CS 886
Applied Machine Learning
Non-Linear Classifiers

Dan Lizotte

University of Waterloo

25 Sept 2012

Instance-based learning, Decision Trees

- Non-parametric learning
- k-nearest neighbour
- Efficient implementations
- Variations

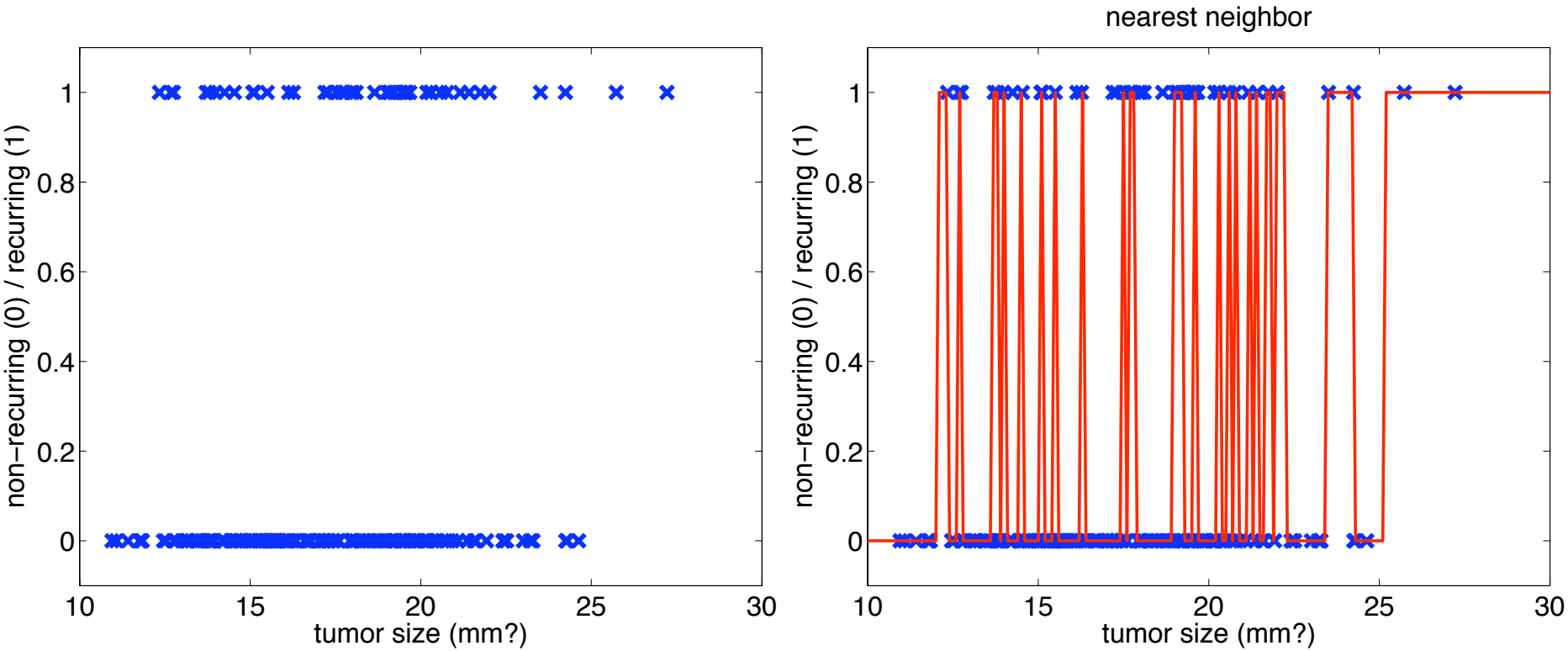
Parametric supervised learning

- So far, we have assumed that we have a data set D of labeled examples
- From this, we learn a *parameter vector* of a *fixed size* 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

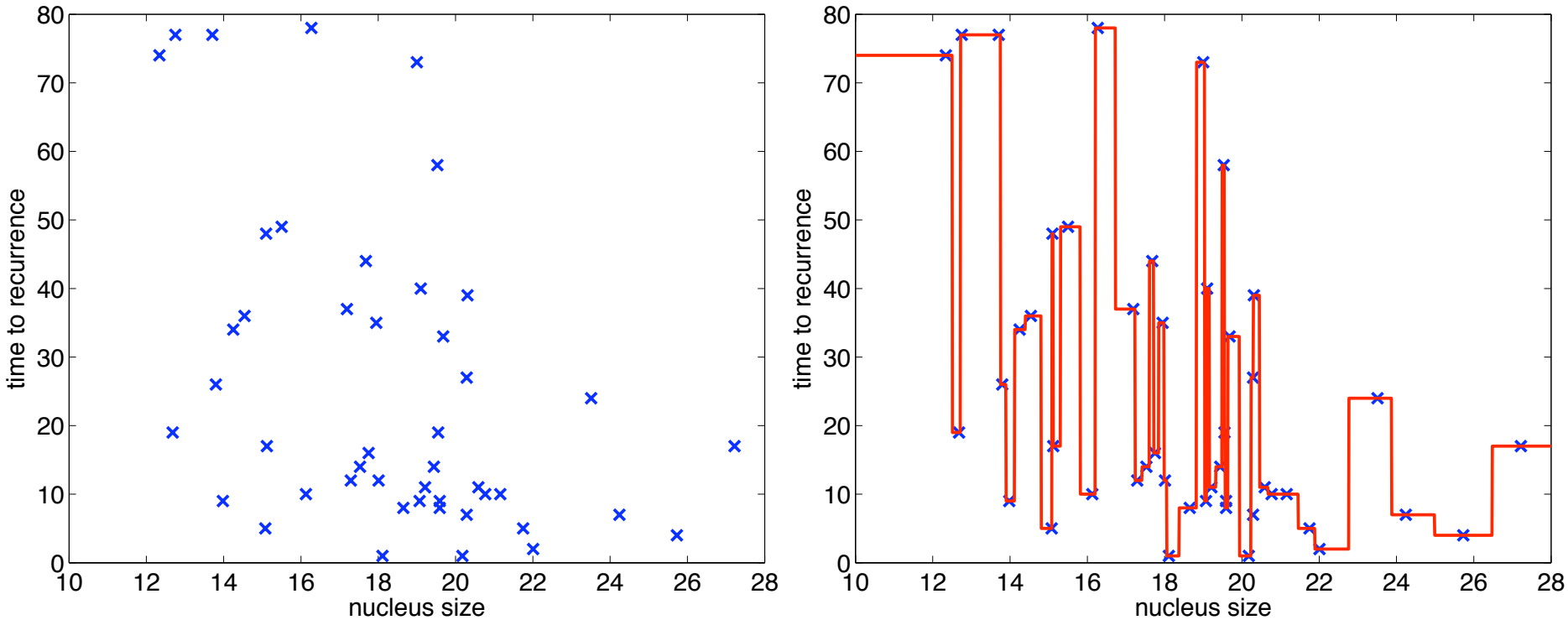
- Key idea: just store all training examples $\langle \mathbf{x}_i, y_i \rangle$
- When a query is made, compute the value of the new instance based on the values of the *closest (most similar) points*
- Requirements:
  - A distance function
  - How many closest points (neighbors) to look at?
  - How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

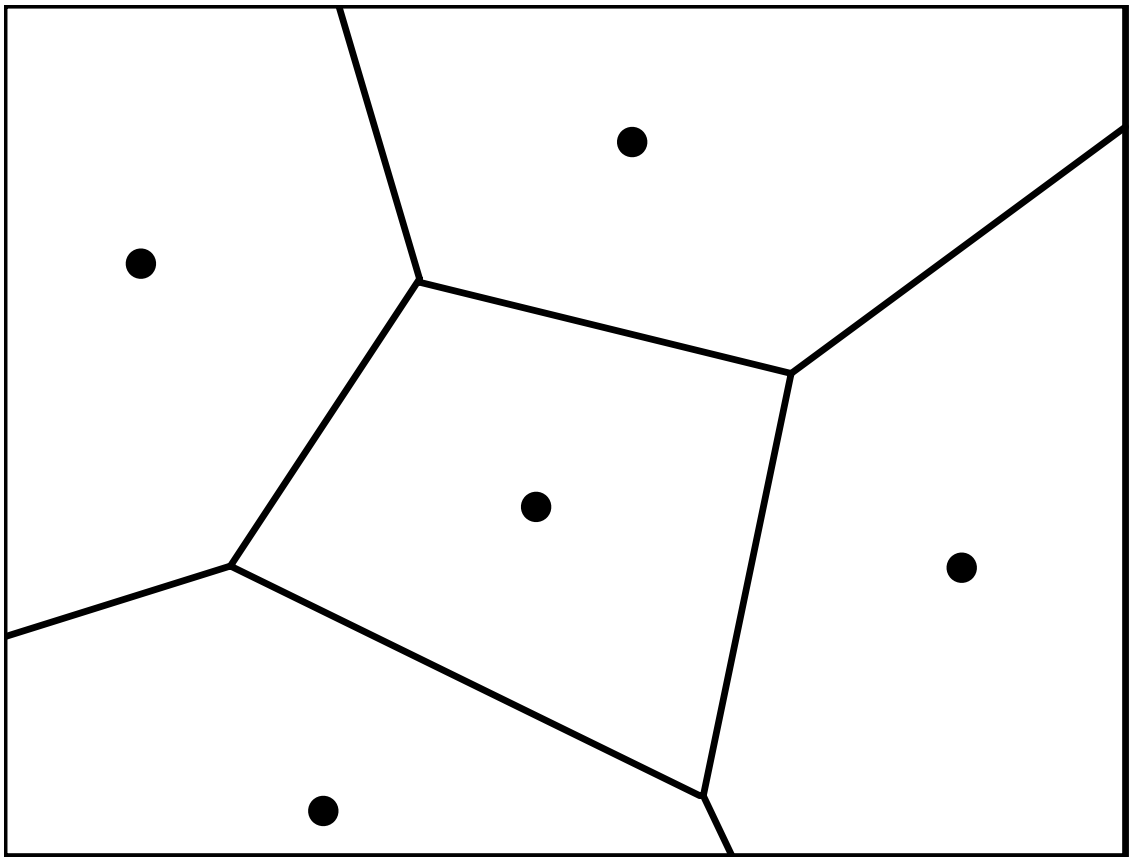
- Given: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, distance metric d on $\mathcal{X}$.
- Training: Nothing to do! (just store data)
- Prediction: for $\mathbf{x} \in \mathcal{X}$
  - Find nearest training sample to $\mathbf{x}$.

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

  - Predict $y = y_{i^*}$.

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.

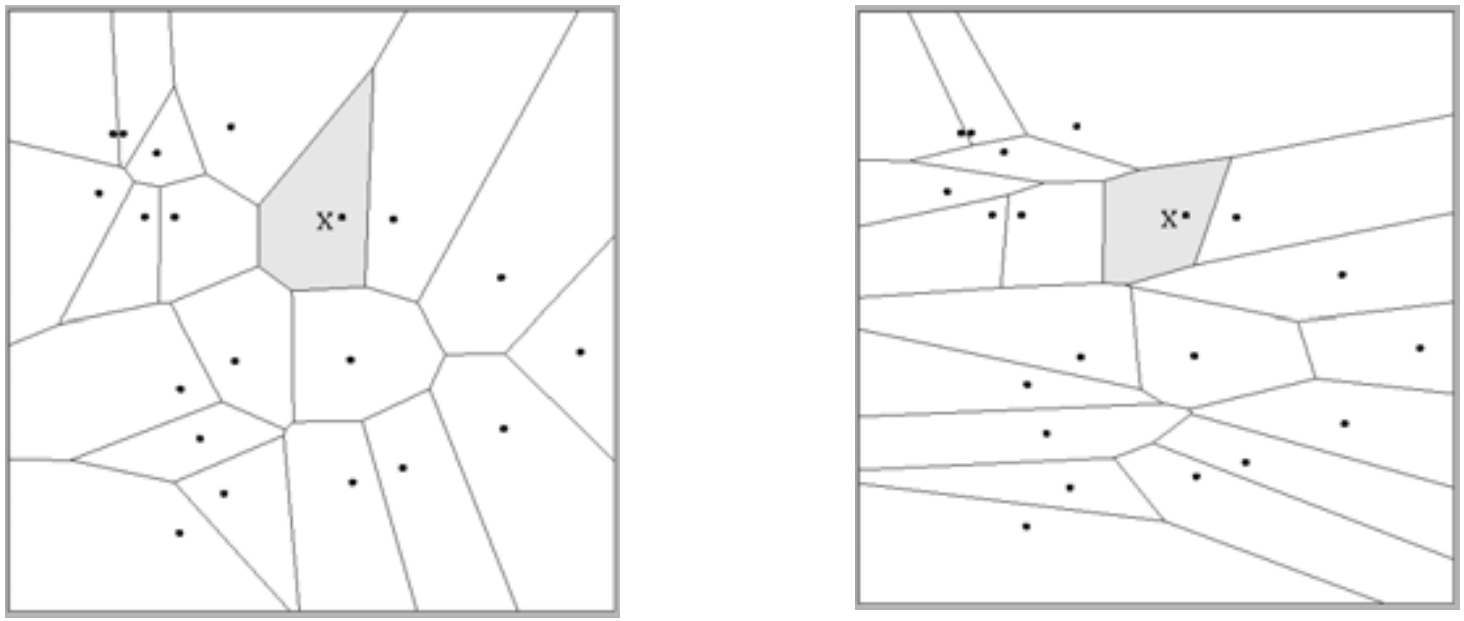
What kind of distance metric?

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

$$d(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{n} u_j (\mathbf{x}_j - \mathbf{x}'_j)^2$$

- An arbitrary distance or similarity function d, specific for the application at hand (works best, if you have one)

Distance metric is really important!



Left: attributes weighted equally Right: unequal weighting

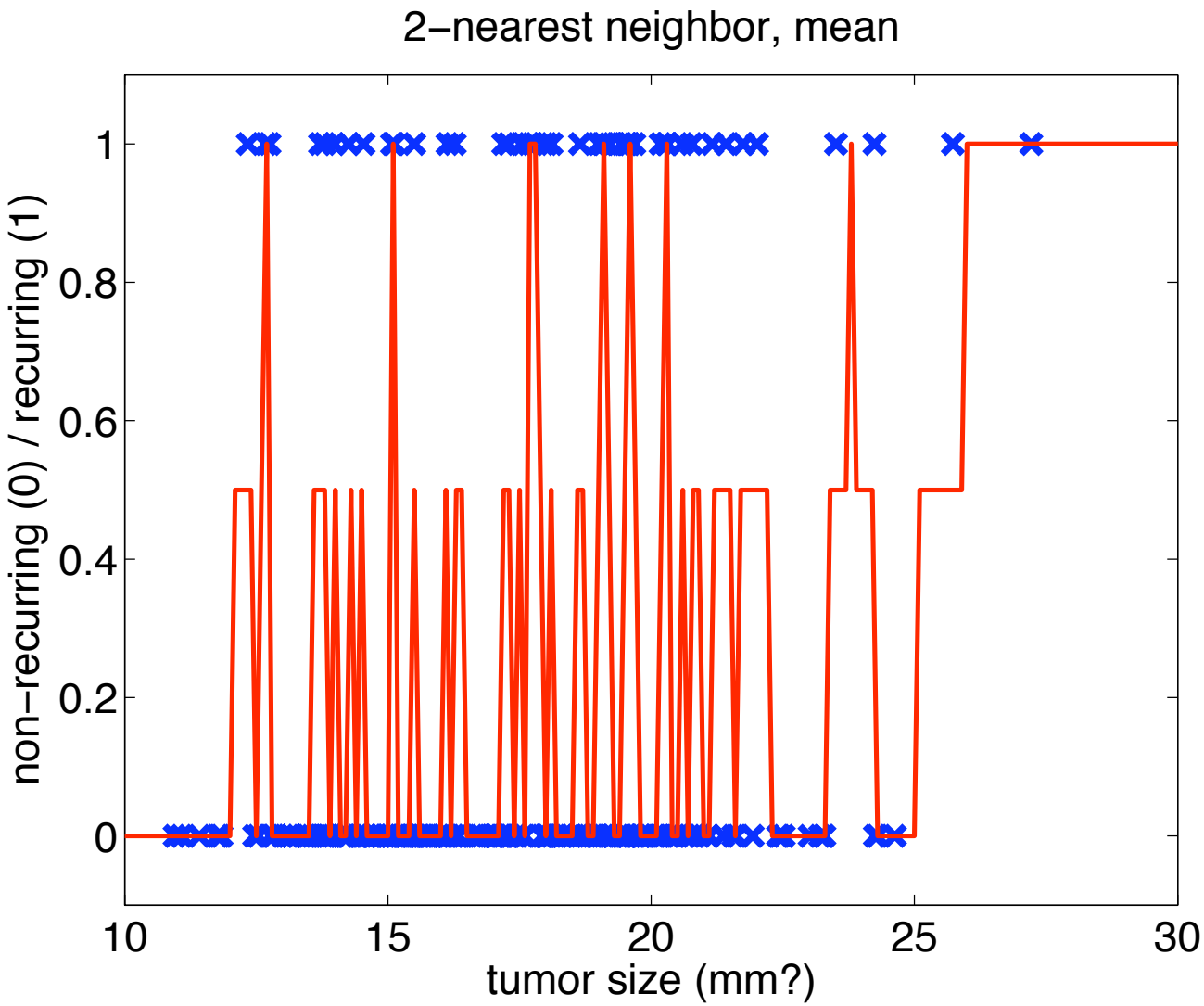
Distance metric tricks

- You may need to do preprocessing:
  - *Scale* the input dimensions (or normalize them)
  - Remove noisy inputs
  - Determine weights for attributes based on cross-validation (or information-theoretic methods)
- Distance metric is often domain-specific
  - E.g. string edit distance in bioinformatics
  - E.g. trajectory distance in time series models for walking data
- Distance metric can be learned sometimes (more on this later)

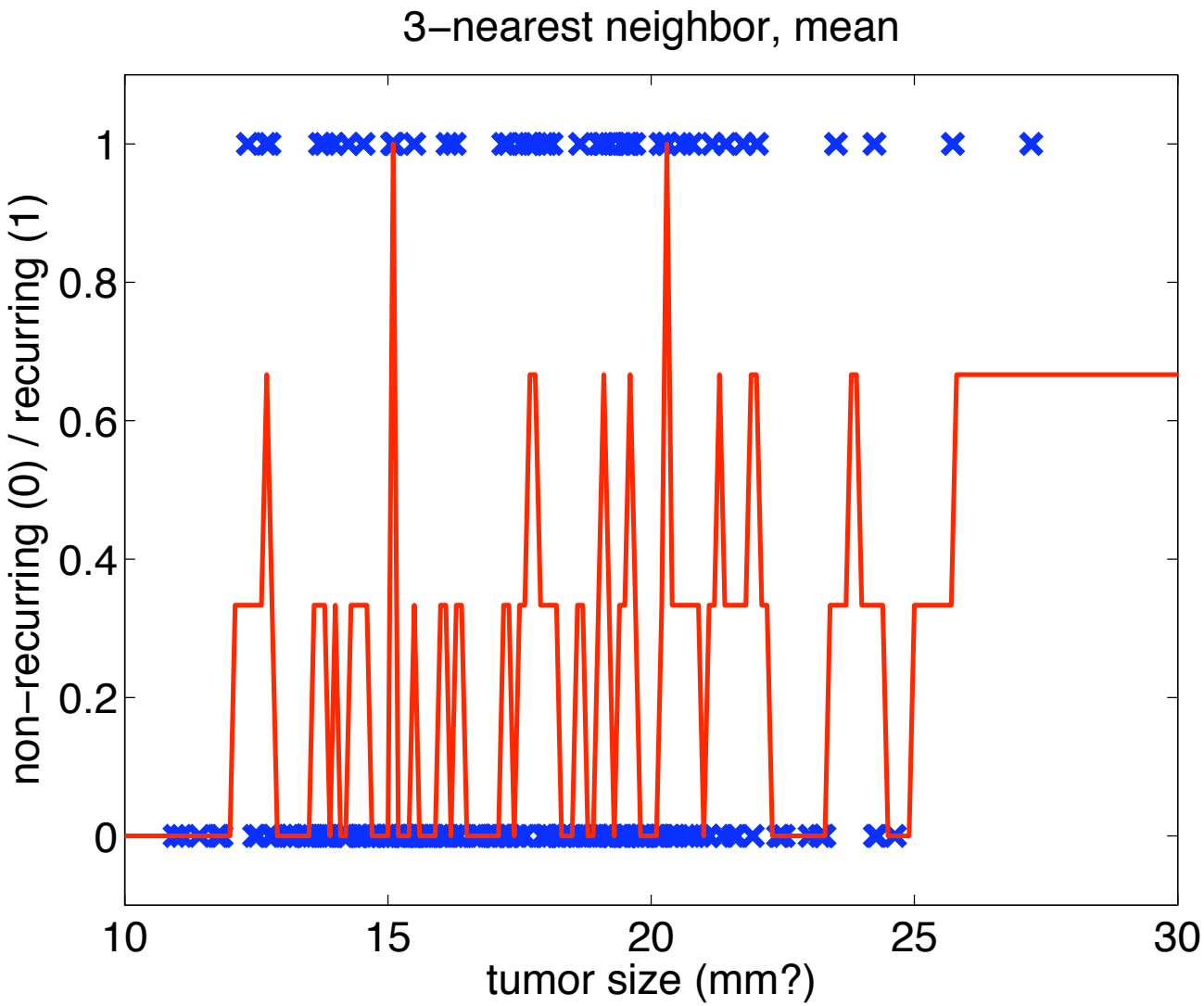
k-nearest neighbor

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

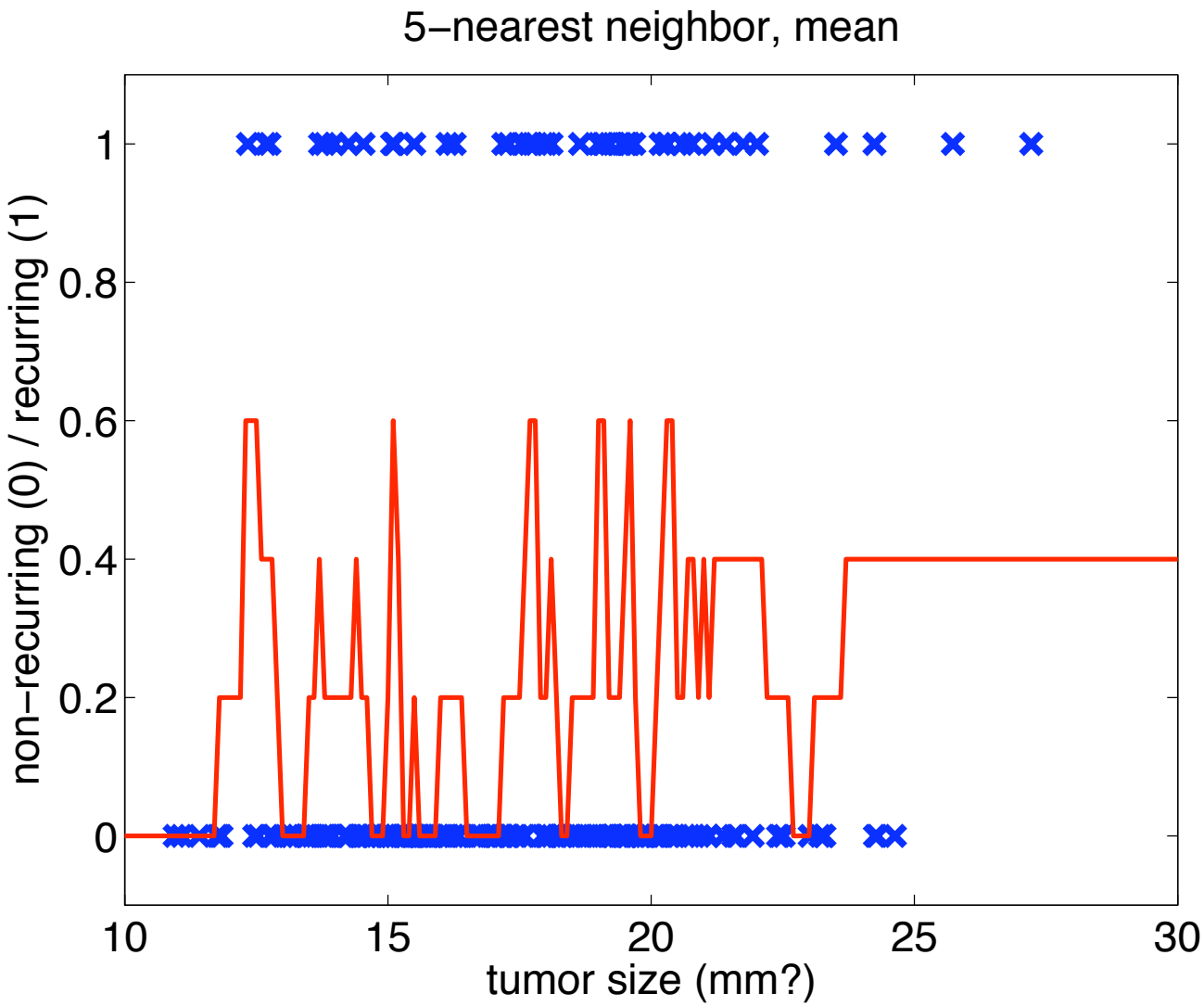
Classification, 2-nearest neighbor, empirical distribution



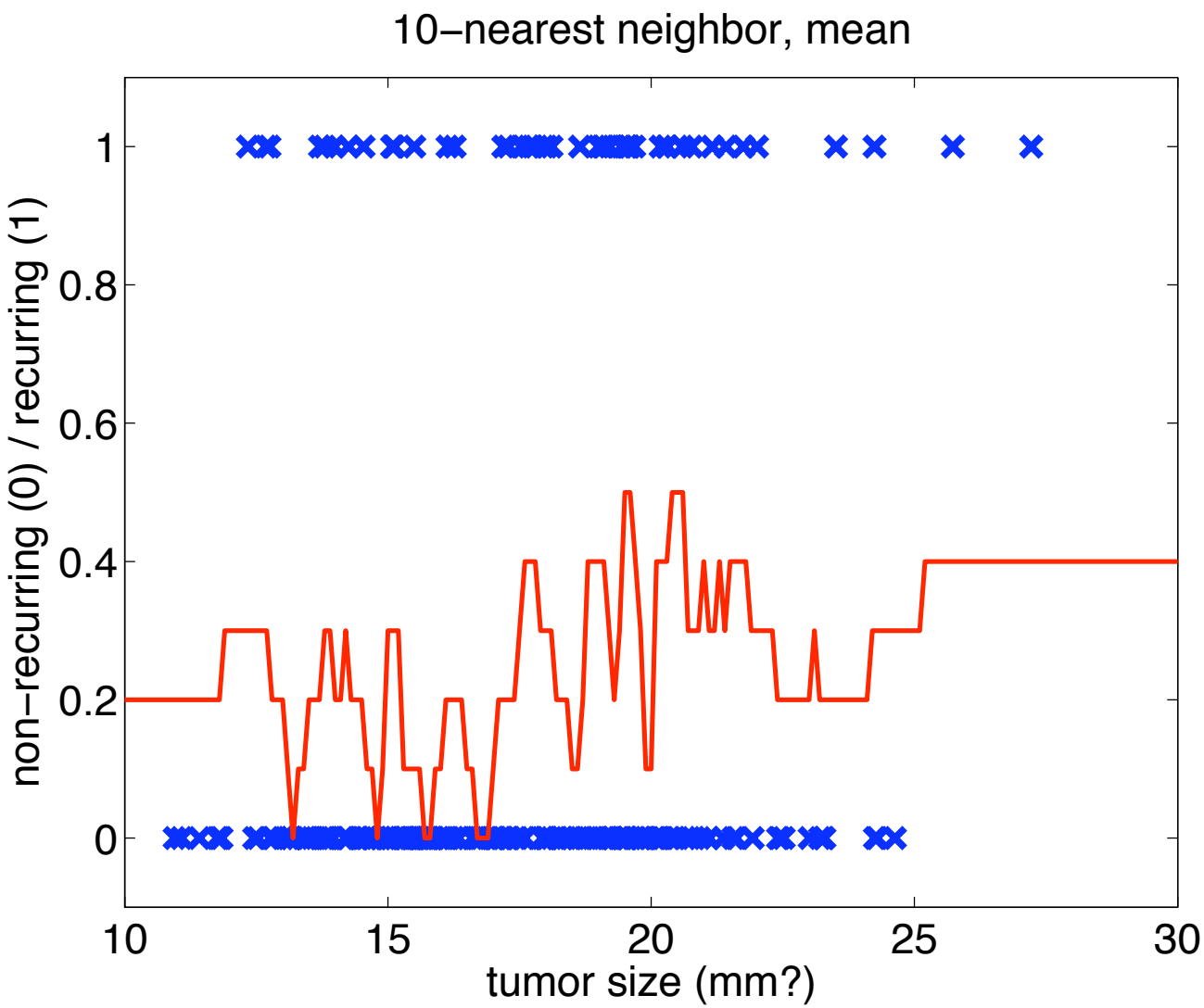
Classification, 3-nearest neighbor



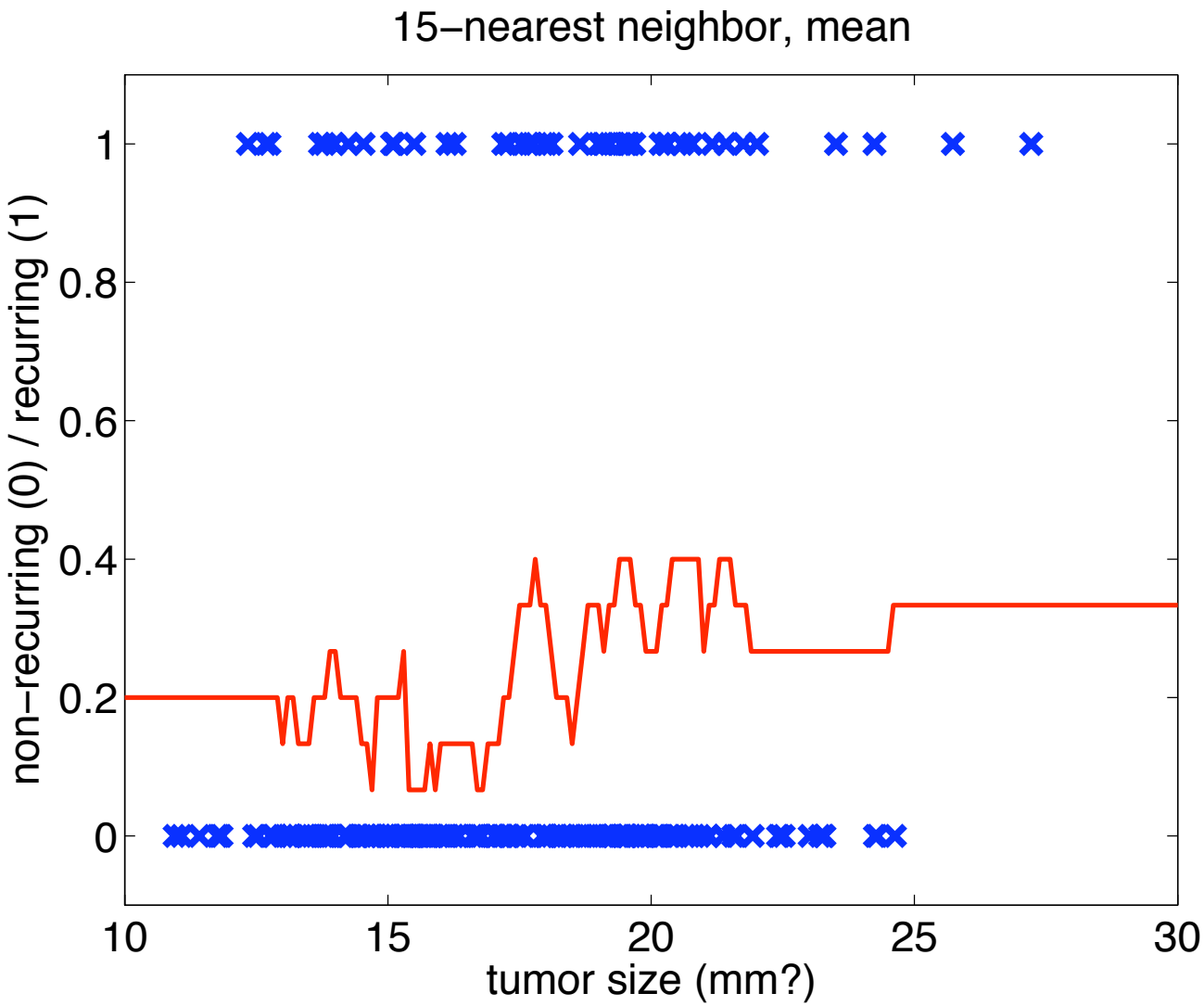
Classification, 5-nearest neighbor



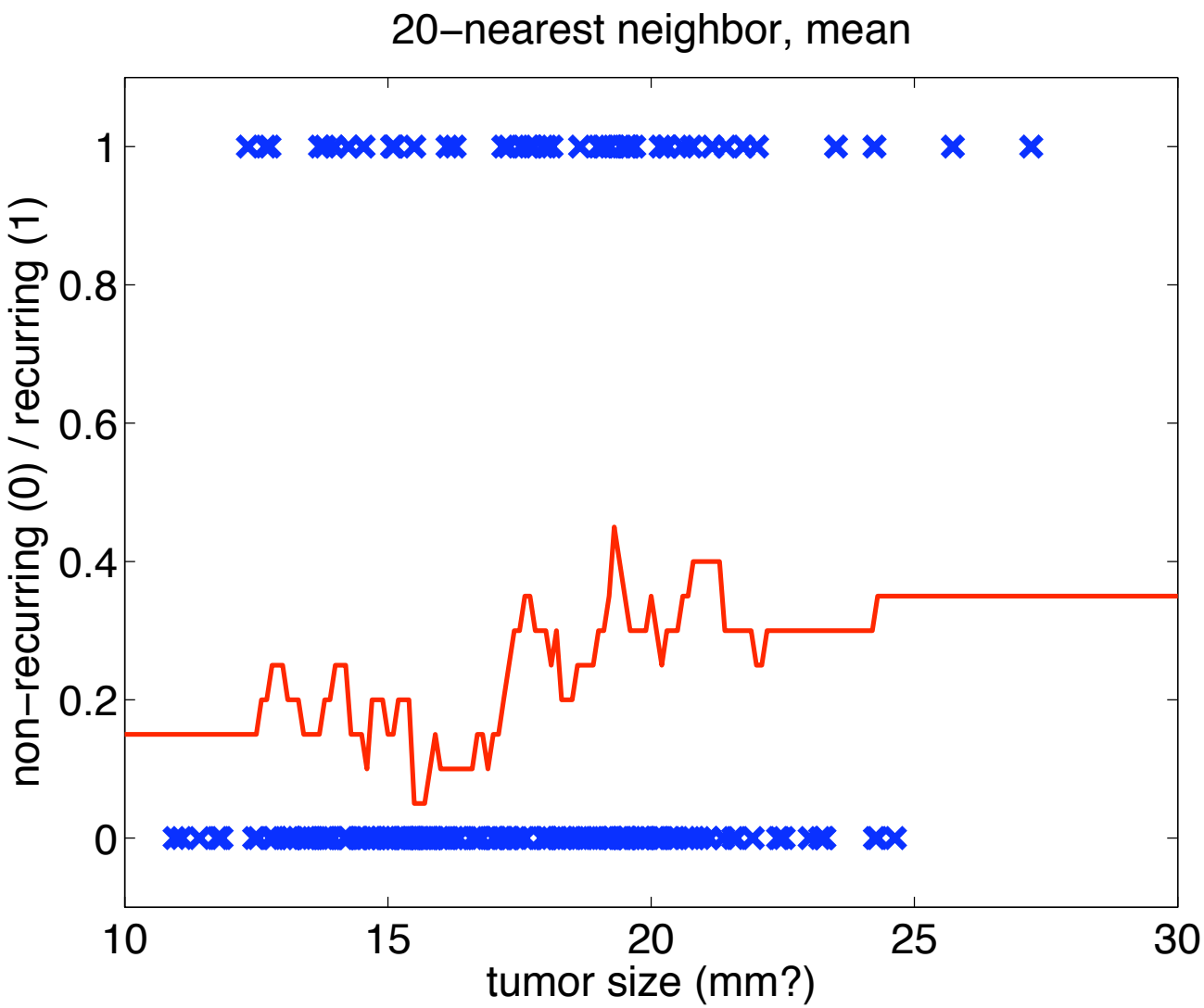
Classification, 10-nearest neighbor



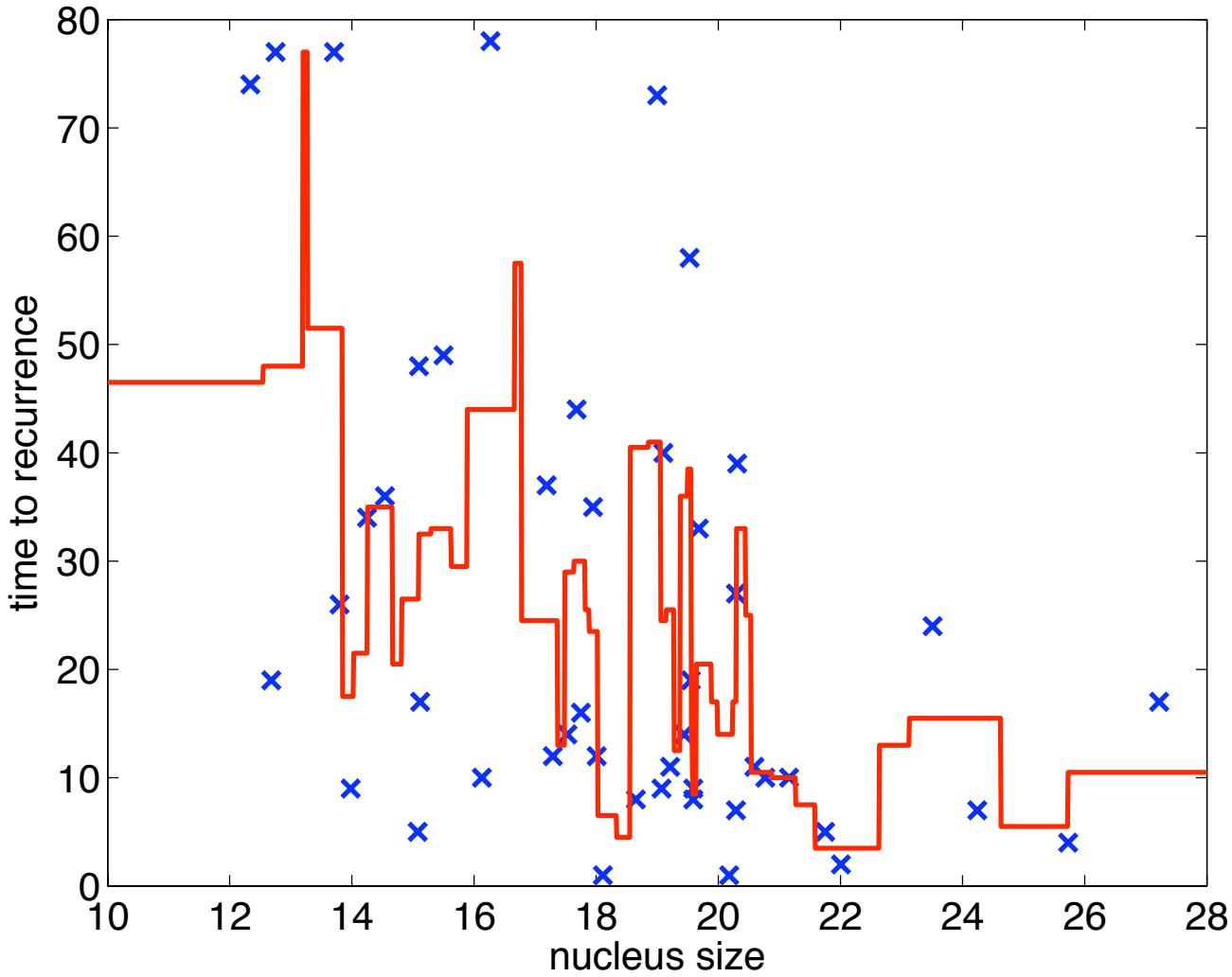
Classification, 15-nearest neighbor



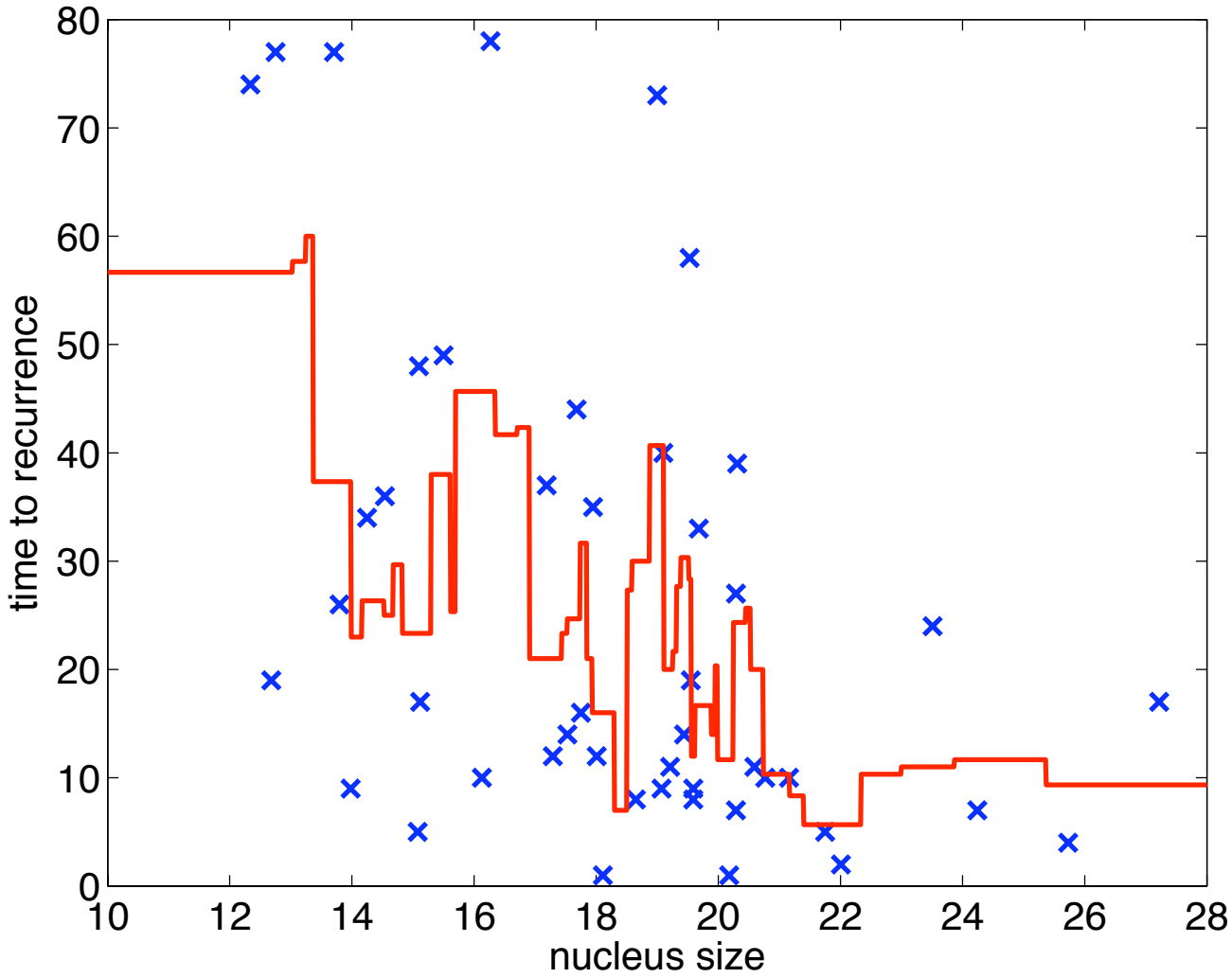
Classification, 20-nearest neighbor



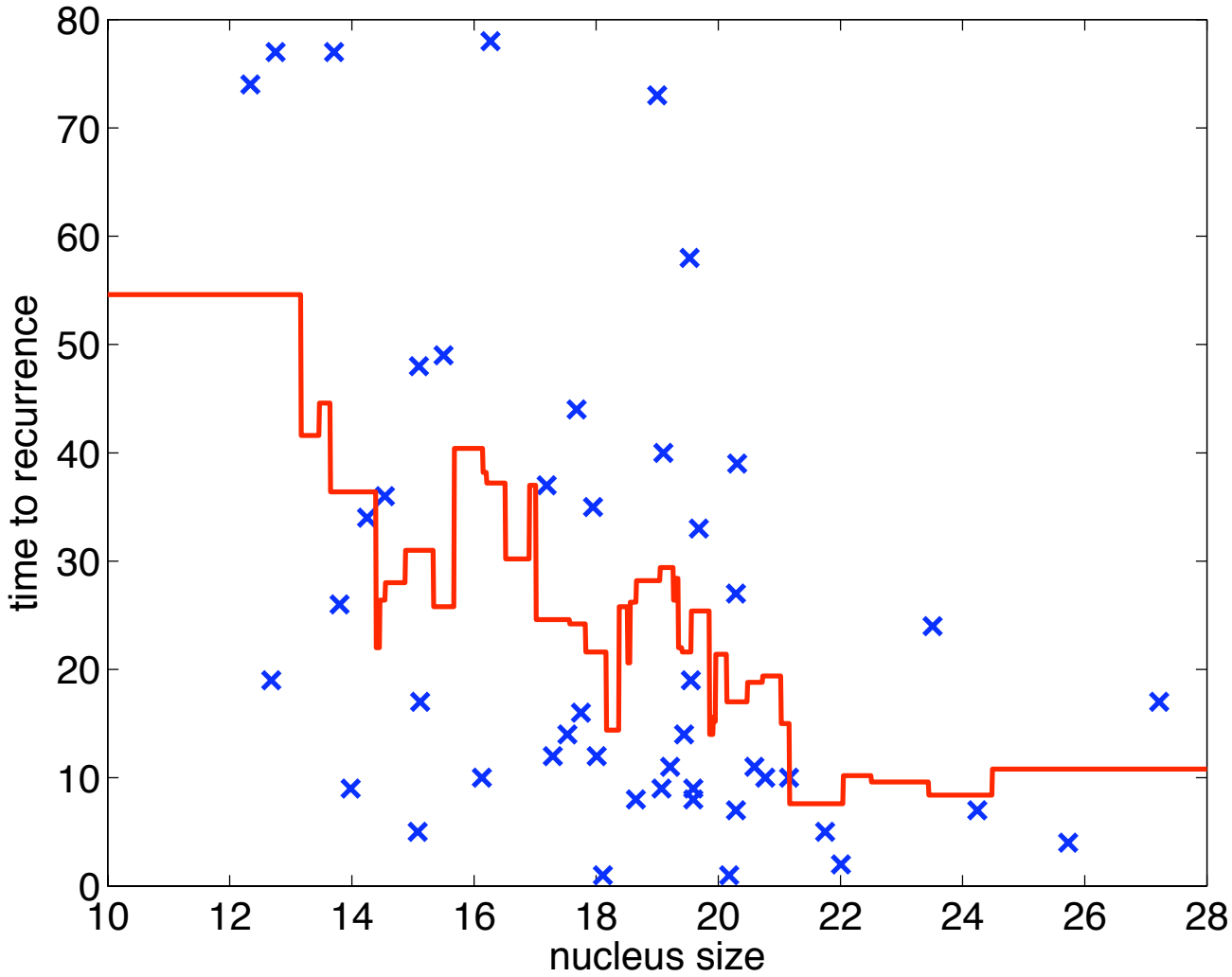
Regression, 2-nearest neighbor, mean prediction



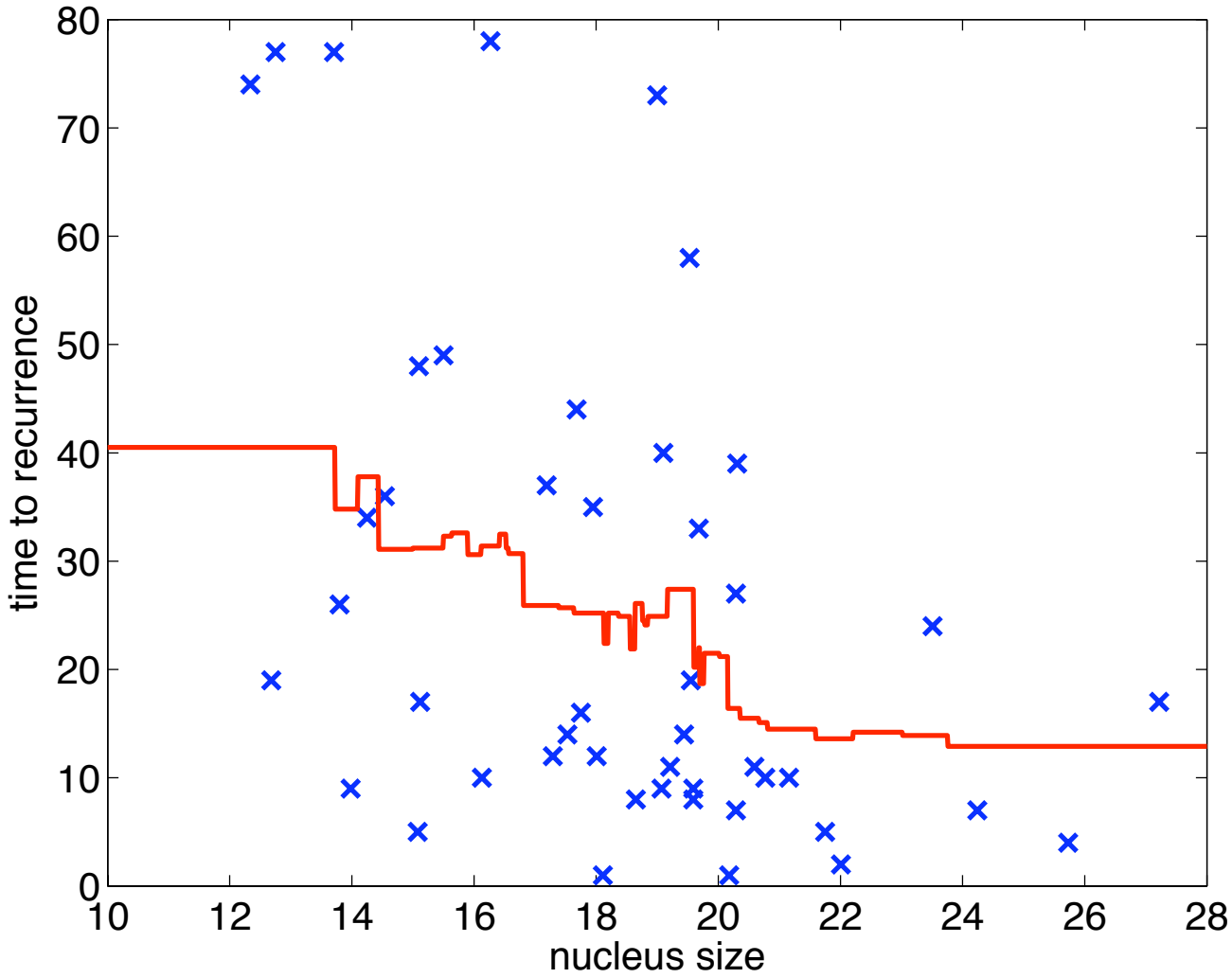
Regression, 3-nearest neighbor



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

Distance-weighted (kernel-based) nearest neighbor

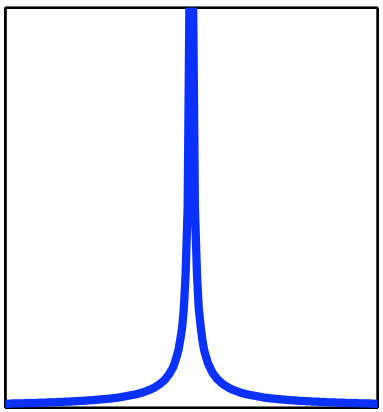
- Inputs: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, distance metric d on $\mathcal{X}$, weighting function $w : \mathbb{R} \mapsto \mathbb{R}$.
- Learning: Nothing to do!
- Prediction: On input $\mathbf{x}$,
  - For each i compute $w_i = w(d(\mathbf{x}_i, \mathbf{x}))$.
  - Predict weighted majority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

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

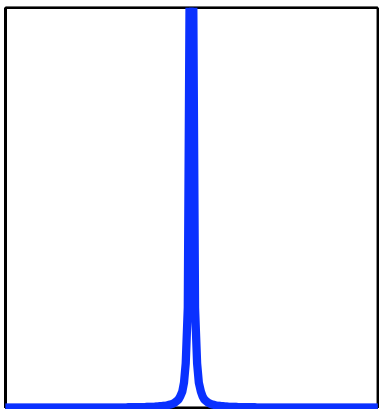
- How to weight distances?

Some weighting functions

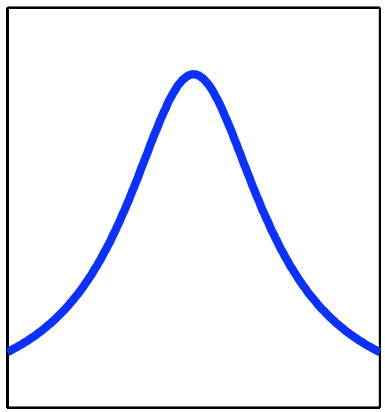
$$\frac{1}{d(\mathbf{x}_i, \mathbf{x})}$$



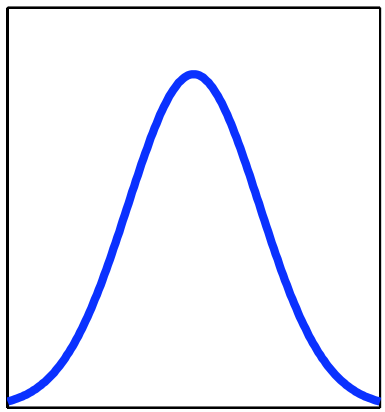
$$\frac{1}{d(\mathbf{x}_i, \mathbf{x})^2}$$



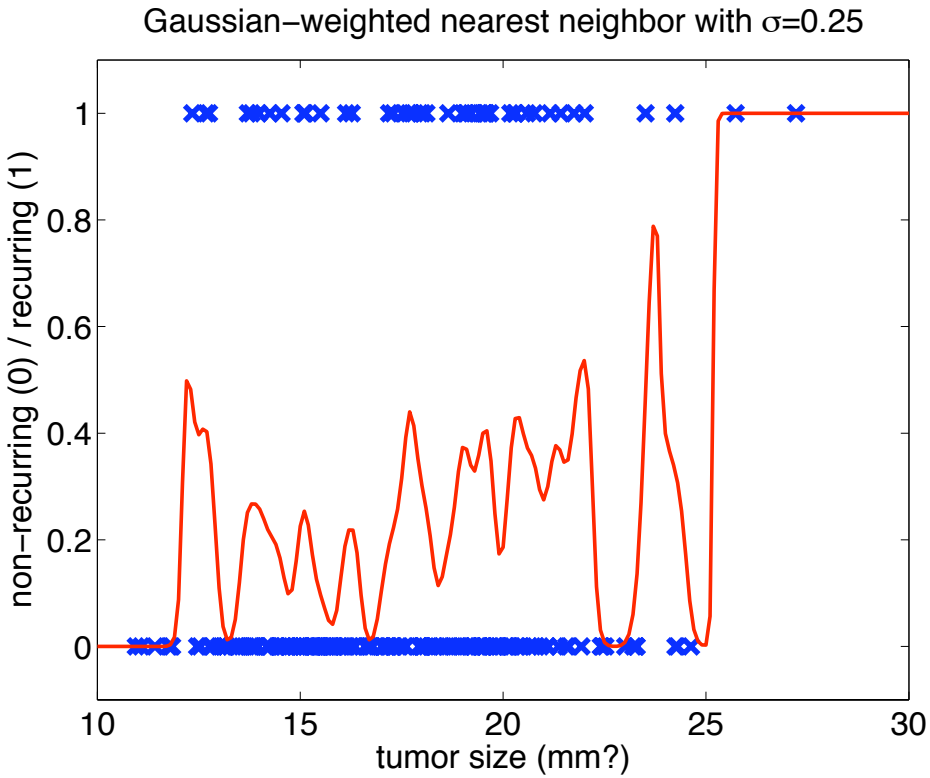
$$\frac{1}{c + d(\mathbf{x}_i, \mathbf{x})^2}$$



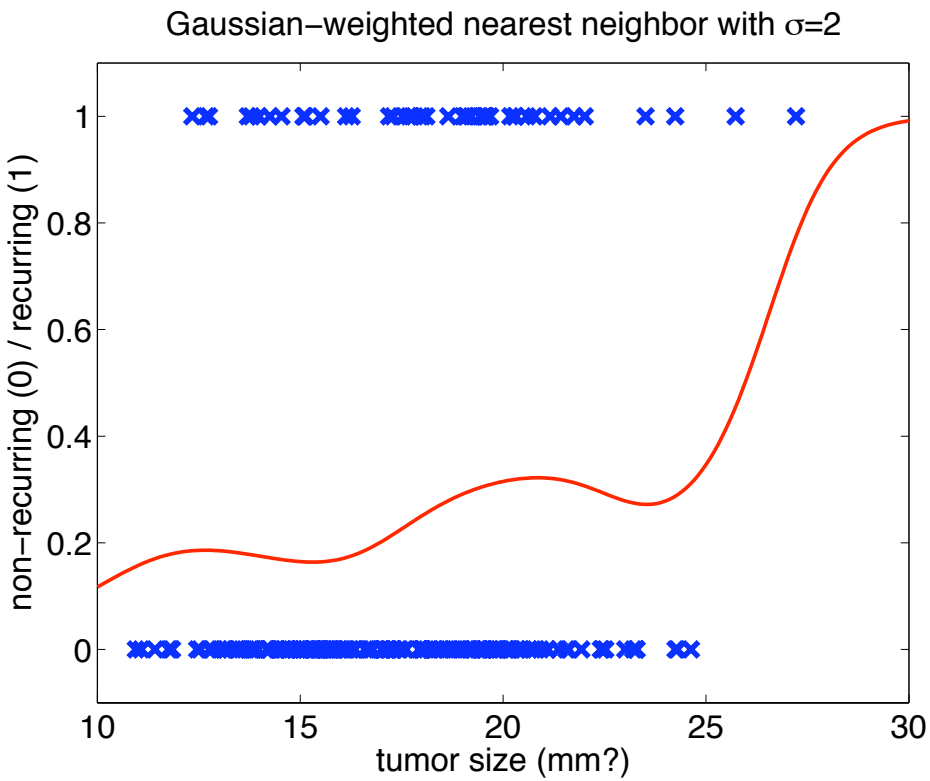
$$e^{-\frac{d(\mathrm{x}_i, \mathrm{x})^2}{\sigma^2}}$$



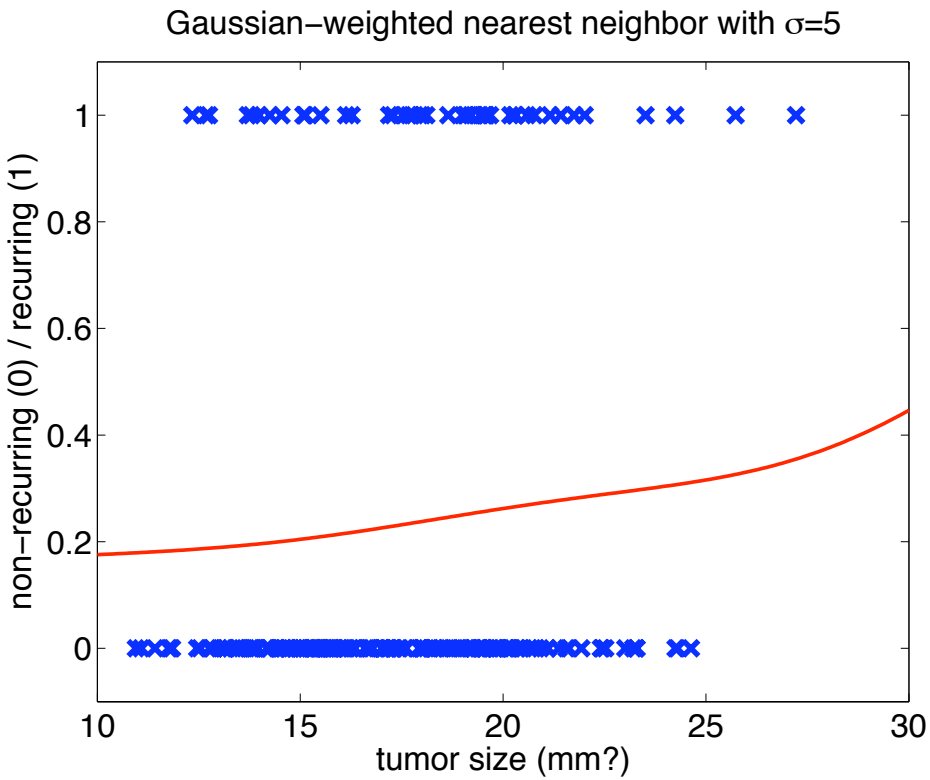
Example: Gaussian weighting, small σ



Gaussian weighting, medium σ



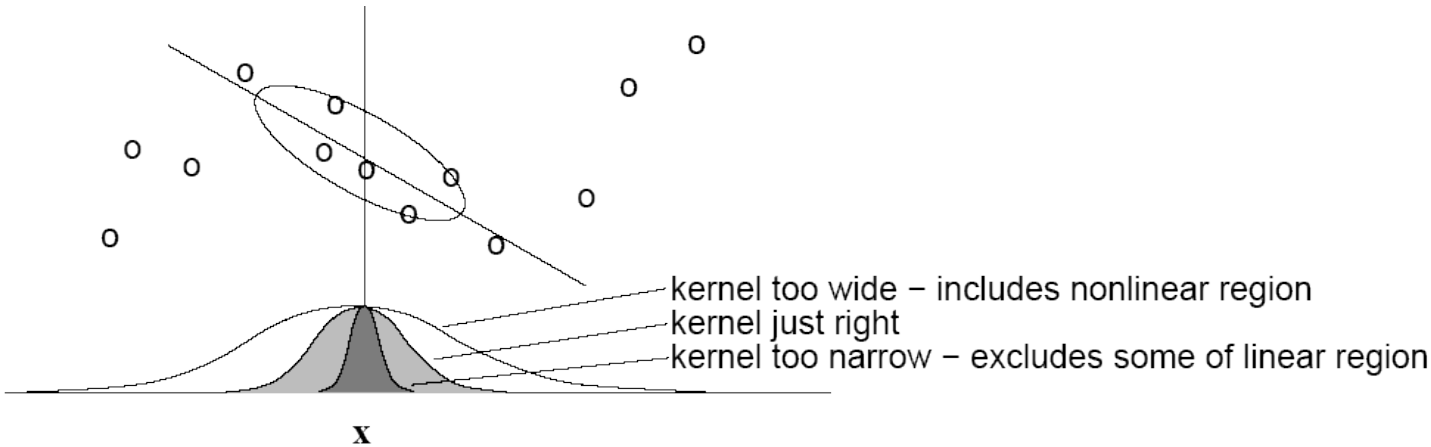
Gaussian weighting, large σ



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*
- Uses a linear fit rather than just an average



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 learners must create global approximation
- Lazy learners 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)
- An eager learner does the work off-line, summarizes lots of data with few parameters
- A 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 $\mathbb{R}^p$
- Not too many attributes per instance (< 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 density of data points)
  - Does not lose any information
  - Often wins if you have enough data
- Disadvantages:
  - Slow at query time
  - Query answering complexity depends on the number of instances
  - *Easily fooled by irrelevant attributes* (for most distance metrics)

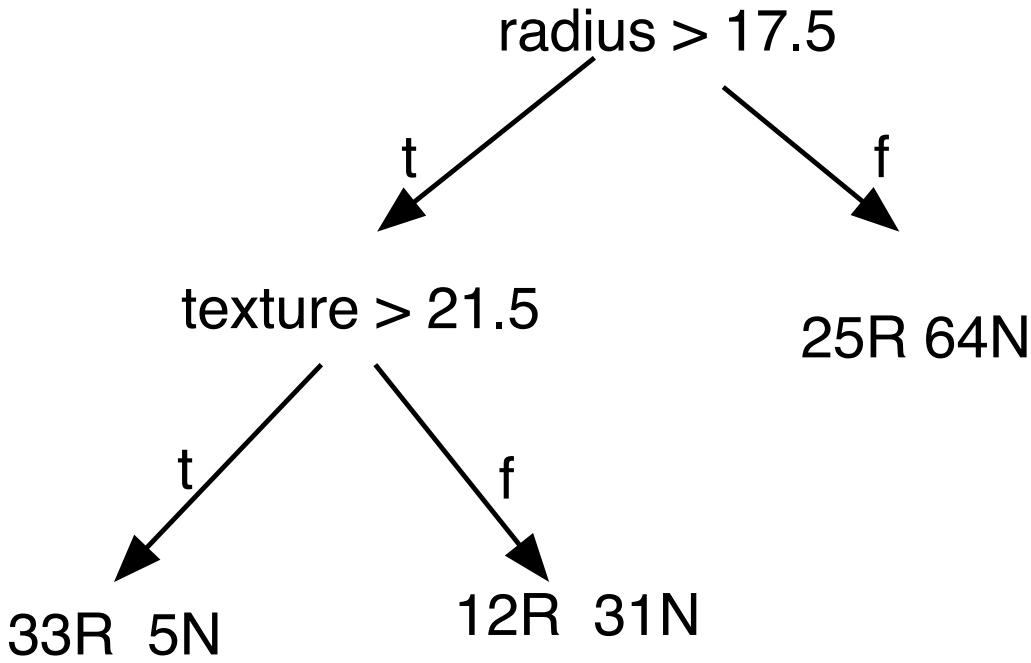
Decision Trees

- What are decision trees?
- Methods for constructing decision trees
- Overfitting avoidance

Non-metric learning

- The result of learning is not a set of parameters, but there is **no distance metric** to assess similarity of different instances
- Typical examples:
  - Decision trees
  - Rule-based systems

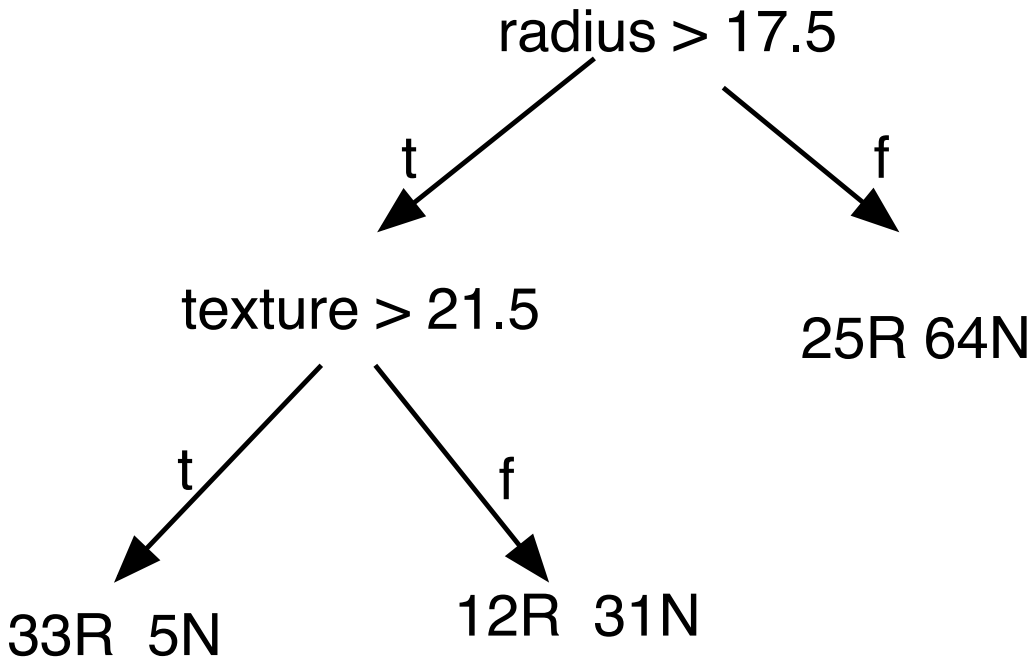
Example: Decision tree for Wisconsin data



- Internal nodes are tests on the values of different attributes
- Tests can be binary or multi-valued
- Each training example $\langle \mathbf{x}_i, y_i \rangle$ falls in precisely one leaf.

Using decision trees for classification

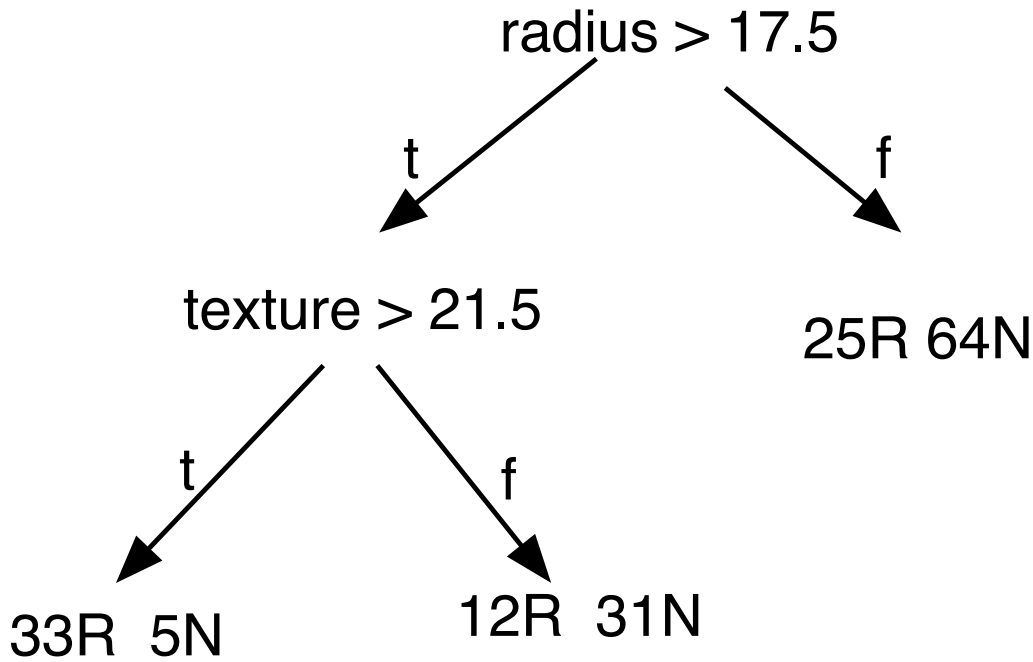
How do we classify a new a new instance, e.g.: *radius=18, texture=12, . . .*



- At every node, test the corresponding attribute
- Follow the appropriate branch of the tree
- At a leaf, one can predict the class of the majority of the examples for the corresponding leaf, or the probabilities of the two classes.

Decision trees as logical representations

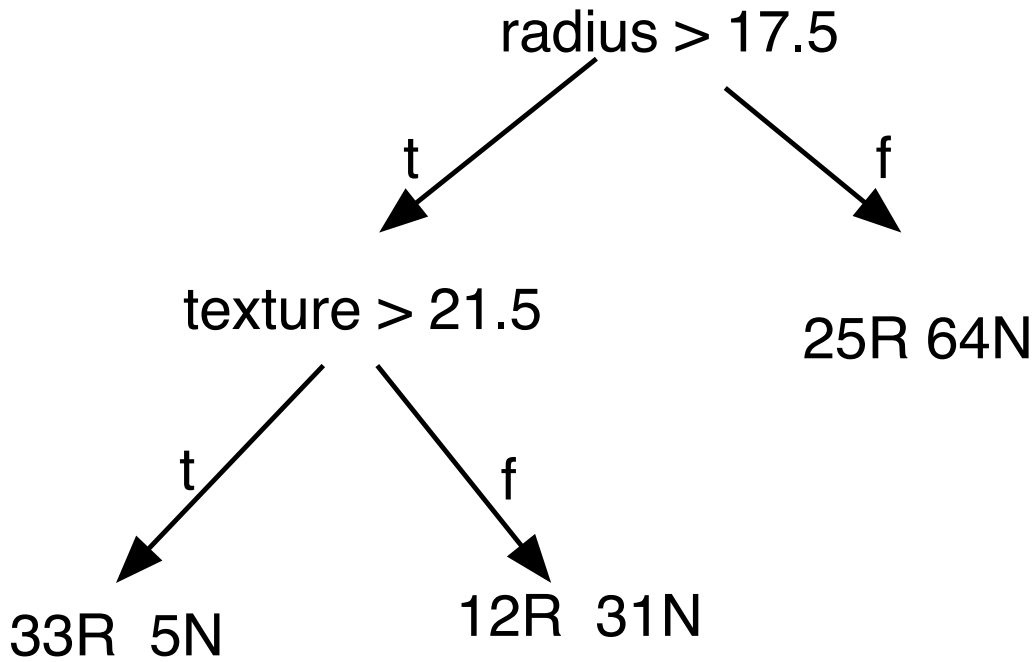
A decision tree can be converted an equivalent set of if-then rules.



IF	THEN most likely class is
radius>17.5 AND texture>21.5	R
radius>17.5 AND texture≤21.5	N
radius≤17.5	N

Decision trees as logical representations

A decision tree can be converted an equivalent set of if-then rules.



IF	THEN P(R) is
radius>17.5 AND texture>21.5	$33/(33+5)$
radius>17.5 AND texture$\leq$21.5	$12/(12+31)$
radius$\leq$17.5	$25/(25+64)$

Decision trees, more formally

- Each internal node contains a *test*, on the value of one (typically) or more feature values
- A test produces discrete outcomes, e.g.,
  - radius > 17.5
  - radius $\in [12, 18]$
  - grade is $\{A, B, C, D, F\}$
  - grade is $\geq B$
  - color is RED
  - $2*\text{radius} - 3*\text{texture} > 16$
- For discrete features, typically branch on some, or all, possible values
- For real features, typically branch based on a threshold value

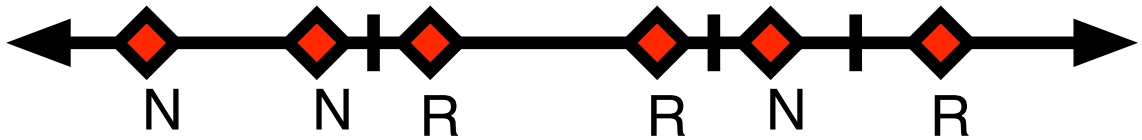
$\Rightarrow$ *A finite set of possible tests* is usually decided before learning the tree; learning comprises choosing the shape of the tree and the tests at every node.

More on tests for real-valued features

- Suppose feature j is real-valued,
- How do we choose a finite set of possible thresholds, for tests of the form $x_j > \tau$?
  - Regression: choose midpoints of the observed data values,

    $x_{1,j}, x_{2,j}, \ldots, x_{m,j}$

  - Classification: choose midpoints of data values with different y values



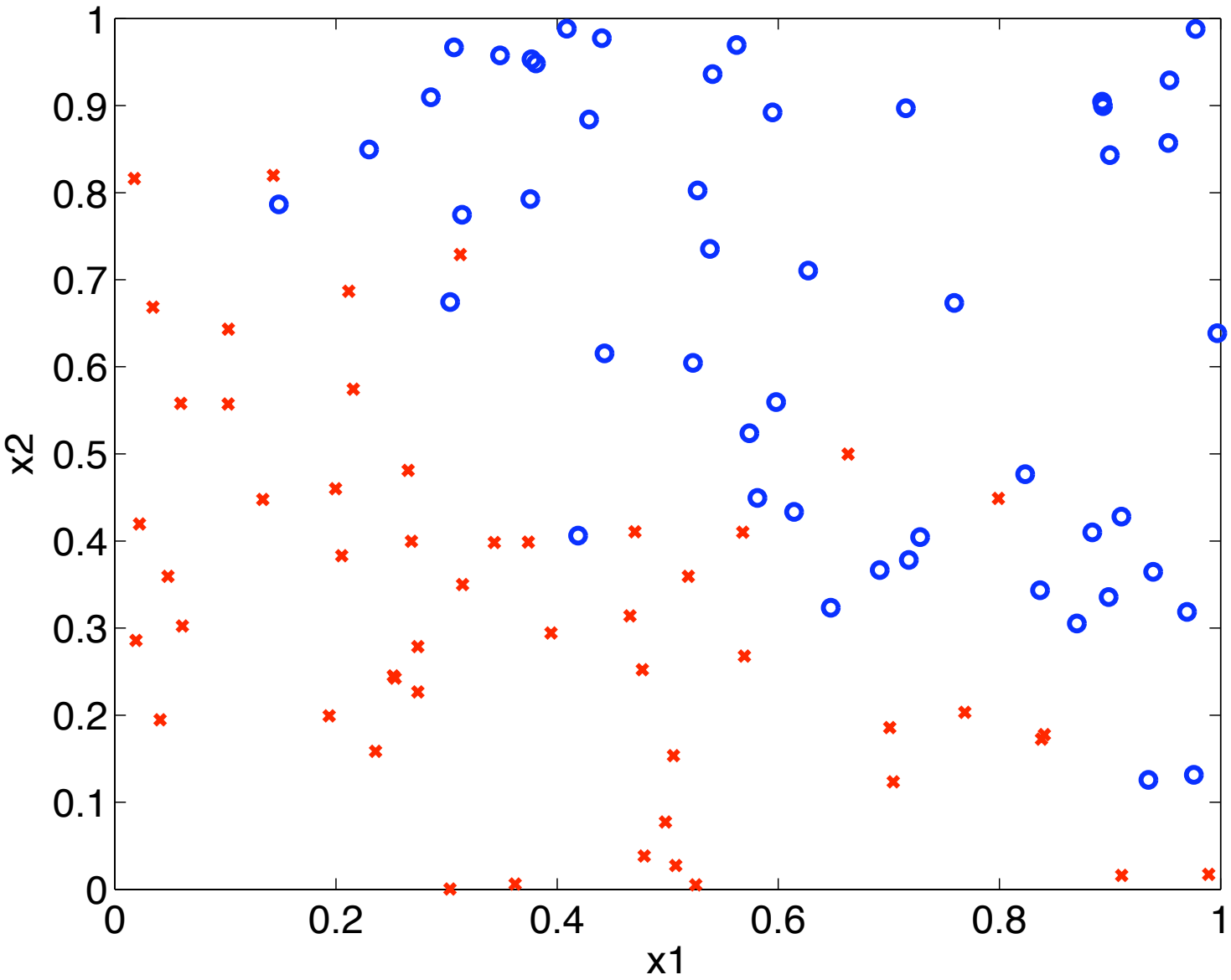
Representational power and efficiency of decision trees

- Suppose the input $\mathbf{x}$ consists of n binary features
- How can a decision tree represent:
  - $y = x_1$ AND x_2 AND $\ldots$ AND x_n
  - $y = x_1$ OR x_2 OR $\ldots$ OR x_n
  - $y = x_1$ XOR x_2 XOR $\ldots$ XOR x_n

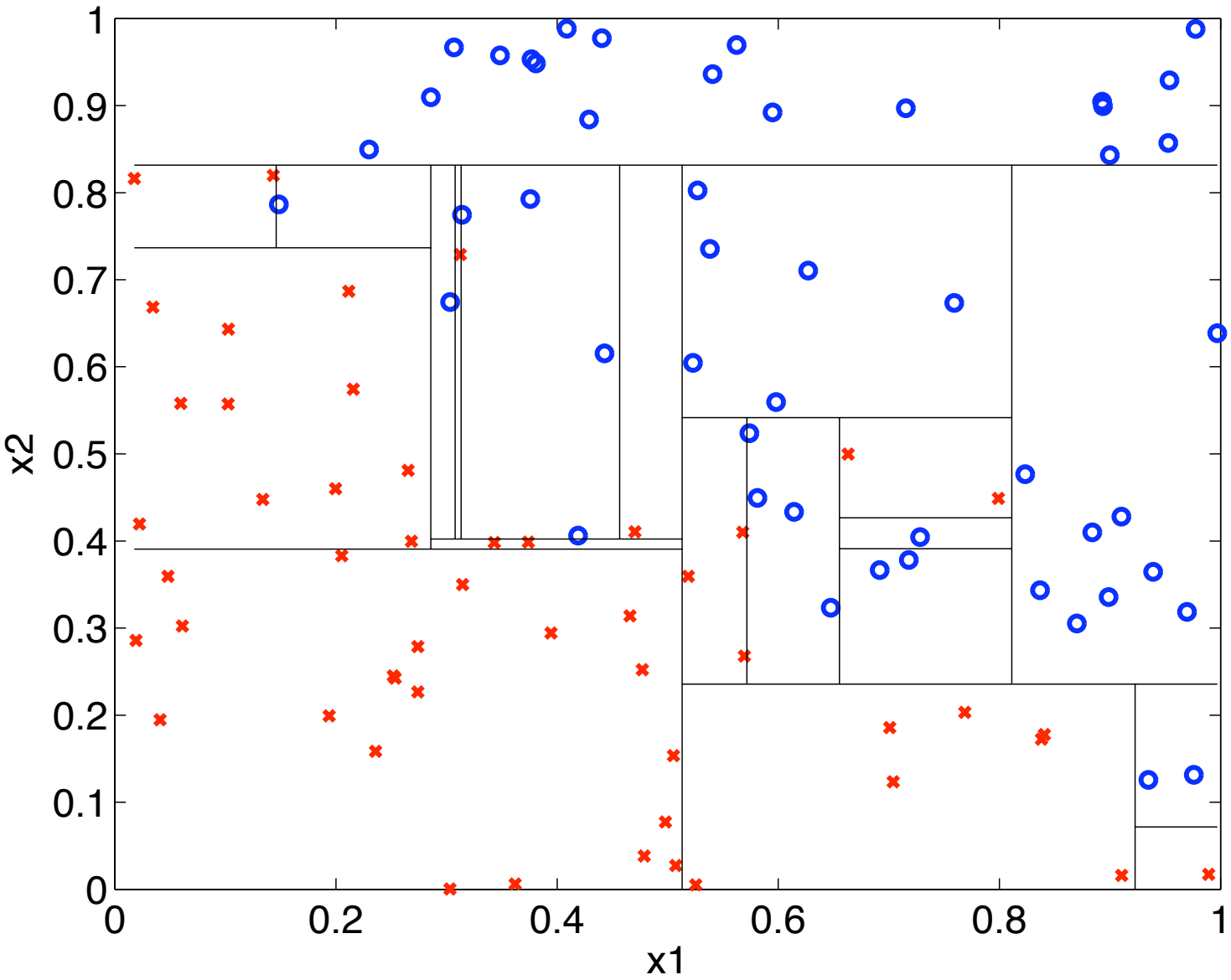
Representational power and efficiency of decision trees

- With typical univariate tests, AND and OR are easy, taking $O(n)$ tests
- Parity/XOR type problems are hard, taking $O(2^n)$ tests
- With real-valued features, decision trees are good at problems in which the class label is **constant in large, connected, axis-orthogonal regions of the input space.**

An artificial example



Example: Decision tree decision surface



How do we learn decision trees?

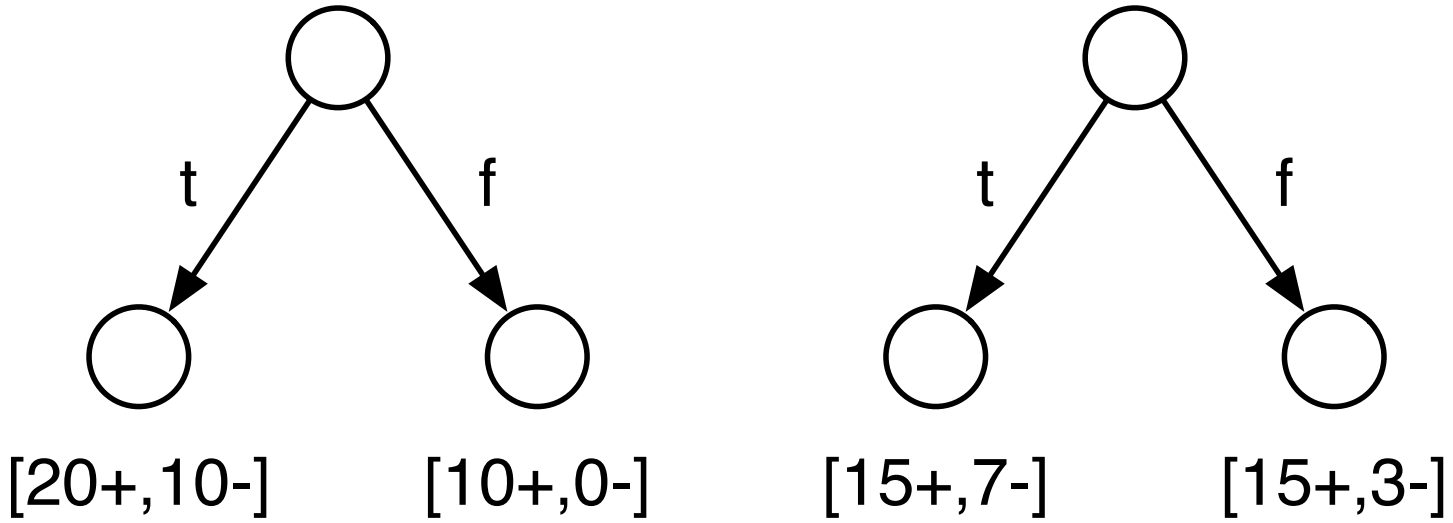
- We could enumerate all possible trees (assuming number of possible tests is finite),
  - Each tree could be evaluated using the training set or, better yet, a validation set
  - But there are many possible trees! Combinatorial problem...
  - We'd probably overfit the data anyway
- Usually, decision trees are constructed in two phases:
  1. An recursive, top-down procedure "grows" a tree (possibly until the training data is completely fit)
  2. The tree is "pruned" back to avoid overfitting
- Both typically use *greedy heuristics*

Top-down induction of decision trees

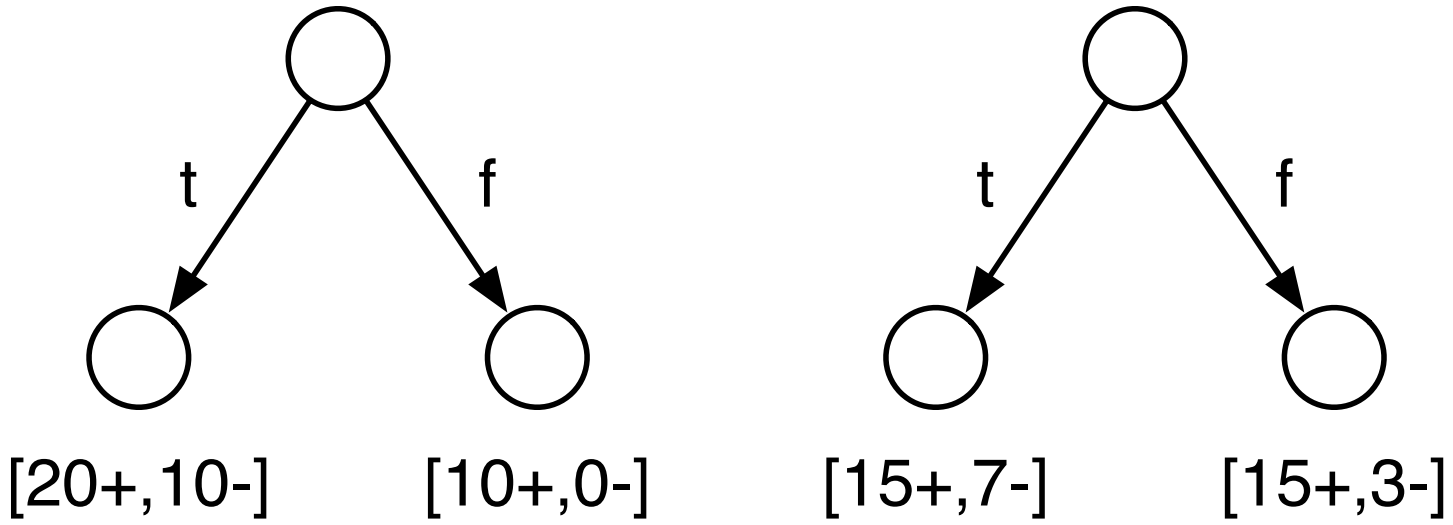
- For a classification problem:
  1. If all the training instances have the same class, create a leaf with that class label and exit.
  2. Pick the *best test* to split the data on
  3. Split the training set according to the value of the outcome of the test
  4. Recurse on each subset of the training data
- For a regression problem - same as above, except:
  - The decision on when to stop splitting has to be made earlier
  - At a leaf, either predict the mean value, or do a linear fit

Which test is best?

- The test should provide *information* about the class label.
- Suppose we have 30 positive examples, 10 negative ones, and we are considering two tests that would give the following splits of instances:



Which test is best?



- Intuitively, we would like an attribute that *separates* the training instances as well as possible
- If each leaf was pure, the attribute would provide maximal information about the label at the leaf
- We need a mathematical measure for the purity of a set of instances

What is information?

- Imagine:
  1. You are about to observe the outcome of a dice roll
  2. You are about to observe the outcome of a coin flip
  3. You are about to observe the outcome of a biased coin flip
  4. Someone is about to tell you your own name
- Intuitively, in each situation you have a different amount of uncertainty as to what outcome / message you will observe.

Information=Reduction in uncertainty

Let E be an event that occurs with probability $P(E)$. If we are told that E has occurred with certainty, then we received

$$I(E) = \log_2 \frac{1}{P(E)}$$

bits of information.

- You can also think of information as the amount of "surprise" in the outcome (e.g., consider $P(E) = 1$, $P(E) \approx 0$)
- E.g., result of a fair coin flip provides $\log_2 2 = 1$ bit of information
- E.g., result of a fair dice roll provides $\log_2 6 \approx 2.58$ bits of information.
- E.g., result of being told your own name (or any other deterministic event) produces 0 bits of information

Information is additive

Suppose you have k independent fair coin tosses. How much information do they give?

$$I(k \text{ fair coin tosses }) = \log_2 \frac{1}{1/2^k} = k \text{ bits}$$

A cute example:

- Consider a random word drawn from a vocabulary of 100,000 words: $I(\text{word}) = \log_2 100,000 \approx 16.61$ bits
- Now consider a 1000 word document drawn from the same source: $I(\text{document}) \approx 16610$ bits
- Now consider a 480×640 gray-scale image with 16 grey levels: $I(\text{picture}) = 307,200 \cdot \log_2 16 = 1\,228\,800$ bits!

$\Longrightarrow$ A picture is worth (more than) a thousand words!

Entropy

- Suppose we have an information source S which emits symbols from an alphabet $\{s_1, \ldots s_k\}$ with probabilities $\{p_1, \ldots p_k\}$. Each emission is independent of the others.
- What is the *average amount of information* when observing the output of S?

$$H(S) = \sum_i p_i I(s_i) = \sum_i p_i \log \frac{1}{p_i} = -\sum_i p_i \log p_i$$

  This is called the *entropy* of S.
- Note that this depends *only on the probability distribution* and not on the actual alphabet (so we can really write $H(P)$).

Interpretations of entropy

$$H(P) = \sum_i p_i \log \frac{1}{p_i}$$

- Average amount of information per symbol
- Average amount of surprise when observing the symbol
- Uncertainty the observer has before seeing the symbol
- Average number of bits needed to communicate the symbol

Entropy and coding theory

- Suppose I will get data from a 4-value alphabet $\mathbf{z}_j$ and I want to send it over a channel. I know that the probability of item $\mathbf{z}_j$ is p_j.
- Suppose all values are equally likely. Then I can encode them in two bits each, so on every transmission I need 2 bits
- Suppose now $p_0 = 0.5$, $p_1 = 0.25$, $p_2 = p_3 = 0.125$. What is the best encoding?

Entropy and coding theory

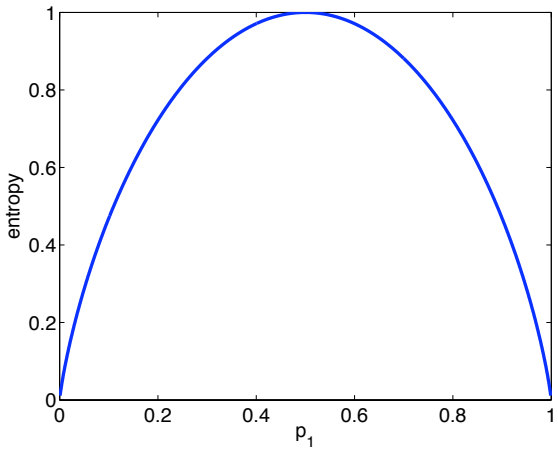
- Suppose I will get data from a 4-value alphabet $\mathbf{z}_j$ and I want to send it over a channel. I know that the probability of item $\mathbf{z}_j$ is p_j.
- Suppose all values are equally likely. Then I can encode them in two bits each, so on every transmission I need 2 bits
- Suppose now $p_0 = 0.5$, $p_1 = 0.25$, $p_2 = p_3 = 0.125$. What is the best encoding?
  $\mathbf{z}_0 = 0$, $\mathbf{z}_1 = 10$, $\mathbf{z}_2 = 110$, $\mathbf{z}_3 = 111$.
  What is the expected length of the message over time?

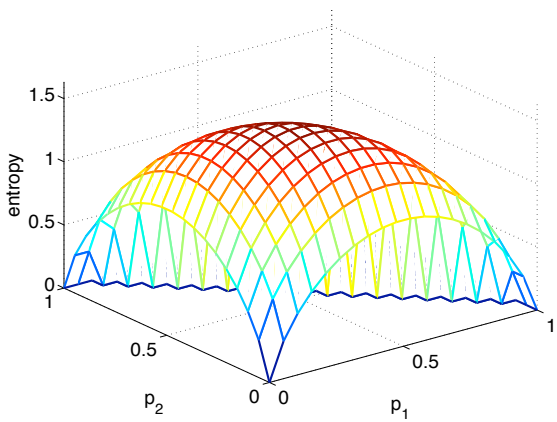
Shannon: there are codes that will communicate the symbols with efficiency arbitrarily close to $H(P)$ bits/symbol. There are no codes that will do it with efficiency greater than $H(P)$ bits/symbol.

Properties of entropy

$$H(P) = \sum_{i=1}^{k} p_i \log \frac{1}{p_i}$$

- Non-negative: $H(P) \geq 0$, with equality if and only if any $p_i = 1$.
- $H(P) \leq \log k$ with equality if and only if $p_i = \frac{1}{k}, \forall i$
- The further P is from uniform, the lower the entropy





Entropy applied to binary classfication

- Consider data set D and let
  - $p_{\oplus}$ = the proportion of positive examples in D
  - $p_{\ominus}$ = the proportion of negative examples in D
- Entropy measures the impurity of D, based on empirical probabilities of the two classes:

$$H(D) \equiv p_{\oplus} \log_2 \frac{1}{p_{\oplus}} + p_{\ominus} \log_2 \frac{1}{p_{\ominus}}$$

So we can use it to measure purity!

Example

Suppose I am trying to predict output y and I have input x, e.g.:

x =HasKids	y =OwnsDoraVideo
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
No	No
No	No
Yes	No
Yes	No

- From the table, we can estimate $P(y = YES) = 0.5 = P(y = NO)$.
- Thus, we estimate $H(y) = 0.5 \log \frac{1}{0.5} + 0.5 \log \frac{1}{0.5} = 1$.

Example: Conditional entropy

x =HasKids	y =OwnsDoraVideo
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
No	No
No	No
Yes	No
Yes	No

Specific conditional entropy is the uncertainty in y given a particular x value. E.g.,

- $P(y = YES|x = YES) = \frac{2}{3}$, $P(y = NO|x = YES) = \frac{1}{3}$
- $H(y|x = YES) = \frac{2}{3} \log \frac{1}{(\frac{2}{3})} + \frac{1}{3} \log \frac{1}{(\frac{1}{3})} \approx 0.9183.$

Conditional entropy

- *The conditional entropy, $H(y|x)$*, is the average specific conditional entropy of y given the values for x:

$$H(y|x) = \sum_v P(x = v) H(y|x = v)$$

- E.g. (from previous slide),
  - $H(y|x = YES) = \frac{2}{3} \log \frac{1}{(\frac{2}{3})} + \frac{1}{3} \log \frac{1}{(\frac{1}{3})} \approx 0.6365$
  - $H(y|x = NO) = 0 \log \frac{1}{0} + 1 \log \frac{1}{1} = 0.$
  - $H(y|x) = H(y|x = YES)P(x = YES) + H(y|x = NO)P(x = NO) = 0.6365 * \frac{3}{4} + 0 * \frac{1}{4} \approx 0.4774$
- Interpretation: the expected number of bits needed to transmit y if both the emitter and the receiver know the possible values of x (but before they are told x's specific value).

Information gain

- Suppose one has to transmit y. How many bits on the average would it save if both the transmitter and the receiver knew x?

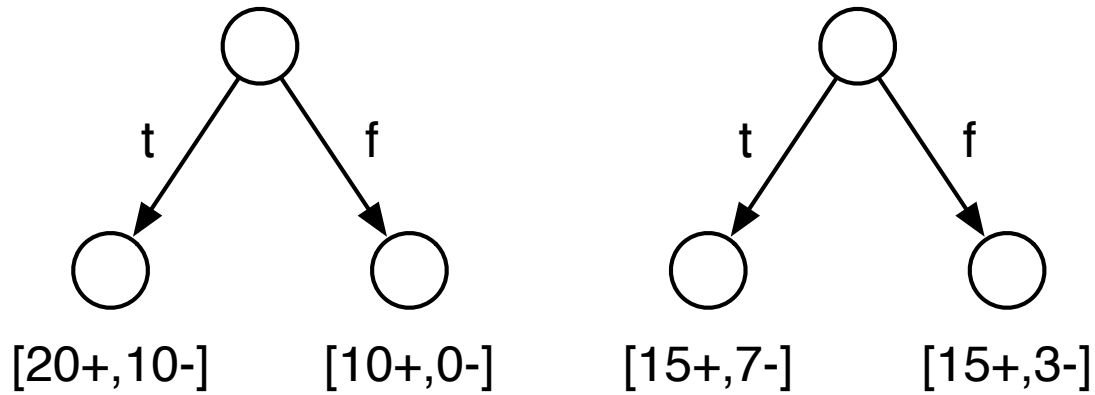
$$IG(y|x) = H(y) - H(y|x)$$

  This is called *information gain*
- Alternative interpretation: what reduction in entropy would be obtained by knowing x
- Intuitively, this has the meaning we seek for decision tree construction

Information gain to determine best test

- We choose, recursively at each interior node, the test that has highest information gain.
  Equivalently, test that results in lowest conditional entropy.
- If tests are binary:

$$\begin{aligned} IG(D, \text{Test}) &= H(D) - H(D|\text{Test}) \\ &= H(D) - \frac{|D_{\text{Test}}|}{|D|} H(D_{\text{Test}}) - \frac{|D_{\neg\text{Test}}|}{|D|} H(D_{\neg\text{Test}}) \end{aligned}$$



Check that in this case, Test1 wins.

Caveats on tests with multiple values

- If the outcome of a test is *multi-valued*, the number of possible values influences the information gain
- The more possible values, the higher the gain! (the more likely it is to form small, but pure partitions)
- C4.5 (the most popular decision tree construction algorithm in ML community) uses only binary tests:
  - Attribute=Value for discrete attributes
  - Attribute $<$ or $>$ Value for continuous attributes
- Other approaches consider smarter metrics (e.g. gain ratio), which account for the number of possible outcomes

Alternative purity measures

- For classification, an alternative to the information gain is the *Gini index*:

$$\sum_y P(y)(1 - P(y)) = 1 - \sum_y (P(y))^2$$

  Same qualitative behavior as the entropy, but not the same interpretation

- For regression trees, purity is measured by the average mean-squared error at each leaf
  E.g. CART (Breiman et al., 1984) [HTF 9.2]

Dealing with noise in the training data

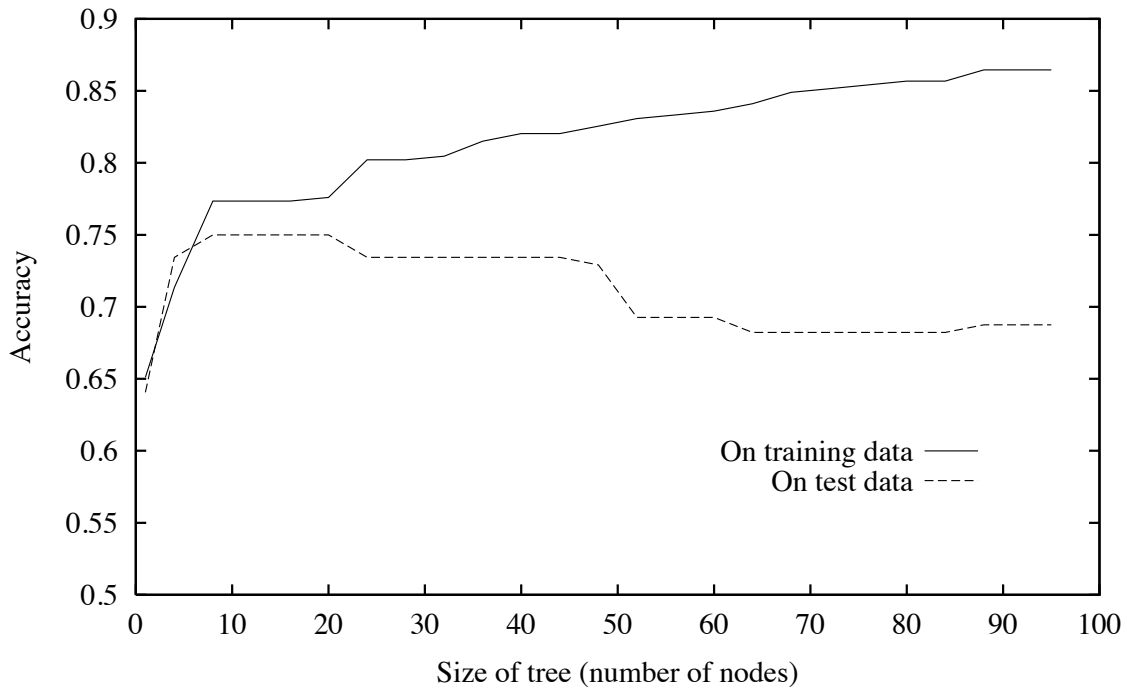
Noise is inevitable!

- Values of attributes can be misrecorded
- Values of attributes may be missing
- The class label can be misrecorded

What happens when adding a noisy example?

Overfitting in decision trees

- Remember, decision tree construction proceeds until all leaves are pure – all examples having the same y value.
- As the tree grows, the generalization performance starts to degrade, because the algorithm is finding *irrelevant attributes / tests*.



Example from (Mitchell, 1997)

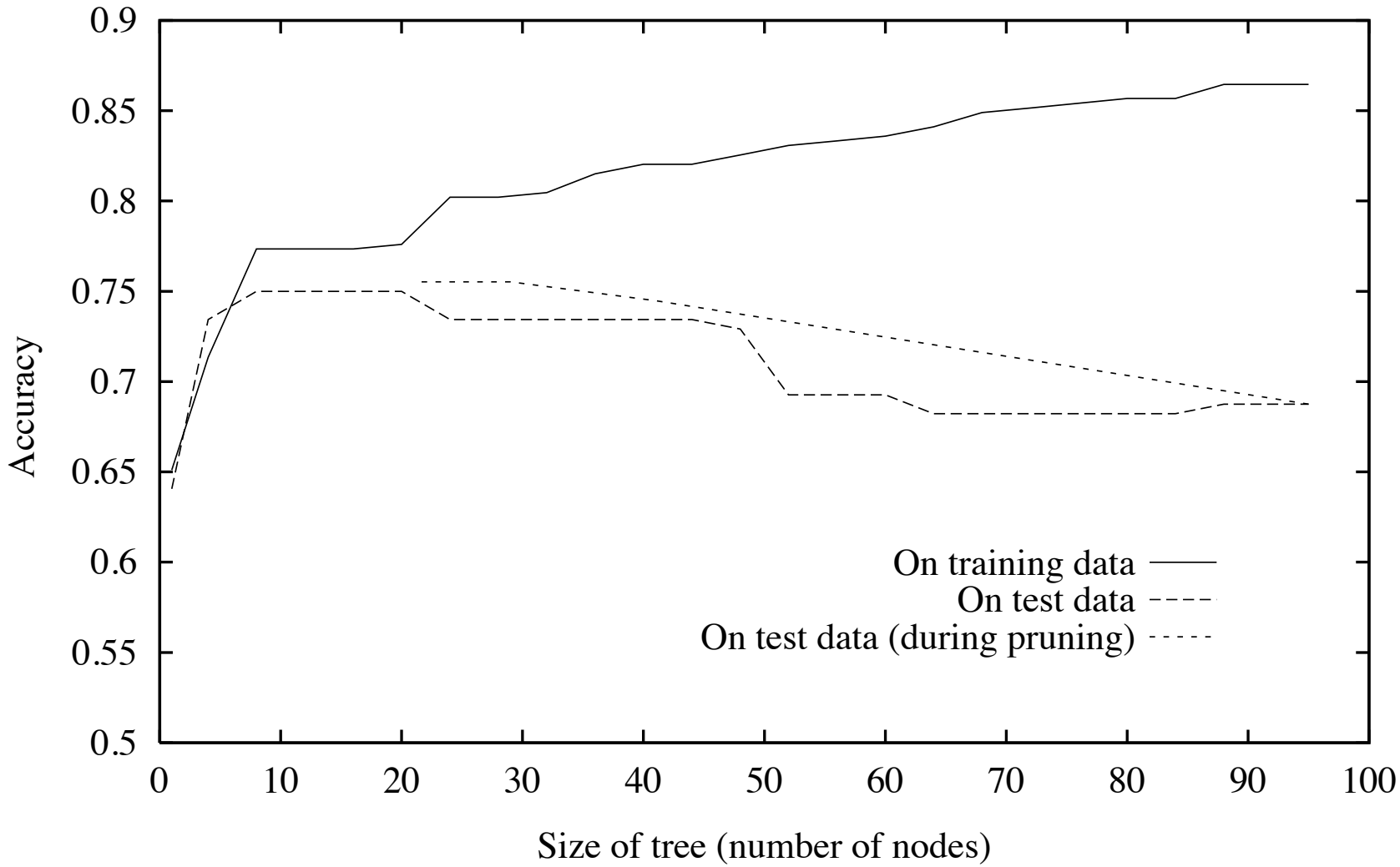
Avoiding overfitting

- Two approaches:
  1. Stop growing the tree when further splitting the data does not yield a statistically significant improvement
  2. Grow a full tree, then *prune* the tree, by eliminating nodes
- The second approach has been more successful in practice, because in the first case it might be hard to decide if the information gain is sufficient or not (e.g. for multivariate functions)
- We will select the best tree, for now, by measuring performance on a separate validation data set.

Example: Reduced-error pruning

1. Split the "training data" into a training set and a validation set
2. Grow a large tree (e.g. until each leaf is pure)
3. For each node:
   1. Evaluate the validation set accuracy of pruning the subtree rooted at the node
   2. Greedily remove the node that most improves validation set accuracy, with its corresponding subtree
   3. Replace the removed node by a leaf with the majority class of the corresponding examples.
4. Stop when pruning starts hurting the accuracy on the validation set.

Example: Effect of reduced-error pruning



Example: Rule post-pruning in C4.5

1. Convert the decision tree to rules
2. Prune each rule independently of the others, by removing preconditions such that the accuracy is improved
3. Sort final rules in order of estimated accuracy

Advantages:

- Can prune attributes higher up in the tree *differently on different paths*
- There is no need to reorganize the tree if pruning an attribute that is higher up
- Most of the time people want rules anyway, for readability

Missing values during classification

- Assign "most likely" value based on all the data that reaches the current node. This is a form of imputation.
- Assign all possible values with some probability.
  - Count the occurrences of the different attribute values in the instances that have reached the same node.
  - Predict all the possible class labels with the appropriate probabilities
  - Introduce a value that means "unknown"
  - Think hard: Is the fact that the value is missing informative?
  - What will happen when you deploy the system?

Variations: Constructing tests on the fly

- Suppose we are interested in more sophisticated tests at nodes, such a linear combinations of features:

$$3.1 \times \text{radius} - 1.9 \times \text{texture} \geq 1.3$$

- We can use a fast, simple classifier (such as logistic) to determine a decision boundary, and us it as a test
- Special-purpose methods also search over logical formulae for tests

Costs of attributes

- Include cost in the metric, e.g.

$$\frac{IG^2(D, \text{Test})}{Cost(\text{Test})}$$

- Mostly a problem in specific domains (e.g. medicine). Multiple metrics have been studied and proposed, without a consensus.

Decision and regression tree summary

- Very fast learning algorithms (e.g. C4.5, CART)
- Attributes may be discrete or continuous, no preprocessing needed
- Provide a general representation of classification rules
- Easy to understand! Though...
  - Exact tree output may be sensitive to small changes in data
  - With many features, tests may not be meaningful
- In standard form, good for (nonlinear) piecewise axis-orthogonal decision boundaries – not good with smooth, curvilinear boundaries
- In regression, the function obtained is discontinuous, which may not be desirable
- Good accuracy in practice – many applications!
  - Equipment/medical diagnosis
  - Learning to fly
  - Scene analysis and image segmentation