Lecture 6: Instance-Based Learning

- Nearest-neighbor methods
- Kernel regression
- Locally weighted regression

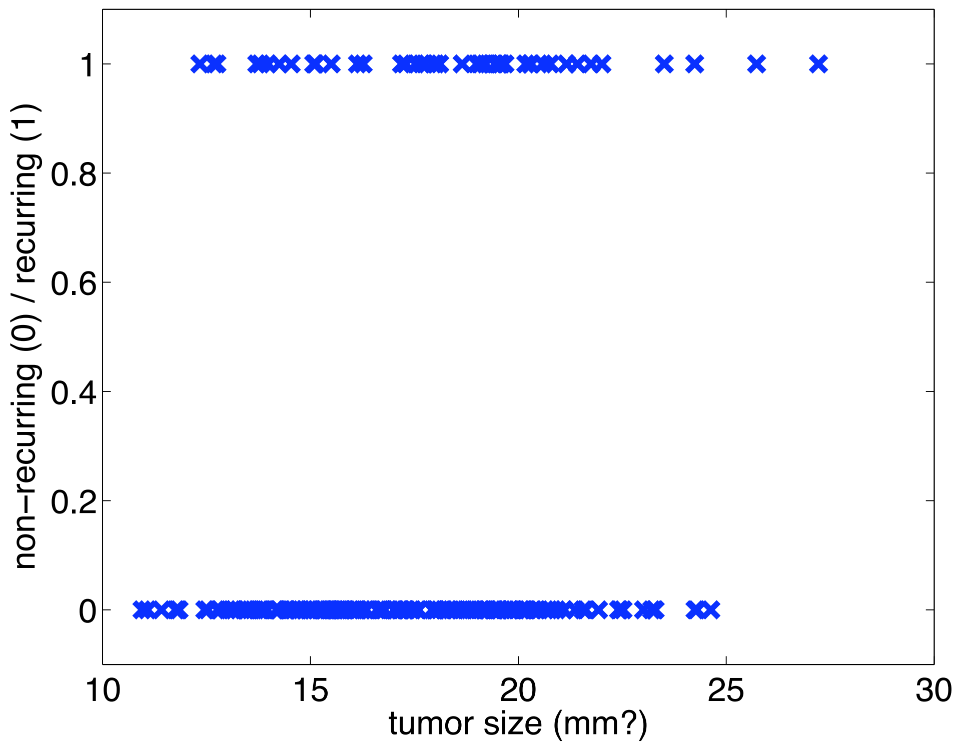
Recall: Parametric supervised learning

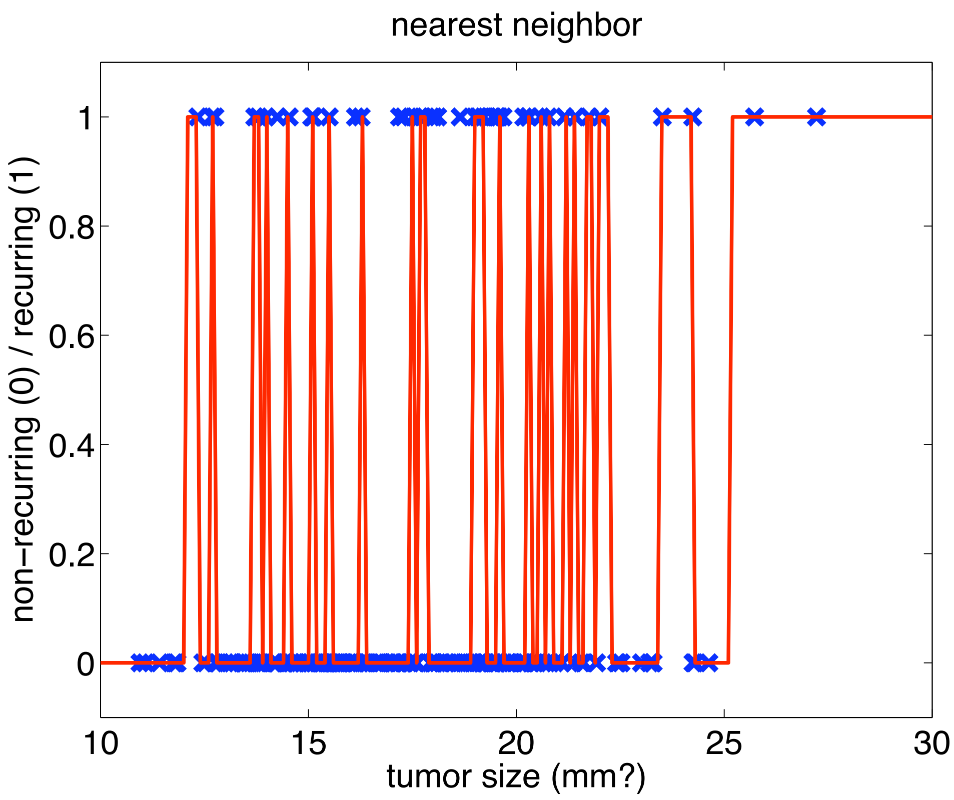
- So far, we have assumed that we have a data set D of labeled examples
- From this, we learn a *parameter vector* or such that some error measure based on the training data is minimized
- These methods are called *parametric*, and their main goal is to summarize the data using the parameters
- Parametric methods are typically global, i.e. have one set of parameters for the entire data space
- But what if we just remembered the data?
- When new instances arrive, we will compare them with what we know, and determine the answer

Non-parametric (memory-based) learning methods

- Key idea: just store all training examples $\langle \mathbf{x_i}, y_i \rangle$
- When a query is made, compute the value of the new instance based on the values of the *closest (most similar) points*
- Requirements:
  - A distance function
  - How many closest points (neighbors) to look at?
  - How do we compute the value of the new point based on the existing values?

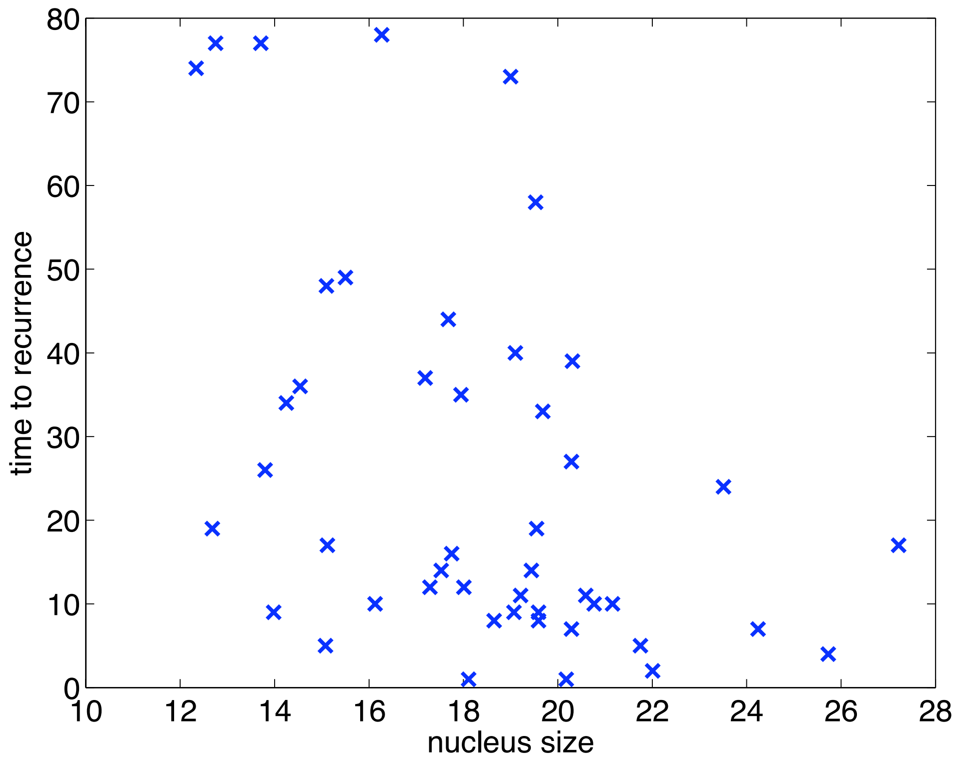
Simple idea: Connect the dots!

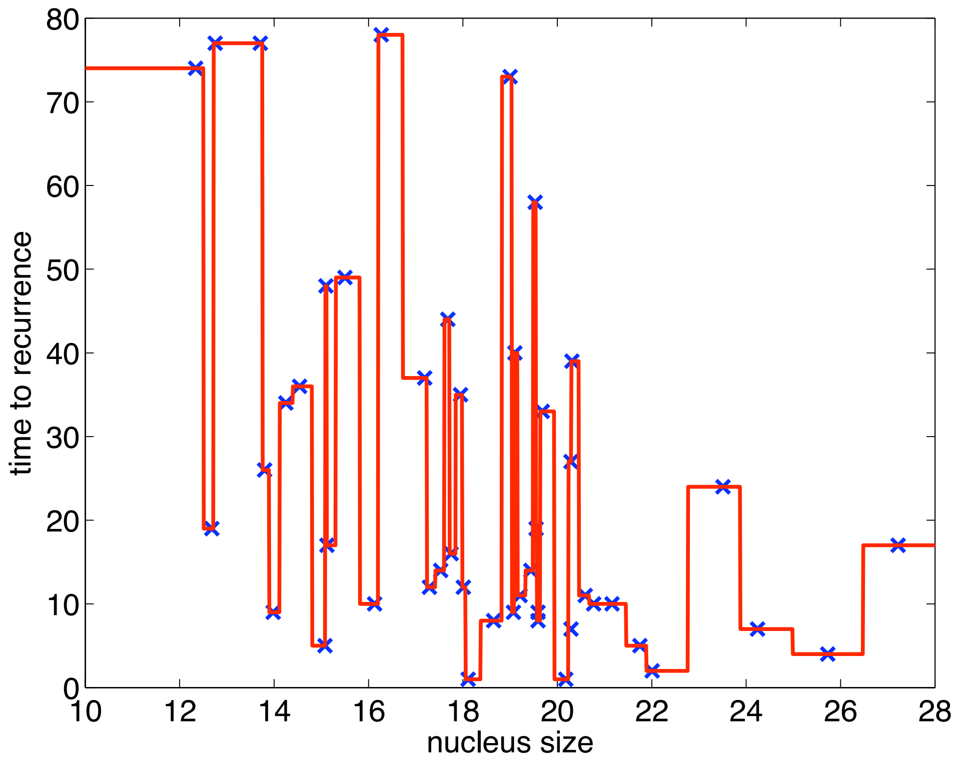




Wisconsin data set, classification

Simple idea: Connect the dots!





Wisconsin data set, regression

One-nearest neighbor

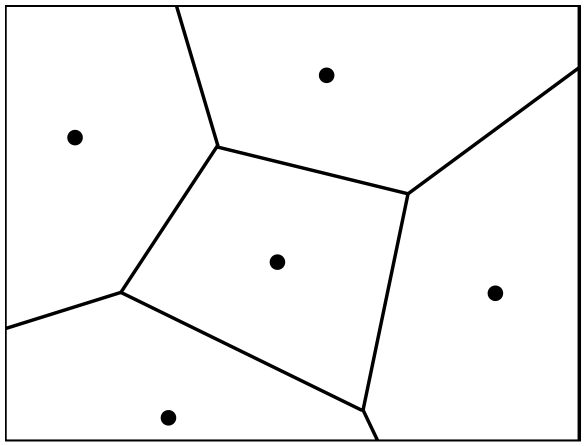
- Given: Training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$, distance metric d on $\mathcal{X}$.
- Learning: Nothing to do! (just store data)
- Prediction: for $\mathbf{x} \in \mathcal{X}$
  - Find nearest training sample to $\mathbf{x}$.

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

  - Predict $\mathbf{y} = \mathbf{y}_i$.

What does the approximator look like?

- Nearest-neighbor does not explicitly compute decision boundaries
- 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.

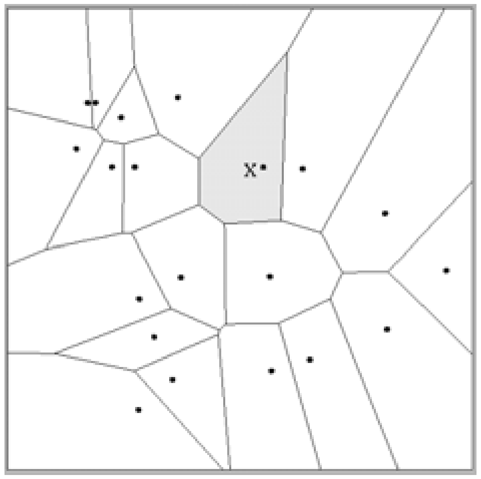
What kind of distance metric?

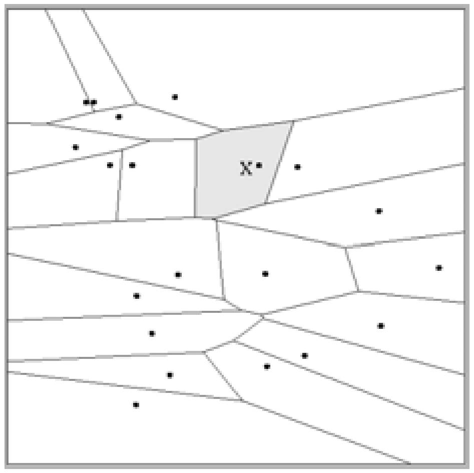
- Euclidian distance
- Maximum/minimum difference along any axis
- Weighted Euclidian distance (with weights based on domain knowledge)

$$\sum_i w_i (x_{q,i} - x_{t,i})^2$$

- An arbitrary distance or similarity function d, specific for the application at hand (works best, if you have one)
- Most often the distance function is fixed (more on how to learn this later)

Distance metric is really important!





Left: both attributes weighted equally; Right: second attributes weighted more

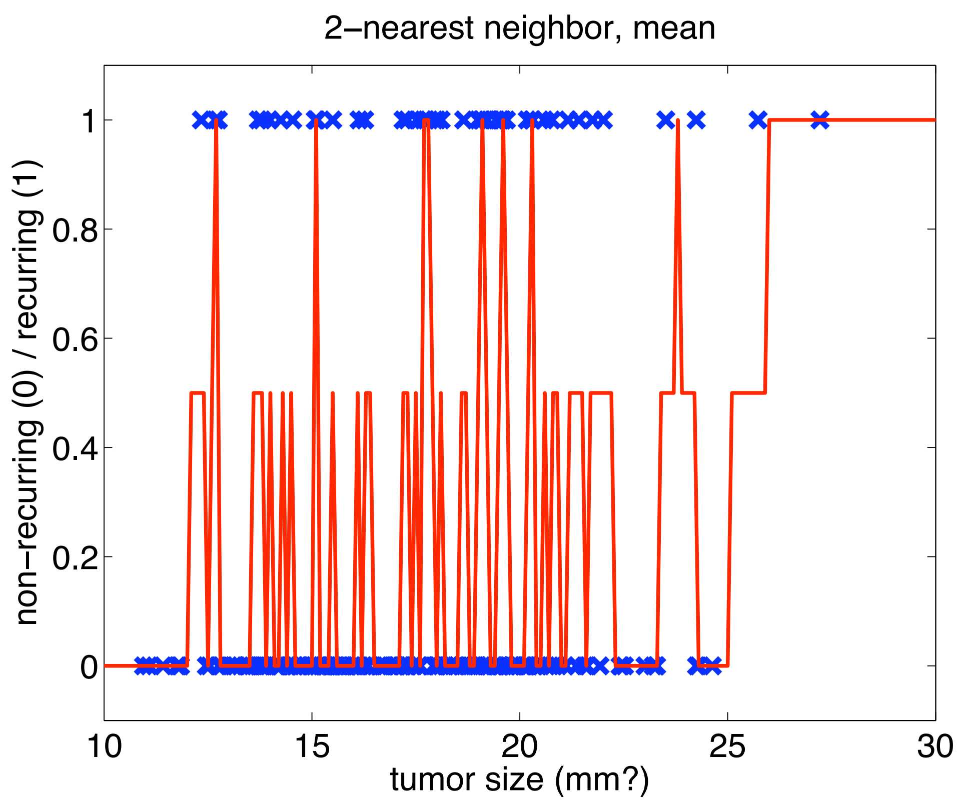
You may need to do preprocessing:

- *Scale* the input dimensions (or normalize them)
- Remove noisy inputs
- Determine weights based on cross-validation (or information-theoretic methods)

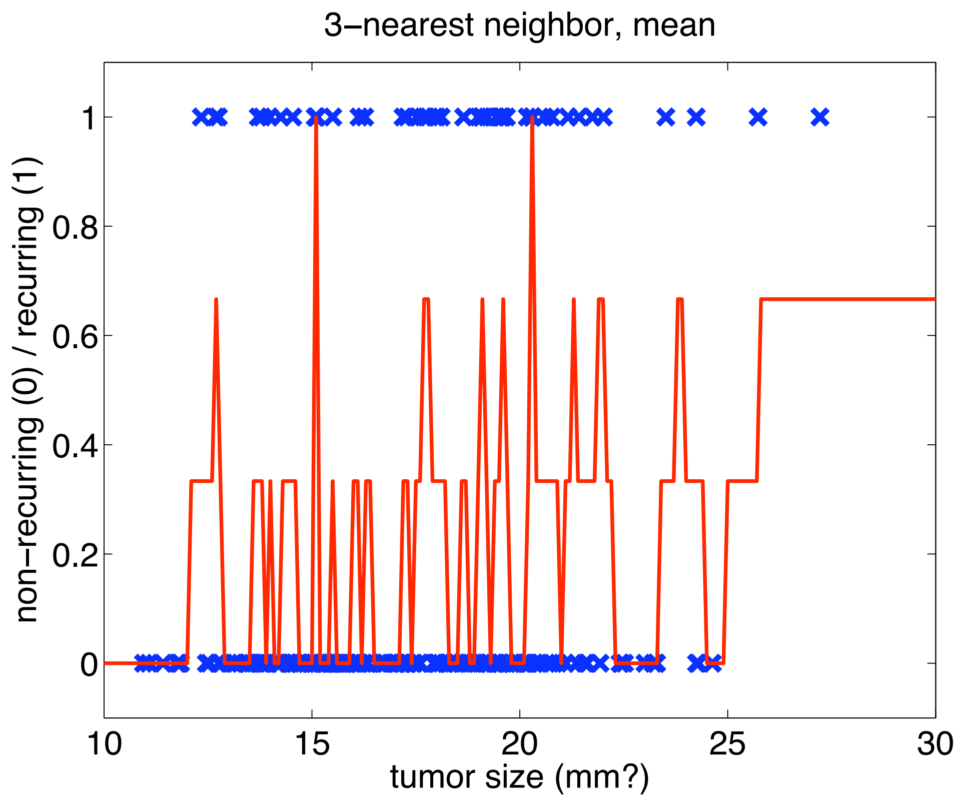
k-nearest neighbor

- Given: Training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$, 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
    * $\mathbf{y} =$ mean/median of $\{\mathbf{y}_{i_1}, \mathbf{y}_{i_2}, \ldots, \mathbf{y}_{i_k}\}$ for regression
    * $\mathbf{y} =$ majority of $\{\mathbf{y}_{i_1}, \mathbf{y}_{i_2}, \ldots, \mathbf{y}_{i_k}\}$ for classification, or empirical probability of each class

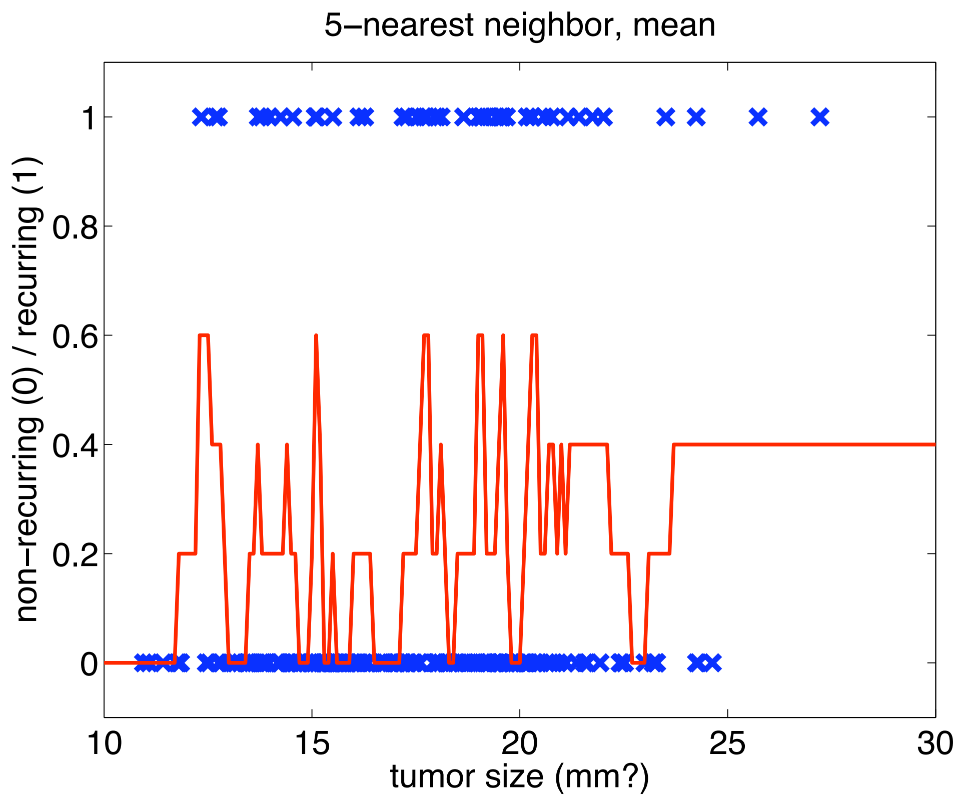
Classification, 2-nearest neighbor, empirical distribution



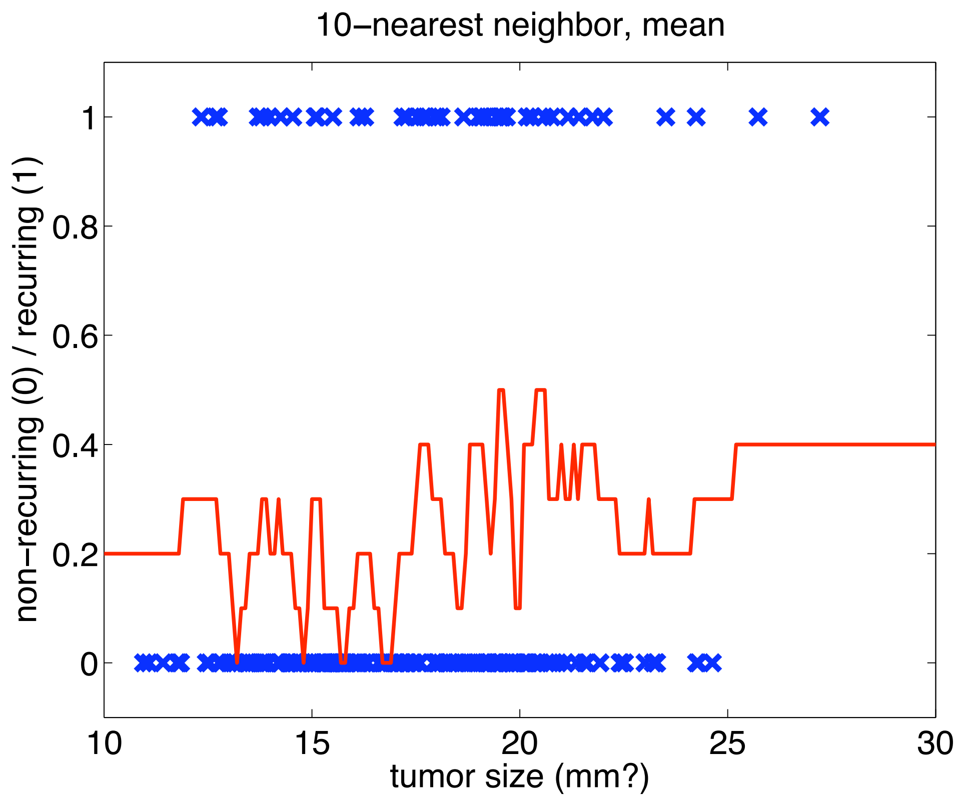
Classification, 3-nearest neighbor



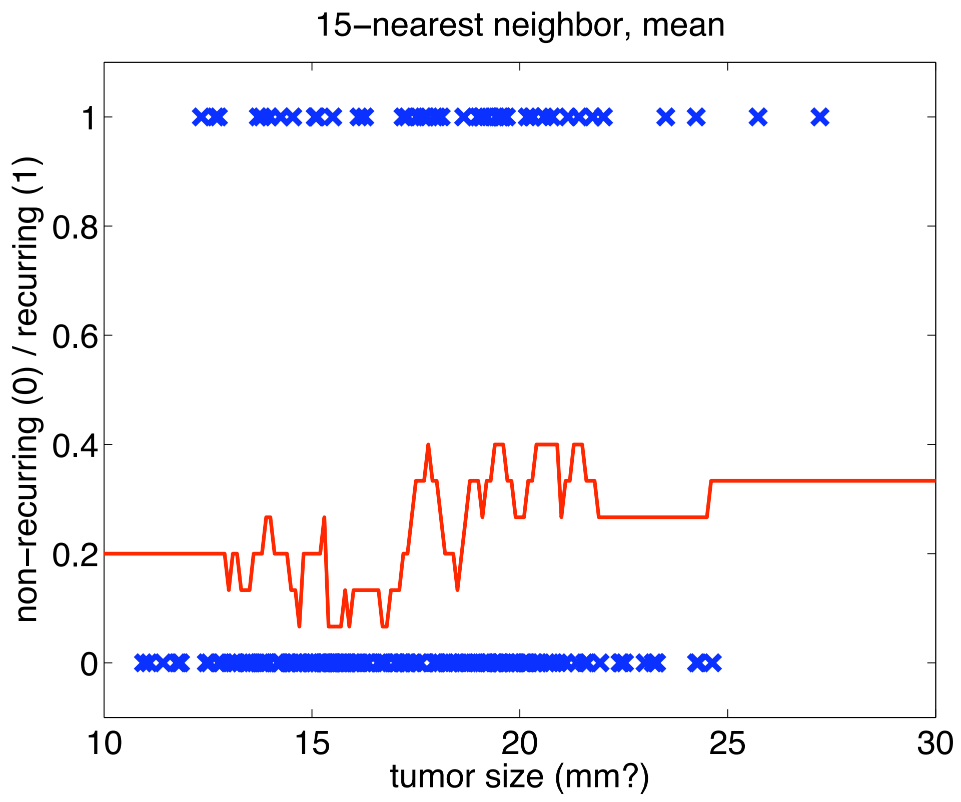
Classification, 5-nearest neighbor



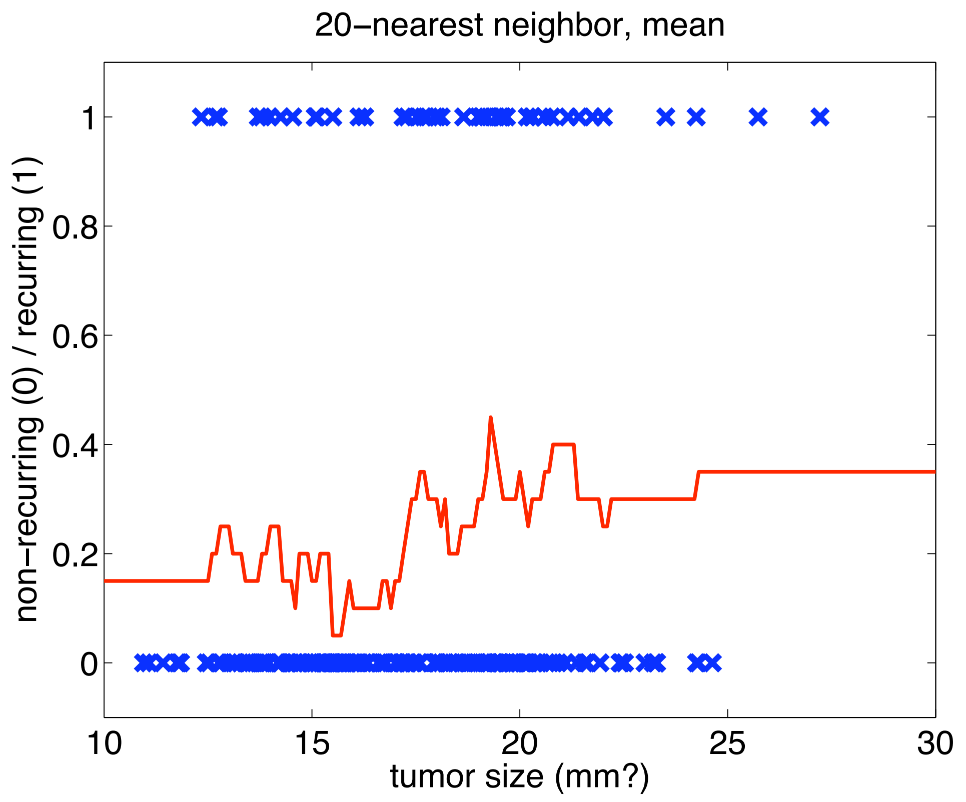
Classification, 10-nearest neighbor



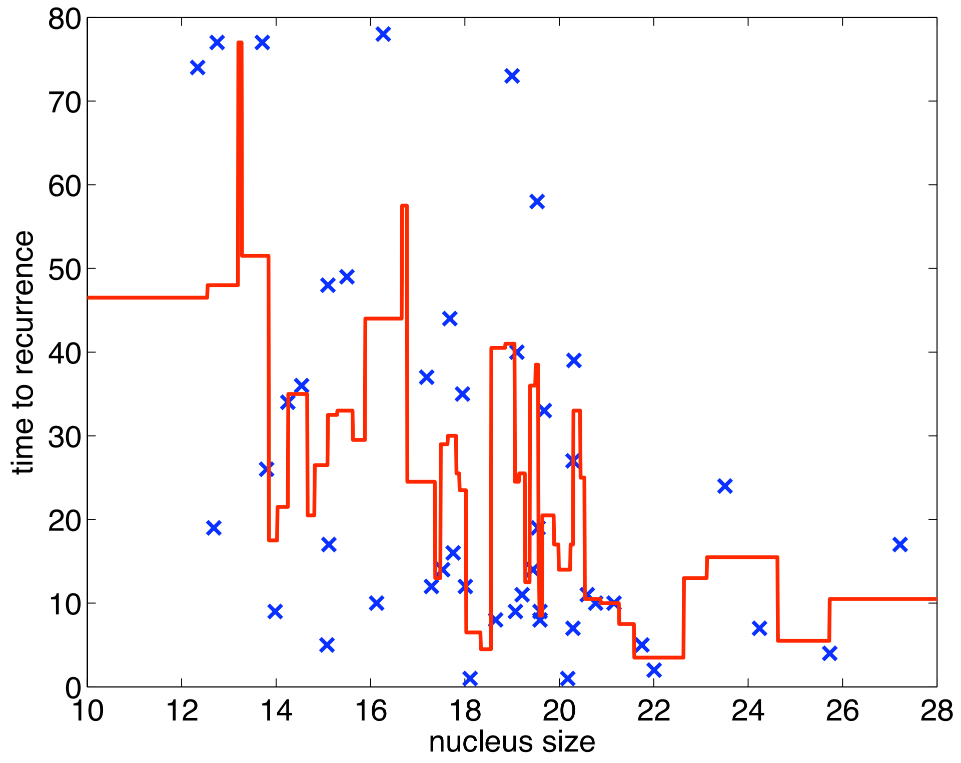
Classification, 15-nearest neighbor



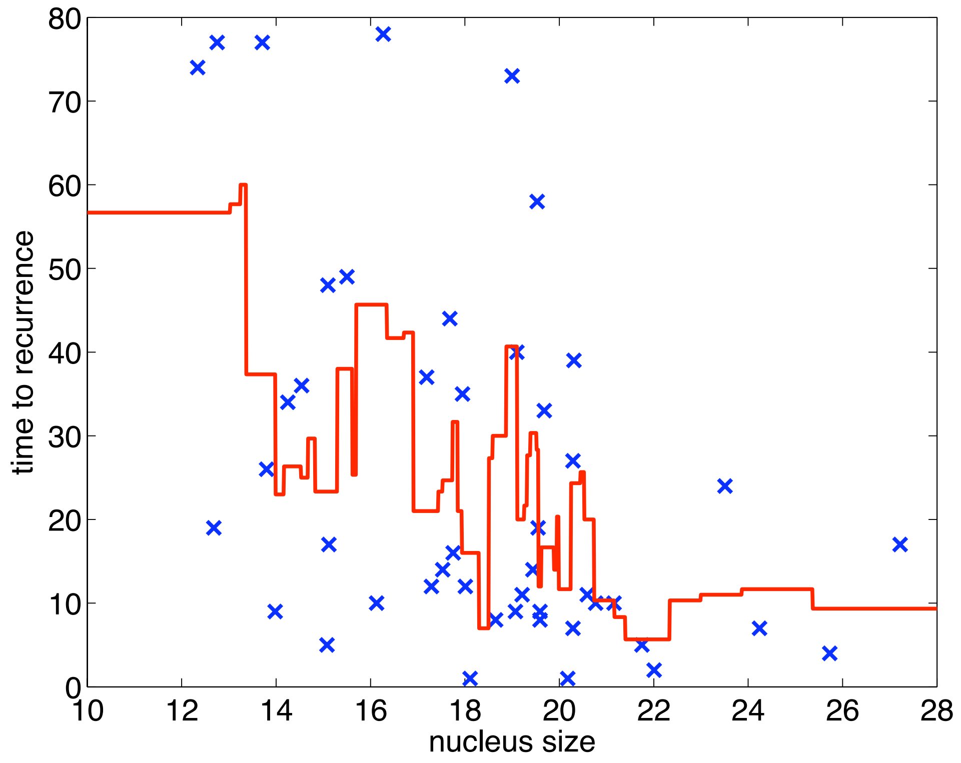
Classification, 20-nearest neighbor



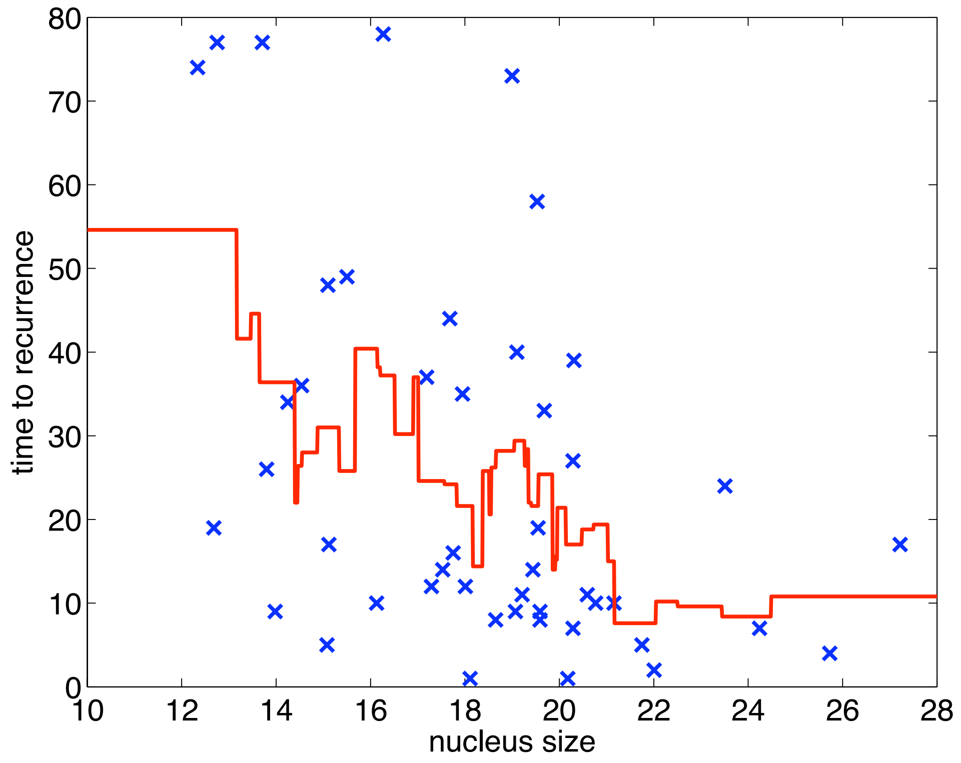
Regression, 2-nearest neighbor, mean prediction



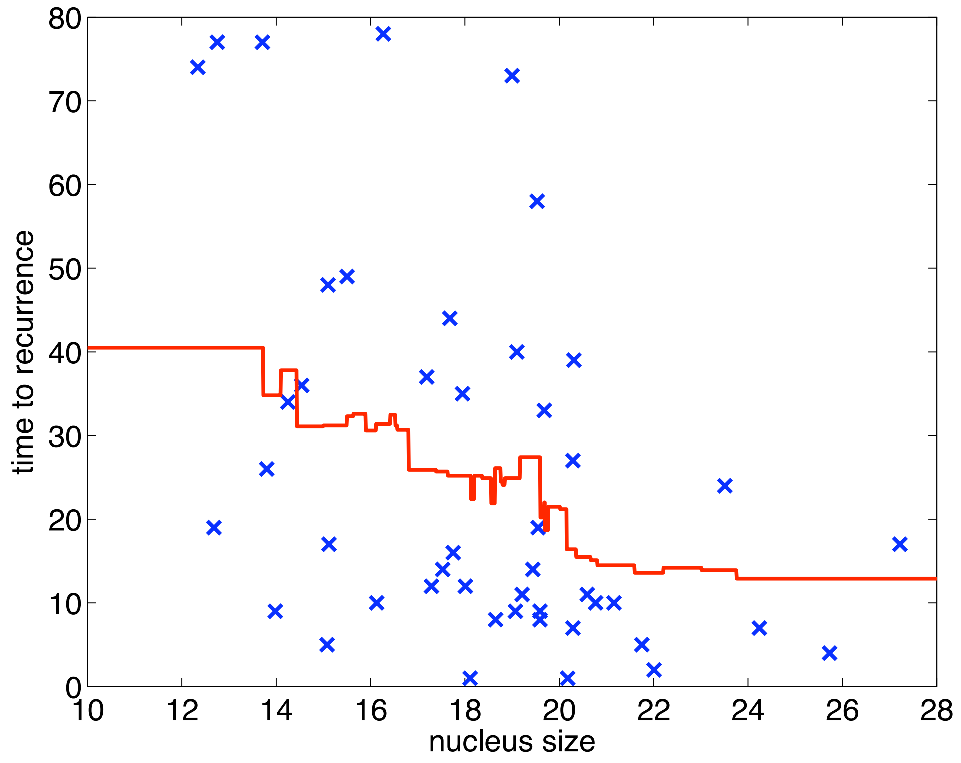
Regression, 3-nearest neighbor



Regression, 5-nearest neighbor



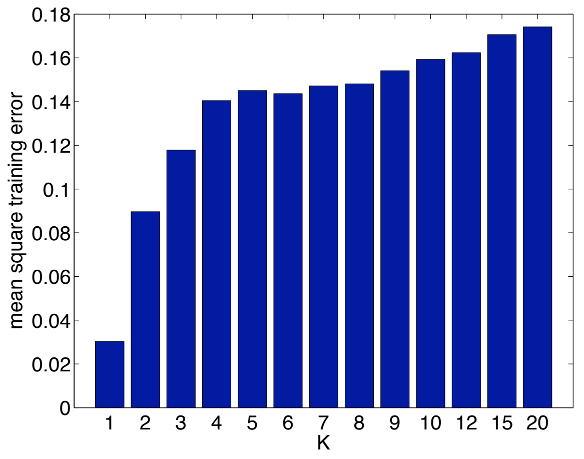
Regression, 10-nearest neighbor

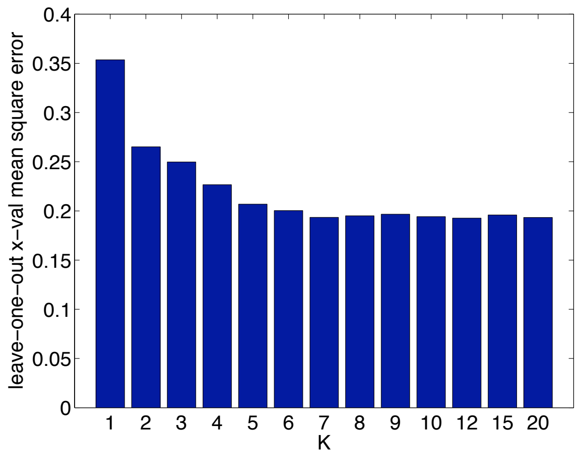


Bias-variance trade-off

- If k is low, very non-linear functions can be approximated, but we also capture the noise in the data
  Bias is low, variance is high
- If k is high, the output is much smoother, less sensitive to data variation
  High bias, low variance
- A validation set can be used to pick the best k

Choosing k





- Pick the best value according to the error on the validation set
- Makes the training more expensive, but results are typically much better

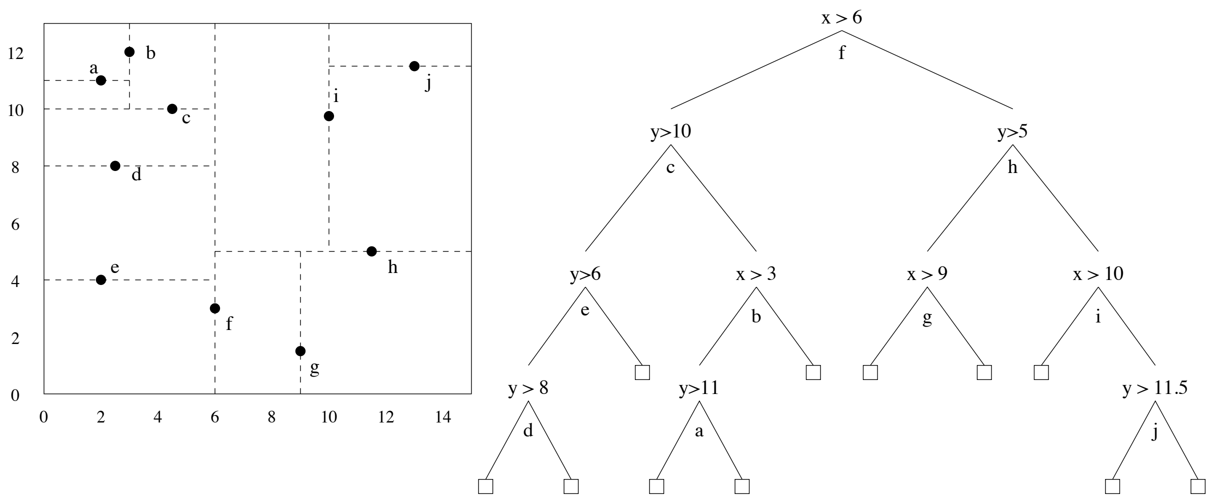
Improving query efficiency

- If the data set is large, searching through all points to compute the set of nearest neighbors is very slow
- Possible solutions:
  - Condensation of the data set
  - Hash tables in which the hashing function is based on the distance metric
  - kd-trees

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.
- Heuristic; go through the data set, if a point is classified correctly do nothing, otherwise add it to the “condensed” set
- More generally: dictionary methods try to determine a subset of data that is worth keeping

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

kd-tree search

- Go down the appropriate branches until we find a match - this gives a candidate best distance
- At every node along the path, check if a better distance could have been obtained on a different branch
  Compute intersection of a hypersphere of the candidate distance, centered at the point, with the hyperplane at that node.
- If a better solution is possible, recurse down other branches

Features of kd-trees

- Makes it easy to do 1-nearest neighbor
- To compute weighted nearest-neighbor efficiently, we can leave out some neighbors, if their influence on the prediction will be small
- But the tree needs to be restructured periodically if we acquire more data, to keep it balanced

Problems with k-nearest neighbor

- A lot of discontinuities!
- Sensitive to small variations in the input data

Can we fix this but still keep it (fairly) local?

Distance-weighted (kernel-based) nearest neighbor

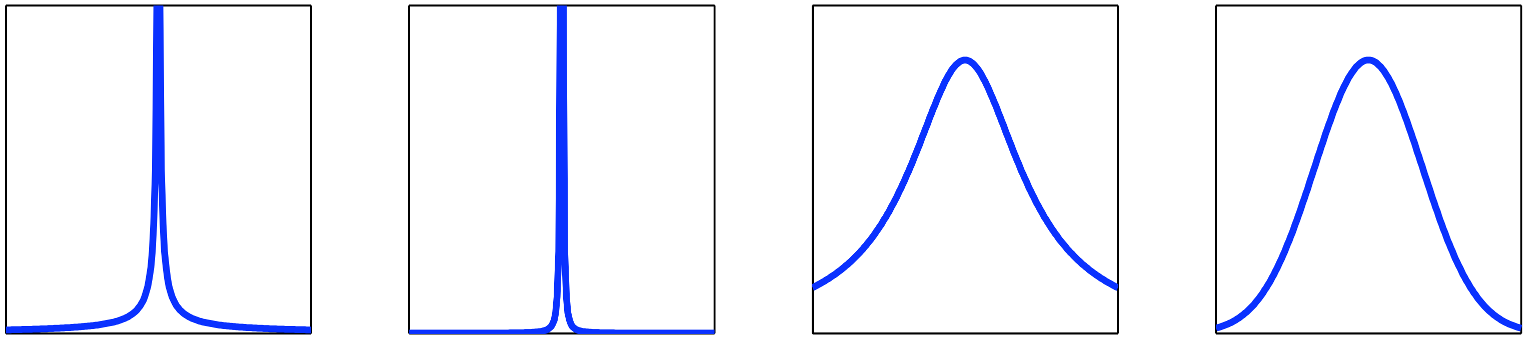
- Inputs: Training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$, distance metric d on $\mathcal{X}$, weighting function $w : \Re \mapsto \Re$.
- Learning: Nothing to do!
- Prediction: On input $\mathbf{x}$,
  - For each i compute $w_i = w(d(\mathbf{x}_i, \mathbf{x}))$.
  - Predict weighted majority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

$$\mathbf{y} = \frac{\sum_i w_i \mathbf{y}_i}{\sum_i w_i}$$

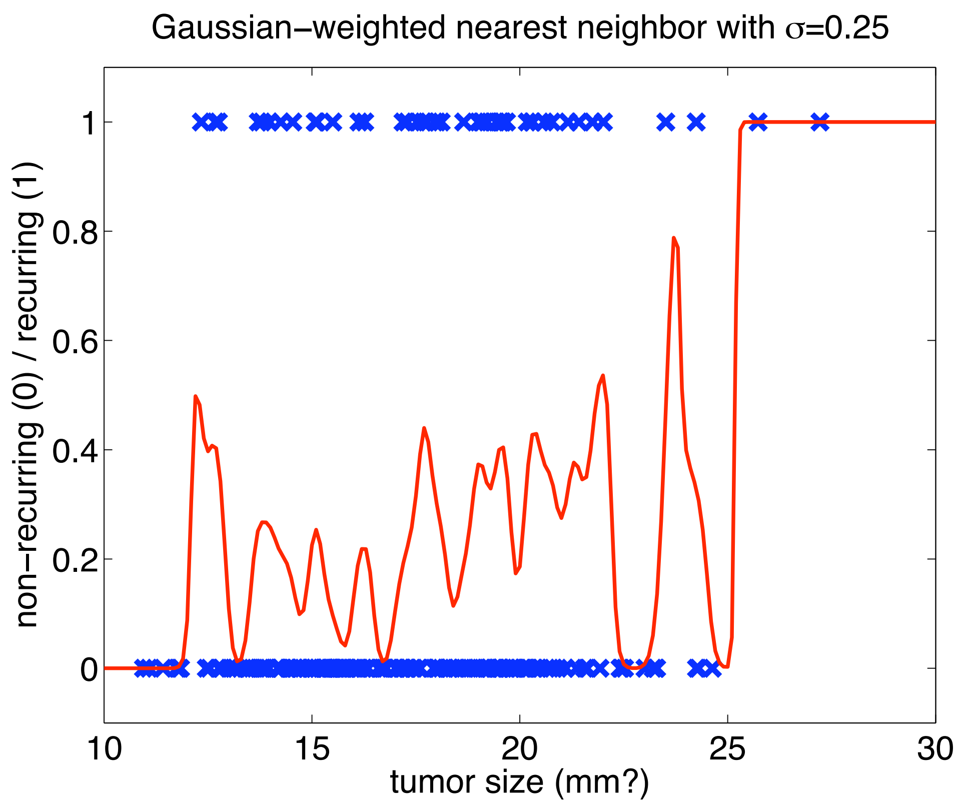
How to weight distances?

Some weighting functions

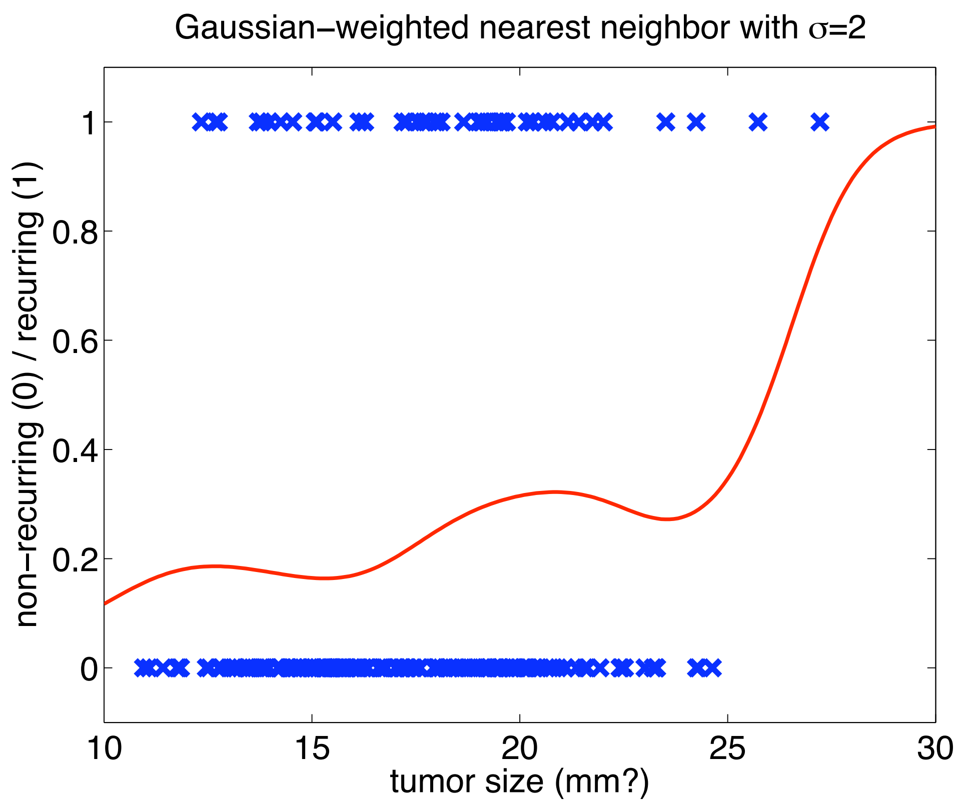
$$\frac{1}{d(\mathbf{x}_i, \mathbf{x})} \qquad \frac{1}{d(\mathbf{x}_i, \mathbf{x})^2} \qquad \frac{1}{c + d(\mathbf{x}_i, \mathbf{x})^2} \qquad e^{-\frac{d(\mathbf{x}_i, \mathbf{x})^2}{\sigma^2}}$$



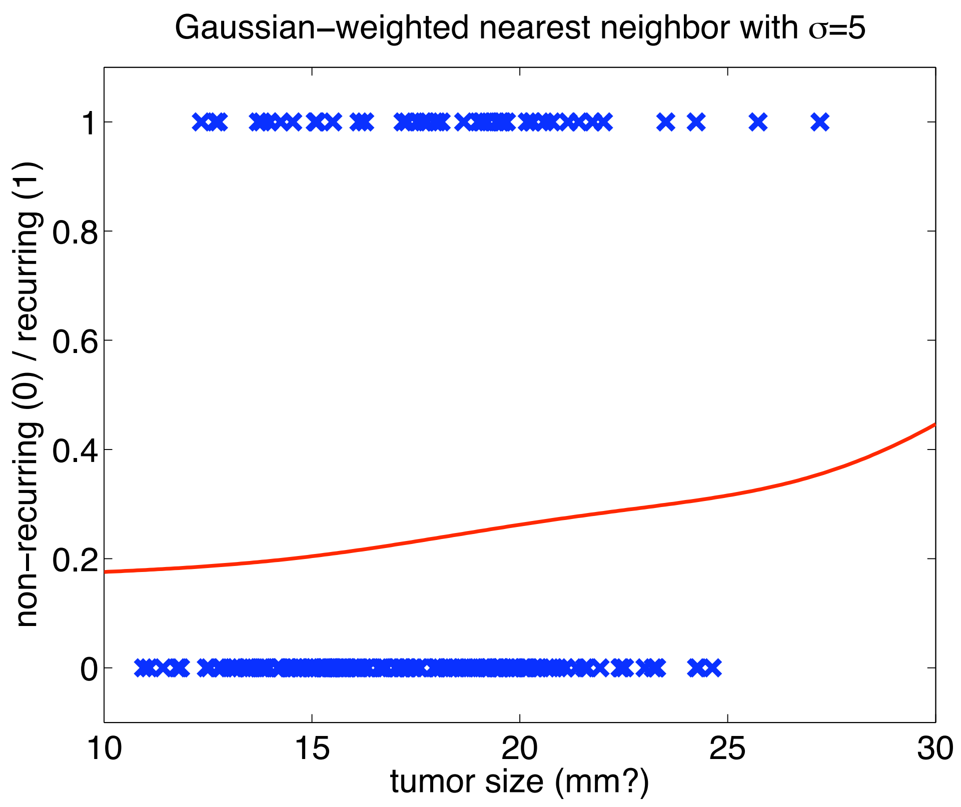
Example: Gaussian weighting, small σ



Gaussian weighting, medium σ



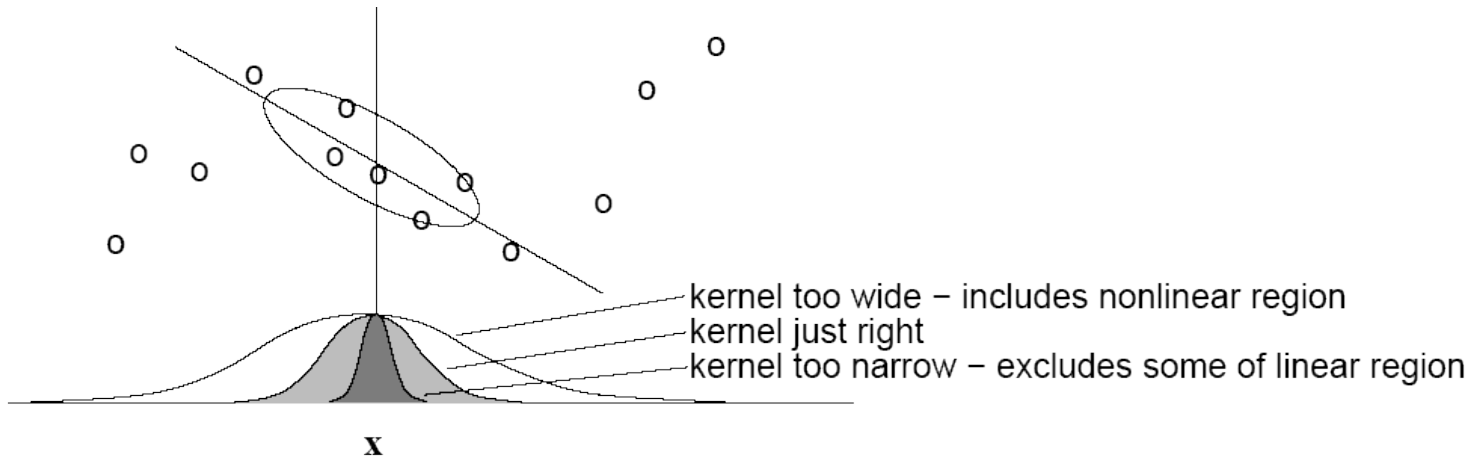
Gaussian weighting, large σ



All examples get to vote! Curve is smoother, but perhaps too smooth.

Locally-weighted linear regression

- Weighted linear regression: different weights in the error function for different points (see homework 1)
- Locally weighted linear regression: weights *depend on the distance* to the query point
- Compared to kernel-based regression: use a linear fit rather than just an average


kernel too wide – includes nonlinear region
kernel just right
kernel too narrow – excludes some of linear region
x

Lazy and eager learning

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

Does it matter?

Pros and cons of lazy and eager learning

- Eager learner must create global approximation
- Lazy learner can create many local approximations
- If they use same hypothesis space H, a lazy learner can represent more complex functions (e.g., consider H = linear functions)
- Eager learner does the work off-line, summarizes lots of data with few parameters
- Lazy learner has to do lots of work sifting through the data at query time
- Typically lazy learners take longer time to answer queries and require more space

When to consider instance-based learning

- Instances map to points in $\mathbb{R}^n$
- Not too many attributes per instace (< 20)
- 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 lots of inputs