



Nearest Neighbor Methods

Nicholas Ruozzi

University of Texas at Dallas

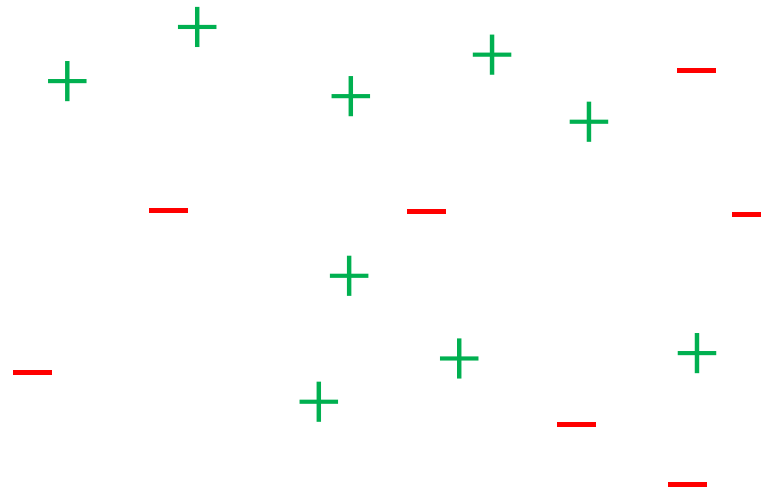
Based on the slides of Vibhav Gogate and David Sontag

Nearest Neighbor Methods

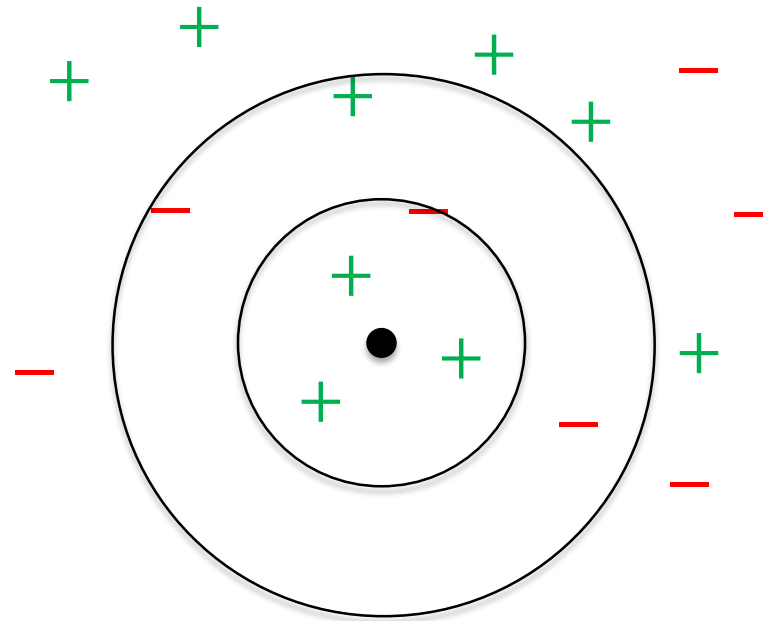


- Learning
 - Store all training examples
- Classifying a new point x'
 - Find the training example $(x^{(i)}, y^{(i)})$ such that $x^{(i)}$ is closest (for some notion of close) to x'
 - Classify x' with the label $y^{(i)}$

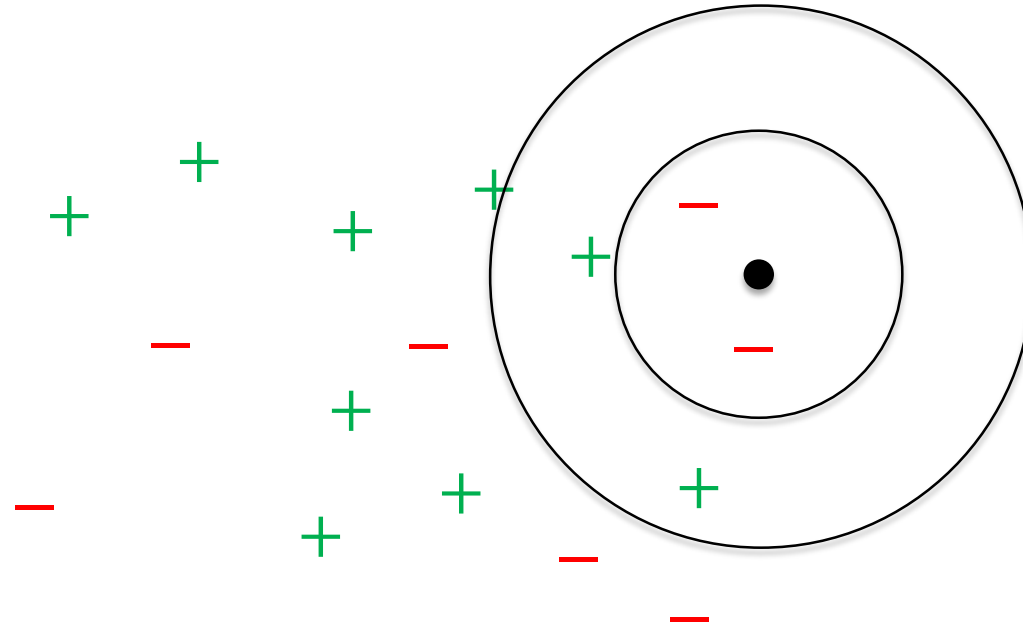
Nearest Neighbor Methods



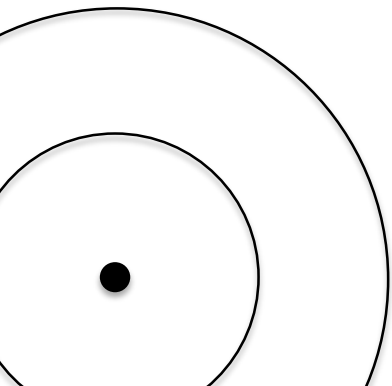
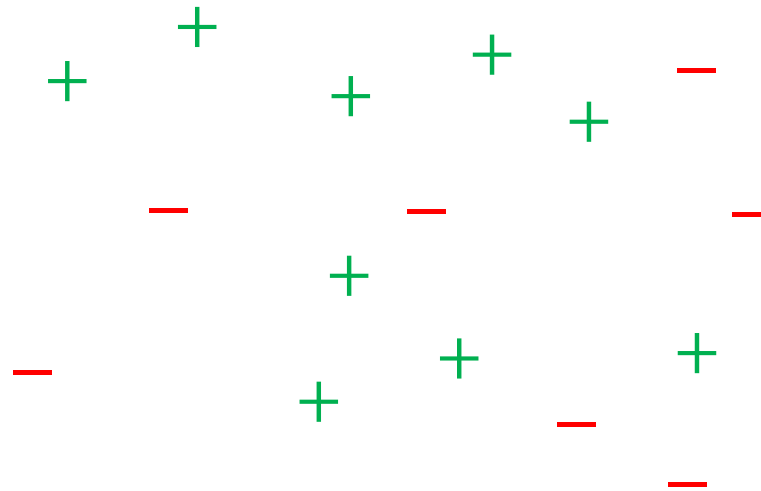
Nearest Neighbor Methods



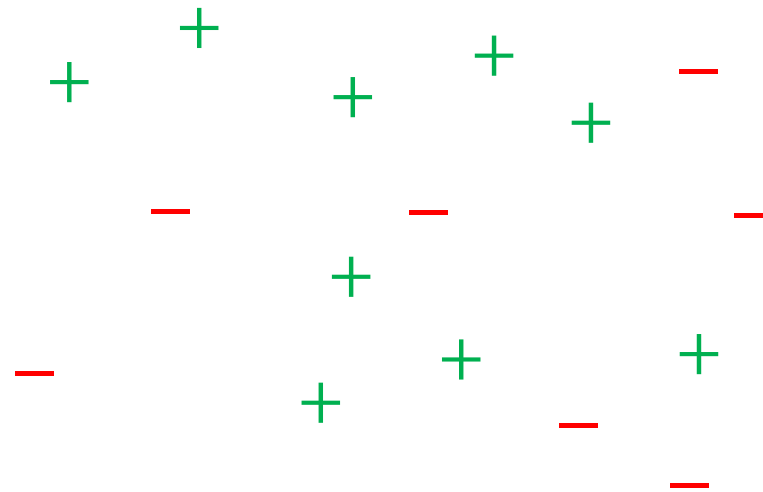
Nearest Neighbor Methods



Nearest Neighbor Methods

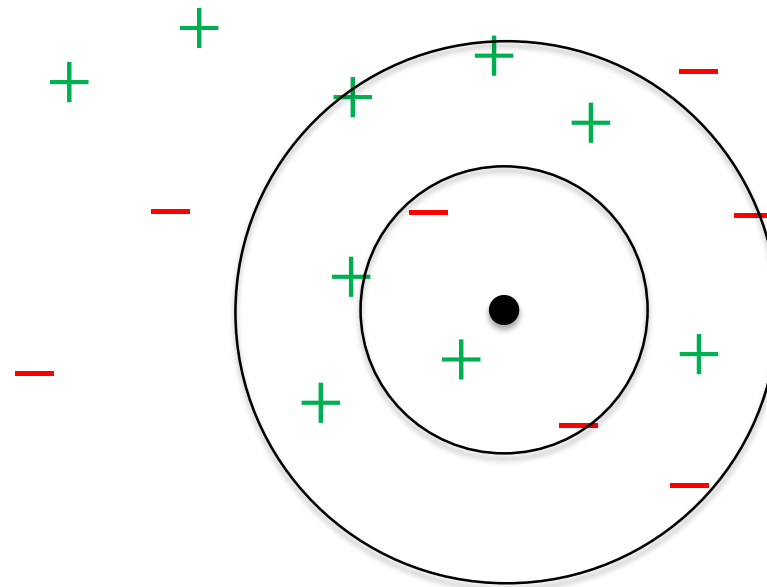


Nearest Neighbor Methods



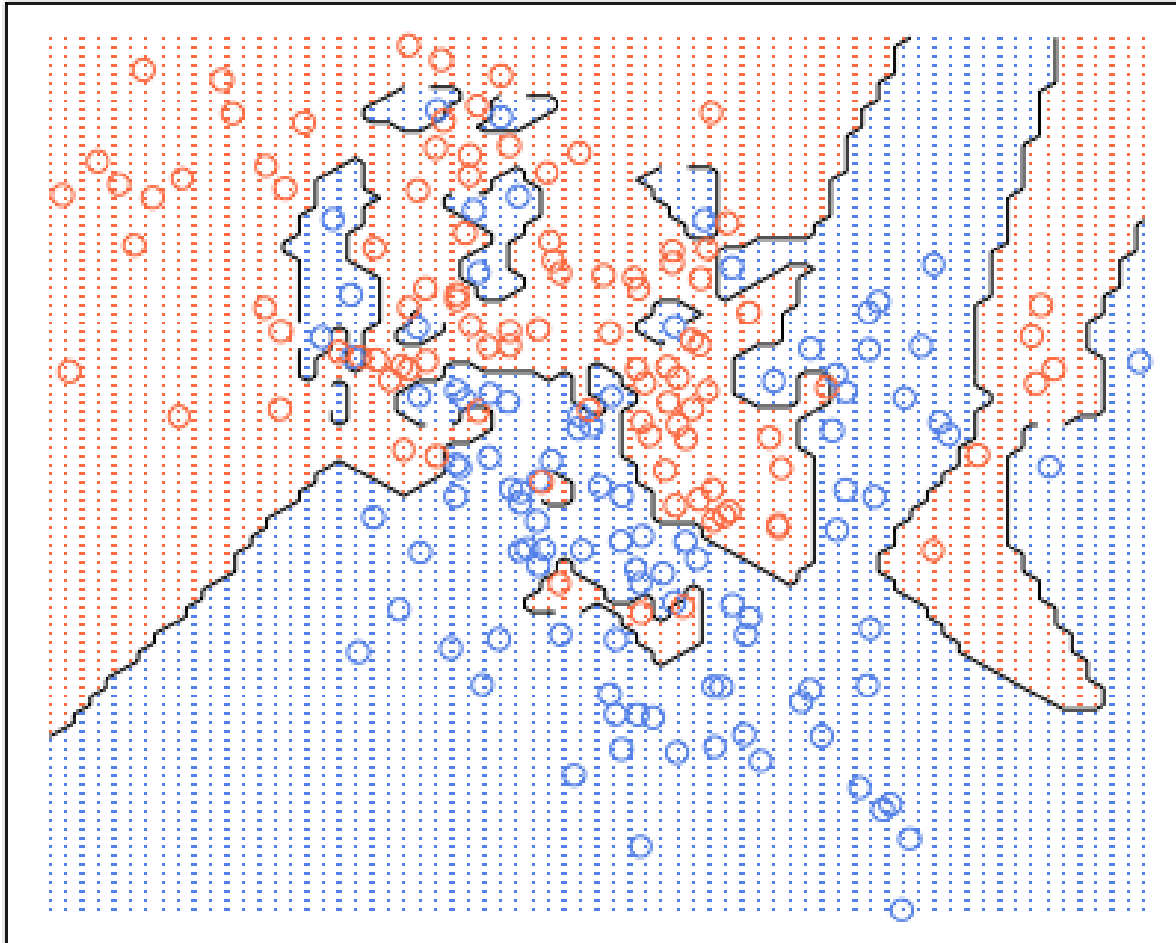
k -nearest neighbor methods look at the k closest points in the training set and take a majority vote
(should choose k to be odd)

Nearest Neighbor Methods

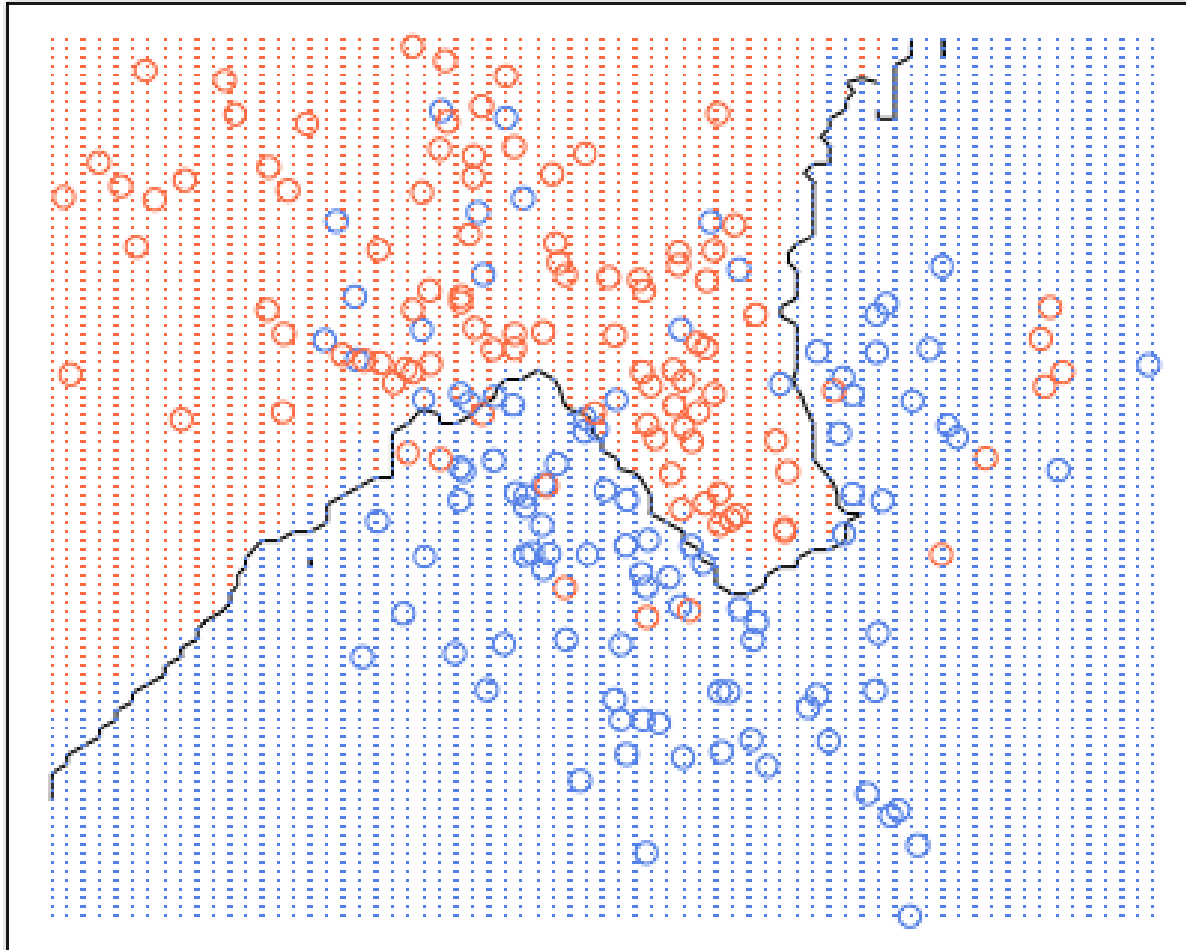


k -nearest neighbor methods look at the k closest points in the training set and take a majority vote (should choose k to be odd)

1-NN Example



20-NN Example



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
 - Can learn complicated decision boundaries
- Disadvantages
 - Classification is slow (need to keep the entire training set around)
 - 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 ?
 - Too small and the estimates are noisy, too large and the accuracy suffers
- What if the nearest neighbor is really far away?

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

- Sample mean

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

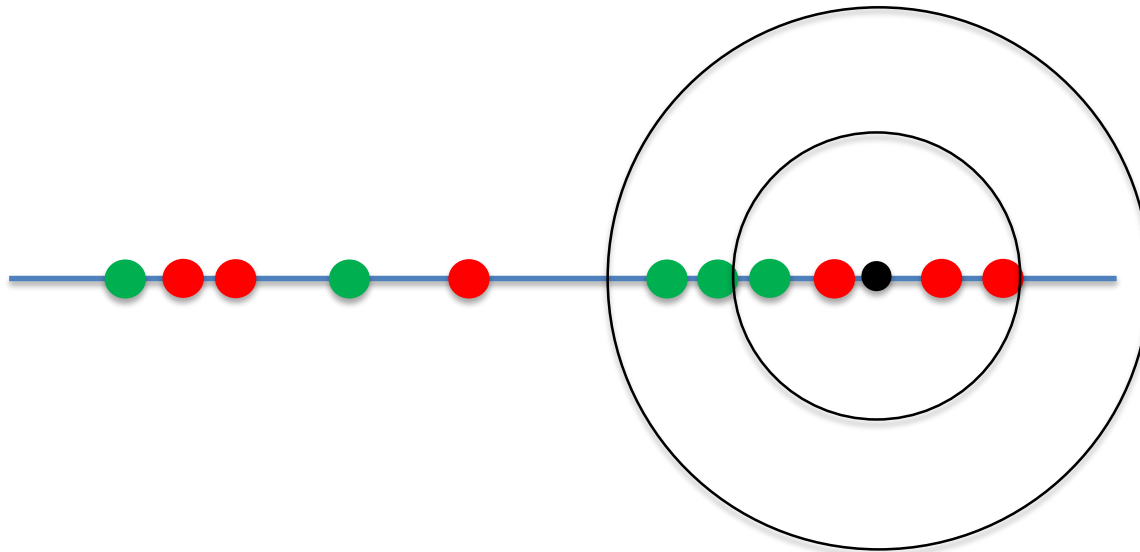
- Sample variance (biased)

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

Irrelevant Attributes



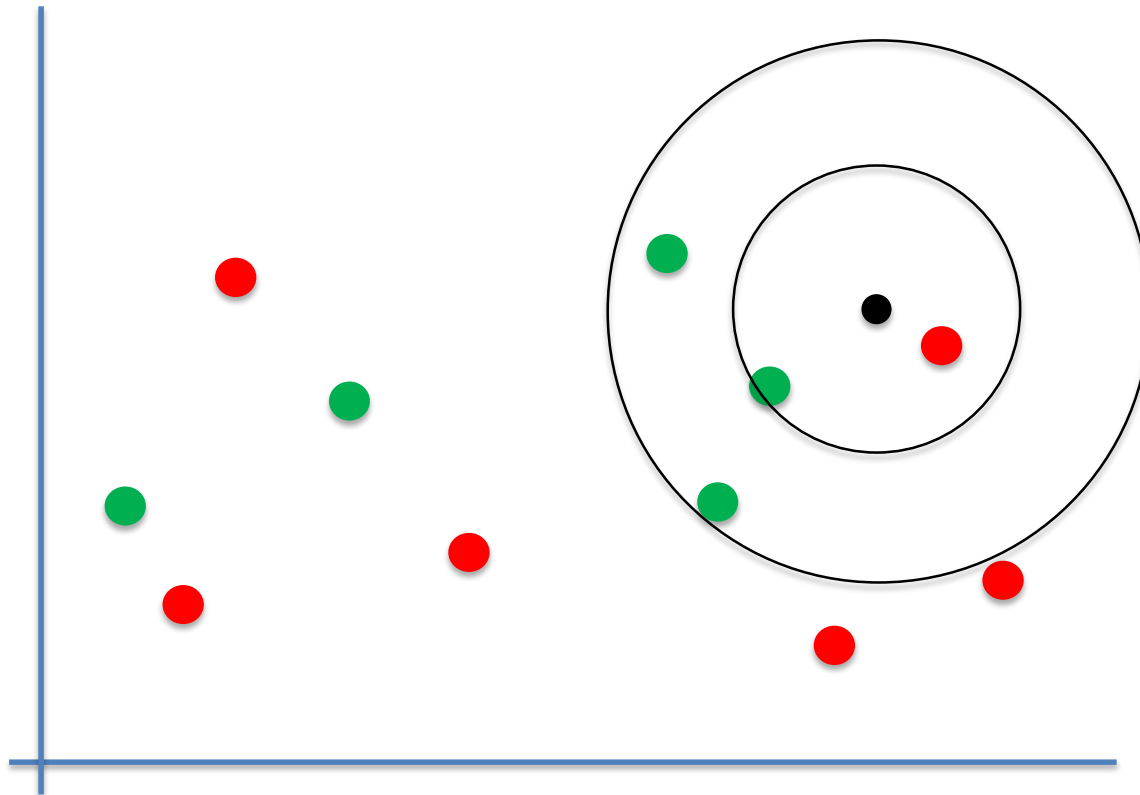
Consider the nearest neighbor problem in one dimension



Irrelevant Attributes



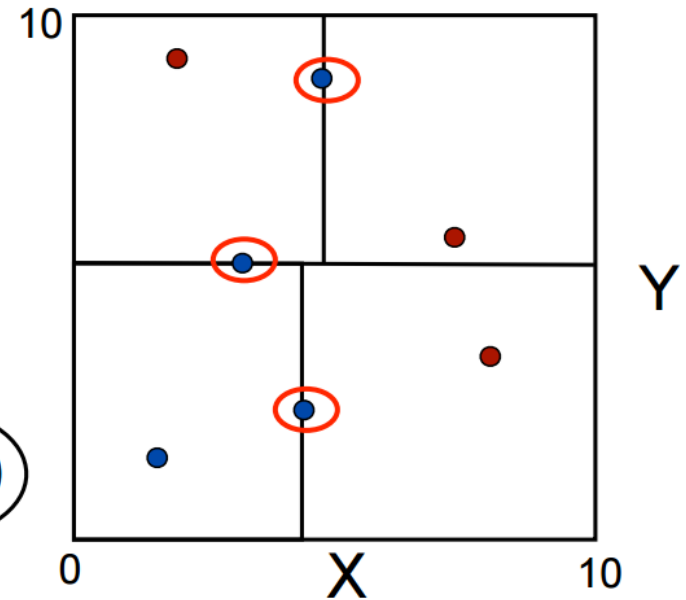
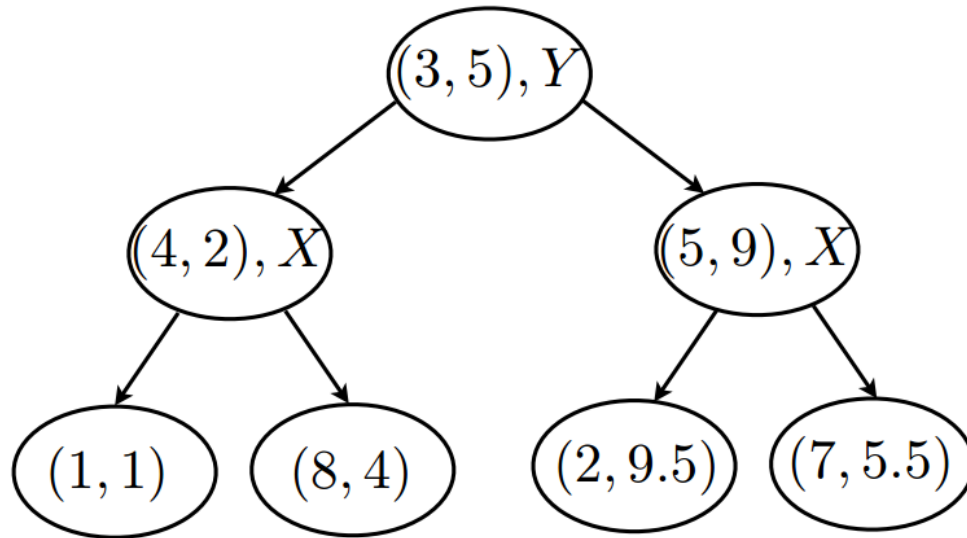
Now, add a new attribute that is just random noise...



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

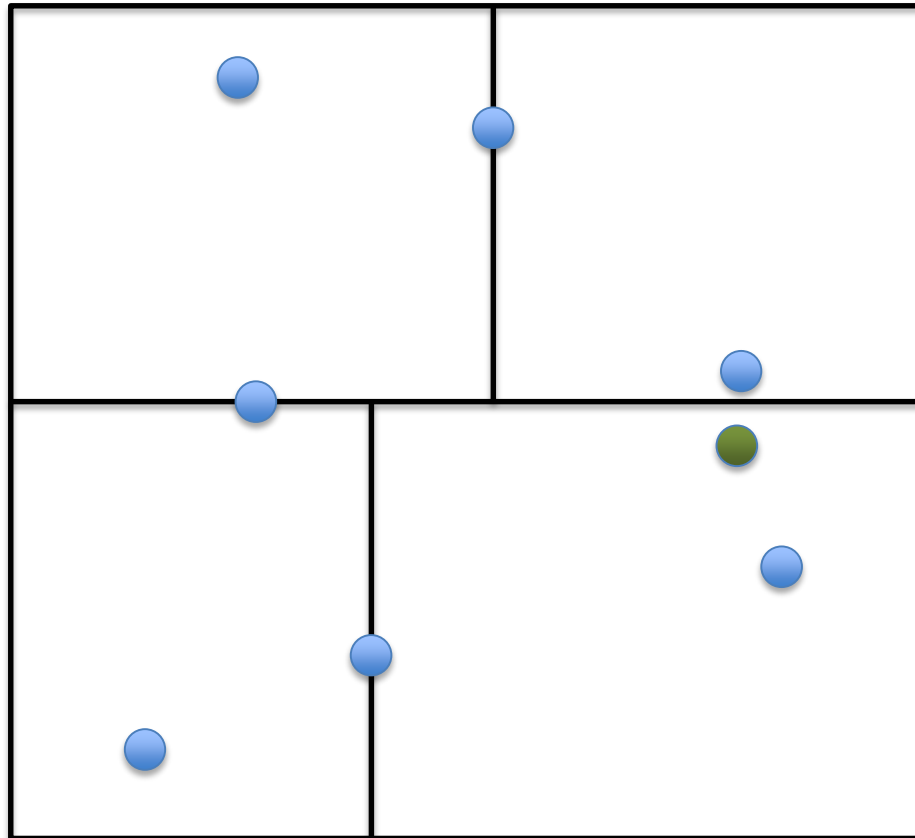
- k-d trees provide a data structure that can help simplify the classification task by constructing a tree that partitions the search space
 - Starting with the entire training set, choose some dimension, i
 - Select an element of the training data whose i^{th} dimension has the median value among all 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

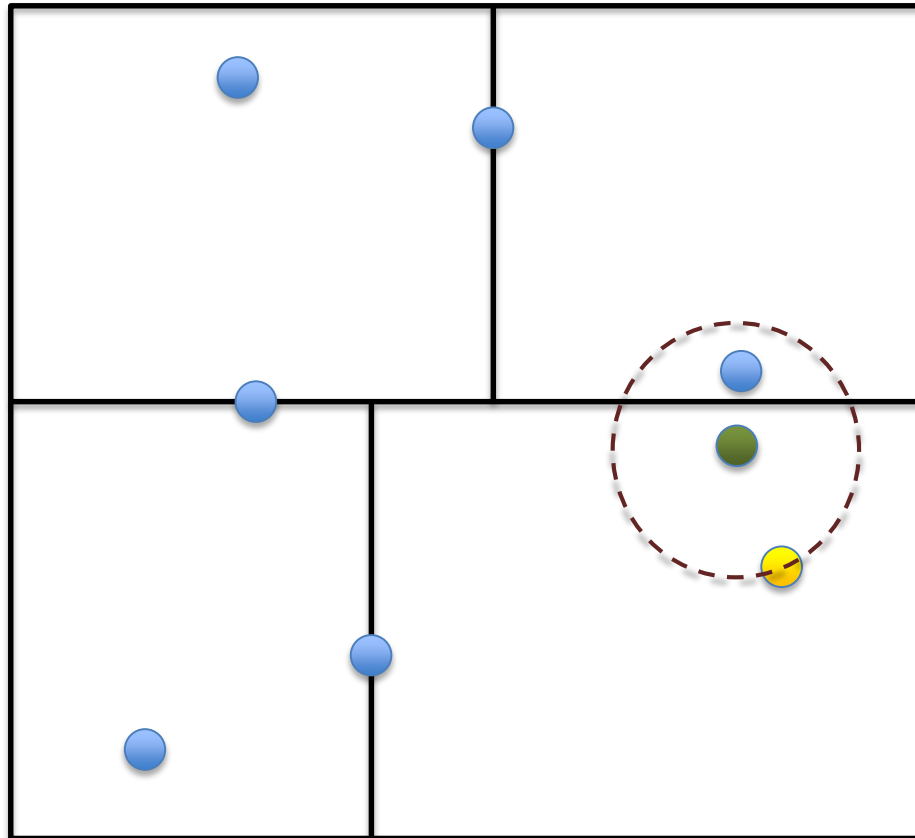


- Start at the top of the k-d tree and traverse it to a leaf of the tree based on where the point to classify should fall
- Once a leaf node is reached, it is selected to be the current closest point to x'
- Follow the path, in the opposite direction, from the leaf to the root
 - If the current node along the path is closer to x' than the selected closest point it becomes the new closest point
 - Before moving up the tree, the algorithm checks if there **could** be any points in the opposite partition that are closer to x' than the current closest point
 - If so, then closest point in that subtree is computed recursively
 - Otherwise, the parent of the current node along the path becomes the new current node

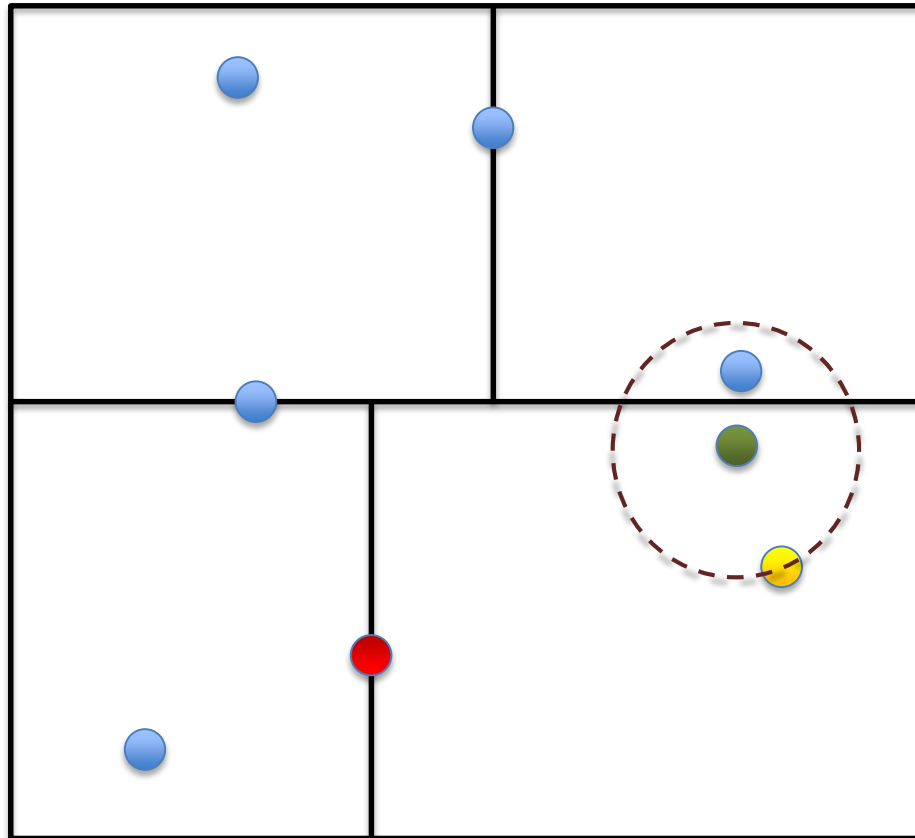
K-Dimensional Trees



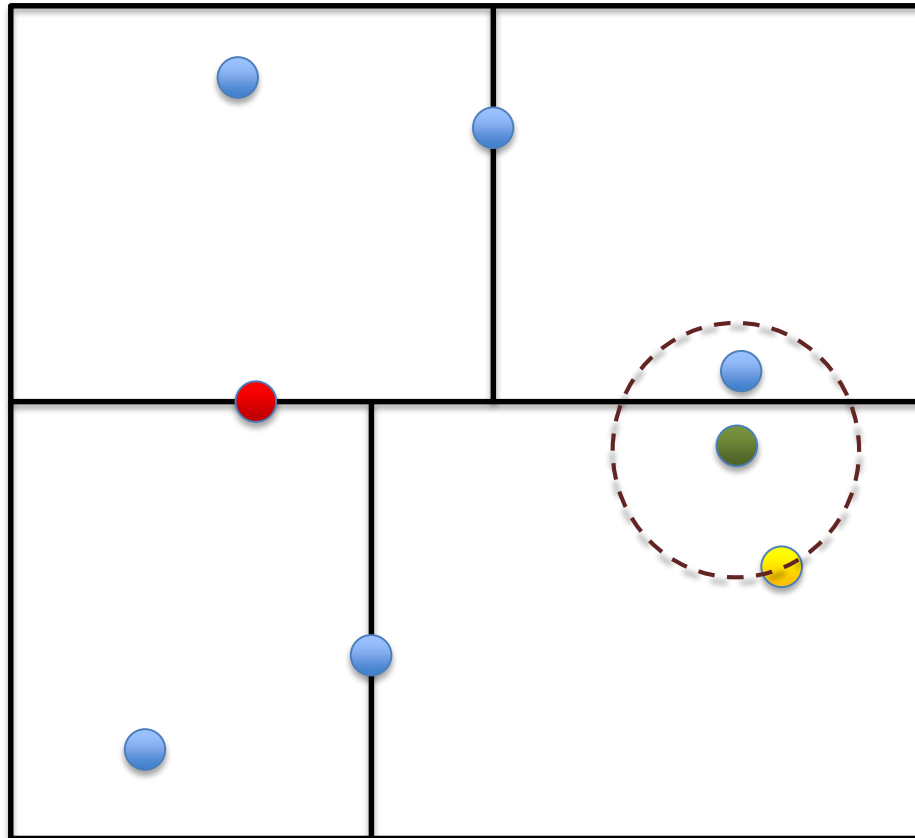
K-Dimensional Trees



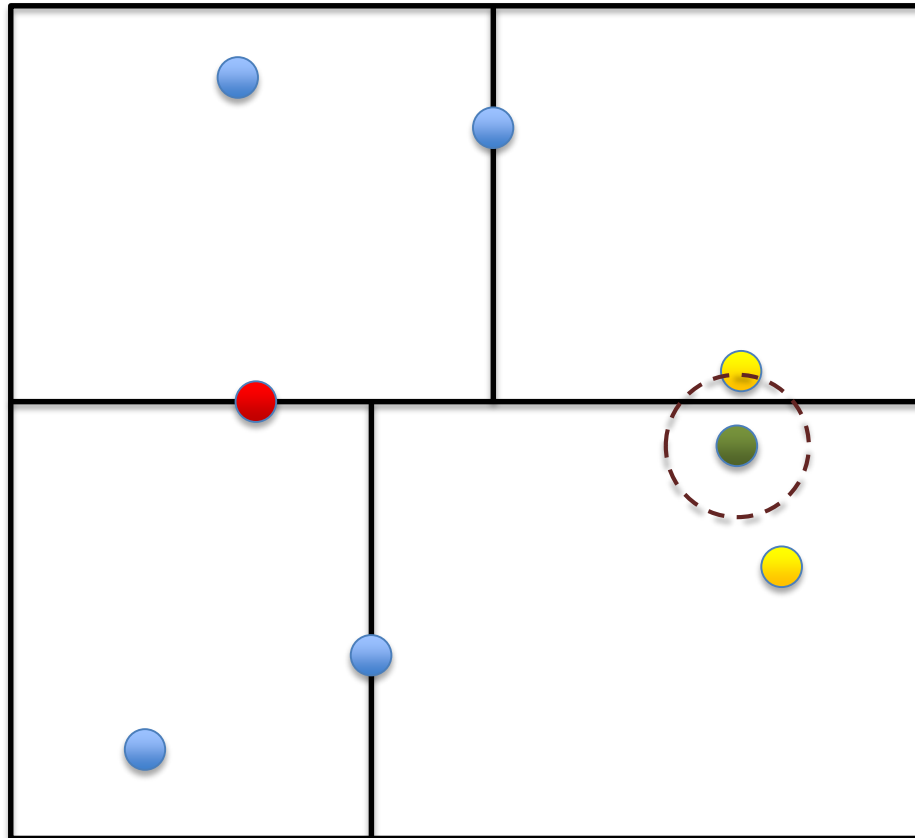
K-Dimensional Trees



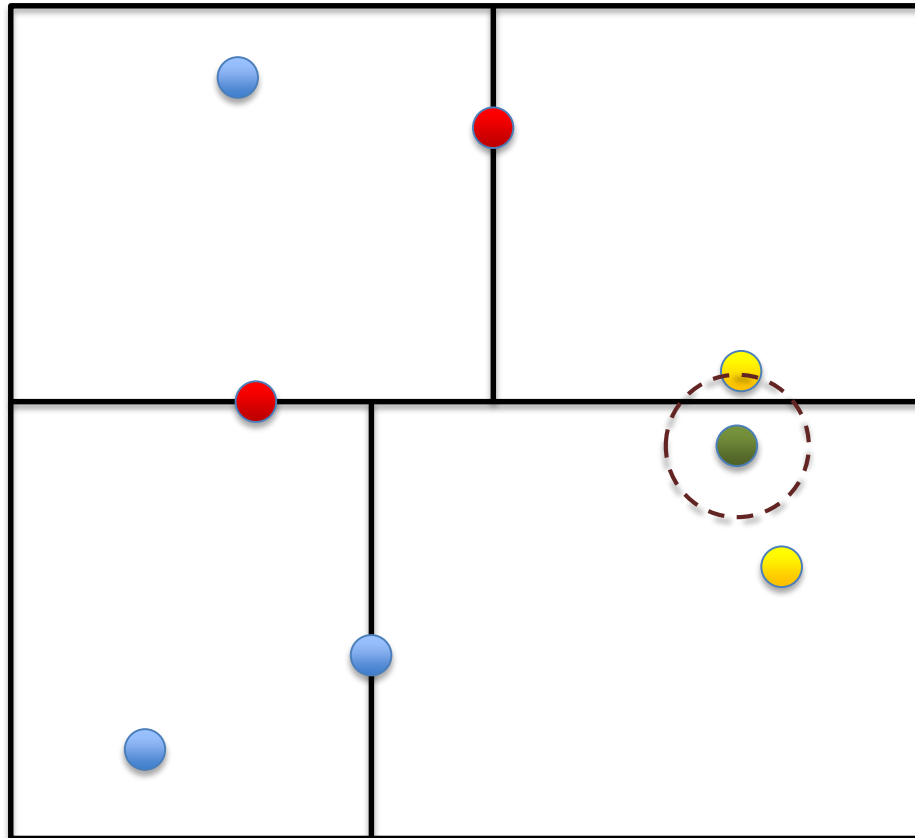
K-Dimensional Trees



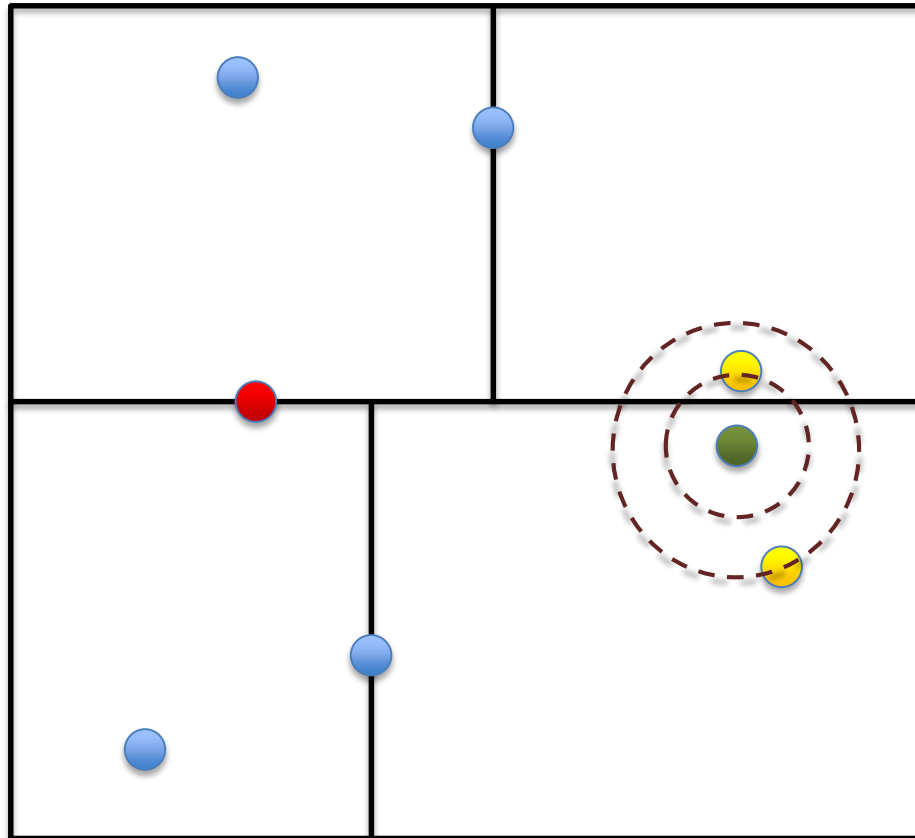
K-Dimensional Trees



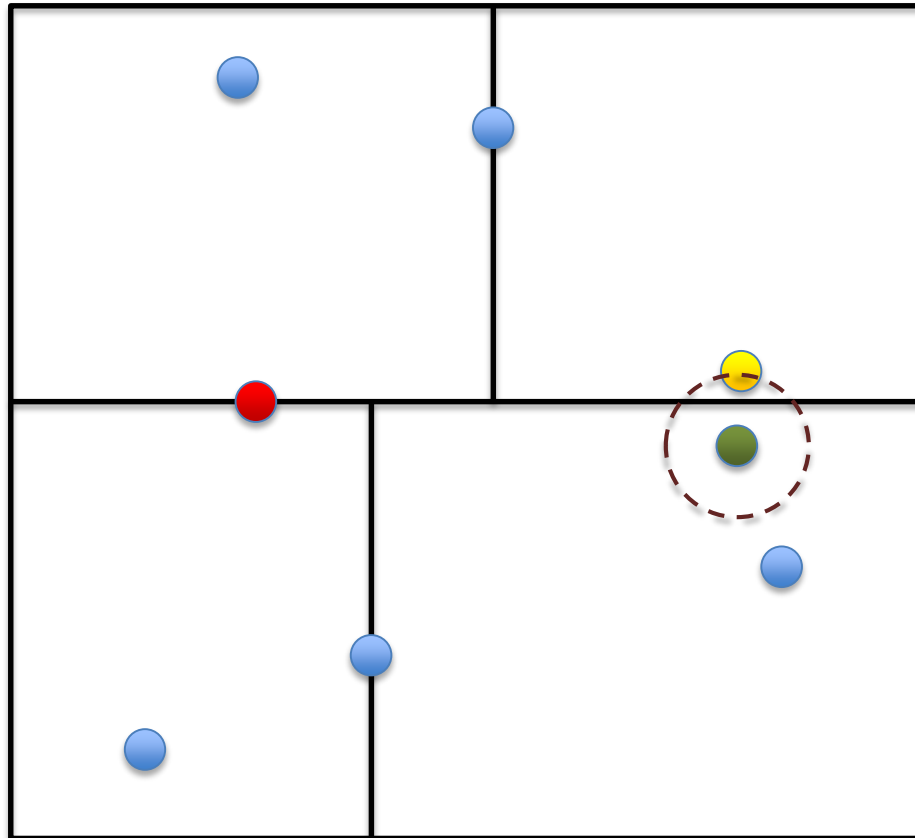
K-Dimensional Trees



K-Dimensional Trees



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)