CS 4375
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

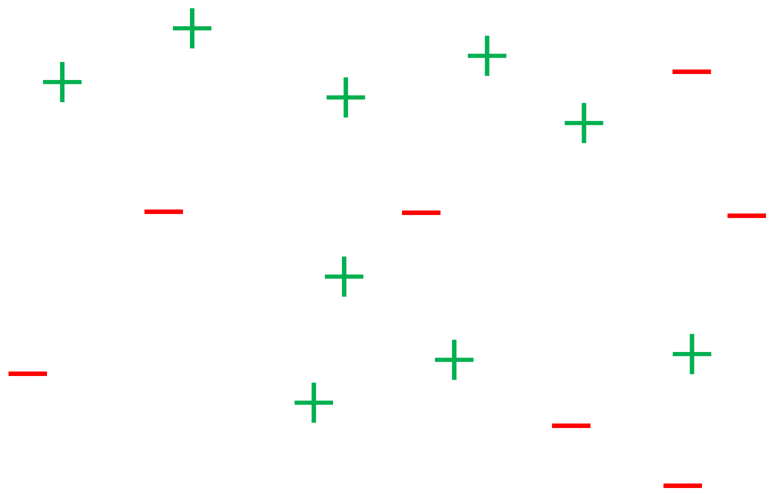
Nearest Neighbor Methods

- Learning
  - Store all training examples
- 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 $y^{(i)}$

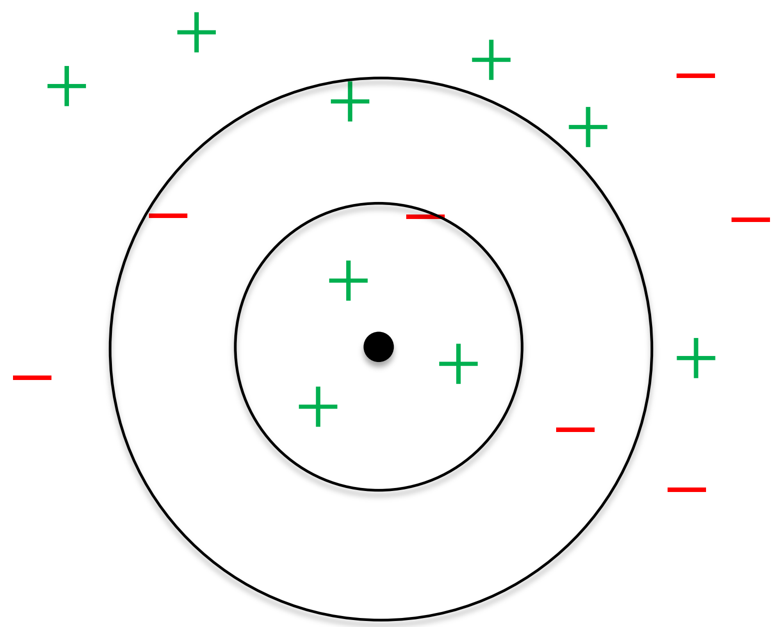
Linear Models (Perceptrons, linear SVMs)

$y_{pred} = \text{Sign}(w^T x' + b) \rightarrow$ Binary Classifn

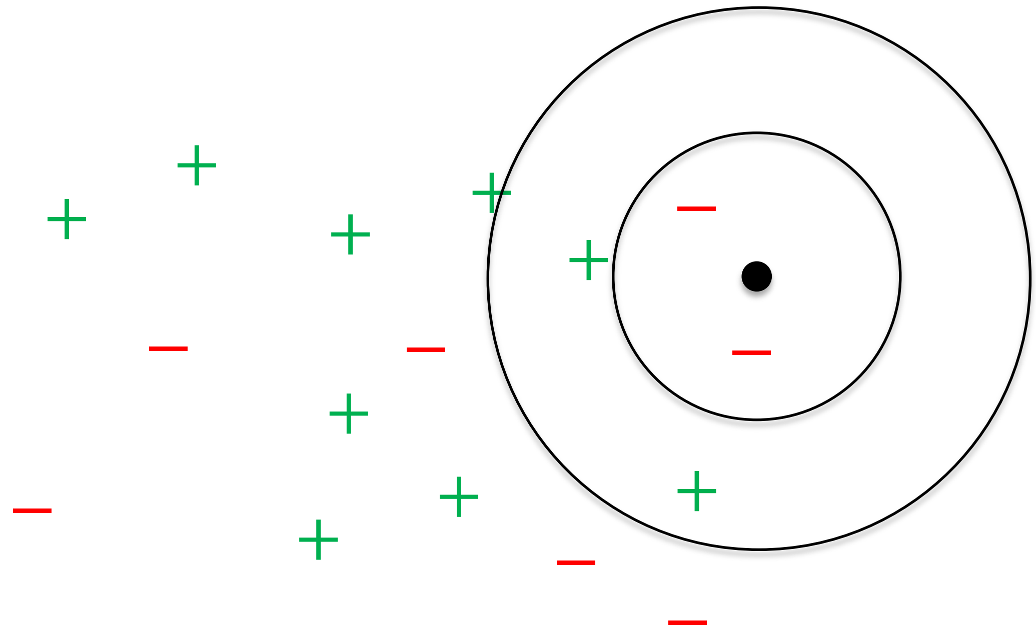
Nearest Neighbor Methods



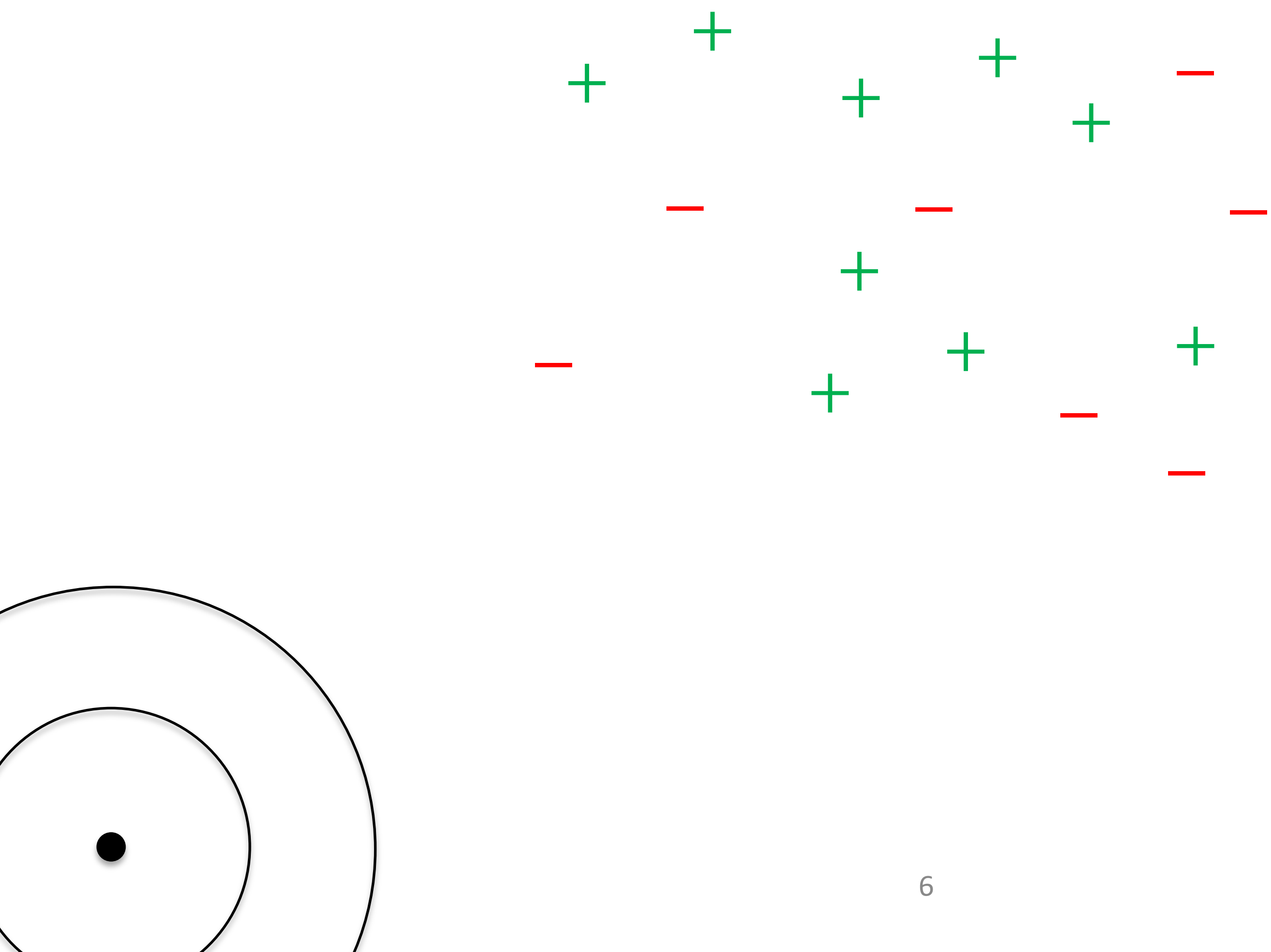
Nearest Neighbor Methods



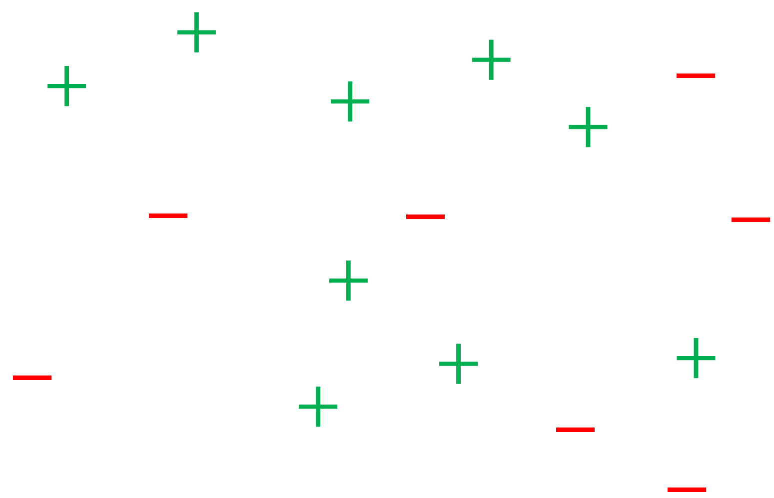
Nearest Neighbor Methods



Nearest Neighbor Methods

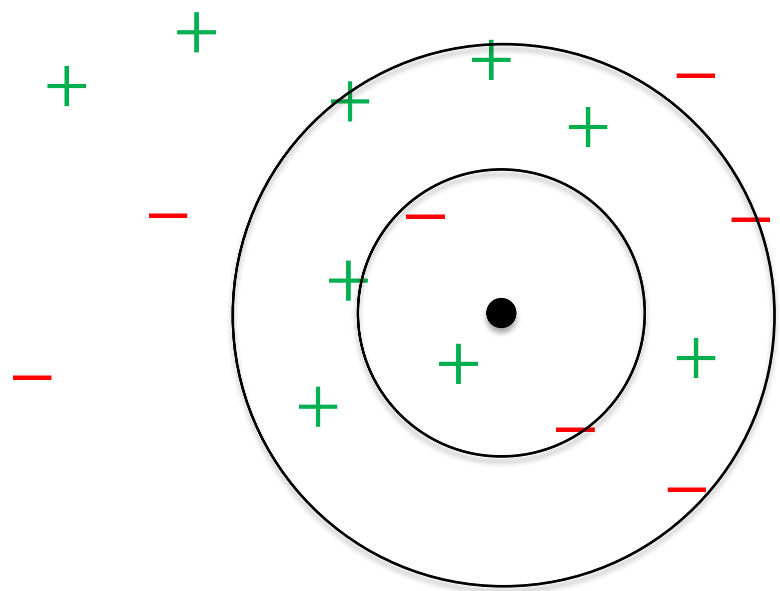


Nearest Neighbor Methods



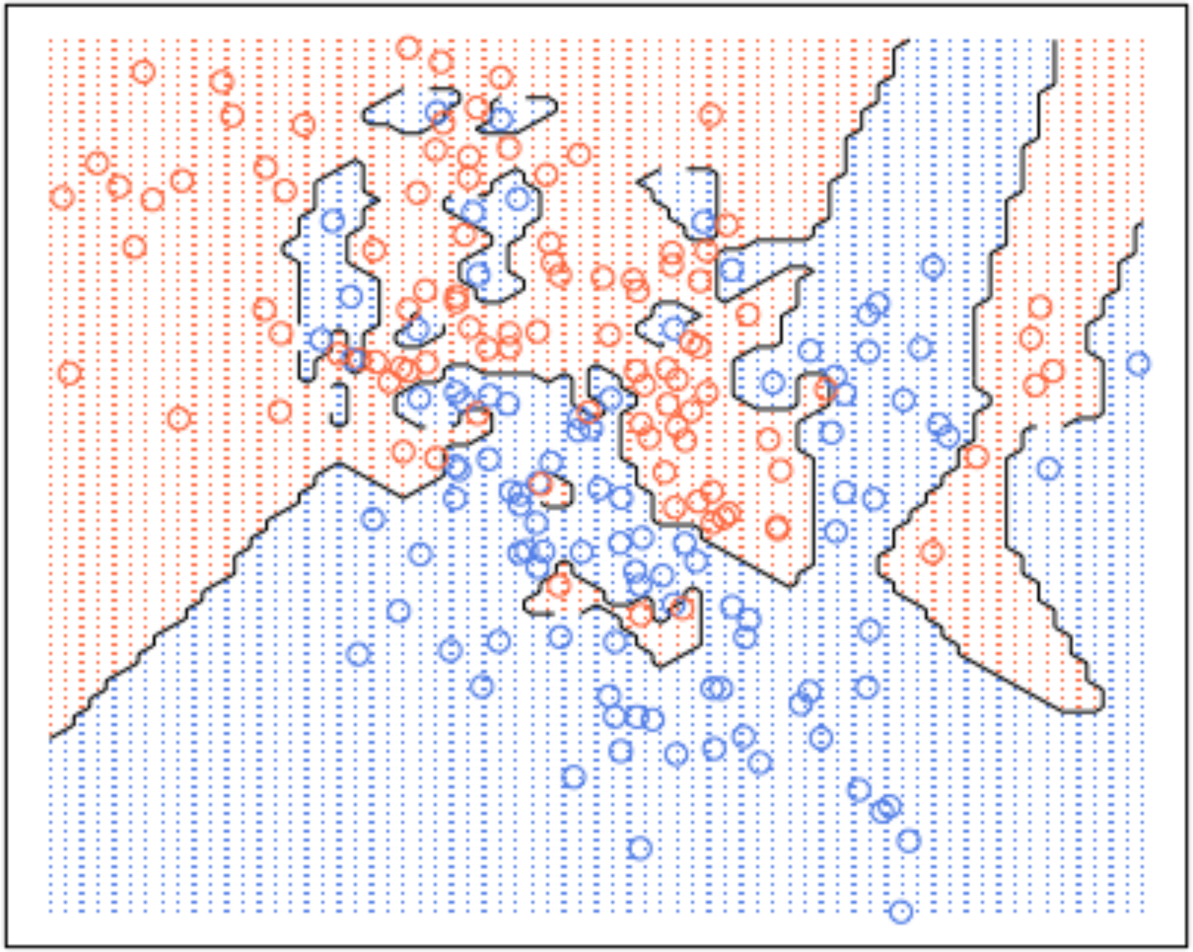
k-nearest neighbor methods look at the k closest points in the training set and take a majority vote (should choose k to be odd)

Nearest Neighbor Methods



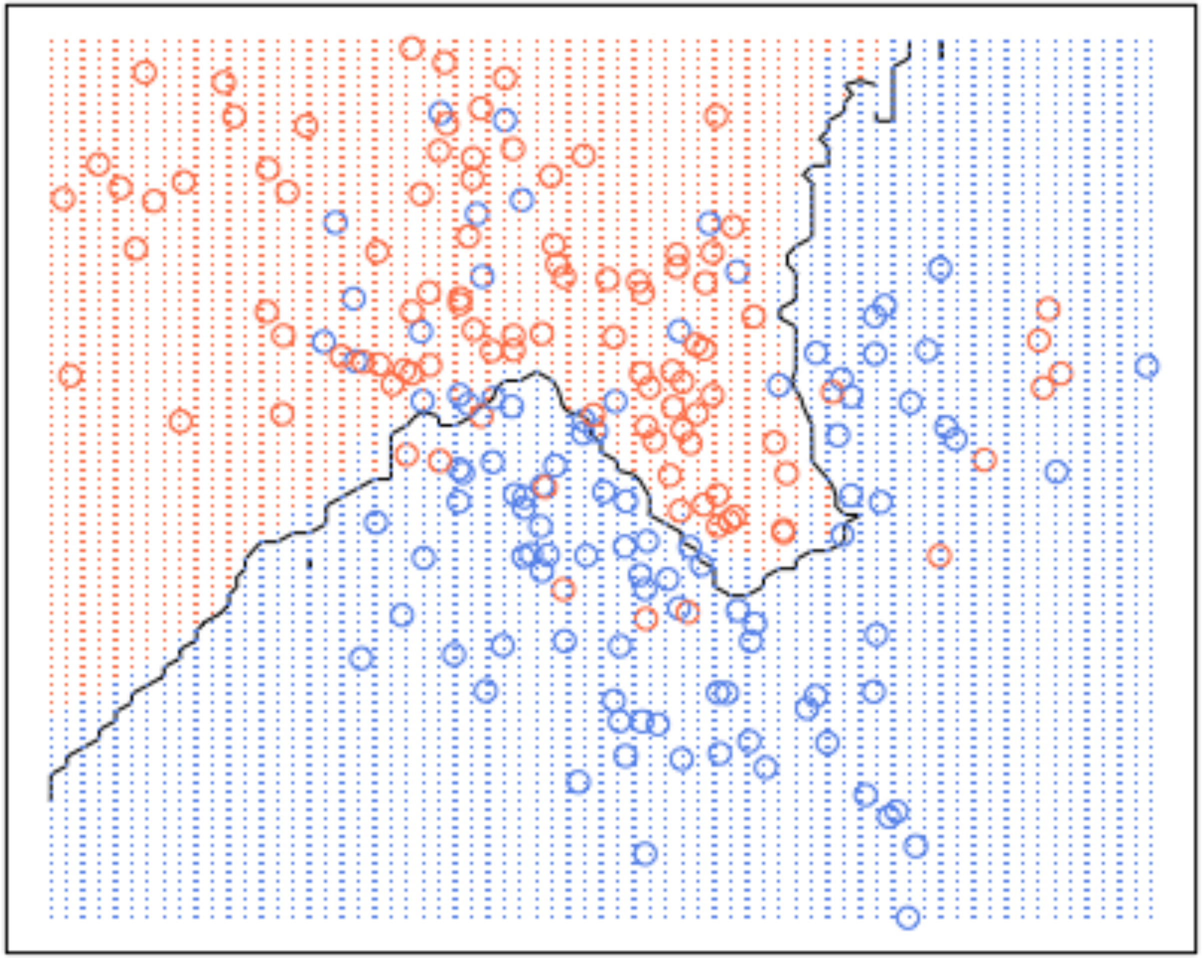
k-nearest neighbor methods look at the k closest points in the training set and take a majority vote (should choose k to be odd)

1-NN Example



[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
  - 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?

$$\begin{bmatrix} x_1^i \end{bmatrix}_{x_1} \quad \begin{bmatrix} x_2^i \end{bmatrix}_{x_2}$$

$$\longrightarrow \sqrt{\sum_{i=1}^{d} \left[x_1^i - x_2^i\right]^2} \rightarrow L2 \text{ Distance. } \|x_1 - x_2\|_2$$

$$\longrightarrow \sum_{i=1}^{d} |x_1^i - x_2^i| \rightarrow L1 \text{ Distance. } \|x_1 - x_2\|_1$$

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

Normalization

$$x_k^{norm} = \frac{x_k - \bar{x}_k}{\hat{\sigma}_k}$$

- Sample mean

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

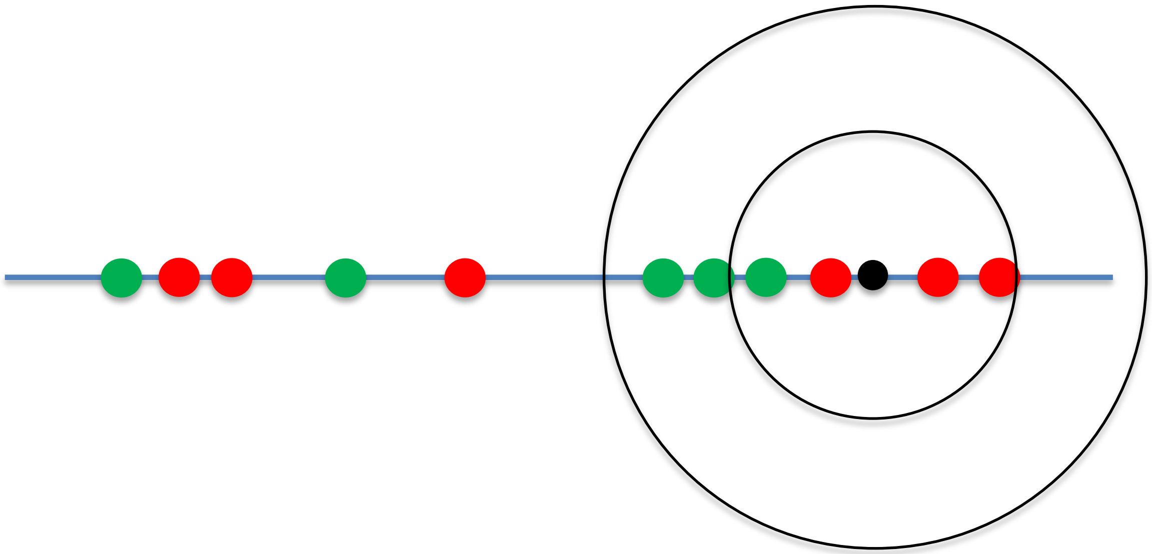
$$E[X_k^{norm}] = E[X_k] - \bar{X}_k$$
$$= 0$$

- Sample variance (biased)

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

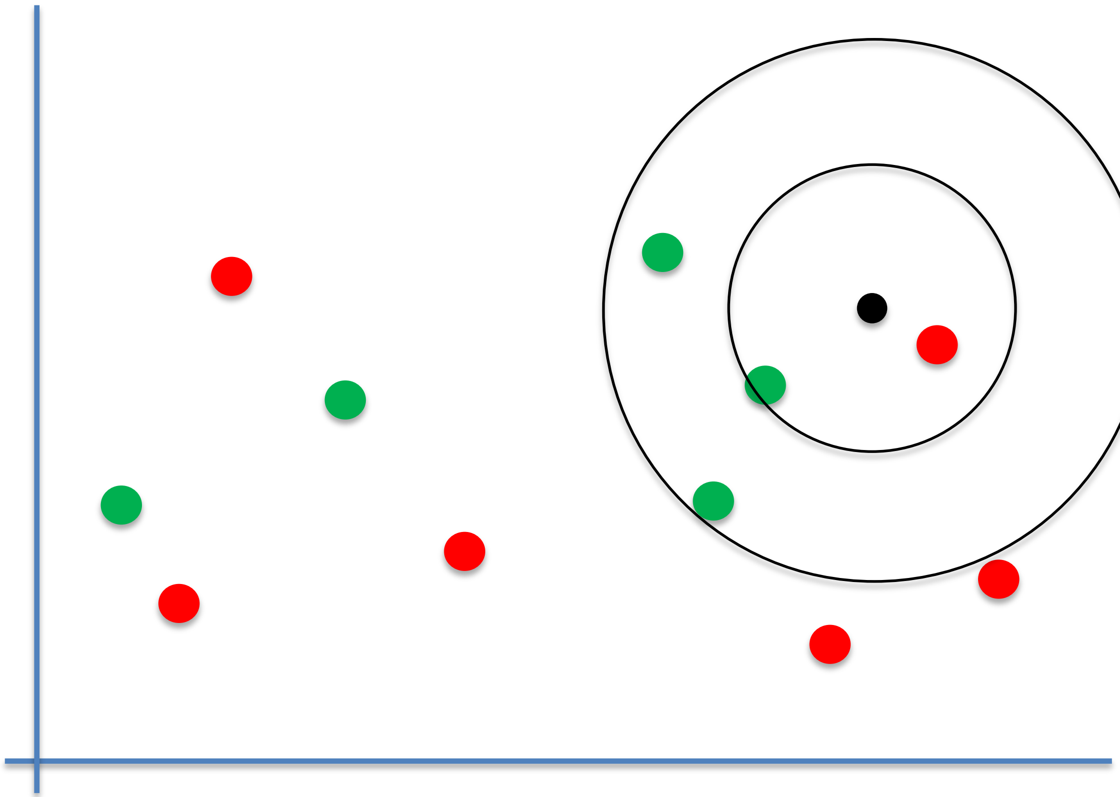
Irrelevant Attributes

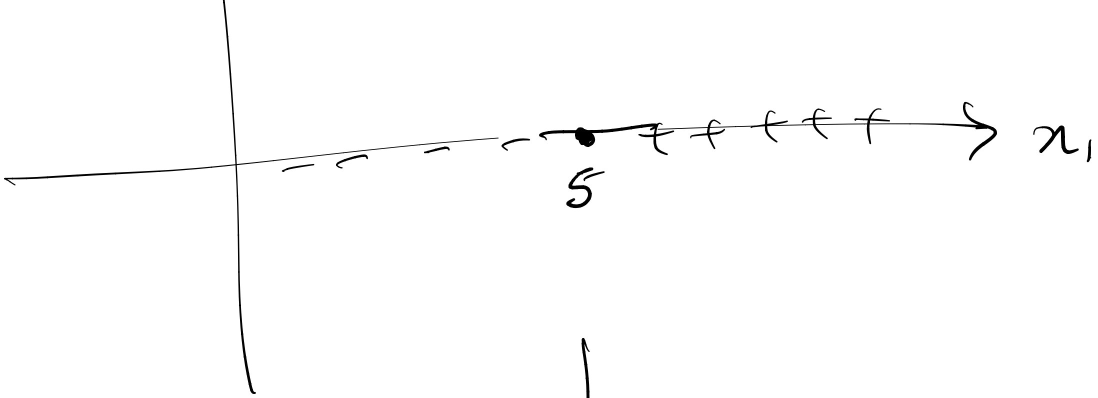
Consider the nearest neighbor problem in one dimension



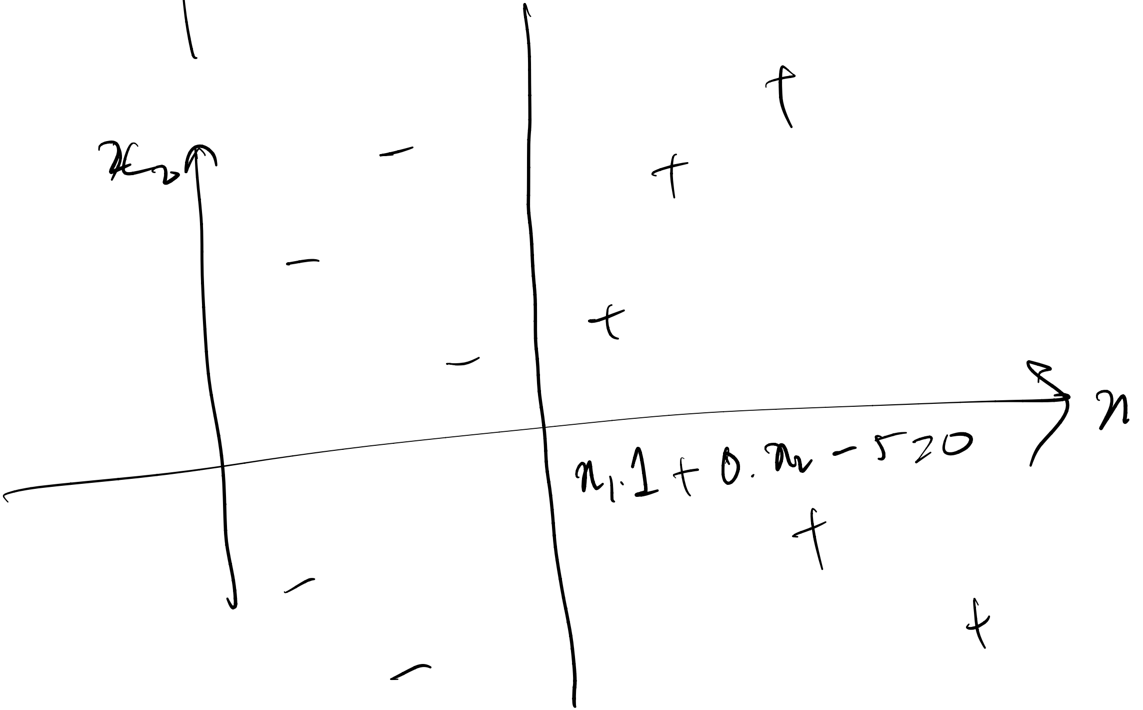
Irrelevant Attributes

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





x_1
5



x_2
x_1
$x_1 \cdot 1 + 0 \cdot x_2 - 5 \geq 0$

K-Dimensional Trees

- 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 ← O(n) × O(m) → Single test Ex.
    # Feat # points
  - It is possible to do better if we do some preprocessing on the training data

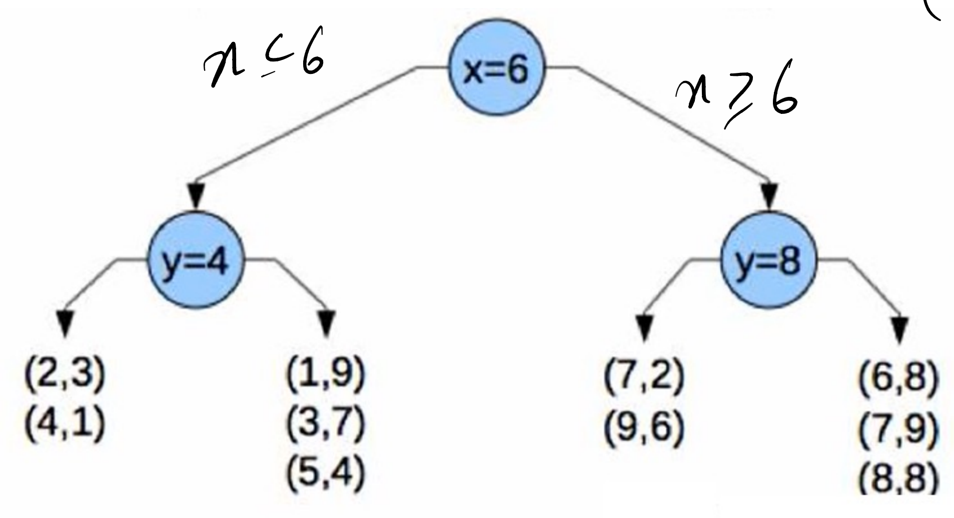
K-Dimensional Trees

- 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

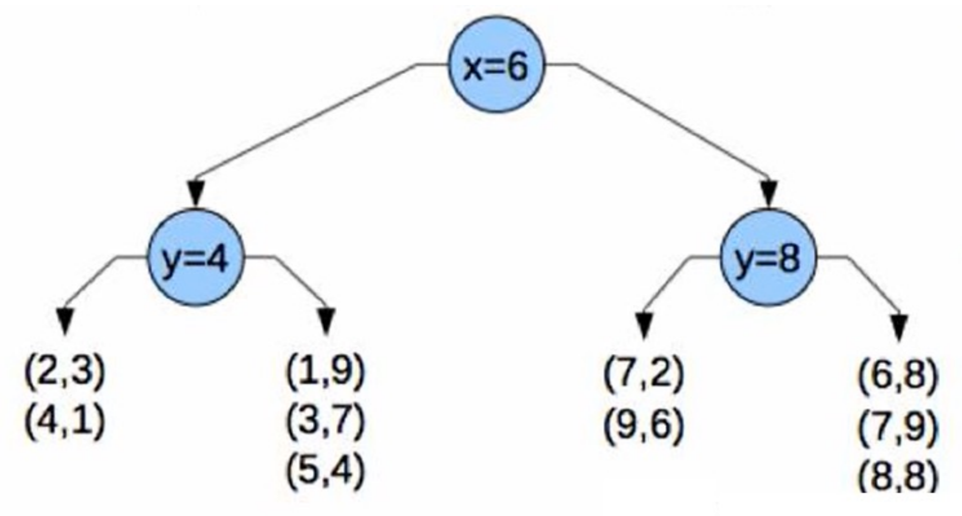
(x,y)

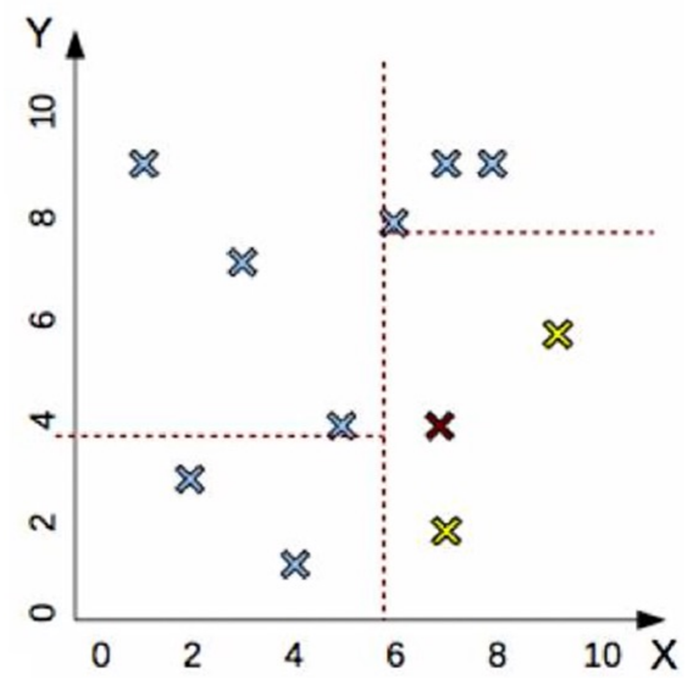


Adapted from Victor Lavrenko

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

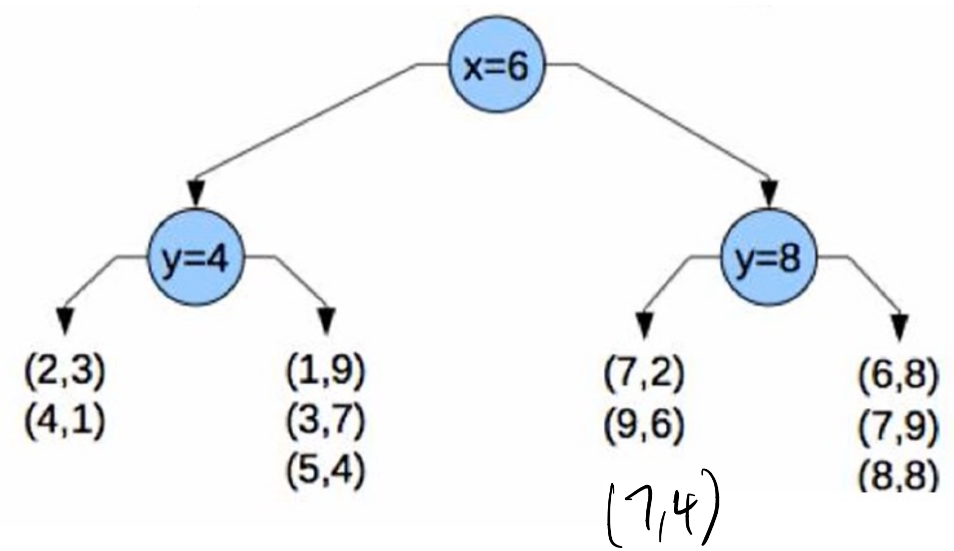


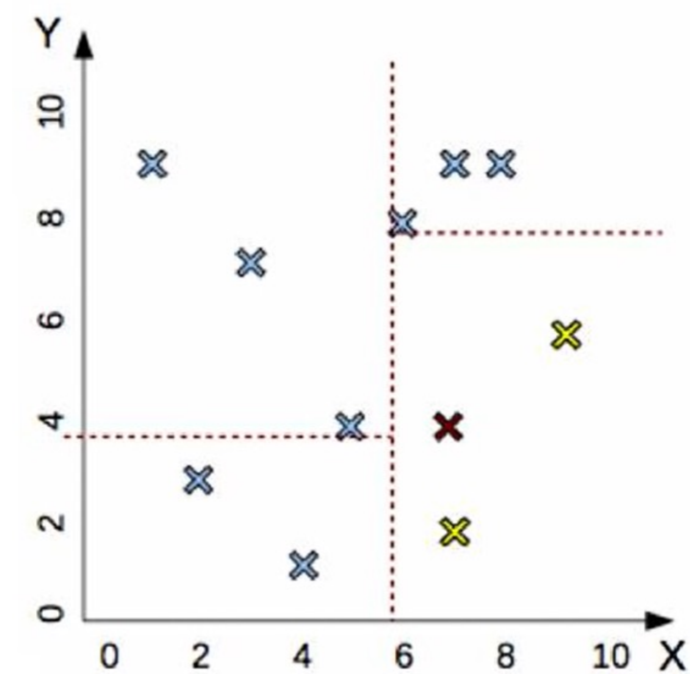


Adapted from Victor Lavrenko

K-Dimensional Trees: Inference

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





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

Dim # Train Examples

- Summary
  - k-NN is fast and easy to implement
  - No training required
  - Can be good in practice (where applicable)