Instance-based Learning

Machine Learning

Hamid R Rabiee – Zahra Dehghanian

Spring 2025

Sharif University
of Technology

Outline

- Non-parametric approach
  - Unsupervised: Non-parametric density estimation
    - Parzen Windows
    - Kn-Nearest Neighbor Density Estimation
  - Supervised: Instance-based learners
    - Classification
      - kNN classification
      - Weighted (or kernel) kNN
    - Regression
      - kNN regression
      - Locally linear weighted regression

Introduction

- Estimation of arbitrary density functions
  - Parametric density functions cannot usually fit the densities we encounter in practical problems
    - e.g., parametric densities are unimodal
  - Non-parametric methods don't assume that the model (from) of underlying densities is known in advance

- Non-parametric methods (for classification) can be categorized into
  - Generative
    - Estimate $p(\boldsymbol{x}|\mathcal{C}_i)$ from $\mathcal{D}_i$ using non-parametric density estimation
  - Discriminative
    - Estimate $p(\mathcal{C}_i|\boldsymbol{x})$ from $\mathcal{D}$

Parametric vs. nonparametric methods

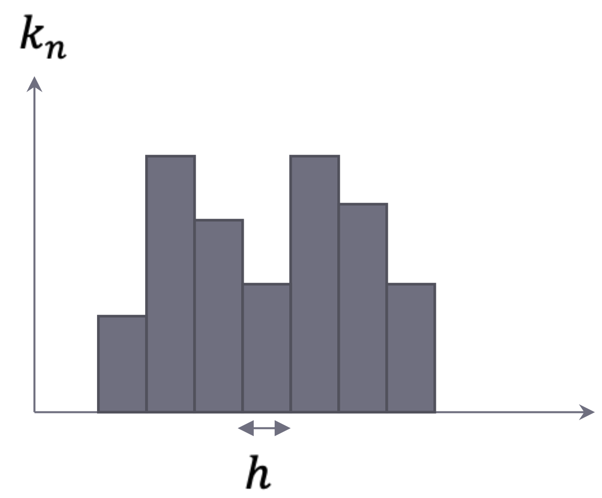
- Parametric methods need to find parameters from data and then use the inferred parameters to decide on new data points
  - Learning: finding parameters from data

- Nonparametric methods
  - Training examples are explicitly used
    - Training phase is not required

- Both supervised and unsupervised learning methods can be categorized into parametric and non-parametric methods.

Histogram approximation idea

- **Histogram** approximation of an unknown pdf
  - $P(b_l) \approx k_n(b_l)/n \quad l = 1, \dots, L$
    - $k_n(b_l)$: number of samples (among n ones) lied in the bin b_l



- The corresponding estimated pdf:
  - $\hat{p}(x) = \frac{P(b_l)}{h} \qquad \left|x - \bar{x}_{b_l}\right| \le \frac{h}{2}$

Mid-point of the bin b_l

Non-parametric density estimation

- Probability of falling in a region $\mathcal{R}$:
  - $P = \int_{\mathcal{R}} p(\boldsymbol{x}')d\boldsymbol{x}'$ (smoothed version of $p(\boldsymbol{x})$)

- $\mathcal{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{n}$: a set of samples drawn i.i.d. according to $p(\boldsymbol{x})$
  - The probability that k of the n samples fall in $\mathcal{R}$:
    - $P_k = \binom{n}{k} P^k (1-P)^{n-k}$
    - $E[k] = nP$

Non-parametric density estimation

- Probability of falling in a region $\mathcal{R}$:
  - $P = \int_{\mathcal{R}} p(\boldsymbol{x}')d\boldsymbol{x}'$ (smoothed version of $p(\boldsymbol{x})$)

- $\mathcal{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{n}$: a set of samples drawn i.i.d. according to $p(\boldsymbol{x})$
  - The probability that k of the n samples fall in $\mathcal{R}$:
    - $P_k = \binom{n}{k} P^k (1-P)^{n-k}$
    - $E[k] = nP$
  - This binomial distribution peaks sharply around the mean:
    - $k \approx nP \Rightarrow \frac{k}{n}$ as an estimate for P
    - More accurate for larger n

Non-parametric density estimation

- We can estimate smoothed $p(\boldsymbol{x})$ by estimating P:
- Assumptions: $p(\boldsymbol{x})$ is continuous and the region $\mathcal{R}$ enclosing $\boldsymbol{x}$ is so small that p is near constant in it:

$$P = \int_{\mathcal{R}} p(\boldsymbol{x}')d\boldsymbol{x}' = p(\boldsymbol{x}) \times V$$

$$V = Vol(\mathcal{R})$$

$$\boldsymbol{x} \in \mathcal{R} \Rightarrow p(\boldsymbol{x}) = \frac{P}{V} \approx \frac{k/n}{V}$$

- Let V approach zero if we want to find $p(\boldsymbol{x})$ instead of the averaged version.

Necessary conditions for converge

- $p_n(\boldsymbol{x})$ is the estimate of $p(\boldsymbol{x})$ using n samples:
  - V_n: the volume of region around $\boldsymbol{x}$
  - k_n: the number of samples falling in the region

$$p_n(\boldsymbol{x}) = \frac{k_n/n}{V_n}$$

- Necessary conditions for converge of $p_n(\boldsymbol{x})$ to $p(\boldsymbol{x})$:
  - $\lim_{n\to\infty} V_n = 0$
  - $\lim_{n\to\infty} k_n = \infty$
  - $\lim_{n\to\infty} k_n/n = 0$

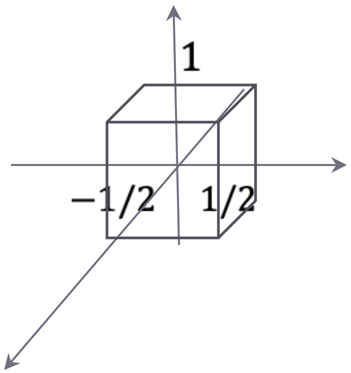
Non-parametric density estimation: Main approaches

- Two approaches of satisfying conditions:
  - Kernel density estimator (Parzen window): fix V and determine K from the data
    - Number of points falling inside the volume can vary from point to point
    - converges to the true probability density in the limit $n \rightarrow \infty$ when V shrinks suitably with n (e.g., $V_n = V_1/\sqrt{n}$)
  - k-nearest neighbor density estimator: fix k and determine the value of V from the data
    - Volume grows until it contains k neighbors of x
    - converges to the true probability density in the limit $n \rightarrow \infty$ when k grows with n (e.g., $k_n = k_1\sqrt{n}$)

Parzen window

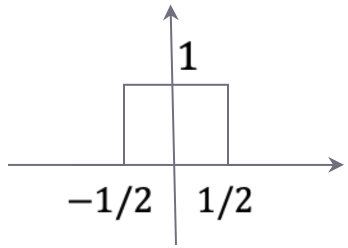
- Extension of histogram idea:
  - Hyper-cubes with length of side h (i.e., volume h^d) are located on the samples

- Hypercube as a simple window function:



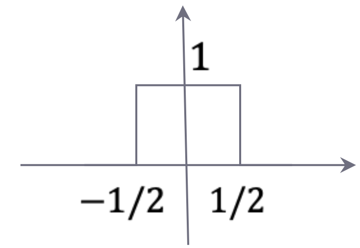
$$\varphi(\boldsymbol{u}) = \begin{cases} 1 & (|u_1| \le \frac{1}{2} \wedge \ldots \wedge |u_d| \le \frac{1}{2}) \\ 0 & o.w. \end{cases}$$

$$p_n(x) = \frac{1}{n} \times \frac{1}{h_n^d} \sum_{i=1}^{n} \varphi\left(\frac{\boldsymbol{x} - \boldsymbol{x}^{(i)}}{h_n}\right)$$



Parzen window

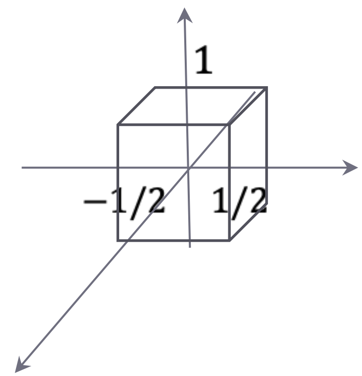
- Extension of histogram idea:
  - Hyper-cubes with length of side h (i.e., volume h^d) are located on the samples



- Hypercube as a simple window function:

$$\varphi(\boldsymbol{u}) = \begin{cases} 1 & (|u_1| \leq \frac{1}{2} \wedge \ldots \wedge |u_d| \leq \frac{1}{2}) \\ 0 & o.w. \end{cases}$$

$$p_n(x) = \frac{1}{n} \times \frac{1}{h_n^d} \sum_{i=1}^{n} \varphi\left(\frac{\boldsymbol{x} - \boldsymbol{x}^{(i)}}{h_n}\right)$$



- $p_n(\boldsymbol{x}) = \dfrac{k_n}{nV_n}$
  - $k_n = \sum_{i=1}^{n} \varphi\left(\frac{\boldsymbol{x}-\boldsymbol{x}^{(i)}}{h_n}\right) \longrightarrow$ number of samples in the hypercube around $\boldsymbol{x}$
  - $V_n = (h_n)^d$

Window function

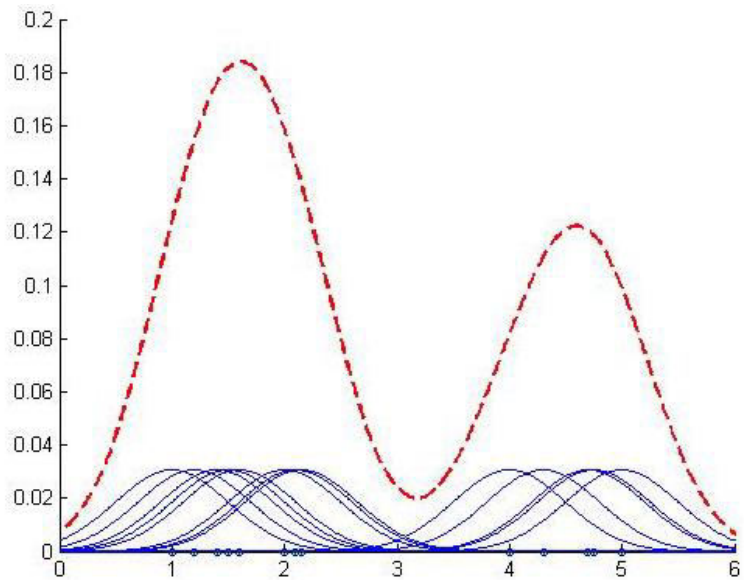
- Necessary conditions for window function to find legitimate density function:
  - $\varphi(\boldsymbol{x}) \geq 0$
  - $\int \varphi(\boldsymbol{x}) d\boldsymbol{x} = 1$

- Windows are also called **kernels** or potential functions.

Density estimation: non-parametric

$$\hat{p}_n(x) = \frac{1}{n}\sum_{i=1}^{n} \boxed{N(x|x^{(i)}, h^2)} \rightarrow \frac{1}{\sqrt{2\pi}h} e^{-\frac{\left(x-x^{(i)}\right)^2}{2h^2}}$$

1 1.2 1.4 1.5 1.6 2 2.1 2.15 4 4.3 4.7 4.75 5



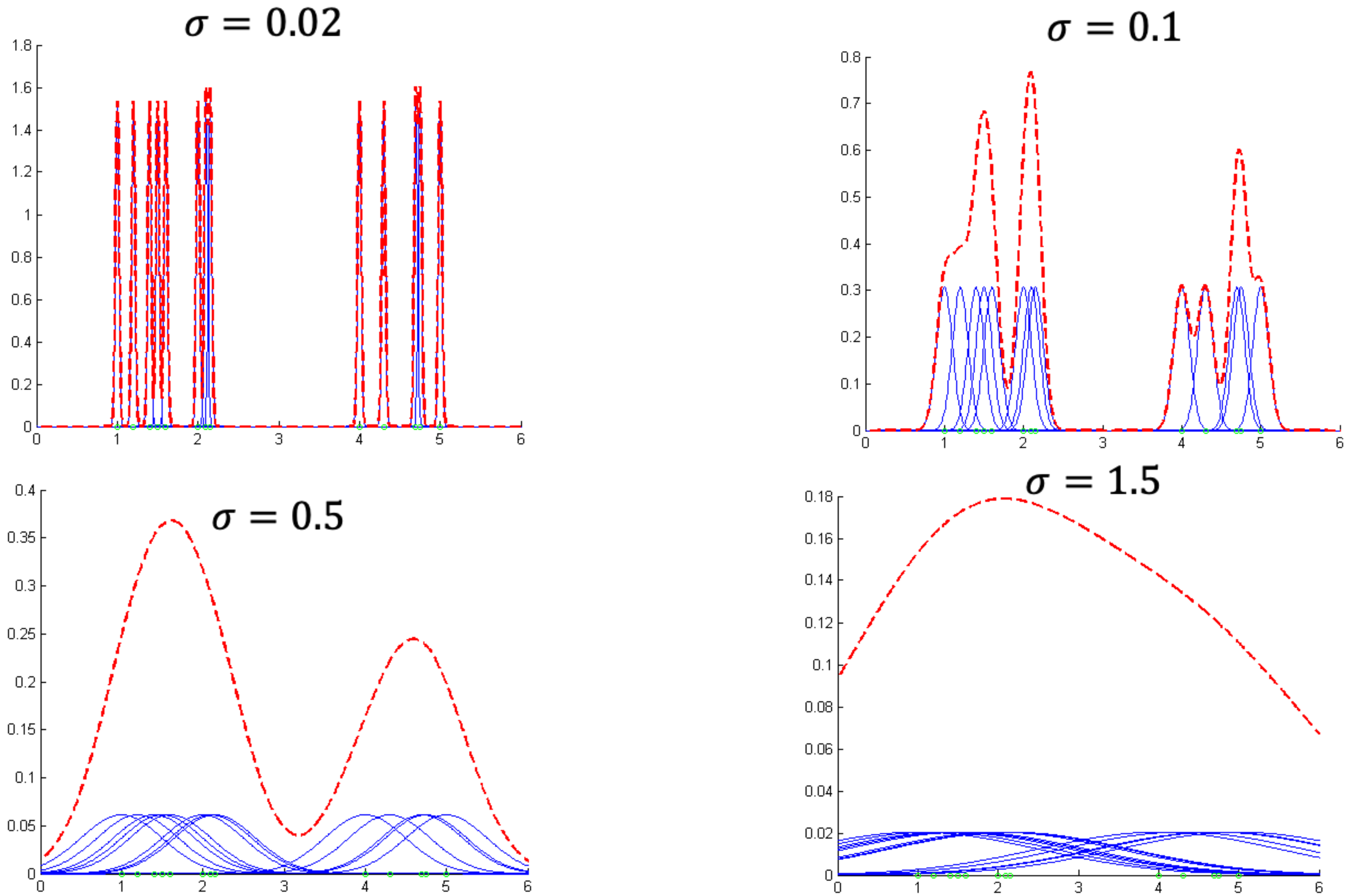
$$\sigma = h$$

$$\hat{p}(x) = \frac{1}{n}\sum_{i=1}^{n} N(x|x^{(i)}, \sigma^2)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(x-x^{(i)}\right)^2}{2\sigma^2}}$$

Choice of σ is crucial.

Density estimation: non-parametric

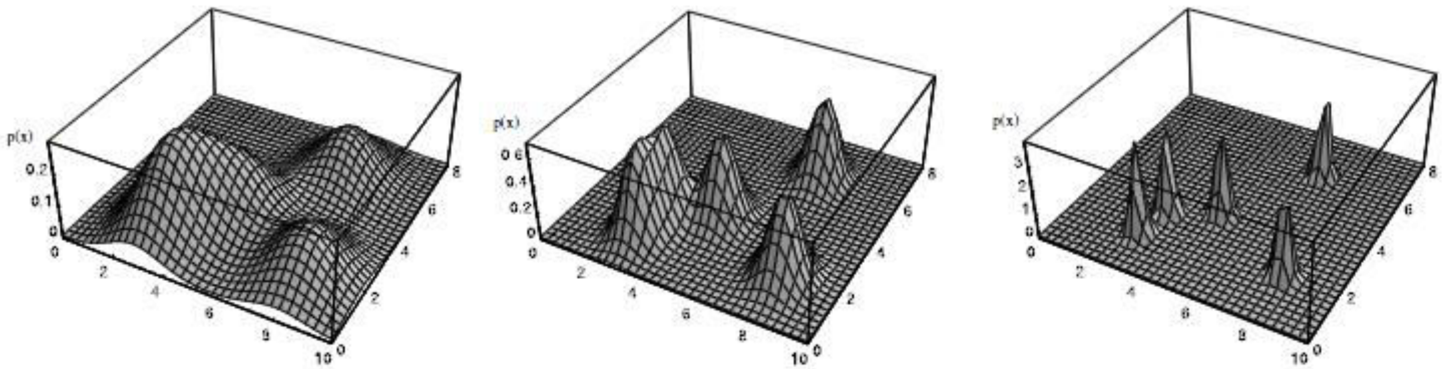


Window (or kernel) function: Width parameter

- $$p_n(x) = \frac{1}{n} \times \frac{1}{h_n^d} \sum_{i=1}^{n} \varphi\left(\frac{\boldsymbol{x} - \boldsymbol{x}^{(i)}}{h_n}\right)$$

- Choosing h_n:
  - Too large: low resolution
  - Too small: much variability

[Duda, Hurt, and Stork]



- For unlimited n, by letting V_n slowly approach zero as n increases $p_n(\boldsymbol{x})$ converges to $p(\boldsymbol{x})$

Width parameter

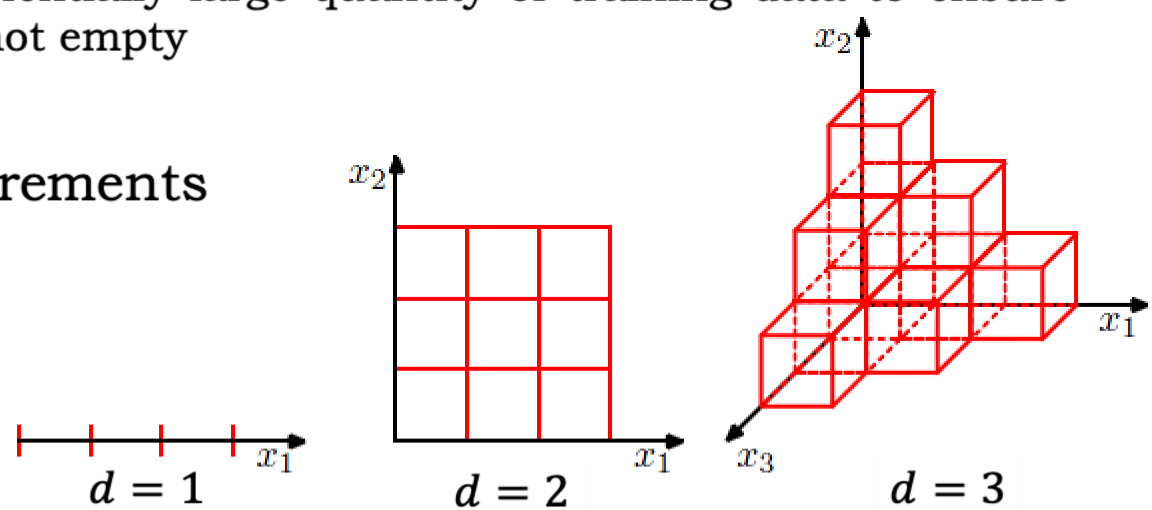
- For fixed n, a smaller h results in higher variance while a larger h leads to higher bias.

- For a fixed h, the variance decreases as the number of sample points n tends to infinity
  - for a large enough number of samples, the smaller h the better the accuracy of the resulting estimate

- In practice, where only a finite number of samples is possible, a compromise between h and n must be made.
  - h can be set using techniques like cross-validation where the density estimation used for learning tasks such as classification

Practical issues: Curse of dimensionality

- Large n is necessary to find an acceptable density estimation in high dimensional feature spaces
  - n must grow exponentially with the dimensionality d.
    - If n equidistant points are required to densely fill a one-dim interval, n^d points are needed to fill the corresponding d-dim hypercube.
      - We need an exponentially large quantity of training data to ensure that the cells are not empty



- Also complexity requirements

Non-parametric density estimation: Main approaches

- Two approaches of satisfying conditions:
  - Kernel density estimator (Parzen window): fix V and determine K from the data
    - Number of points falling inside the volume can vary from point to point
    - converges to the true probability density in the limit $n \to \infty$ when V shrinks suitably with n (e.g., $V_n = V_1/\sqrt{n}$)

  - k-nearest neighbor density estimator: fix k and determine the value of V from the data
    - Volume grows until it contains k neighbors of $\boldsymbol{x}$
    - converges to the true probability density in the limit $n \to \infty$ when k grows with n (e.g., $k_n = k_1\sqrt{n}$)

k_n-nearest neighbor estimation

- Cell volume is a function of the point location
  - To estimate $p(\boldsymbol{x})$, let the cell around $\boldsymbol{x}$ grow until it captures k_n samples called k_n nearest neighbors of $\boldsymbol{x}$.

- Two possibilities can occur:
  - high density near $\boldsymbol{x} \Rightarrow$ cell will be small which provides a good resolution
  - low density near $\boldsymbol{x} \Rightarrow$ cell will grow large and stop until higher density regions are reached

k_n-nearest neighbor estimation

- $$p_n(\boldsymbol{x}) = \frac{k_n/n}{V_n} \Rightarrow V_n \approx \frac{1}{p(\boldsymbol{x})} \times k_n/n$$

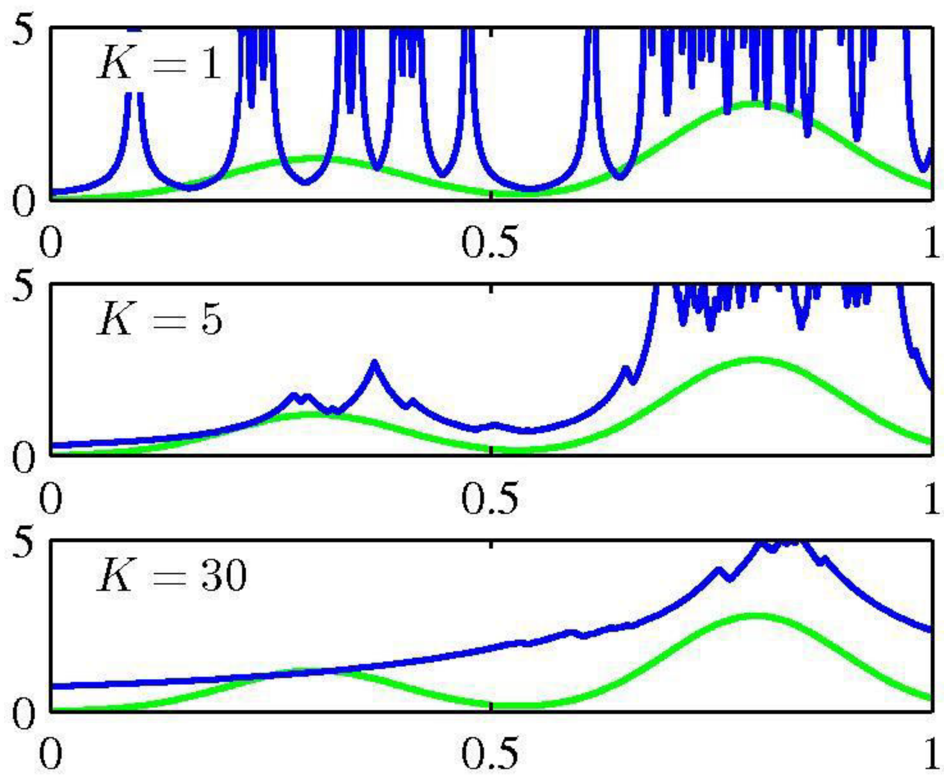
- A family of estimates by setting $k_n = k_1\sqrt{n}$ and choosing different values for k_1:

$$p_n(\boldsymbol{x}) = \frac{k_n/n}{V_n} \Rightarrow V_n \approx \frac{1/p(\boldsymbol{x})}{\sqrt{n}} \qquad k_1 = 1$$

V_n is a function of $\boldsymbol{x}$

k_n-Nearest Neighbor Estimation: Example

- Discontinuities in the slopes

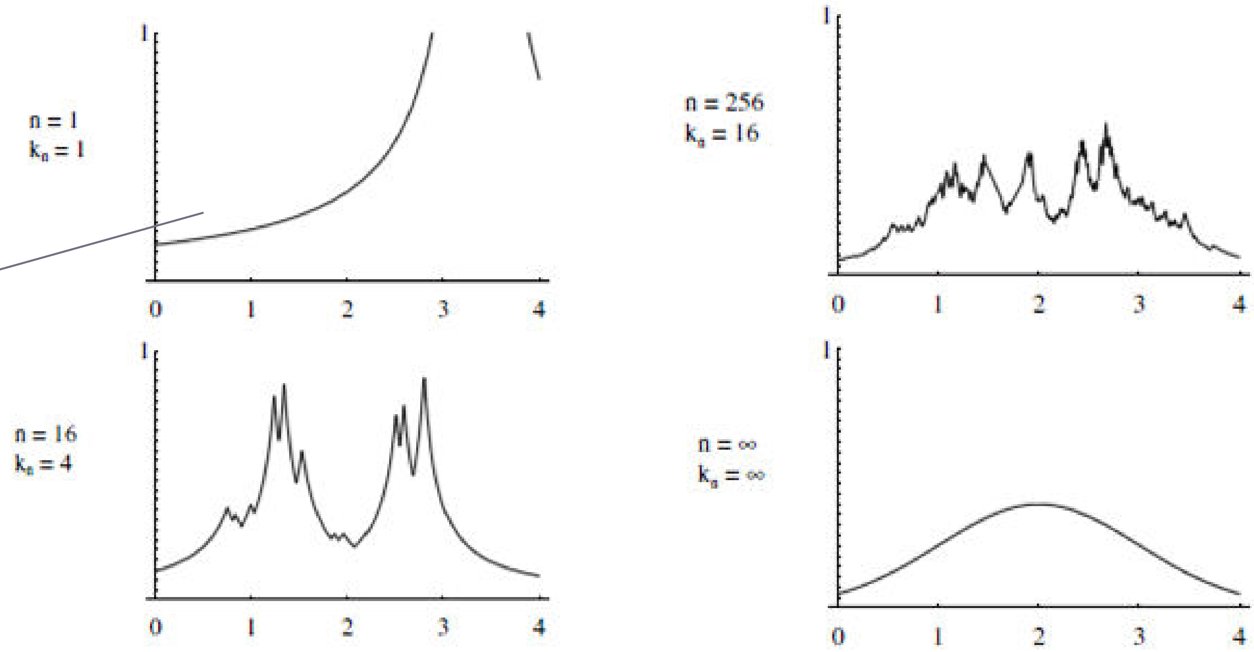


[Bishop]

k_n-Nearest Neighbor Estimation: Example

- $k_n = \sqrt{n}$

$$p_1(x) = \frac{1}{2|x - x^{(1)}|}$$



[Duda, Hurt, and Stork]

Non-parametric density estimation: Summary

- Generality of distributions
  - With enough samples, convergence to an arbitrarily complicated target density can be obtained.

- The number of required samples must be very large to assure convergence
  - grows exponentially with the dimensionality of the feature space

- These methods are very sensitive to the choice of window width or number of nearest neighbors

- There may be severe requirements for computation time and storage (needed to save all training samples).
  - 'training' phase simply requires storage of the training set
  - computational cost of evaluating $p(\boldsymbol{x})$ grows linearly with the size of the data set

Nonparametric learners

- **Memory-based** or **instance-based** learners
  - lazy learning: (almost) all the work is done at the test time.

- Generic description:
  - Memorize training $(\boldsymbol{x}^{(1)}, y^{(1)}), \dots, (\boldsymbol{x}^{(n)}, y^{(n)})$.
  - Given test $\boldsymbol{x}$ predict: $\hat{y} = f(\boldsymbol{x}; \boldsymbol{x}^{(1)}, y^{(1)}, \dots, \boldsymbol{x}^{(n)}, y^{(n)})$.

- f is typically expressed in terms of the similarity of the test sample $\boldsymbol{x}$ to the training samples $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}$

Parzen window & generative classification

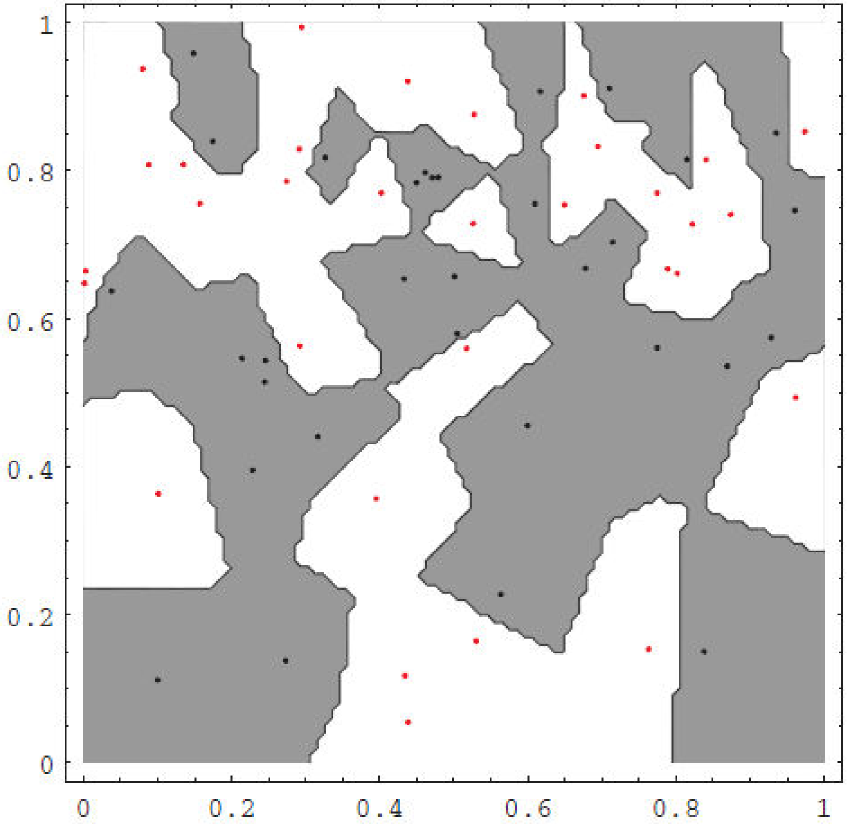
- If $\dfrac{\frac{1}{n_1}\times\frac{1}{h^d}\sum_{x^{(i)}\in\mathcal{D}_1}\varphi\left(\frac{x-x^{(i)}}{h}\right)}{\frac{1}{n_2}\times\frac{1}{h^d}\sum_{x^{(i)}\in\mathcal{D}_2}\varphi\left(\frac{x-x^{(i)}}{h}\right)}\times\dfrac{P(\mathcal{C}_1)}{P(\mathcal{C}_2)} > 1$ decide $\mathcal{C}_1$

  otherwise decide $\mathcal{C}_2$

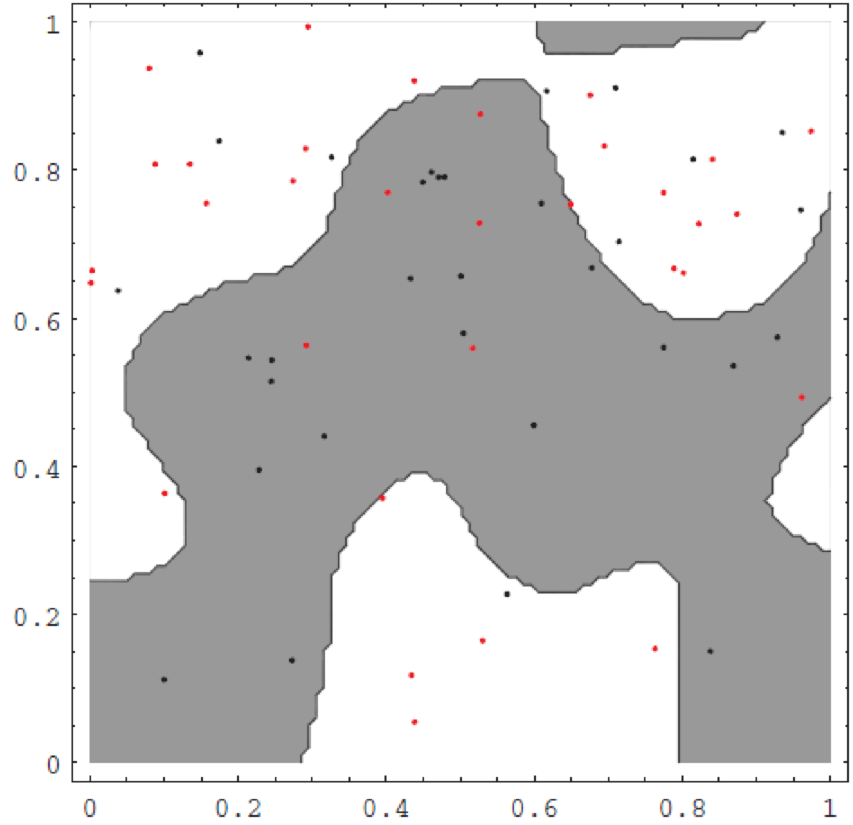
  - $n_j = |\mathcal{D}_j|$ $(j = 1,2)$: number of training samples in class $\mathcal{C}_j$
    - $\mathcal{D}_j$: set of training samples labels as $\mathcal{C}_j$

- For large n, it needs both high time and memory requirements

Parzen window & generative classification: Example



Smaller h



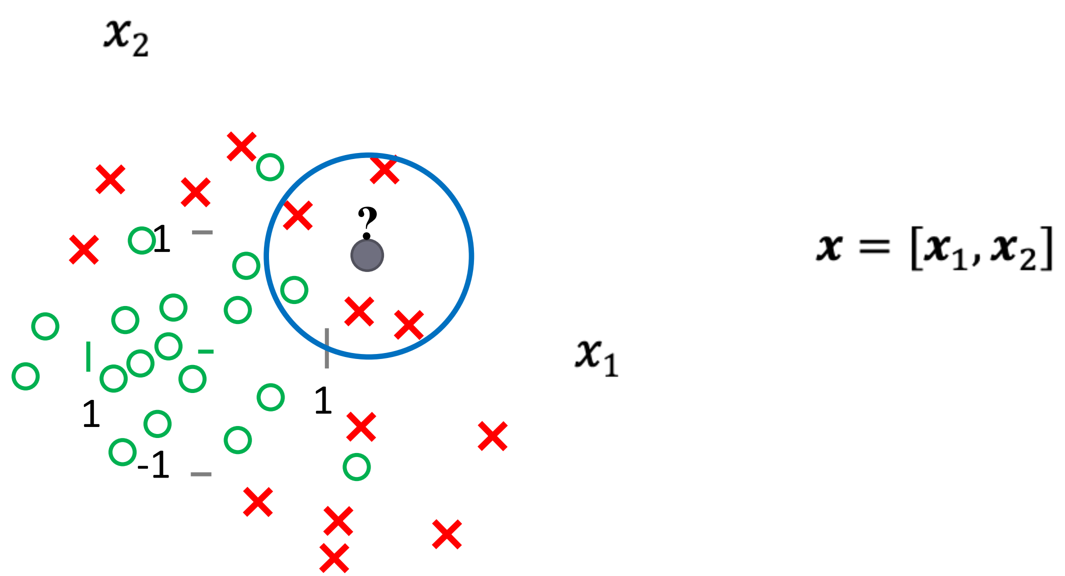
larger h

[Duda, Hurt, and Stork]

Classification: k-Nearest-Neighbor (kNN)

- k-NN classifier: $k > 1$ nearest neighbors
  - Label for $\boldsymbol{x}$ predicted by majority voting among its k-NN.

- $k = 5$



- What is the effect of k?

kNN classifier

- Given
  - Training data $\{(\boldsymbol{x}^{(1)}, y^{(1)}), \dots, (\boldsymbol{x}^{(n)}, y^{(n)})\}$ are simply stored.
  - Test sample: $\boldsymbol{x}$

- To classify $\boldsymbol{x}$:
  - Find k nearest training samples to $\boldsymbol{x}$
  - Out of these k samples, identify the number of samples k_j belonging to class $\mathcal{C}_j$ ($j = 1, \dots, C$).
  - Assign $\boldsymbol{x}$ to the class $\mathcal{C}_{j^*}$ where $j^* = \underset{j=1,\dots,c}{\operatorname{argmax}}\, k_j$

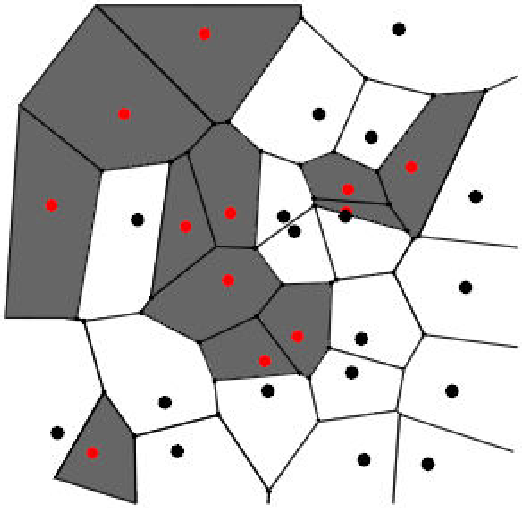
- It can be considered as a discriminative method.

Probabilistic perspective of kNN

- kNN as a discriminative nonparametric classifier
  - Non-parametric density estimation for $P(C_j|\boldsymbol{x})$
    - $P(C_j|\boldsymbol{x}) \approx \frac{k_j}{k}$ where k_j shows the number of training samples among k nearest neighbors of $\boldsymbol{x}$ that are labeled C_j
  - Bayes decision rule for assigning labels

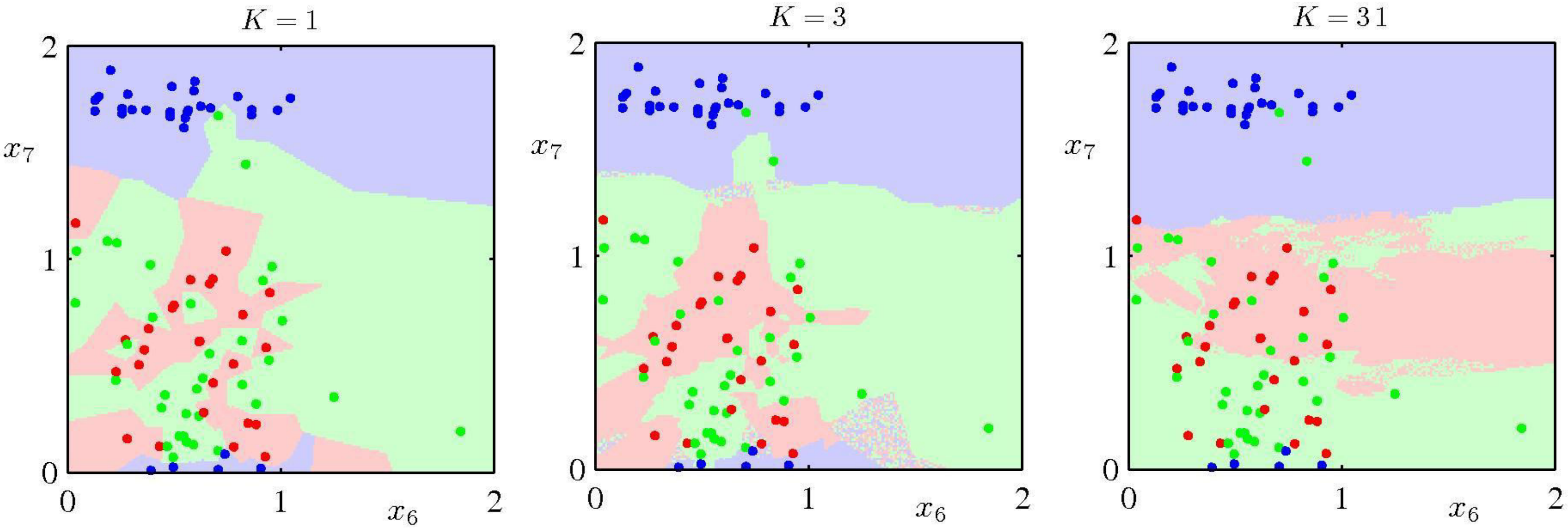
Nearest-neighbor classifier: Example

- Voronoi tessellation:
  - Each cell consists of all points closer to a given training point than to any other training points
    - All points in a cell are labeled by the category of the corresponding training point.



[Duda, Hurt, and Strok's Book]

kNN classifier: Effect of k



[Bishop]

Need to determine an appropriate value for k (e.g., by crossvalidation)

Nearest neighbor classifier: error bound

- Nearest-Neighbor: kNN with $k = 1$
  - Decision rule: $\hat{y} = y^{NN(\boldsymbol{x})}$ where $NN(\boldsymbol{x}) = \underset{i=1,\dots,N}{\operatorname{argmin}} \left\| \boldsymbol{x} - \boldsymbol{x}^{(i)} \right\|$

- Cover & Hart 67: asymptotic risk of NN classifier satisfies:

$$R^* \leq R_\infty^{NN} \leq 2R^*(1 - R^*) \leq 2R^*$$

R_n: expected risk of NN classifier with n training examples drawn from $p(\boldsymbol{x}, y)$

$$R_\infty^{NN} = \lim_{n \to \infty} R_n^{NN}$$

R^*: the optimal Bayes risk

k-NN classifier: error bound

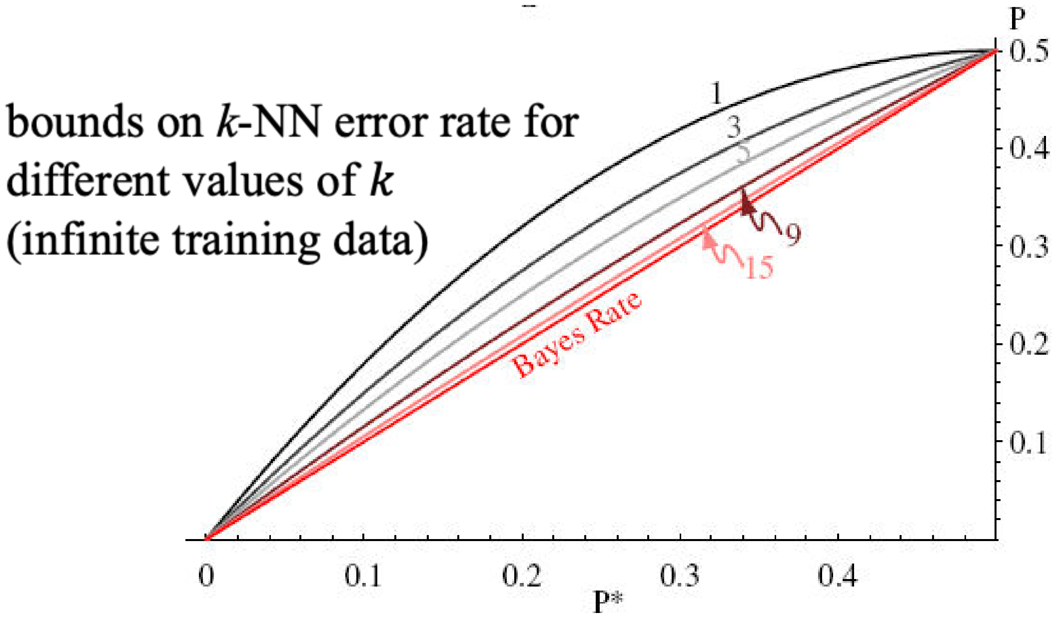
- Devr 96: the asymptotic risk of the kNN classifier $R_\infty = \lim_{n\to\infty} R_n$ satisfies

$$R^* \le R_\infty^{kNN} \le R^* + \sqrt{\frac{2R_\infty^{NN}}{k}}$$

where R^* is the optimal Bayes risk.

- As k increases, the upper bounds get closer to the lower bound (the Bayes Error rate).
  - When $k \to \infty$, the two bounds meet and k-NN rule becomes optimal.

Bound on the k-Nearest Neighbor Error Rate

- 

[Duda, Hurt, and Strok]

- As k increases, the upper bounds get closer to the lower bound (the Bayes Error rate).
  - When $k \to \infty$, the two bounds meet and k-NN rule becomes optimal.

Instance-based learner

- Main things to construct an instance-based learner:
  - A distance metric
  - Number of nearest neighbors of the test data that we look at
  - A weighting function (optional)
  - How to find the output based on neighbors?

Distance measures

- Euclidean distance

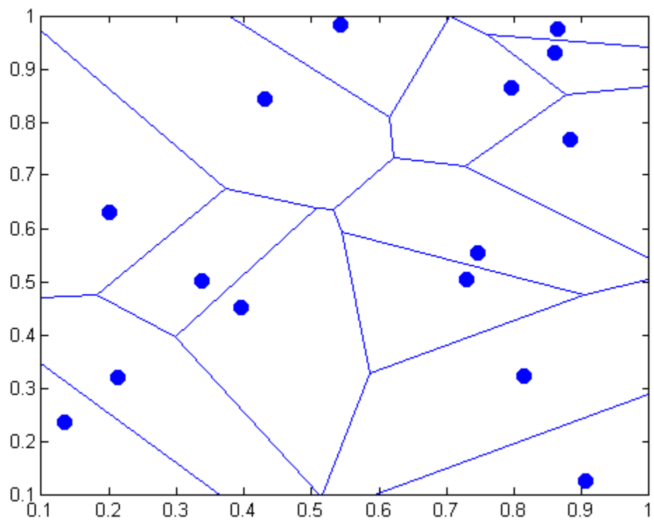
$$d(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2} = \sqrt{(x_1 - x_1')^2 + \cdots + (x_d - x_d')^2}$$

Sensitive to irrelevant features

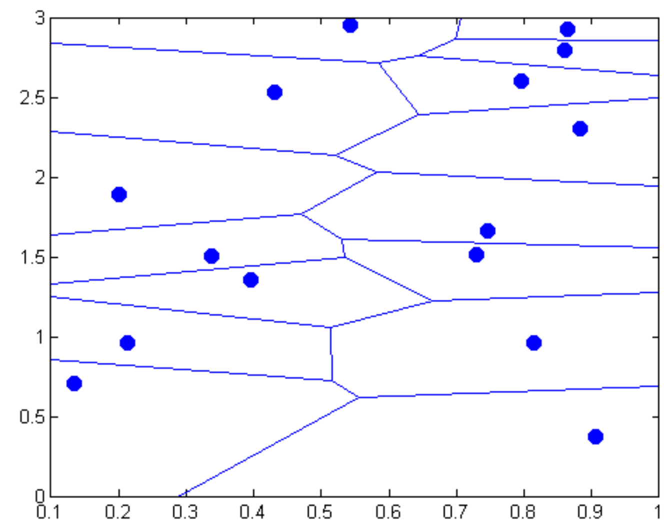
- Distance learning methods for this purpose
  - Weighted Euclidean distance
    - $d_w(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{w_1(x_1 - x_1')^2 + \cdots + w_d(x_d - x_d')^2}$
  - Mahalanobis distance
    - $d_A(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}')}$

- Other distances:
  - Hamming, angle, …
  - $L_p(\boldsymbol{x}, \boldsymbol{x}') = \sqrt[p]{\sum_{i=1}^{d}(x_i - x_i')^p}$

Distance measure: example

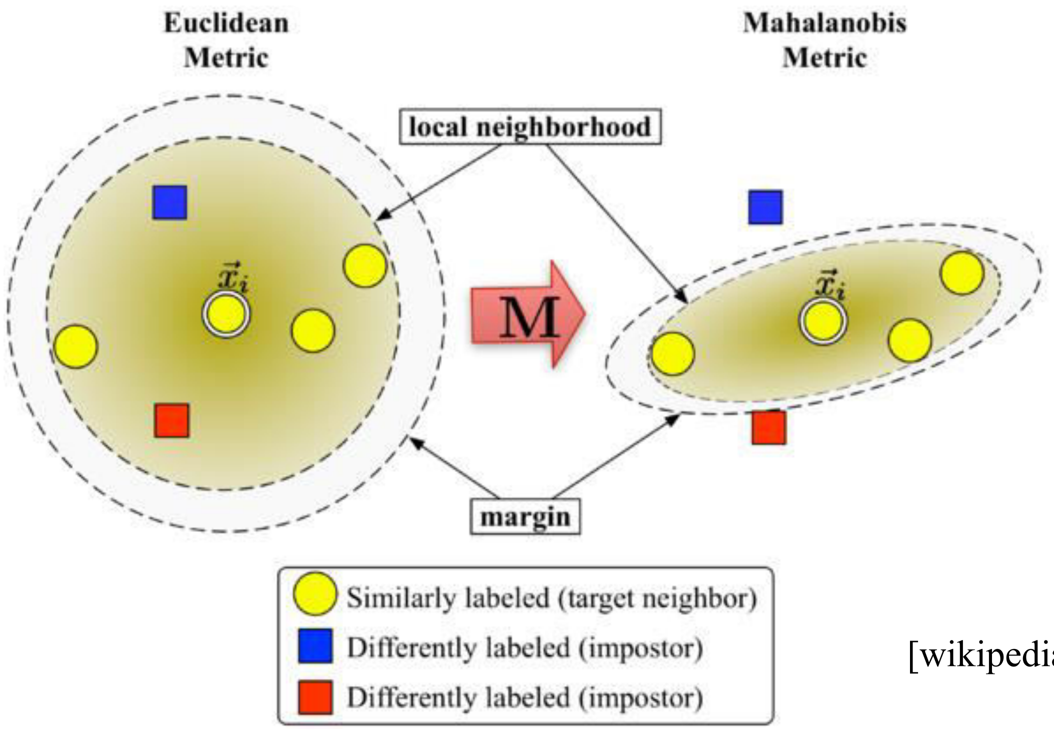


$$d(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2}$$



$$d(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(x_1 - x_1')^2 + 3(x_2 - x_2')^2}$$

Metric Learning Example: LMNN



$$\min_{\mathbf{M}} \sum_{i,j \in N_i} d(\vec{x}_i, \vec{x}_j) + \sum_{i,j,l} \xi_{ijl}$$

$$\forall_{i,j \in N_i, l, y_l \neq y_i}$$

$$d(\vec{x}_i, \vec{x}_j) + 1 \leq d(\vec{x}_i, \vec{x}_l) + \xi_{ijl}$$

$$\xi_{ijl} \geq 0$$

$$\mathbf{M} \succeq 0$$

[wikipedia]

$$d^2(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{M} (\boldsymbol{x} - \boldsymbol{x}')$$

Weighted kNN classification

- Weight nearer neighbors more heavily:

$$\hat{y} = f(\boldsymbol{x}) = \underset{c=1,\dots,C}{\operatorname{argmax}} \sum_{j \in N_k(\boldsymbol{x})} w_j(\boldsymbol{x}) \times I(c = y^{(j)})$$

$$w_j(\boldsymbol{x}) = \frac{1}{\|\boldsymbol{x} - \boldsymbol{x}^{(j)}\|^2}$$

An example of weighting function

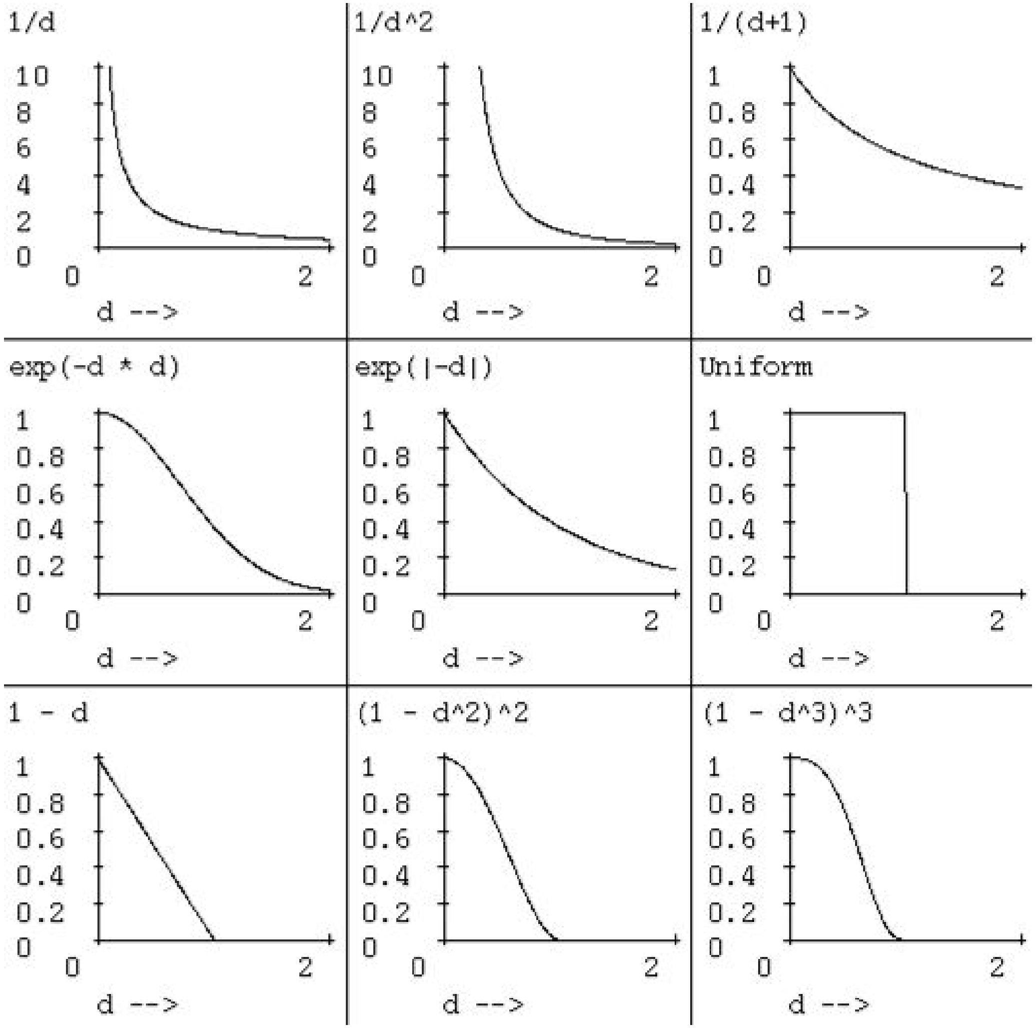
- In the weighted kNN, we can use all training examples instead of just k (Stepard's method):

$$\hat{y} = f(\boldsymbol{x}) = \underset{c=1,\dots,C}{\operatorname{argmax}} \sum_{j=1}^{n} w_j(\boldsymbol{x}) \times I(c = y^{(j)})$$

- Weights can be found using a kernel function $w_j(\boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{x}^{(j)})$:
  - e.g., $K(\boldsymbol{x}, \boldsymbol{x}^{(j)}) = e^{-\frac{d(\boldsymbol{x}, \boldsymbol{x}^{(j)})}{\sigma^2}}$

Weighting functions

$$d = d(\boldsymbol{x}, \boldsymbol{x}')$$



[Fig. has been adopted from Andrew Moore's tutorial on "Instance-based learning"]

kNN regression

- Simplest k-NN regression:
  - Let $\boldsymbol{x}'^{(1)}, \dots, \boldsymbol{x}'^{(k)}$ be the k nearest neighbors of $\boldsymbol{x}$ and $y'^{(1)}, \dots, y'^{(k)}$ be their labels.

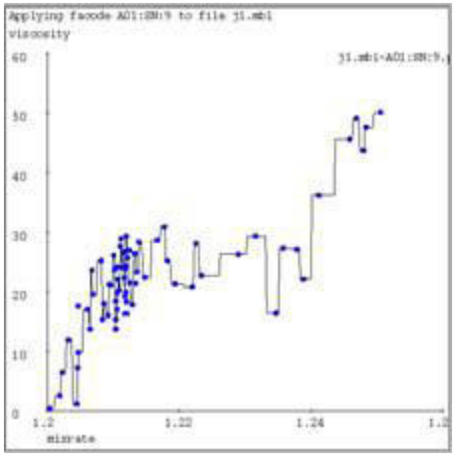
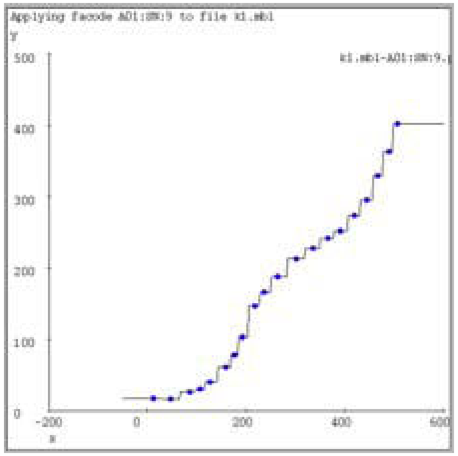
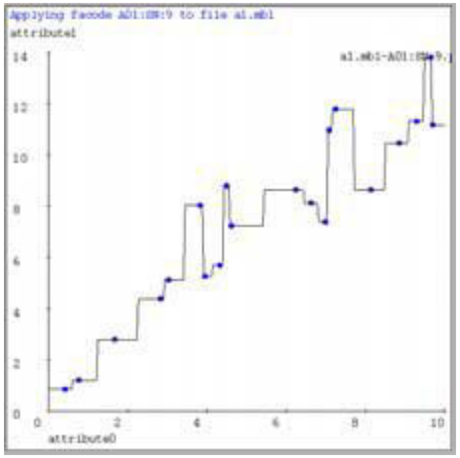
$$\hat{y} = \frac{1}{k} \sum_{j \in N_k(\boldsymbol{x})} y'^{(j)}$$

- Problems of kNN regression for fitting functions:
  - Problem 1: Discontinuities in the estimated function
    - Solution: Weighted (or kernel) regression
  - 1NN: noise-fitting problem
  - kNN ($k > 1$) smoothes away noise, but there are other deficiencies.
    - flats the ends

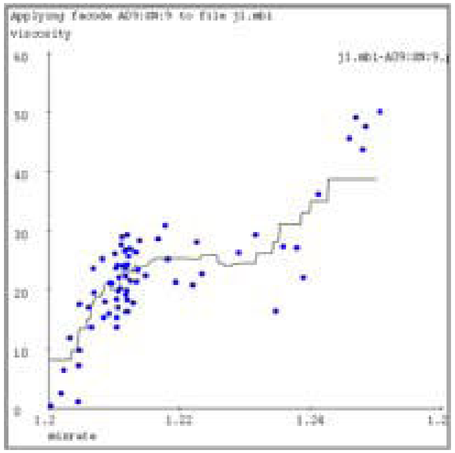
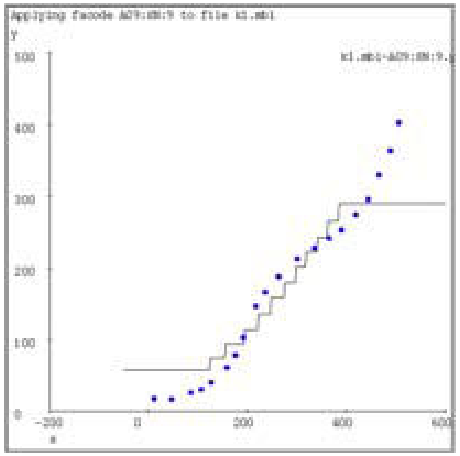
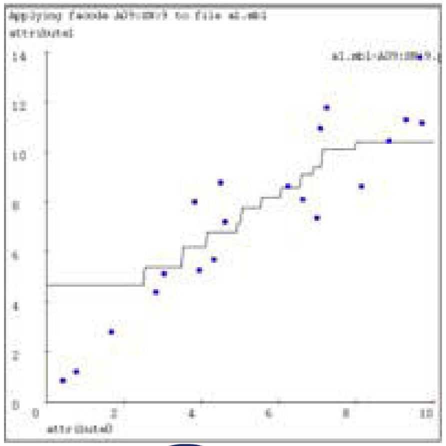
kNN regression: examples

Dataset 1 | Dataset 2 | Dataset 3

$k = 1$

$k = 9$

Weighted (or kernel) kNN regression

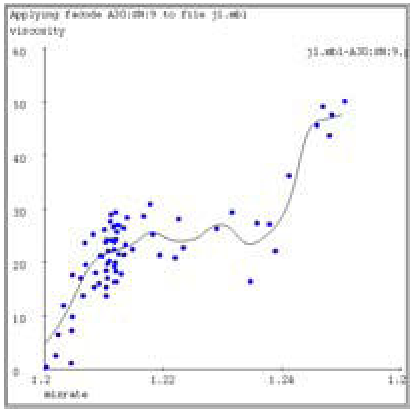
- Higher weights to nearer neighbors:

$$\hat{y} = f(\boldsymbol{x}) = \frac{\sum_{j \in N_k(\boldsymbol{x})} w_j(\boldsymbol{x}) \times y^{(j)}}{\sum_{j \in N_k(\boldsymbol{x})} w_j(\boldsymbol{x})}$$

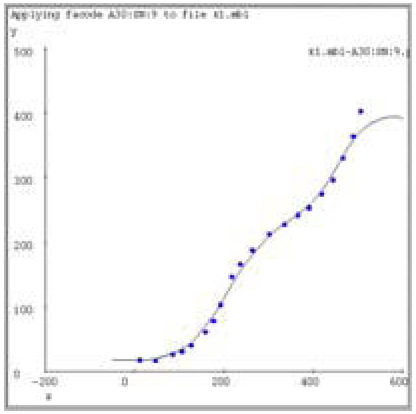
- In the weighted kNN regression, we can use all training examples instead of just k in the weighted form:

$$\hat{y} = f(\boldsymbol{x}) = \frac{\sum_{j=1}^{n} w_j(\boldsymbol{x}) \times y^{(j)}}{\sum_{j=1}^{n} w_j(\boldsymbol{x})}$$

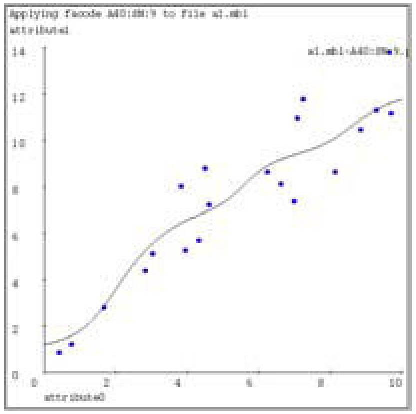
Kernel kNN regression



$\sigma = \frac{1}{32}$ of x-axis width



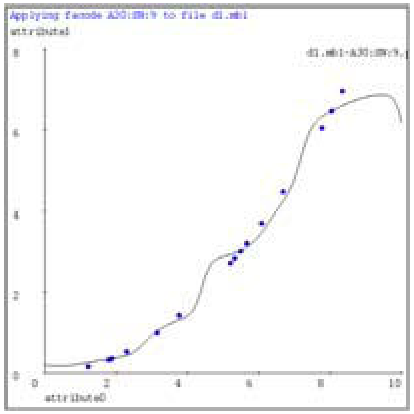
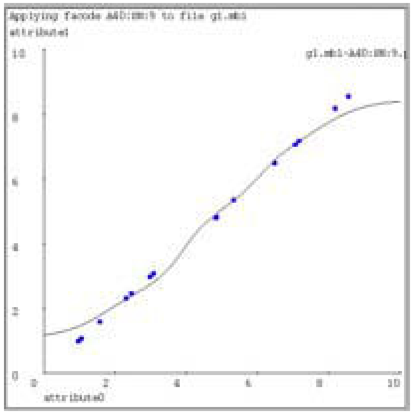
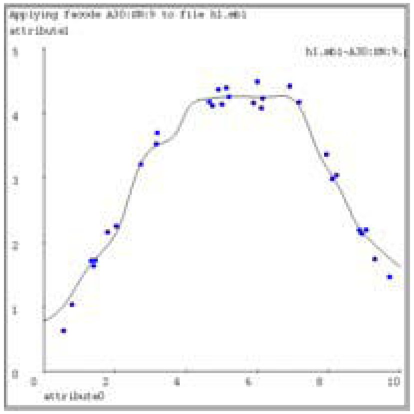
$\sigma = \frac{1}{32}$ of x-axis width



$\sigma = \frac{1}{16}$ of x-axis width

- Choosing a good parameter (kernel width) is important.

Kernel kNN regression

In these datasets, some regions are without samples
Best kernel widths have been used

- Disadvantages:
  - not capturing the simple structure of the data
  - failure to extrapolate at edges

[Figs. have been adopted from Andrew Moore's tutorial on "Instance-based learning"]

Locally weighted linear regression

- For each test sample, it produces linear approximation to the target function in a local region
- Instead of finding the output using weighted averaging (as in the kernel regression), we fit a parametric function locally:

$$\hat{y} = f\left(\boldsymbol{x}, \boldsymbol{x}^{(1)}, y^{(1)}, \dots, \boldsymbol{x}^{(n)}, y^{(n)}\right)$$

$$\hat{y} = f(\boldsymbol{x}; \boldsymbol{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d$$

$$J(\boldsymbol{w}) = \sum_{i \in N_k(\boldsymbol{x})} \left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)^2$$

Unweighted linear

$\boldsymbol{w}$ is found for each test sample

Locally weighted linear regression

- $$\hat{y} = f\left(\boldsymbol{x}, \boldsymbol{x}^{(1)}, y^{(1)}, \dots, \boldsymbol{x}^{(n)}, y^{(n)}\right)$$

$$J(\boldsymbol{w}(\boldsymbol{x})) = \sum_{i \in N_k(\boldsymbol{x})} \left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)^2$$

unweighted

e.g. $K\left(\boldsymbol{x}, \boldsymbol{x}^{(i)}\right) = e^{-\frac{\left\|\boldsymbol{x}-\boldsymbol{x}^{(i)}\right\|^2}{2\sigma^2}}$

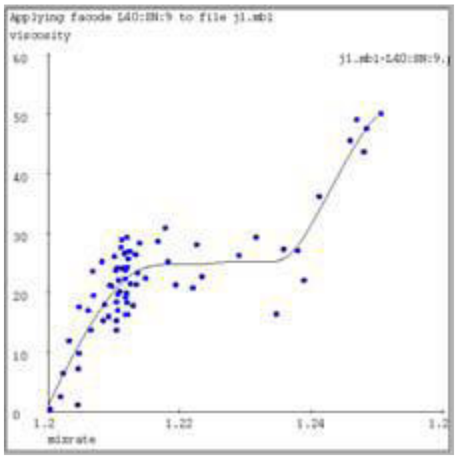
$$J(\boldsymbol{w}(\boldsymbol{x})) = \sum_{i \in N_k(\boldsymbol{x})} K\left(\boldsymbol{x}, \boldsymbol{x}^{(i)}\right)\left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)^2$$

weighted

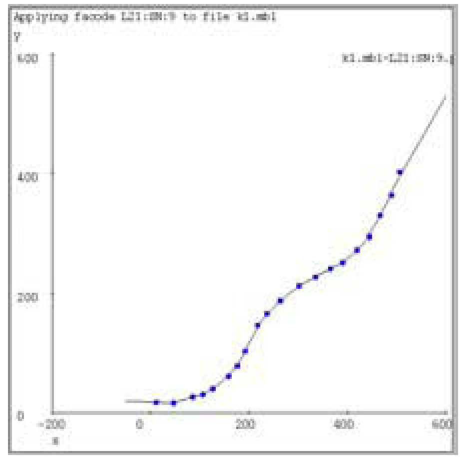
$$J(\boldsymbol{w}(\boldsymbol{x})) = \sum_{i=1}^{n} K\left(\boldsymbol{x}, \boldsymbol{x}^{(i)}\right)\left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)^2$$

Weighted on all training examples

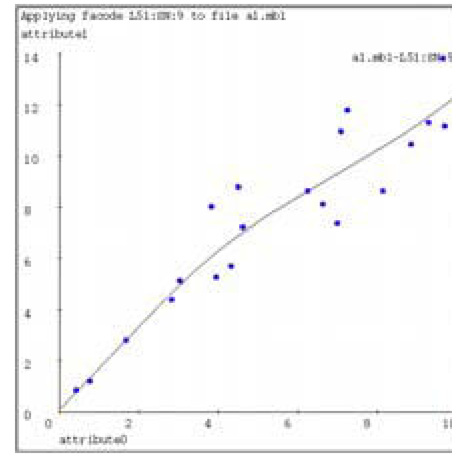
Locally weighted linear regression: example



$\sigma = \frac{1}{16}$ of x-axis width



$\sigma = \frac{1}{32}$ of x-axis width



$\sigma = \frac{1}{8}$ of x-axis width

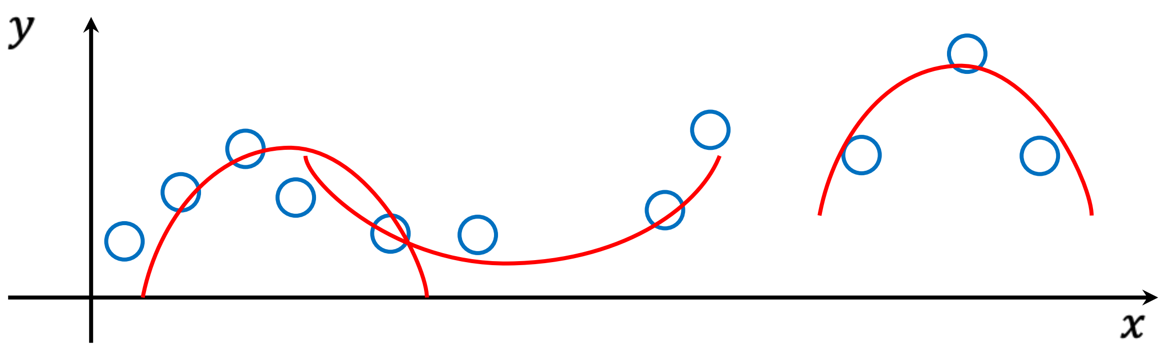
- More proper result than weighted kNN regression

Locally weighted regression: summary

- Idea 1: weighted kNN regression
  - using the weighted average on the output of $\boldsymbol{x}$'s neighbors (or on the outputs of all training data):

$$\hat{y} = \frac{\sum_{i=1}^{k} y'^{(i)} K(\boldsymbol{x}, \boldsymbol{x}'^{(i)})}{\sum_{j=1}^{k} K(\boldsymbol{x}, \boldsymbol{x}'^{(j)})}$$

- Idea 2: Locally weighted parametric regression
  - Fit a parametric model (e.g. linear function) to the neighbors of $\boldsymbol{x}$ (or on all training data).
  - Implicit assumption: the target function is reasonably smooth.



Parametric vs. nonparametric methods

- Is SVM classifier parametric?

$$\hat{y} = \text{sign}(w_0 + \sum_{\alpha_i > 0} \alpha_i y^{(i)} K(\boldsymbol{x}, \boldsymbol{x}^{(i)}))$$

  - In general, we can not summarize it in a simple parametric form.
    - Need to keep around support vectors (possibly all of the training data).
  - However, α_i are kind of parameters that are found in the training phase

Instance-based learning: summary

- Learning is just storing the training data
  - prediction on a new data based on the training data themselves

- An instance-based learner does not rely on assumption concerning the structure of the underlying density function.

- With large datasets, instance-based methods are slow for prediction on the test data
  - kd-tree, Locally Sensitive Hashing (LSH), and other kNN approximations can help.

Reference

- T. Mitchell, “Machine Learning”, 1998. [Chapter 8]
- C.M. Bishop, “Pattern Recognition and Machine Intelligence”, Section 2.5.