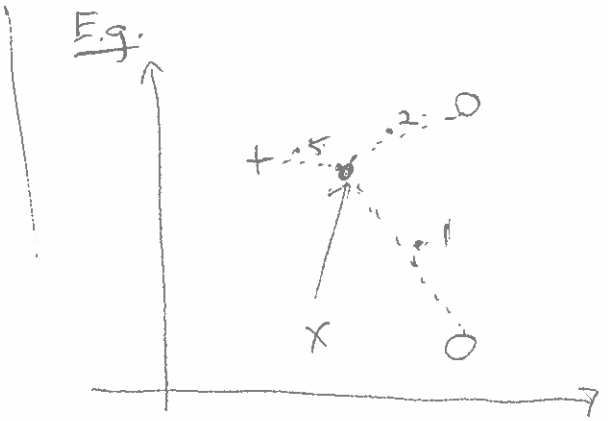
Kernel regression algorithm

$$h(\vec{x}) = \frac{\sum_{i=1}^{N} K(\vec{x}, \vec{x}_i) \cdot y_i}{\sum_{i=1}^{N} K(\vec{x}, \vec{x}_i)}$$

← weight of $\vec{x}_i$

← total weight

E.g.



$$h(x) = \frac{(0.5)(1) + (0.2)(0) + (0.1)(0)}{0.5 + 0.2 + 0.1}$$

$$= \frac{0.5}{0.8}$$

$$= \frac{5}{8} > \frac{1}{2}$$

$\Rightarrow$ predict '+'