

---

# Machine Learning

## Instance-based learning

(with slides/ideas from Bryan Pardo, Pedro Domingos, and Andrew Moore)

# Nearest Neighbor Classifier

---

- Example of instance-based (a.k.a case-based) learning
- The basic idea:
  1. Get some example set of cases with known outputs  
e.g diagnoses of infectious diseases by experts
  2. When you see a new case, assign its output to be the same as the most similar known case.  
Your symptoms most resemble Mr X.  
Mr X had the flu.  
Ergo you have the flu.

# General Learning Task

---

There is a set of possible examples  $X = \{x_i\}$

Each example is an n-tuple of attribute values

$$\vec{x}_1 = \langle a_1, \dots, a_k \rangle$$

There is a target function that maps  $X$  onto some set  $Y$

$$f : X \rightarrow Y$$

The DATA is a set of duples <example, target function values>

$$D = \{ \langle \vec{x}_1, f(\vec{x}_1) \rangle, \dots, \langle \vec{x}_m, f(\vec{x}_m) \rangle \}$$

Find a **hypothesis**  $h$  such that...

$$\forall \vec{x}, h(\vec{x}) \approx f(\vec{x})$$

# Nearest neighbor!

---

Task: Given some set of training data...

$$D = \{ \langle \vec{x}_1, f(\vec{x}_1) \rangle, \dots, \langle \vec{x}_m, f(\vec{x}_m) \rangle \}$$

...and query point  $\vec{x}_q$ , predict  $f(\vec{x}_q)$

1. Find the nearest member of data set to the query

$$\vec{x}_{nn} = \arg \min_{x \in D} (d(\vec{x}, \vec{x}_q))$$

distance  
function



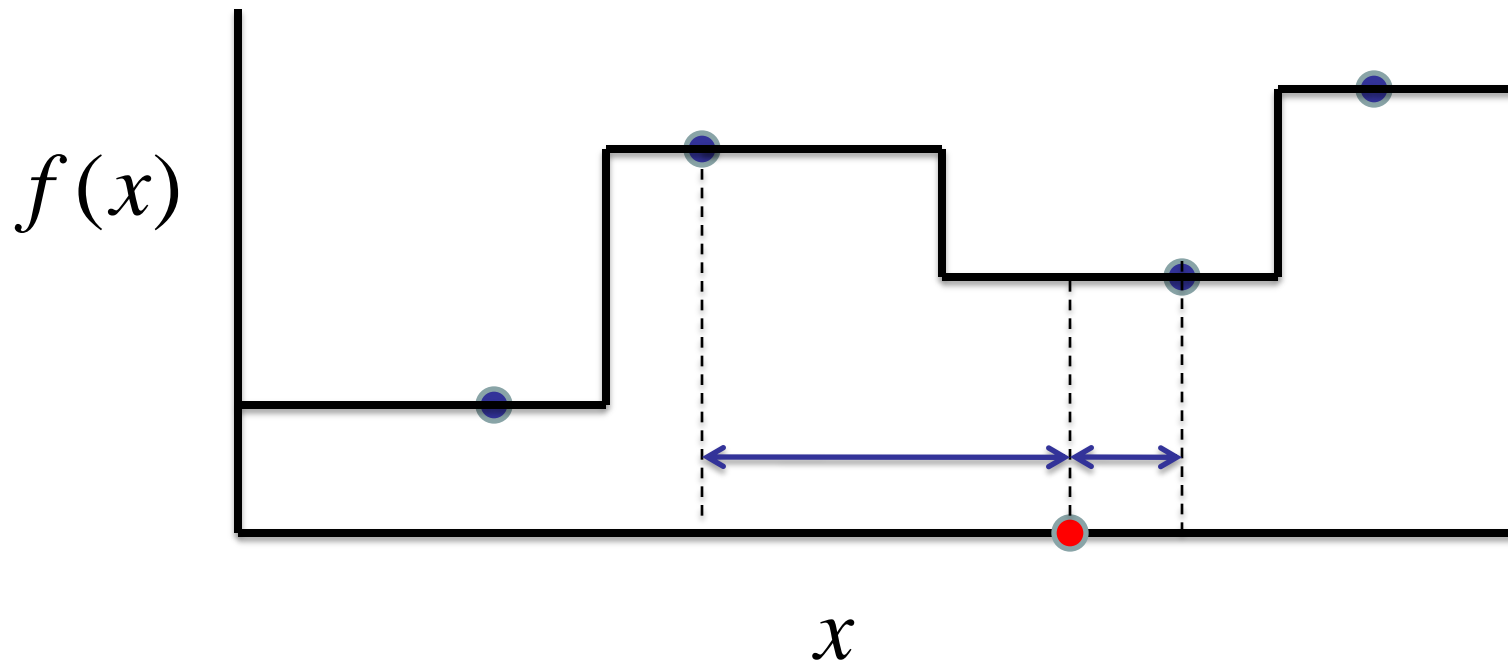
2. Assign the nearest neighbor's output to the query

$$h(\vec{x}_q) = f(\vec{x}_{nn})$$

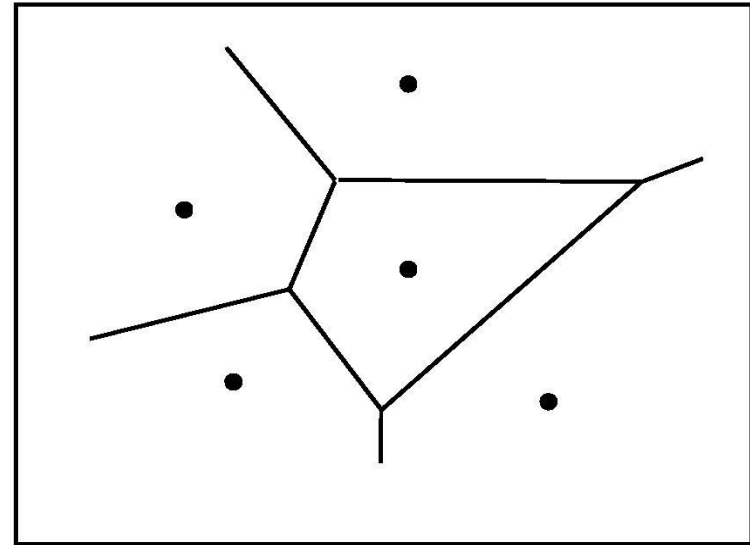
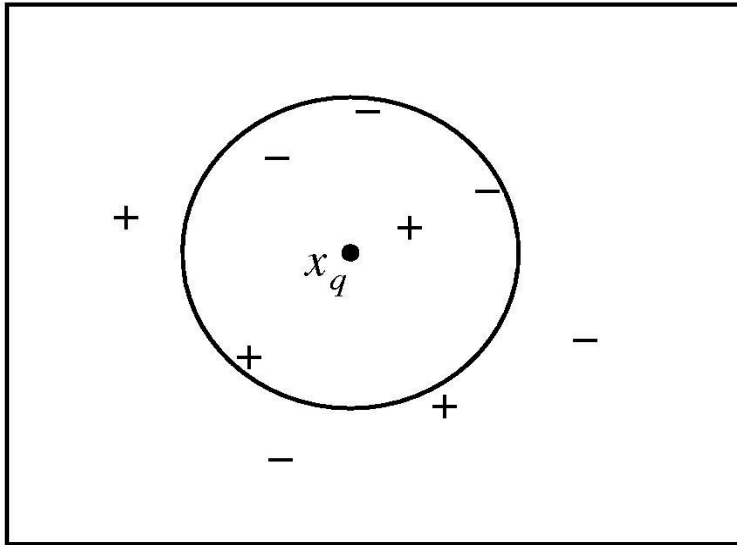
Our hypothesis

# A Single-attribute Example

- Find closest point.  $\vec{x}_{nn} = \arg \min_{x \in D} (d(\vec{x}, \vec{x}_q))$
- Give query its value  $f(\vec{x}_q) = f(\vec{x}_{nn})$



# Voronoi Diagram



$S$ : Training set

**Voronoi cell** of  $\mathbf{x} \in S$ :

All points closer to  $\mathbf{x}$  than to any other instance in  $S$

**Region of class  $C$ :**

Union of Voronoi cells of instances of  $C$  in  $S$

# What makes an instance-based learner?

---

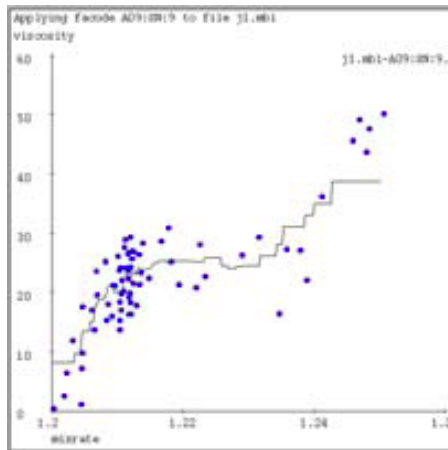
- A distance measure  
*Nearest neighbor: typically Euclidean*
- Number of neighbors to consider  
*Nearest neighbor: One*
- A weighting function (optional)  
*Nearest neighbor: unused (equal weights)*
- How to fit with the neighbors  
*Nearest neighbor: Same output as nearest neighbor*

# K-nearest neighbor

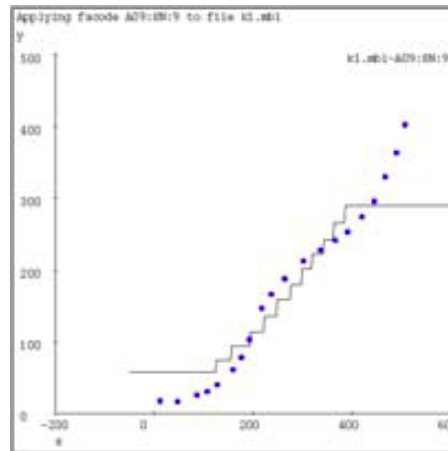
---

- A distance measure  
*Typically Euclidean*
- Number of neighbors to consider  
*K*
- A weighting function (optional)  
*Unused (i.e. equal weights)*
- How to fit with the neighbors  
*Vote using K nearest neighbors (or take average, for regression)*

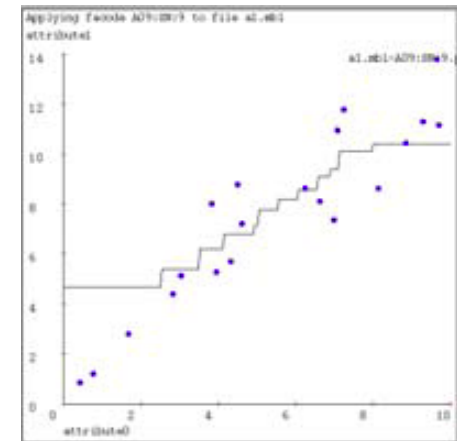
# Examples of KNN where K=9



Reasonable job  
Did smooth noise



Screws up on the ends



OK, but problem on  
the ends again.

# Pros and Cons

---

- Advantages
  - Fast training (a “lazy” method)
  - Learn complex functions easily
  - Don’t lose information
- Disadvantages
  - Slow at query time
  - Lots of storage
  - **Easily fooled by irrelevant attributes**

# Irrelevant Attributes

---

- The Curse of Dimensionality
  - Nearest Neighbor easily misled when  $X$  high-dim
  - Low-dimensional intuitions don't extend to high dim
- Example:
  - Uniform distribution on hypercube
  - Sphere approximation of cube
    - Exercise: prove that the maximal intersection of hypersphere of volume 1 and hypercube of volume 1 goes to zero as dim increases
      - (if it's true)

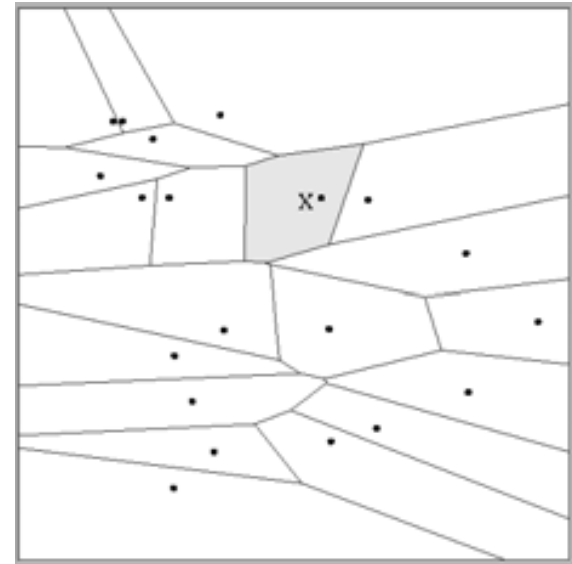
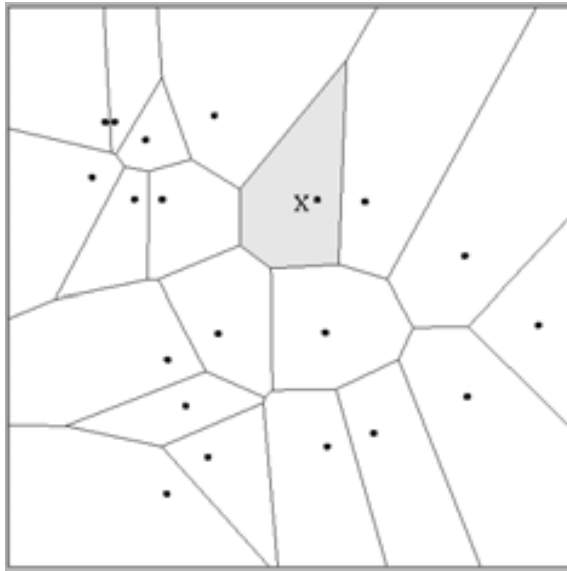
# Feature Selection

---

- Pre-selection
  - Identify a good set of  $R$  features
  - By e.g. information gain (as in decision trees)
- Wrapping
  - Starting with zero features, iterate:
    - greedily add a new feature based on NN performance

# Weighting dimensions

- Suppose data points are two-dimensional
- Different dimensional weightings affect region shapes



$$d(x, y) = (x_1 - y_1)^2 + (x_2 - y_2)^2 \quad d(x, y) = (x_1 - y_1)^2 + (3x_2 - 3y_2)^2$$

# Computational Cost?

---

- Optimized distance computations
  - Use cheap approximation to weed out most instances
  - Compute expensive measure on remainder
- Edited k-NN
  - For each  $x$ 
    - If  $x$  correctly classified by  $D - \{x\}$ , remove  $x$  from  $D$
- Form prototypes
  - *Merge* instances where no accuracy impact

# Avoiding overfitting

---

- Choose  $k$  in  $k$ -nearest neighbor by
  - Cross validation
- Form prototypes
- Remove noisy instances

# Kernel Regression

---

- A distance measure: *Scaled Euclidean*
- Number of neighbors to consider: *All of them*
- A weighting function (optional)

$$w_i = \exp\left(\frac{-d(x_i, x_q)^2}{K_w^2}\right)$$

*Nearby points to the query are weighted strongly, far points weakly. The  $K_w$  parameter is the Kernel Width.*

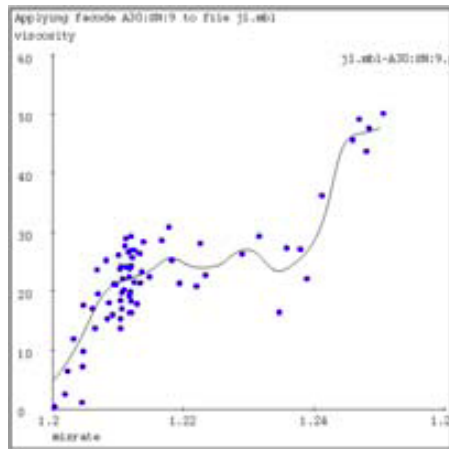
- How to fit with the neighbors

$$h(x_q) = \frac{\sum_i w_i \cdot f(x_i)}{\sum_i w_i}$$

*A weighted average*

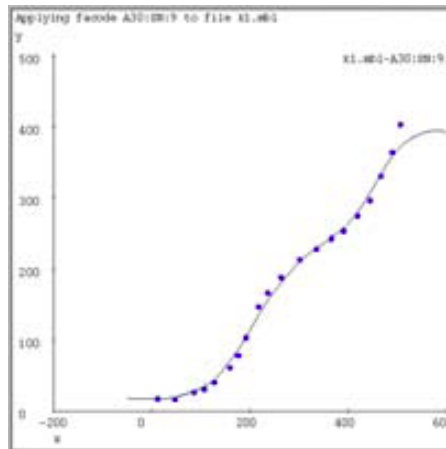
# Kernel-weighted Regression

Kernel Weight =  $1/32$   
of X-axis width



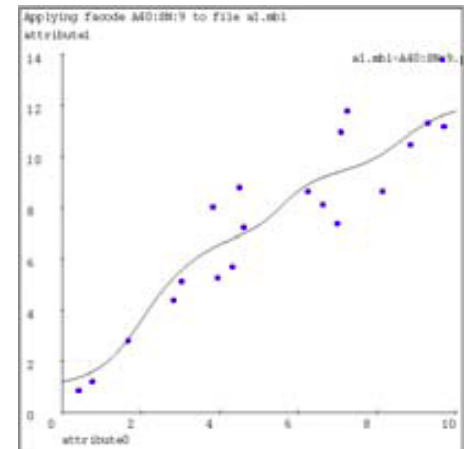
A better fit than KNN?

Kernel Weight =  $1/32$   
of X-axis width



Definitely better than  
KNN! Catch: Had to play  
with kernel width to get  
This result

Kernel Weight =  $1/16$   
of X-axis width



Nice and smooth, but  
are the bumps  
justified, or is this  
overfitting?