Nearest Neighbor Methods

Rishabh Iyer
University of Texas at Dallas

Based on the slides of Vibhav Gogate, Nick Rouzzi, David Sontag and few other sources

Nearest Neighbor Methods

- Learning

  - Store all training examples

$T = \{(x_1, y_1), \ldots, (x_M, y_M)\}$

$\underbrace{\qquad\qquad\qquad}_{M \text{ Train Data points}}$

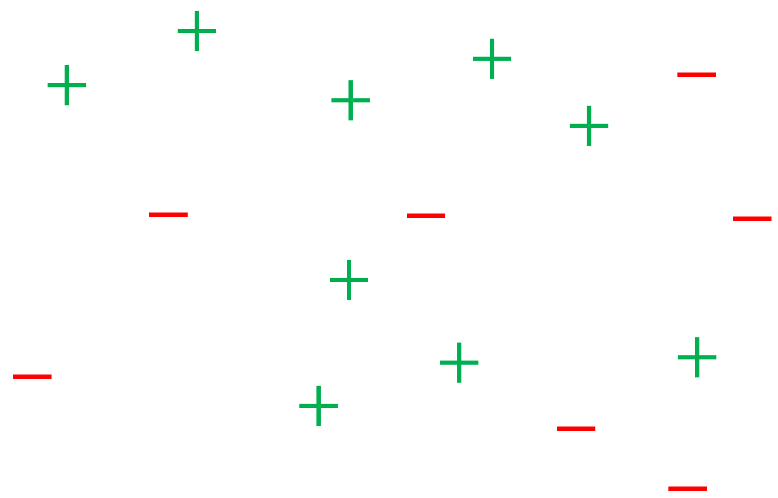
- Classifying a new point x'

  - Find the training example $(x^{(i)}, y^{(i)})$ such that $x^{(i)}$is closest (for some notion of close) to x'

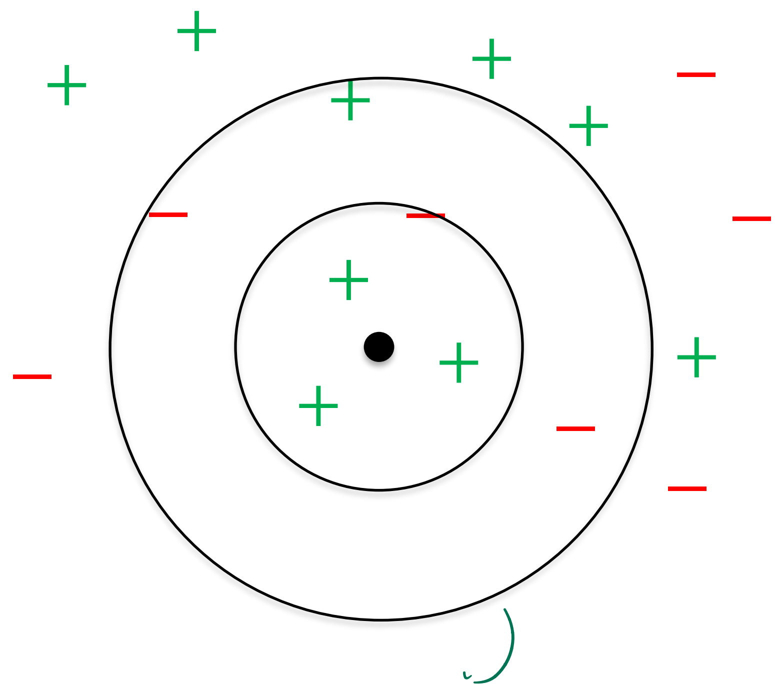
$x' \rightarrow$ Test point.

  - Classify x' with the label $y^{(i)}$

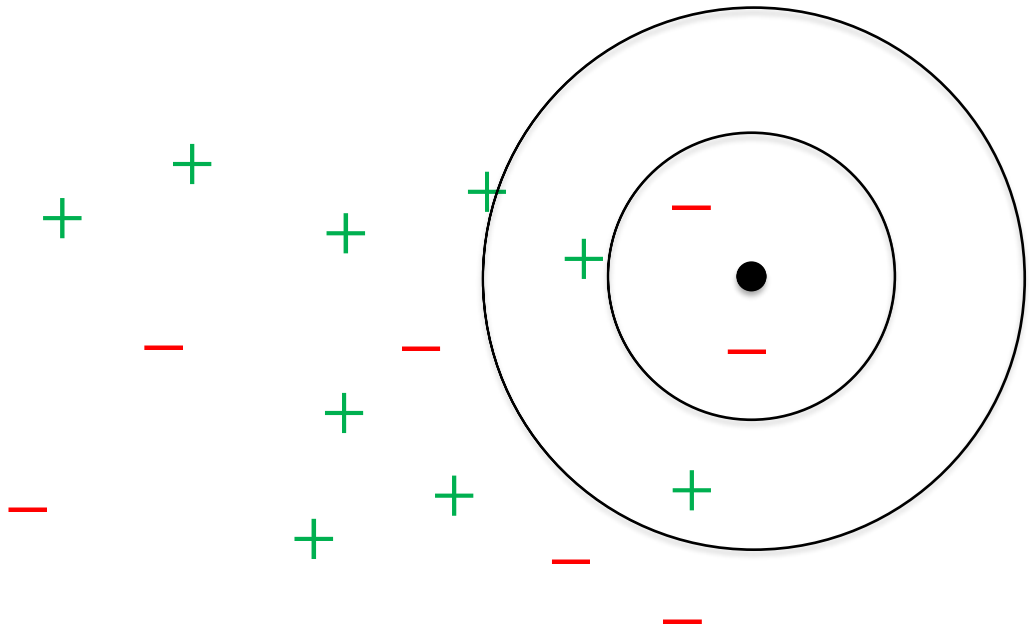
Nearest Neighbor Methods



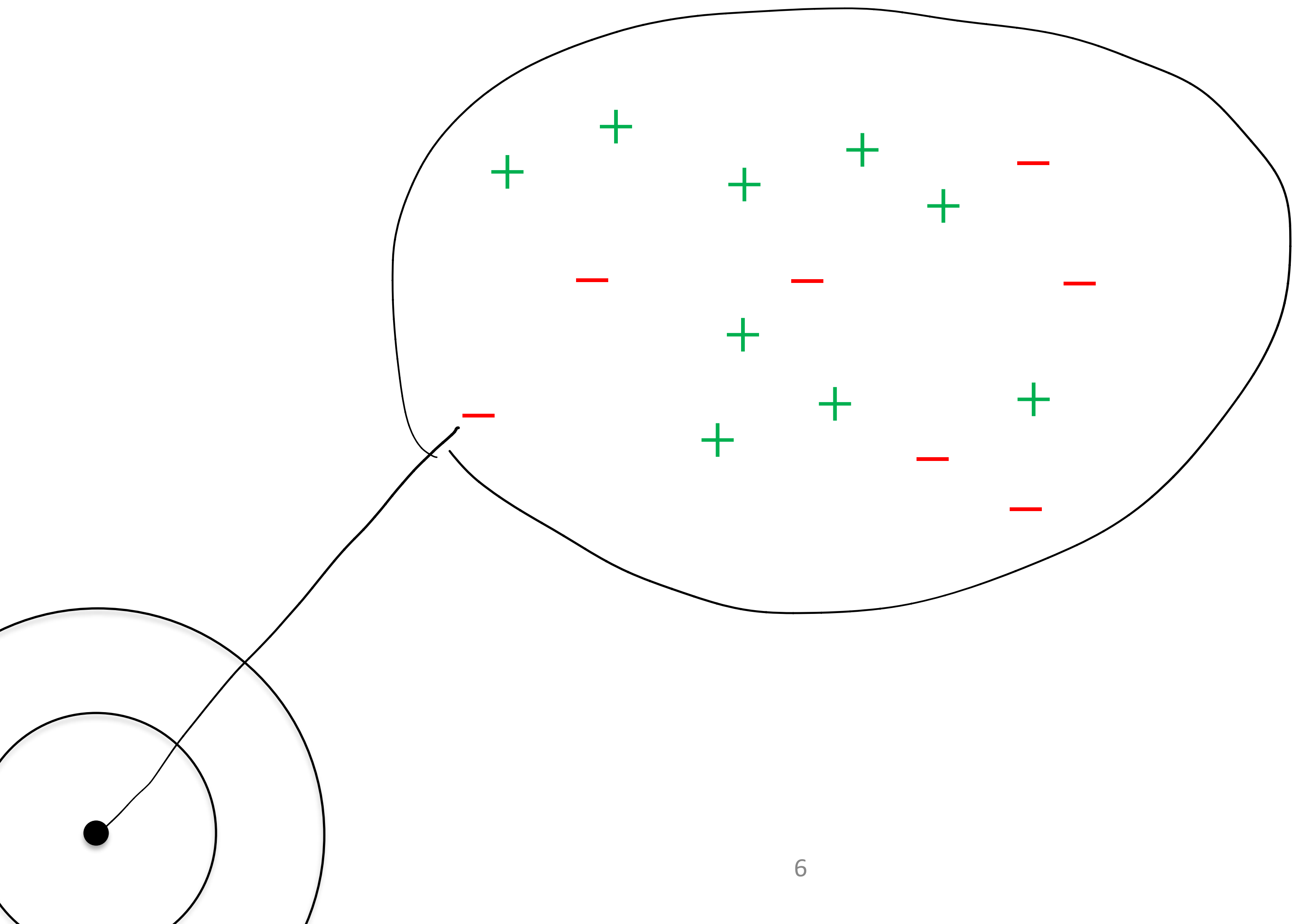
Nearest Neighbor Methods



Nearest Neighbor Methods



Nearest Neighbor Methods

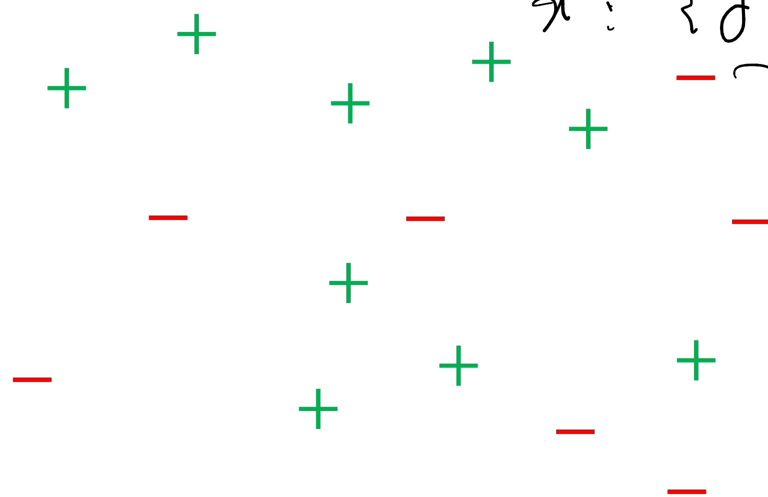


Nearest Neighbor Methods

1-NN : Closest.

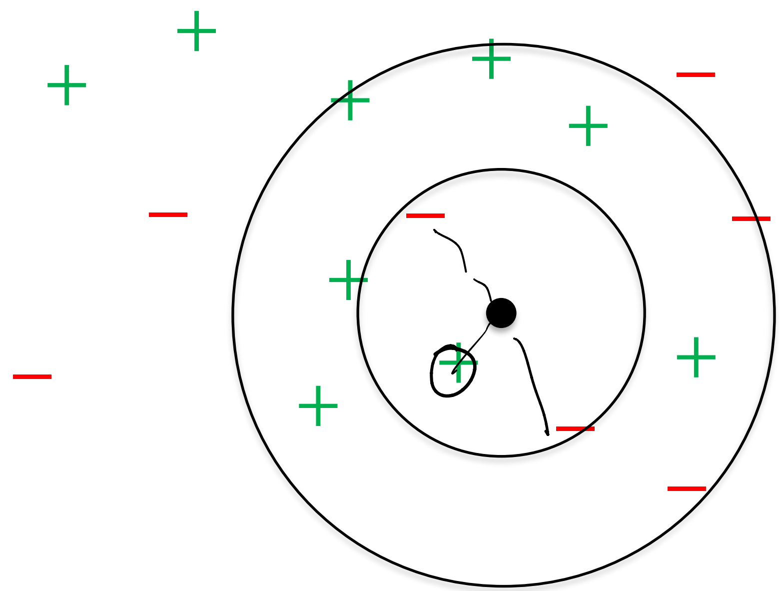
K-NN : k Closest points

x^1 : $\{y^1, \dots, y^k\}$



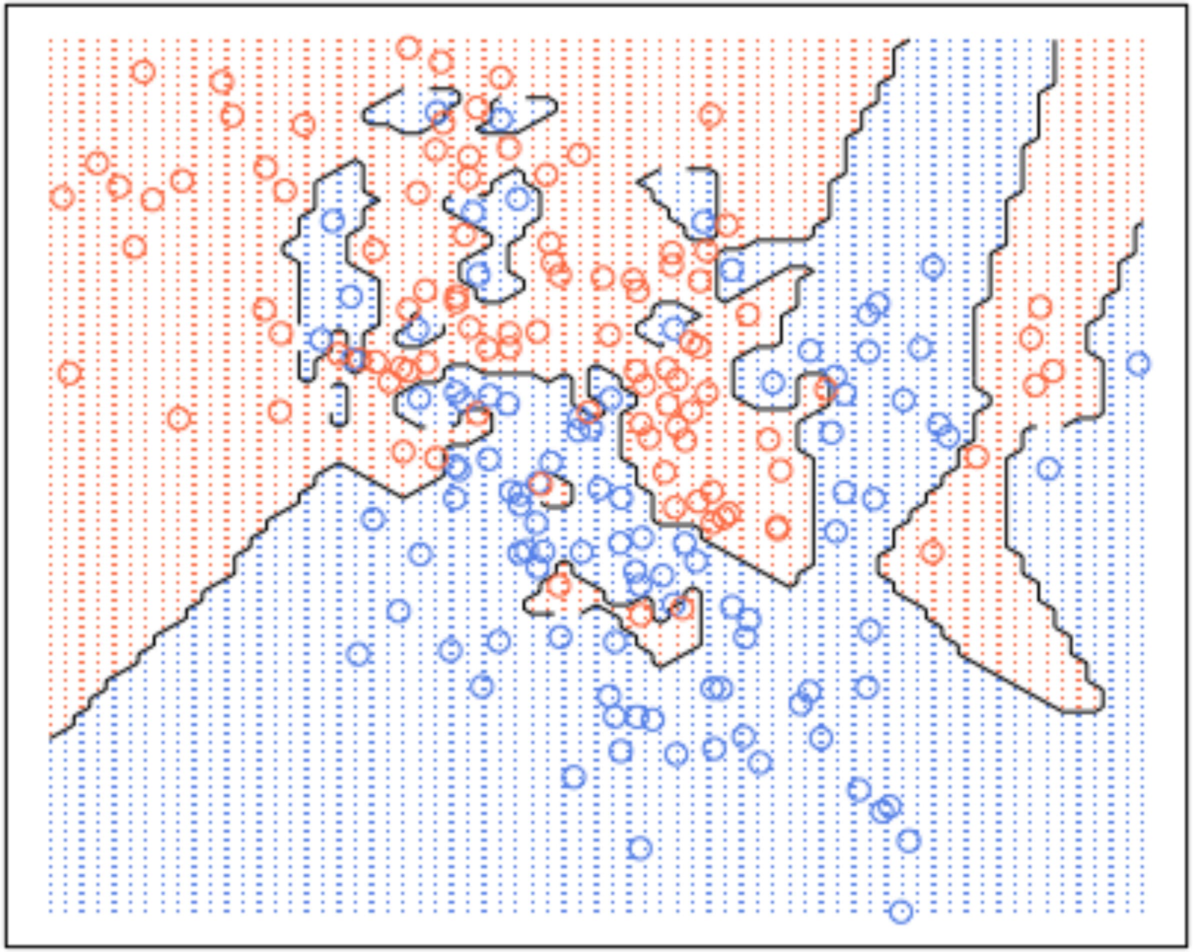
k-nearest neighbor methods look at the k closest points in the training set and take a majority vote (should choose k to be odd)

Nearest Neighbor Methods



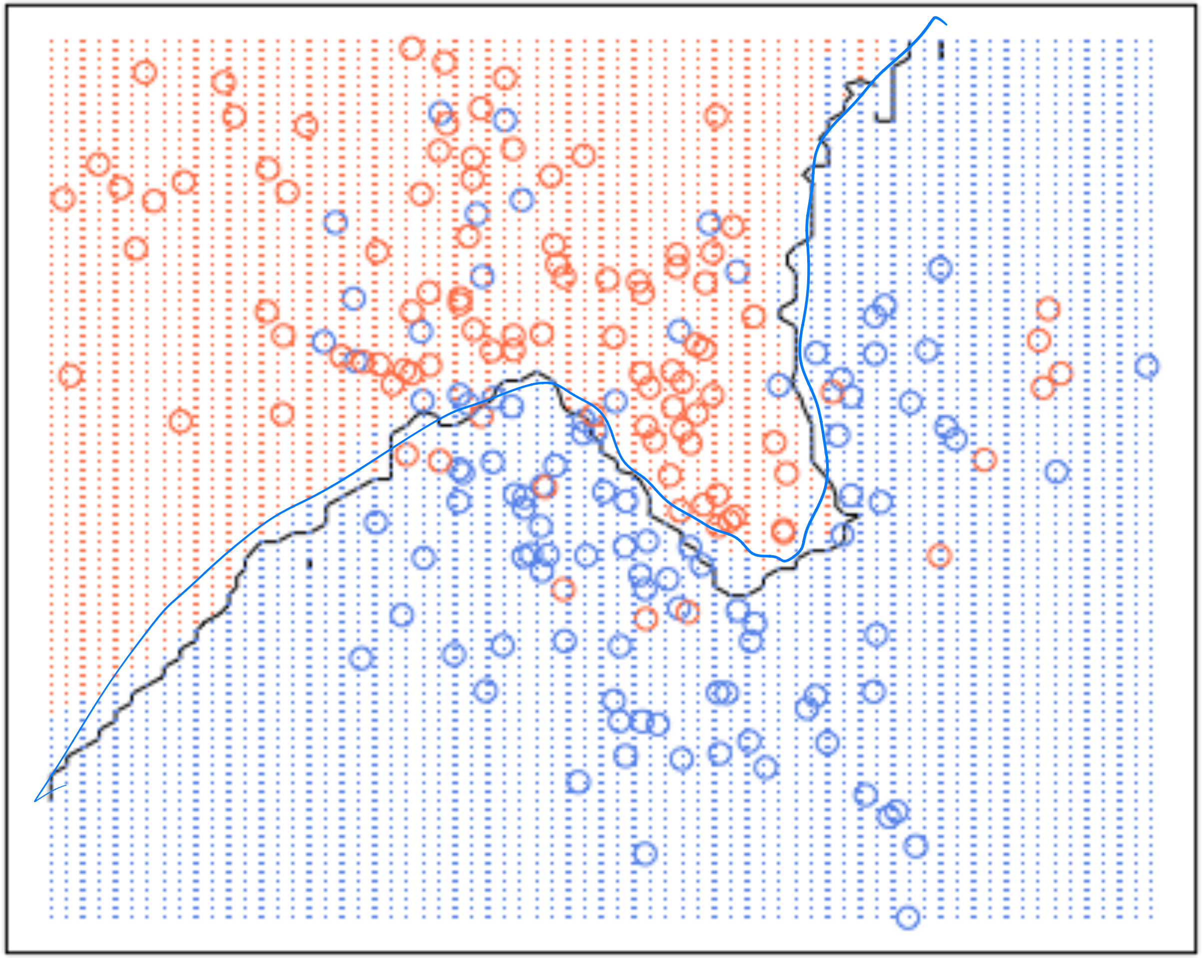
k-nearest neighbor methods look at the k closest points in the training set and take a majority vote
(should choose k to be odd)

1-NN Example



[Kevin Zakka]

20-NN Example



[Kevin Zakka]

Nearest Neighbor Methods

- Applies to data sets with points in $\mathbb{R}^d$
  - Best for large data sets with only a few (< 20) attributes
- Advantages
  - Learning is easy ← No training
  - Can learn complicated decision boundaries
- Disadvantages
  - Classification is slow (need to keep the entire training set around) M ~ 1M. ← ① Inference is slow testing ② High memory
  - Easily fooled by irrelevant attributes

Practical Challenges

- How to choose the right measure of closeness?
  - Euclidean distance is popular, but many other possibilities
- How to pick k?

$x', x^{(i)} \in \mathbb{R}^d$

  - Too small and the estimates are noisy, too large and the accuracy suffers
- What if the nearest neighbor is really far away?

$$\|\underset{\text{test}}{x'} - \underset{\text{train}}{x^{(i)}}\|_2^2 = \sum_{j=1}^{d} (x'_j - x^{(i)}_j)^2$$

Choosing the Distance

- Euclidean distance makes sense when each of the features is roughly on the same scale
  - If the features are very different (e.g., height and age), then Euclidean distance makes less sense as height would be less significant than age simply because age has a larger range of possible values

    5 20 - 30
  - To correct for this, feature vectors are often recentered around their means and scaled by the standard deviation over the training set

Normalization [Mean – Variance Normalization]

- Sample mean

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x^{(i)}$$

- Sample variance (biased)

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

x
$\downarrow$

$$x'_k = \frac{x_k - \bar{x}_k}{\sigma_k^2}$$

Min-Max Normalization

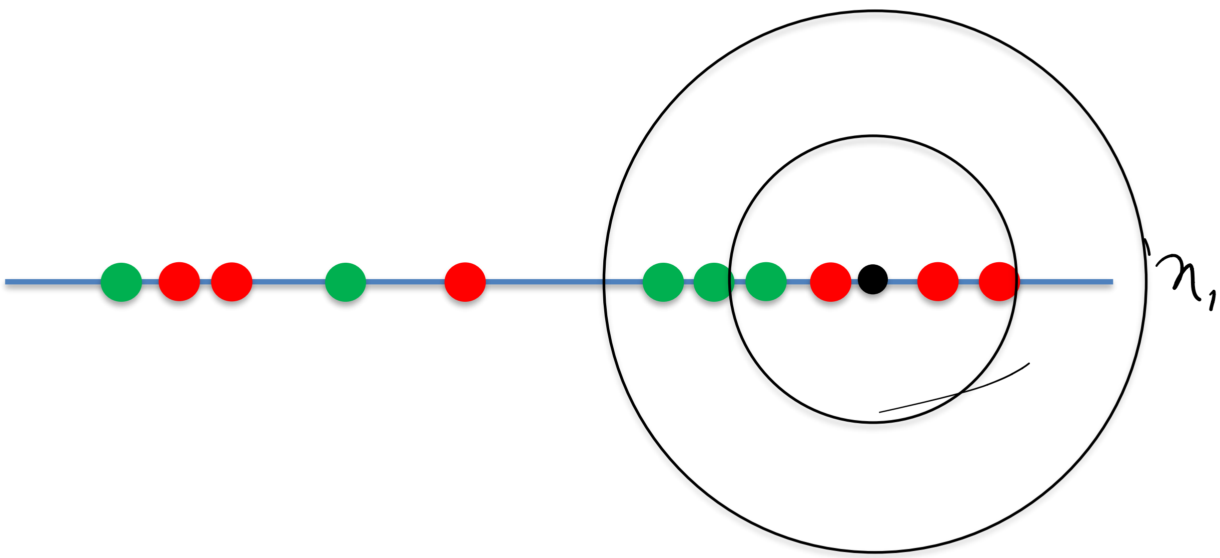
$$x_k^{max} = \max_{i=1:n} x_k^{(i)}$$

$$x_k^{min} = \min_{i=1:n} x_k^{(i)}$$

$$x_k'' = \frac{x_k - x_k^{min}}{x_k^{max} - x_k^{min}} \in [0,1]$$

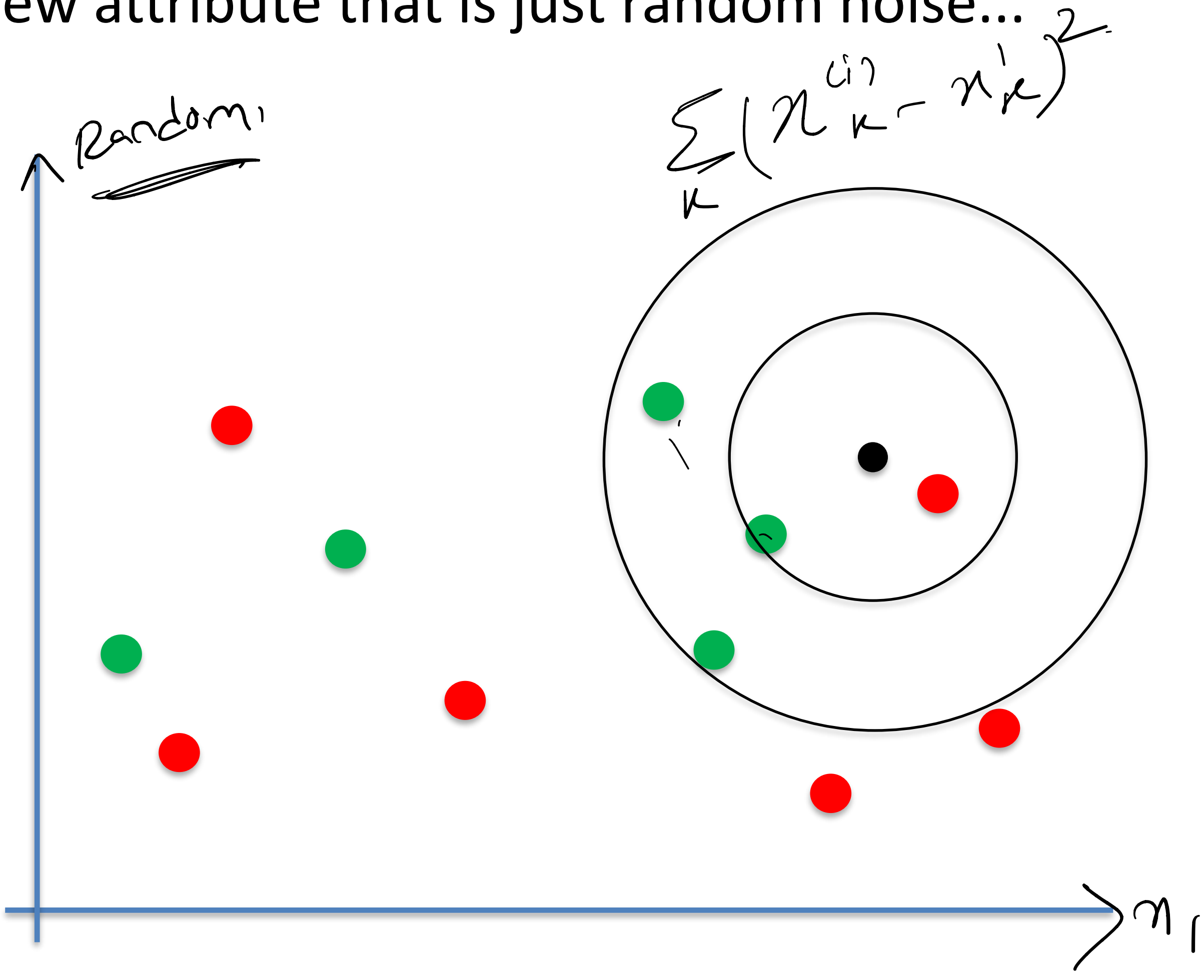
Irrelevant Attributes

Consider the nearest neighbor problem in one dimension



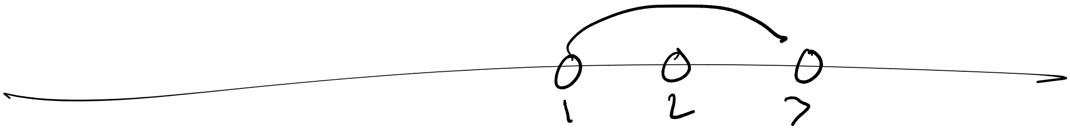
Irrelevant Attributes

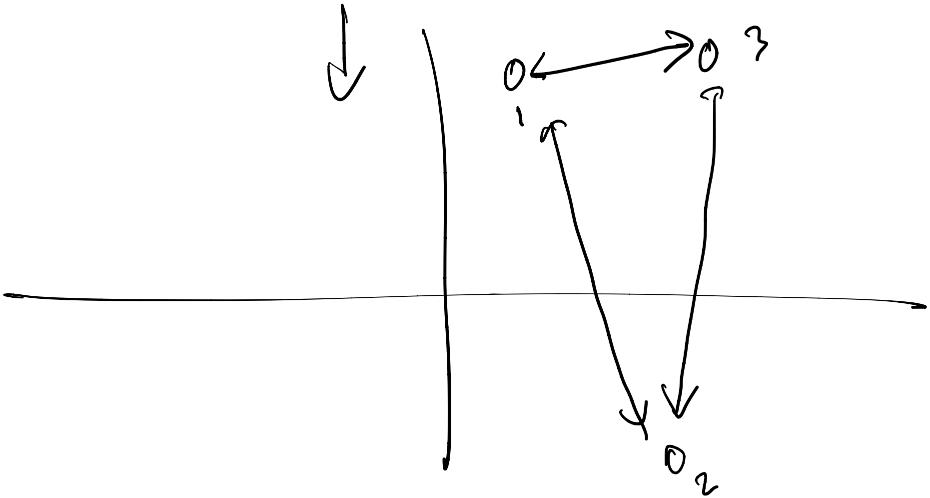
Now, add a new attribute that is just random noise...



$1 \to 2 \to 3$

$d_{12} \le d_{13}$





K-Dimensional Trees

inference / testing.

- In order to do classification, we can compute the distances between all points in the training set and the point we are trying to classify
  - With m data points in n-dimensional space, this takes $O(mn)$ time for Euclidean distance

    1M   100

  - It is possible to do better if we do some preprocessing on the training data

Test : ne ~ 100M (slow)

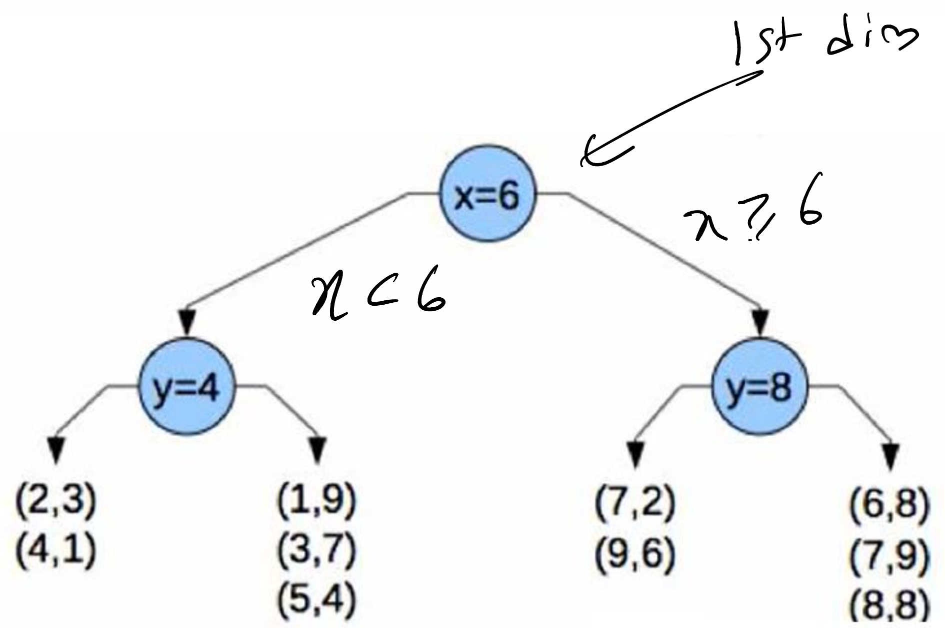
Prediction ~ ms

K-Dimensional Trees

- k-d trees provide a data structure that can help simplify the classification task by constructing a tree that partitions the search space
  - Starting with the entire training set, choose some dimension, i
  - Select an element of the training data whose i^{th} dimension has the median value among all elements of the training set
  - Divide the training set into two pieces: depending on whether their i^{th} attribute is smaller or larger than the median
  - Repeat this partitioning process on each of the two new pieces separately

K-Dimensional Trees

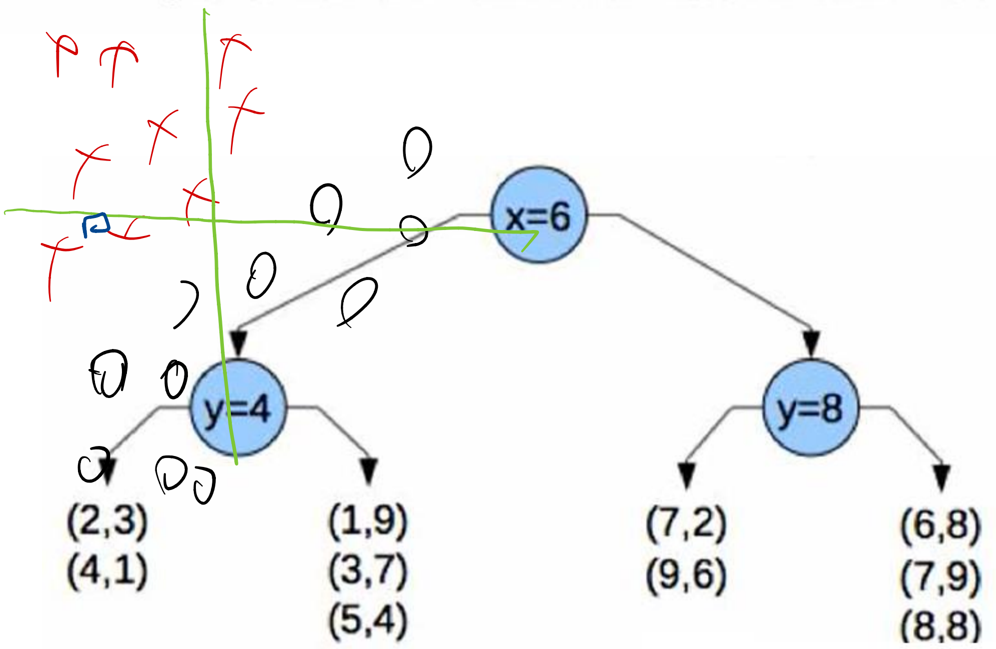
- Building a K-D tree from training data:
  - {(1,9), (2,3), (4,1), (3,7), (5,4), (6,8), (7,2), (8,8), (7,9), (9,6)}
  - pick random dimension, find median, split data, repeat

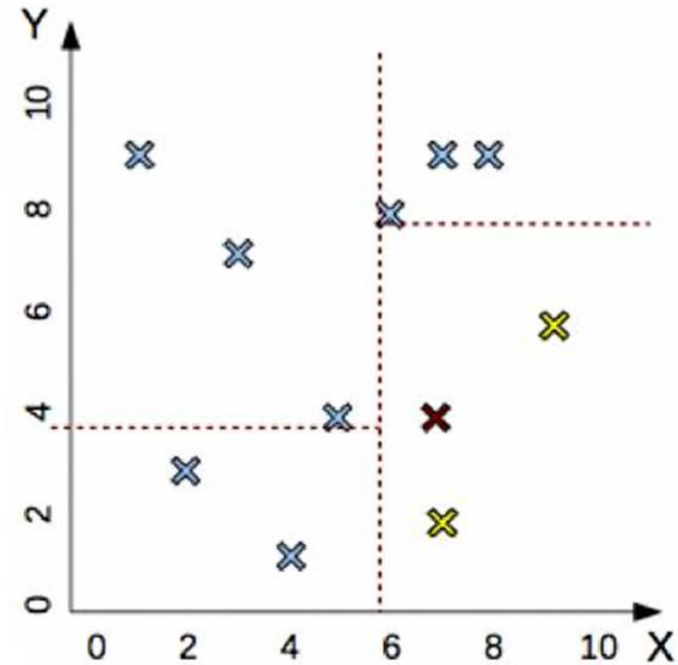


Adapted from Victor Lavrenko

K-Dimensional Trees

- Building a K-D tree from training data:
  - {(1,9), (2,3), (4,1), (3,7), (5,4), (6,8), (7,2), (8,8), (7,9), (9,6)}
  - pick random dimension, find median, split data, repeat



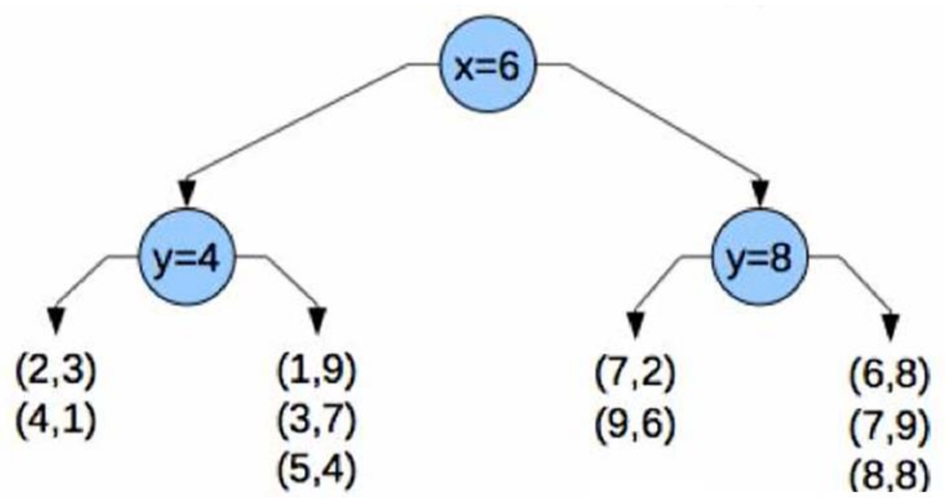


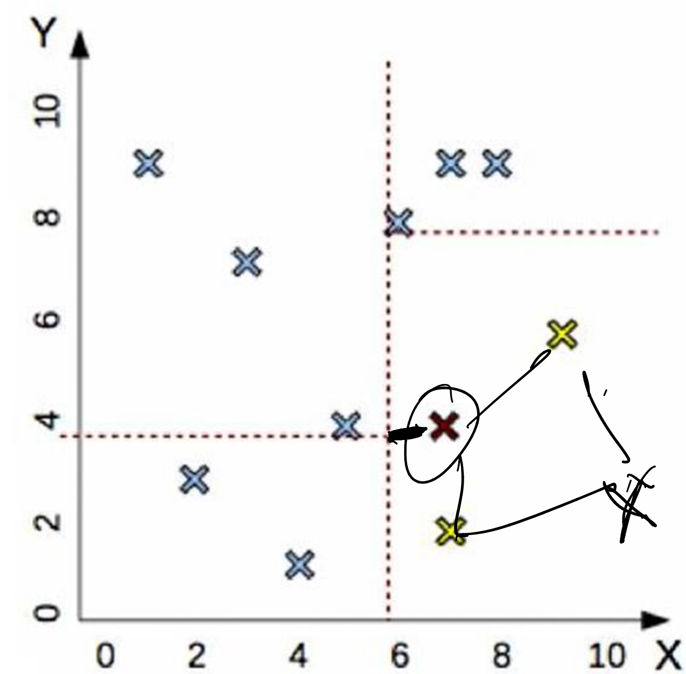
Adapted from Victor Lavrenko

K-Dimensional Trees: Inference

- Find NNs for new point (7,4) ← Test
  - find region containing (7,4)
  - compare to all points in region

Approximate NN



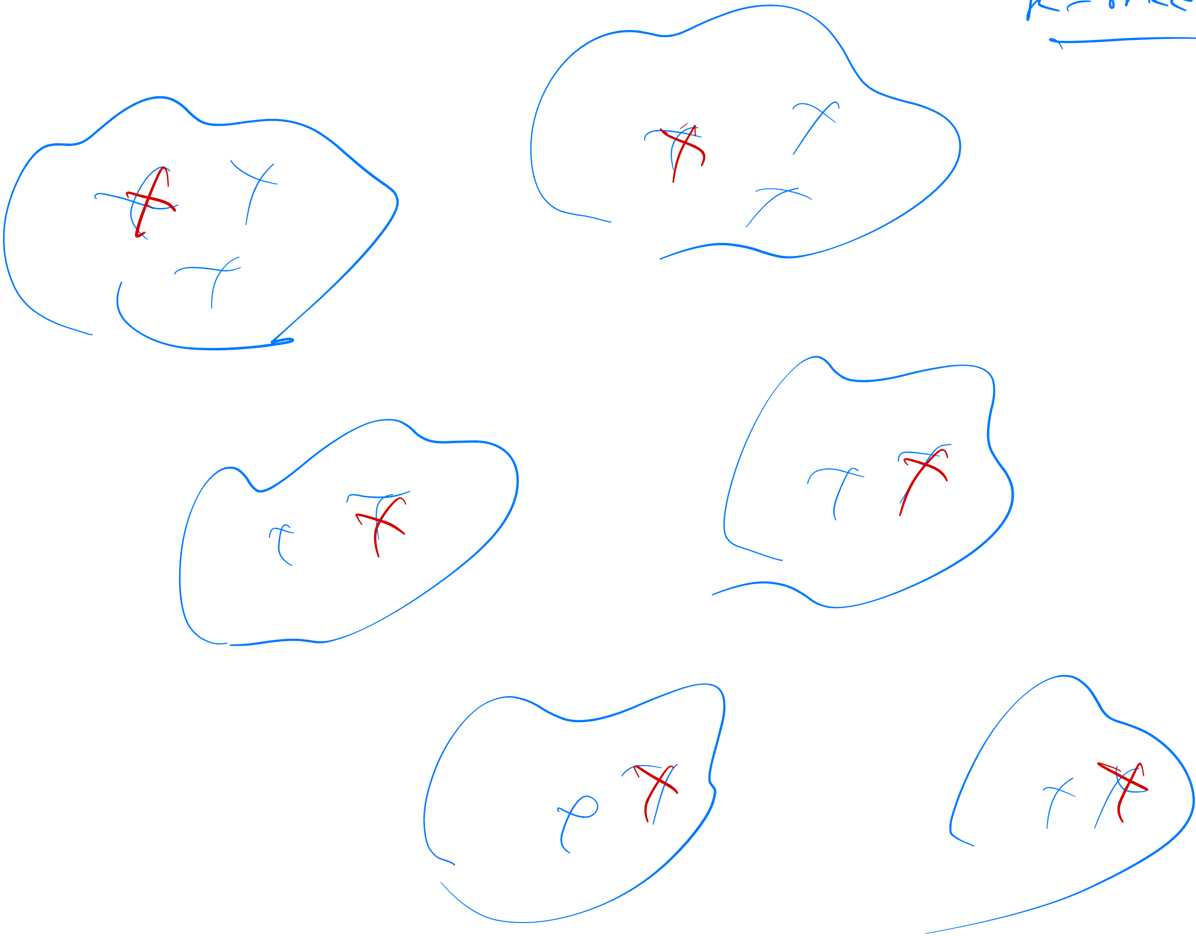


Adapted from Victor Lavrenko

K-Dimensional Trees

- By design, the constructed k-d tree is “bushy”
  - The idea is that if new points to classify are evenly distributed throughout the space, then the expected (amortized) cost of classification is approximately $O(d \log n)$ operations
- Summary
  - k-NN is fast and easy to implement
  - No training required
  - Can be good in practice (where applicable)

K-mediod



$10 = m.$

Color = $\{$red, green, blue, ... $\}$

$\rightarrow [1 \quad 0 \quad 0 \quad \ldots \quad 0]$

1-Hot Encoding.

$[0 \quad 0 \quad 1 \quad 0 \quad 0 \quad \ldots \quad 0]$