ITCS 5356: Machine Learning

k-Nearest Neighbor Algorithms

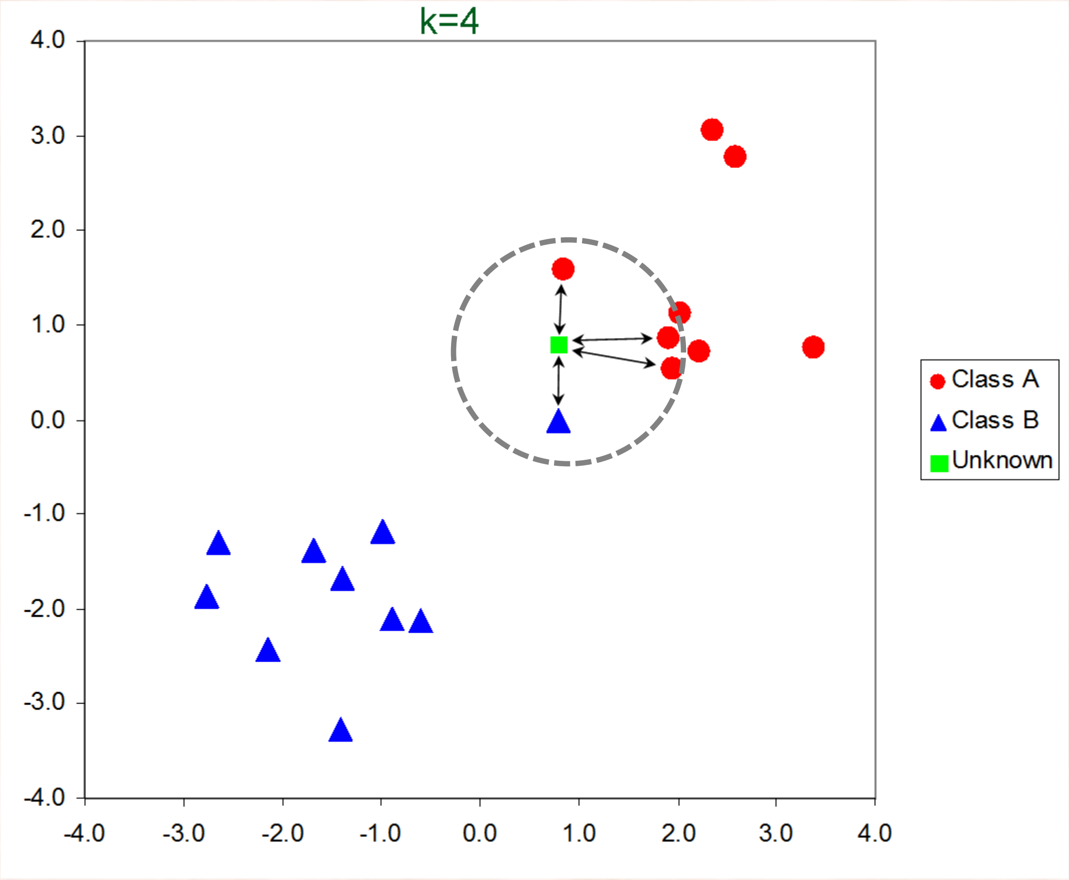
Razvan C. Bunescu

Department of Computer Science @ CCI

rbunescu@uncc.edu

k-Nearest Neighbors (kNN)

- Euclidean *distance*, $k = 4$



Nonparametric Methods: k-Nearest Neighbors

Input:

- A training dataset $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots (\mathbf{x}_n, t_n)$.
- A test instance $\mathbf{x}$.

Output:

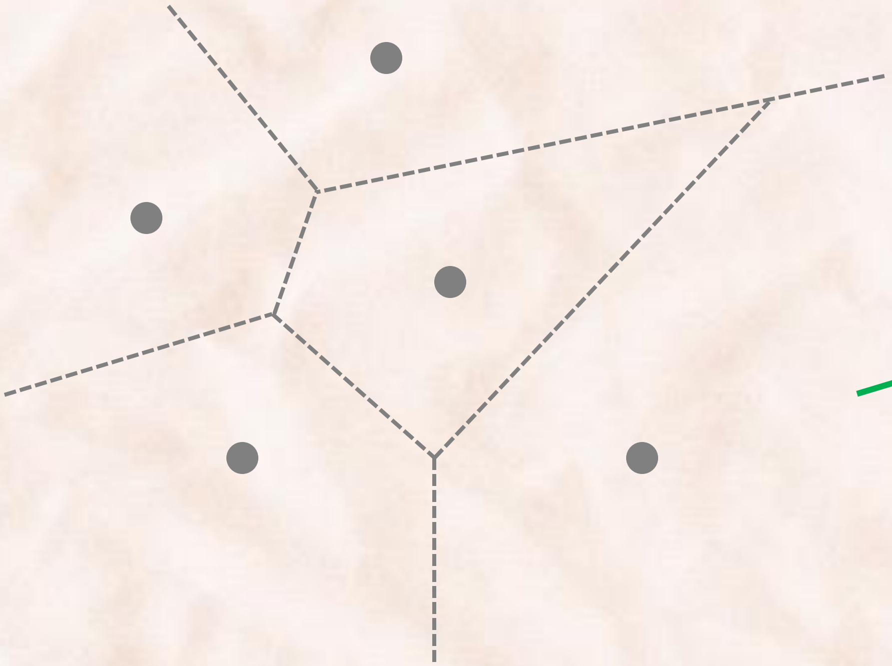
- Estimated class label $y(\mathbf{x})$.

1. Find k instances $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ *nearest* to $\mathbf{x}$.
2. Let $y(x) = \arg\max_{t \in T} \sum_{i=1}^{k} \delta_t(t_i)$

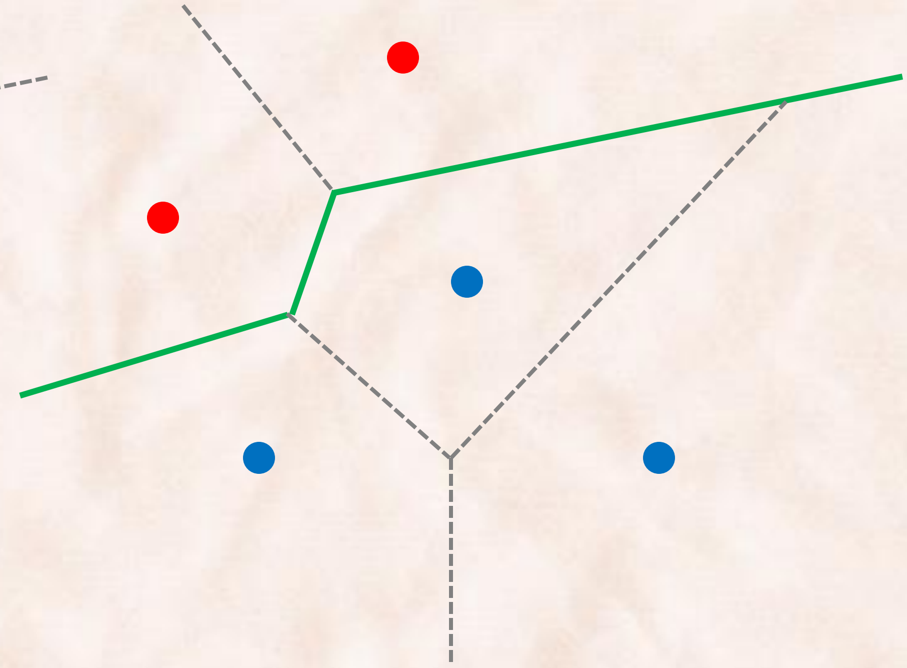
where $\delta_t(x) = \begin{cases} 1 & x = t \\ 0 & x \neq t \end{cases}$ is the *Kronecker delta* function.

k-Nearest Neighbors (k-NN)

- Euclidian distance, k = 1.



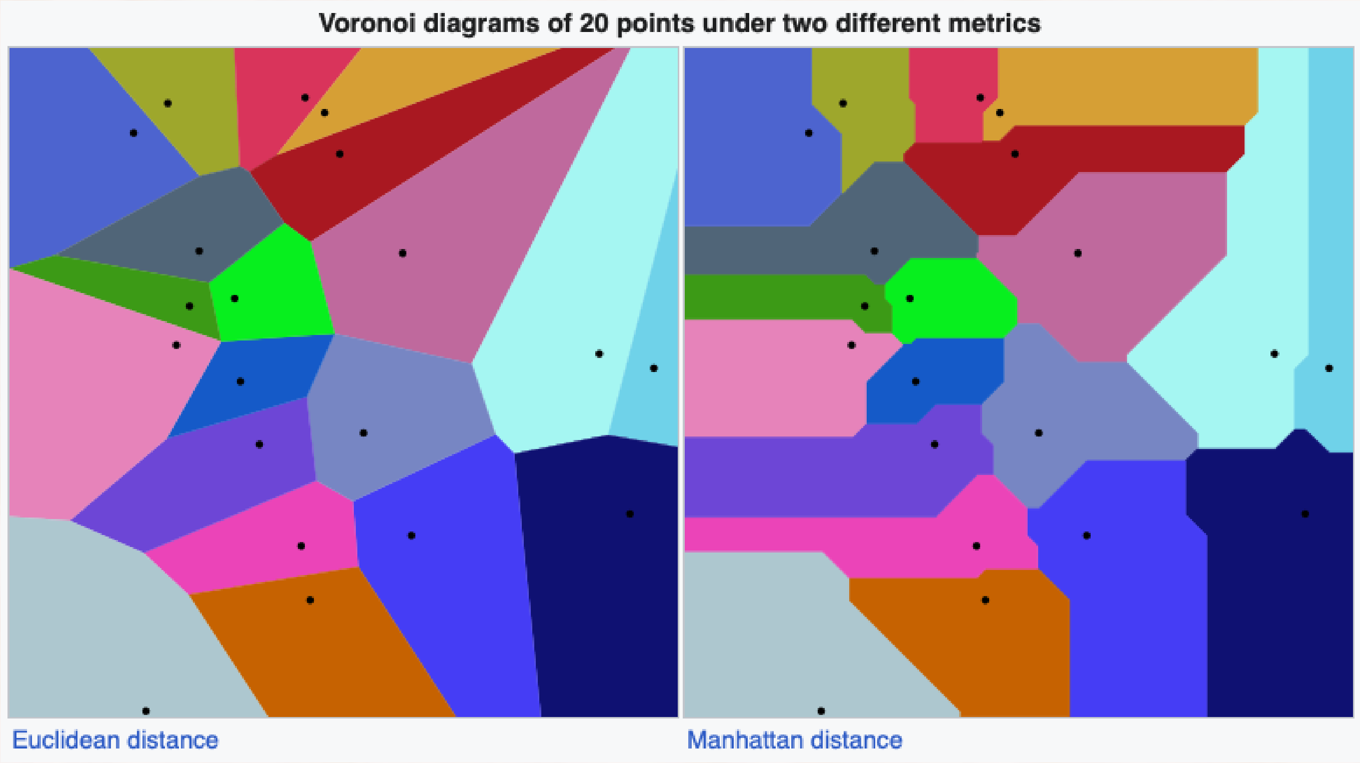
Voronoi diagram



decision boundary

Voronoi Diagrams

- The Voronoi diagram depends on the distance measure:



https://en.wikipedia.org/wiki/Voronoi_diagram

Distance Metrics

- Euclidean distance:

$$d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2 = \sqrt{(\mathbf{x}-\mathbf{y})^T(\mathbf{x}-\mathbf{y})}$$

- Hamming distance:

  # of (discrete) features that have different values in **x** and **y**.

- Mahalanobis distance:

(sample) covariance matrix

$$d(\mathbf{x},\mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^T S^{-1}(\mathbf{x}-\mathbf{y})}$$

  – scale-invariant metric that normalizes for variance.

  – if $S = I \Rightarrow$ Euclidean distance.

  – if $S = diag(\sigma_1^{-2}, \sigma_2^{-2}, \ldots \sigma_K^{-2}) \Rightarrow$ *normalized* Euclidean distance.

Distance Metrics

- Cosine similarity:

$$d(\mathbf{x},\mathbf{y}) = 1 - \cos(\mathbf{x},\mathbf{y}) = 1 - \frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$$

  – used for text and other high-dimensional data.

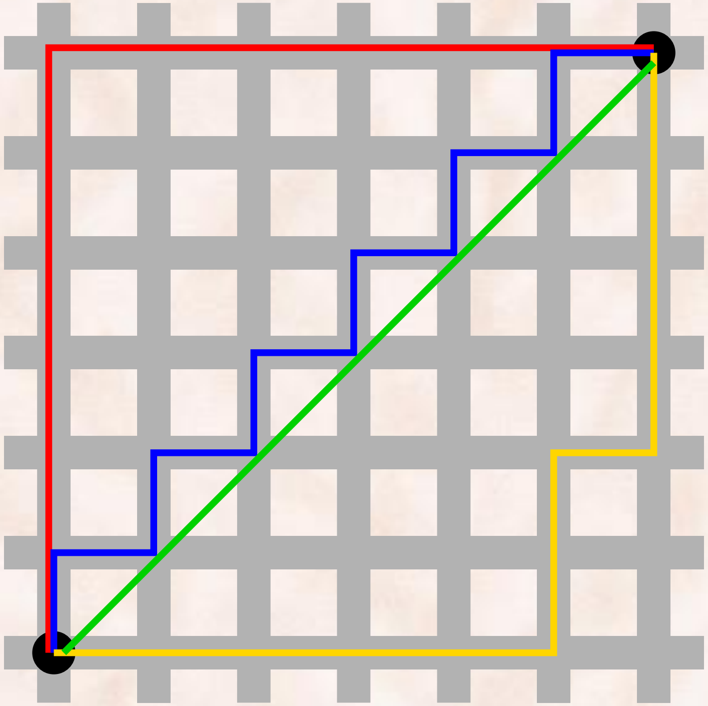
- Levenshtein distance (Edit distance):
  – distance metric on strings (sequences of symbols).
  – min. # of basic edit operations that can transform one string into the other (delete, insert, substitute).

$$\left.\begin{aligned}\mathbf{x} &= \text{“athens”}\\ \mathbf{y} &= \text{“hints”}\end{aligned}\right\} \Rightarrow d(\mathbf{x},\mathbf{y}) = 4$$

  – used in bioinformatics.

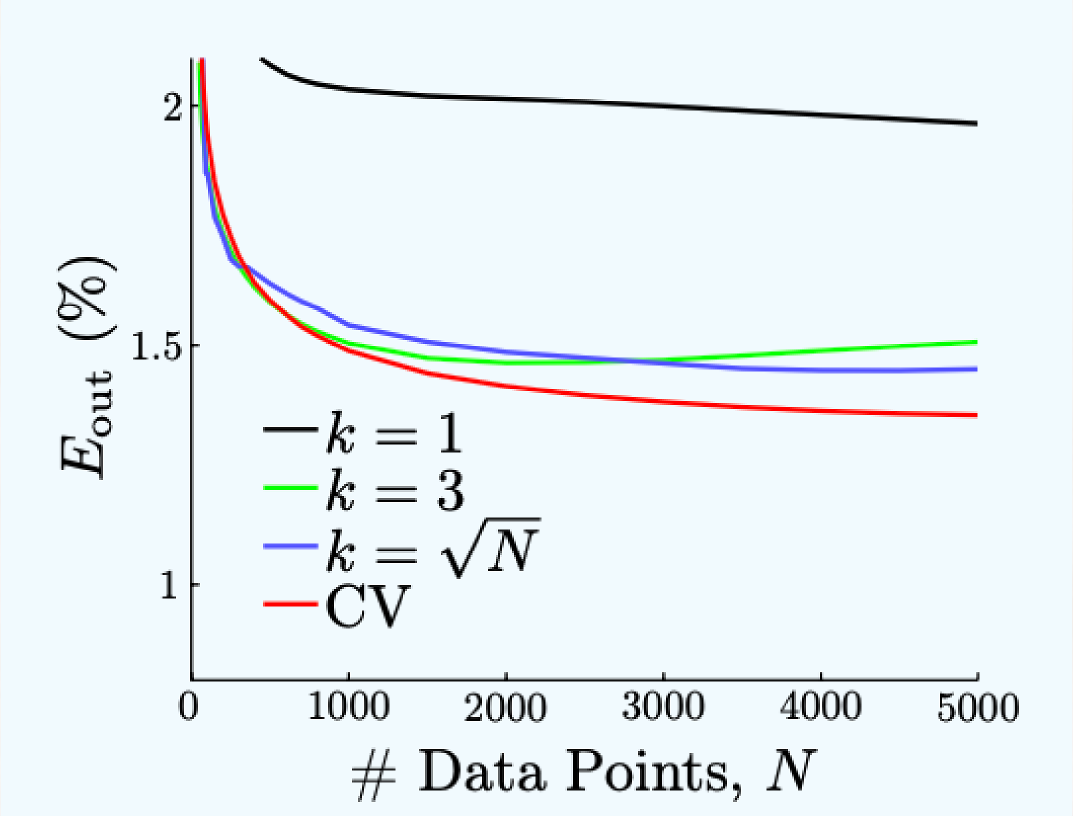
Distance metrics

- Manhattan distance: $d(\mathbf{x},\mathbf{y}) = \sum_{k=1}^{K} |x_k - y_k|$



How to choose k?

- The value of *k* can be chosen using *grid search* on *development* data.

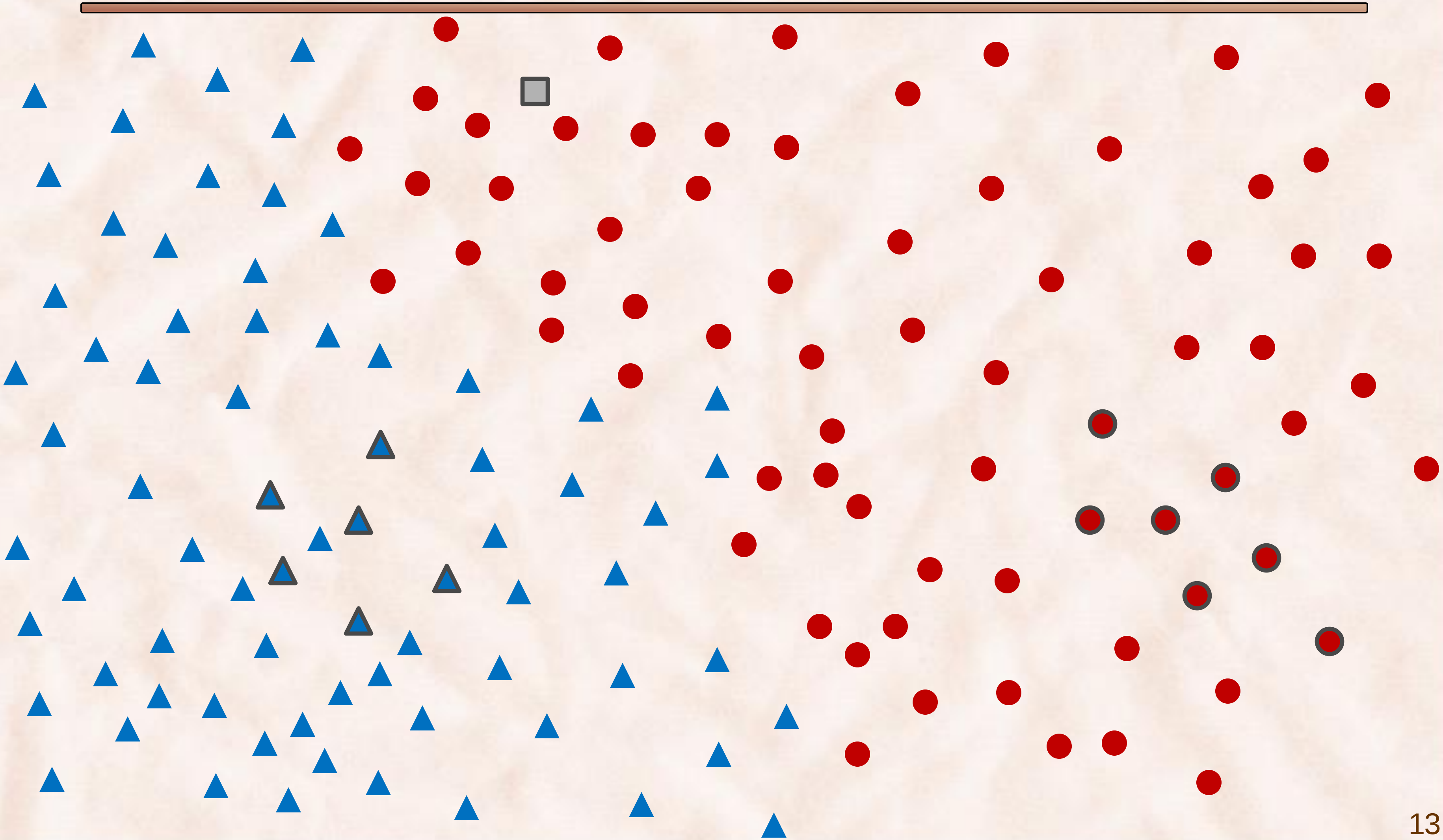


Efficient Indexing

- Linear searching for *k*-nearest neighbors is not efficient for large training sets:
  - O(N) time complexity.
- For Euclidean distance use a **kd-tree**:
  - instances stored at leaves of the tree.
  - internal nodes branch on threshold test on individual features.
  - expected time to find the nearest neighbor is O(log N)
- Indexing structures depend on distance function:
  - **inverted index** for text retrieval with cosine similarity.

k-NN and The Curse of Dimensionality

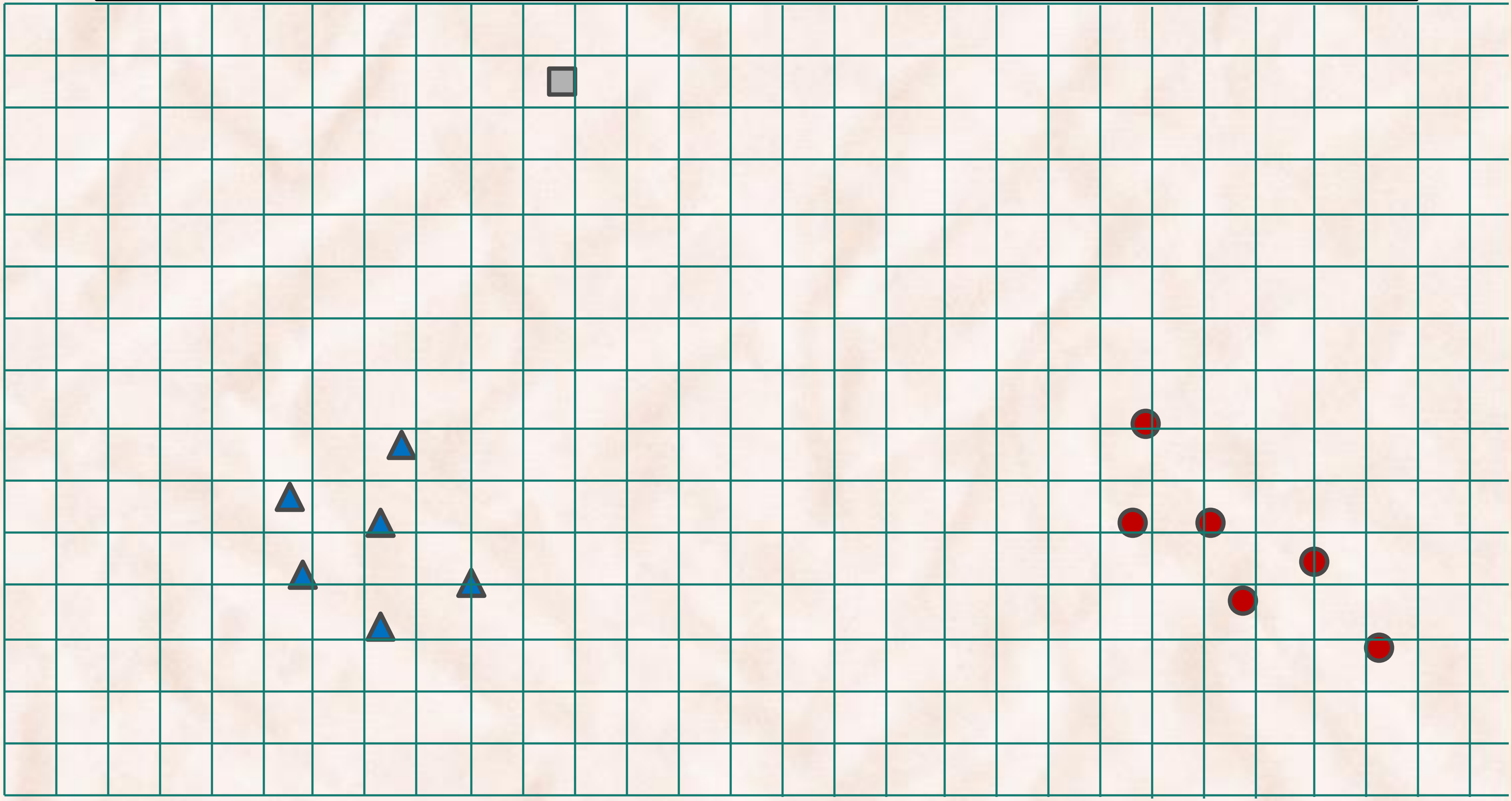
k-NN and The Curse of Dimensionality



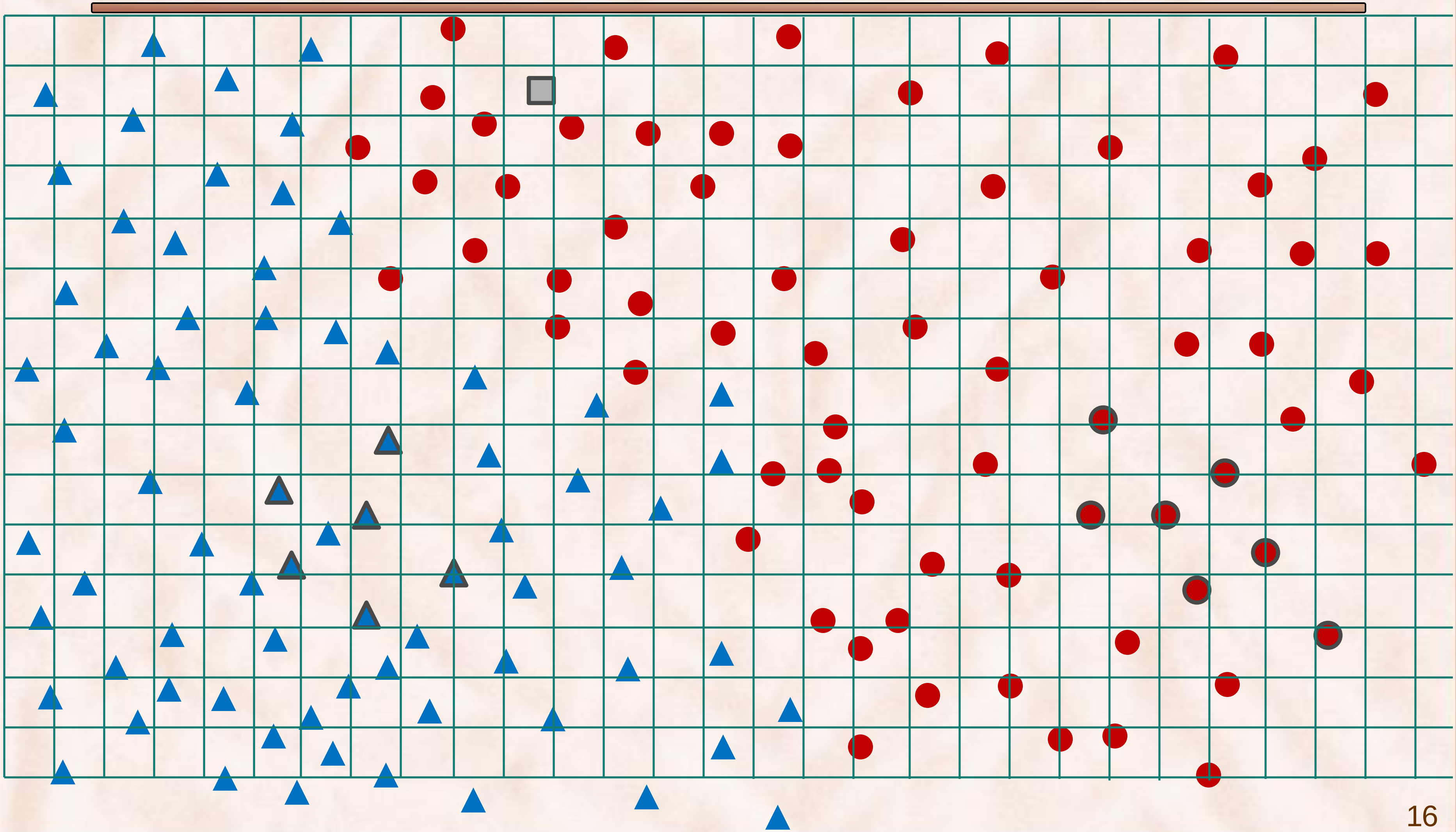
k-NN and The Curse of Dimensionality

- We would like to have the input area "covered" by training samples:
  - For an arbitrary test sample $\mathbf{x}$, there should be at least one training sample $\mathbf{x}_n$ that is close to it, i.e. $d(\mathbf{x}, \mathbf{x}_n) < \tau$.
  - One way of ensuring this is to divide the input space into a grid of regular cells, where:
    - each grid cell is small;
    - each grid cell contains at least one training sample.

k-NN and The Curse of Dimensionality

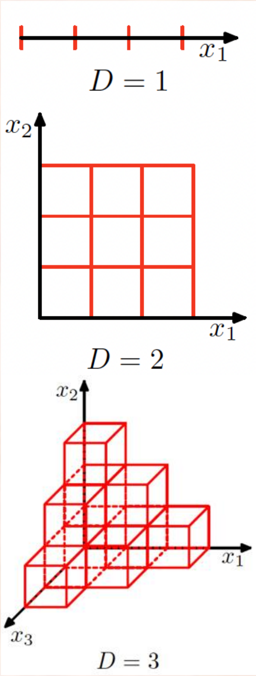


k-NN and The Curse of Dimensionality



k-NN and The Curse of Dimensionality

- How many cells of side 0.1 are needed to cover:
  - The 1D unit interval [0,1]
    - N = 10
  - The 2D unit square $[0,1]^2$
    - N = 100
  - The 3D unit cube $[0,1]^3$
    - N = 1,000
  - The K dimensional hypercube $[0,1]^K$
    - N = 10K



- We need an exponential number of examples!

k-NN and The Curse of Dimensionality

- Standard metrics weigh each feature equally:
  - Problematic when many features are irrelevant.
    - Let's look at an example ...

- One solution is to weigh each feature differently:
  - Use measure indicating ability to discriminate between classes, such as:
    - Information Gain, Chi-square Statistic
    - Pearson Correlation, Signal to Noise Ration, T test.
  - "Stretch" the axes:
    - lengthen for relevant features, shorten for irrelevant features.
  - Equivalent with Mahalanobis distance with diagonal covariance.

Distance-Weighted k-NN

For any test point **x**, weight each of the *k* neighbors according to their distance from **x**.

1. Find *k* instances $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ nearest to **x**.

2. Let $y(x) = \arg\max_{t \in T} \sum_{i=1}^{k} \boxed{w_i} \delta_t(t_i)$

   where $w_i = \|\mathbf{x} - \mathbf{x}_i\|^{-2}$ measures the similarity between **x** and $\mathbf{x}_i$

Kernel-based Distance-Weighted NN

For any test point **x**, weight all training instances according to their similarity with **x**.

1. Assume binary classification, T = {+1, –1}.
2. Compute weighted majority:

$$y(\mathbf{x}) = sign\left(\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i) t_i\right)$$

Regression with k-Nearest Neighbor

Input:

- A training dataset $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots (\mathbf{x}_n, t_n)$.
- A test instance $\mathbf{x}$.

Output:

- Estimated function value $y(\mathbf{x})$.

1. Find k instances $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ nearest to $\mathbf{x}$.
2. Let $y(x) = \frac{1}{k}\sum_{i=1}^{k} t_i$

kNN Regression in NumPy

```
[12] import numpy as np
     from numpy import linalg as la

     def knn_regression(X, y, x, k, d):
       """
         X: a 2D array, with rows storing training feature vectors.
         y: a 1D array storing the labels of the training examples.
         x: the feature vector of a test example.
         d: a distance function.
       """
       x_to_X = d(x, X)
       neighbors = np.argpartition(x_to_X, k - 1)[:k]
       label = np.mean(y[neighbors])

       return label


     def euclidean_distance(x, X):
       return la.norm(X - x, axis = 1)
```

kNN Regression in one line in NumPy

```
import numpy as np
from numpy import linalg as la

def knn_regression(X, y, x, k, d):
  return np.mean(y[np.argpartition(d(x, X), k - 1)[:k]])

def euclidean_distance(x, X):
  return la.norm(X - x, axis = 1)
```

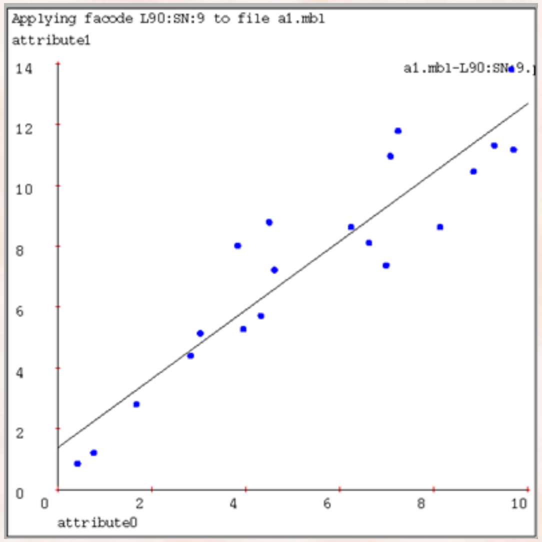
Testing on a dataset with 5 training examples:

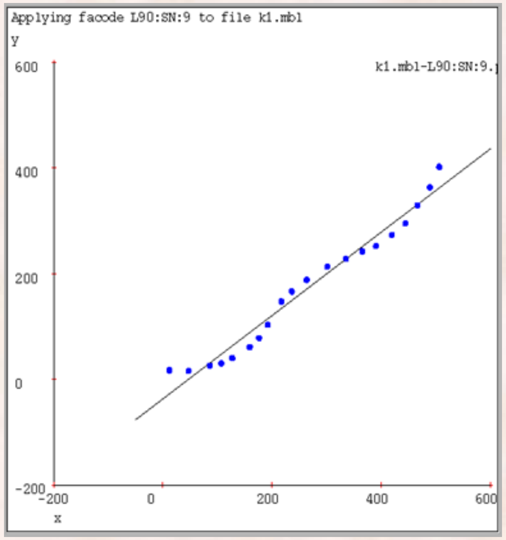
```
X = np.array([[-1, 1],
              [-2, 2],
              [0, 2],
              [2, 3],
              [4, 5]])
y = np.array([1, 2, 3, 4, 5])
x = np.array([0, 0])

knn_regression(X, y, x, 3, euclidean_distance)
```

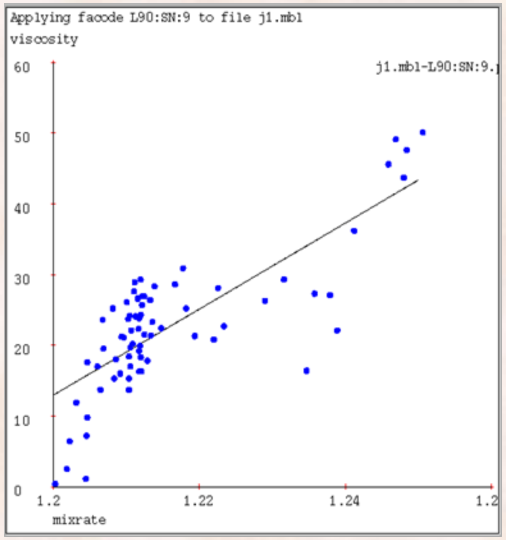
```
2.0
```

3 Datasets & Linear Interpolation

[http://www.autonlab.org/tutorials/mbl08.pdf]

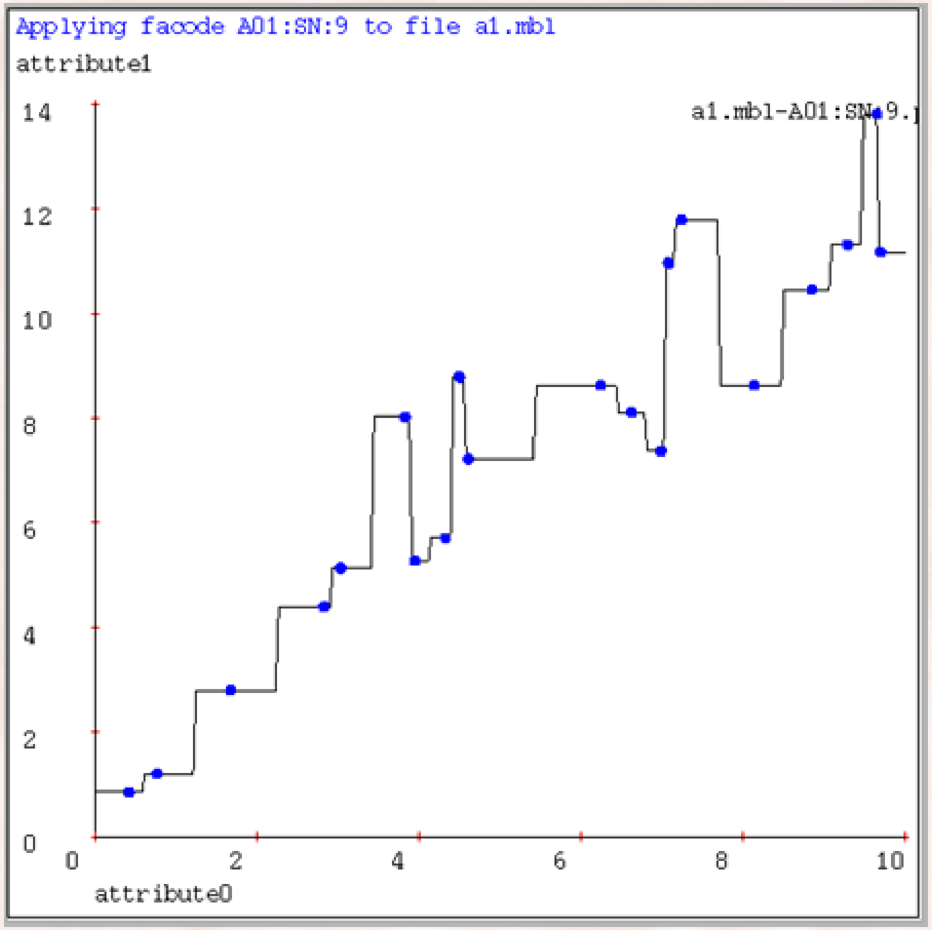




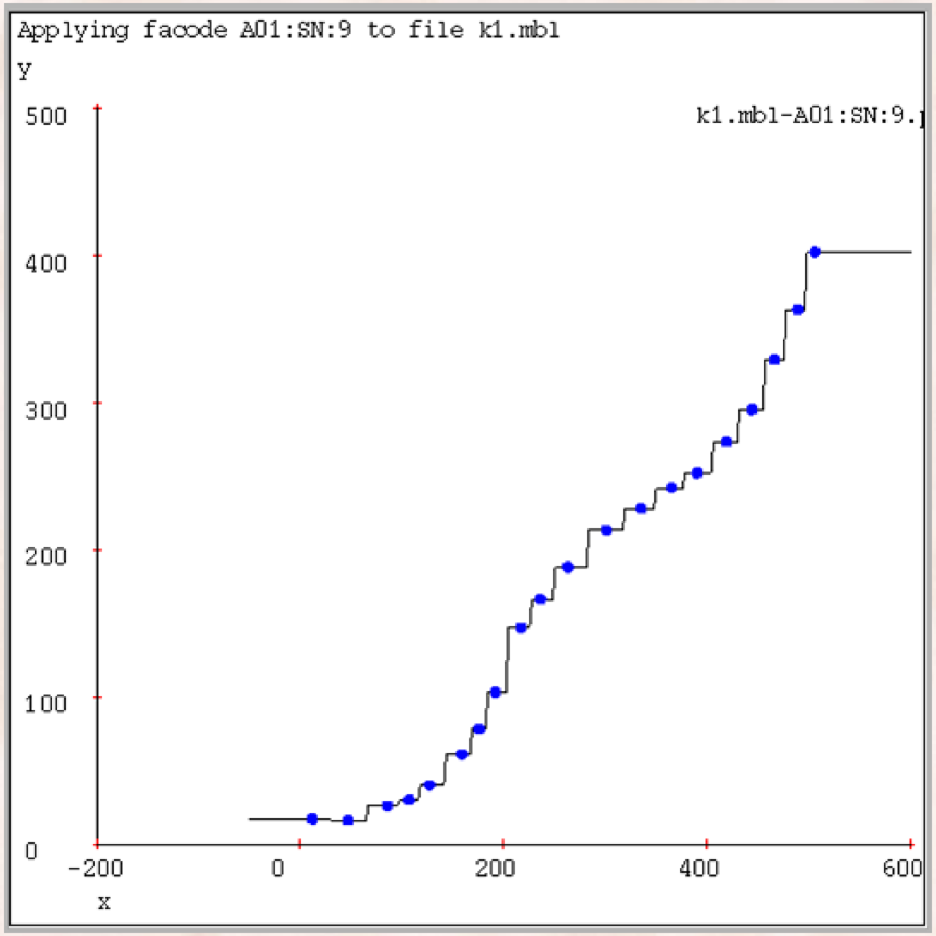


Linear interpolation does not always lead to good models of the data.

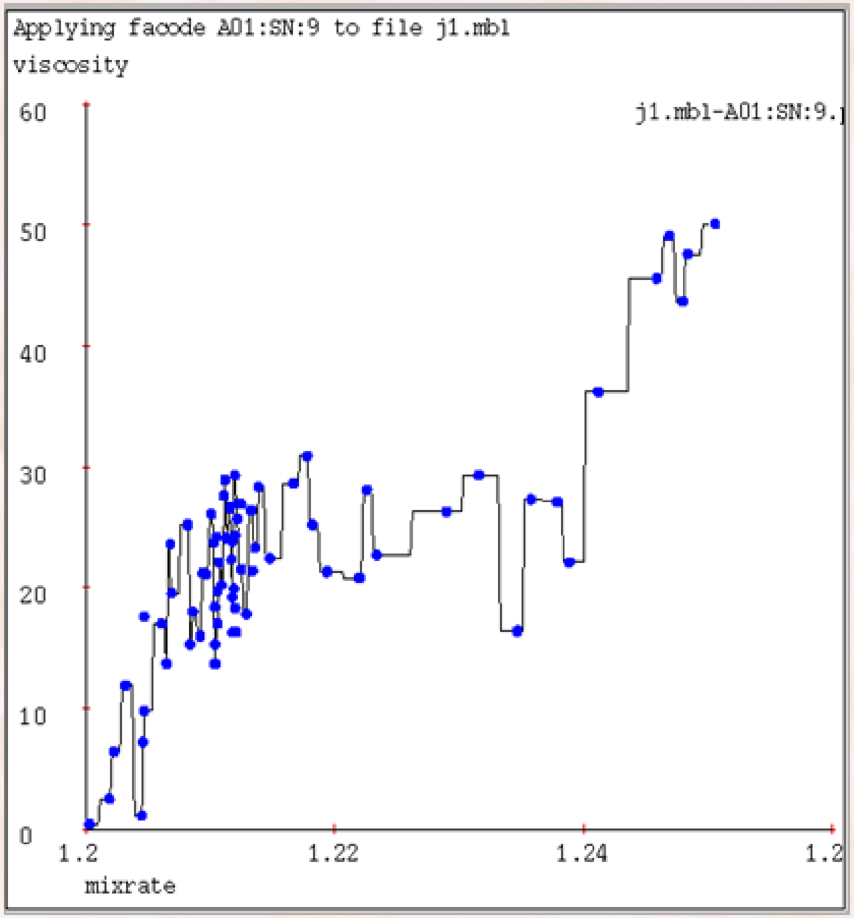
Regression with 1-Nearest Neighbor



Regression with 1-Nearest Neighbor



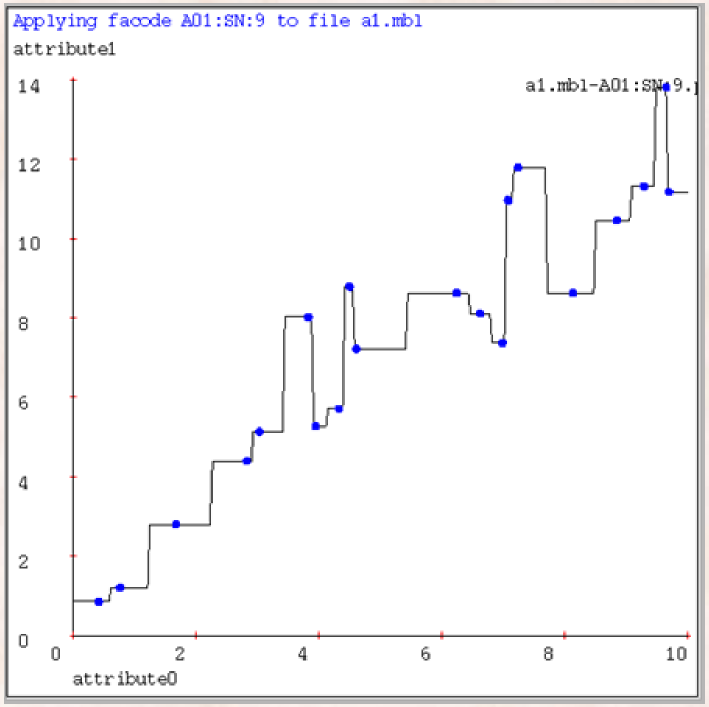
Regression with 1-Nearest Neighbor



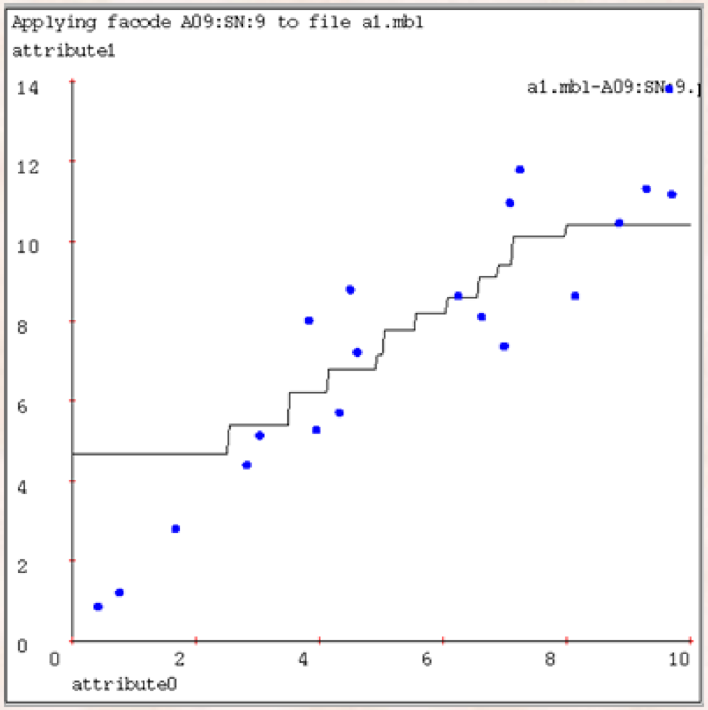
⇒ 1-NN has high variance

Regression with 9-Nearest Neighbor

$k = 1$

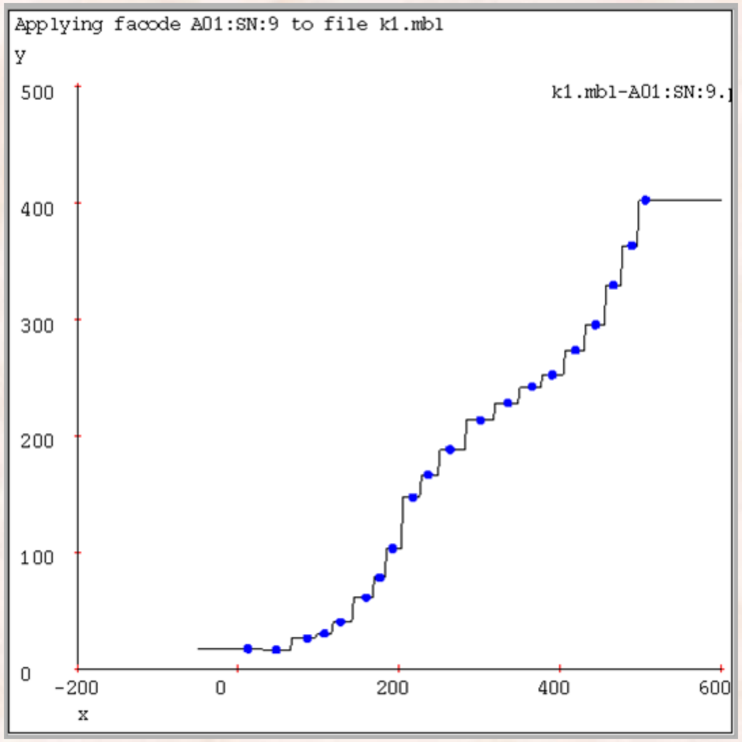


$k = 9$

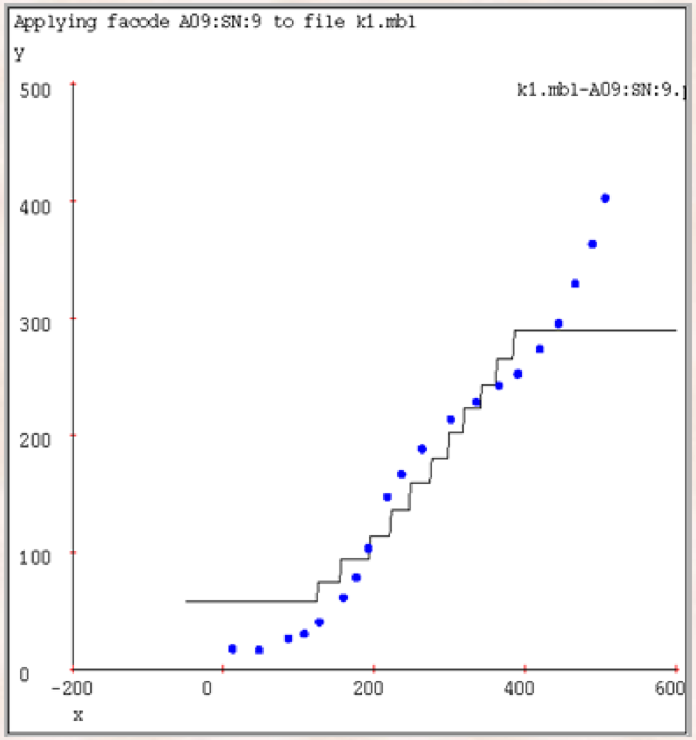


Regression with 9-Nearest Neighbor

$k = 1$

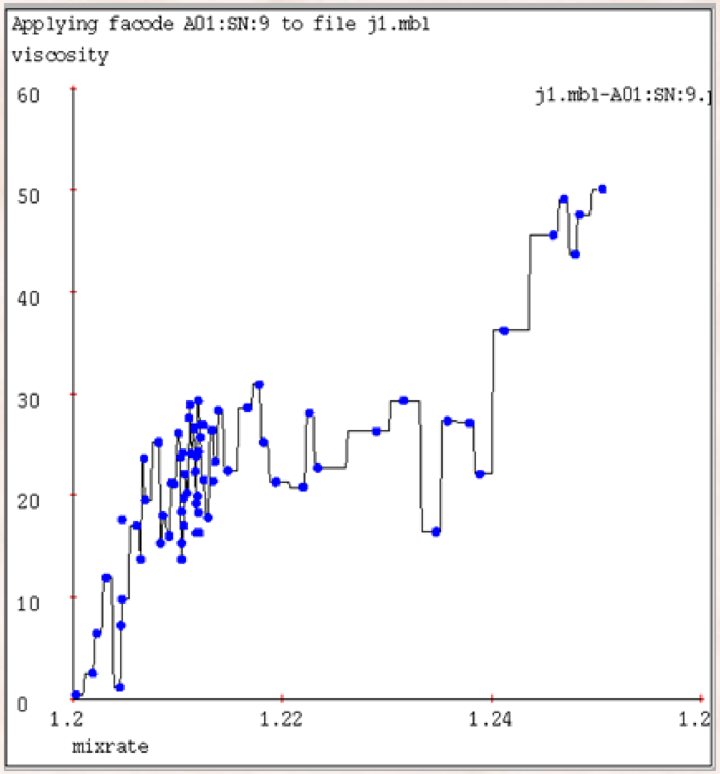


$k = 9$

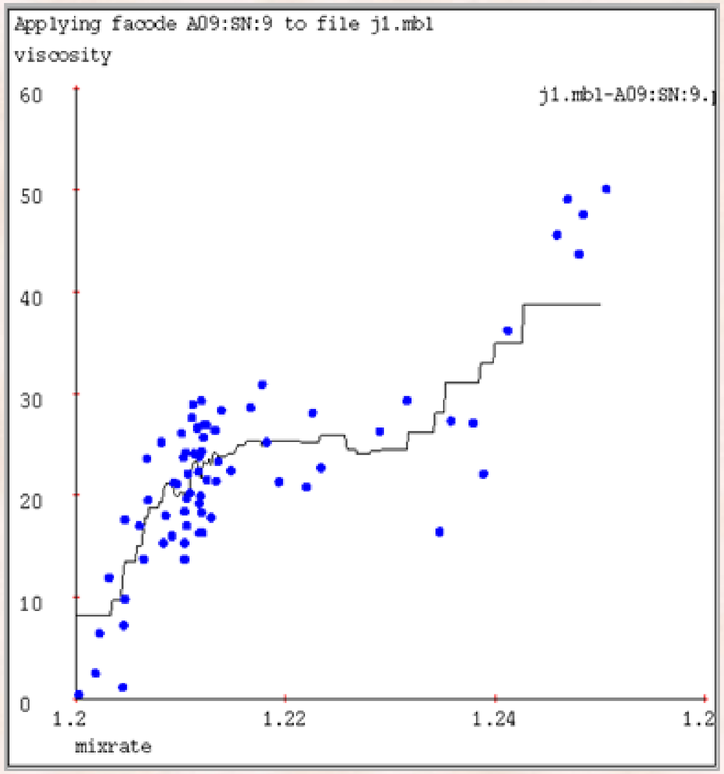


Regression with 9-Nearest Neighbor

$k = 1$



$k = 9$



Distance-Weighted k-NN for Regression

For any test point $\mathbf{x}$, weight each of the k neighbors according to their similarity with $\mathbf{x}$.

1. Find k instances $\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_k$ nearest to $\mathbf{x}$.
2. Let $y(x) = \sum_{i=1}^{k} w_i t_i \Big/ \sum_{i=1}^{k} w_i$

$$\text{where } w_i = \left\|\mathbf{x} - \mathbf{x}_i\right\|^{-2}$$

For k = N $\Rightarrow$ Shepard's method [Shepard, ACM '68].

Kernel-based Distance Weighted NN Regression

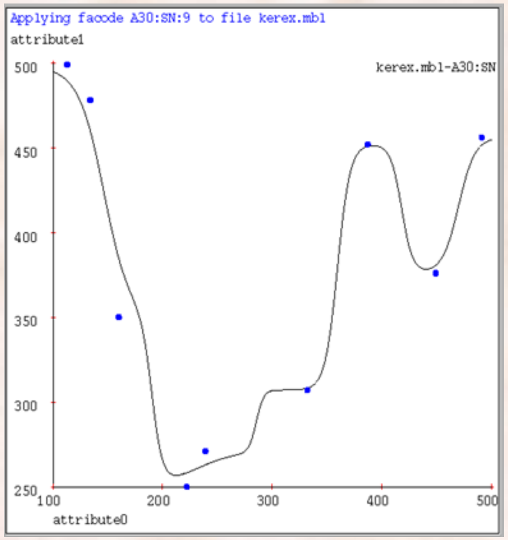
For any test point **x**, weight all training instances according to their similarity with **x**.

1. Return weighted average:

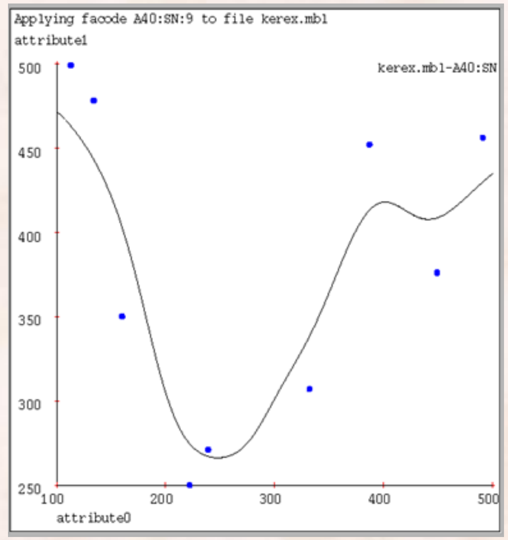
$$y(\mathbf{x}) = \frac{\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i) t_i}{\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i)}$$

NN Regression with Gaussian Kernel

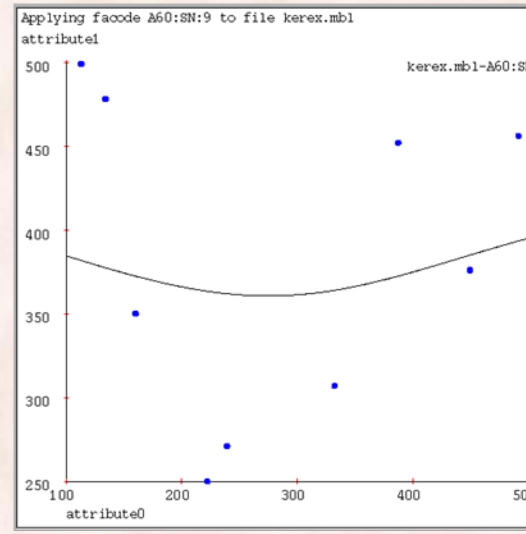
$2\sigma^2=10$



$2\sigma^2=20$



$2\sigma^2=80$

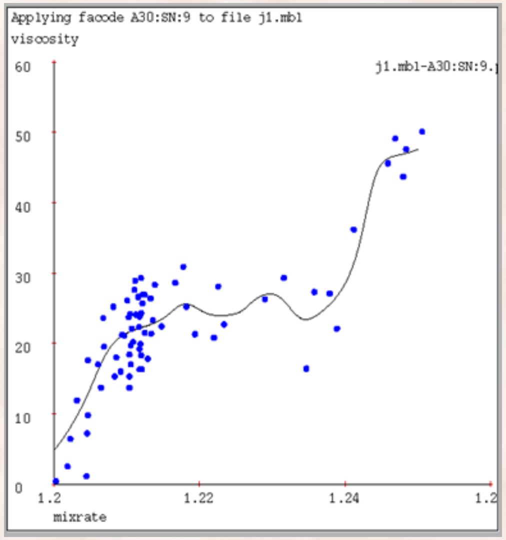


$$K(\mathbf{x},\mathbf{x}_i) = e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}}$$

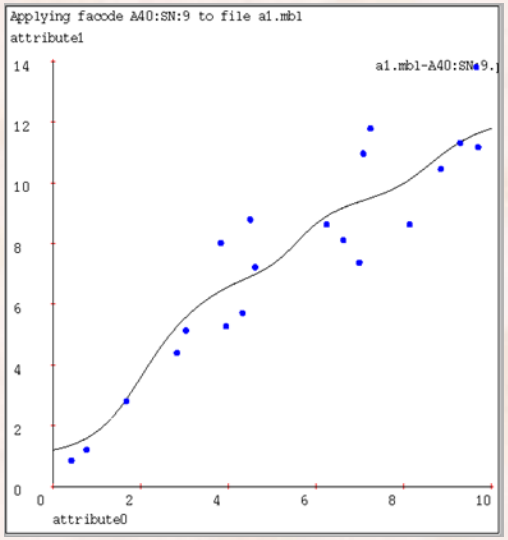
Increased kernel width means more influence from distant points.

NN Regression with Gaussian Kernel

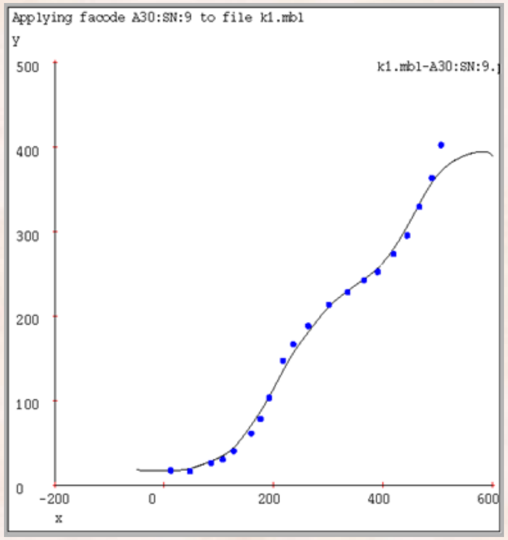
$2\sigma^2$=1/16 of x axis



$2\sigma^2$=1/32 of x axis



$2\sigma^2$=1/32 of x axis



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

k-Nearest Neighbor Summary

- Training: memorize the training examples.
- Testing: compute distance/similarity with training examples.
- Trades decreased training time for increased test time.
- Use kernel trick to work in implicit high dimensional space.
- Needs feature selection when many irrelevant features.
- An Instance-Based Learning (IBL) algorithm:
  - Memory-based learning
  - Lazy learning
  - Exemplar-based
  - Case-based