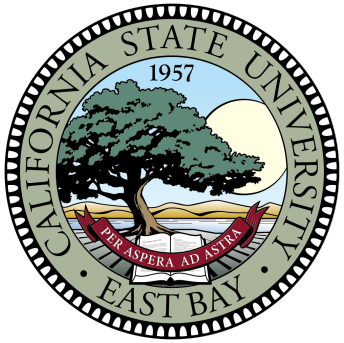


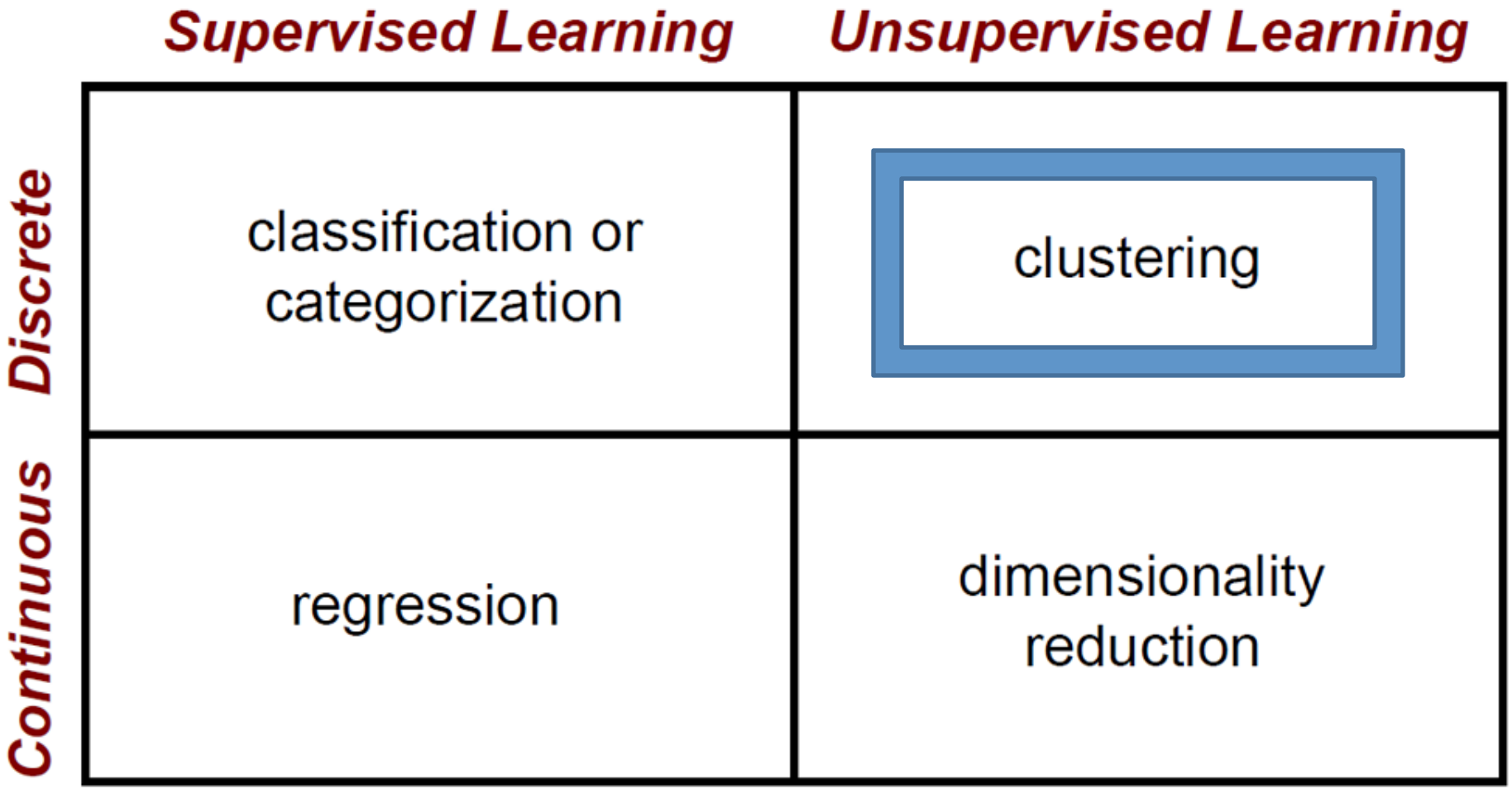
CS 6820 – Machine Learning

Lecture 13

Instructor: Eric S. Gayles, PhD.

Feb 21, 2018

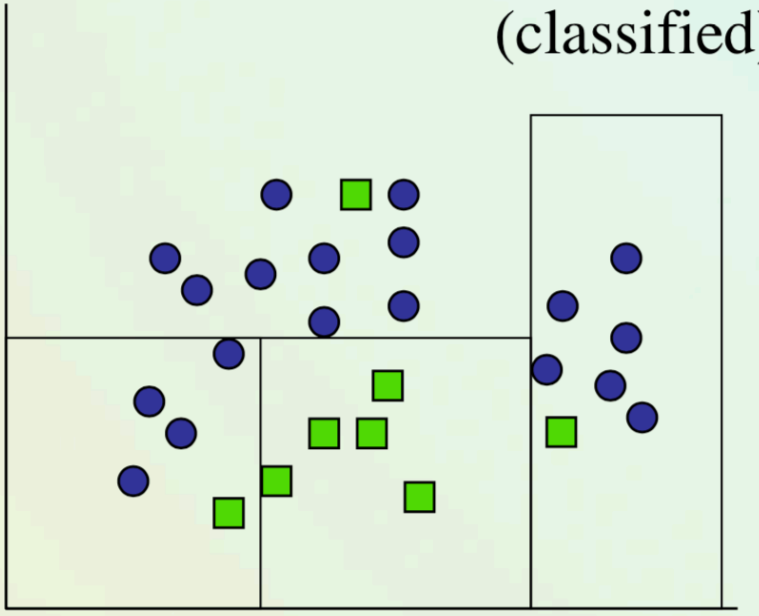
Machine Learning Problems



Classification vs. Clustering

Classification: Supervised learning:

Learns a method for predicting the instance class from pre-labeled (classified) instances

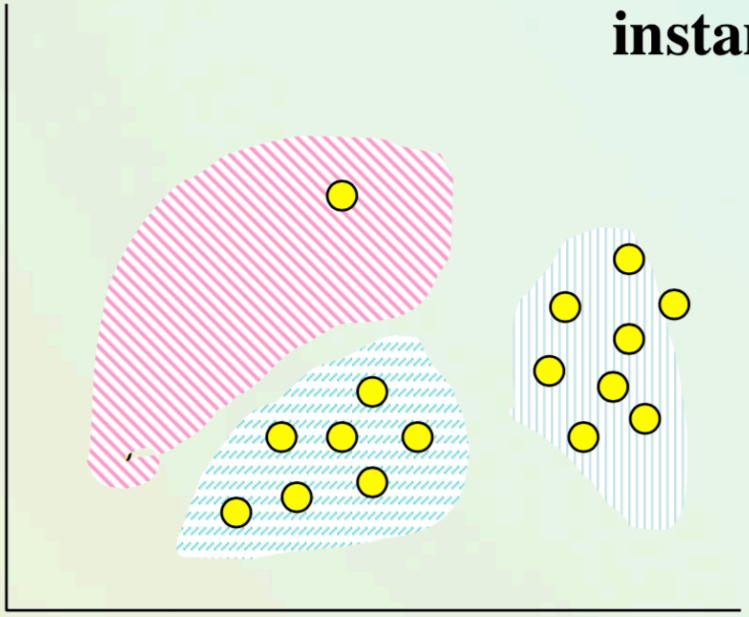


Clustering

- Cluster analysis divides data into groups (clusters) that are meaningful, useful, or both.

- If meaningful groups are the goal, then the clusters should capture the natural structure of the data.

Clustering

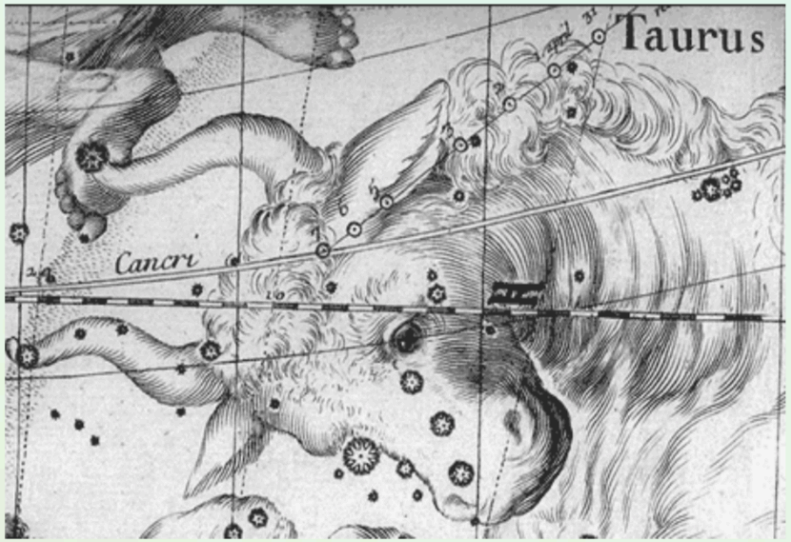


Unsupervised learning:

Finds "natural" grouping of instances given un-labeled data

Cluster Analysis

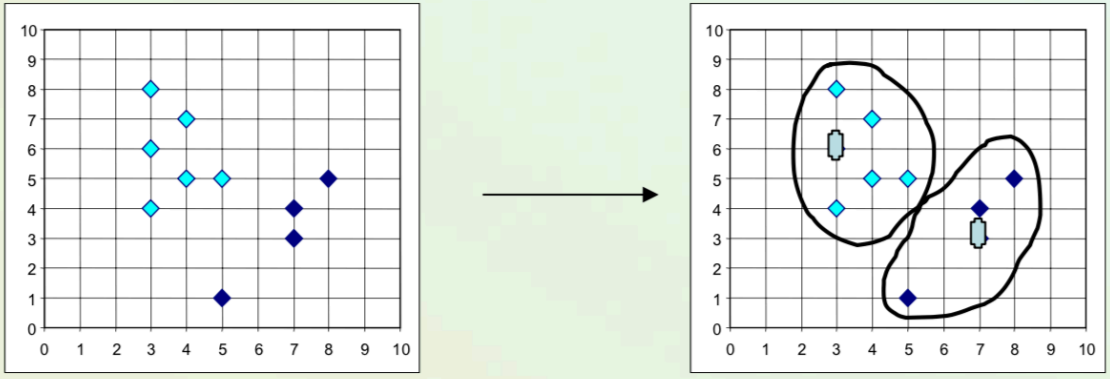
Astronomy - aggregation of stars, galaxies, or super galaxies, …



Problem Statement

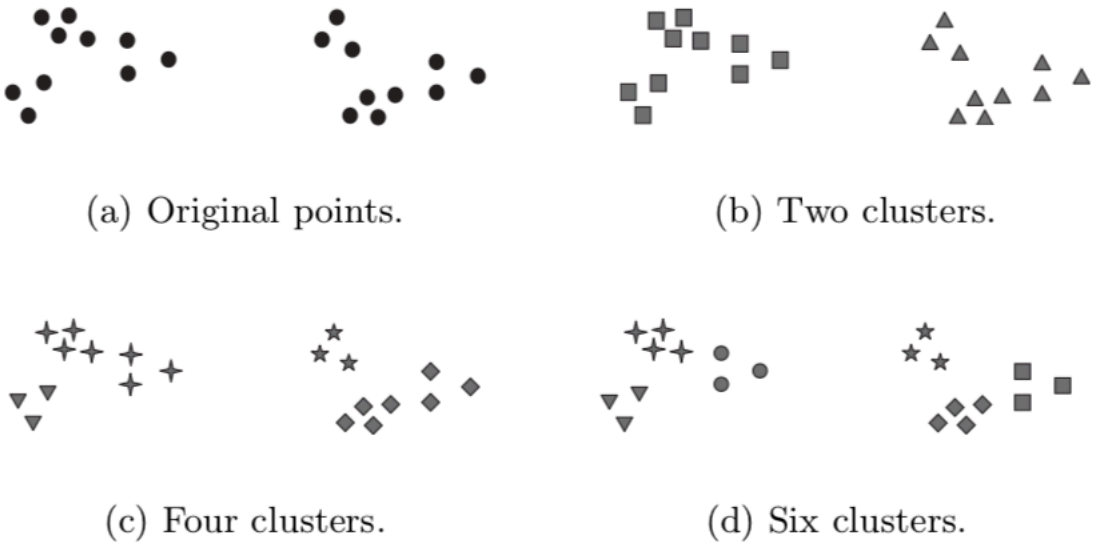
Given a set of records (instances, examples, objects, observations, …), organize them into clusters (groups, classes)

- Clustering: the process of grouping physical or abstract objects into classes of similar objects



Clustering

- Different ways of clustering the same data



(a) Original points.

(b) Two clusters.

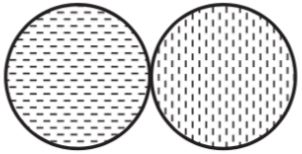
(c) Four clusters.

(d) Six clusters.

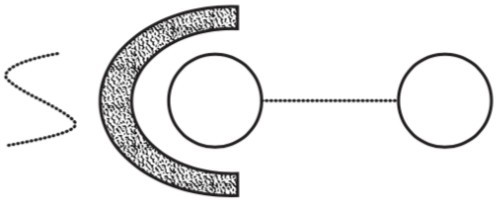
Clustering

- Cluster Concepts

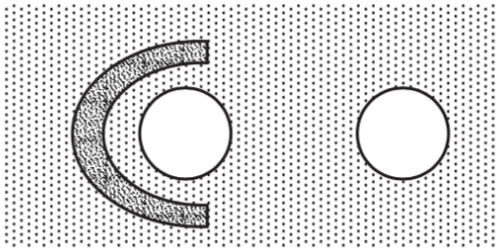
(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.



(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.



(c) Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster.



(d) Density-based clusters. Clusters are regions of high density separated by regions of low density.

Data Representation: Normalization

Data comes in all shapes and sizes - algorithms are useful for abstract data

- Can can have numerous similarity/proximity measures
- It is not always obvious how to weight relative measures

Example:

- For each measure subtract the mean and scale by the variance (*i*=data point, *j*=feature)

$$x_{ij} = \frac{x_{ij} - \bar{x}_j}{\sigma(x_j)}$$

- Thus each feature has a mean of zero and unit variance
- Beware! Must be careful, normalizing can reduce desired/necessary separation, and worsen clustering performance!

K-means Clustering

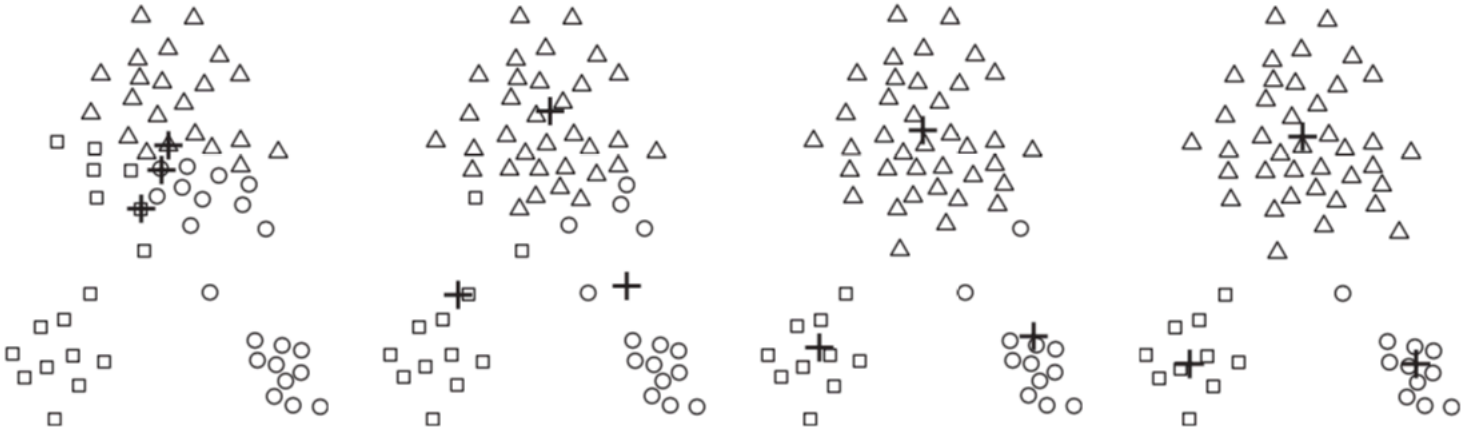
- The K-means clustering technique is simple
- Basic algorithm.
  - First choose K initial centroids, where K is a user- specified parameter, namely, the number of clusters desired.
  - Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster.
  - The centroid of each cluster is then updated based on the points assigned to the cluster.
  - We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same.

K-means Clustering

- High Level Algorithm

1: Select K points as initial centroids.
2: **repeat**
3: Form K clusters by assigning each point to its closest centroid.
4: Recompute the centroid of each cluster.
5: **until** Centroids do not change.

K-means Clustering



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

K-means Clustering

1. Initialization
   - Data are $\mathbf{x}_{1:N}$
   - Choose initial cluster means $\mathbf{m}_{1:k}$ (same dimension as data).
2. Repeat
   1. Assign each data point to its closest mean

$$z_n = \arg \min_{i \in \{1,\ldots,k\}} d(\mathbf{x}_n, \mathbf{m}_i)$$

   2. Compute each cluster mean to be the coordinate-wise average over data points assigned to that cluster,

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{\{n : z_n = k\}} \mathbf{x}_n$$

3. Until assignments $\mathbf{z}_{1:N}$ do not change

K-means Clustering

- The centroid for the K-means algorithm can be mathematically derived when the proximity function is Euclidean distance and the objective is to minimize the SSE.

- We can best update a cluster centroid so that the cluster SSE is minimized.

$$\text{SSE} = \sum_{i=1}^{K} \sum_{\mathbf{x} \in C_i} dist(\mathbf{c}_i, \mathbf{x})^2$$

$$\mathbf{c}_i = \frac{1}{m_i} \sum_{\mathbf{x} \in C_i} \mathbf{x}$$

$$\text{SSE} = \sum_{i=1}^{K} \sum_{x \in C_i} (c_i - x)^2$$

$\boldsymbol{C_i}$ is the centroid mean of the *ith* cluster

Symbol	Description
$\mathbf{x}$	An object.
C_i	The i^{th} cluster.
$\mathbf{c}_i$	The centroid of cluster C_i.
$\mathbf{c}$	The centroid of all points.
m_i	The number of objects in the i^{th} cluster.
m	The number of objects in the data set.
K	The number of clusters.

K-means Clustering

- We can solve for the *kth* centroid $\boldsymbol{c_k}$, which minimizes the equation by differentiating the SSE, setting it equal to 0, and solving, as indicated below.

$$\begin{aligned}\frac{\partial}{\partial c_k}\text{SSE} &= \frac{\partial}{\partial c_k}\sum_{i=1}^{K}\sum_{x\in C_i}(c_i - x)^2 \\ &= \sum_{i=1}^{K}\sum_{x\in C_i}\frac{\partial}{\partial c_k}(c_i - x)^2 \\ &= \sum_{x\in C_k} 2 * (c_k - x_k) = 0\end{aligned}$$

$$\sum_{x\in C_k} 2 * (c_k - x_k) = 0 \Rightarrow m_k c_k = \sum_{x\in C_k} x_k \Rightarrow c_k = \frac{1}{m_k}\sum_{x\in C_k} x_k$$

- Thus the best centroid for minimizing the SSE of $\boldsymbol{x \in C_k}$ a cluster is the mean of the points in the cluster.

K-means Clustering

- Steps 3 and 4 of the K-means algorithm directly attempt to minimize the SSE.
  - Step 3 forms clusters by assigning points to their nearest centroid, which minimizes the SSE for the given set of centroids.
  - Step 4 re-computes the centroids so as to further minimize the SSE.
- However, the actions of K-means in Steps 3 and 4 are only guaranteed to find a local minimum with respect to the SSE.
  - They are based on optimizing the SSE for specific choices of the centroids and clusters, rather than for all possible choices.

K-means Clustering

- The space requirements for K-means are modest because only the data points and centroids are stored.

- Specifically, the storage required is $O((m+K)n)$, where
  - m is the number of points

  - n is the number of attributes.

- The time requirements for K-means are also modest—basically linear in the number of data points, $O(I*K*m*n)$ where,
  - I is the number of iterations required for convergence.

    - I is often small and can usually be safely bounded, as most changes typically occur in the first few iterations.

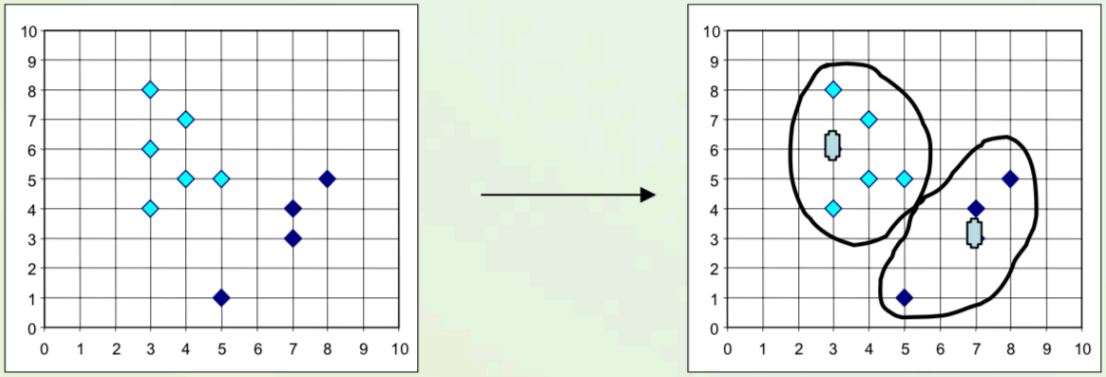
  - Therefore, K-means is linear in m, the number of points, and is efficient as well provided that K, the number of clusters, is significantly less than m.

K-means Clustering

1: **for** `iter`$= 1$ to `iternum` **do**
2: Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$;
3: **while** not convergence **do**
4: **for** $i = 1$ to m **do**
5: $c^{(i)} \leftarrow \arg\min_k \|\mathbf{x}^{(i)} - \mu_k\|$;
6: **end for**
7: **for** $k = 1$ to K **do**
8: $\mu_k \leftarrow$ element-wise average of points assigned to cluster k;
9: **end for**
10: **end while**
11: **end for**
12: Pick the one that gives the lowest cost;

What is the problem of k-Means Method?

- The k-means algorithm is sensitive to outliers !
  - Since an object with an extremely large value may substantially distort the distribution of the data.
- There are other limitations – still a need for reducing costs of calculating distances to centroids.
- **K-Medoids**: Instead of taking the **mean** value of the object in a cluster as a reference point, **medoids** can be used, which is the **most centrally located** object in a cluster.



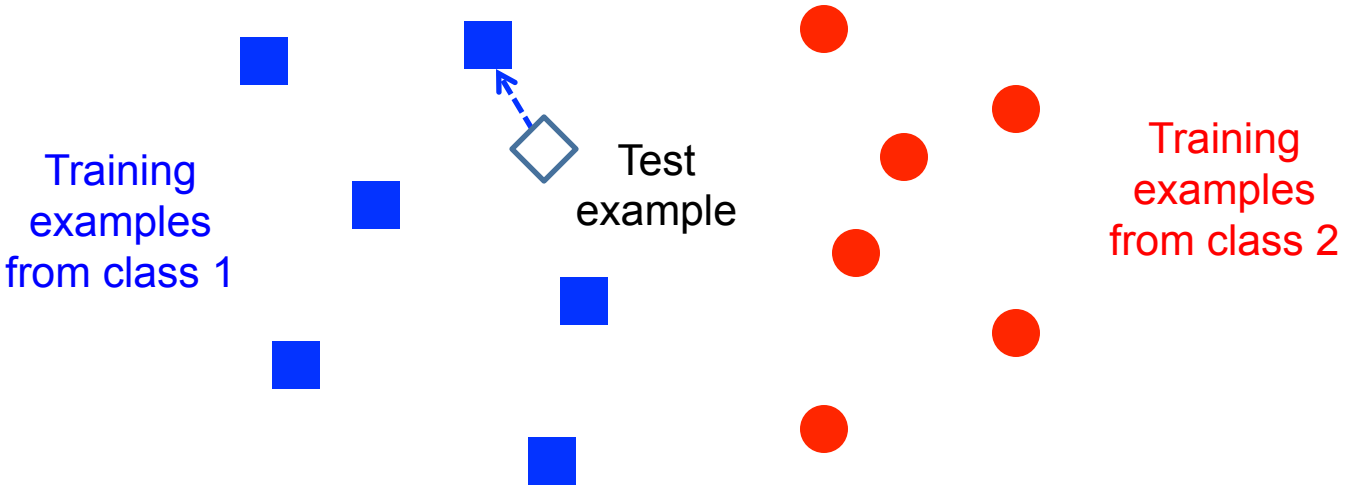
PCA

- **principal components analysis (PCA)** is a technique that can be used to simplify a dataset
- It is a linear transformation that chooses a new coordinate system for the data set such that

  greatest variance by any projection of the data set comes to lie on the first axis (then called the first principal component),

  the second greatest variance on the second axis, and so on.
- PCA can be used for reducing dimensionality by eliminating the later principal components.

K-Nearest Neighbors



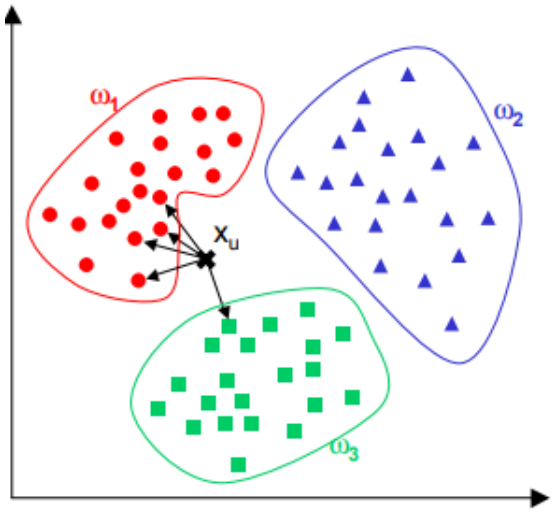
f(**x**) = label of the training example nearest to **x**

- All we need is a distance function for our inputs
- No training required!

Slide credit: L. Lazebnik

Introduction

- **The K Nearest Neighbor Rule (k-NNR) is a very intuitive method that classifies unlabeled examples based on their similarity with examples in the training set**
  - **For a given unlabeled example $x_u \in \Re_D$, find the k "closest" labeled examples in the training data set and assign x_u to the class that appears most frequently within the k-subset**
- **The k-NNR only requires**
  - An integer k
  - A set of labeled examples (training data)
  - A metric to measure "closeness"
- **Example**
  - In the example below we have three classes and the goal is to find a class label for the unknown example x_u
  - In this case we use the Euclidean distance and a value of k=5 neighbors
  - Of the 5 closest neighbors, 4 belong to ω1 and 1 belongs to ω3, so x_u is assigned to ω1, the predominant class



K-Nearest Neighbors

- **K-nearest neighbours** uses the local neighborhood to obtain a prediction
- The K memorized examples more similar to the one that is being classified are retrieved
- A distance function is needed to compare the examples similarity
  - Euclidean distance ($d(x_j, x_k) = \sqrt{\sum_i (x_{j,i} - x_{k,i})^2}$)
  - Mahnattan distance ($d(x_j, x_k) = \sum_i |x_{j,i} - x_{k,i}|$)
- This means that if we change the distance function, we change how examples are classified

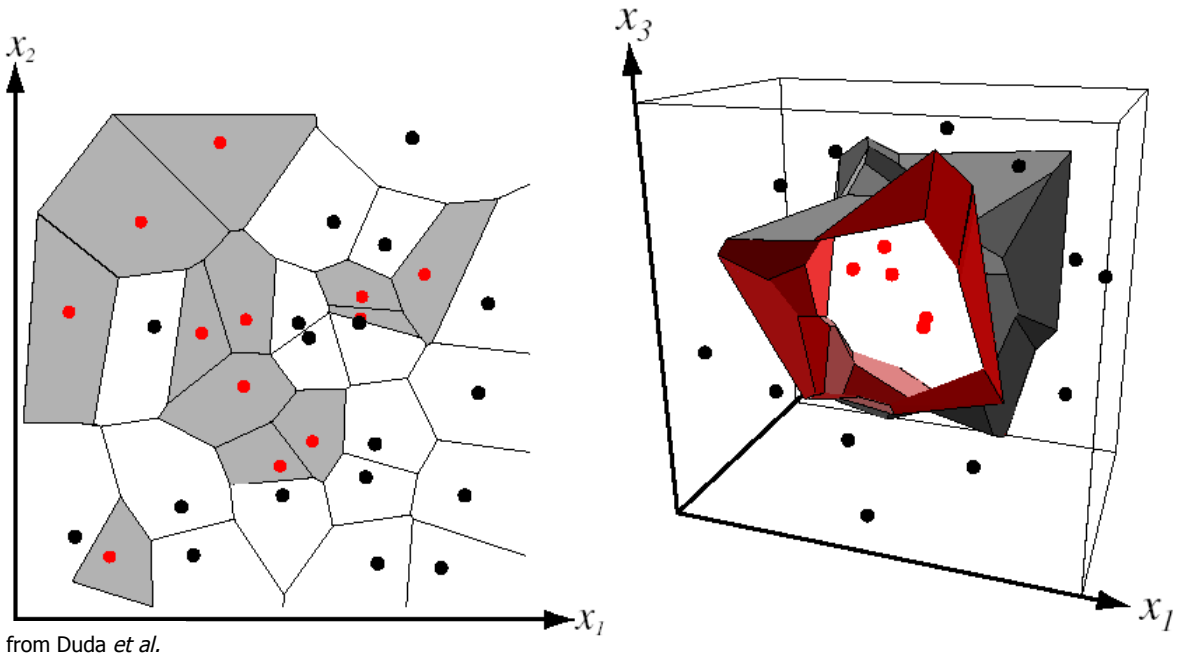
K-Nearest Neighbors

$$\text{dist}(\mathbf{x}, \mathbf{z}) = \left(\sum_{r=1}^{d} |x_r - z_r|^p \right)^{1/p}$$

- $p = 1$: Manhattan Distance (l_1 norm)
- $p = 2$: Euclidean Distance (l_2 norm)
- $p \to 0$: (not well defined)
- $p \to \infty$: Maximum Norm

Nearest Neighbor Classifier

- Assign label of nearest training data point to each test data point

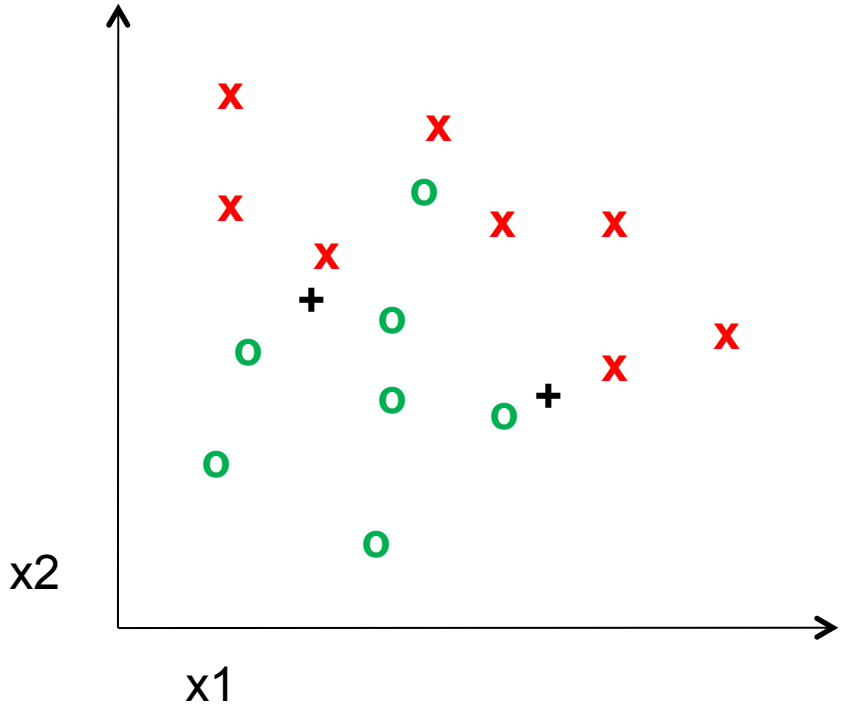


from Duda *et al.*

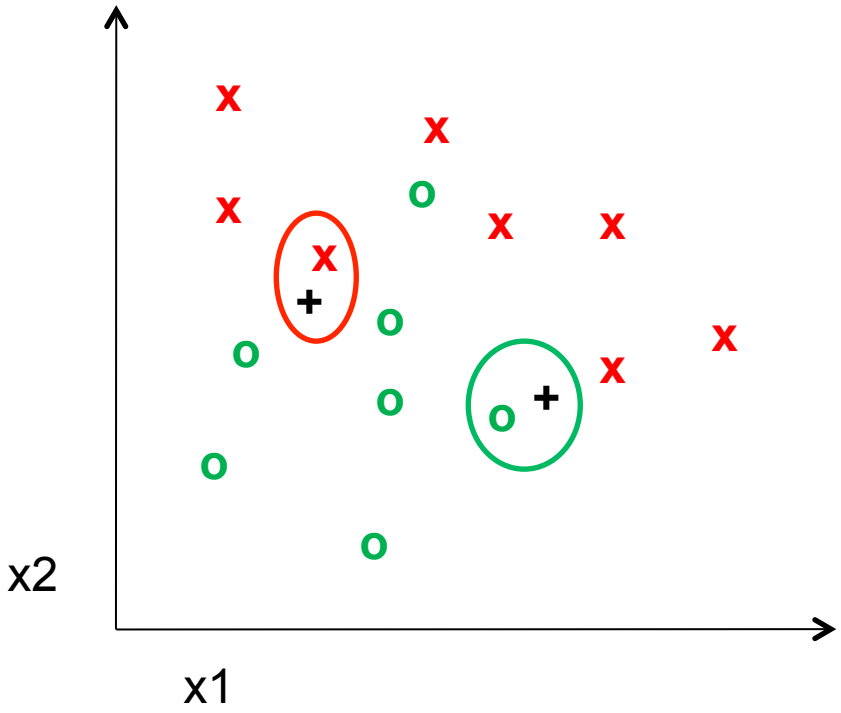
Partitioning of feature space
for two-category 2D and 3D data

Source: D. Lowe

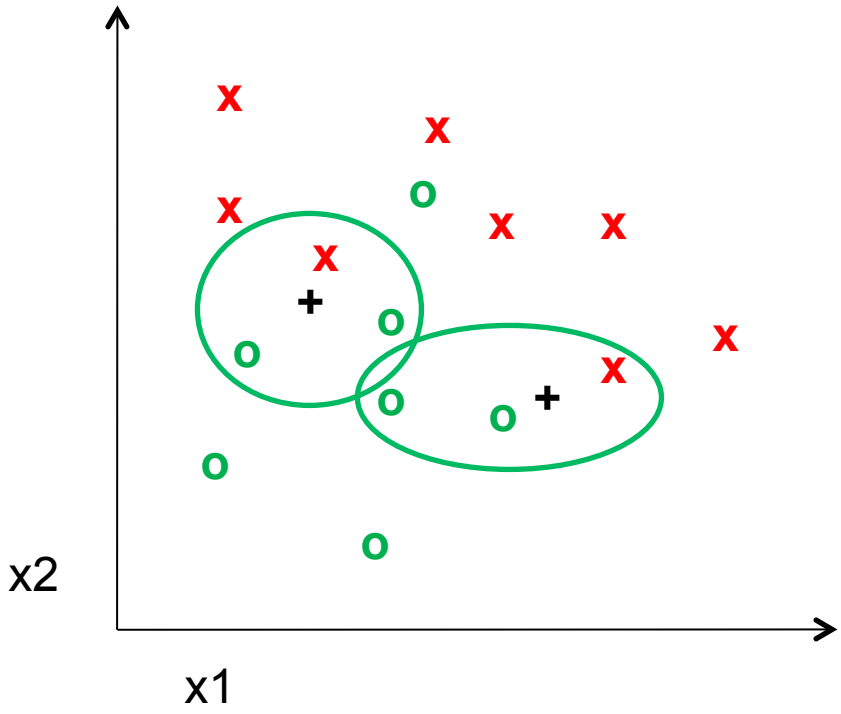
K-nearest neighbor - Example



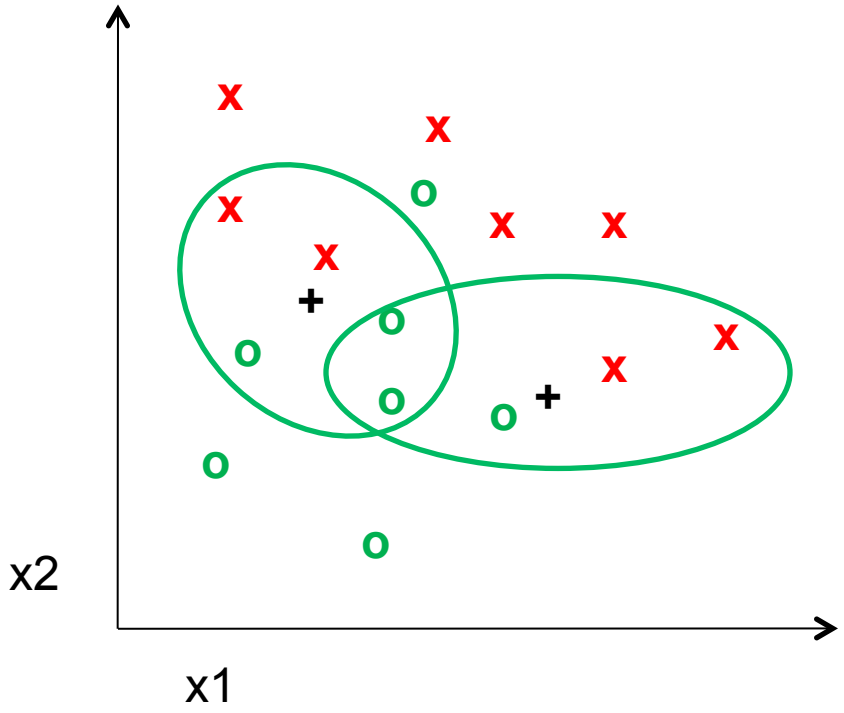
1-nearest neighbor



3-nearest neighbor



5-nearest neighbor



Example: k-Nearest Neighbors

Regression task

Given: $(x_1, y_1), \ldots, (x_m, y_m)$
Task: predict the response value $\hat{y}$ for a new input x

$\rightsquigarrow$ Idea: Let $\hat{y}(x)$ be the average of the k closest points:

1. Rank the data points $(x_1, y_1), \ldots, (x_m, y_m)$ in increasing order of distance from x in the input space, ie, $d(x_i, x) = |x_i - x|$.
2. Set the k best ranked points in $N_k(x)$.
3. Return the average of the y values of the k data points in $N_k(x)$.

In mathematical notation:

$$\hat{y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i = g(x)$$

Example: k-Nearest Neighbors

Classification task

Given: $(x_1, y_1), \ldots, (x_m, y_m)$
Task: predict the class $\hat{y}$ for a new input x.

$\rightsquigarrow$ Idea: let the k closest points vote and majority decide

1. Rank the data points $(x_1, y_1), \ldots, (x_m, y_m)$ in increasing order of distance from $\vec{x}$ in the input space, ie, $d(\vec{x_i}, \vec{x}) = |x_i - x|$.
2. Set the k best ranked points in $N_k(x)$.
3. Return the class that is most represented in the k data points of $N_k(x)$.

In mathematical notation:

$$\hat{G}(x) = \text{argmax}_{G \in \mathcal{G}} \sum_{x_i \in N_k(x) | y_i = G} \frac{1}{k}$$

Some slides used from

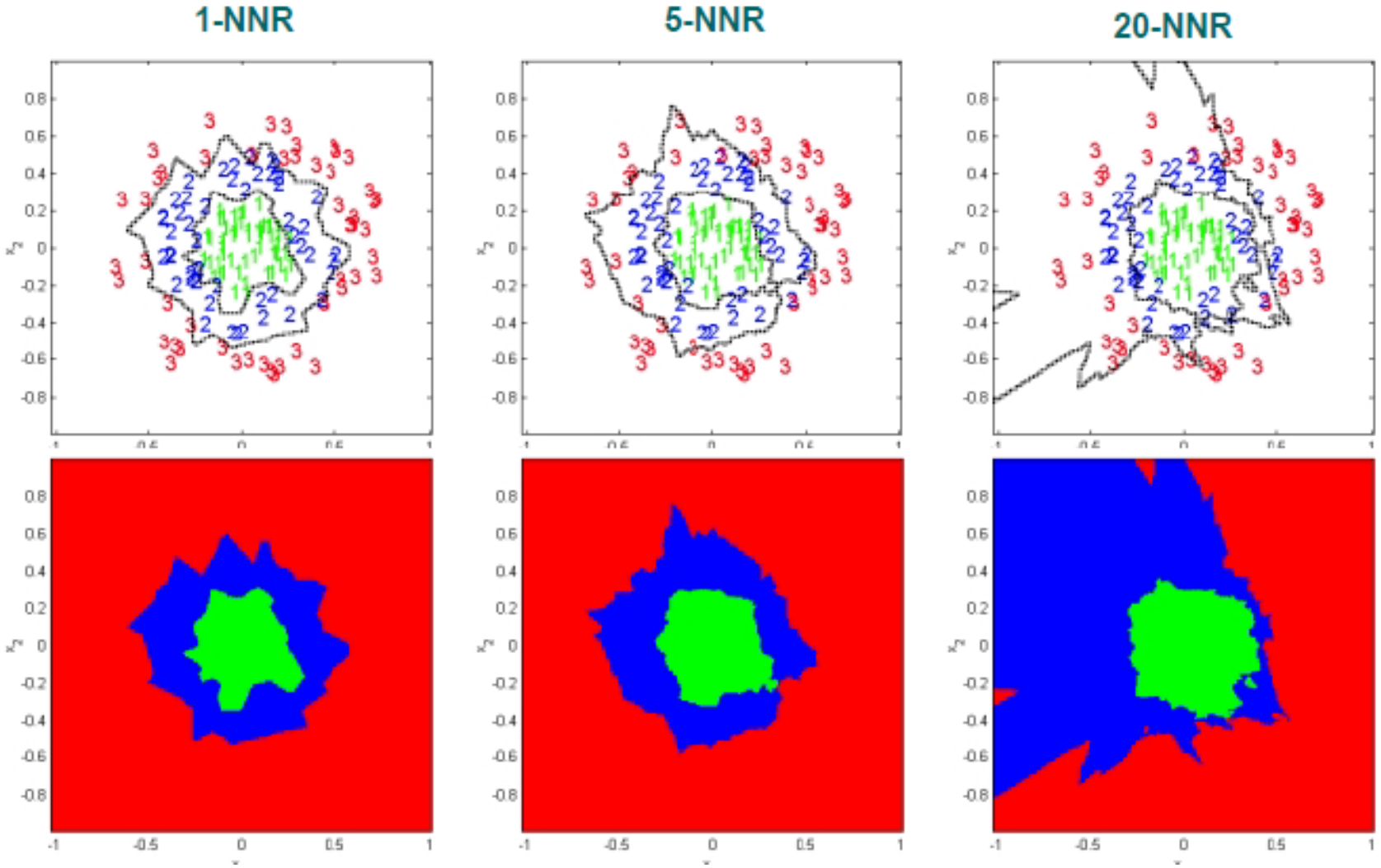
Introduction to Pattern Recognition

Ricardo Gutierrez-Osuna
Wright State University

Characteristics of the k-NNR classifier

- **Advantages**
  - Analytically tractable
  - Simple implementation
  - Nearly optimal in the large sample limit ($N \rightarrow \infty$)
    - $P[error]_{Bayes} > P[error]_{1\text{-}NNR} < 2P[error]_{Bayes}$
  - Uses local information, which can yield highly adaptive behavior
  - Lends itself very easily to parallel implementations
- **Disadvantages**
  - Large storage requirements
  - Computationally intensive recall
  - Highly susceptible to the curse of dimensionality
- **1-NNR versus k-NNR**
  - The use of large values of k has two main advantages
    - Yields smoother decision regions
    - Provides probabilistic information
  - The ratio of examples for each class gives information about the ambiguity of the decision
  - However, too large values of k are detrimental
    - It destroys the locality of the estimation since farther examples are taken into account
    - In addition, it increases the computational burden

k-NNR versus 1-NNR



* ***Ricardo Gutierrez-Osuna***

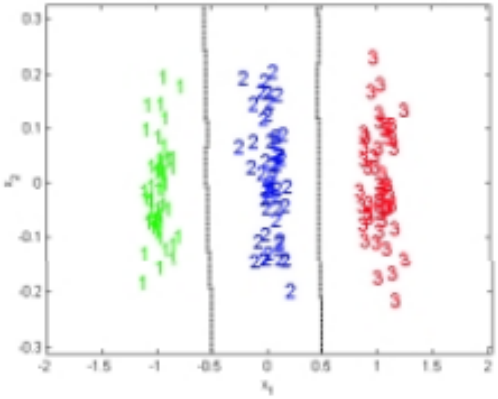
k-NNR versus 1-NNR

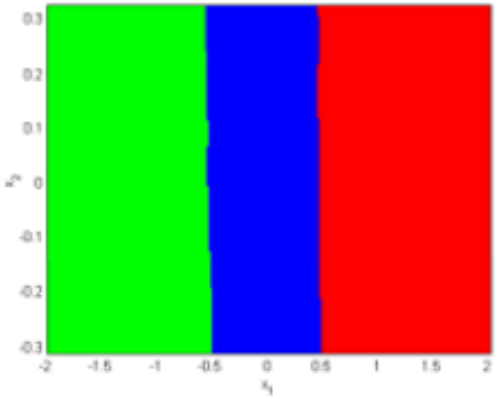
- **The basic k-NNR algorithm stores all the examples in the training set, creating high storage requirements (and computational cost)**
  - However, the entire training set need not be stored since the examples contain information that is highly redundant
    - A degenerate case is the earlier example with the multimodal classes. In that example, each of the clusters could be replaced by its mean vector, and the decision boundaries would be practically identical
  - In addition, almost all of the information that is relevant for classification purposes is located around the decision boundaries
- **A number of methods, called edited k-NNR, have been derived to take advantage of this information redundancy**
  - One alternative [Wilson 72] is to classify all the examples in the training set and remove those examples that are misclassified, in an attempt to separate classification regions by removing ambiguous points
  - The opposite alternative [Ritter 75], is to remove training examples that are classified correctly, in an attempt to define the boundaries between classes by eliminating points in the interior of the regions
- **A different alternative is to reduce the training examples to a set of prototypes that are representative of the underlying data**
  - The issue of selecting prototypes will be the subject of the lectures on clustering

k-NNR and the problem of feature weighting

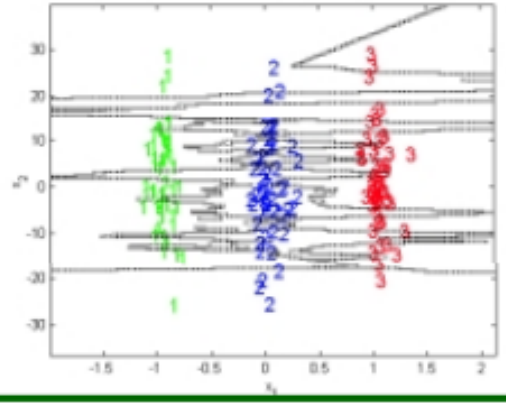
- **The basic k-NNR computes the similarity measure based on the Euclidean distance**
  - This metric makes the k-NNR very sensitive to noisy features
  - As an example, we have created a data set with three classes and two dimensions
    - The first axis contains all the discriminatory information. In fact, class separability is excellent
    - The second axis is white noise and, thus, does not contain classification information

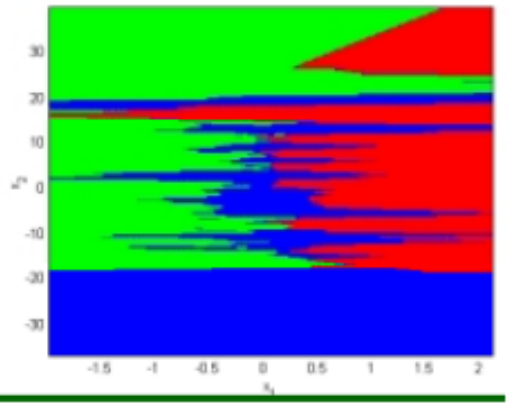
- **In the first example, both axes are scaled properly**
  - The k-NNR (k=5) finds decision boundaries fairly closed to the optimal





- **In the second example, the magnitude of the second axis has been increased two order of magnitudes (see axes tick marks)**
  - The k-NNR is biased by the large values of the second axis and its performance is very poor





* ***Ricardo Gutierrez-Osuna***

Feature weighting

- **The previous example illustrated the Achilles' heel of the k-NNR classifier: its sensitivity to noisy axes**
  - A possible solution would be to normalize each feature to N(0,1)
  - However, normalization does not resolve the curse of dimensionality. A close look at the Euclidean distance shows that this metric can become very noisy for high dimensional problems where only a few of the features carry the classification information

$$d(x_u, x) = \sqrt{\sum_{k=1}^{D} (x_u(k) - x(k))^2}$$

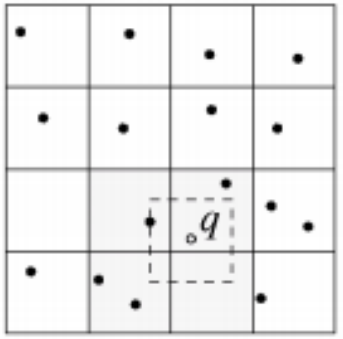
- **The solution to this problem is to modify the Euclidean metric by a set of weights that represent the information content or "goodness" of each feature**

$$d_w(x_u, x) = \sqrt{\sum_{k=1}^{D} \left(w[k] \cdot (x_u[k] - x[k])\right)^2}$$

  - Note that this procedure is identical to performing a linear transformation where the transformation matrix is diagonal with the weights placed in the diagonal elements
    - From this prospective, feature weighting can be thought of as a special case of feature extraction where the different features are not allowed to interact (null off-diagonal elements in the transformation matrix)
    - Feature subset selection, which will be covered at the end of the course, can be viewed as a special case of feature weighting where the weights can only take binary [0,1] values
  - Do not confuse feature-weighting with distance-weighting, a k-NNR variant that weights the contribution of each of the k nearest neighbors according to their distance to the unlabeled example
    - Distance-weighting distorts the k-NNR estimate of $P(\omega_i|x)$ and is NOT recommended
    - Studies have shown that distance-weighting DOES NOT improve k-NNR classification performance

Improving the nearest neighbor search procedure

- **The problem of nearest neighbor can be stated as follows**
  - Given a set of N points in D-dimensional space and an unlabeled example $x_u \in \Re^D$, find the point that minimizes the distance to x_u
  - The naïve approach of computing a set of N distances (N×k distances for k-NNR), and finding the (k) smallest becomes impractical for large values of N and D
- **There are two classical algorithms that speed up the nearest neighbor search**
  - Bucketing (a.k.a Elias's algorithm) [Welch 1971]
  - k-d trees [Bentley, 1975], [Friedman et al, 1977]
- **Bucketing**
  - In the Bucketing algorithm, the space is divided into identical cells and for each cell the data points inside it are stored in a list
  - The cells are examined in order of increasing distance from the query point and for each cell the distance is computed between its internal data points and the query point
  - The search terminates when the distance from the query point to the cell exceeds the distance to the closest point already visited

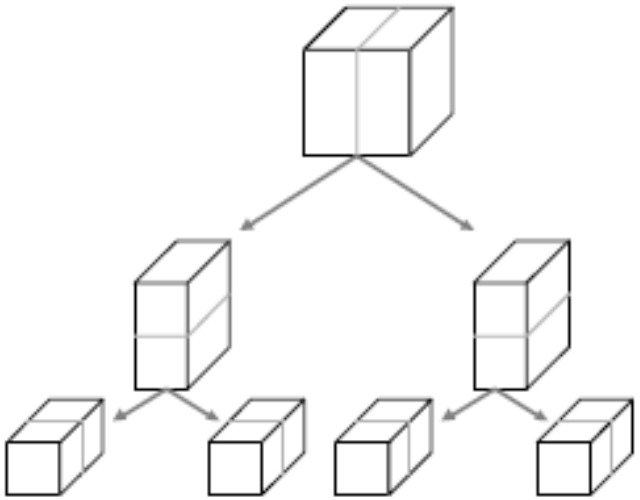


- **k-d trees**
  - A k-d tree is a generalization of a binary search tree in high dimensions
    - Each internal node in a k-d tree is associated with a hyper-rectangle and a hyper-plane orthogonal to one of the coordinate axis
    - The hyper-plane splits the hyper-rectangle into two parts, which are associated with the child nodes
    - The partitioning process goes on until the number of data points in the hyper-rectangle falls below some given threshold
  - The effect of a k-d tree is to partition the (multi-dimensional) sample space according to the underlying distribution of the data, the partitioning being finer in regions where the density of pints is higher
    - For a given query point, the algorithm works by first descending the the tree to find the data points lying in the cell that contains the query point
    - Then it examines surrounding cells if they overlap the ball centered at the query point and the closest data point so far

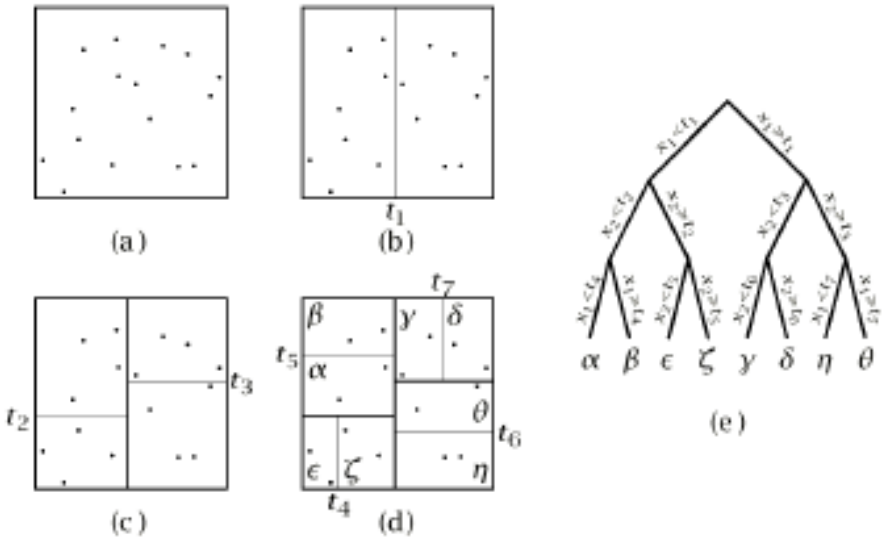
* ***Ricardo Gutierrez-Osuna***

k-d tree example

Data structure (3D case)



Partitioning (2D case)



* ***Ricardo Gutierrez-Osuna***