COMP 551 - Applied Machine Learning Lecture 9 — Instance learning

William L. Hamilton

(with slides and content from Joelle Pineau)

MiniProject 2

#	Team Name	Kernel	Team Members	Score	Entries	Last
1	**Group 80**			0.93933	3	11h
2	**Group 98**			0.93813	3	14h
	Prof. Hamilton's Benchmark			**0.91866**		
3	**Group 17**			0.90133	3	16h
4	**Group 68**			0.87866	1	11h
	Basic Naive Bayes			**0.83106**		
5	**Group 10**			0.82706	4	16h
6	**Group 3**			0.81573	1	1h
	Random			**0.51173**		

Bernoulli Naïve Bayes classifier

Solving for θ_1 we get:

$\theta_1 = P(y=1) = (1/n) \Sigma_{i=1:n} y_i$

$=$ (# of examples where y=1) / (total # of examples)

Similarly, we get:

$\theta_{j,1} = P(x=1 \mid y=1) =$ (# of examples where x_j=1 and y=1) / (# of examples where y=1)

$\theta_{j,0} = P(x=1 \mid y=0) =$ (# of examples where x_j=1 and y=0) / (# of examples where y=0)

Prediction:

$$\delta(\mathbf{x}) = \log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}$$

$$= \log\left(\frac{\theta_1}{1-\theta_1}\right) + \sum_{j=1}^{m} (\log(\theta_{j,0})(1-x_j) + \log(\theta_{j,1})x_j)$$

Bernoulli Naïve Bayes classifier

Solving for θ_1 we get:

$\theta_1 = P(y=1) = (1/n) \Sigma_{i=1:n} y_i$

$=$ (# of examples where y=1) / (total # of examples)

Similarly, we get:

$\theta_{j,1} = P(x=1 \mid y=1) =$ (# of examples where x_j=1 and y=1) / (# of examples where y=1)

$\theta_{j,0} = P(x=1 \mid y=0) =$ (# of examples where x_j=1 and y=0) / (# of examples where y=0)

Prediction: $\delta(\mathbf{x}) = \log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}$

If $\delta(\boldsymbol{x}) > 0$ *then classify as* 1

If $\delta(\boldsymbol{x}) < 0$ *then classify as* 0

$$= \log\left(\frac{\theta_1}{1-\theta_1}\right) + \sum_{j=1}^{m} (\log(\theta_{j,0})(1-x_j) + \log(\theta_{j,1})x_j)$$

Bernoulli Naïve Bayes classifier

Solving for θ_1 we get:

$\theta_1 = P(y=1) = (1/n) \Sigma_{i=1:n} y_i$

$=$ (# of examples where y=1) / (total # of examples)

Similarly, we get:

$\theta_{j,1} = P(x=1 \mid y=1) =$ (# of examples where x_j=1 and y=1) / (# of examples where y=1)

$\theta_{j,0} = P(x=1 \mid y=0) =$ (# of examples where x_j=1 and y=0) / (# of examples where y=0)

Prediction: $\delta(\mathbf{x}) = \log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}$

If $\delta(\boldsymbol{x}) > 0$ *then classify as* 1

If $\delta(\boldsymbol{x}) < 0$ *then classify as* 0

$$= \log\left(\frac{\theta_1}{1-\theta_1}\right) + \sum_{j=1}^{m}\left(x_j \log\frac{\theta_{j,1}}{\theta_{j,0}} + (1-x_j)\log\frac{1-\theta_{j,1}}{1-\theta_{j,0}}\right)$$

Bernoulli Naïve Bayes classifier

Solving for θ_1 we get:

$\theta_1 = P(y=1) = (1/n) \Sigma_{i=1:n} y_i$

$= (\#$ of examples where $y=1) / (\text{total } \#$ of examples$)$

Similarly, we get:

$\theta_{j,1} = P(x=1 \mid y=1) = (\#$ of examples where $x_j=1$ and $y=1) / (\#$ of examples where $y=1)$

$\theta_{j,0} = P(x=1 \mid y=0) = (\#$ of examples where $x_j=1$ and $y=0) / (\#$ of examples where $y=0)$

Prediction: $\delta(\mathbf{x}) = \log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}$

If $\delta(\boldsymbol{x}) > 0$ *then classify as* 1

If $\delta(\boldsymbol{x}) < 0$ *then classify as* 0

From slides 27-34 in Lecture 5.

$$= \log\left(\frac{\theta_1}{1-\theta_1}\right) + \sum_{j=1}^{m}\left(x_j \log\frac{\theta_{j,1}}{\theta_{j,0}} + (1-x_j)\log\frac{1-\theta_{j,1}}{1-\theta_{j,0}}\right)$$

Bernoulli Naïve Bayes classifier

- Decision boundary vs. probability.
- Prediction: $\delta(\mathbf{x}) = \log \frac{P(y=1|\mathbf{x})}{P(y=0|\mathbf{x})}$ If $\delta(\boldsymbol{x}) > 0$ *then classify as* 1 If $\delta(\boldsymbol{x}) < 0$ *then classify as* 0

$$= \log\left(\frac{\theta_1}{1-\theta_1}\right) + \sum_{j=1}^{m}\left(x_j \log\frac{\theta_{j,1}}{\theta_{j,0}} + (1-x_j)\log\frac{1-\theta_{j,1}}{1-\theta_{j,0}}\right)$$

- $\delta(\boldsymbol{x})$ gives the **decision boundary.** But, since this decision boundary corresponds to the log-odds, we can also compute the class probability:

$$\hat{P}(y|\mathbf{x}) = \frac{1}{1-e^{-\delta(\mathbf{x})}}$$

Parametric supervised learning

- So far, we assumed we have a dataset of labeled examples.
- From this, 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 main goal is to summarize the data using the parameters.
  - Parametric methods are typically global = one set of parameters for the entire data space.

Non-parametric learning methods

- Key idea: just store all training examples $< \mathbf{x}_i, y_i >$.

- When a query is made, computer the value of the new instance based on the values of the closest (most similar) points.

Non-parametric learning methods

- Key idea: just store all training examples $< \mathbf{x}_i, y_i >$.

- When a query is made, computer 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 computer the value of the new point based on the existing values?

What kind of distance metric?

- Euclidean distance?

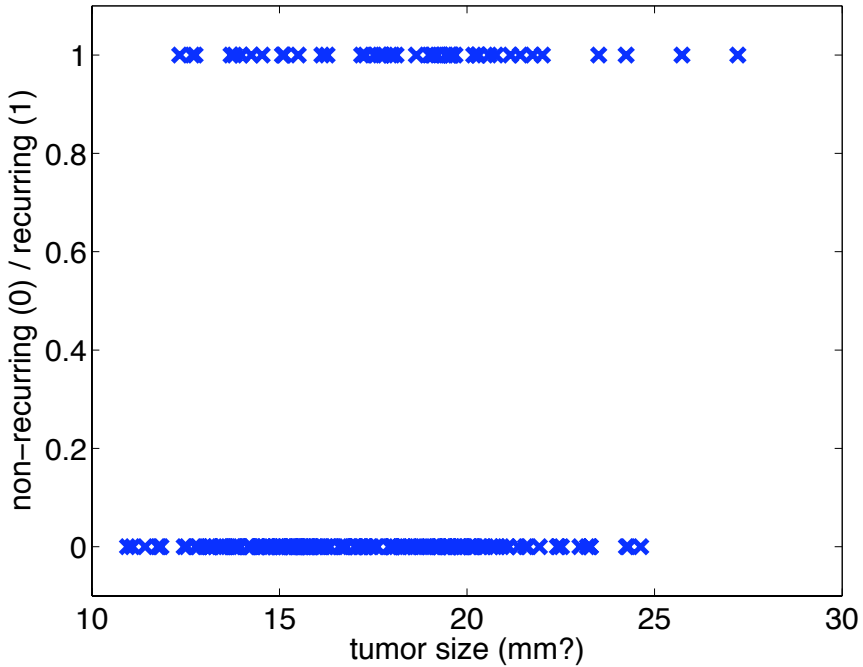
What kind of distance metric?

- Euclidean distance?
- Weighted Euclidean distance (with weights based on domain knowledge): $d(\mathbf{x}, \mathbf{x}') = \sum_{j=1:m} w_j (x_j - x_j')^2$?
- Maximum or minimum difference along any axis?

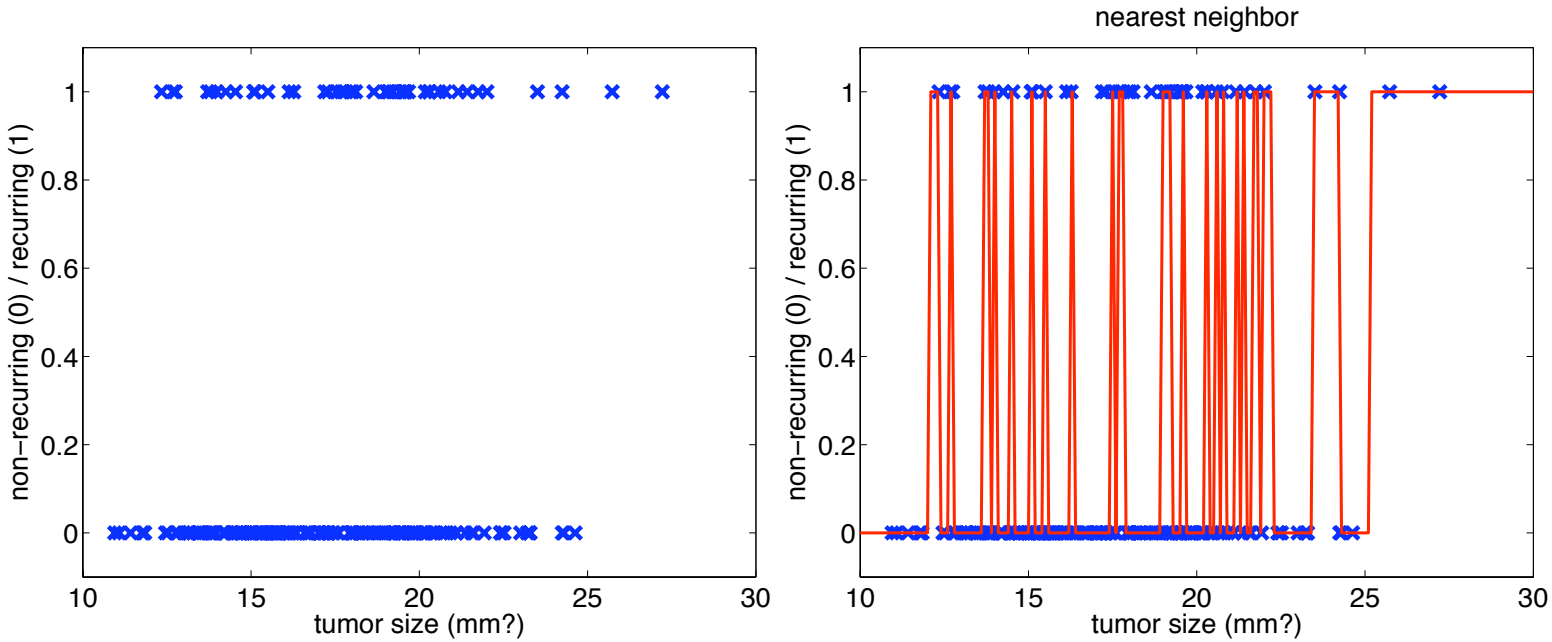
What kind of distance metric?

- Euclidean distance?
- Weighted Euclidean distance (with weights based on domain knowledge): $\mathrm{d}(\mathbf{x}, \mathbf{x}') = \sum_{\mathrm{j}=1:\mathrm{m}} \mathrm{w}_\mathrm{j} \, (\mathrm{x}_\mathrm{j} - \mathrm{x}_\mathrm{j}')^2$?
- Maximum or minimum difference along any axis?
- An arbitrary distance or similarity function d, specific for the application at hand (works best, if you have one.)

Simple idea: Connect the dots!

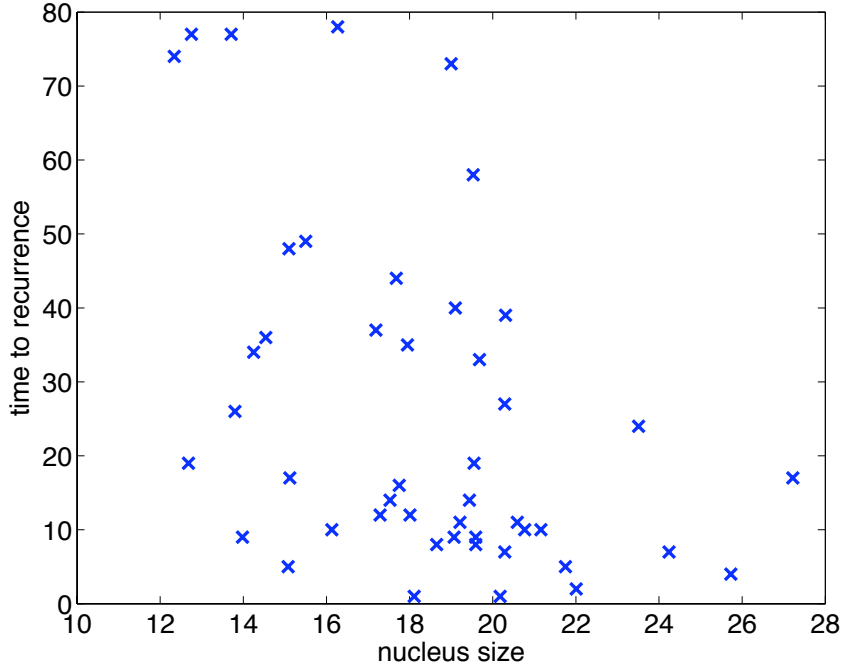


Simple idea: Connect the dots!

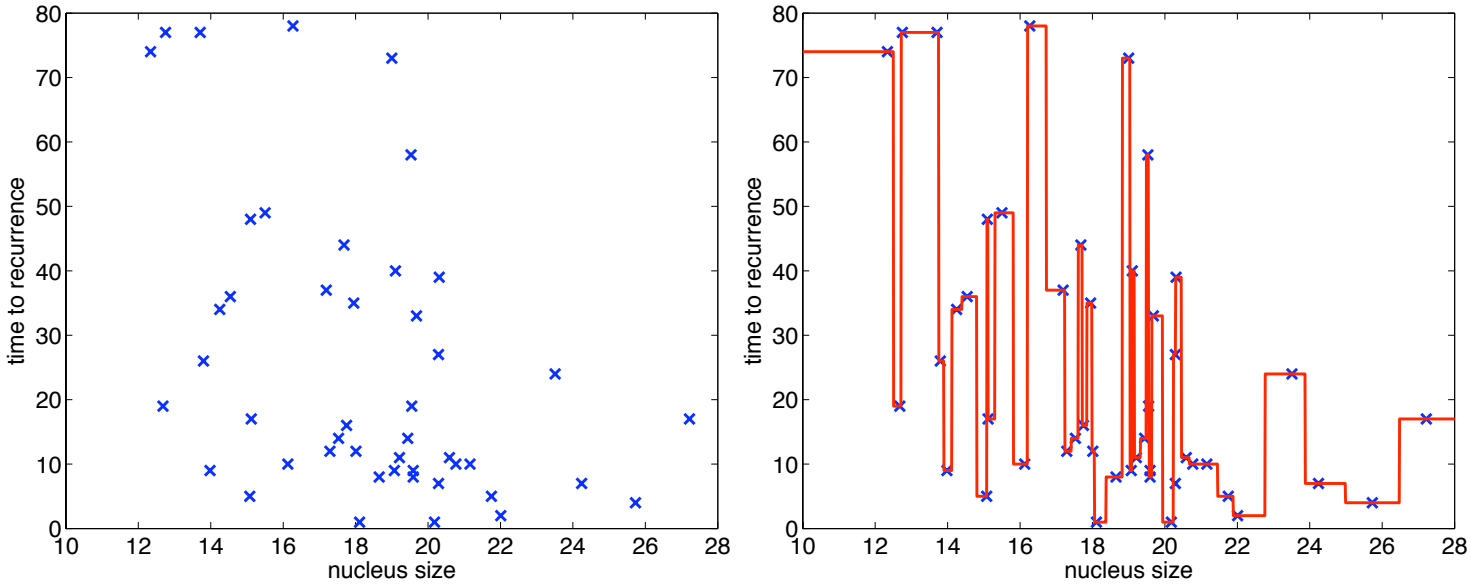


Wisconsin data set, classification

Simple idea: Connect the dots!



Simple idea: Connect the dots!



Wisconsin data set, regression

One-nearest neighbor

- **Given**: Training data $\mathbf{X}$, distance metric $\mathbf{d}$ on $\mathbf{X}$.

- **Learning**: Nothing to do! (Just store the data).

- **Prediction**: For a new point $\mathbf{x_{new}}$

  Find nearest training sample $\mathbf{x_{i*}}$

  $$\mathbf{x_{i*} = \text{argmin}_i \, d(x_i, x)}$$

  Predict $\mathbf{y_{new} = y_{i*}}$

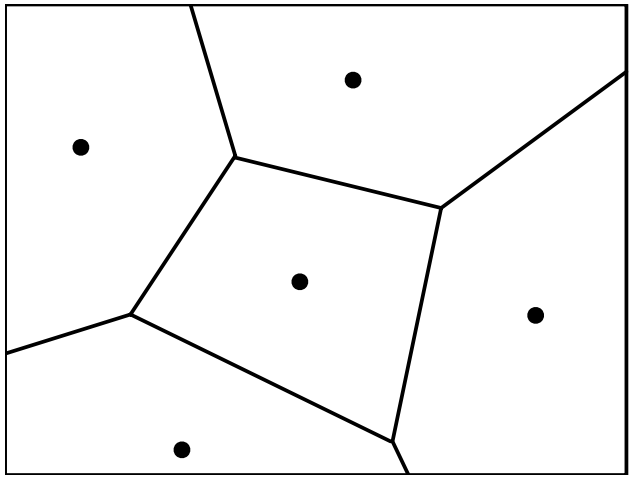
One-nearest neighbor

- **Given**: Training data $\mathbf{X}$, distance metric $\mathbf{d}$ on $\mathbf{X}$.

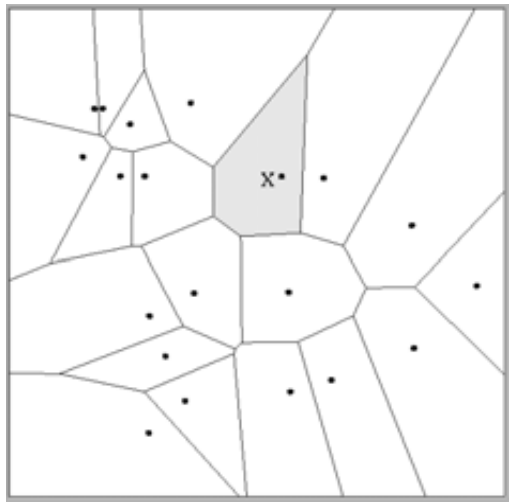
- **Learning**: Nothing to do! (Just store the data).

What does the approximator look like?

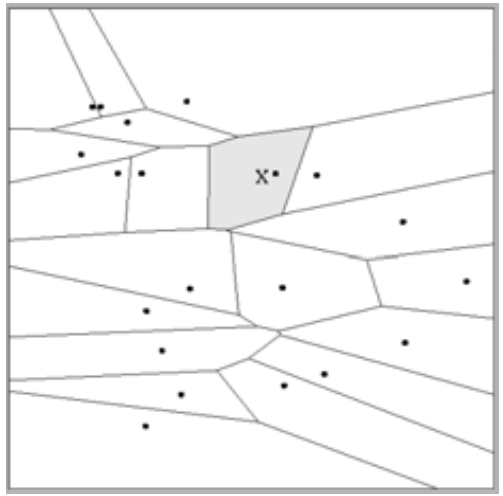
- Nearest-neighbor does not explicitly compute decision boundaries.
- But the effective decision boundaries are a subset of the Voronoi diagram for the training data.
- Each decision boundary is a line segment that is equidistant between two points of opposite classes.



Choice of distance metric is important!



Left: both attributes weighted equally;



Right: second attributes weighted more

Distance metric tricks

- You may need to do feature preprocessing:
  - Scale the input dimensions (or normalize them).
  - Remove noisy inputs.
  - Determine weights for attributes based on cross-validation (or information-theoretic methods).

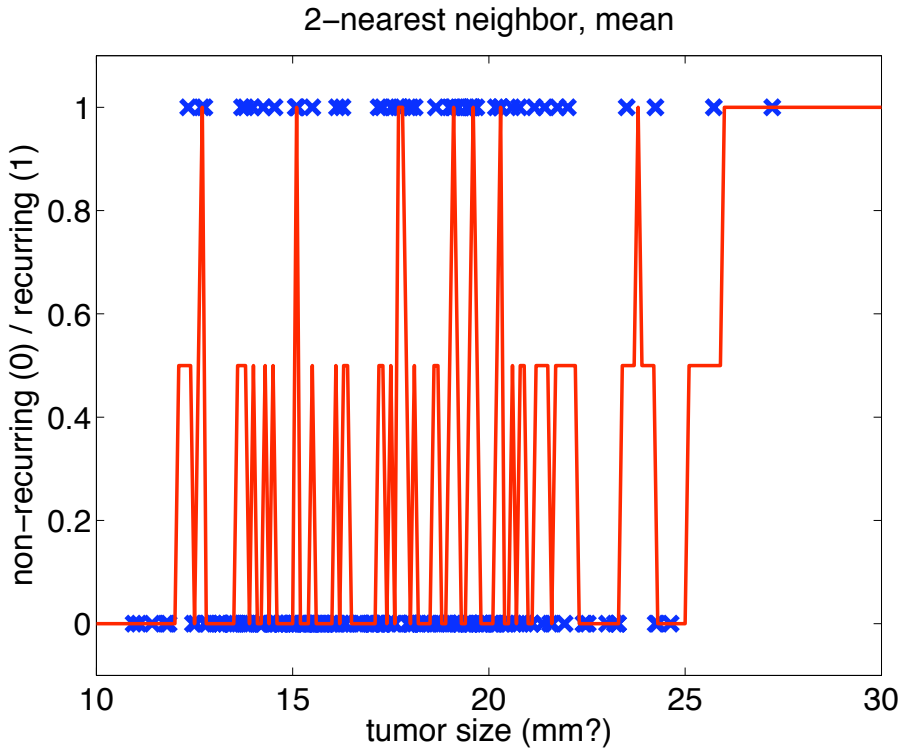
Distance metric tricks

- You may need to do feature 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 can be learned sometimes.

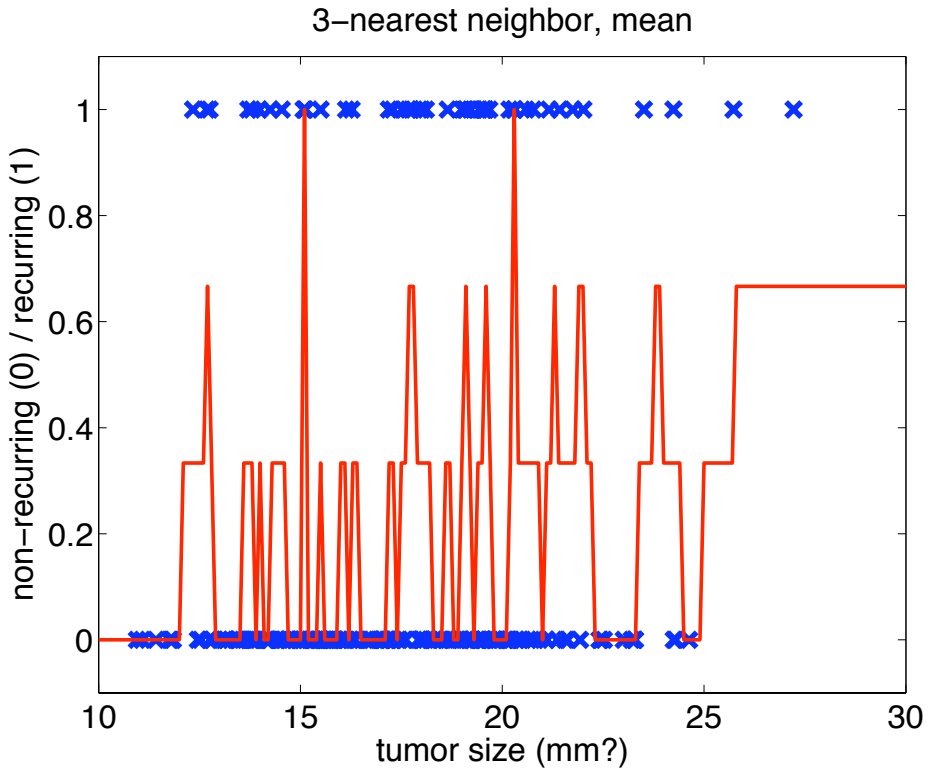
k-nearest neighbor (kNN)

- **Given**: Training data $\mathbf{X}$, distance metric $\mathbf{d}$ on $\mathbf{X}$.
- **Learning**: Nothing to do! (Just store the data).
- **Prediction**:
  - For a new point $\mathbf{x_{new}}$, find the $\mathbf{k}$ nearest training samples to $\mathbf{x}$.
  - Let their indices be $i_1, i_2, \ldots, i_k$.
  - Predict: $y =$ mean/median of $\{y_{i1}, y_{i2}, \ldots, y_{ik}\}$ for regression
    $y =$ majority of $\{y_{i1}, y_{i2}, \ldots, y_{ik}\}$ for classification, or empirical probability of each class.

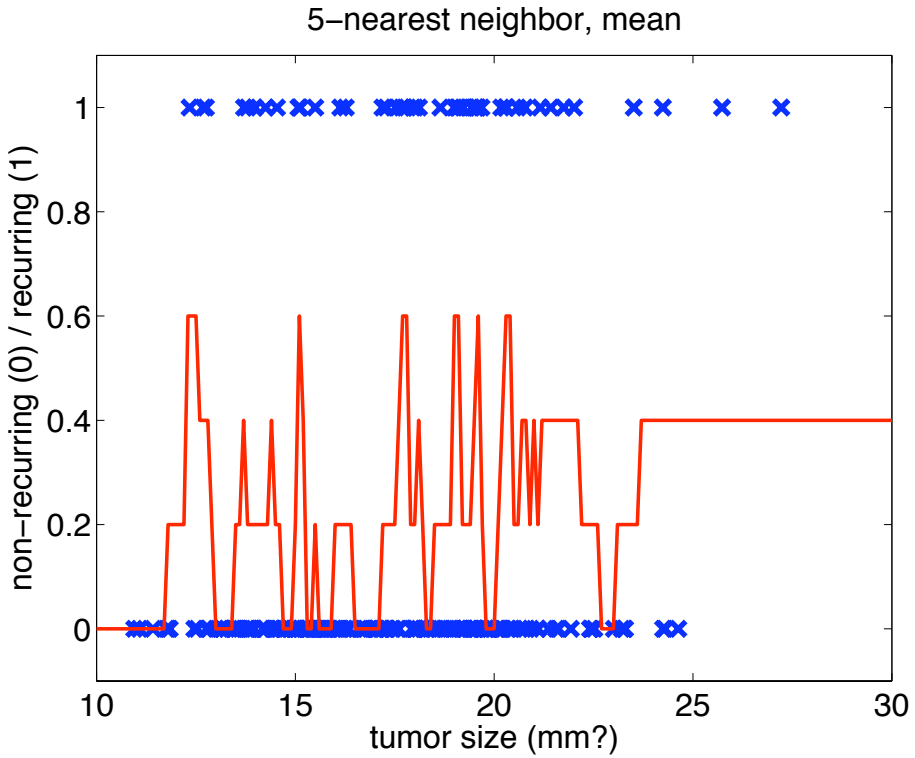
Classification, 2-nearest neighbor



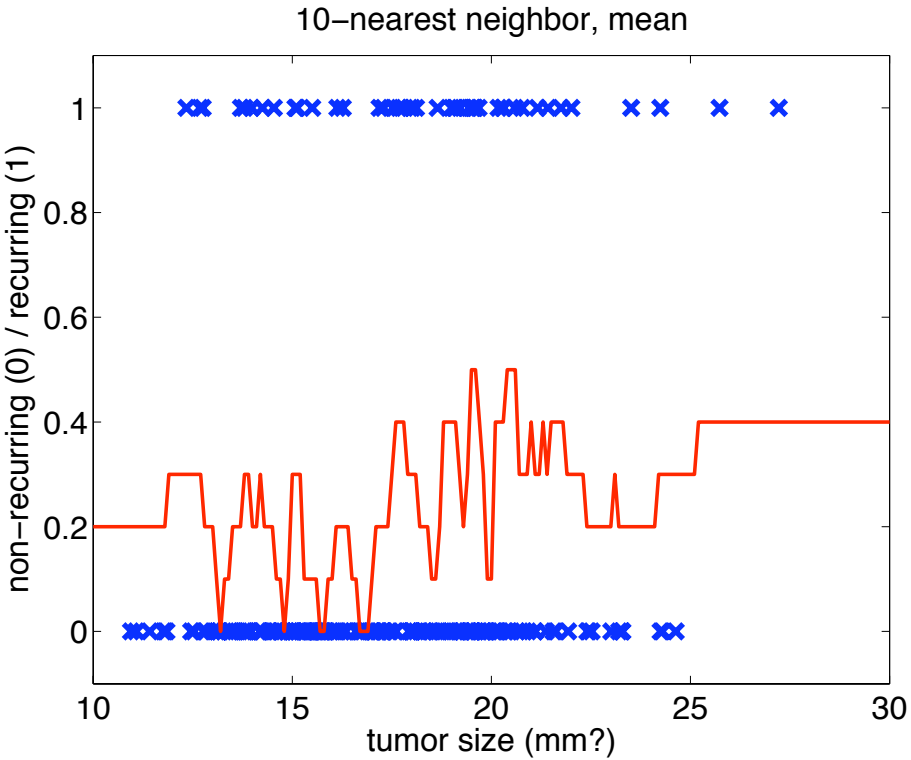
Classification, 3-nearest neighbor



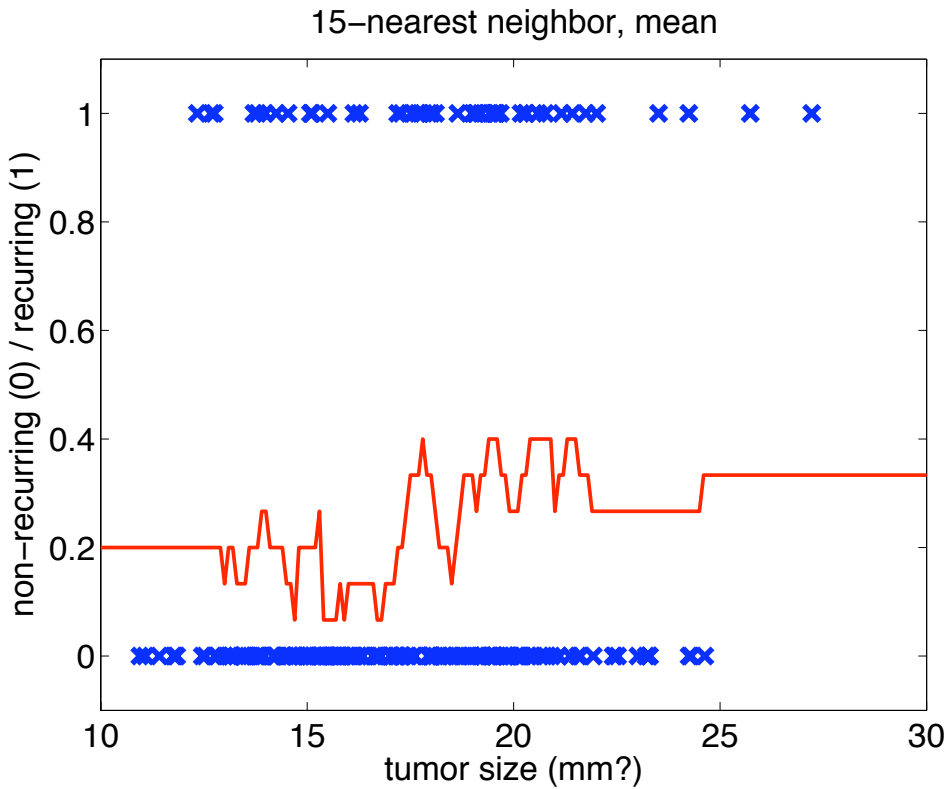
Classification, 5-nearest neighbor



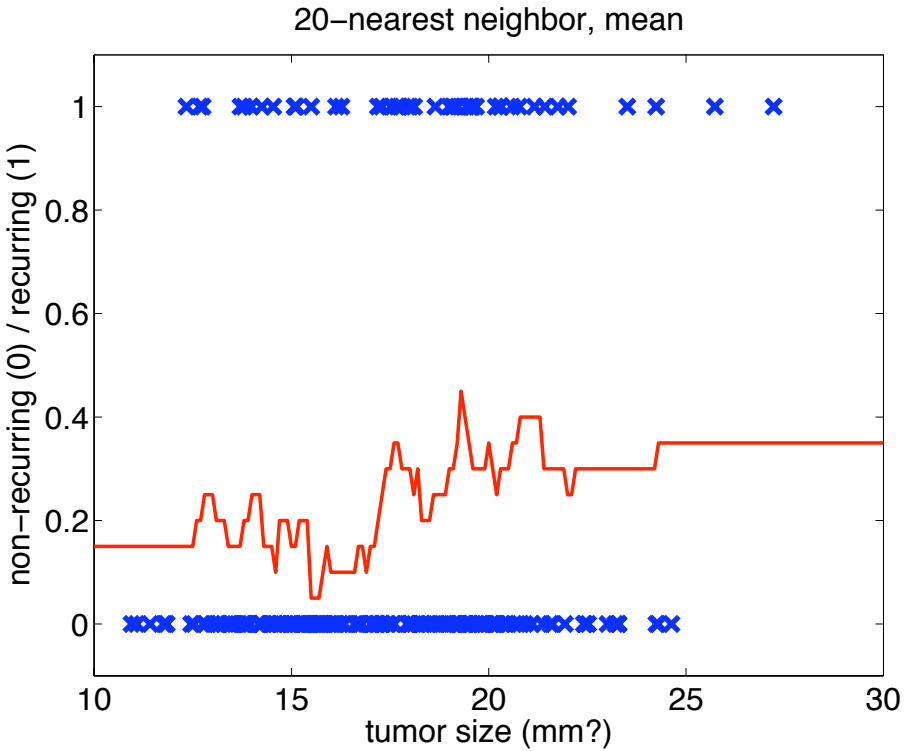
Classification, 10-nearest neighbor



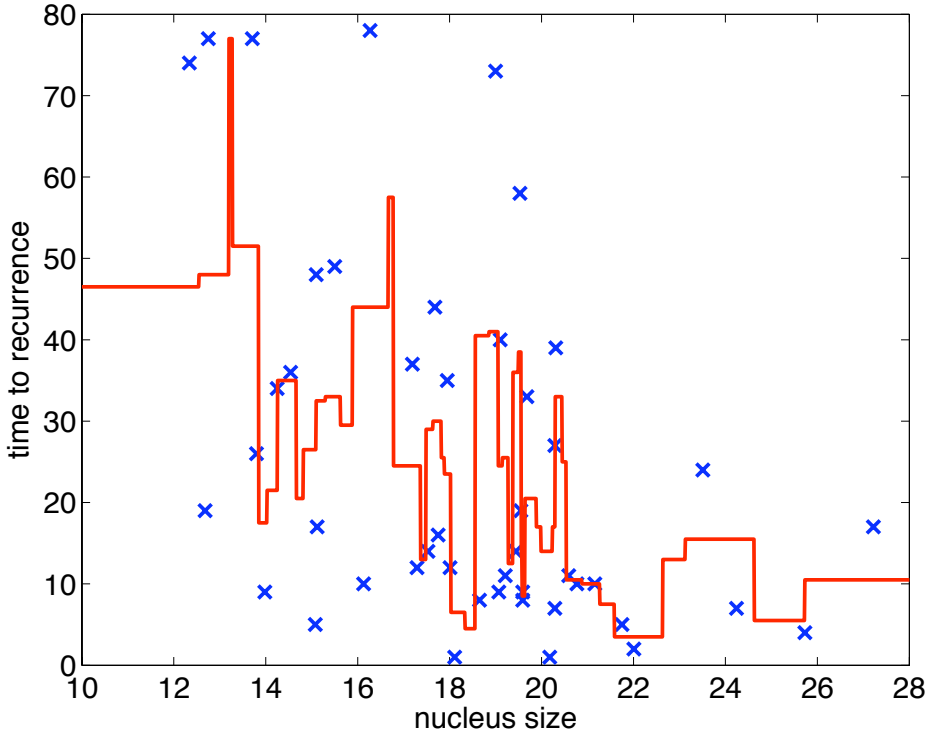
Classification, 15-nearest neighbor



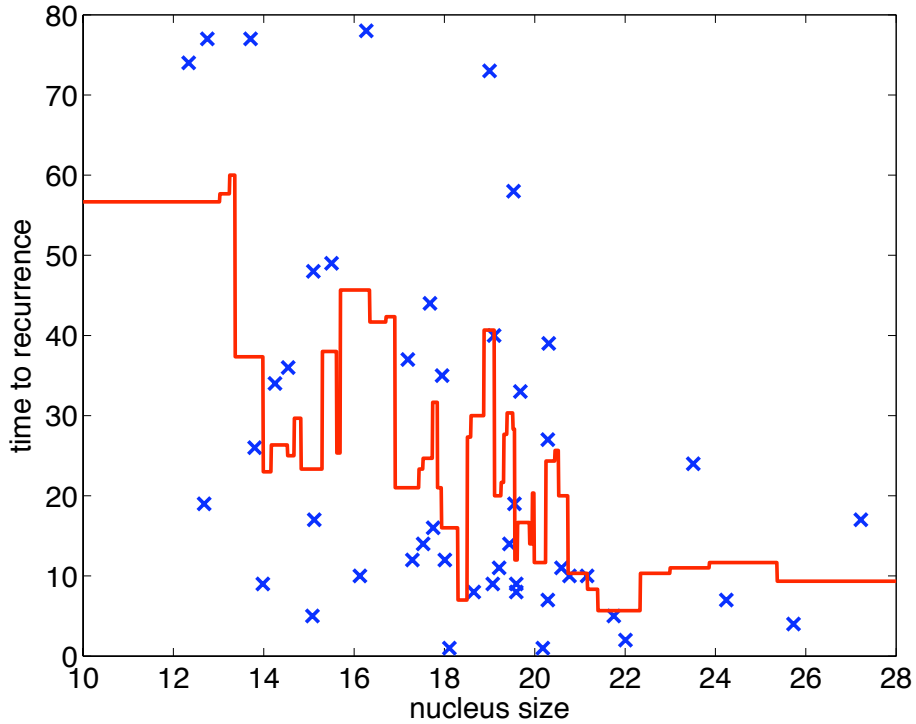
Classification, 20-nearest neighbor



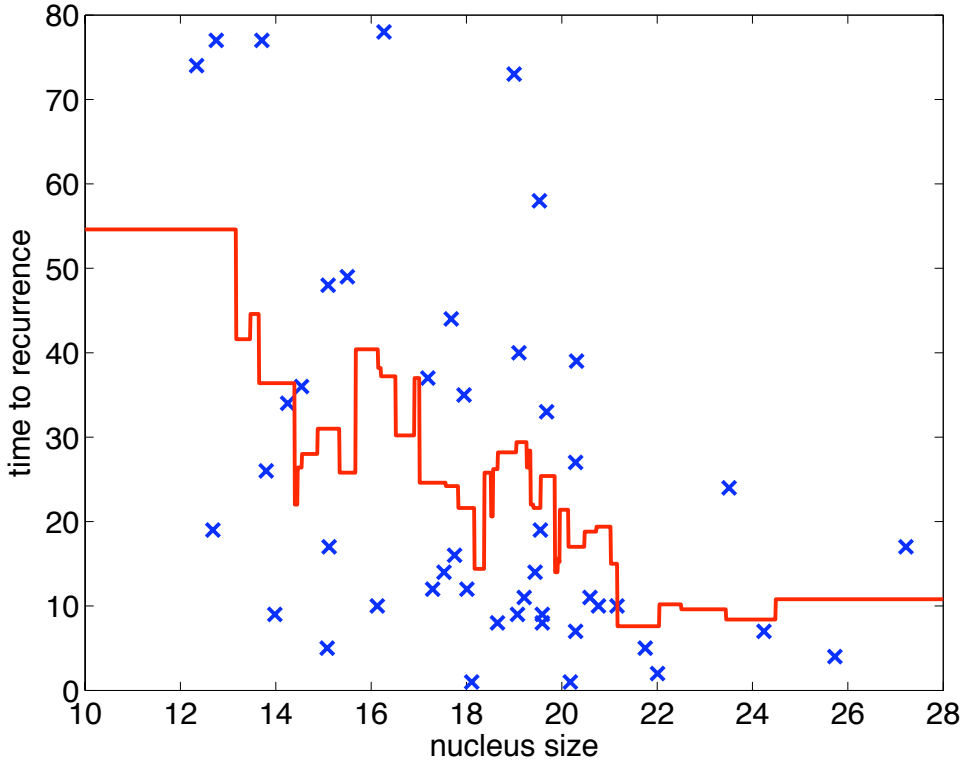
Regression, 2-nearest neighbor



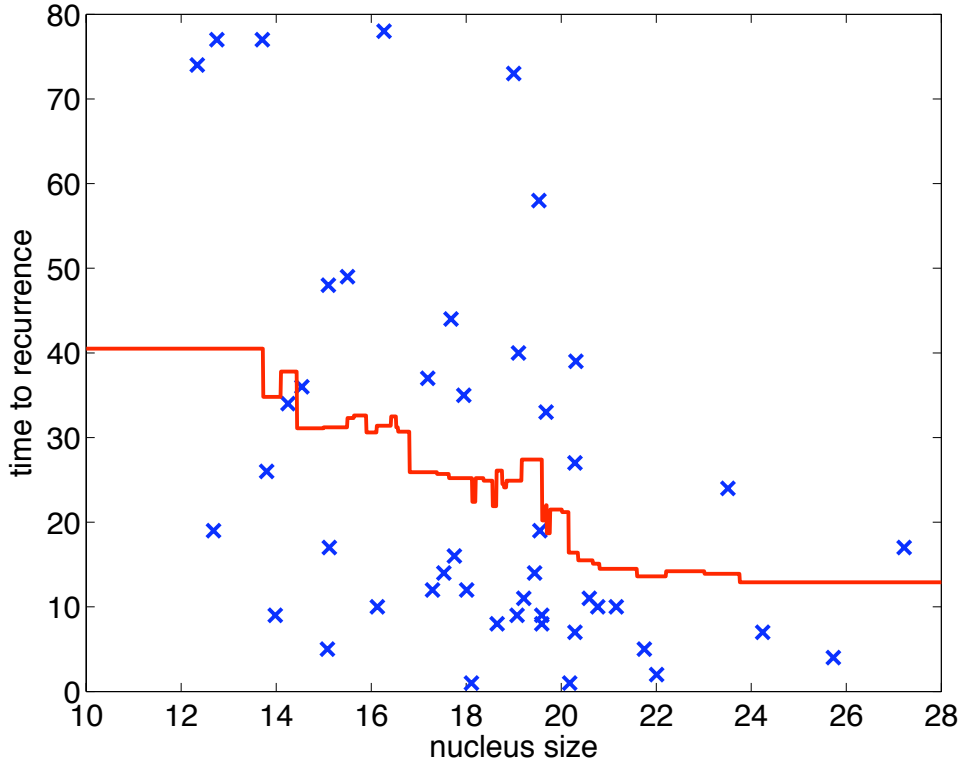
Regression, 3-nearest neighbor



Regression, 5-nearest neighbor



Regression, 10-nearest neighbor



Bias-variance trade-off

- What happens if k is low?

- What happens if k is high?

Bias-variance trade-off

- What happens 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.

- What happens 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.

Limitations of *k*-nearest neighbor (kNN)

- A lot of discontinuities!
- Sensitive to small variations in the input data.
- Can we fix this but still keep it (fairly) local?

k-nearest neighbor (kNN)

- **Given**: Training data $\mathbf{X}$, distance metric $\mathbf{d}$ on $\mathbf{X}$.
- **Learning**: Nothing to do! (Just store the data).
- **Prediction**:
  - For a new point $\mathbf{x_{new}}$, find the $\mathbf{k}$ nearest training samples to $\mathbf{x}$.
  - Let their indices be $i_1, i_2, \ldots, i_k$.
  - Predict: $y =$ mean/median of $\{y_{i1}, y_{i2}, \ldots, y_{ik}\}$ for regression
    $y =$ majority of $\{y_{i1}, y_{i2}, \ldots, y_{ik}\}$ for classification, or empirical probability of each class.

Distance-weighted (kernel-based) NN

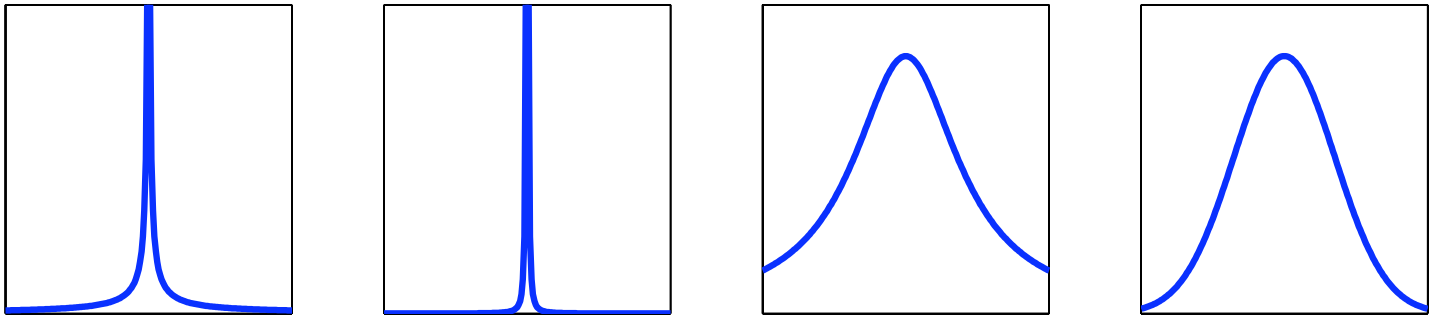
- **Given**: Training data X, distance metric d, **weighting function** $w : R \rightarrow R$.
- **Learning**: Nothing to do! (Just store the data).
- **Prediction**:
  - Given input x_{new}.
  - For each x_i in the training data X, compute $w_i = w(d(x_i, x_{new}))$.
  - Predict: $y = \sum_i w_i y_i \; / \; \sum_i w_i$.

Distance-weighted (kernel-based) NN

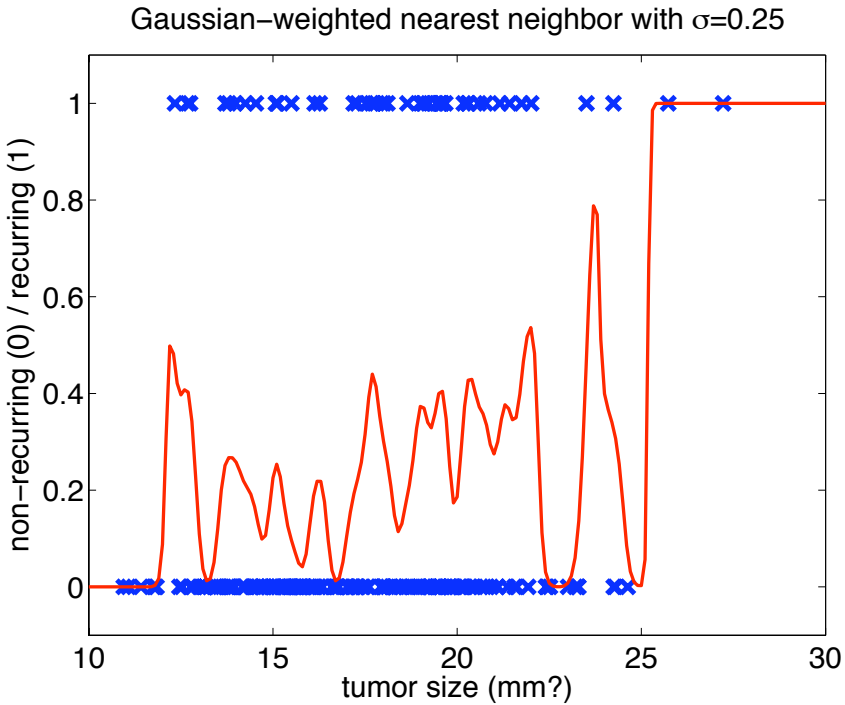
- **Given**: Training data X, distance metric d, **weighting function** $w : R \rightarrow R$.
- **Learning**: Nothing to do! (Just store the data).
- **Prediction**:
  - Given input x_{new}.
  - For each x_i in the training data X, compute $w_i = w(d(x_i, x_{new}))$.
  - Predict: $y = \sum_i w_i y_i \; / \; \sum_i w_i$.
- How should we weigh the distances?

Some weighting functions

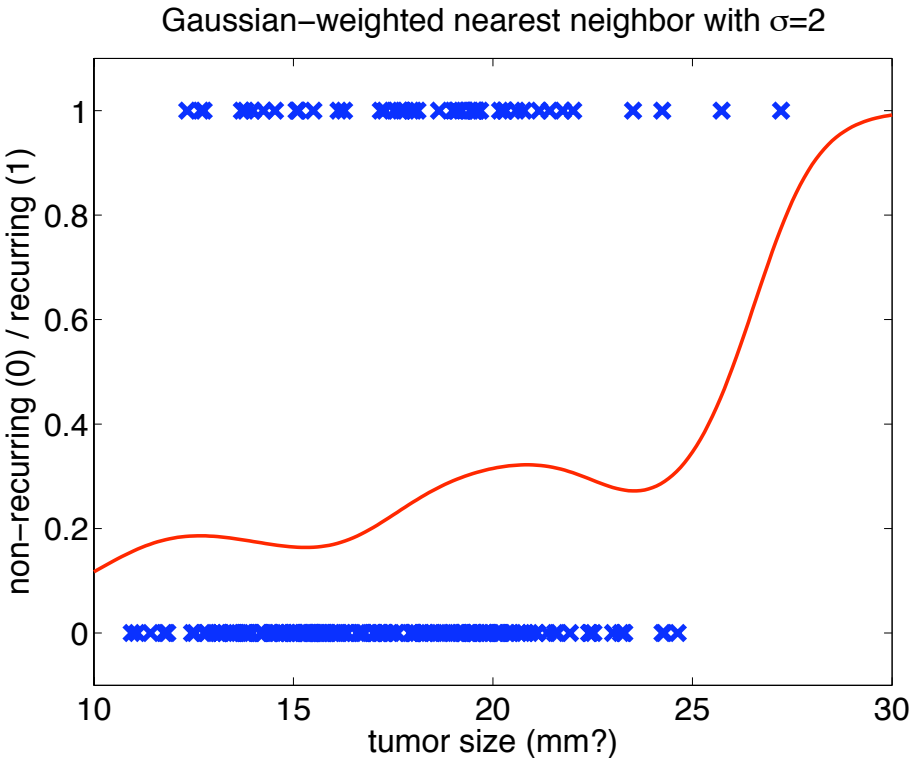
$$\frac{1}{d(\mathbf{x}_i, \mathbf{x})} \qquad \frac{1}{d(\mathbf{x}_i, \mathbf{x})^2} \qquad \frac{1}{c + d(\mathbf{x}_i, \mathbf{x})^2} \qquad e^{-\frac{d(\mathbf{x}_i, \mathbf{x})^2}{\sigma^2}}$$



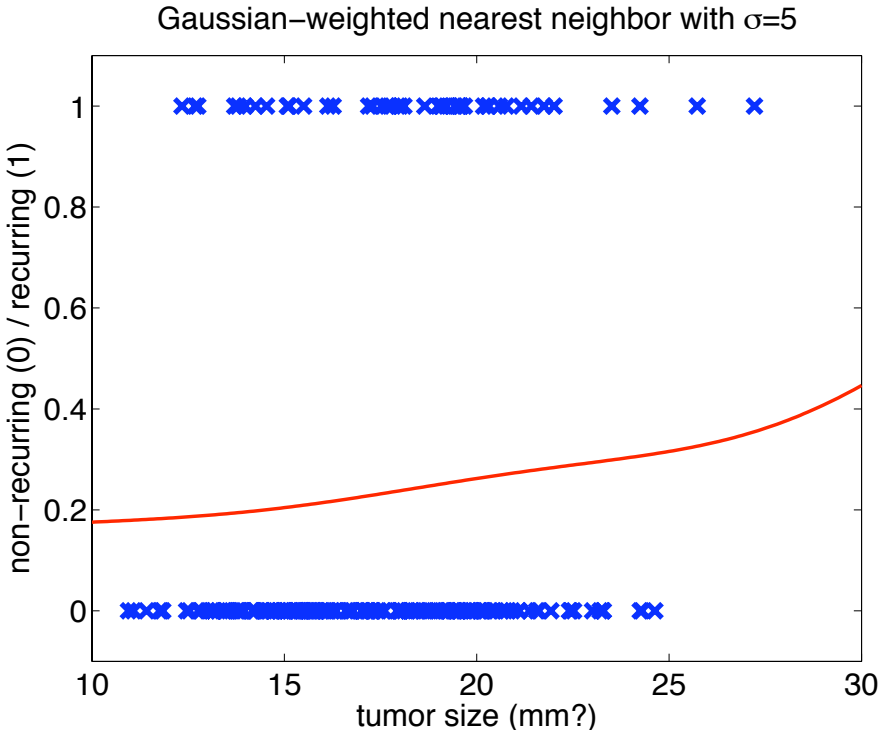
Gaussian weighting, small σ



Gaussian weighting, medium σ



Gaussian weighting, large σ



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

Lazy vs eager learning

- Lazy learning: Wait for query before generalization.
  - E.g. Nearest neighbour.

- Eager learning: Generalize before seeing query.
  - E.g. Logistic regression, LDA, decision trees, neural networks.

Pros and cons of lazy and eager learning

- Eager learners create global approximation.
- Lazy learners create many local approximations.
- If they use the same hypothesis space, a lazy learner can represent more complex functions (e.g., consider H = linear function).

Pros and cons of lazy and eager learning

- Eager learners create global approximation.
- Lazy learners create many local approximations.
- If they use the same hypothesis space, a lazy learner can represent more complex functions (e.g., consider H = linear function).

- Lazy learning has much faster training time.
- Eager learner does the work off-line, summarizes lots of data with few parameters.

Pros and cons of lazy and eager learning

- Eager learners create global approximation.
- Lazy learners create many local approximations.
- If they use the same hypothesis space, a lazy learner can represent more complex functions (e.g., consider H = linear function).

- Lazy learning has much faster training time.
- Eager learner does the work off-line, summarizes lots of data with few parameters.

- Lazy learner typically has slower query answering time (depends on number of instances and number of features) and requires more memory (must store all the data).

Scaling up

- kNN in high-dimensional feature spaces?

Scaling up

- kNN in high-dimensional feature spaces?
  - In high dim. spaces, the distance between near and far points appears similar.
  - A few points ("hubs") show up repeatedly in the top kNN (*Radovanovic et al., 2009*).

Scaling up

- kNN in high-dimensional feature spaces?
  - In high dim. spaces, the distance between near and far points appears similar.
  - A few points ("hubs") show up repeatedly in the top kNN (*Radovanovic et al., 2009*).

- kNN with larger number of datapoints?
  - Can be implemented efficiently, *O(log n)* at retrieval time, if we use smart data structures:
    - Condensation of the dataset.
    - Hash tables in which the hashing function is based on the distance metric.
    - KD-trees (Tutorial: *http://www.autonlab.org/autonweb/14665*)

When to use instance-based learning

- Instances map to points in $\mathbb{R}^n$. Or else a given distance metric.

- Not too many attributes per instance (e.g. <20), otherwise all points look at a similar distance, and noise becomes a big issue.
- Easily fooled by irrelevant attributes (for most distance metrics.)
- Can produce confidence intervals in addition to the prediction.
- Provides a variable resolution approximation (based on density of points).

Application

Hays & Efros, Scene Completion Using Millions of Photographs, CACM, 2008.
http://graphics.cs.cmu.edu/projects/scene-completion/scene_comp_cacm.pdf

What you should know

- Difference between **eager** vs **lazy** learning.
- Key idea of **non-parametric** learning.
- The **k-nearest neighbor** algorithm for classification and regression, and its properties.
- The distance-weighted NN algorithm and locally-weighted linear regression.