COMP 652: Machine Learning

Lecture 10

Today: Instance-based learning

- Non-parametric supervised learning
  - k-Nearest neighbor
  - Weighted nearest neighbor
- Parametric-Nonparametric hybrids
  - Locally weighted regression
  - Radial basis functions
- Non-parametric density estimation

Recall: Parametric supervised learning

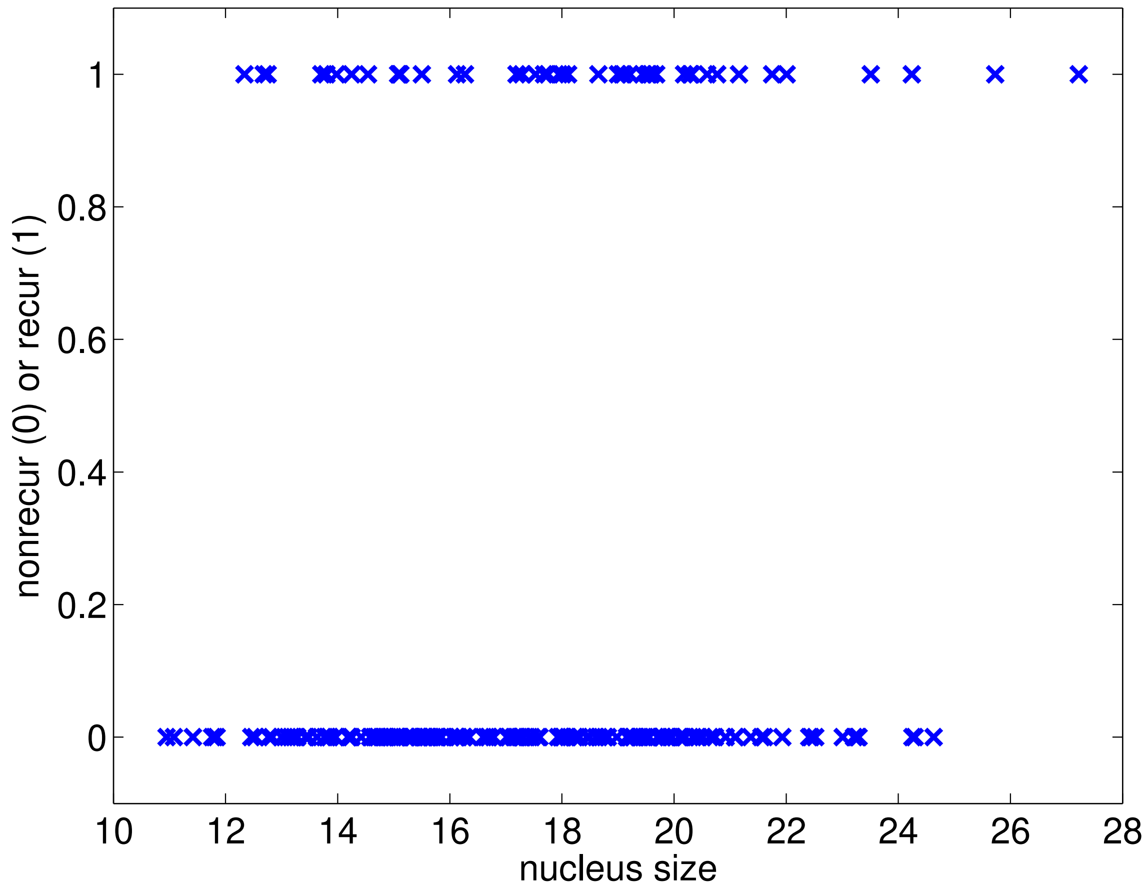
- So far, we have assumed that we have a data set D of labeled examples for supervised learning
- From this, we learn a **parameter vector** or **weights**
  - Discriminative learning optimizes the parameters so that some measure of error is minimized
  - Generative learning uses density estimation to fit the parameters
- These methods are called **parametric**, and their main goal is to summarize the data using the parameters
- Parametric methods are typically **global**, i.e. have one set of parameters for the entire data space, and often very good at approximating **smooth** functions

- But what if we just "remembered" the data?
- To classify a new input vector, just compare it to input vectors in the data set.

Non-parametric learning methods

- Key idea: just store all training examples $\langle \mathbf{x_i}, y_i \rangle$
  (Don't summarize them into weights / parameters.)
- When a query is made, compute the value of the new instance based on the values of the *closest (most similar)* points
- Prediction is based on the y values for those points
- There are different ways of evaluating the similarity, and different ways of computing the resulting value.

Wisconsin data Revisited



We want to predict presence or absence of tumor based on tumor size

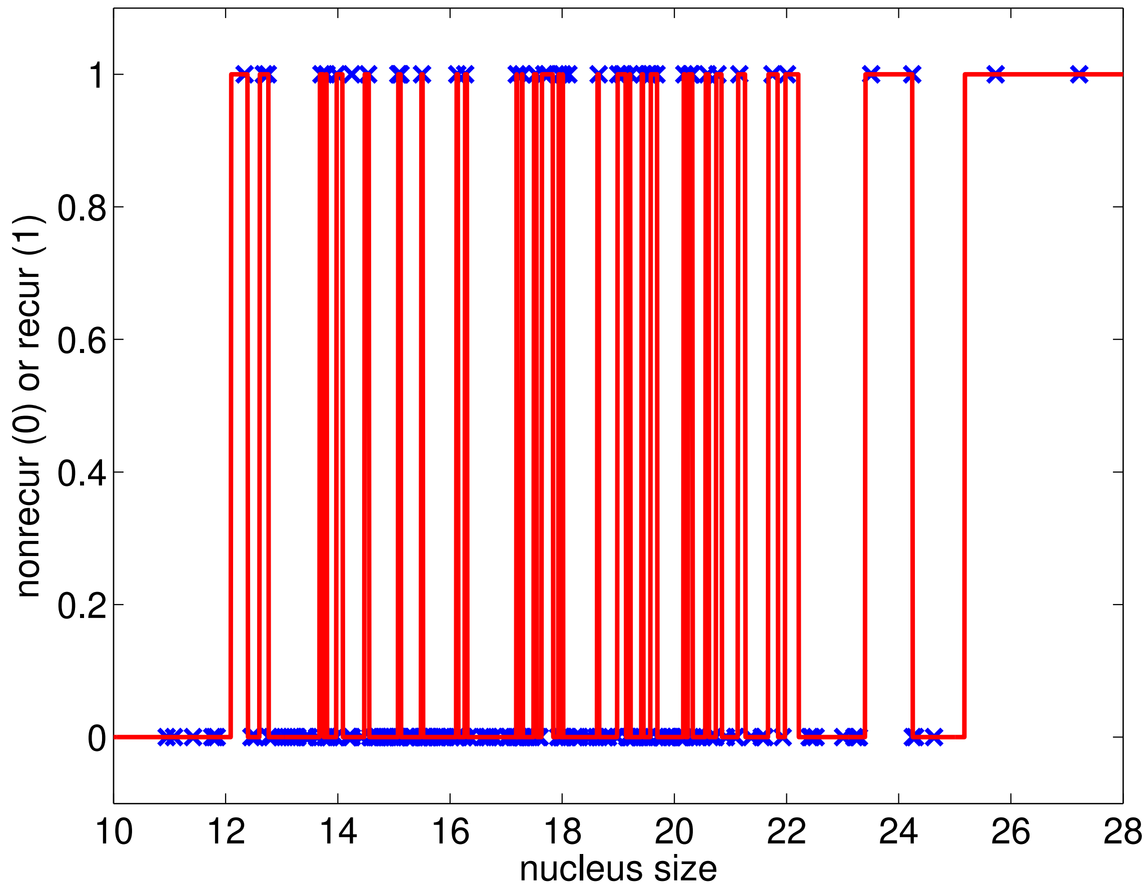
[Single] nearest neighbor

- Given: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, distance metric d on $\mathcal{X}$.
- Learning: Nothing to do!
- Prediction: for $\mathbf{x} \in \mathcal{X}$
  - Find nearest training sample to $\mathbf{x}$.

$$i \in \arg\min_i d(\mathbf{x}_i, \mathbf{x})$$

  - Predict $y = y_i$.

Wisconsin data predicted by nearest neighbor

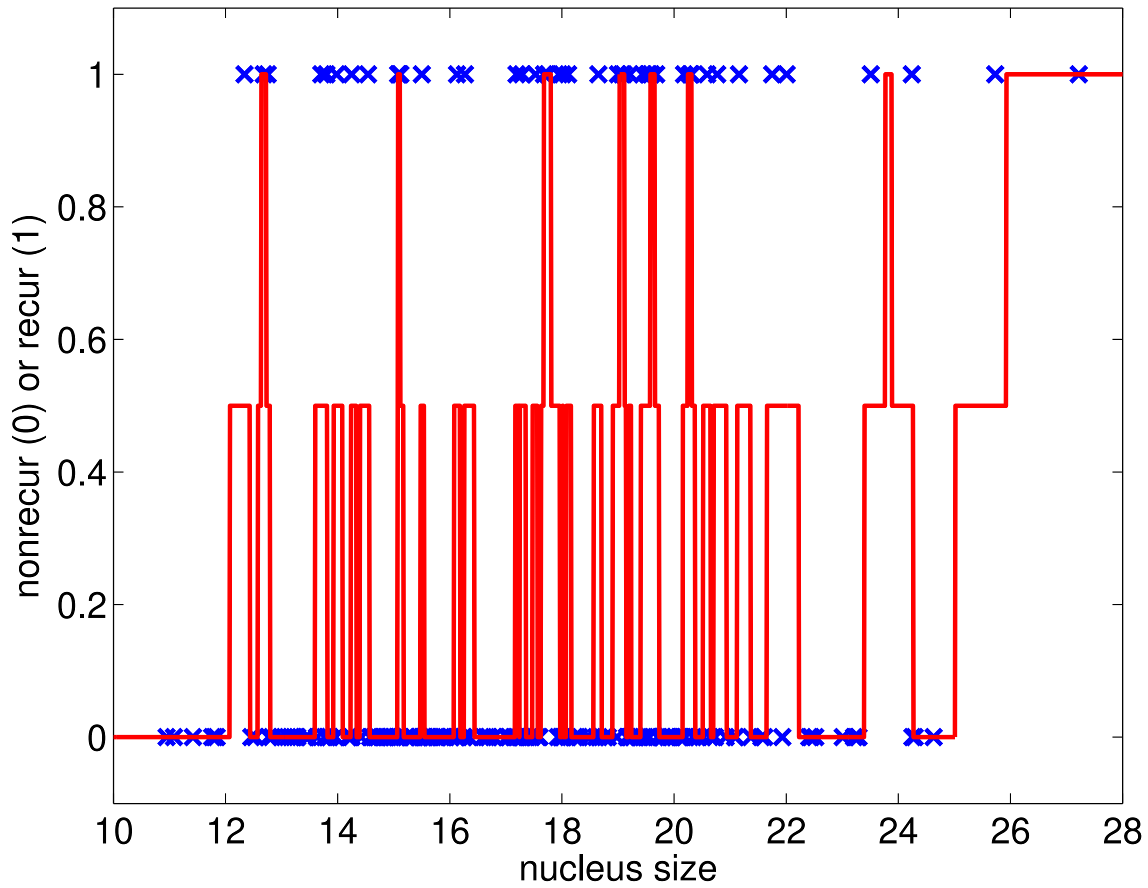


- Euclidean distance as similarity metric.
- Predictions very irregular.

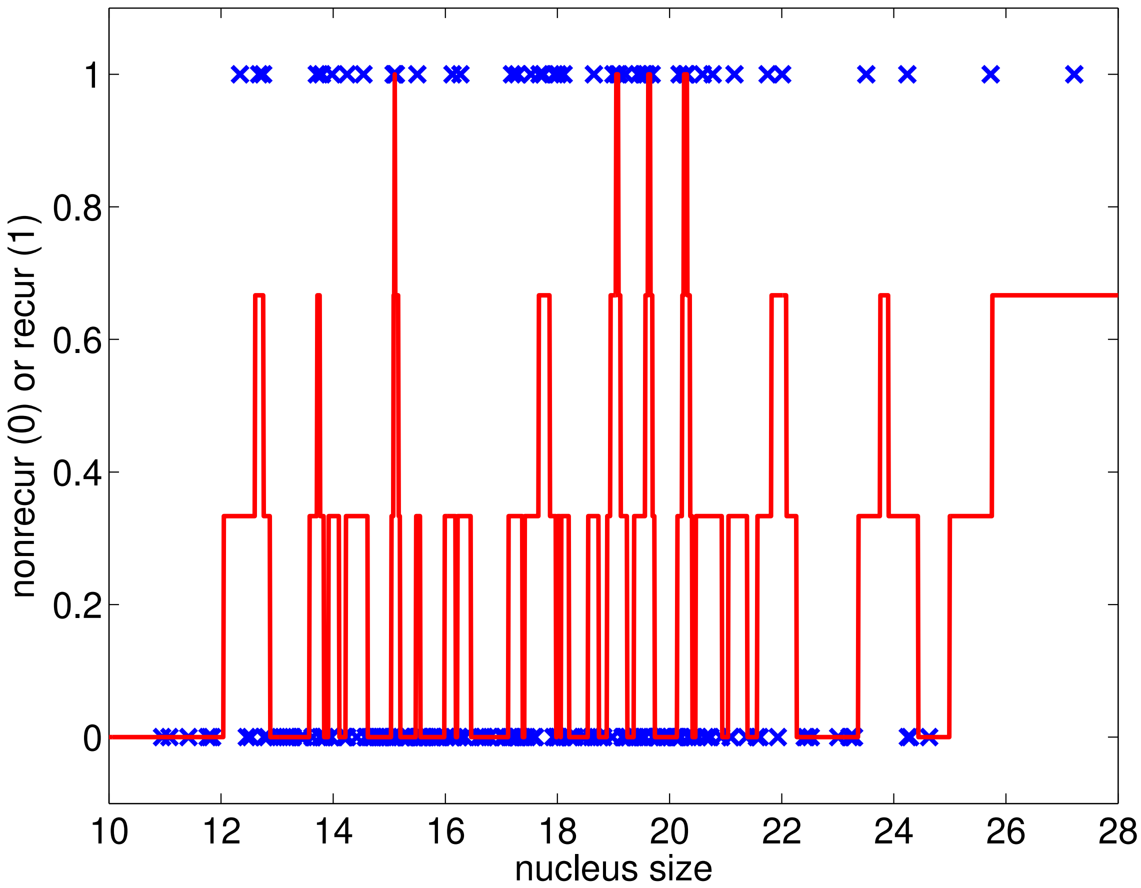
k-nearest neighbor

- Given: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$, distance metric d on $\mathcal{X}$.
- Learning: Nothing to do!
- Prediction: for $\mathbf{x} \in \mathcal{X}$
  - Find the k nearest training samples to $\mathbf{x}$.
    Let their indeces be $i_1, i_2, \ldots, i_k$.
  - Predict
    - $y =$ majority of $\{y_{i_1}, y_{i_2}, \ldots, y_{i_k}\}$ for classification, or empirical probability of each possible class
    - $y =$ mean/median of $\{y_{i_1}, y_{i_2}, \ldots, y_{i_k}\}$ for regression

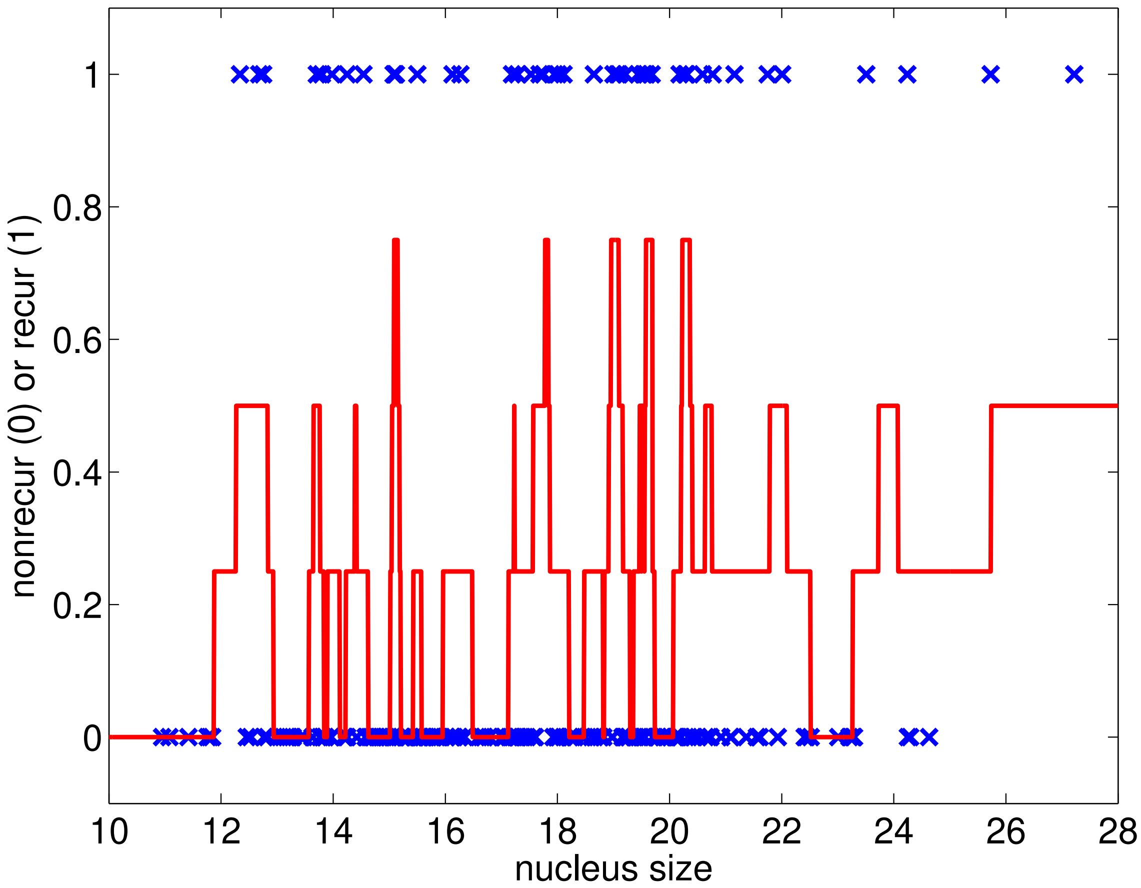
2-nearest neighbor, empirical probability



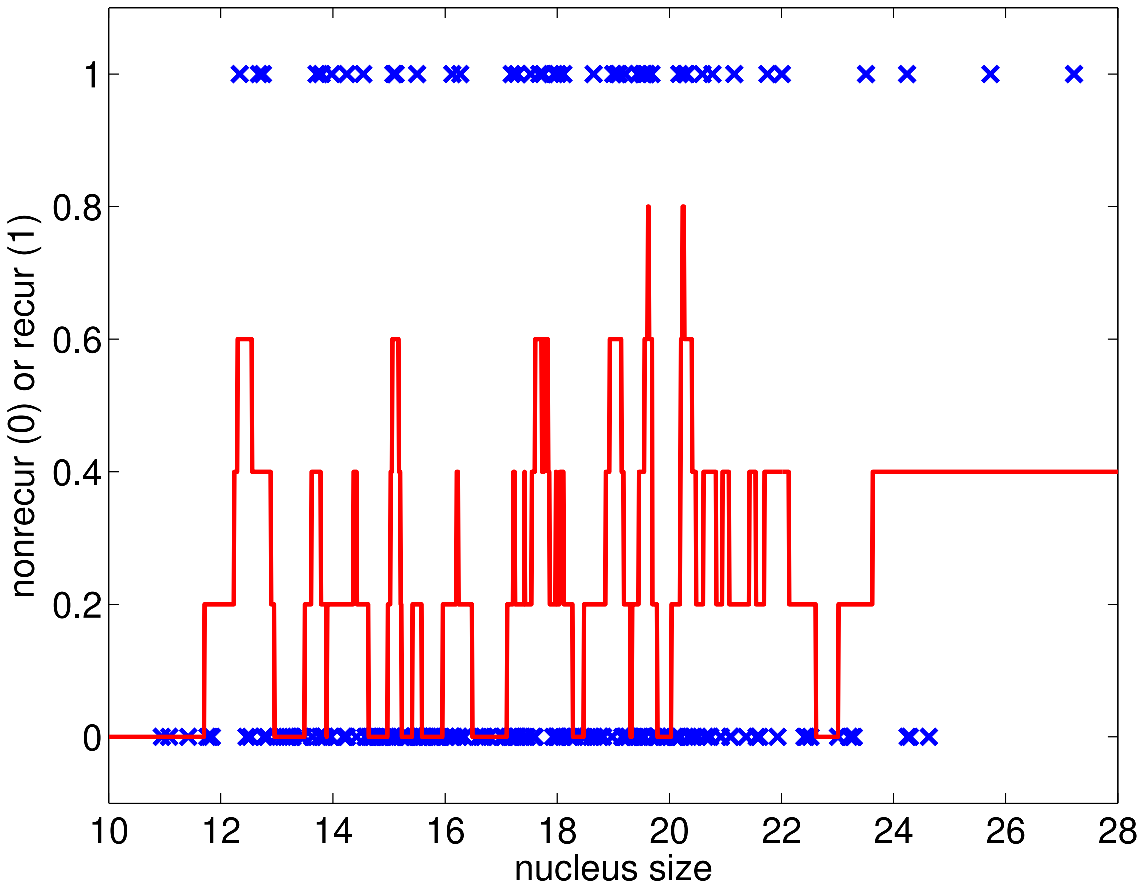
3-nearest neighbor



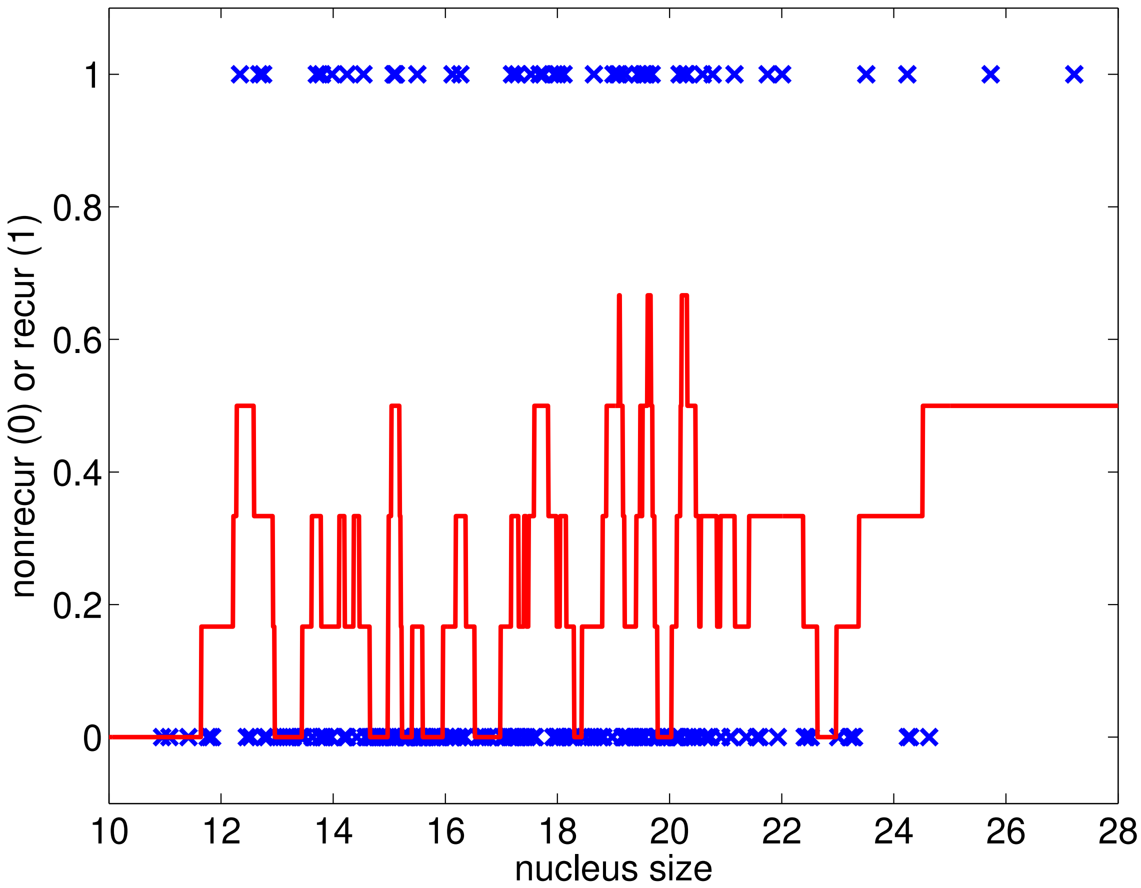
4-nearest neighbor



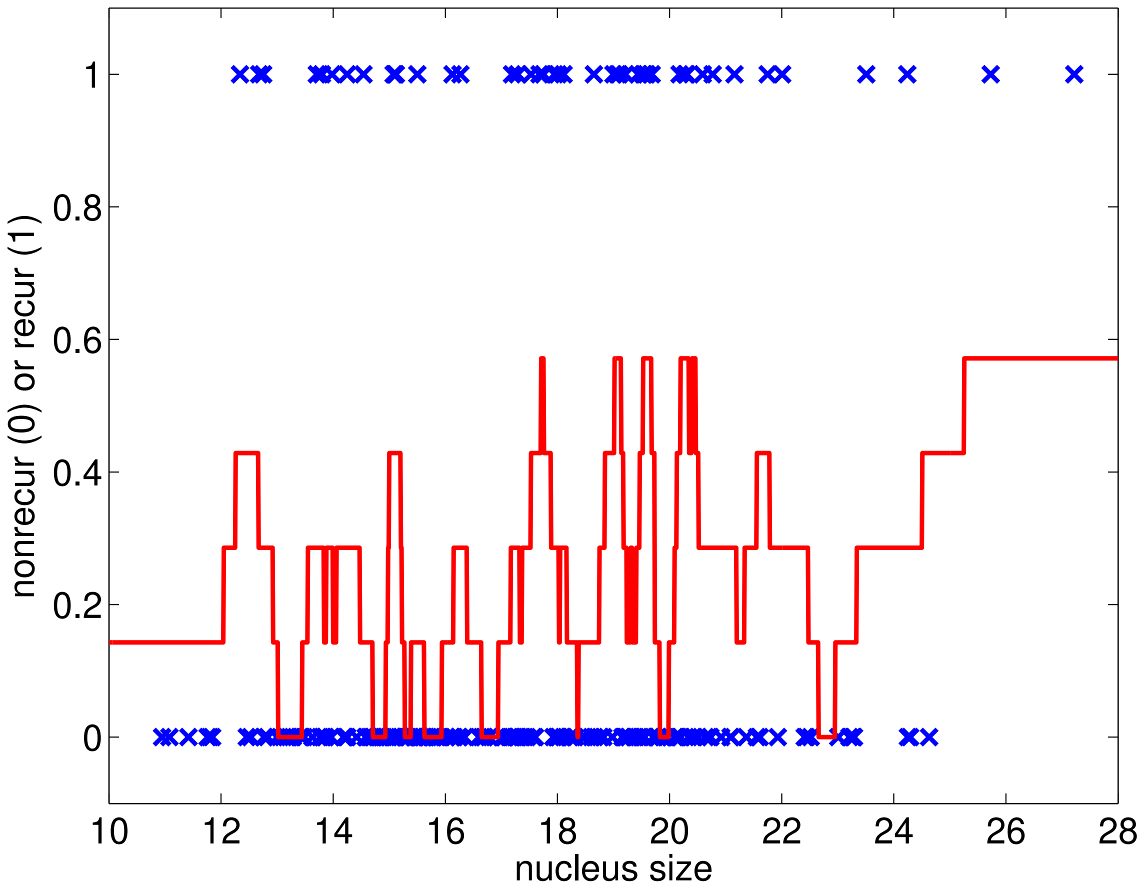
5-nearest neighbor



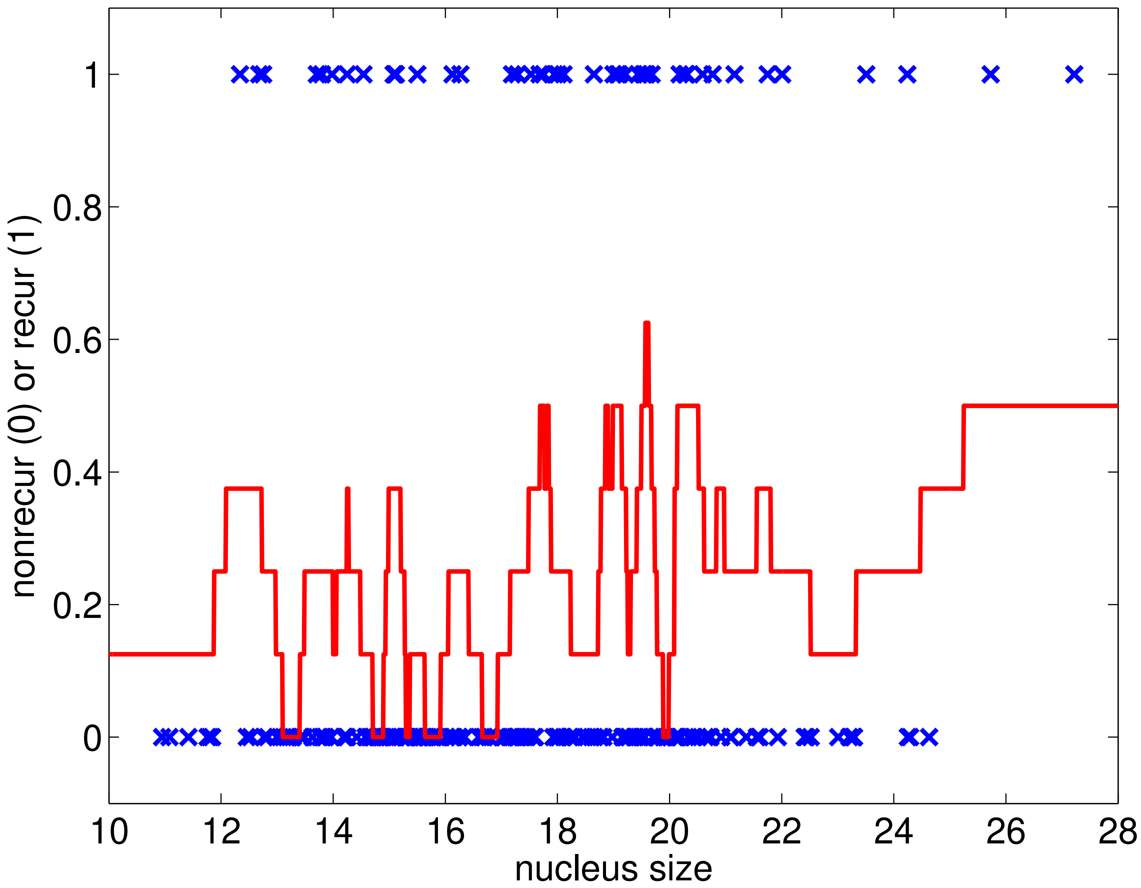
6-nearest neighbor



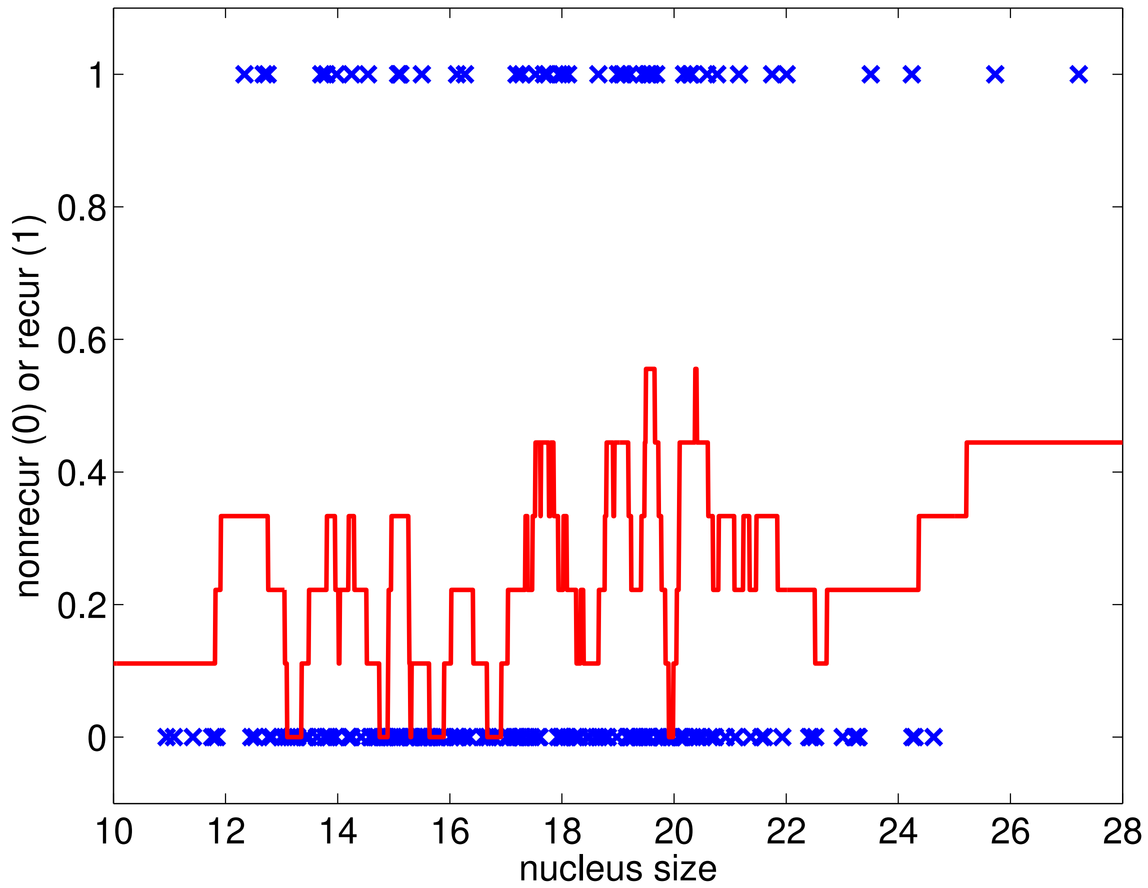
7-nearest neighbor



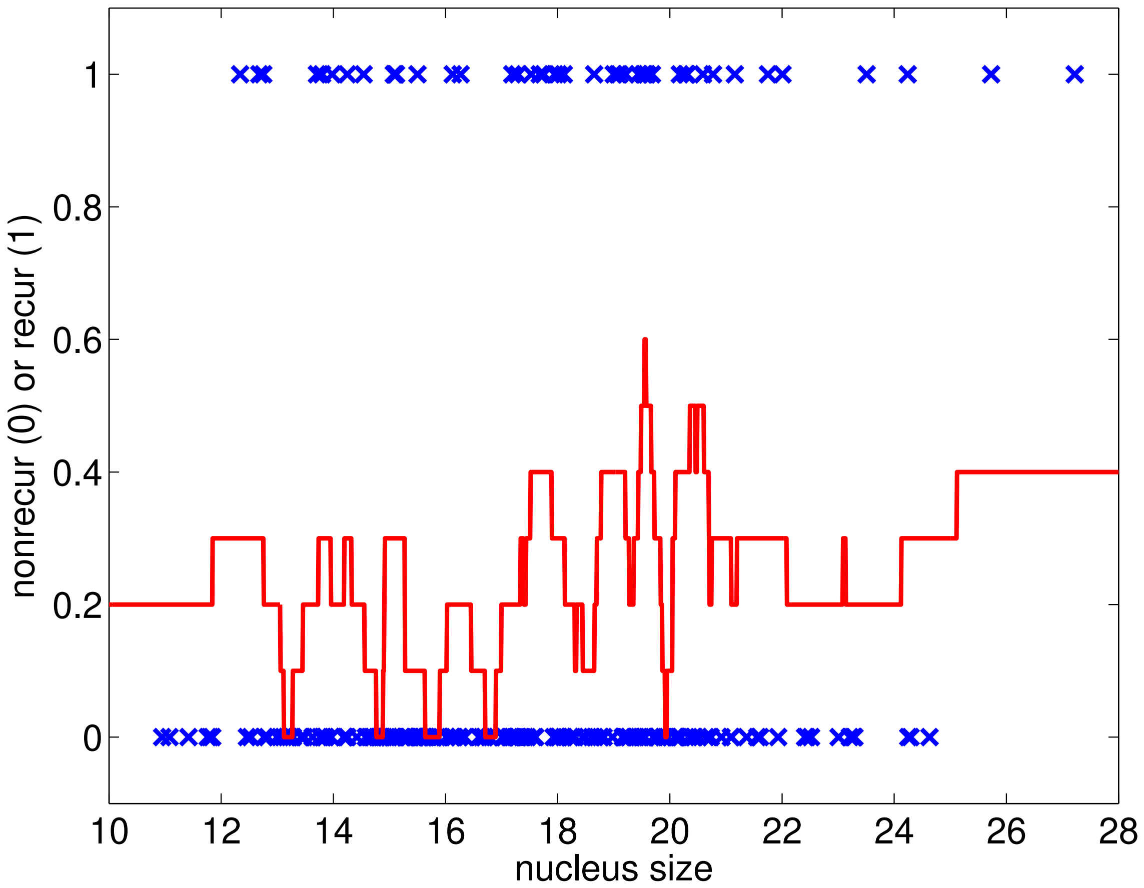
8-nearest neighbor



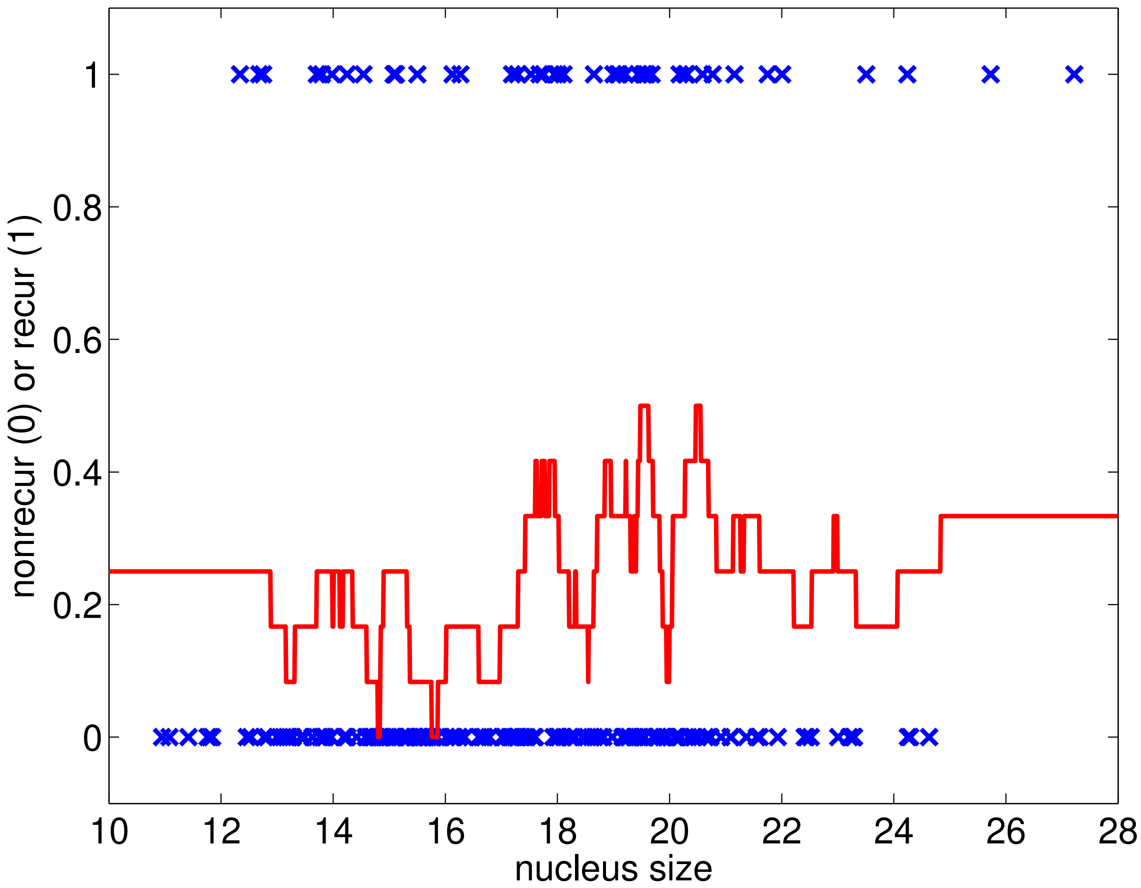
9-nearest neighbor



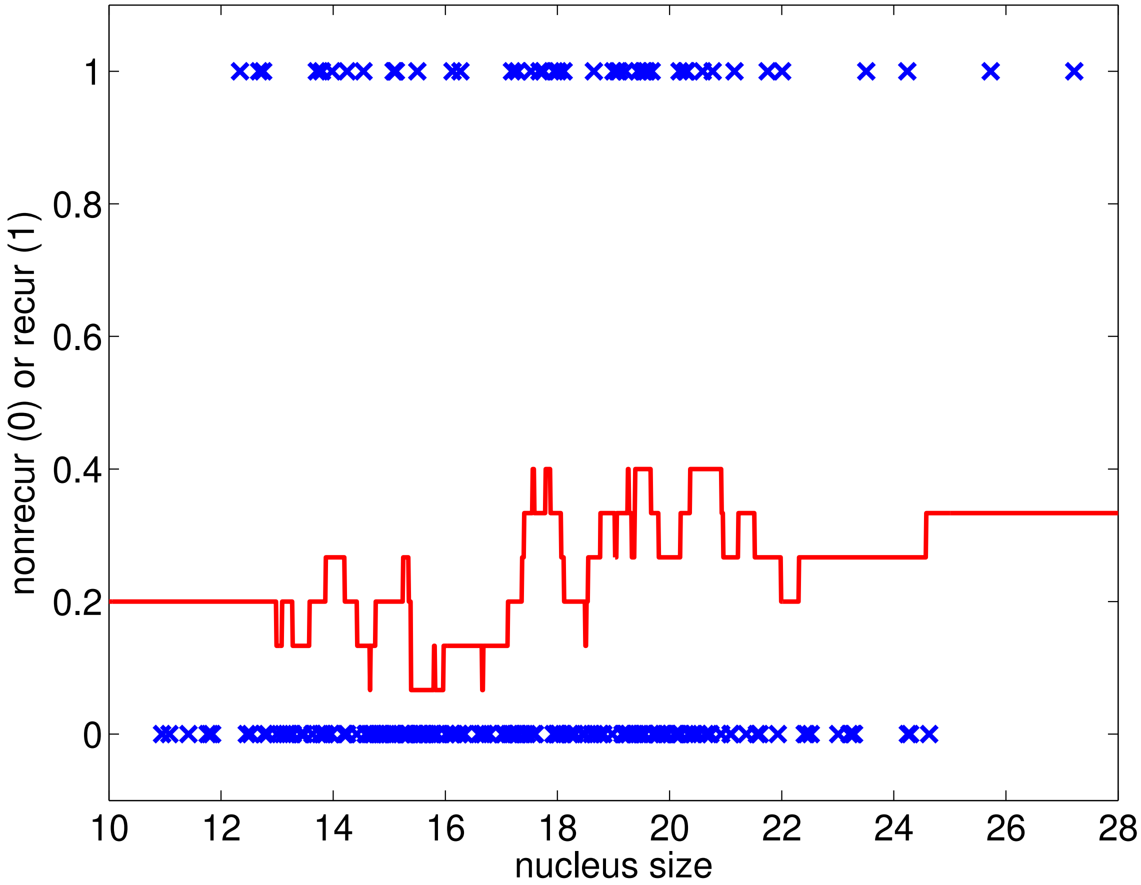
10-nearest neighbor



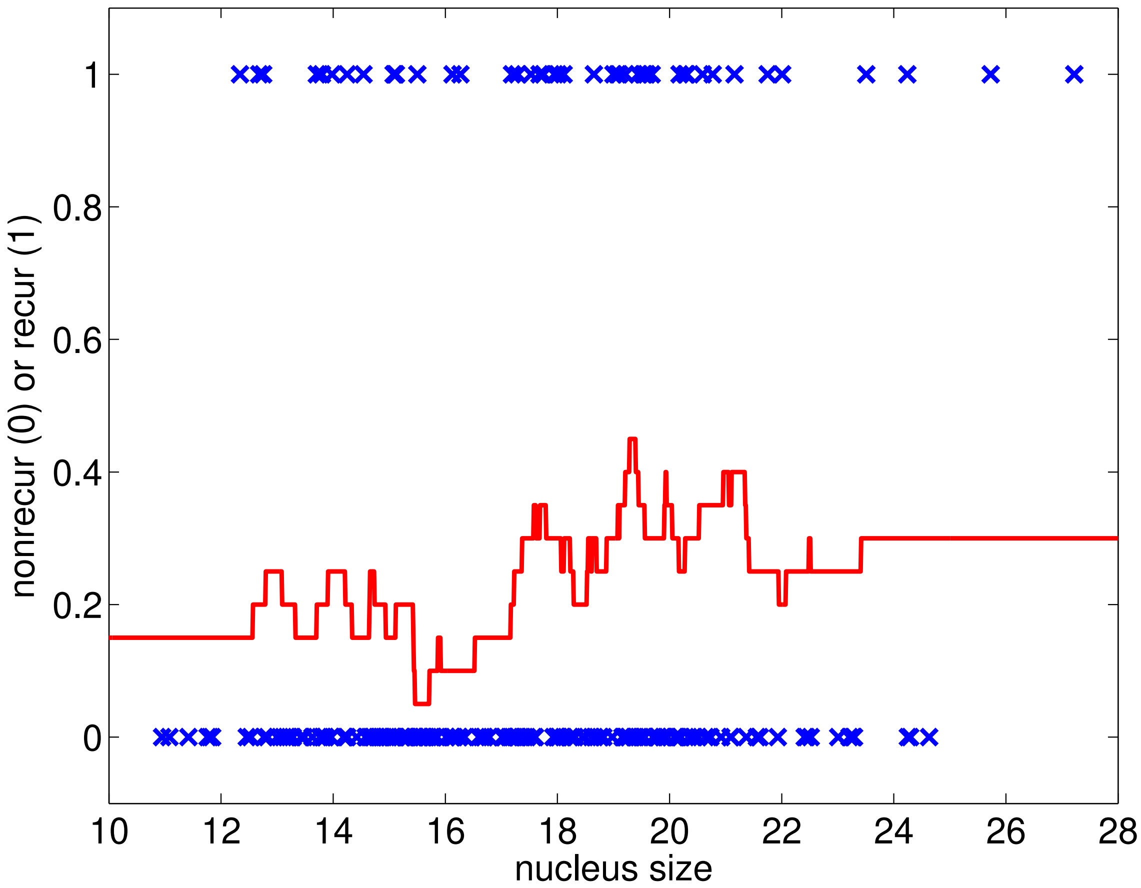
12-nearest neighbor



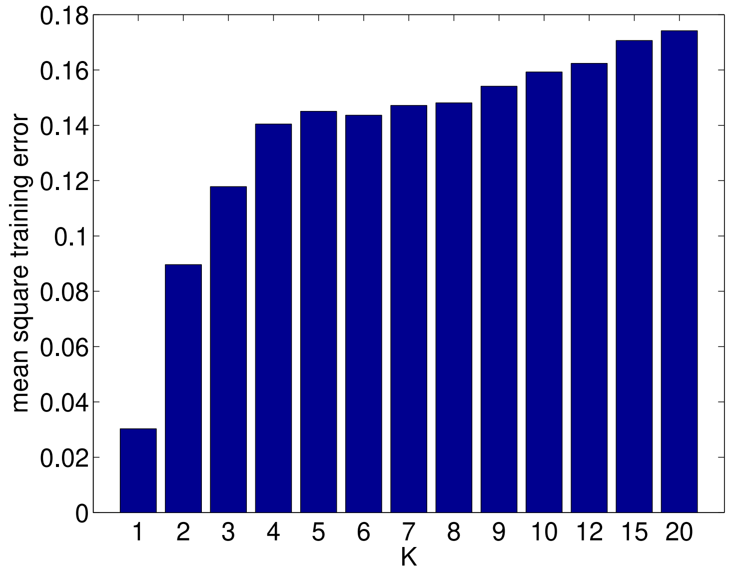
15-nearest neighbor

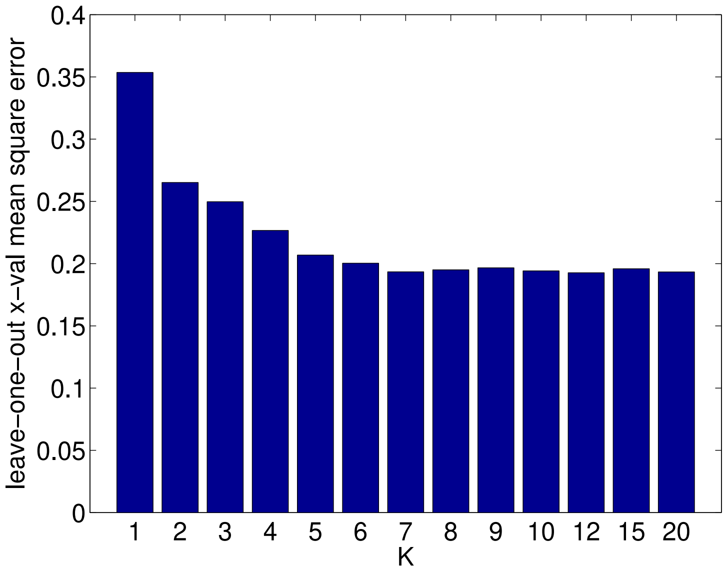


20-nearest neighbor



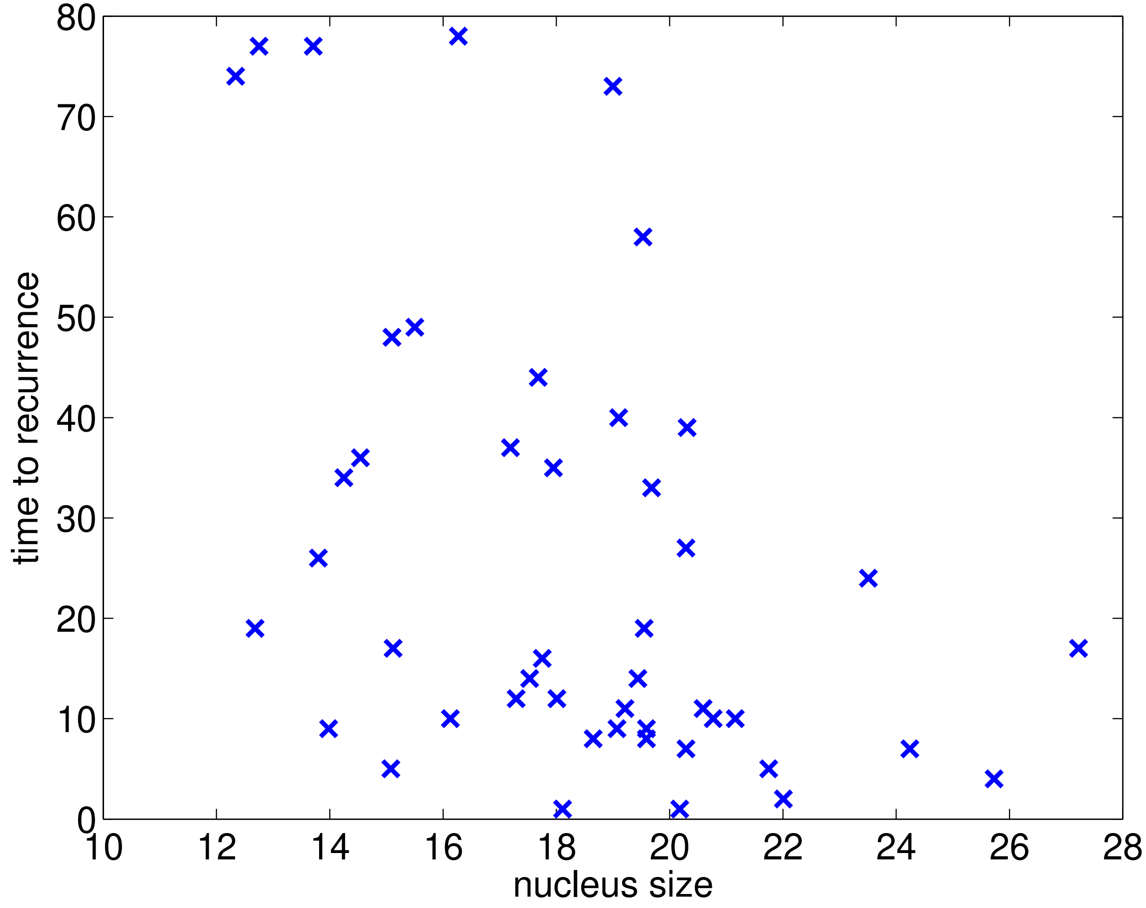
Choosing k



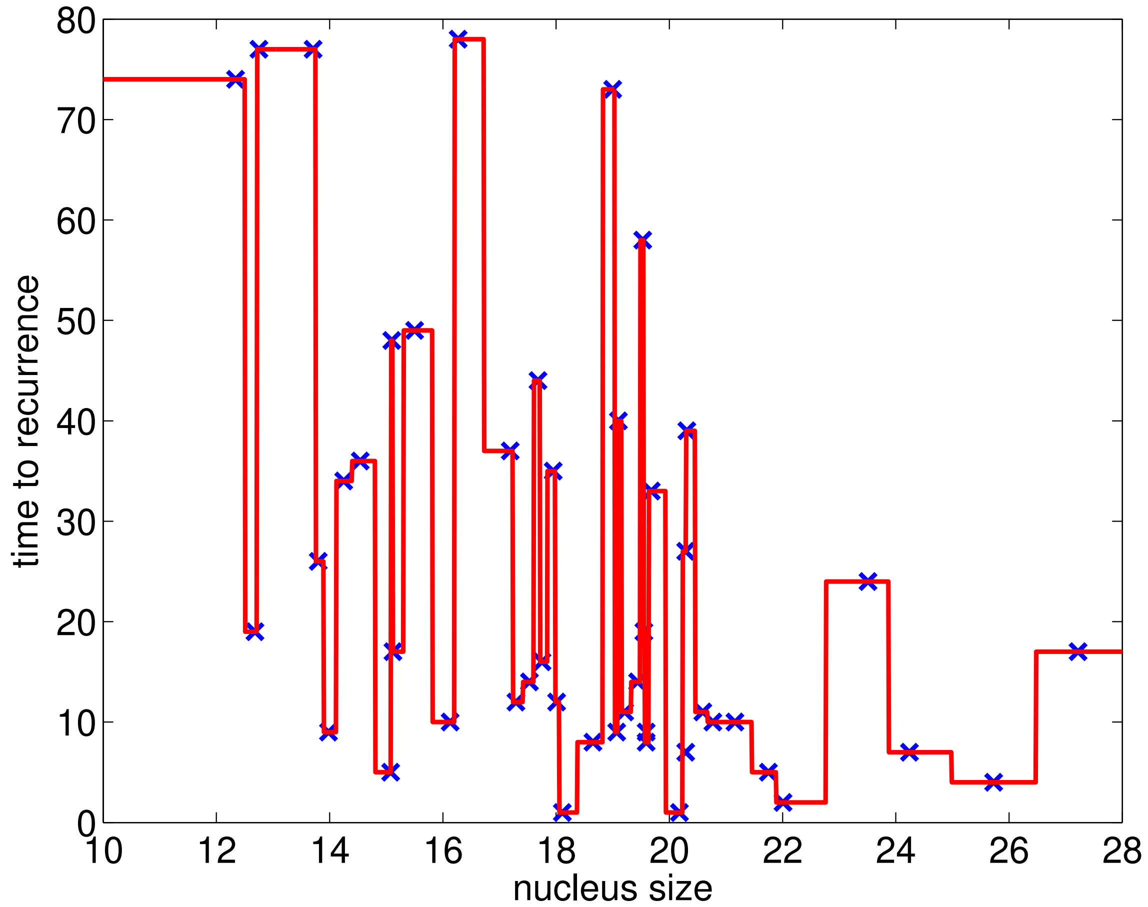


Pick the best value according to the error on the validation set

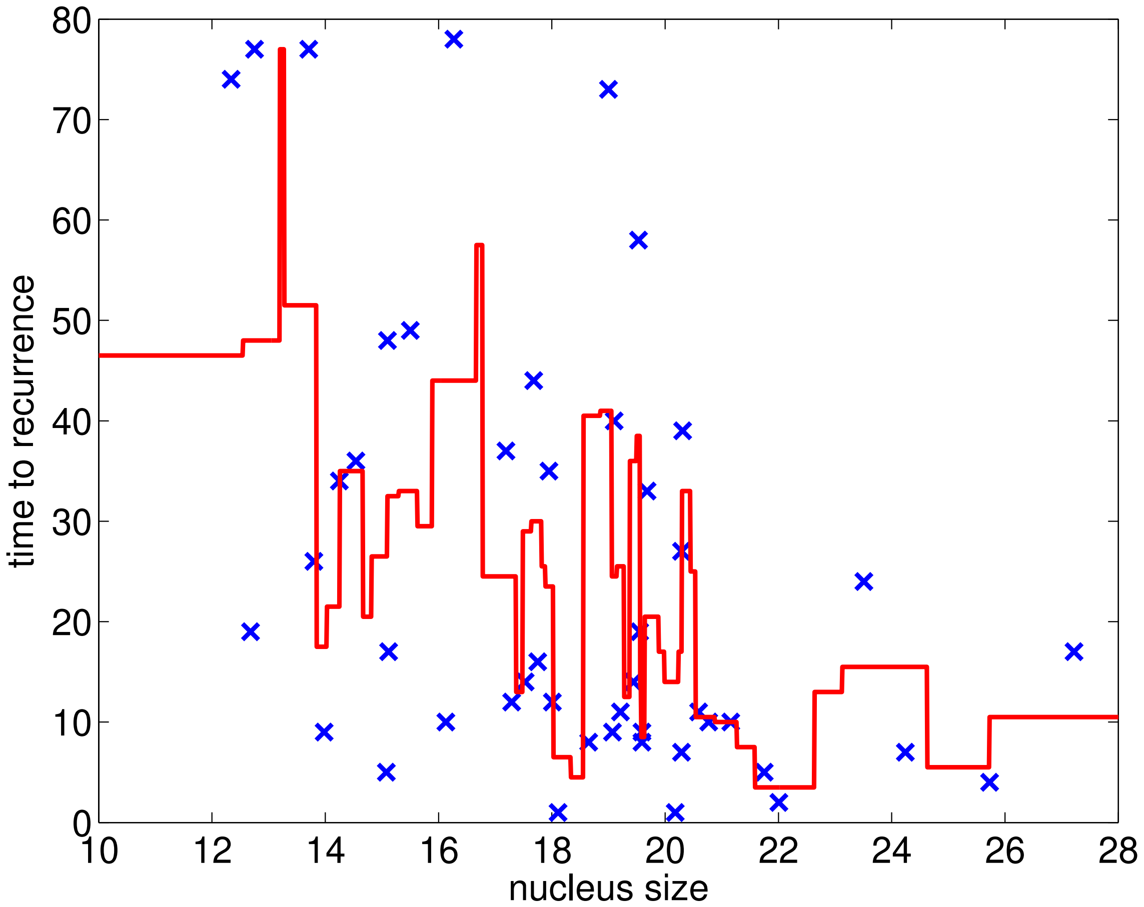
Time-to-recurrence regression



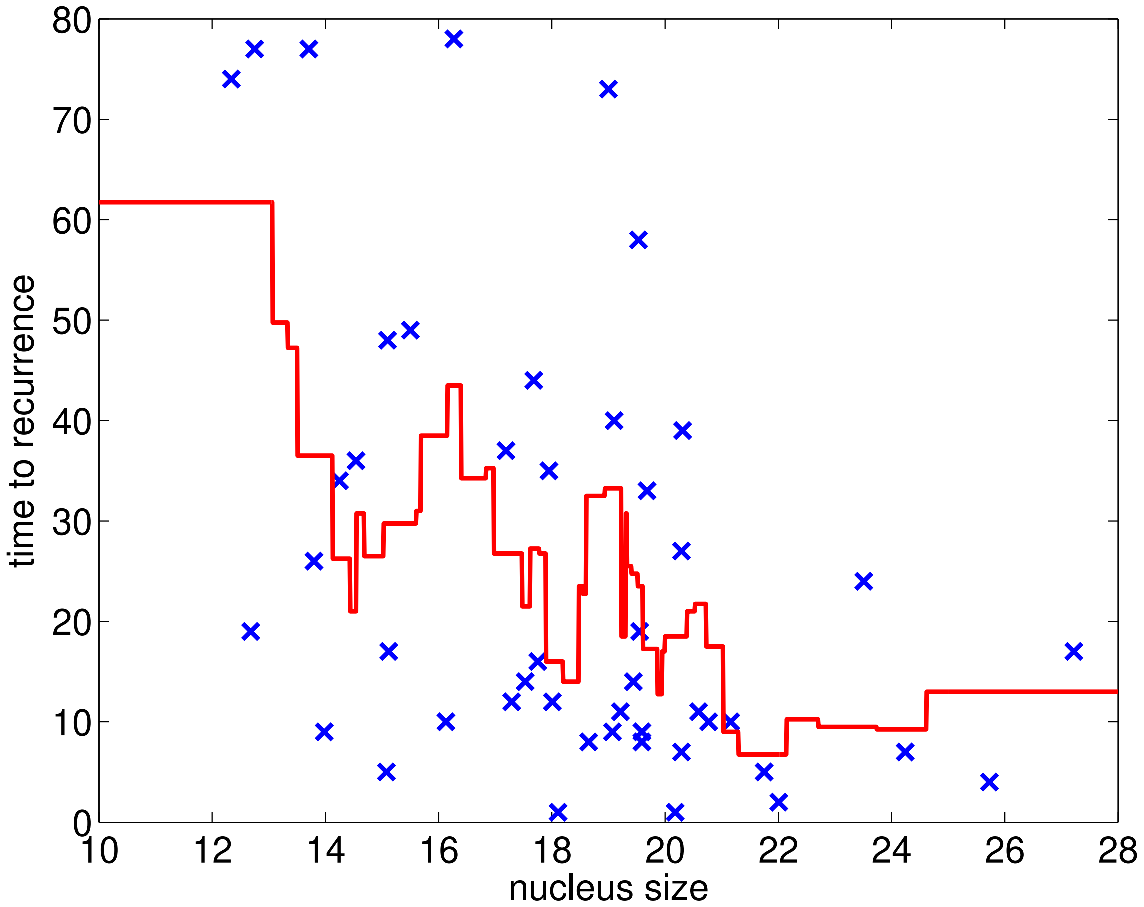
1-nearest neighbor



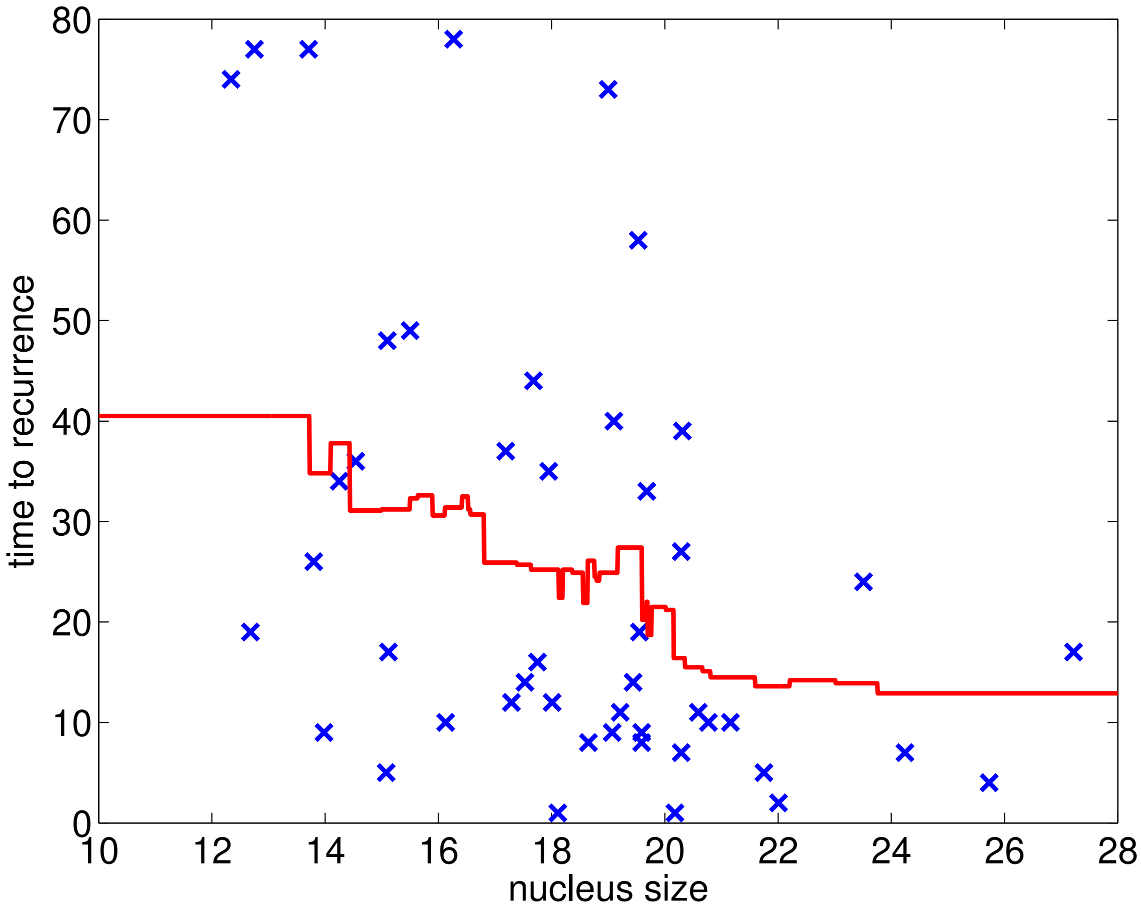
2-nearest neighbor



4-nearest neighbor



10-nearest neighbor



Some comments on k-nearest neighbor

- The output can be quite nonlinear as a function of the input $\mathbf{x}$. (Question: What's the hypothesis space?)
- The output is piecewise constant—which may or may not be a problem.
- Classifier may output zero probability for some classes for some $\mathbf{x}$.
- Training time is virtually nil—unless one does cross-validation to choose k, for example.
- Predictions may take more computation than a parametric approximator.
- Does not produce a compact description of the target function.
- k-nearest neighbor is an example of a *lazy learning algorithm*.

Lazy and Eager Learning

- **Lazy**: wait for query before generalizing
  E.g. Nearest Neighbor
- **Eager**: generalize before seeing query
  E.g. Linear/Logistic regression, Backpropagation, Generative learning, ...

Generally speaking, lazy learners:

- Are more easily nonlinear / fit local structure of target function;
- Fast training
- Non-compact
- Potentially slow prediction

Some problems with k-NN output

- The curve is jagged – piecewise constant.
- Zero probability is attached to some outcomes.
- What can we do?

Distance-weighted nearest neighbor

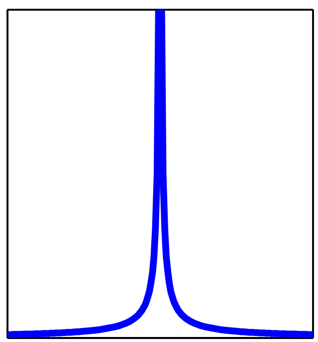
- Inputs: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$, distance metric d on $\mathcal{X}$, weighting function $w : \Re \mapsto \Re$.
- Learning: Nothing to do!
- Prediction: On input $\mathbf{x}$,
  - For each i compute $w_i = w(d(\mathbf{x}_i, \mathbf{x}))$.
  - Predict weighted majority or mean. For example,

$$y = \frac{\sum_i w_i y_i}{\sum_i w_i}$$

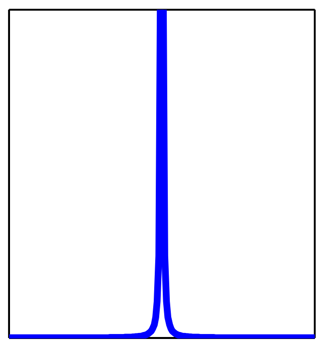
How to weight distances?

Some weighting functions

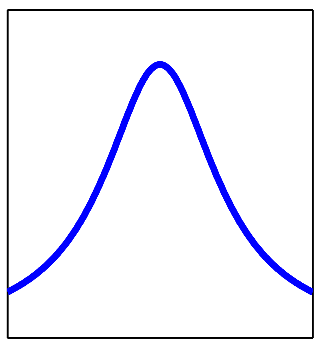
$$\frac{1}{d(\mathbf{x}_i, \mathbf{x})}$$



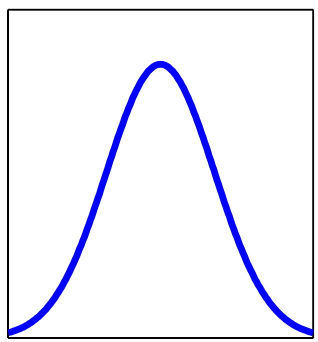
$$\frac{1}{d(\mathbf{x}_i, \mathbf{x})^2}$$



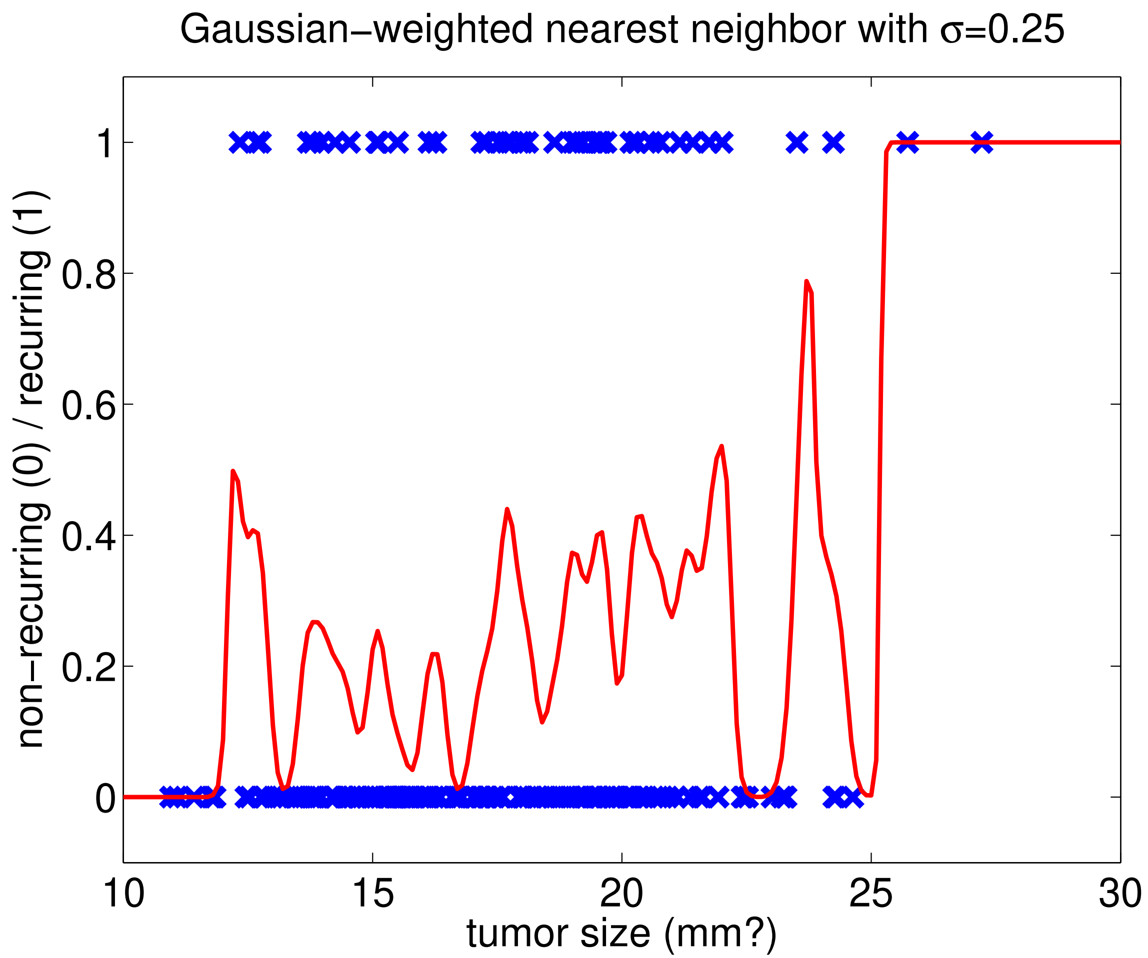
$$\frac{1}{c + d(\mathbf{x}_i, \mathbf{x})^2}$$



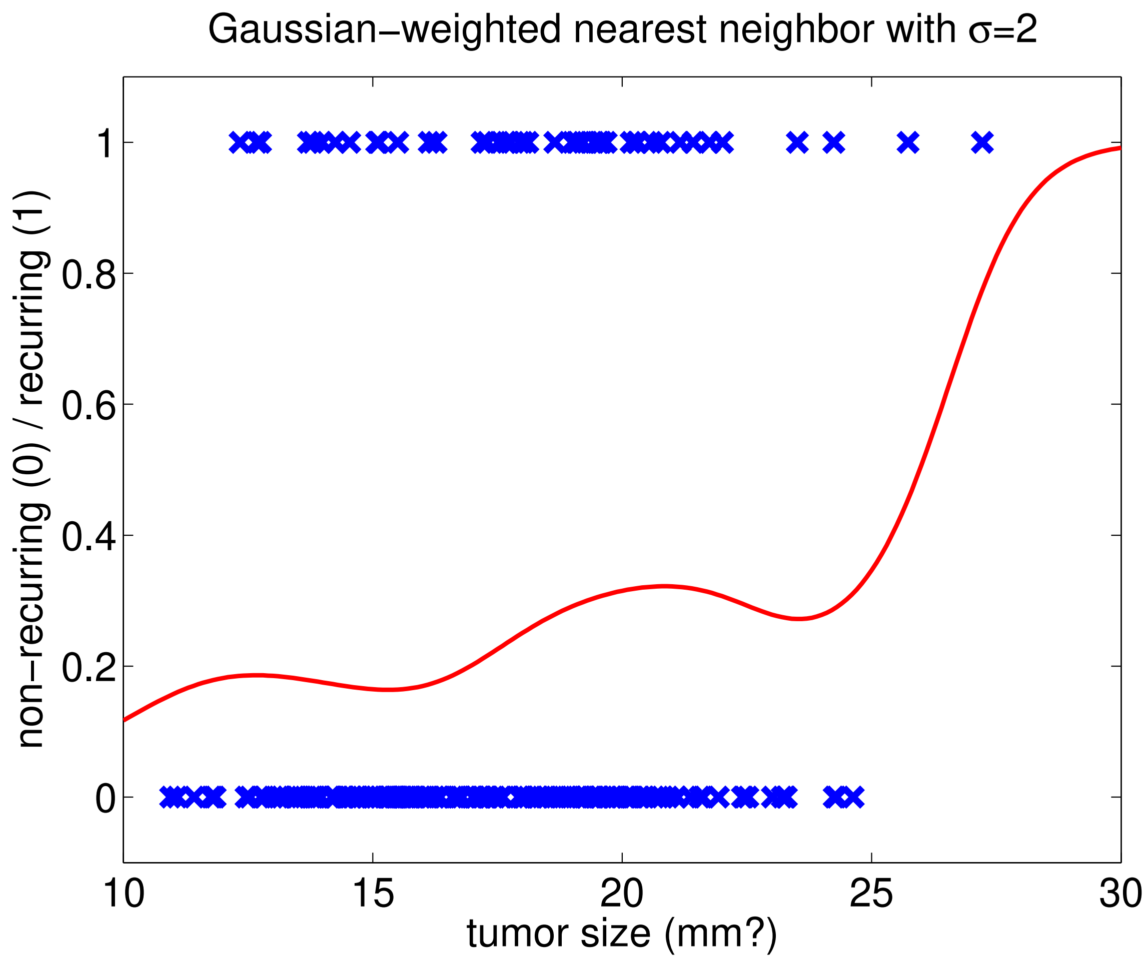
$$e^{-\frac{d(\mathbf{x}_i, \mathbf{x})^2}{\sigma^2}}$$



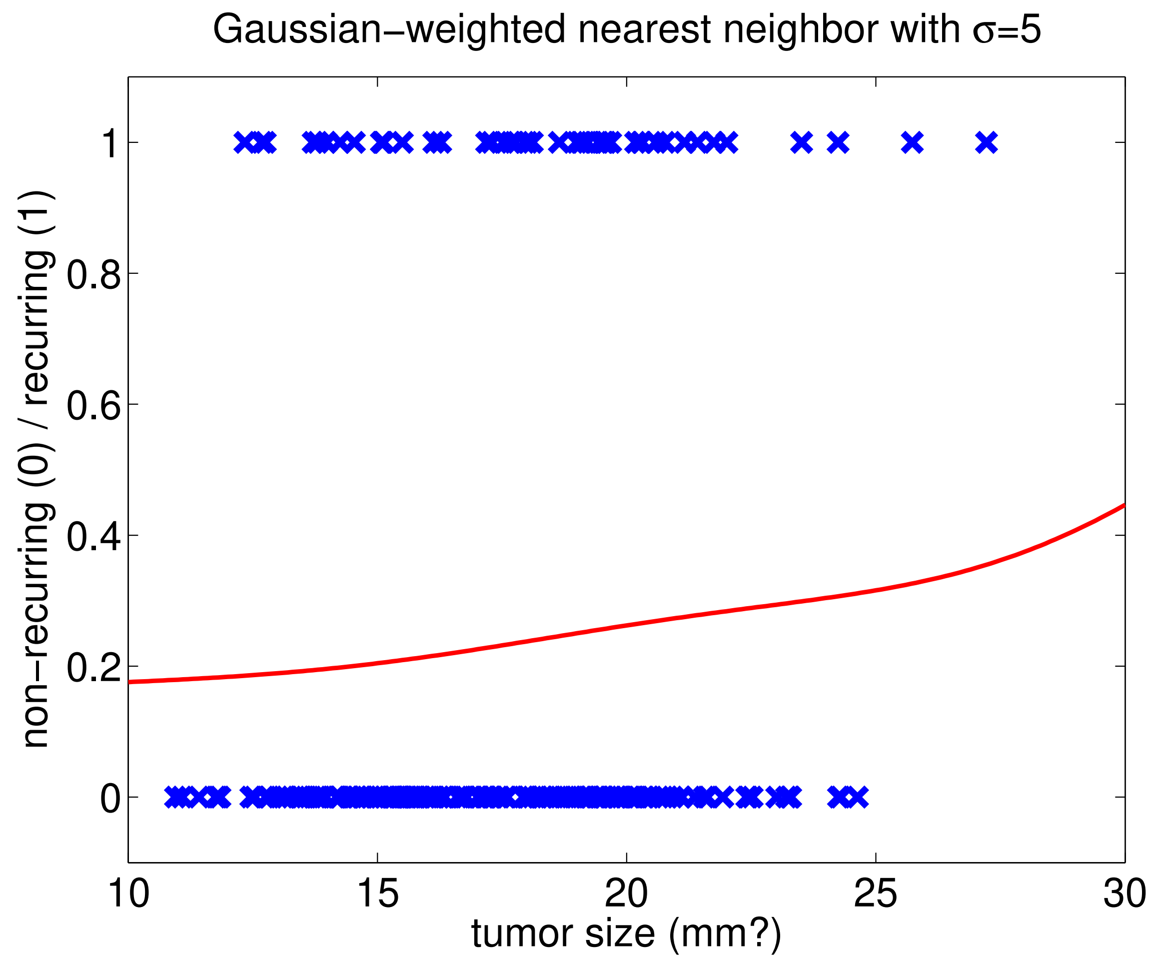
Example: Gaussian weighting, small σ



Gaussian weighting, medium σ



Gaussian weighting, large σ



- All examples get to vote!
- Cross-validate to chose level of smoothness.

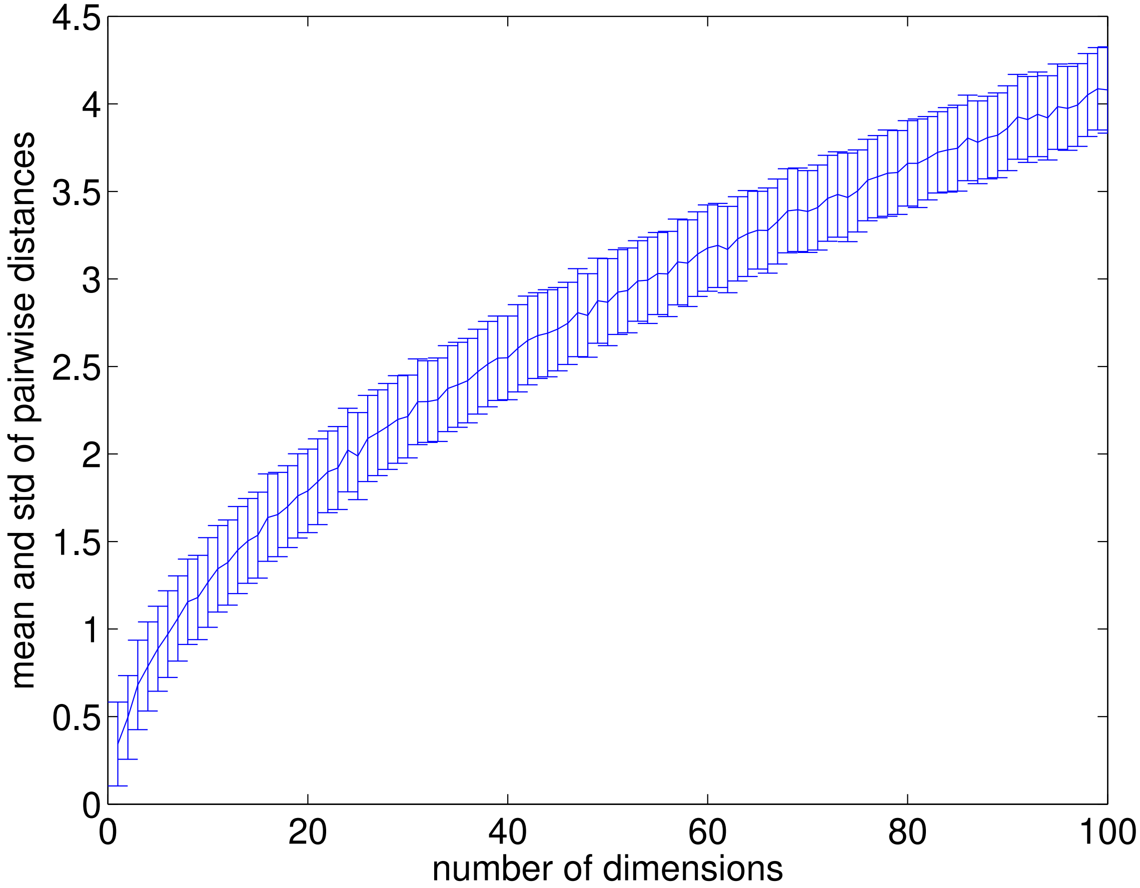
Some warnings about distances and dimensionality

- *Curse of dimensionality:*
  - Nearest-neighbor approaches are supposed to use "local" data to make new predictions.
  - The amount of data needed to "fill up" input space is exponential in the number of dimensions.
  - Distances tend to be decreasingly meaningful in high dimensions. (See next slide.)

  $\Rightarrow$ Nearest-neighber basically can't be used for high-dimensional $\mathbf{x}$. ("High" is problem-dependent.)
- Irrelevant attributes can confound distances. (Imagine adding extra, random dimensions to $\mathbf{x}$!)
- Scale of attributes influences distance calculations.

Distances in high-dimensional spaces

- Generate 100 random points uniformly in the d-dimensional space $[0, 1]^d$
- Compute mean and standard deviation of all pairwise distances



The distance metric is critical!

How do we measure distance?

- Euclidian distance
- Some other p-norm
- Maximum/minimum difference along any axis
- Weighted Euclidian distance

$$d(\mathbf{x}, \mathbf{x_i}) = \sum_{j=1}^{n} z_j (x_j - x_{i,j})^2$$

- An arbitrary distance or similarity (kernel) function d, specific for the application at hand
- The distance function is typically *fixed*, not adjusted based on data—however, one can think of trying to optimize some of these choices based on the data

The distance metric is critical! (II)

Practical considerations, which can often help. (No guarantees, though!)

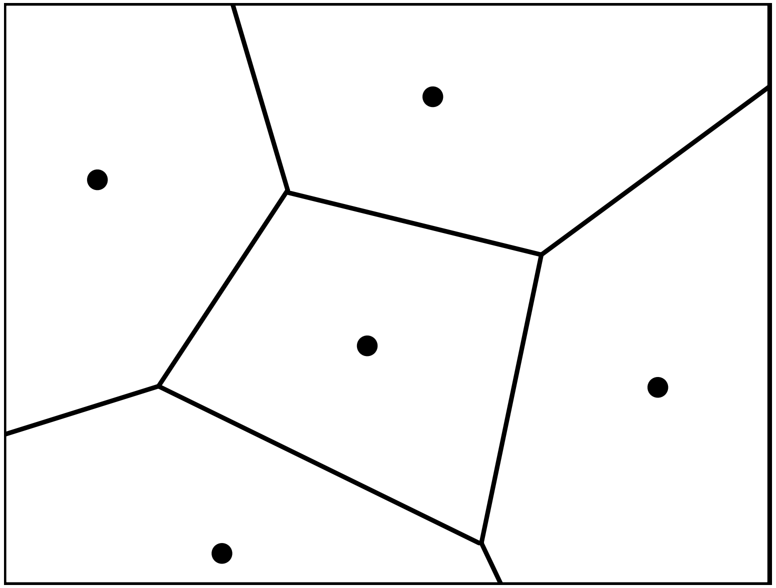
- Normalize the feature values
  (Otherwise, if their magnitudes are different, one feature could have a disproportionate effect on the metric.)
- Remove noisy or irrelevant features.
- Coalesce redundant / highly correlated features.
  (Otherwise, a correlated feature group effectively gets extra weight compared to other features.)

Improving query efficiency

- If the data set is large, computing a query by evaluating its distance to all data points can be very slow
- Possible solutions:
  - Condensation of the data set
  - kd-trees

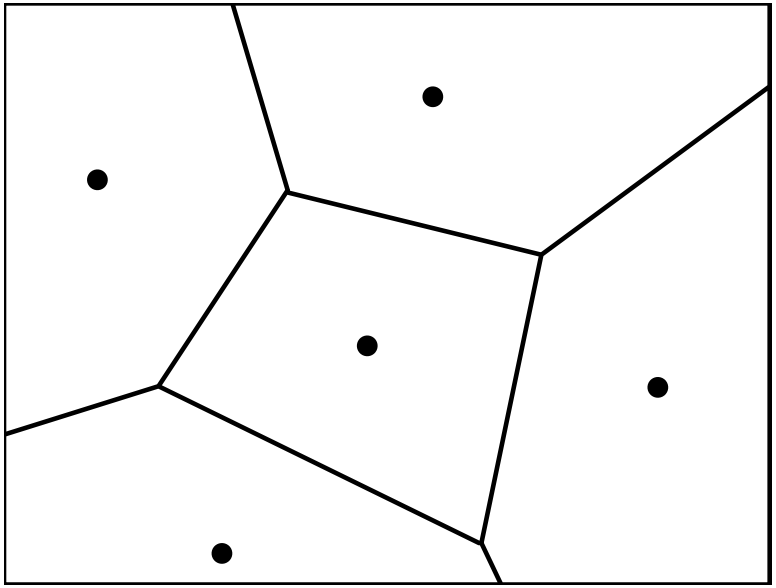
Condensation: Decision boundaries for single nearest neighbor

- Suppose we use single nearest neighbor to solve a classification problem
- Decision boundaries are never explicitly computed
- But the effective decision boundaries are a subset of the **Voronoi diagram** for the training data



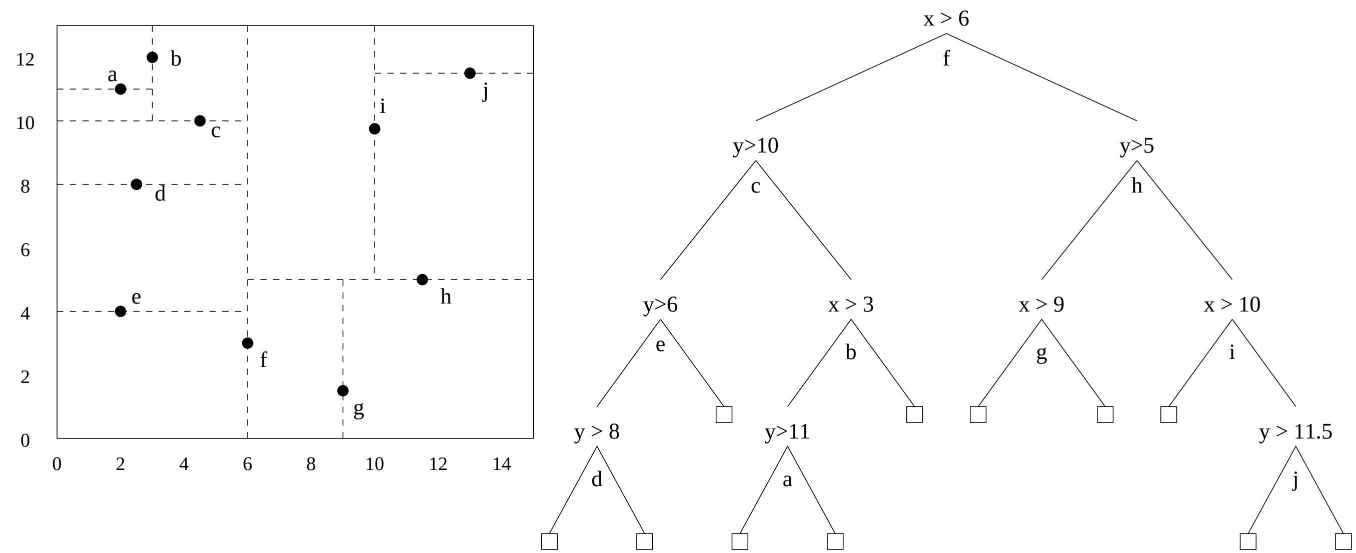
Each line segment is equidistant between two points of opposite classes.

Condensation: Main idea



- Only the points that support the decision boundary are needed to compute the classification
- Unfortunately, finding the minimal set of such points is NP complete.
- There are a number of heuristics...

kd-trees



- Split the examples using the median value of the feature with the highest variance
- Points corresponding to the splitting value are stored in the internal nodes
- We can control the depth of the tree (stop splitting)
- In this case, we will have a pool of points at the leaves, and we still need to go through all of them

Features of kd-trees

- Makes it easy to do 1-nearest neighbor
- k-nearest neighbor not so bad either
- To compute weighted nearest-neighbor efficiently, we can leave out some neighbors, if their influence on the prediction will be small

Locally-weighted regression

- So far, we have seen instance-based solutions to regression problems by averaging the y_i of the nearest neighbors, or by a weighted average of the y_i based on distances to the $\mathbf{x_i}$.
- An alternative is to construct a parametric classifier on the fly. E.g.,
  - For query point $\mathbf{x}$
  - Find k nearest neighbors (i_1, i_2, . . . , i_k)
  - This implies a local, size-k training set: $\{\langle x_{i_l}, y_{i_l}\rangle\}_{l=1}^{k}$
  - Fit a locally linear model using linear regression, giving $\mathbf{w}$.
  - Predict $y = \mathbf{w}^T\mathbf{x}$.
- Pros? Cons?
- Distance-weighted versions are also possible.
- (In principle, this could be done for classification too, though I've never seen it.)

Radial Basis Function (RBF) Networks

- Many parts of the brain have neurons which are “locally tuned” to respond only if the input is within a certain range
  E.g., neurons in the auditory part of the brain are tuned to respond to different frequencies
- But sigmoid neurons do not have this characteristic!
- Main idea: have Gaussian “receptive fields” around known data points
- Like a nearest-neighbor, but creates an **explicit** representation of the function, ahead of time.

Structure of an RBF Network

- There are a number of hidden units of the form:

$$h_i(\mathbf{x}) = \exp\left(-\frac{||\mathbf{x} - \mu_{\mathbf{i}}||^2}{2\sigma_i^2}\right)$$

  I.e. every unit is a Gaussian of mean $\mu_{\mathbf{i}}$ and covariance matrix $\sigma_i^2 I$ which will get "activated" if the input vector $\mathbf{x}$
- Hidden unit i is most active for input vectors $\mathbf{x}$ close to the mean $\mu_{\mathbf{i}}$
- The outputs are just linear combinations of the hidden units:

$$w_0 + \sum_i w_i h_i(\mathbf{x})$$

- Other choices of h_i are possible besides the Gaussian

Training RBF networks

- We want to find good values for the weights w_i, the centers $\mu_\mathbf{i}$ and the widths σ_i
- Main idea: gradient descent!
- We can compute the derivative of the error function with respect to each parameter and get a learning rule that way
- The performance of this procedure is similar to that of sigmoid multi-layered networks. But one would hope for a faster learning process...

- Idea: Train the hidden units first, then it will be easy to determine weights for them

Training RBF Networks (II)

- Heuristics for determining means:
  - Randomly choose a number of training examples $\mathbf{x_i}$ to be means.
  - Cluster the $\mathbf{x_i}$, and use cluster means for the hidden unit means. (We'll discuss clustering later in the course.)
- Heuristic to determine widths: choose the distance to the closest other unit as a width
- These ensure fast training, but generalization performance is worse

Normalized RBF networks

- A variant is a normalized RBF network, which has the same hidden units, but computes the output as

$$\frac{w_0 + \sum_i w_i h_i(\mathbf{x})}{w_0 + \sum_i w_i}$$

- This ensures that the output is not just w_0 if $\mathbf{x}$ is far from any hidden unit's center
- Choosing $w_0 = 0$, and $\mu_i = \mathbf{x_i}$, this is identical to a the weighted nearest-neighbor predictor

Nonparametric density estimation

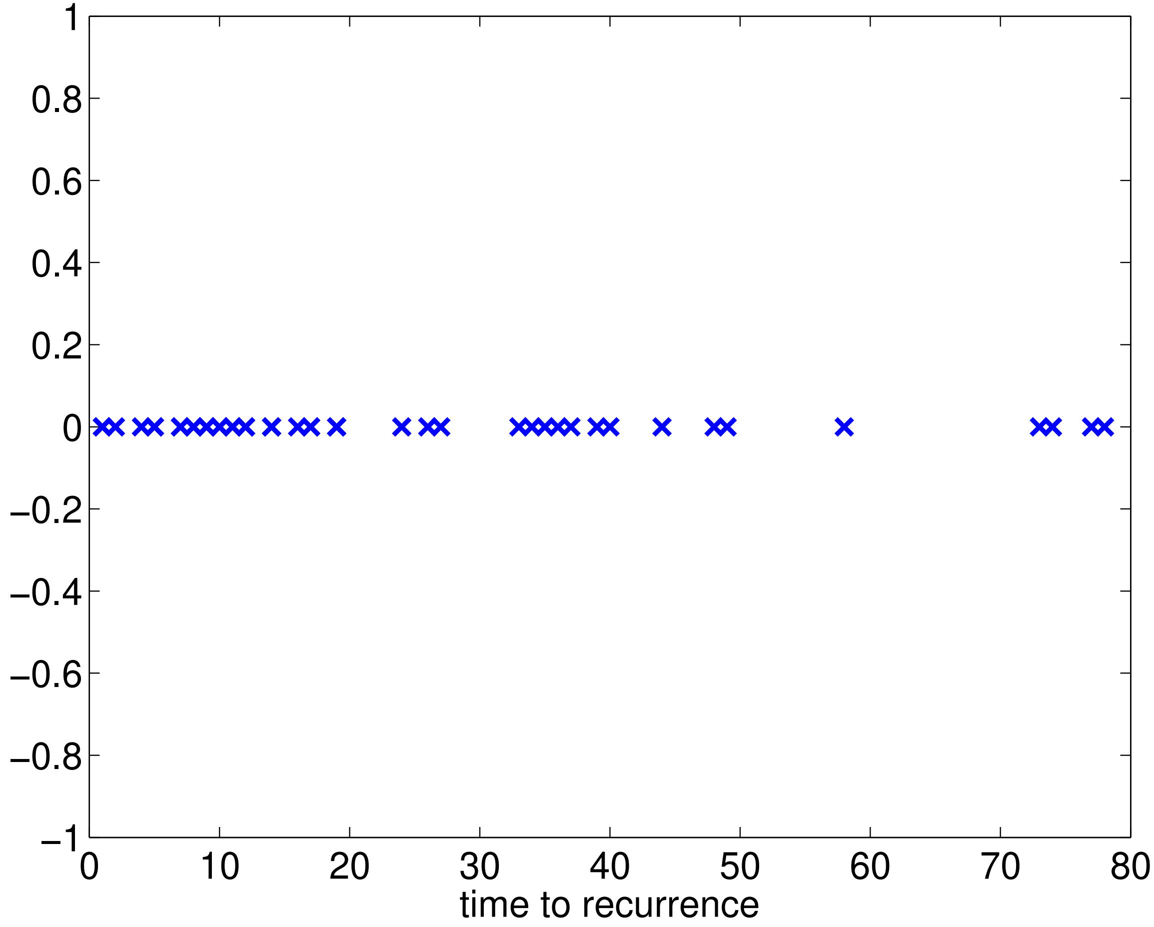
- Nonparametric methods can also be used for density estimation
- For example, supposed we are given univariate data $x_1, x_2, \ldots, x_m$.
- For each data point i, define the function

$$h_i(x) = \begin{cases} \frac{1}{2w} & \text{if } |x - x_i| < w \\ 0 & \text{otherwise} \end{cases}$$

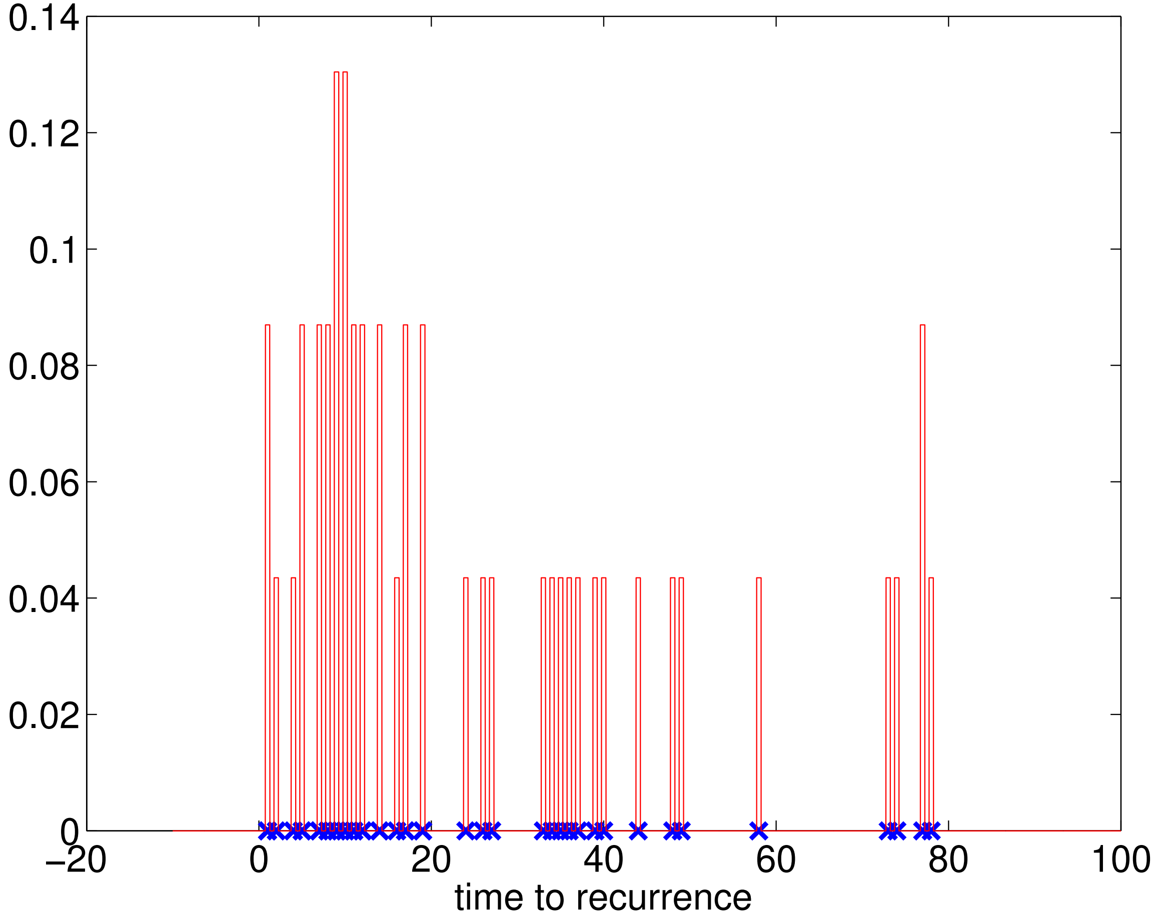
- Here, w is a "width" parameter (which can be set by cross-validation)
- The function h_i is called a "Parzen window"
- The density of x can be estimated as

$$h(x) = \frac{1}{m} \sum_{i=1}^{m} h_i(x)$$

Parzen window density estimation

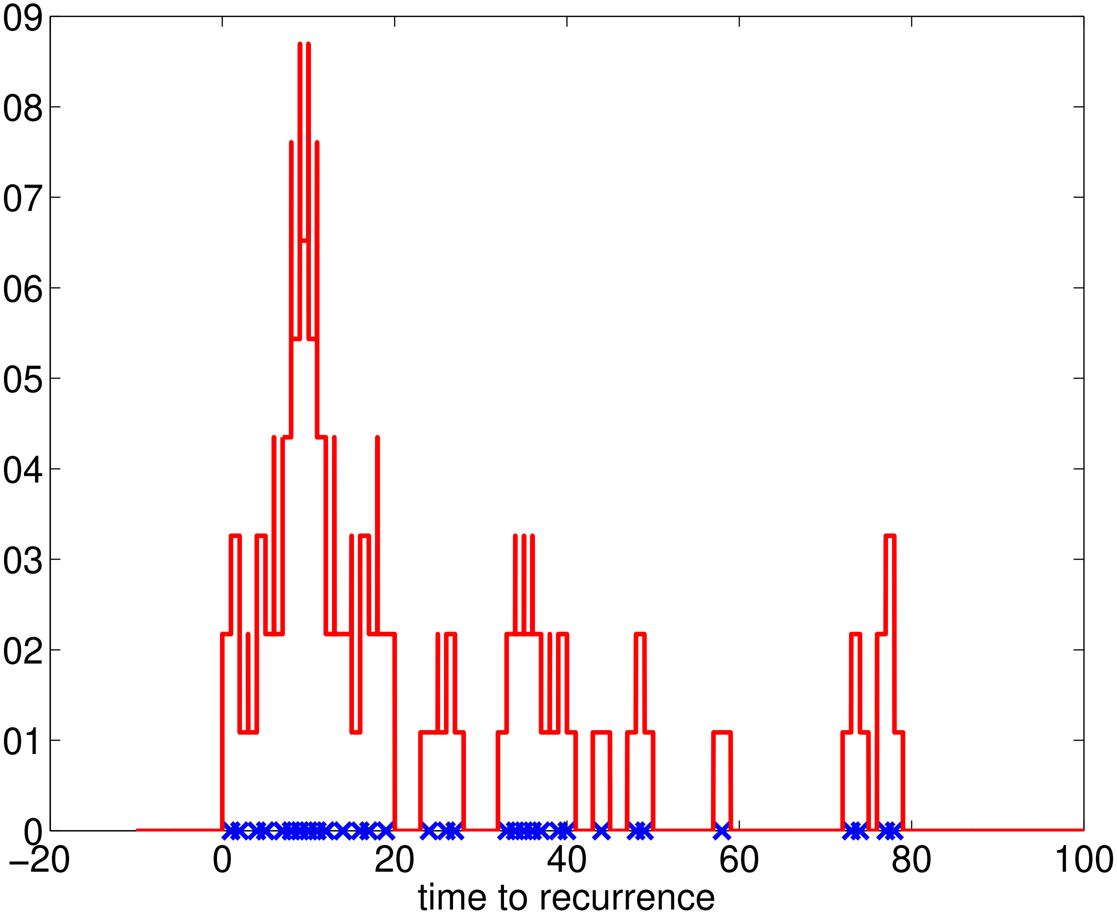


Parzen window density estimation



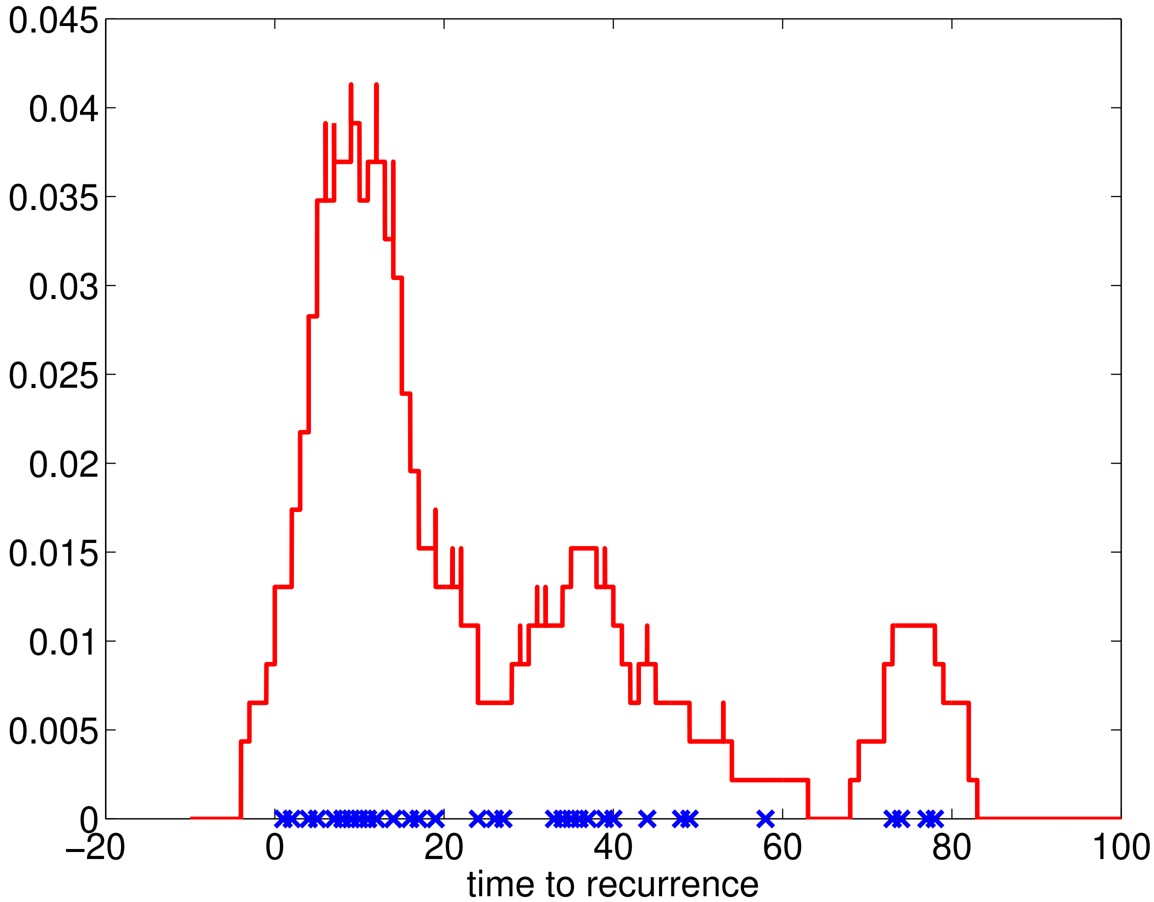
$$w = 0.25$$

Parzen window density estimation



$$w = 1$$

Parzen window density estimation



$$w = 5$$

More generally

- We can let $h_i(x)$ be any density function (uniform, Gaussian, hat, etc.) centered at x_i.
- $h(x)$ still defines a density, as it is nonnegative and integrates to one.

When to consider instance-based learning

- Instances map to points in $\Re^n$
- Less than 20 attributes per instance, lots of data

Advantages:

- Training is very fast
- Easy to learn complex functions over few variables
- Can give back confidence intervals in addition to the prediction
- Variable resolution (depends on the data)
- Does not lose any information

Disadvantages:

- Slow at query time
- *Easily fooled by irrelevant attributes*
- Cannot be used directly for problems with hundreds of inputs