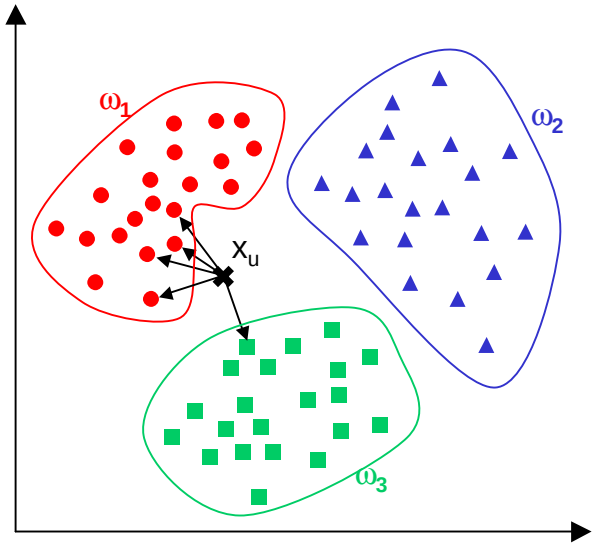
Lecture 8: The K Nearest Neighbor Rule (k-NNR)

- Introduction
- k-NNR in action
- k-NNR as a lazy algorithm
- Characteristics of the k-NNR classifier
- Optimizing storage requirements
- Feature weighting
- Improving the nearest neighbor search

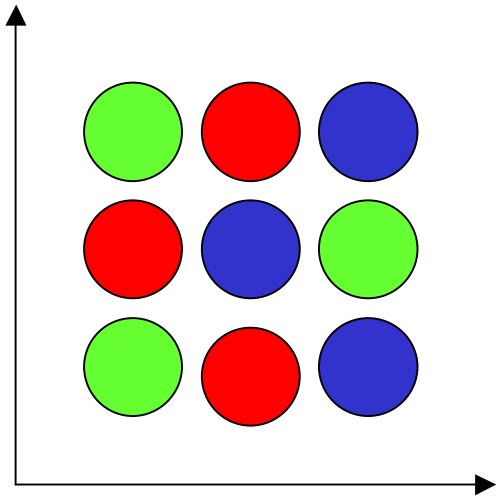
Introduction

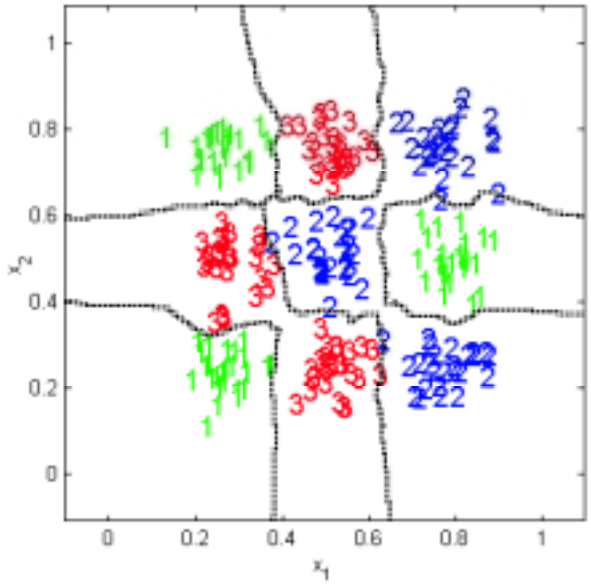
- **The K Nearest Neighbor Rule (k-NNR) is a very intuitive method that classifies unlabeled examples based on their similarity with examples in the training set**
  - **For a given unlabeled example $x_u \in \Re^D$, find the k "closest" labeled examples in the training data set and assign x_u to the 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 ω_1 and 1 belongs to ω_3, so xu is assigned to ω_1, the predominant class

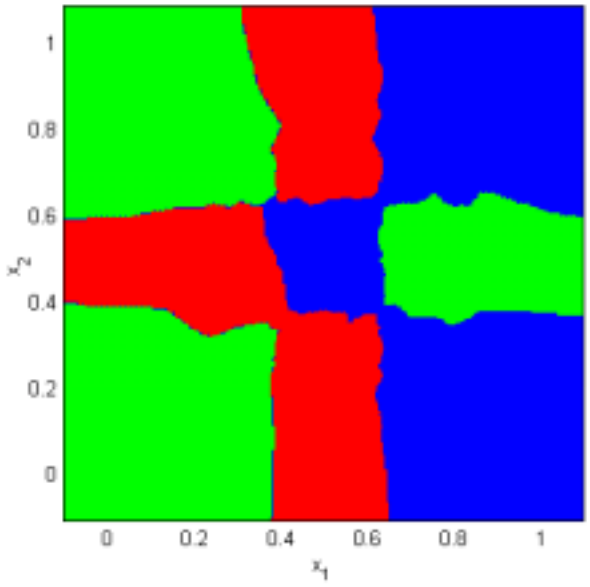


k-NNR in action: example 1

- **We have generated data for a 2-dimensional 3-class problem, where the class-conditional densities are multi-modal, and non-linearly separable, as illustrated in the figure**
- **We used the k-NNR with**
  - k = five
  - Metric = Euclidean distance
- **The resulting decision boundaries and decision regions are shown below**

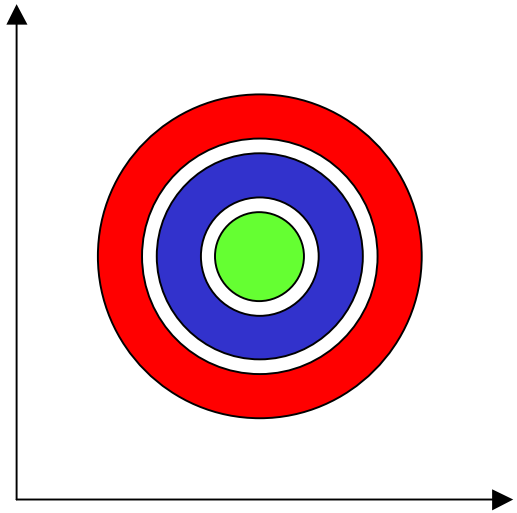


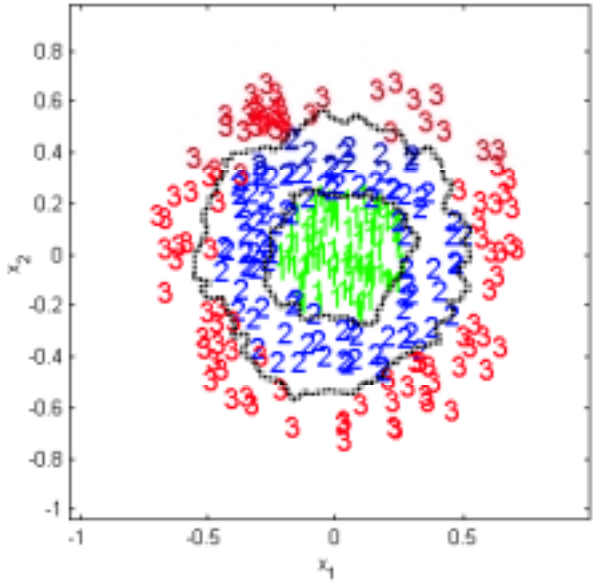


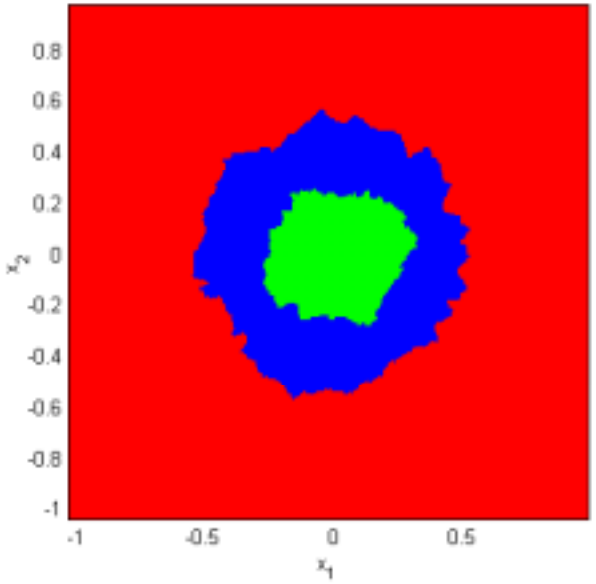


k-NNR in action: example 2

- We have generated data for a 2-dimensional 3-class problem, where the class-conditional densities are unimodal, and are distributed in rings around a common mean. These classes are also non-linearly separable, as illustrated in the figure
- We used the k-NNR with
  - k = five
  - Metric = Euclidean distance
- The resulting decision boundaries and decision regions are shown below







k-NNR, a lazy (machine learning) algorithm

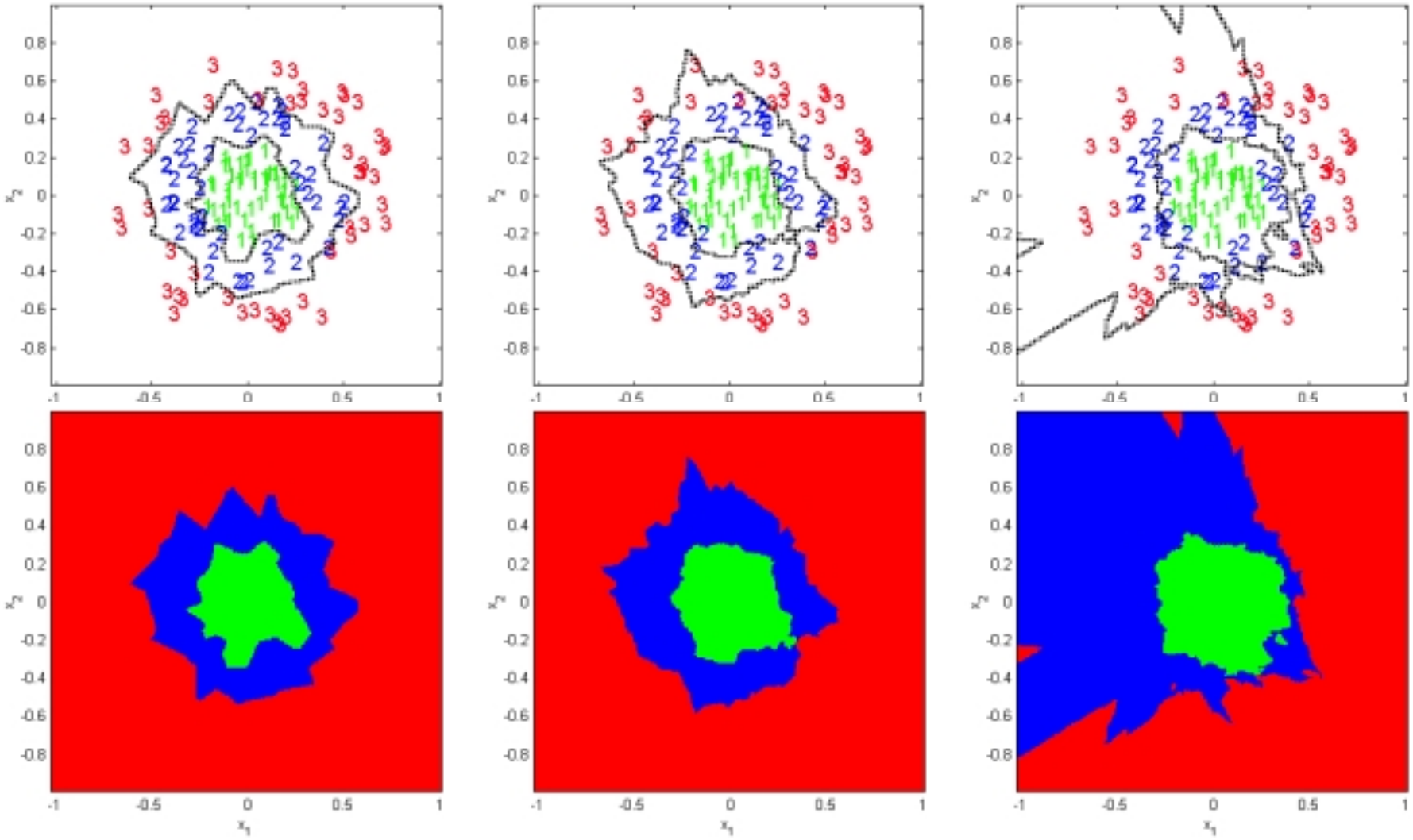
- **K-NNR is considered a lazy learning algorithm [Aha]**
  - **Defers** data processing until it receives a request to classify an unlabelled example
  - Replies to a request for information by **combining** its stored training data
  - **Discards** the constructed answer and any intermediate results
- **Other names for lazy algorithms**
  - Memory-based, Instance-based , Exemplar-based , Case-based, Experience-based
- **This strategy is opposed to an eager learning algorithm which**
  - Compiles its data into a compressed description or model
    - A density estimate or density parameters (statistical PR)
    - A graph structure and associated weights (neural PR)
  - Discards the training data after compilation of the model
  - Classifies incoming patterns using the induced model, which is retained for future requests
- **Tradeoffs**
  - Lazy algorithms have fewer computational costs than eager algorithms during training
  - Lazy algorithms have greater storage requirements and higher computational costs on recall

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

1-NNR **5-NNR** **20-NNR**



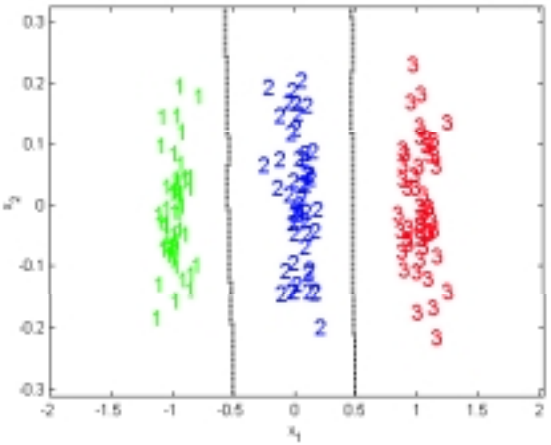
Optimizing storage requirements

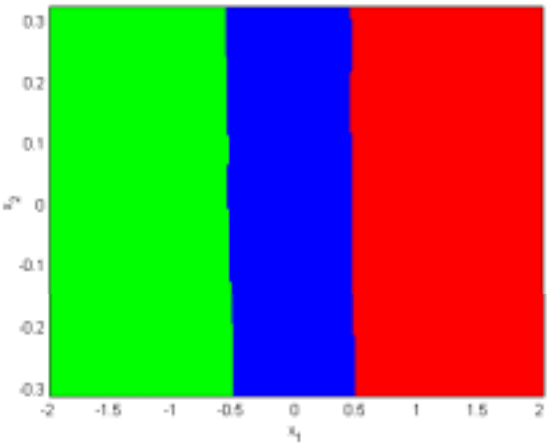
- **The basic k-NNR algorithm stores all the examples in the training set, creating high storage requirements (and computational cost)**
  - However, the entire training set need not be stored since the examples contain information that is highly redundant
    - A degenerate case is the earlier example 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

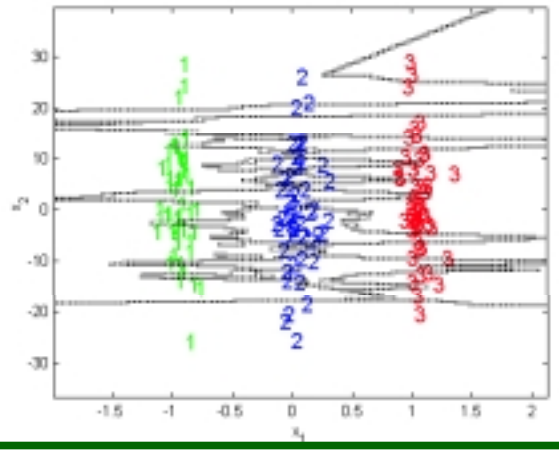
- **The basic k-NNR computes the similarity measure based on the Euclidean distance**
  - This metric makes the k-NNR very sensitive to noisy features
  - As an example, we have created a data set with three classes and two dimensions
    - The first axis contains all the discriminatory information.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

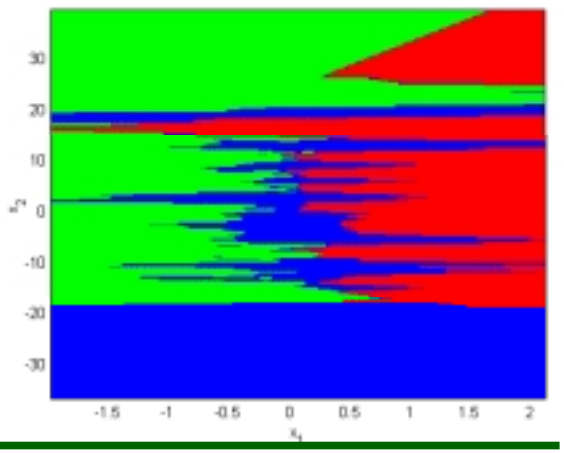
- **In the first example, both axes are scaled properly**
  - The k-NNR (k=5) finds decision boundaries fairly closed to the optimal





- **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





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**

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

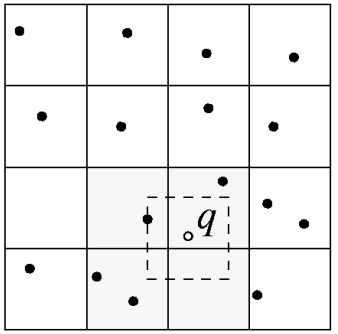
  - Note that this procedure is identical to performing a linear transformation where the transformation matrix is diagonal with the weights placed in the diagonal elements
    - From this prospective, feature weighting can be thought of as a special case of feature extraction where the different features 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

Feature weighting methods

- **Feature weighting methods are divided in two groups**
  - Performance bias methods
  - Preset bias methods
- **Performance bias methods**
  - These methods find a set of weights through an iterative procedure that uses the performance of the classifier as guidance to select a new set of weights
  - These methods normally give good solutions since they can incorporate the classifier's feedback into the selection of weights
- **Preset bias methods**
  - These methods obtain the values of the weights using a pre-existing function that measures the information content of each feature (i.e., mutual information and correlation between each feature and the class label)
  - This methods have the advantage of executing very fast
- **The issue of performance bias versus preset bias will be revisited when we cover feature subset selection (FSS)**
  - In FSS the performance bias methods are called **wrappers** and preset bias methods are called **filters**

Improving the nearest neighbor search procedure

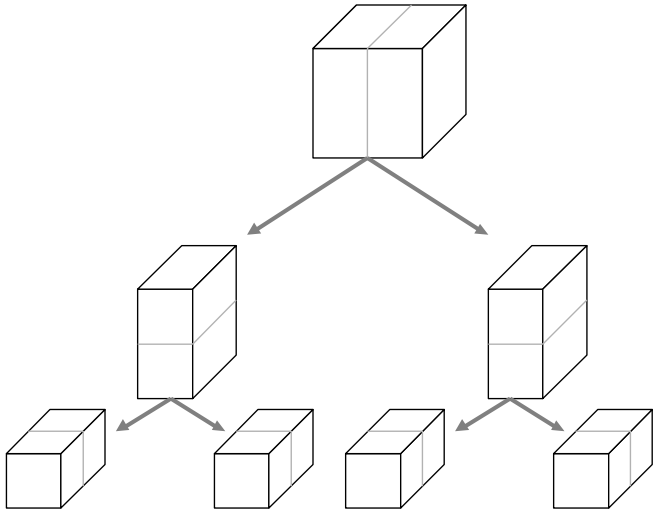
- **The problem of nearest neighbor can be stated as follows**
  - Given a set of N points in D-dimensional space and an unlabeled example $x_u \in \Re^D$, find the point that minimizes the distance to x_u
  - The naïve approach of computing a set of N distances (N×k distances for k-NNR), and finding 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



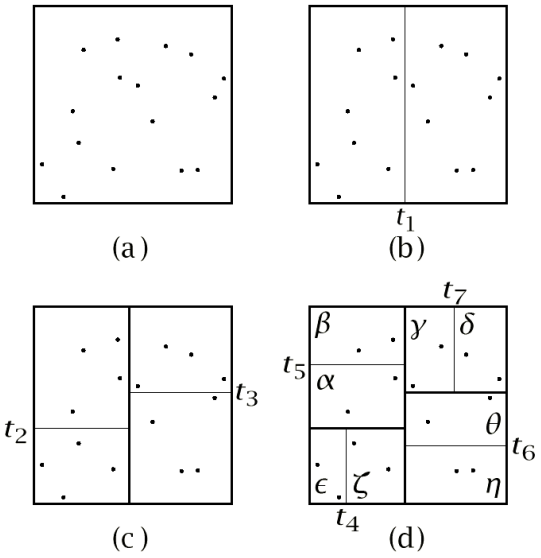
- **k-d trees**
  - A k-d tree is a generalization of a binary search tree in high dimensions
    - Each internal node in a k-d tree is associated with a hyper-rectangle and a hyper-plane orthogonal to one of the coordinate axis
    - The hyper-plane splits the hyper-rectangle into two parts, which are 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

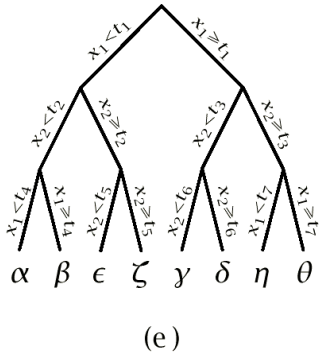
k-d tree example

Data structure (3D case)



Partitioning (2D case)





(e)