Instance Based Learning

The University of Texas at Dallas

Readings: Mitchell, Chapter 8

Instance Based Learning

- *k*-Nearest Neighbor
- Locally weighted Linear regression
- Collaborative Filtering

Some Vocabulary

- **Parametric vs. Non-parametric:**
  - **parametric**:
    - A particular functional form is assumed, e.g., multivariate normal, naïve Bayes.
    - Advantage of simplicity – easy to estimate and interpret
    - may have high bias because the real data may not obey the assumed functional form.
  - **non-parametric:**
    - distribution or density estimate is data-driven and relatively few assumptions are made a priori about the functional form.
- Other terms: Instance-based, Memory-based, Lazy, Case-based, kernel methods…

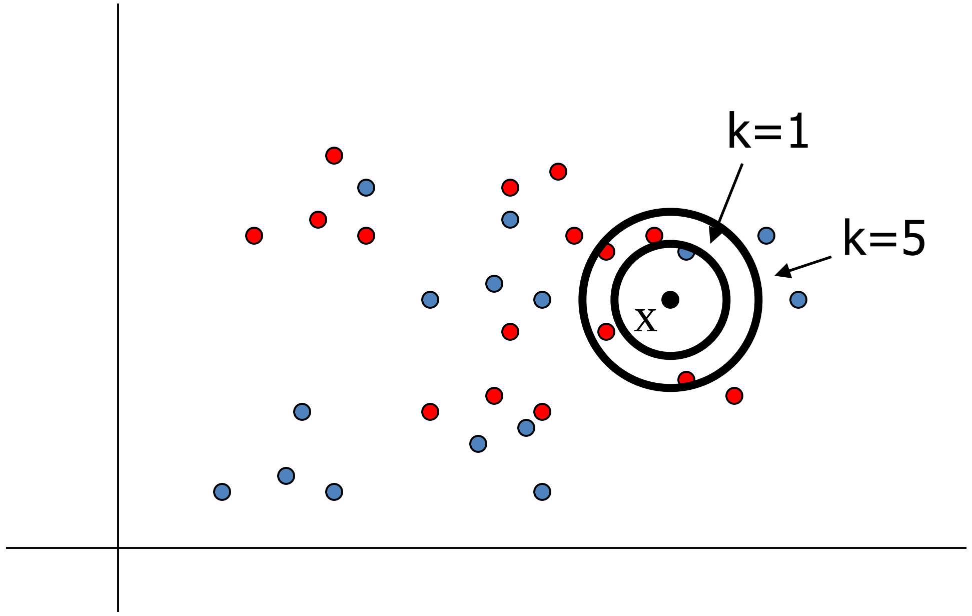
K-Nearest Neighbor Algorithm

- Learning Algorithm:
    - Store all the training examples.
- Prediction Algorithm:
    - For the given test example x, find k training examples $\{(x_i, y_i)\}_{i=1}^{k}$ that are the nearest to x.
    - If *classification problem*, **return** majority class among the k examples
    - If *regression problem*, **return** average y value of the k examples. Namely,

$$y = \frac{1}{k} \sum_{i=1}^{k} y_i$$

K-Nearest Neighbor: Example

- To classify a new input vector x, examine the k-closest training data points to x and assign the object to the most frequently occurring class

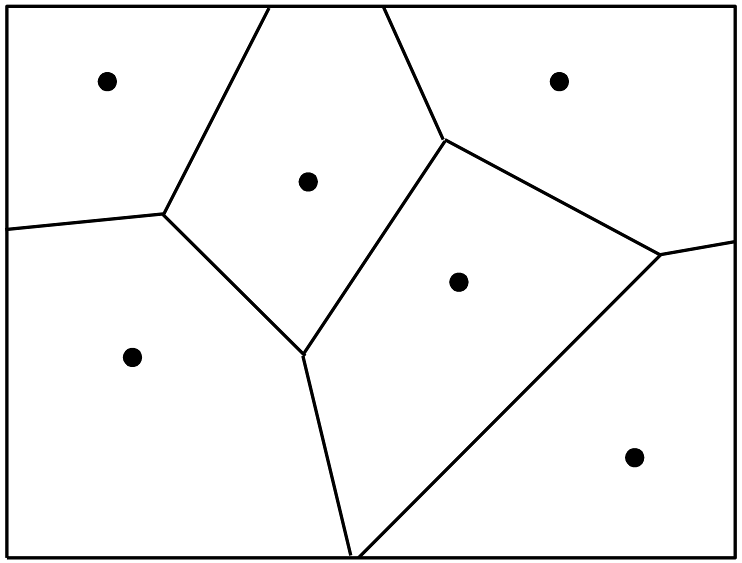


common values for k: 3, 5

Decision Boundaries

- The nearest neighbor algorithm does not explicitly compute decision boundaries. However, the decision boundaries form a subset of the Voronoi diagram for the training data.

1-NN Decision Surface



- Each line segment is equidistant between two points of opposite classes. The more examples that are stored, the more complex the decision boundaries can become.

Distance-Weighted *k*-NN

- Might want to attach a heavier weight to nearer points than farther ones
- Prediction rule:

$$y = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i}$$

- Question: What w_i to use?

$$w_i = \frac{1}{d(x, x_i)^2}$$

  where $d(x, x_i)$ is the distance between x and x_i.
  (Can you think of others?)
- Here it makes more sense to make use of all the training data (costly though).
  - Shephard’s method

Issues

- What Distance measure to use?
- How to speed up Classification?
  - K-NN is a memory-based technique.
  - Must make a pass through the data for each classification. This can be prohibitive for large data sets.
- Disadvantages
  - Curse of Dimensionality
    - In high-dimensional spaces, problem that the nearest neighbor may not be very close at all!
  - Irrelevant Attributes
    - Easily fooled by irrelevant attributes.

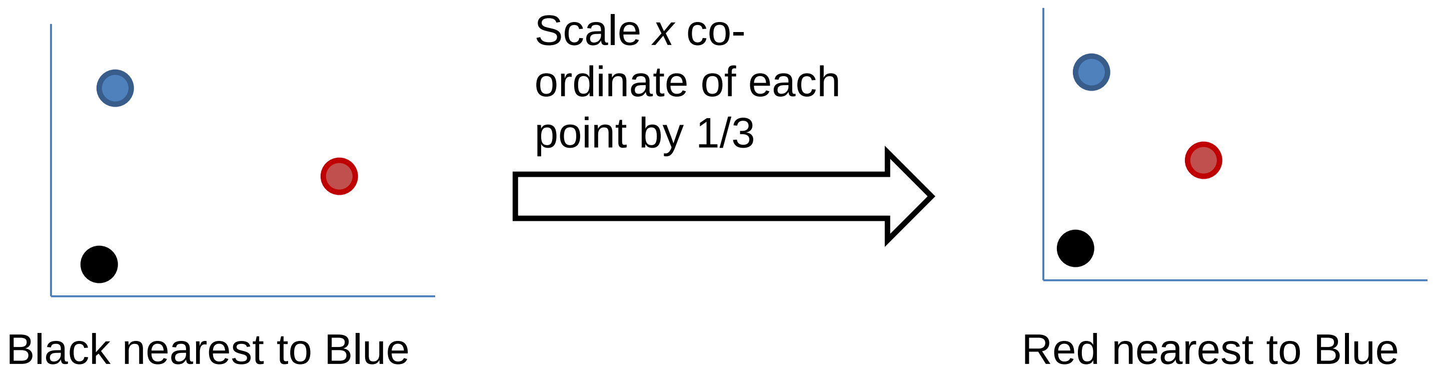
Distance Metrics
Euclidean Distance

- **Properties of Metrics:** For any two vectors *a* and *b*
  - $D(a, b) \geq 0$ (Positive)
  - $D(a, b) = 0$ iff $a = b$ (Reflexive)
  - $D(a, b) = D(b, a)$ (Symmetric)
  - For any other vector c, $D(a, b) + D(b, c) \geq D(a, c)$ (Triangle Inequality)
- **Euclidean Distance:** Most widely used metric

$$D(x_i, x_j) = \left(\sum_{a=1}^{d} (x_{i,a} - x_{j,a})^2 \right)^{\frac{1}{2}}$$

Euclidean Distance: Problems with Scaling

- If we scale each attribute arbitrarily, nearest points may become farthest points and vice versa.
- Example: multiply some co-ordinate of each point by an arbitrary constant.



Euclidean Distance: Practical Considerations

- Euclidean Distance: Makes sense in the case where the different measurements are commensurate; each variable measured in the same units.
- When attributes or features are not commensurate, we can standardize them by dividing each of its value in the data by the sample standard deviation. This makes them all equally important.
- Weighted Euclidean distance: If we have some idea of the relative importance of each feature, we can weigh them:

$$D_w(x_i, x_j) = \left(\sum_{a=1}^{d} w_a (x_{i,a} - x_{j,a})^2 \right)^{\frac{1}{2}}$$

Generalization of Euclidean Distance

- Minkowski distance or L_k norm:

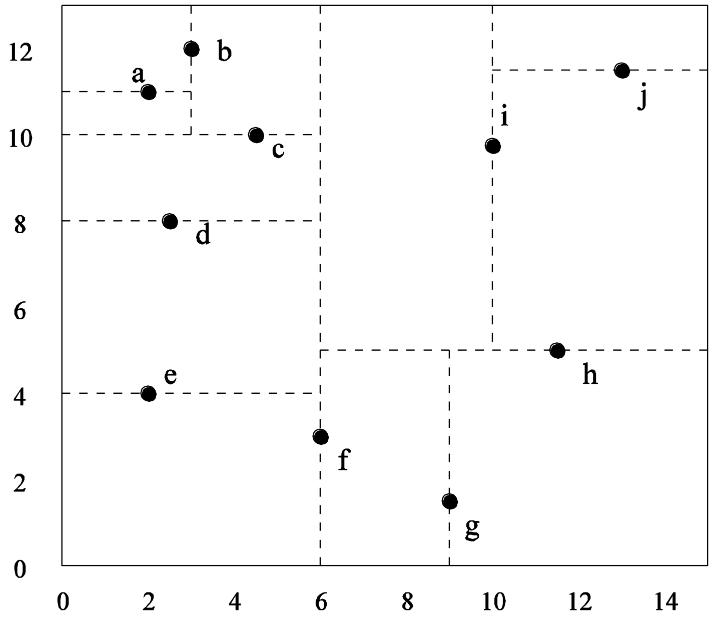
$$L_k(x_i, x_j) = \left(\sum_{a=1}^{d} |x_{i,a} - x_{j,a}|^k \right)^{\frac{1}{k}}$$

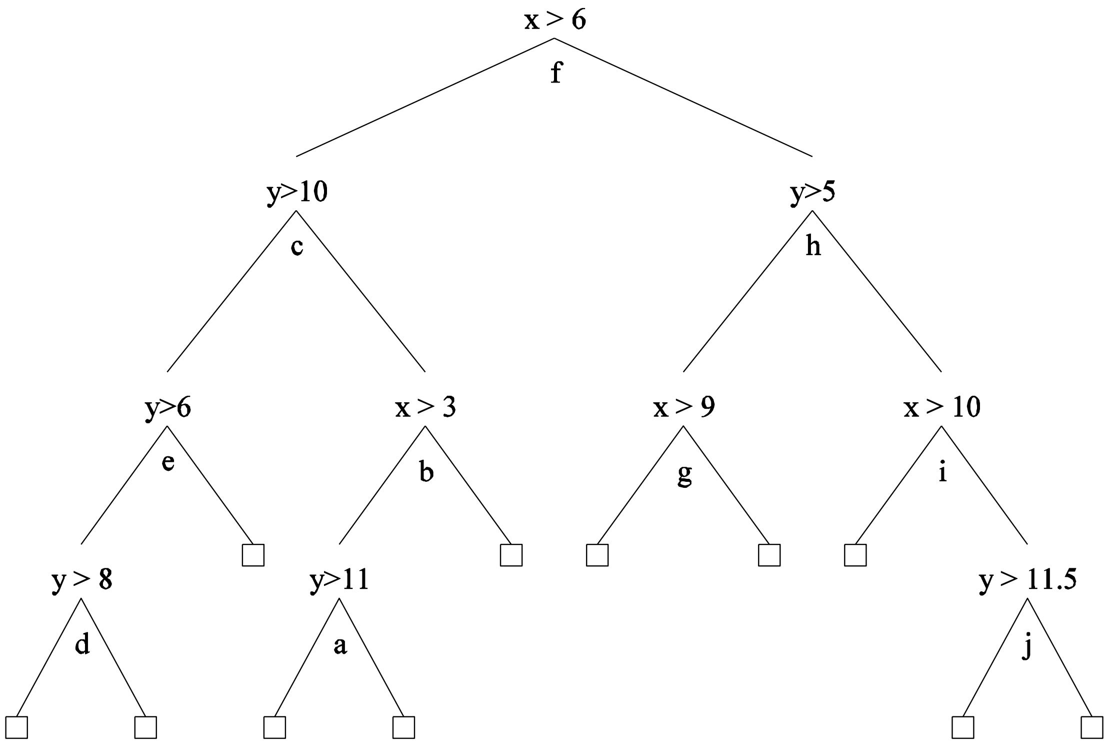
- Manhattan Distance: L_1 norm
- Euclidean Distance: L_2 norm
- L_∞ norm is the maximum of the projected distances.

$$L_\infty(x_i, x_j) = \max_a |x_{i,a} - x_{j,a}|$$

Efficient Indexing: Kd-trees

- A kd-tree is similar to a decision tree, except that we split using the median value along the dimension having the highest variance, and points are stored at the leaves. (See Wikipedia article)!



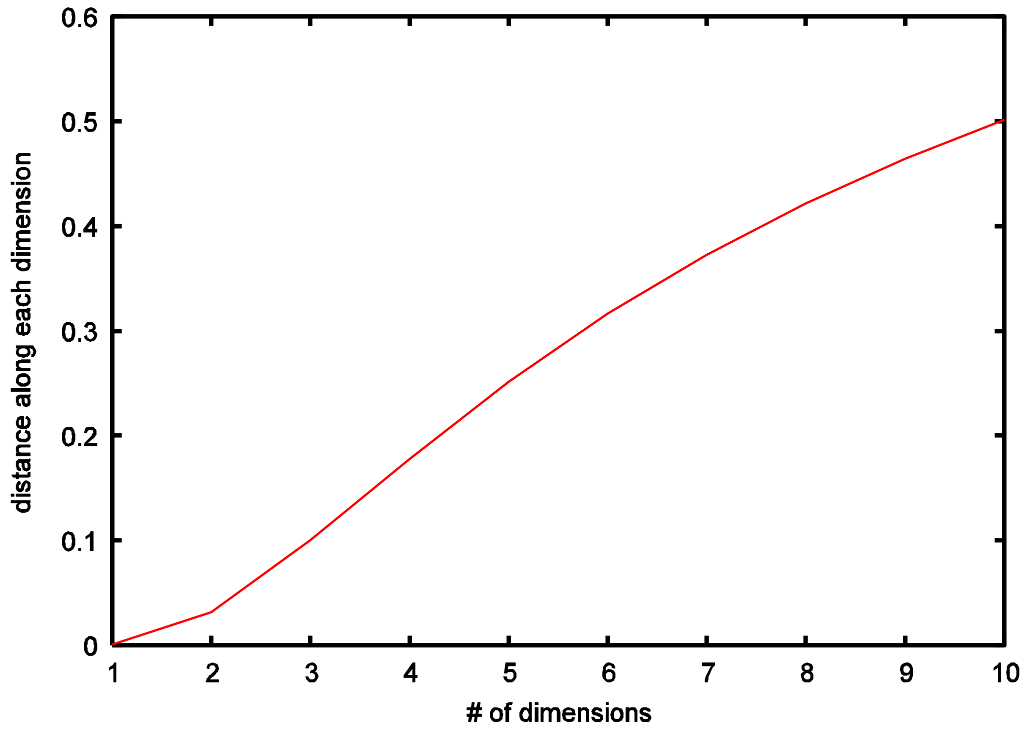


The Curse of Dimensionality

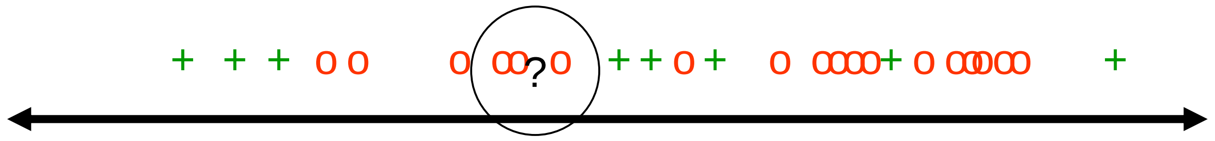
- Nearest neighbor breaks down in high-dimensional spaces because the "neighborhood" becomes very large.
- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply the 5-nearest neighbor algorithm.
- Suppose our query point is at the origin.
  - 1D –
    - On a one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

Curse of Dimensionality cont.

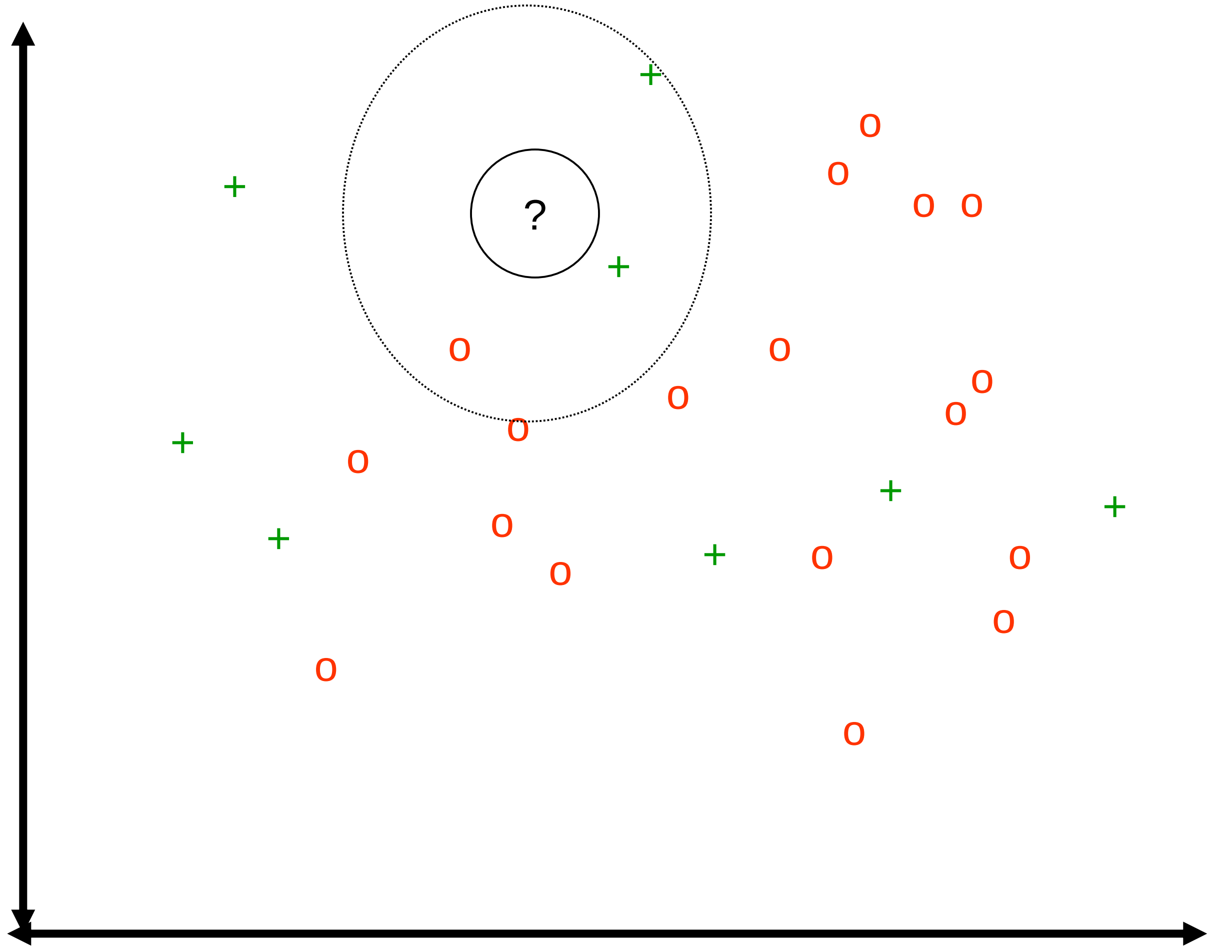
- With 5000 points in 10 dimensions, we must go 0.501 distance along each attribute in order to find the 5 nearest neighbors!



K-NN and irrelevant features



K-NN and irrelevant features



Edited Nearest Neighbor

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

KNN Advantages

- Easy to program
- No optimization or training required
- Classification accuracy can be very good; can outperform more complex models

Nearest Neighbor Summary

- Advantages
  - variable-sized hypothesis space
  - Learning is extremely efficient
    - 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

Locally Weighted Linear Regression: LWLR

- Idea:
  - k-NN forms local approximation for each query point x
  - Why not form an explicit approximation $\hat{f}$ for region surrounding x
    - Fit linear function to k nearest neighbors
    - Fit quadratic, ...
    - Thus producing ``piecewise approximation'' to $\hat{f}$
      - Minimize error over k nearest neighbors of x
      - Minimize error entire set of examples, weighting by distances
      - Combine two above

LWLR: Continued

- Linear Regression: $f(x) = w_0 + \sum_{i=1}^{n} w_i x_i$
- Error:

$$\frac{1}{2}\sum_{d \in D}(y - f(x_d))^2$$

- Minimize error over k nearest neighbors of x

$$\frac{1}{2}\sum_{k \in Nearest(x_d)}(y - f(x_k))^2$$

- Minimize error entire set of examples, weighting by distances

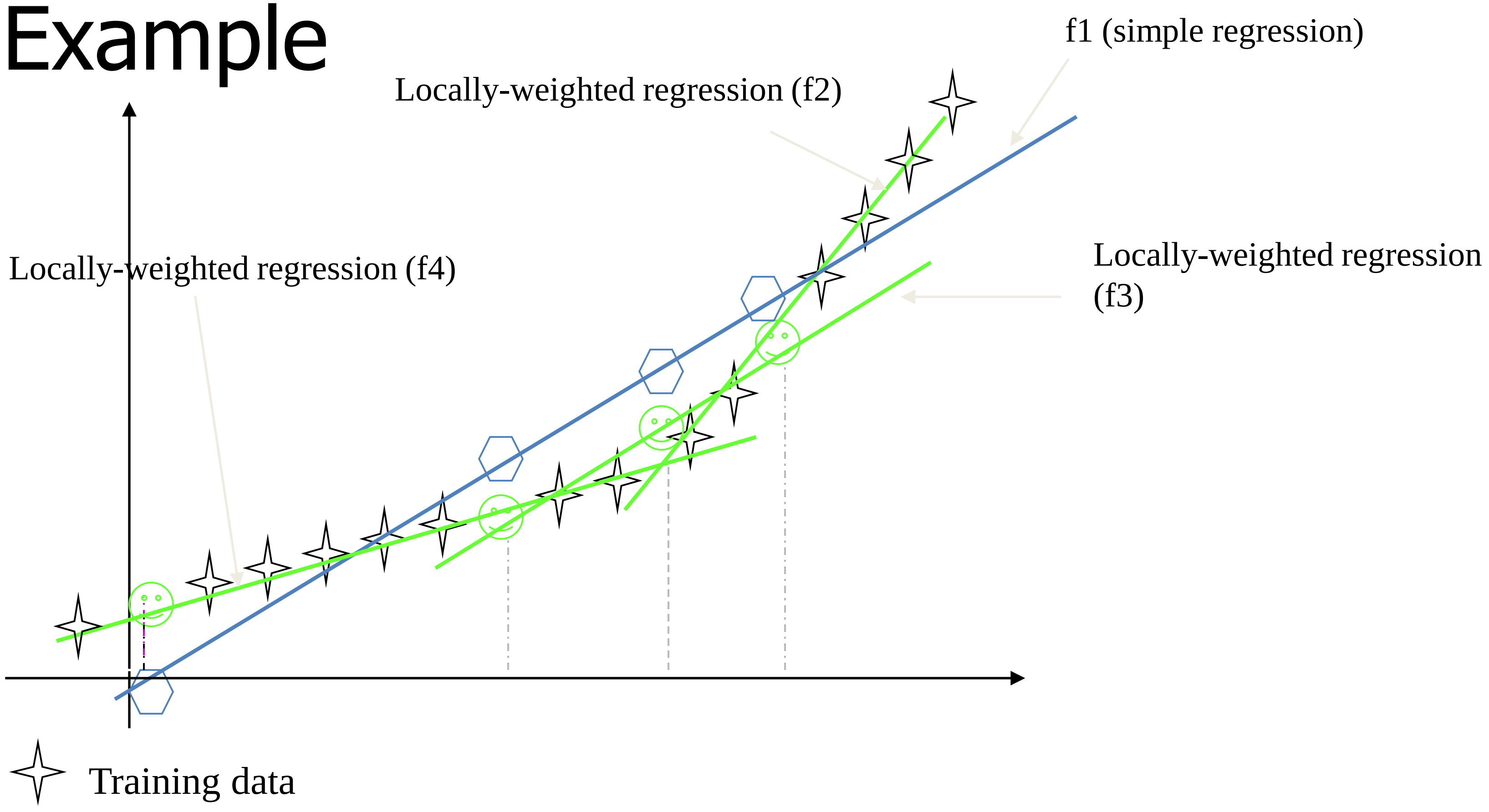
$$\frac{1}{2}\sum_{d \in D}(y - f(x_d))^2 K(Distance(x, x_d))$$

- Combine the two

$$\frac{1}{2}\sum_{k \in Nearest(x_d)}(y - f(x_d))^2 K(Distance(x, x_d))$$

where $K(Distance(x, x_d))$ is some function of the distance.

LWR Example



f1 (simple regression)
Locally-weighted regression (f2)
Locally-weighted regression (f3)
Locally-weighted regression (f4)

✧ Training data

⬡ Predicted value using simple regression

☺ Predicted value using locally weighted (piece-wise) regression

Lazy and Eager Learning

- Lazy: wait for query before generalizing
  - k-Nearest Neighbor
- Eager: generalize before seeing query
  - ID3, Backpropagation, etc.

Does it matter?

- Eager learner must create global approximation
- Lazy learner can create many local approximations
- If they use same *H*, lazy can represent more complex functions

Collaborative Filtering

(AKA Recommender Systems)

- **Problem:**
  Predict whether someone will like a Web page, newsgroup posting, movie, book, CD, etc.
- **Previous approach:**
  Look at content
- **Collaborative filtering:**
  - Look at what similar users liked
  - Similar users = Similar likes & dislikes

Collaborative Filtering

- Represent each user by vector of ratings
- Two types:
  - Yes/No
  - Explicit ratings (e.g., $0 - *****$)
- Predict rating:

$$\hat{R}_{ik} = \overline{R}_i + \alpha \sum_{X_j \in \mathbf{N}_i} W_{ij}(R_{jk} - \overline{R}_j)$$

- Similarity (Pearson coefficient):

$$W_{ij} = \frac{\sum_k (R_{ik} - \overline{R}_i)(R_{jk} - \overline{R}_j)}{\sqrt{\sum_k (R_{ik} - \overline{R}_i)^2 (R_{jk} - \overline{R}_j)^2}}$$

Fine Points

- Primitive version:

$$\hat{R}_{ik} = \alpha \sum_{X_j \in \mathbf{N}_i} W_{ij} R_{jk}$$

- $\alpha = (\sum |W_{ij}|)^{-1}$
- $\mathbf{N}_i$ can be whole database, or only k nearest neighbors
- R_{jk} = Rating of user j on item k
- $\overline{R}_j$ = Average of all of user j's ratings
- Summation in Pearson coefficient is over all items rated by *both* users
- In principle, any prediction method can be used for collaborative filtering

Example

	R_1	R_2	R_3	R_4	R_5	R_6
Alice	2	-	4	4	-	5
Bob	1	5	4	-	3	4
Chris	5	2	-	2	1	-
Diana	3	-	2	2	-	4

What you need to know

- Instance-based learning
  - non-parametric
  - trade decreased learning time for increased classification time
- Issues
  - appropriate distance metrics
  - curse of dimensionality
  - efficient indexing