

Machine Learning: k-Nearest Neighbors

Lecture 08

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Nonparametric Methods: k-Nearest Neighbors

Input:

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

Output:

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

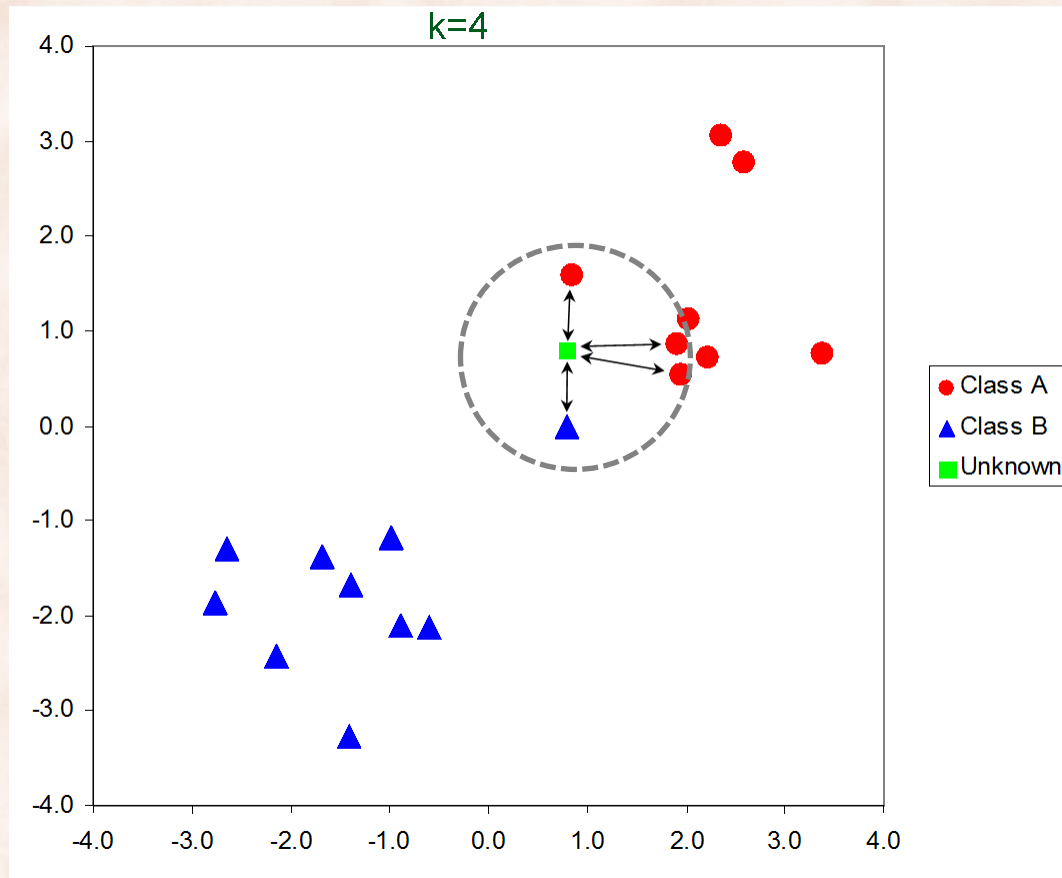
1. Find k instances $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$ *nearest* to \mathbf{x} .

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.

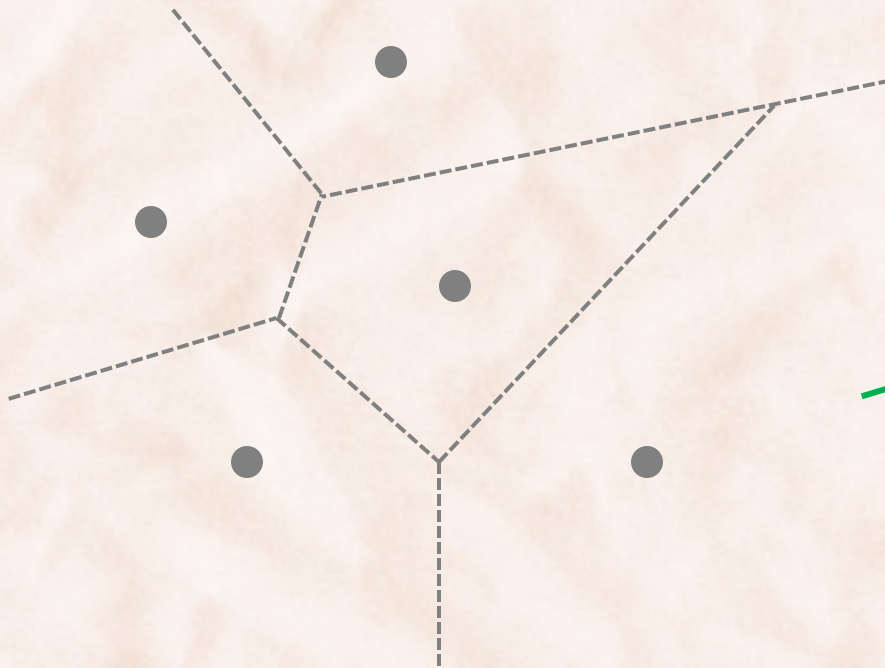
k-Nearest Neighbors (k-NN)

- Euclidean distance, $k = 4$

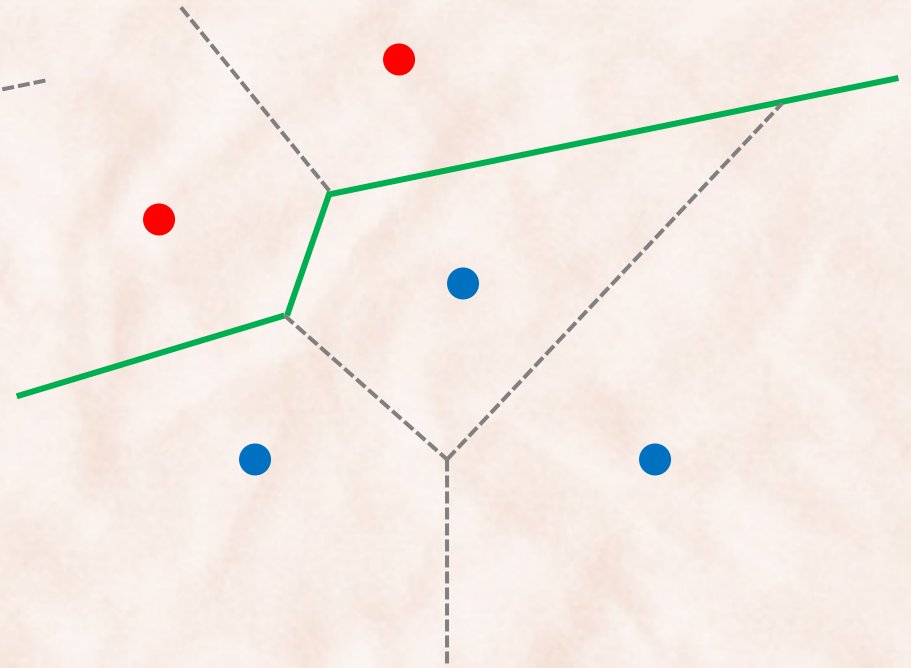


k-Nearest Neighbors (k-NN)

- Euclidian distance, $k = 1$.



Voronoi diagram



decision boundary

k-NN for Classification: Probabilistic Justification

- Assume a dataset with N_j points in class C_j .

$$\Rightarrow \text{total number of points is } N = \sum_j N_j$$

- Draw a sphere centered at \mathbf{x} containing K points:

- sphere has volume V .
- sphere contains K_j points from class C_j .

- If V sufficiently small and K sufficiently large, we can estimate [2.5.1]:

$$p(\mathbf{x} | C_j) = \frac{K_j}{N_j V} \quad p(\mathbf{x}) = \frac{K}{NV} \quad p(C_j) = \frac{N_j}{N}$$

- Bayes' theorem $\Rightarrow p(C_j | \mathbf{x}) = \frac{K_j}{K} \Rightarrow$ choose class C_j with most neighbors.

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 \mathbf{x} and \mathbf{y} .

- **Mahalanobis distance:**

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

(sample) covariance matrix

- scale-invariant metric that normalizes for variance.
- if $S = I \Rightarrow$ Euclidean distance.
- if $S = \text{diag}(\sigma_1^{-2}, \sigma_2^{-2}, \dots, \sigma_K^{-2}) \Rightarrow$ *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).

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

- used in bioinformatics.

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

- Standard metrics weigh each feature equally:
 - Problematic when many features are irrelevant.
- 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 Ratio, 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 \mathbf{x} , weight each of the k neighbors according to their distance from \mathbf{x} .

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

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

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

Kernel-based Distance-Weighted NN

For any test point \mathbf{x} , weight all training instances according to their similarity with \mathbf{x} .

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

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

Regression with k-Nearest Neighbor

Input:

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

Output:

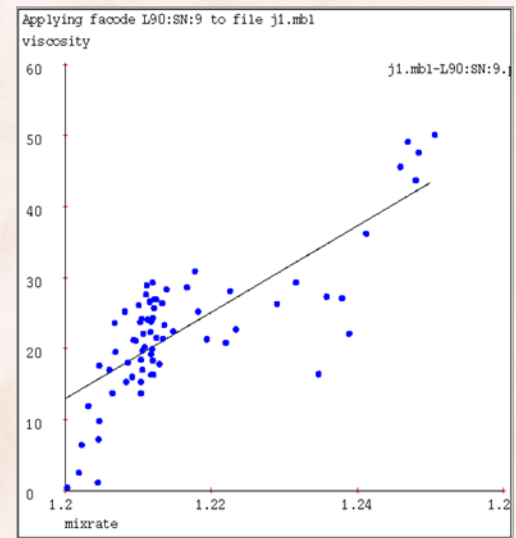
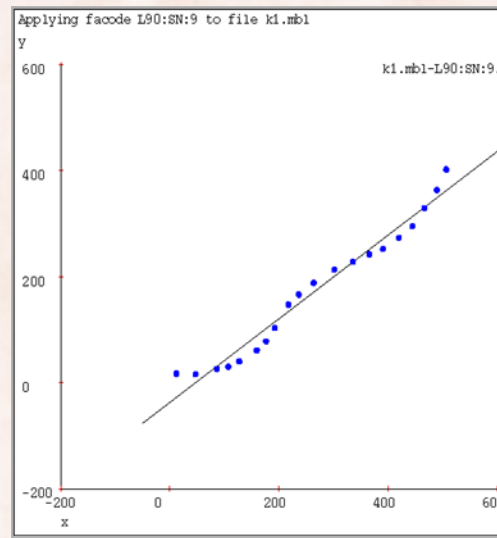
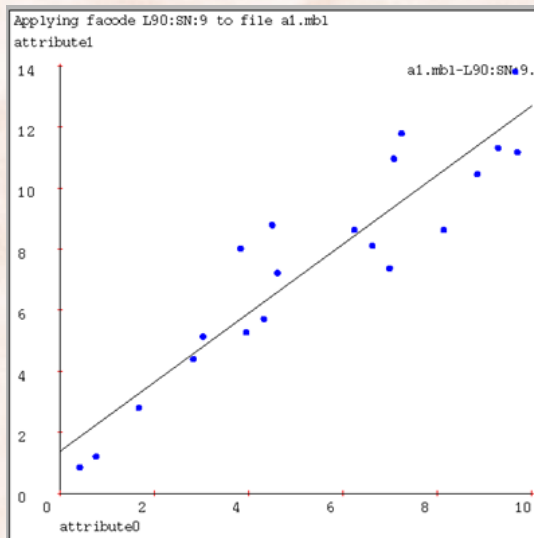
- Estimated function value $y(\mathbf{x})$.
-

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

2. Let $y(x) = \frac{1}{k} \sum_{i=1}^k t_i$

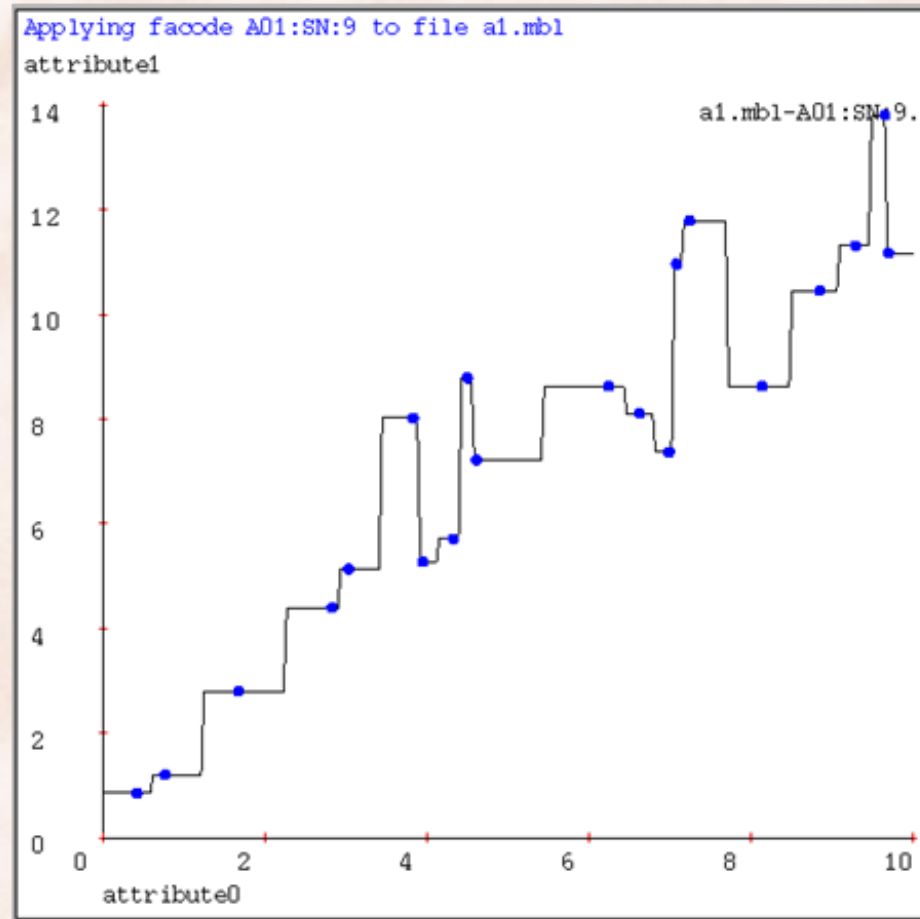
3 Datasets & Linear Interpolation

[<http://www.autonlab.org/tutorials/mb108.pdf>]

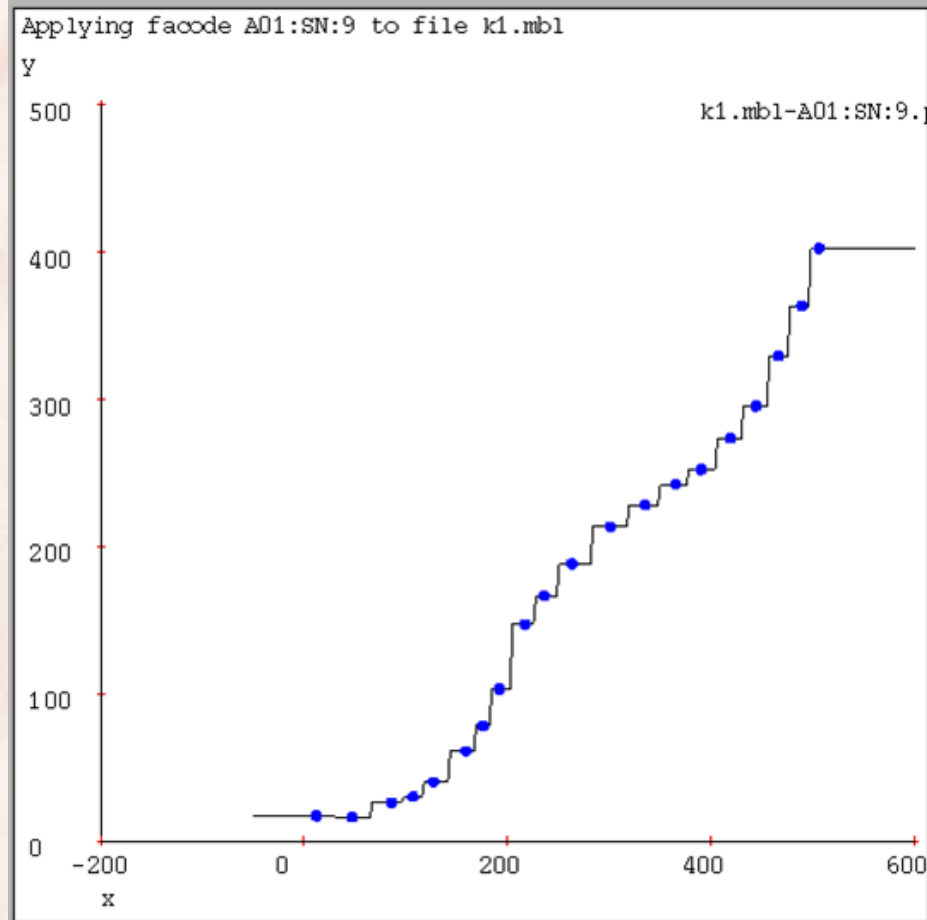


Linear interpolation does not always lead to good models of the data.

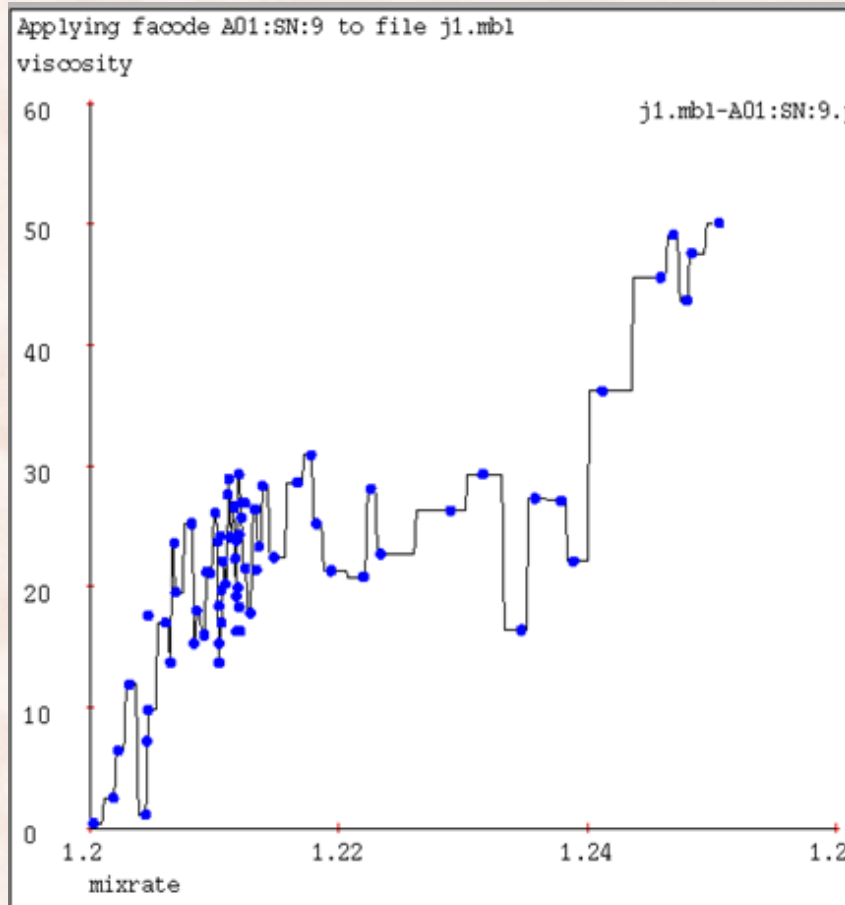
Regression with 1-Nearest Neighbor



Regression with 1-Nearest Neighbor



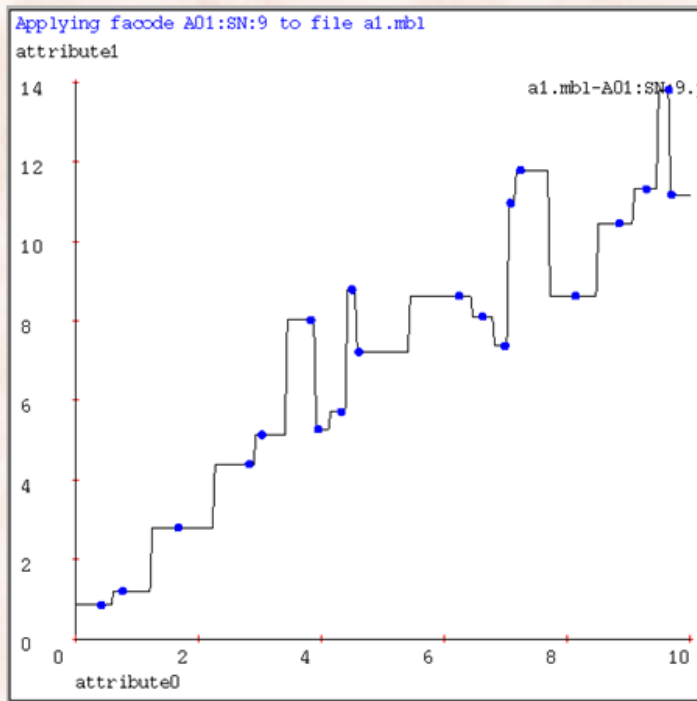
Regression with 1-Nearest Neighbor



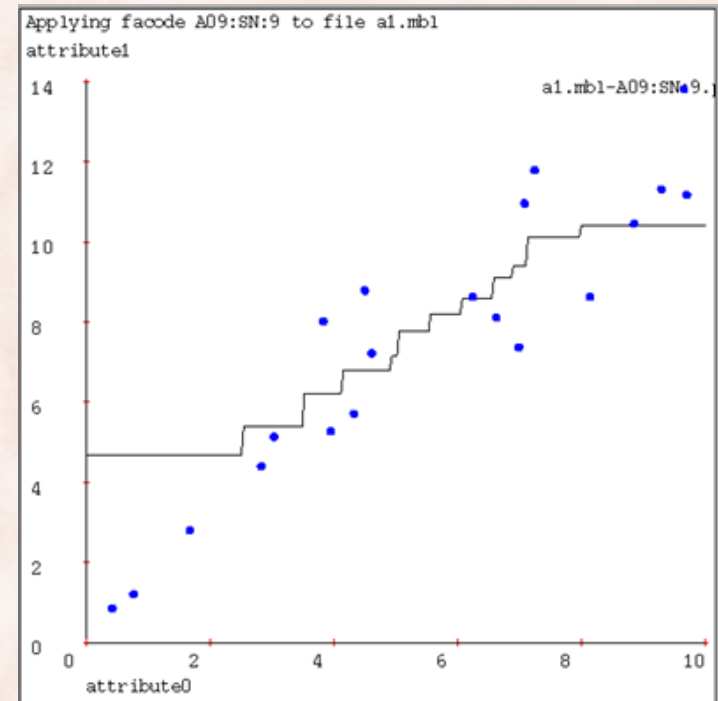
\Rightarrow 1-NN has high variance

Regression with 9-Nearest Neighbor

$k = 1$

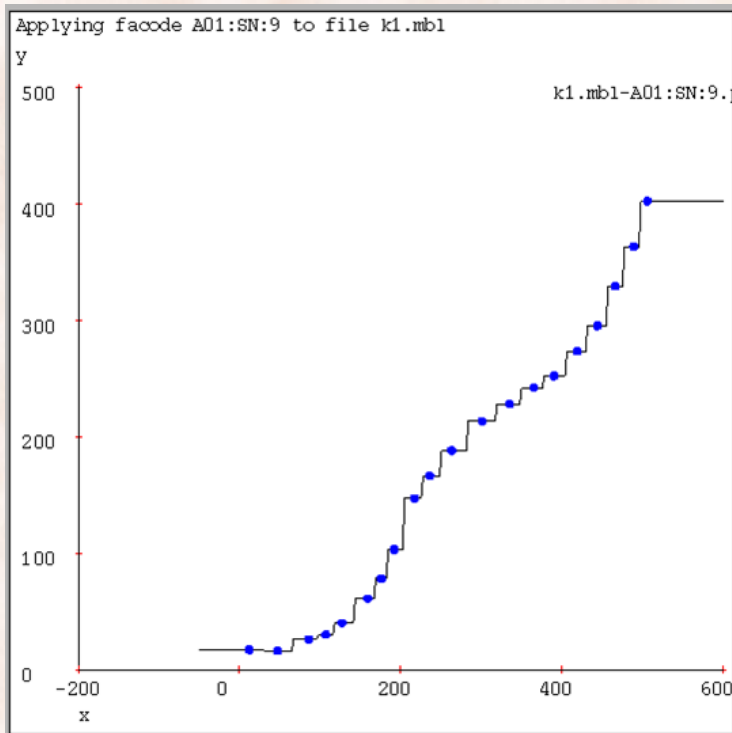


$k = 9$

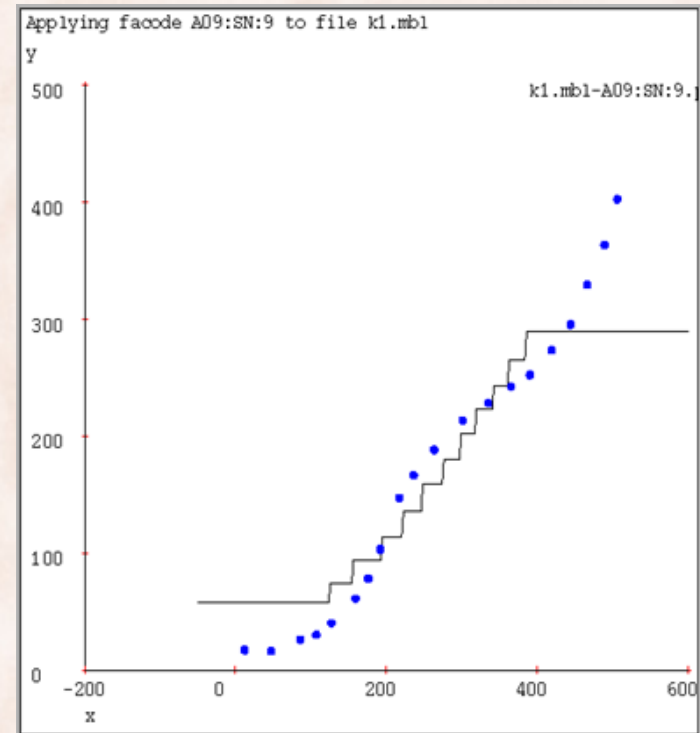


Regression with 9-Nearest Neighbor

$k = 1$

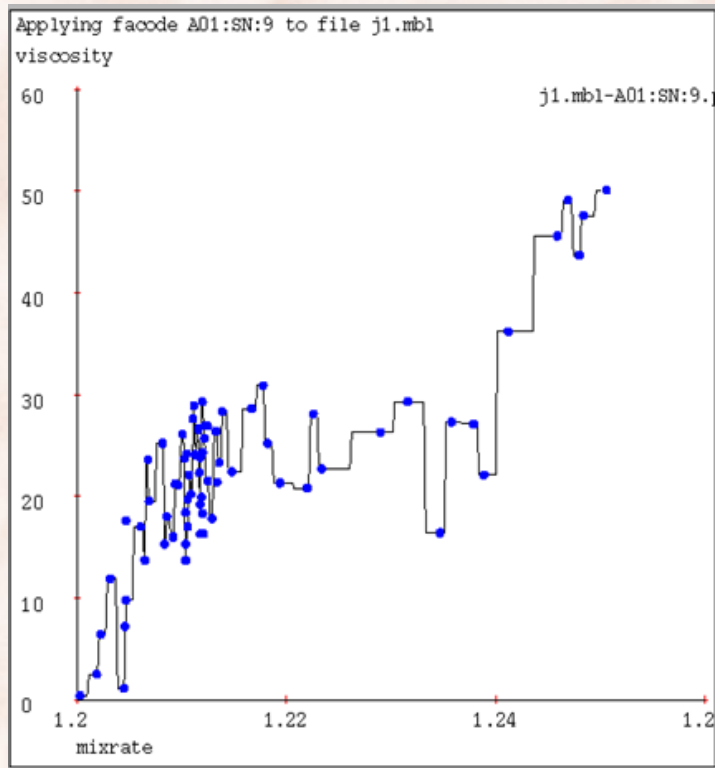


$k = 9$

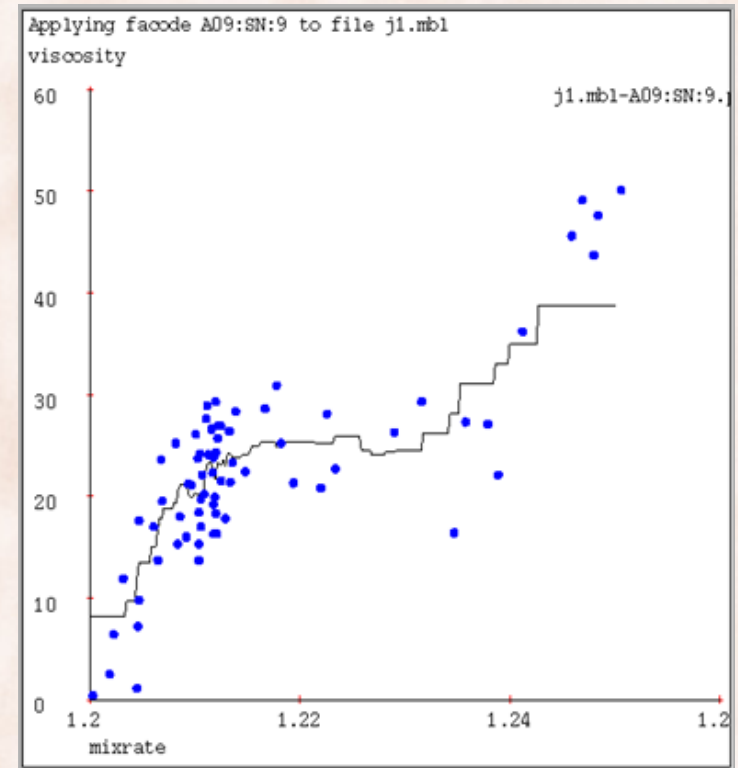


Regression with 9-Nearest Neighbor

$k = 1$



$k = 9$



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 $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$ nearest to \mathbf{x} .

2. Let
$$y(x) = \frac{\sum_{i=1}^k w_i t_i}{\sum_{i=1}^k w_i}$$

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

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

Kernel-based Distance Weighted NN Regression

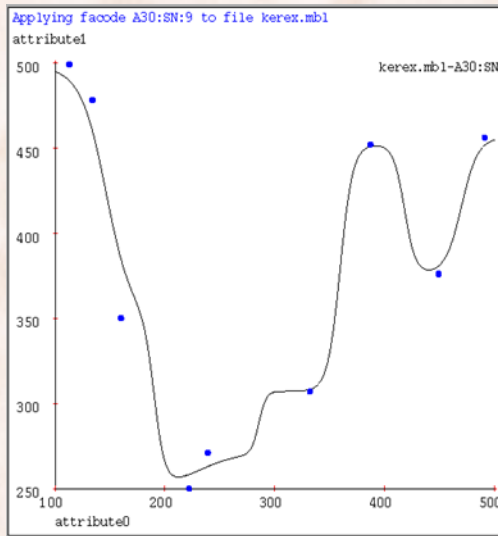
For any test point \mathbf{x} , weight all training instances according to their similarity with \mathbf{x} .

1. Return weighted average:

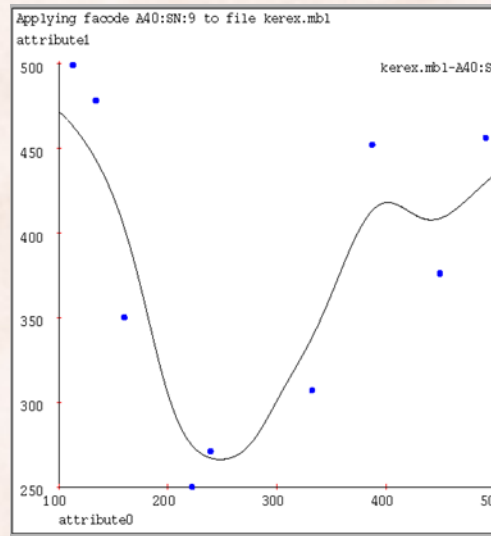
$$y(\mathbf{x}) = \frac{\sum_{i=1}^N K(\mathbf{x}, \mathbf{x}_i) t_i}{\sum_{i=1}^N K(\mathbf{x}, \mathbf{x}_i)}$$

NN Regression with Gaussian Kernel

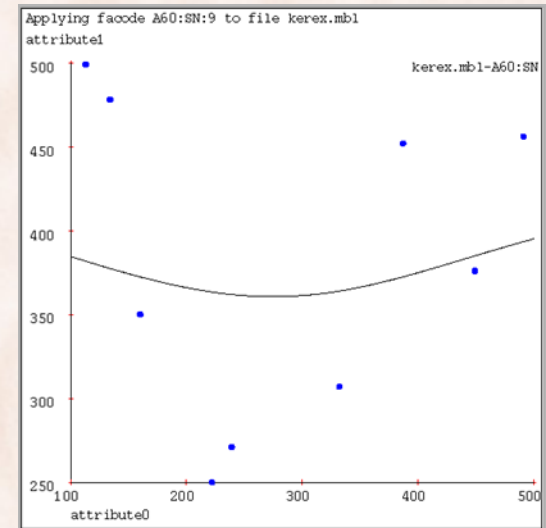
$2\sigma^2=10$



$2\sigma^2=20$



$2\sigma^2=80$

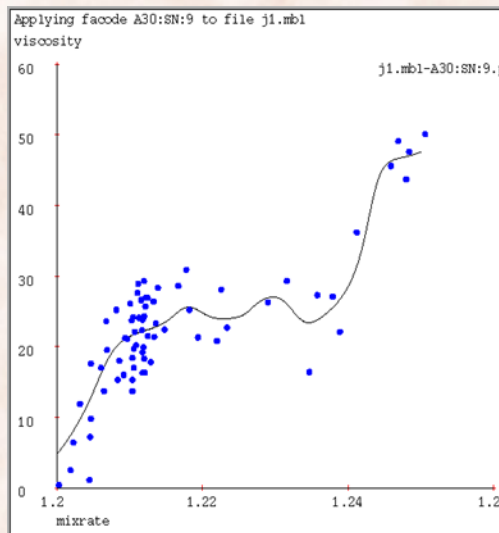


$$K(\mathbf{x}, \mathbf{x}_i) = e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}}$$

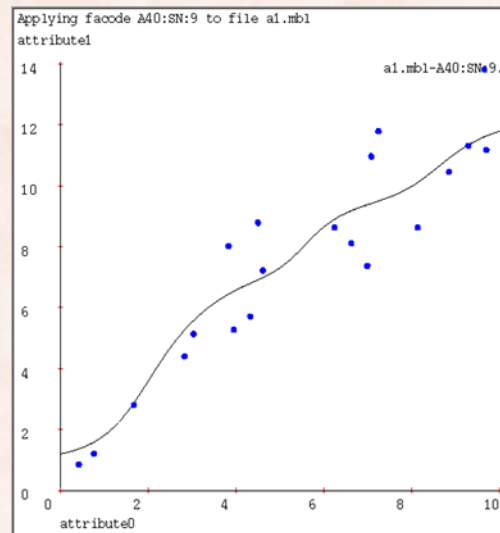
Increased kernel width means more influence from distant points.

NN Regression with Gaussian Kernel

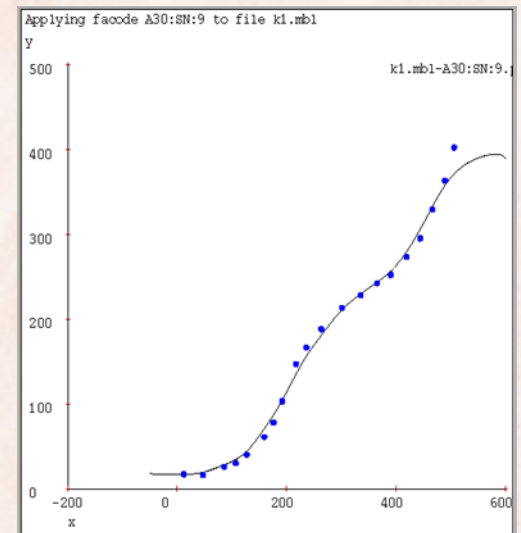
$2\sigma^2=1/16$ of x axis



$2\sigma^2=1/32$ 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