Machine Learning (CS 567) Lecture 8

Fall 2008

Time: T-Th 5:00pm - 6:20pm
Location: GFS 118

Instructor: Sofus A. Macskassy (macskass@usc.edu)
Office: SAL 216
Office hours: by appointment

Teaching assistant: Cheol Han (cheolhan@usc.edu)
Office: SAL 229
Office hours: M 2-3pm, W 11-12

Class web page:
http://www-scf.usc.edu/~csci567/index.html

Lecture 8 Outline

- Nearest Neighbor Method

The Top Five Algorithms

- Decision trees (C4.5)
- Nearest Neighbor Method
- Neural networks (backpropagation)
- Probabilistic networks (Naïve Bayes; Mixture models)
- Support Vector Machines (SVMs)

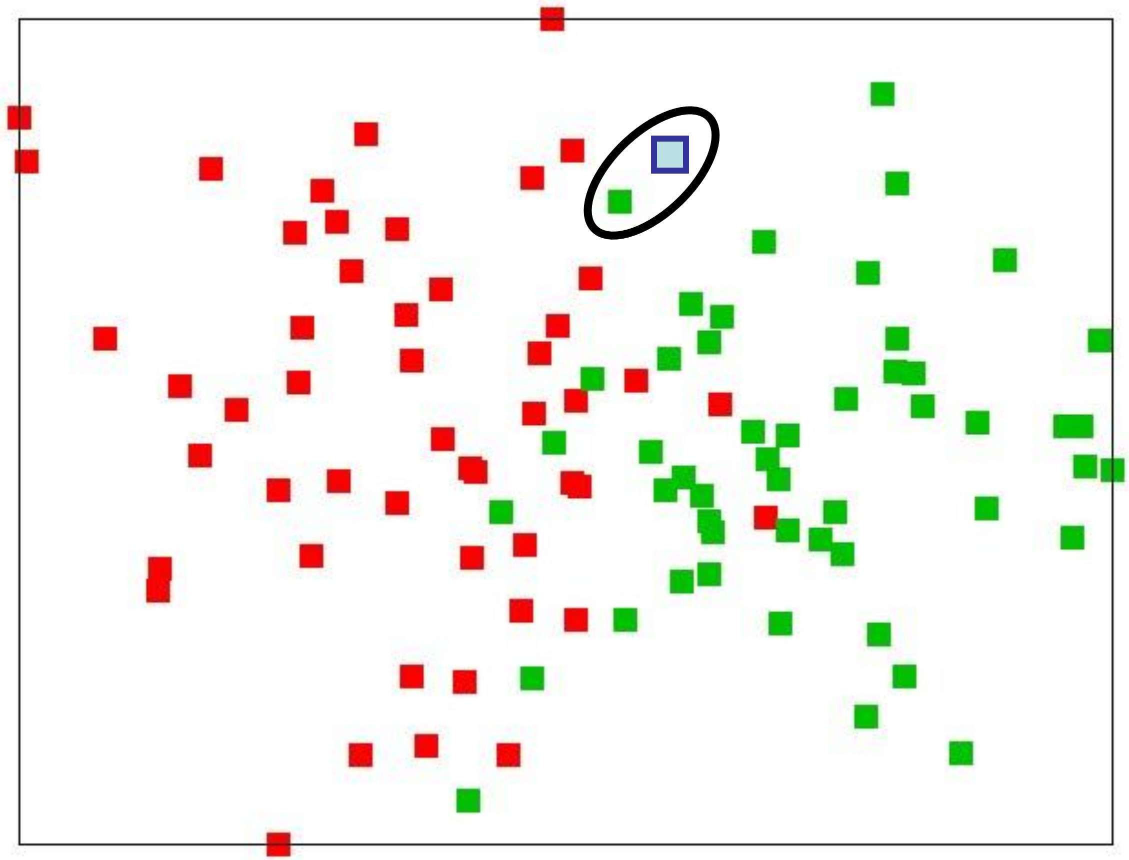
Summary so far

Criterion	Perc	Logistic	LDA	Trees
Mixed data	no	no	no	yes
Missing values	no	no	yes	yes
Outliers	no	yes	no	yes
Monotone transformations	no	no	no	yes
Scalability	yes	yes	yes	yes
Irrelevant inputs	no	no	no	somewhat
Linear combinations	yes	yes	yes	no
Interpretable	yes	yes	yes	yes
Accurate	yes	yes	yes	no

The Nearest Neighbor Algorithm

- Hypothesis Space
  - variable size
  - deterministic
  - continuous parameters
- Learning Algorithm
  - direct computation
  - lazy

The Nearest Neighbor Algorithm



Nearest Neighbor Algorithm

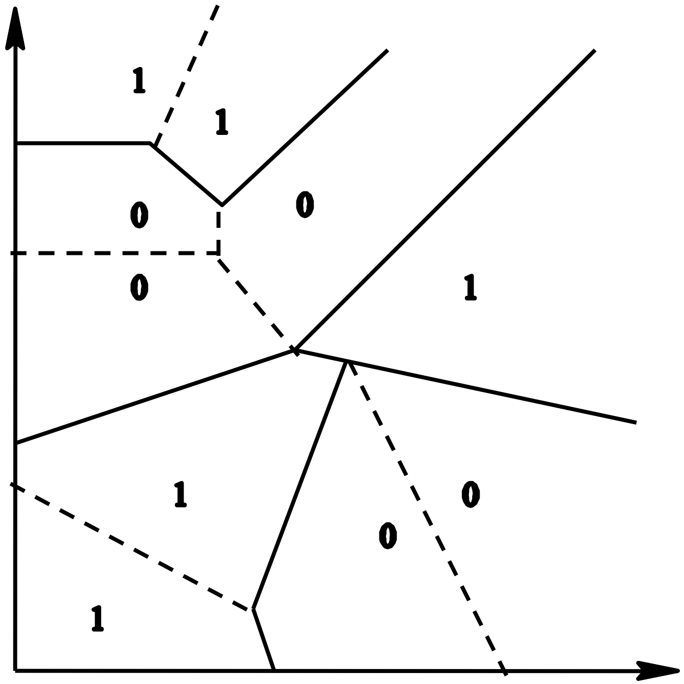
- Store all of the training examples
- Classify a new example **x** by finding the training example $\langle \mathbf{x}_i, y_i \rangle$ that is nearest to **x** according to some distance metric (e.g. Euclidean distance):

$$\|\mathbf{x} - \mathbf{x}_i\| = \sqrt{\sum_j (x_j - x_{ij})^2}$$

  guess the class $\hat{y} = y_i$.
- Efficiency trick: squared Euclidean distance gives the same answer but avoids the square root computation

$$\|\mathbf{x} - \mathbf{x}_i\|^2 = \sum_j (x_j - x_{ij})^2$$

Decision Boundaries: The Voronoi Diagram



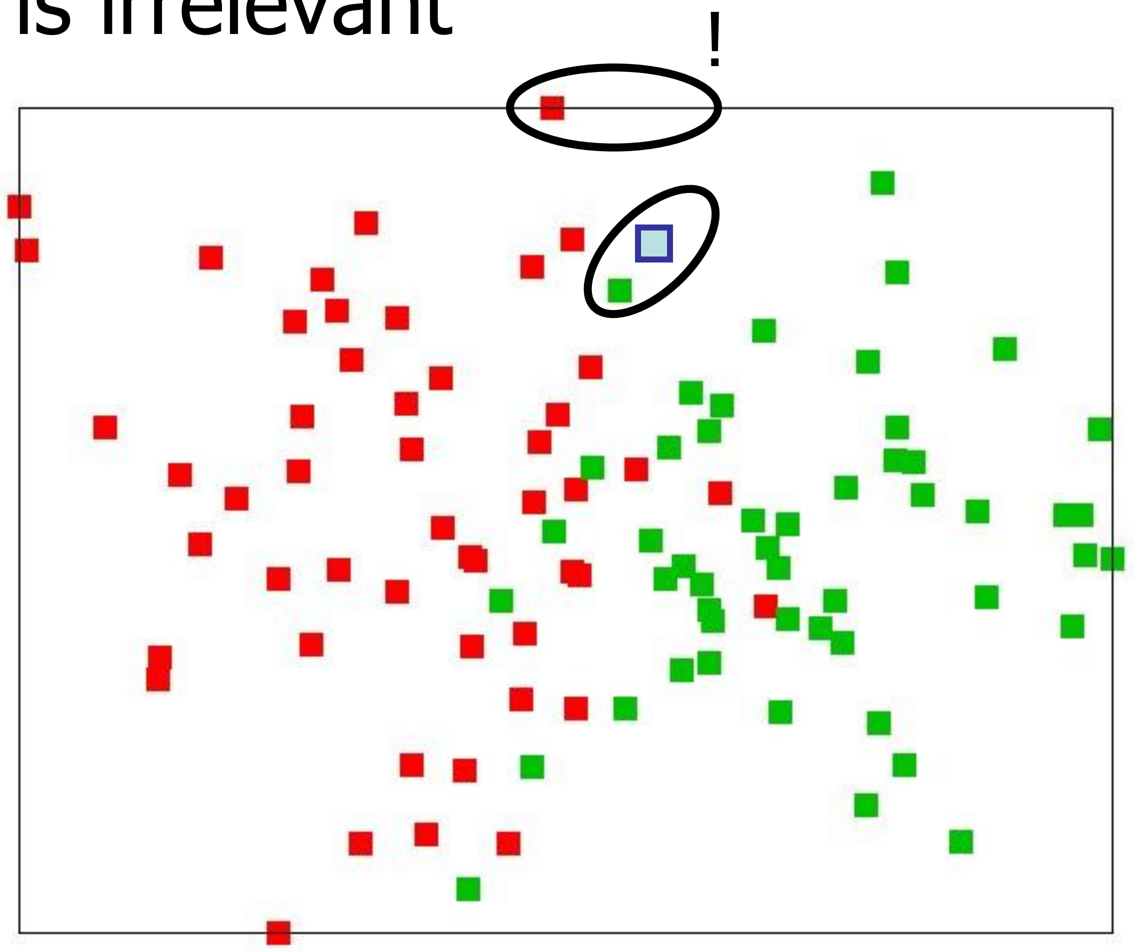
- Nearest Neighbor does not explicitly compute decision boundaries. However, the boundaries form a subset of the Voronoi diagram of the training data
- Each line segment is equidistant between two points of opposite class. The more examples that are stored, the more complex the decision boundaries can become.

Nearest Neighbor depends critically on the distance metric

- Normalize Features:
  - Otherwise features with large ranges could have a disproportionate effect on the distance metric.
- Remove Irrelevant Features:
  - Irrelevant or noisy features add random perturbations to the distance measure and hurt performance
- Learn a Distance Metric:
  - One approach: weight each feature by its mutual information with the class. Let $w_j = I(y \mid x_j)$. Then $d(\mathbf{x},\mathbf{x}') = \sum_{j=1}^{n} w_j(x_j - x'_j)^2$
  - Another approach: Use the Mahalanobis distance: $D_M(\mathbf{x},\mathbf{x}') = (\mathbf{x} - \mathbf{x}')^T\Sigma^{-1}(\mathbf{x} - \mathbf{x}')$
- Smoothing:
  - Find the *k* nearest neighbors and have them vote. This is especially good when there is noise in the class labels.

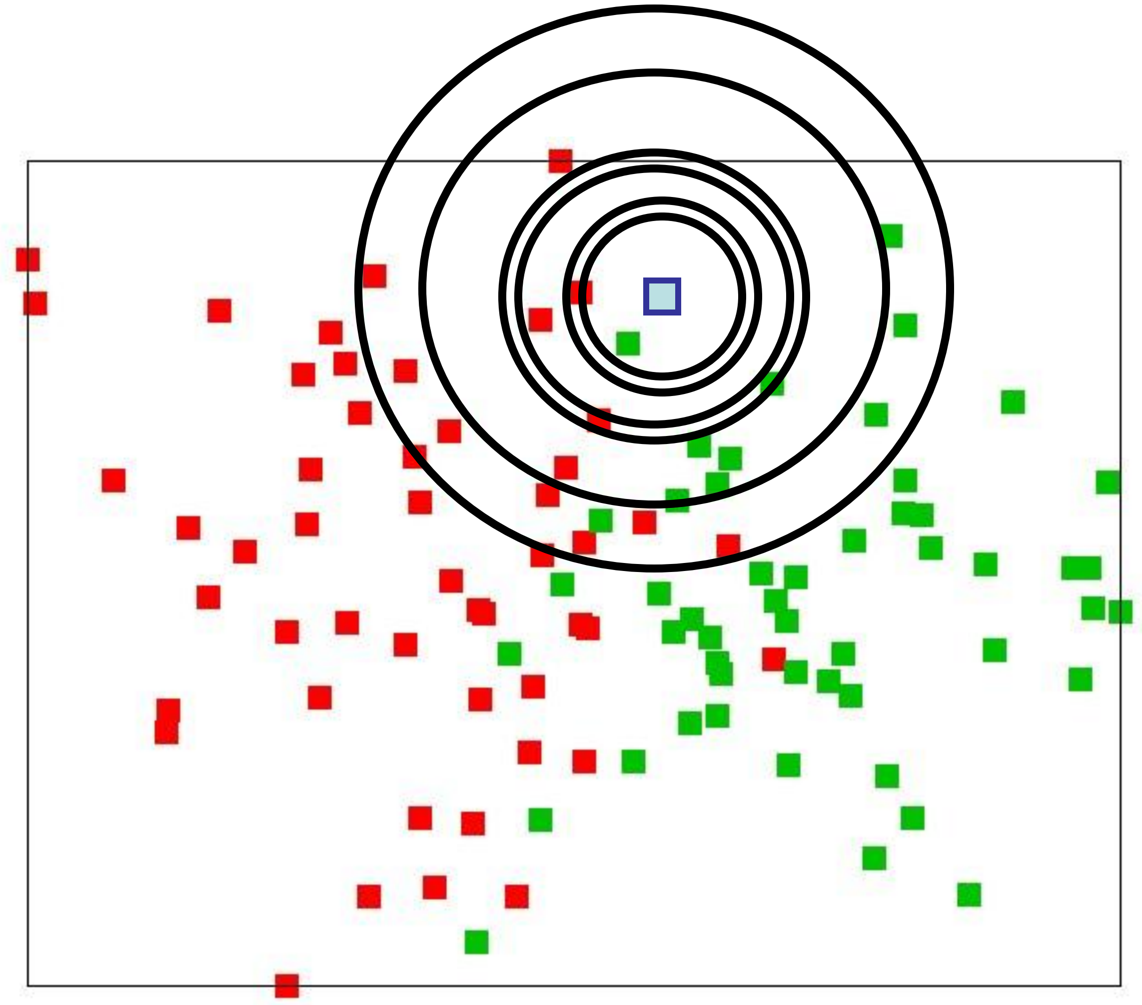
k-NN: Irrelevant features

- y-axis is irrelevant



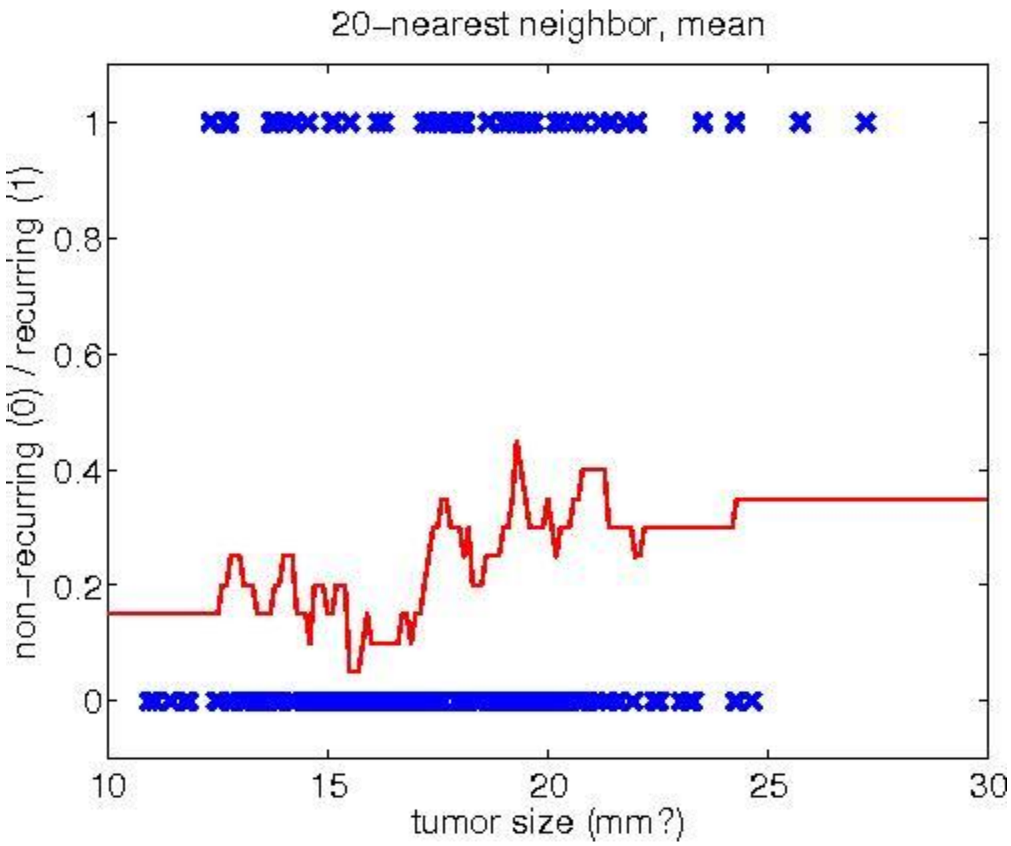
k-NN: Smoothing

- k=4.

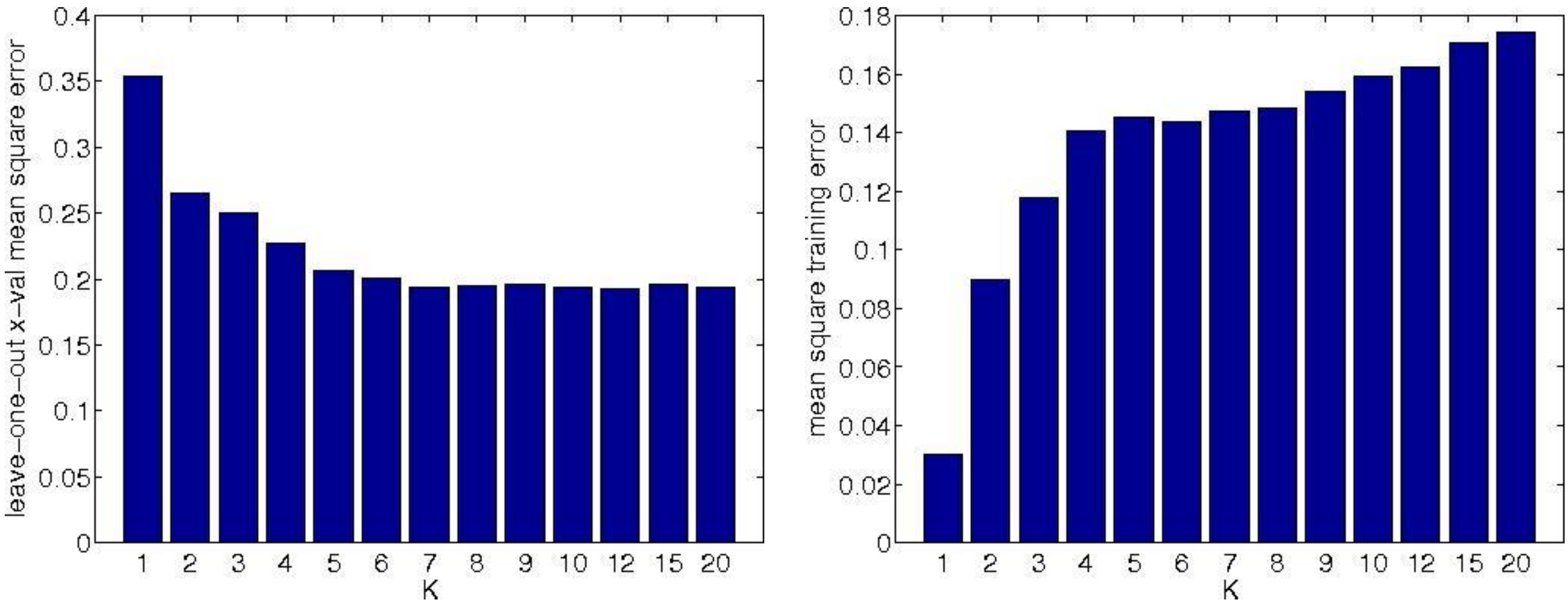


k nearest neighbors example

- Smoothing from k = 1 to 20



Choosing *k*



Pick best value according to the error on the validation set

Distance-weighted nearest neighbor

- Inputs: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ distance metric d on $\mathcal{X}$, weighting function

$$w : \Re \mapsto \Re$$

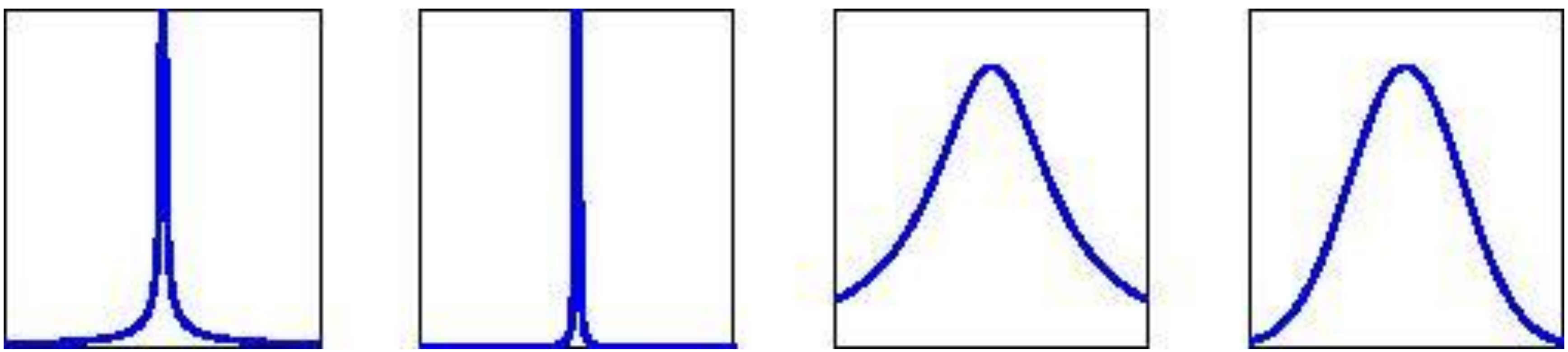
- Learning: Nothing to do!
- Prediction: On input **x**,
  - For each i compute $w_i = w(d(\mathbf{x}_i, \mathbf{x}))$.
  - Predict weighted majority or mean. For example,

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

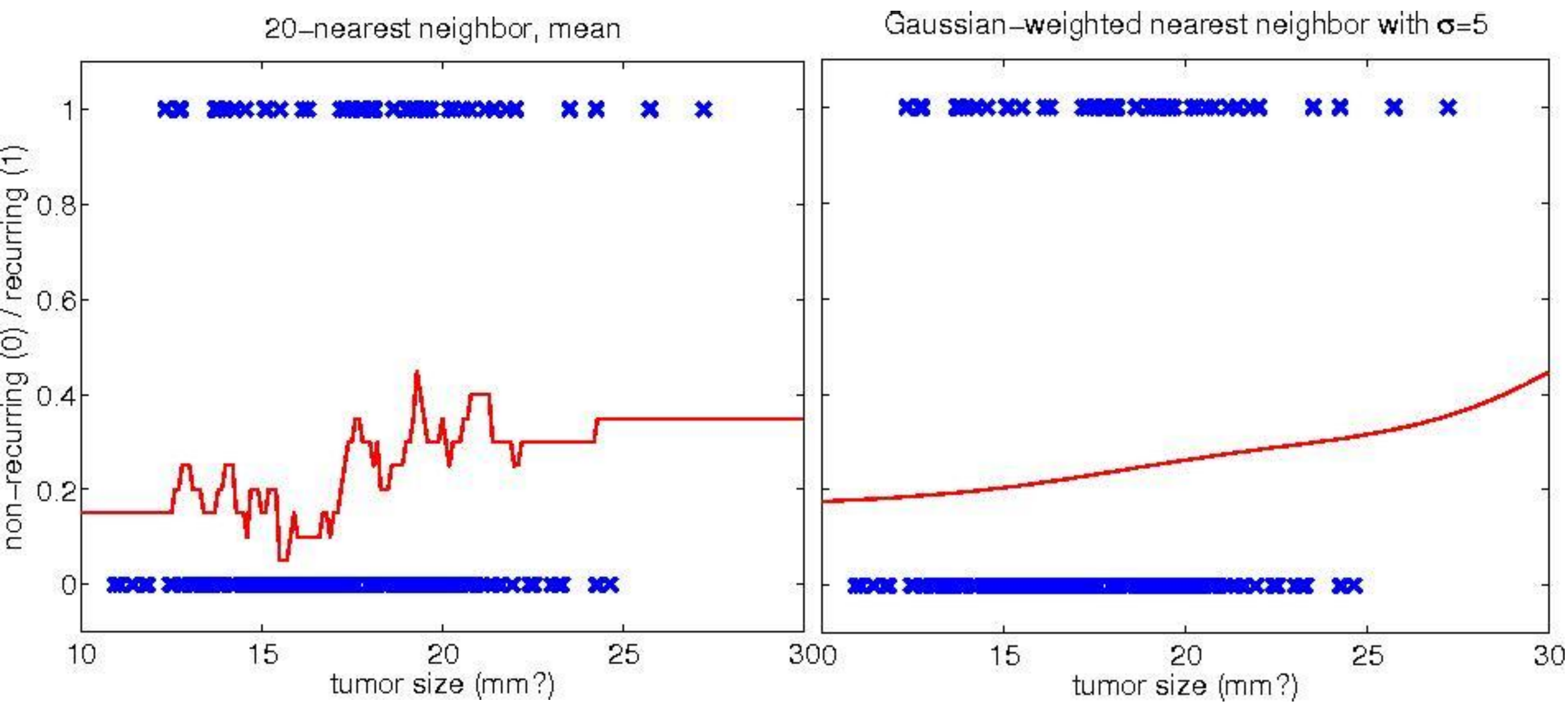
How to weight distances?

Some weighting functions

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



Example: Gaussian weighting, small σ

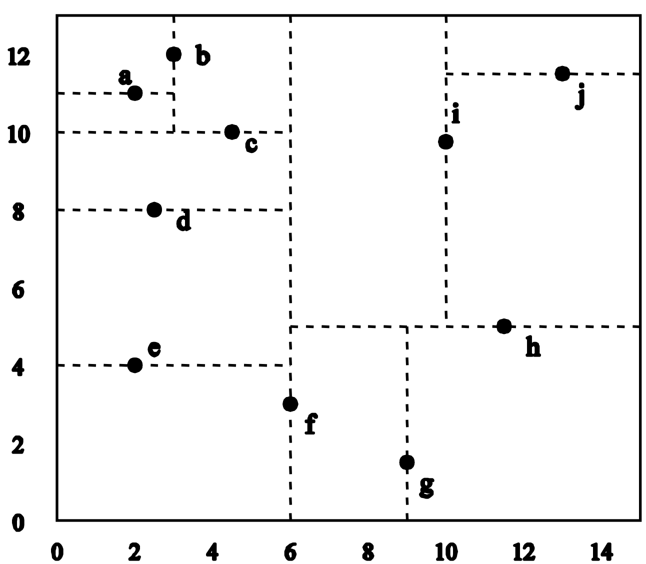


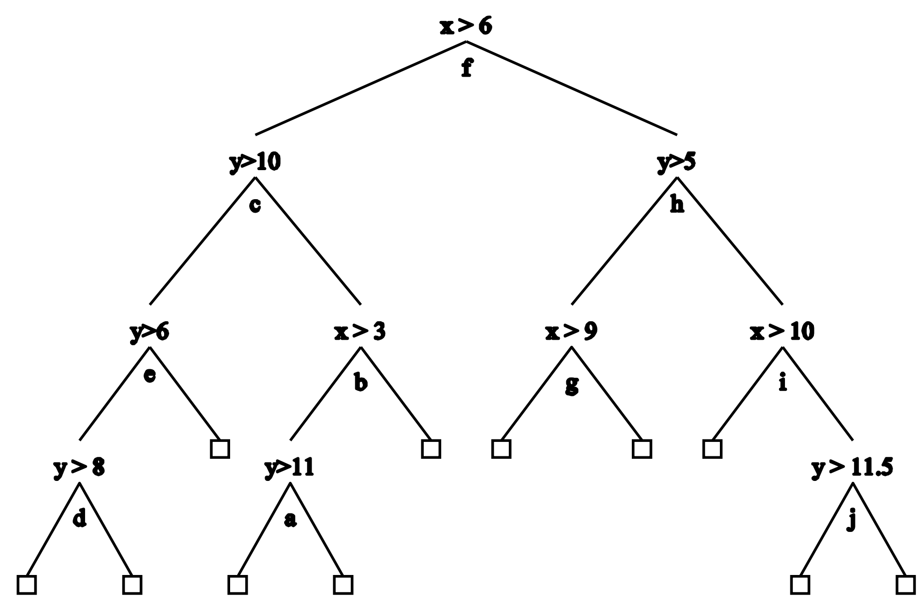
Reducing the Cost of Nearest Neighbor

- Efficient Data Structures for Retrieval (kd-trees)
- Selectively Storing Data Points (editing)
- Pipeline of Filters

kd trees

- A kd-tree is similar to a decision tree except that we 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





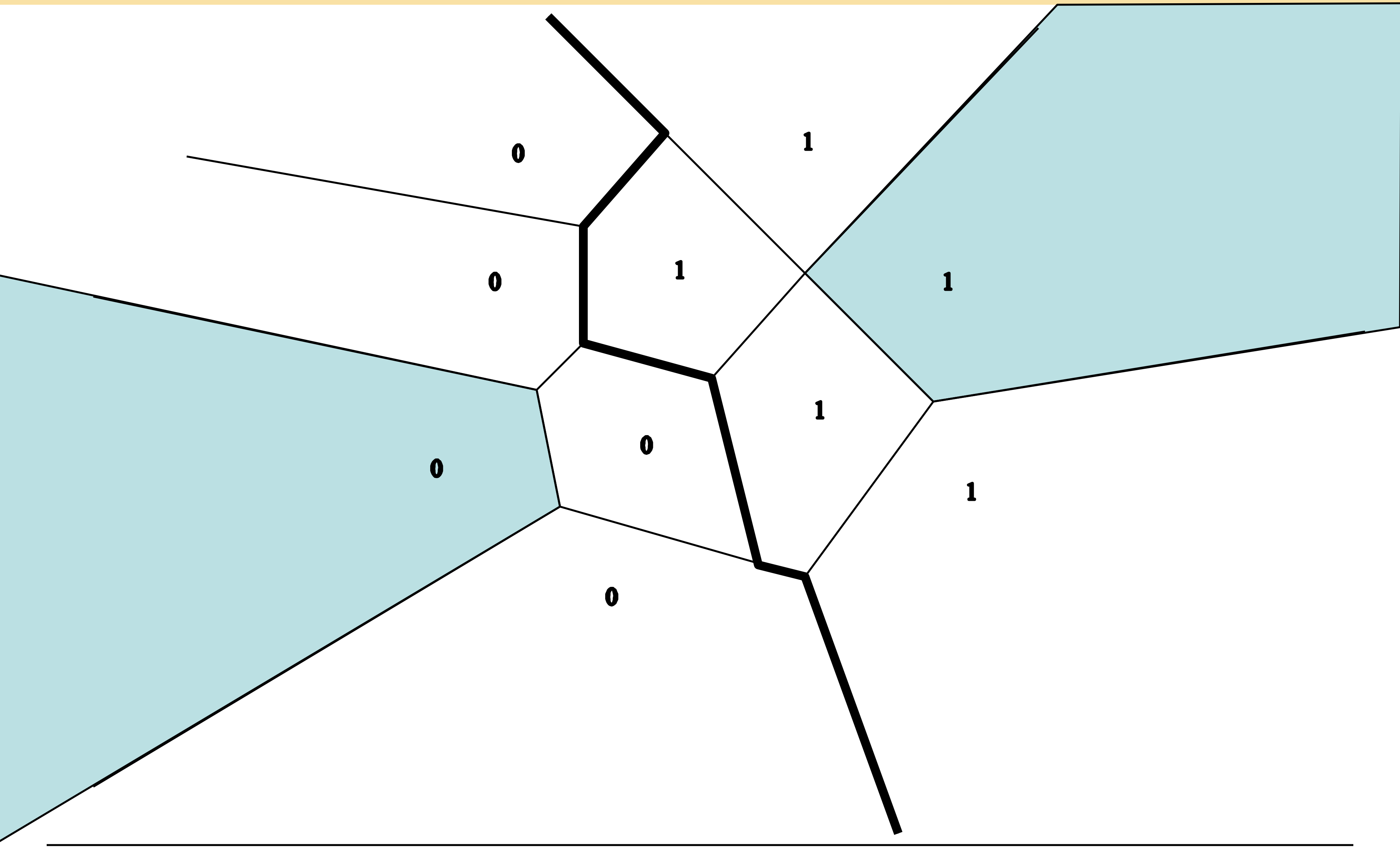
Features of kd-trees

- Makes it easy to do 1-nearest neighbor
- 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

Edited Nearest Neighbor

- Select a subset of the training examples that still gives good classifications
  - Incremental deletion: Loop through the memory and test each point to see if it can be correctly classified given the other points in memory. If so, delete it from the memory.
  - Incremental growth. Start with an empty memory. Add each point to the memory only if it is not correctly classified by the points already stored

Decision Boundaries: The Voronoi Diagram

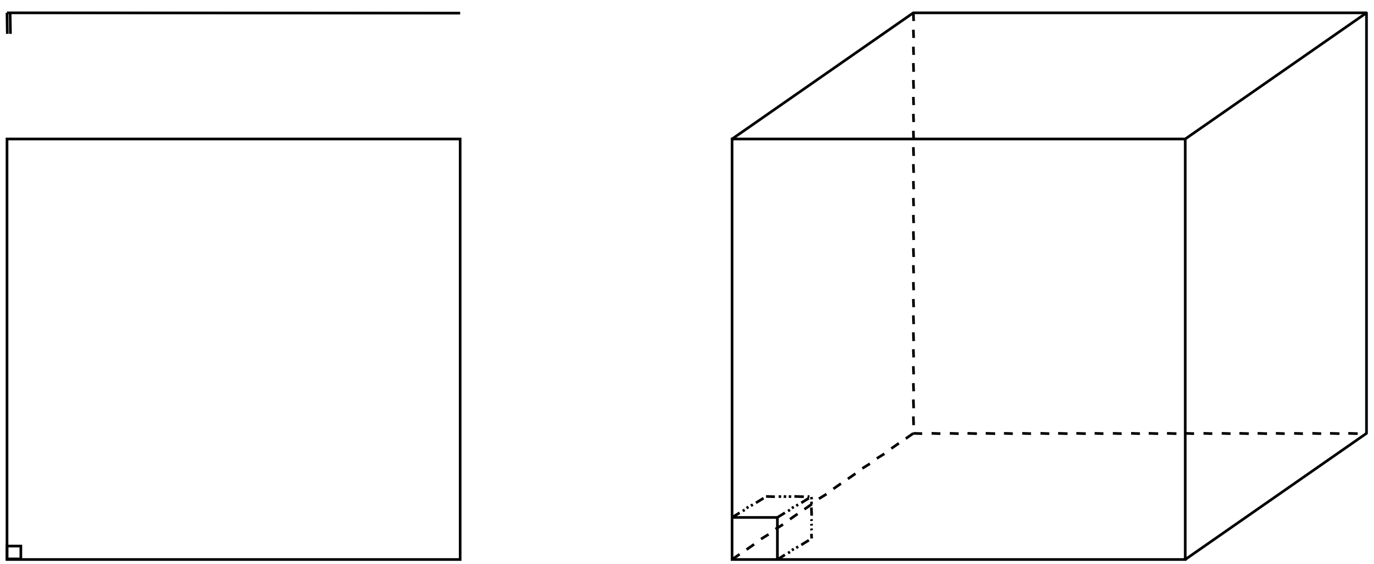


Filter Pipeline

- Consider several distance measures: D_1, D_2, …, D_n where D_{i+1} is more expensive to compute than D_i
- Calibrate a threshold N_i for each filter using the training data
- Apply the nearest neighbor rule with D_i to compute the N_i nearest neighbors
- Then apply filter D_{i+1} to those neighbors and keep the N_{i+1} nearest, and so on

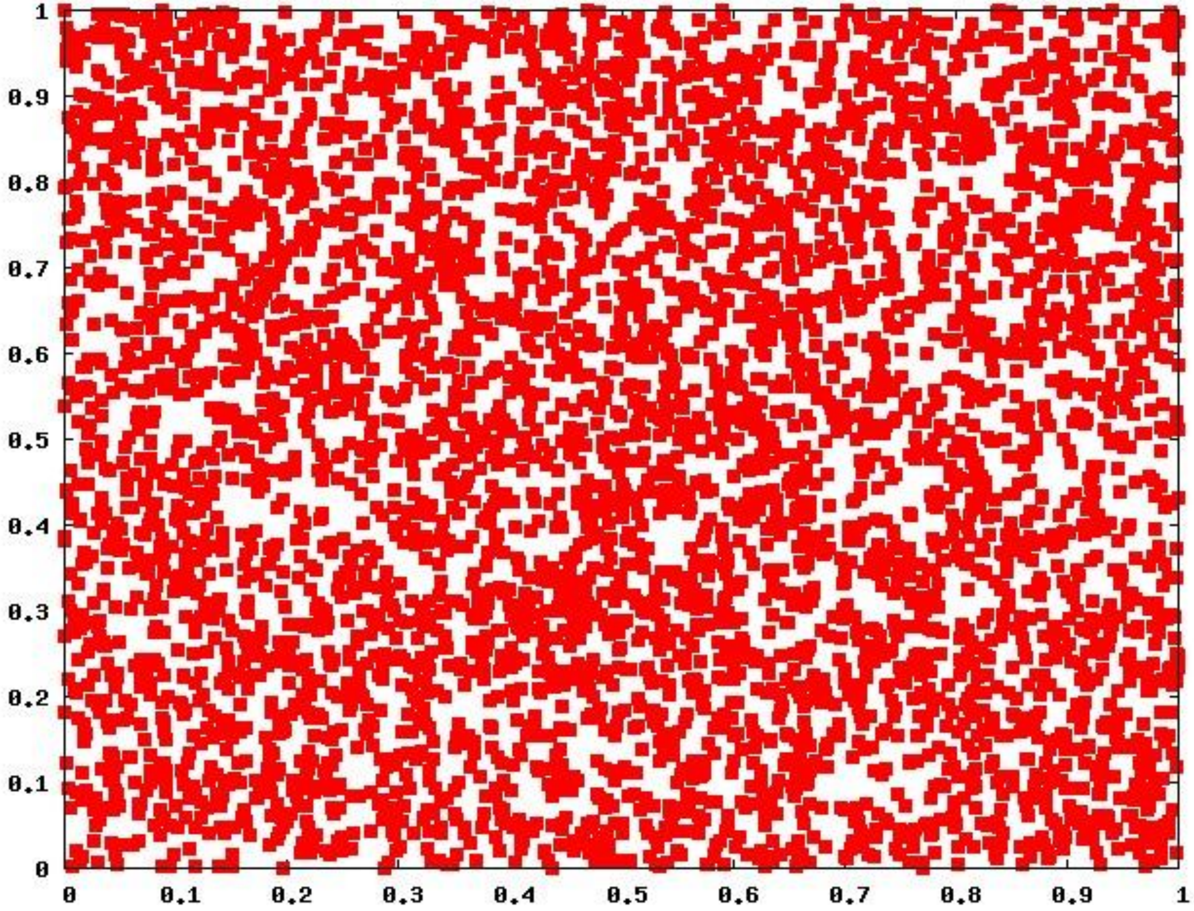
The Curse of Dimensionality

- Nearest neighbor breaks down in high-dimensional spaces, because the "neighborhood" becomes very large.
- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply the 5-nearest neighbor algorithm. Suppose our query point is at the origin.
- Then on the 1-dimensional line, we must go a distance of 5/5000 = 0.001 on the average to capture the 5 nearest neighbors
- In 2 dimensions, we must go $\sqrt{0.001} = 0.0316$ to get a square that contains 0.001 of the volume.
- In D dimensions, we must go $(0.001)^{1/d}$ $[(0.001)^{1/30} = 0.794!]$

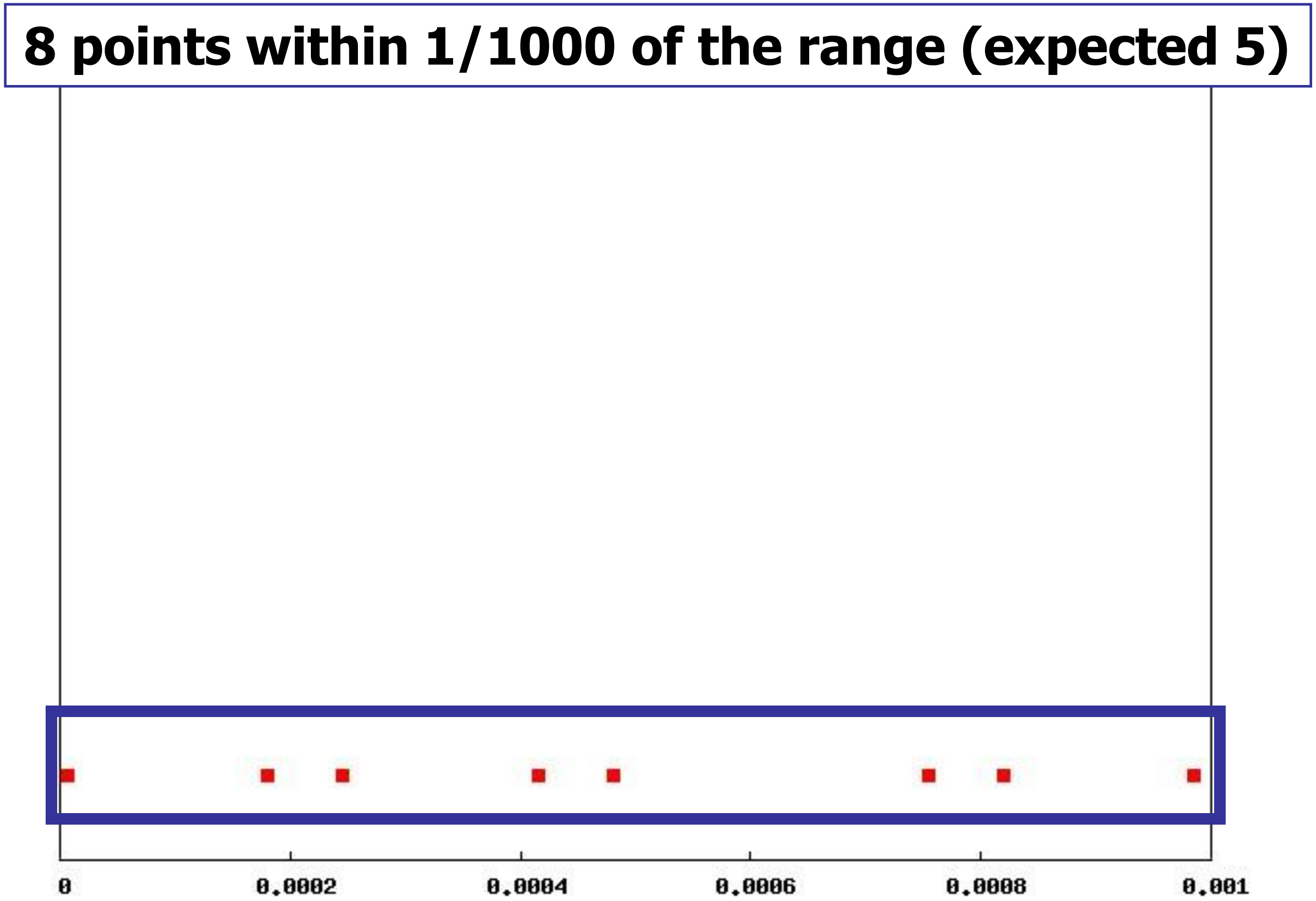


The Curse of Dimensionality (2)

- 5000 points in unit square

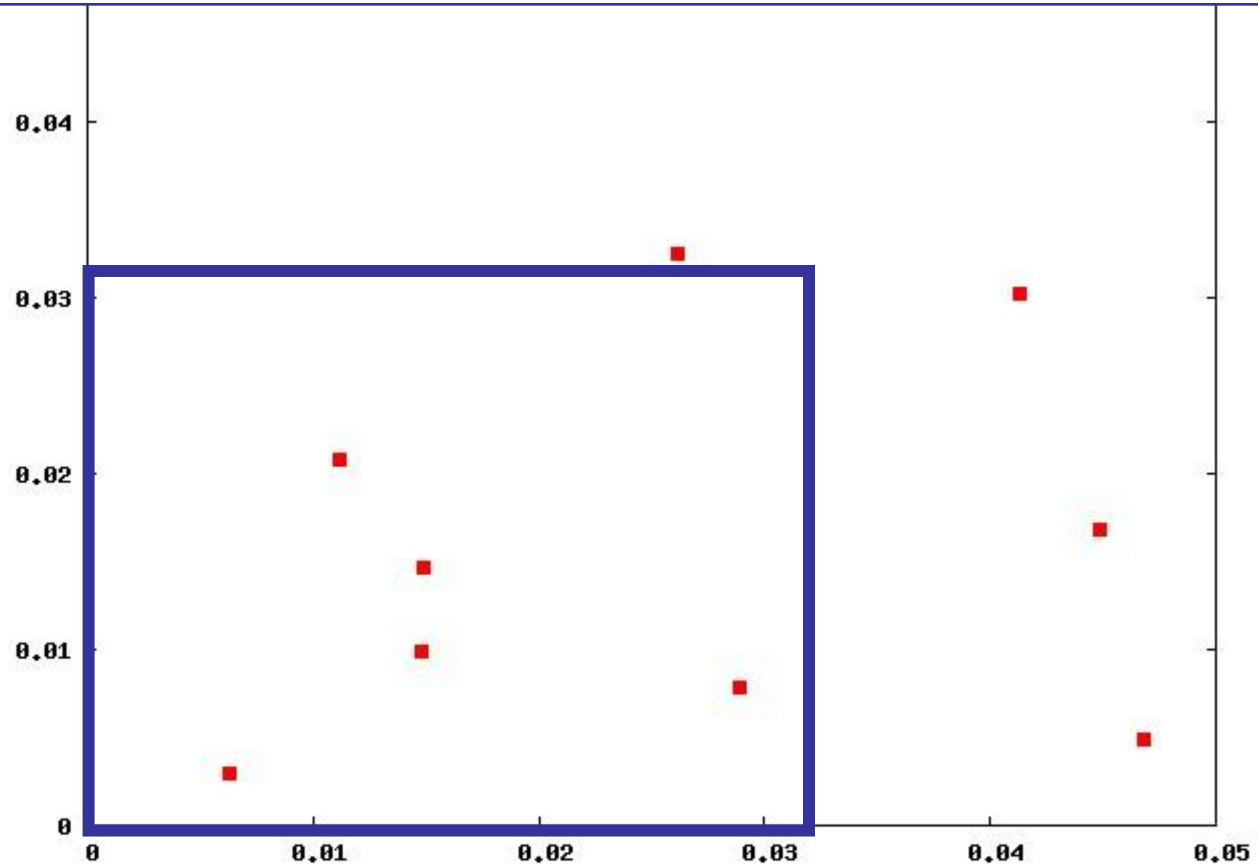


The Curse of Dimensionality (3)



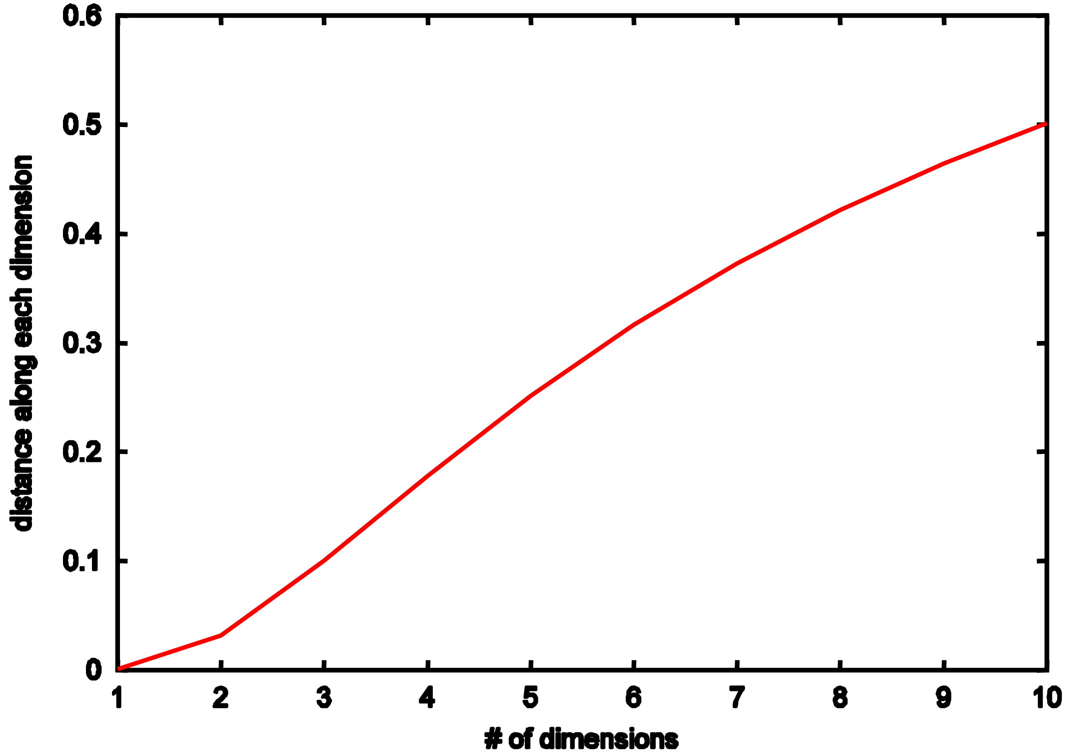
The Curse of Dimensionality (4)

5 points within 1/1000 of the 2D area (expected 5)



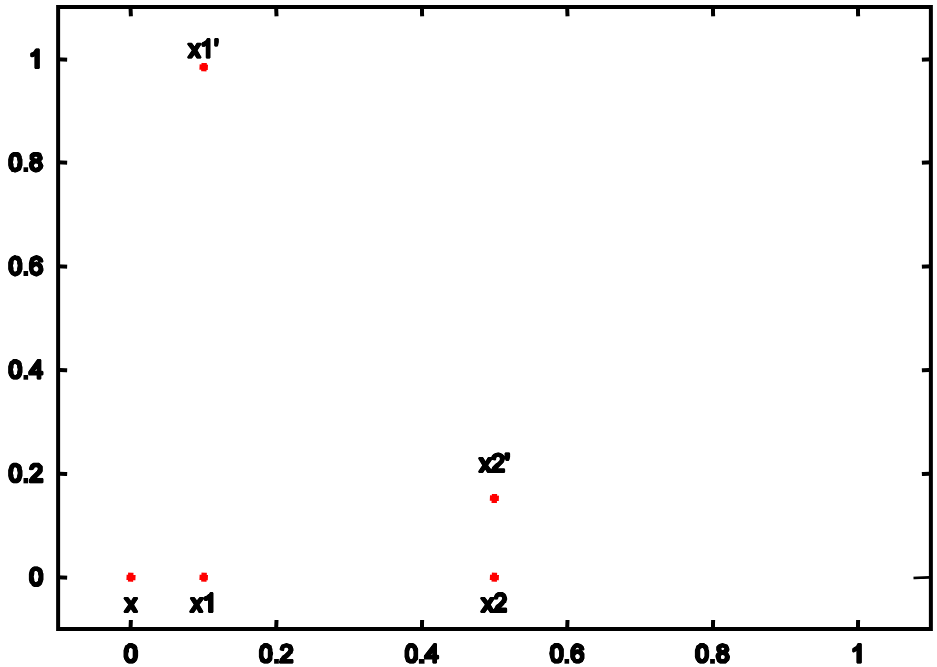
The Curse of Dimensionality (5)

- With 5000 points in 10 dimensions, we must go 0.501 distance along each attribute in order to find the 5 nearest neighbors

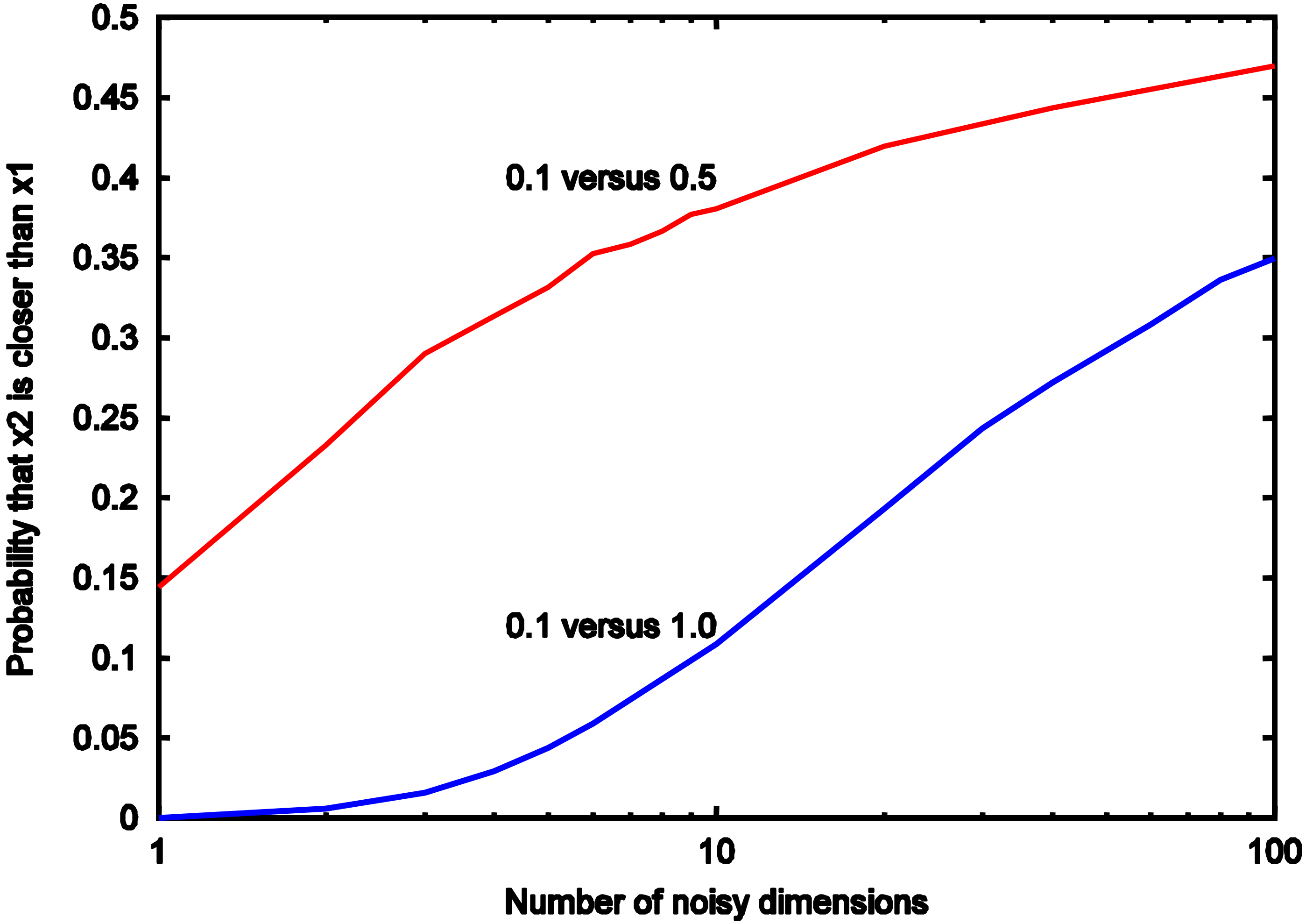


The Curse of Noisy/Irrelevant Features

- NNbr also breaks down when the data contains irrelevant, noisy features.
- Consider a 1D problem where our query *x* is at the origin, our nearest neighbor is *x1* at 0.1, and our second nearest neighbor is *x2* at 0.5.
- Now add a uniformly random noisy feature. What is the probability that *x2'* will now be closer to *x* than *x1'*? Approximately 0.15.



Curse of Noise (2)
Location of *x1* versus *x2*



Nearest Neighbor Summary

- Advantages
  - variable-sized hypothesis space
  - learning is extremely efficient and can be online or batch
    - However, growing a good kd-tree can be expensive
  - Very flexible decision boundaries
- Disadvantages
  - distance function must be carefully chosen
  - irrelevant or correlated features must be eliminated
  - typically cannot handle more than 30 features
  - computational costs: memory and classification-time computation

Nearest Neighbor Evaluation

Criterion	Perc	Logistic	LDA	Trees	NNbr
Mixed data	no	no	no	yes	no
Missing values	no	no	yes	yes	somewhat
Outliers	no	yes	no	yes	yes
Monotone transformations	no	no	no	yes	no
Scalability	yes	yes	yes	yes	no
Irrelevant inputs	no	no	no	somewhat	no
Linear combinations	yes	yes	yes	no	somewhat
Interpretable	yes	yes	yes	yes	no
Accurate	yes	yes	yes	no	no