Nearest neighbor methods
Lecture 10

David Sontag

New York University

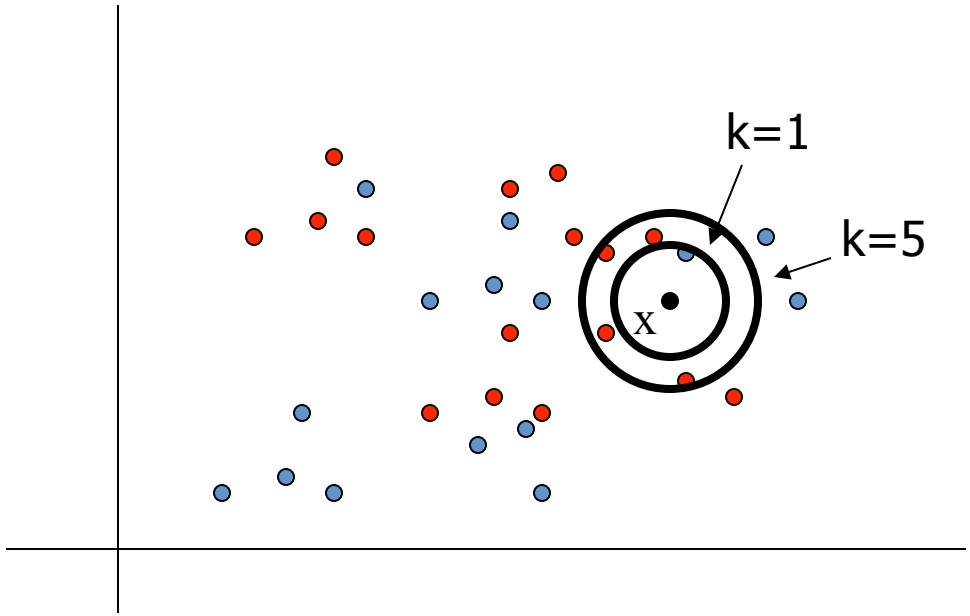
Slides adapted from Vibhav Gogate, Carlos Guestrin, Mehryar Mohri, & Luke Zettlemoyer

Nearest Neighbor Algorithm

- Learning Algorithm:
  - Store training examples
- Prediction Algorithm:
  - To classify a new example **x** by finding the training example ($\mathbf{x}^i$,y^i) that is *nearest* to **x**
  - Guess the class $y = y^i$

K-Nearest Neighbor Methods

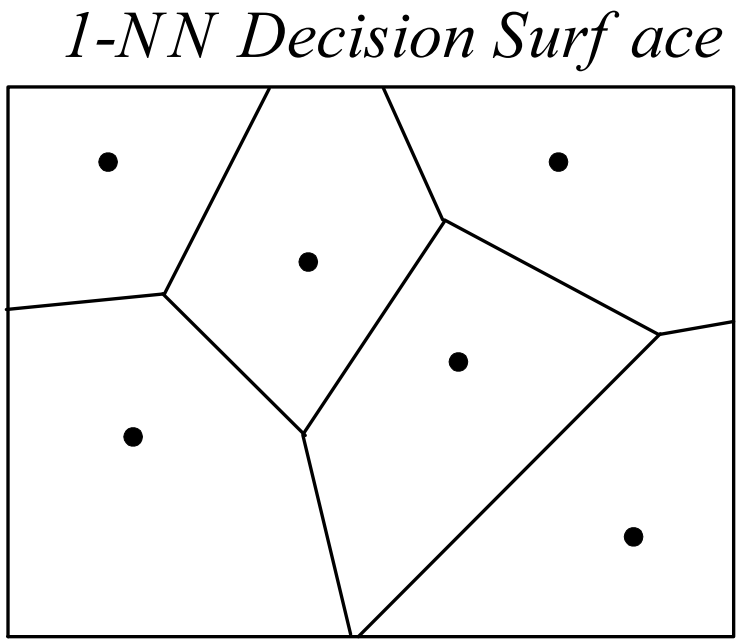
- To classify a new input vector x, examine the k-closest training data points to x and assign the object to the most frequently occurring class



common values for k: 3, 5

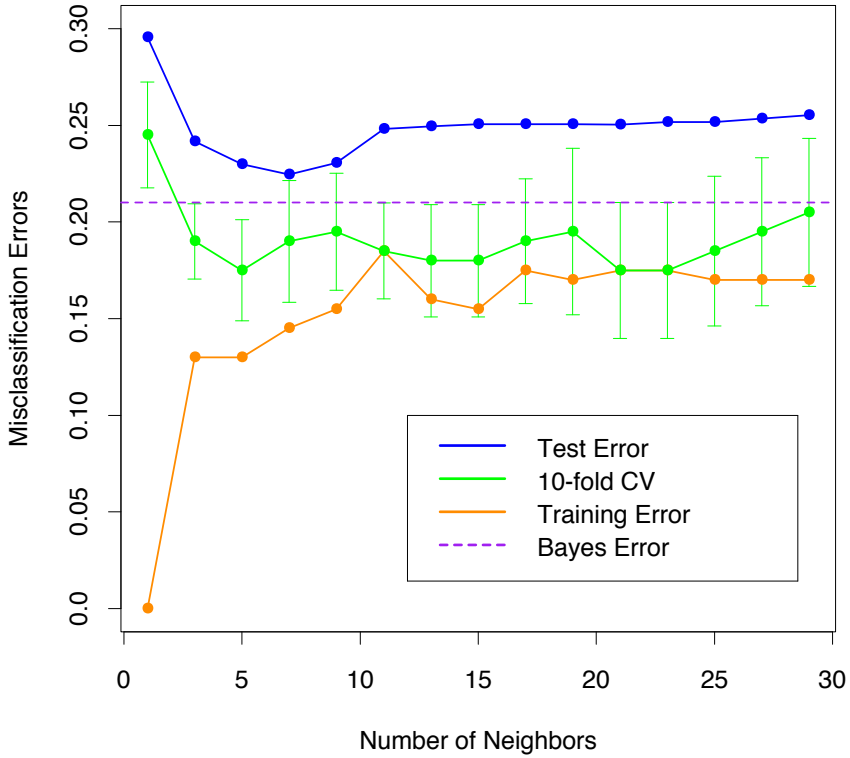
Decision Boundaries

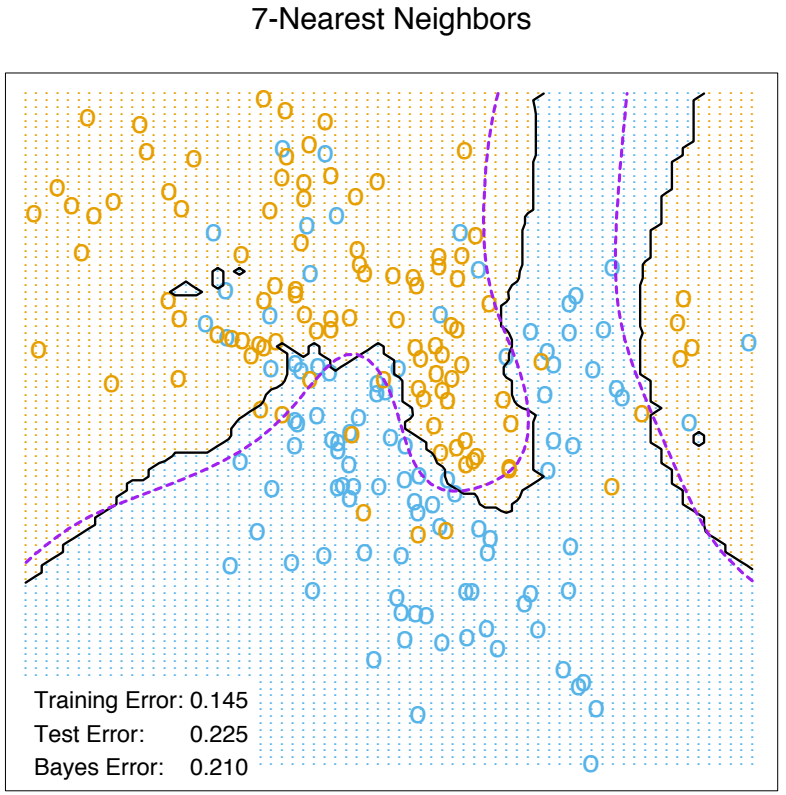
- The nearest neighbor algorithm does not explicitly compute decision boundaries. However, the decision boundaries form a subset of the Voronoi diagram for the training data.



- The more examples that are stored, the more complex the decision boundaries can become

Example results for k-NN





[Figures from Hastie and Tibshirani, Chapter 13]

Nearest Neighbor

When to Consider

- Instance map to points in R^n
- Less than 20 attributes per instance
- Lots of training data

Advantages

- Training is very fast
- Learn complex target functions
- Do not lose information

Disadvantages

- Slow at query time
- Easily fooled by irrelevant attributes

Issues

- Distance measure
  - Most common: Euclidean
- Choosing k
  - Increasing k reduces variance, increases bias
- For high-dimensional space, problem that the nearest neighbor may not be very close at all!
- Memory-based technique. Must make a pass through the data for each classification. This can be prohibitive for large data sets.

Distance

- Notation: object with p measurements

$$x^i = (x^i_1, x^i_2, \ldots, x^i_p)$$

- Most common distance metric is *Euclidean* distance:

$$d_E(x^i, x^j) = \left(\sum_{k=1}^{p} (x^i_k - x^j_k)^2 \right)^{\frac{1}{2}}$$

• ED makes sense when different measurements are commensurate; each is variable measured in the same units.

• If the measurements are different, say length and weight, it is not clear.

Standardization

When variables are not commensurate, we can standardize them by dividing by the sample standard deviation. This makes them all equally important.

The estimate for the standard deviation of x_k :

$$\hat{\sigma}_k = \left(\frac{1}{n} \sum_{i=1}^{n} \left(x_k^i - \bar{x}_k \right)^2 \right)^{\frac{1}{2}}$$

where x_k is the sample mean:

$$\bar{x}_k = \frac{1}{n} \sum_{i=1}^{n} x_k^i$$

Weighted Euclidean distance

Finally, if we have some idea of the relative importance of each variable, we can weight them:

$$d_{WE}(i,j) = \left(\sum_{k=1}^{p} w_k (x^i_k - x^j_k)^2 \right)^{\frac{1}{2}}$$

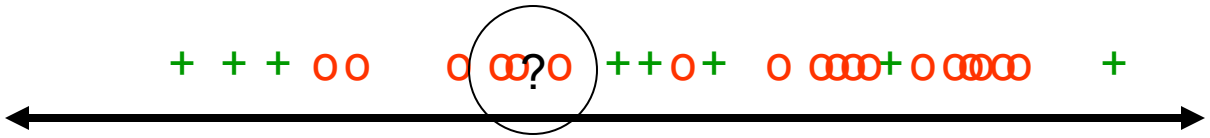
The Curse of Dimensionality

- Nearest neighbor breaks down in high-dimensional spaces because the "neighborhood" becomes very large.

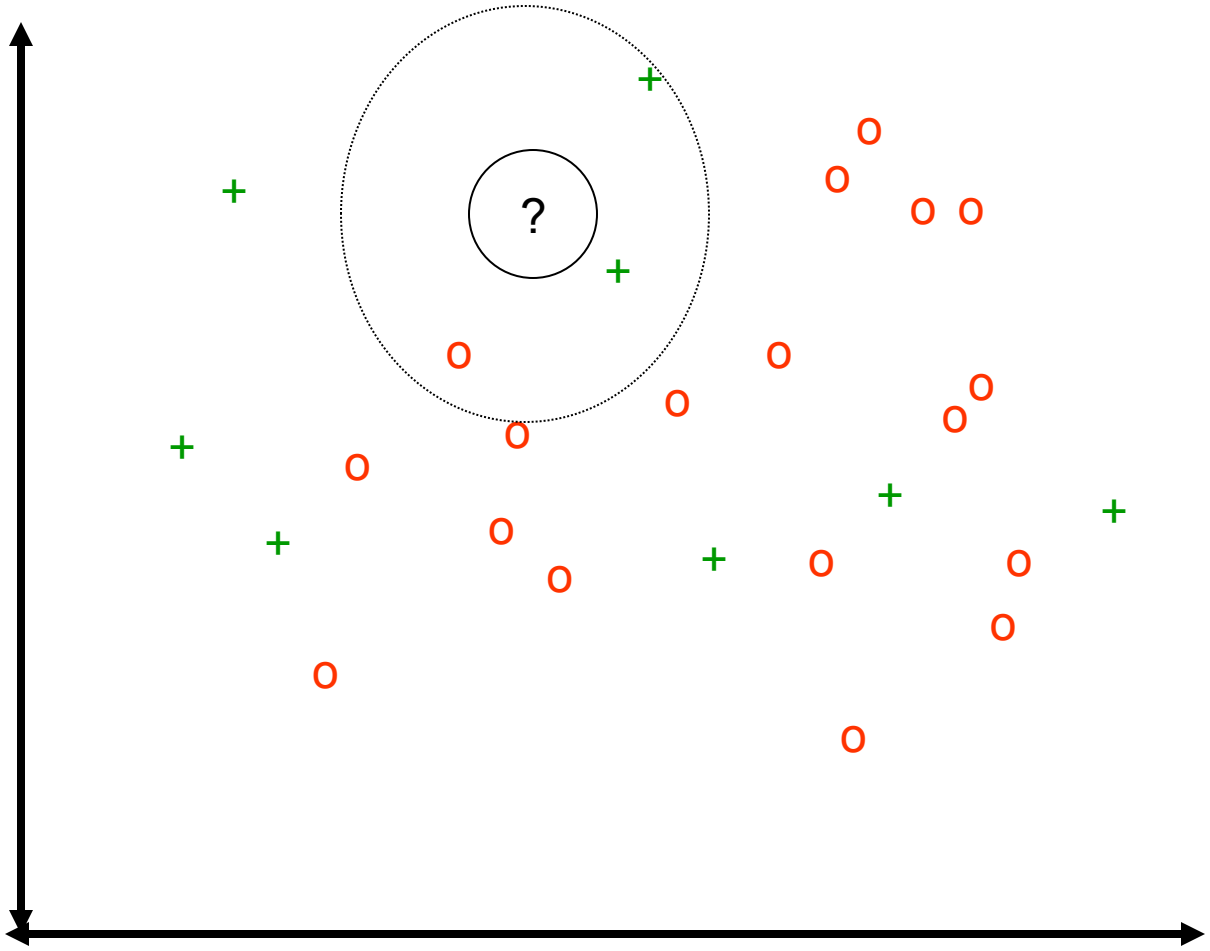
- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply the 5-nearest neighbor algorithm.

- 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 $(0.001)^{1/D}$

K-NN and irrelevant features



K-NN and irrelevant features



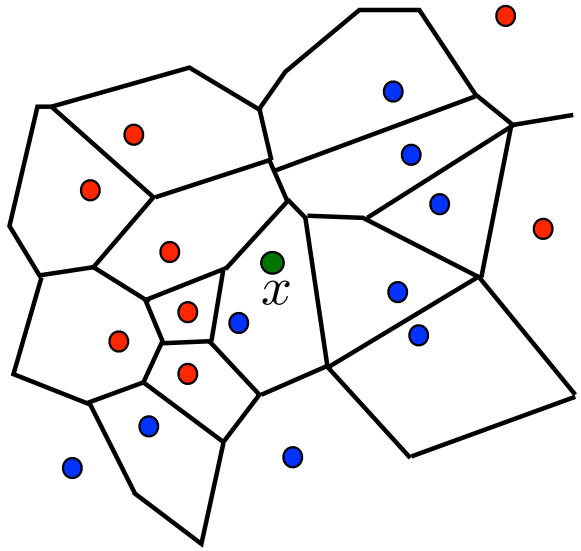
Nearest neighbor problem

- Problem: given sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$, find the nearest neighbor of test point x.
  - general problem extensively studied in computer science.
  - exact vs. approximate algorithms.
  - dimensionality N crucial.
  - better algorithms for small intrinsic dimension (e.g., limited doubling dimension).

[Slides from Mehyrar Mohri]

Efficient Indexing: N=2

- Algorithm:
  - compute Voronoi diagram in $O(m \log m)$.
  - point location data structure to determine NN.
  - complexity: $O(m)$ space, $O(\log m)$ time.



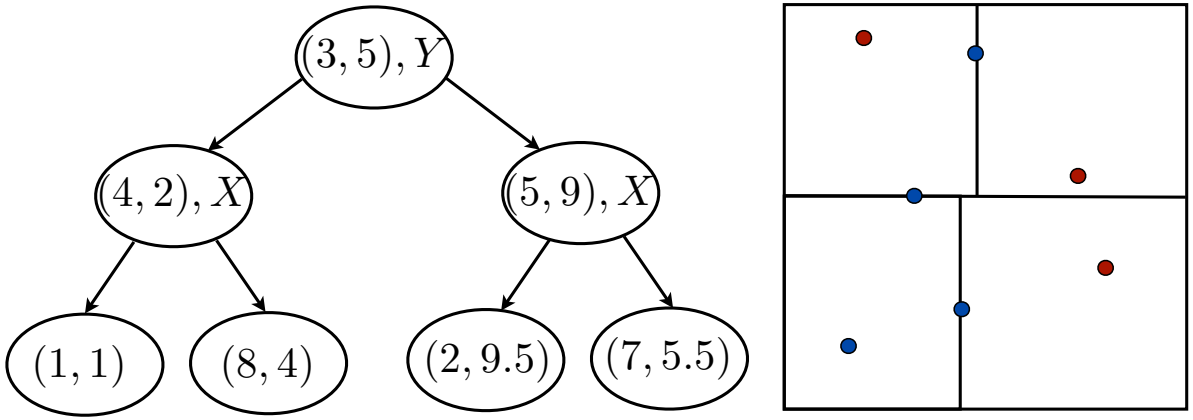
[Slides from Mehyrar Mohri]

Efficient Indexing: N>2

- Voronoi diagram: size in $O\left(m^{\lceil N/2 \rceil}\right)$.
- Linear algorithm (no pre-processing):
  - compute distance $\|x - x_i\|$ for all $i \in [1, m]$.
  - complexity of distance computation: $\Omega(Nm)$.
  - no additional space needed.
- Tree-based data structures: pre-processing.
  - often used in applications: k-d trees (k-dimensional trees).

[Slides from Mehyrar Mohri]

Efficient Indexing for N>2: KD trees



Construction algorithm

- Algorithm: for each non-leaf node,
  - choose dimension (e.g., longest of hyperrectangle).
  - choose pivot (median).
  - split node according to (pivot, dimension).

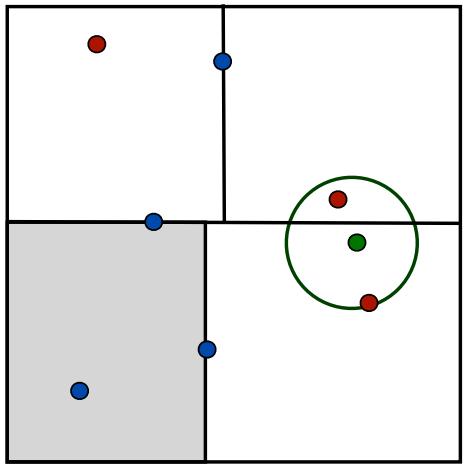
→ balanced tree, binary space partitioning.

[Slides from Mehyrar Mohri]

Efficient Indexing for N>2: KD trees

- Algorithm:
  - find region containing x (starting from root node, move to child node based on node test).
  - save region point x_0 as current best.
  - move up tree and recursively search regions intersecting hypersphere $S(x, \|x - x_0\|)$:
    - update current best if current point is closer.
    - restart search with each intersecting sub-tree.
    - move up tree when no more intersecting sub-tree.

Search algorithm



[Slides from Mehyrar Mohri]

KNN Advantages

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