LECTURE 8: Nearest Neighbors

- Nearest Neighbors density estimation
- The k Nearest Neighbors classification rule
- kNN as a lazy learner
- Characteristics of the kNN classifier
- Optimizing the kNN classifier

Non-parametric density estimation: review

- **Recall from the previous lecture that the general expression for non-parametric density estimation is**

$$p(x) \cong \frac{k}{NV} \quad \text{where} \begin{cases} V \text{ is the volume surrounding } x \\ N \text{ is the total number of examples} \\ k \text{ is the number of examples inside } V \end{cases}$$

- **At that time, we mentioned that this estimate could be computed by**
  - Fixing the volume V and determining the number k of data points inside V
    - This is the approach used in Kernel Density Estimation
  - Fixing the value of k and determining the minimum volume V that encompasses k points in the dataset
    - This gives rise to the **k Nearest Neighbor (kNN)** approach, which is the subject of this lecture

kNN Density Estimation (1)

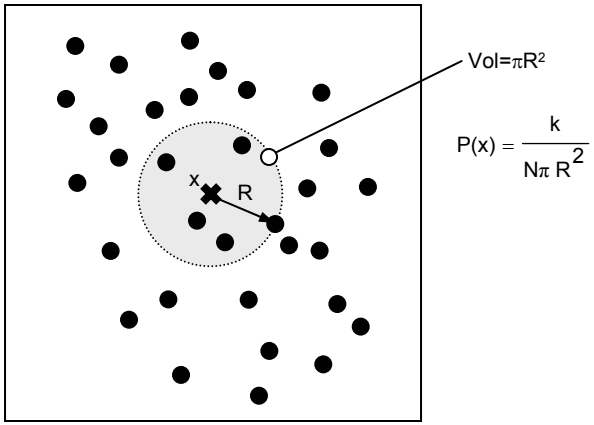
- **In the kNN method we grow the volume surrounding the estimation point *x* until it encloses a total of *k* data points**
- **The density estimate then becomes**

$$P(x) \cong \frac{k}{NV} = \frac{k}{N \cdot c_D \cdot R_k^D(x)}$$

- $R_k(x)$ is the distance between the estimation point *x* and its k-th closest neighbor
- c_D is the volume of the unit sphere in D dimensions, which is equal to

$$c_D = \frac{\pi^{D/2}}{(D/2)!} = \frac{\pi^{D/2}}{\Gamma(D/2+1)}$$

  - Thus $c_1=2$, $c_2=\pi$, $c_3=4\pi/3$ and so on

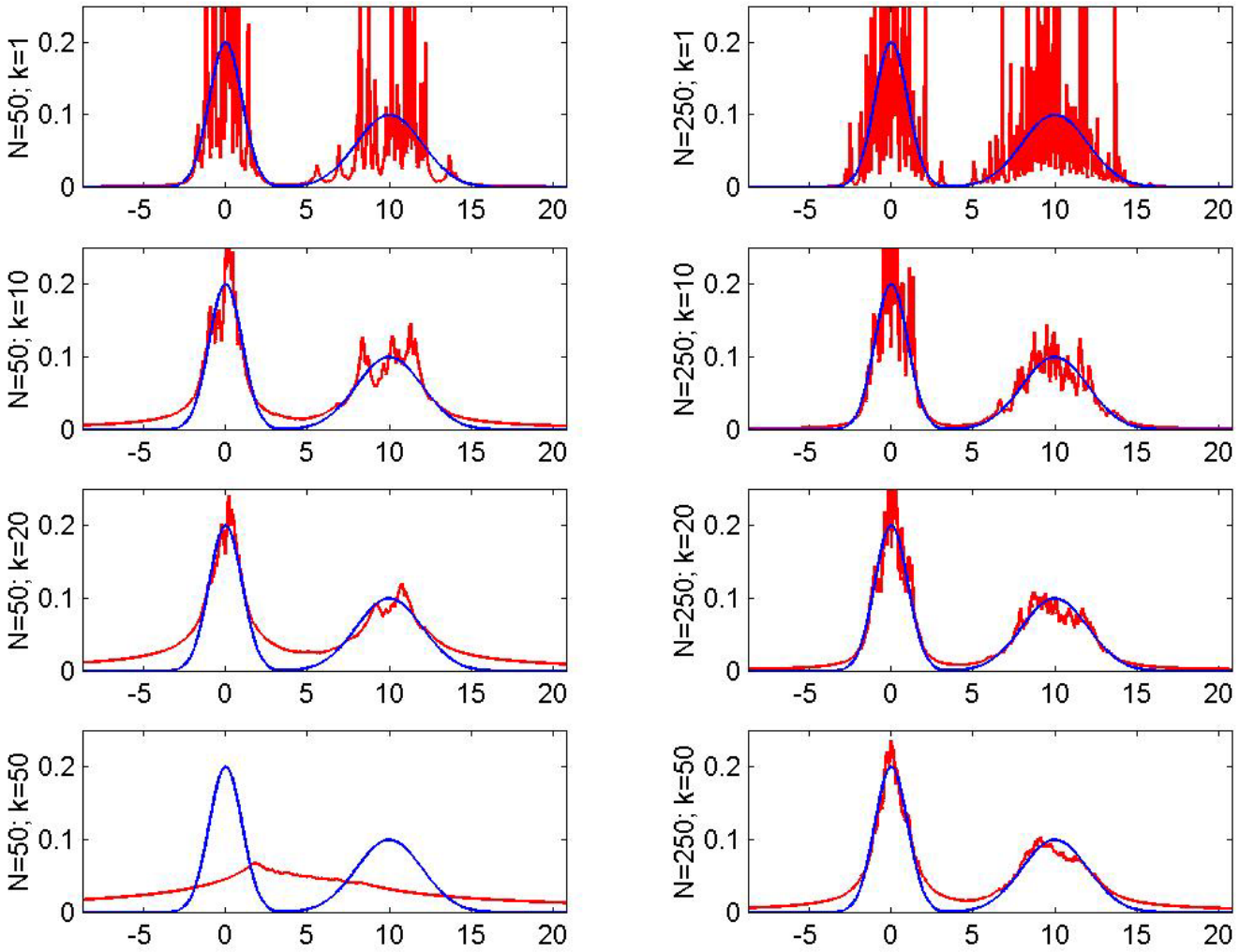


kNN Density Estimation (1)

- **In general, the estimates that can be obtained with the kNN method are not very satisfactory**
  - The estimates are prone to local noise
  - The method produces estimates with very heavy tails
  - Since the function $R_k(x)$ is not differentiable, the density estimate will have discontinuities
  - The resulting density is not a true probability density since its integral over all the sample space diverges
- **These properties are illustrated in the next few slides**

kNN Density Estimation, example 1

- To illustrate the behavior of kNN we generated several density estimates for a univariate mixture of two Gaussians: P(x)=½N(0,1)+½N(10,4) and several values of N and k

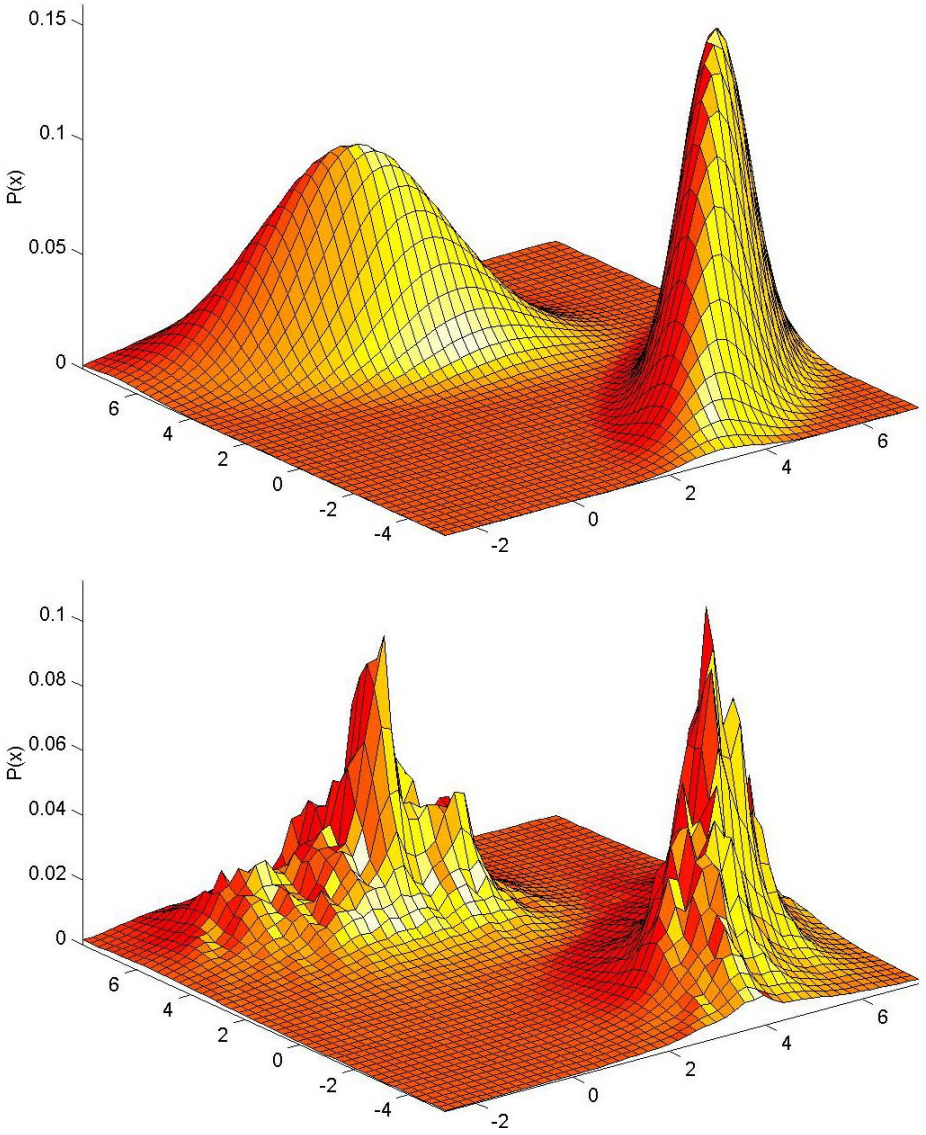


kNN Density Estimation, example 2 (a)

- **The performance of the kNN density estimation technique on two dimensions is illustrated in these figures**
  - The top figure shows the true density, a mixture of two bivariate Gaussians

$$p(x) = \frac{1}{2}N(\mu_1, \Sigma_1) + \frac{1}{2}N(\mu_2, \Sigma_2)$$

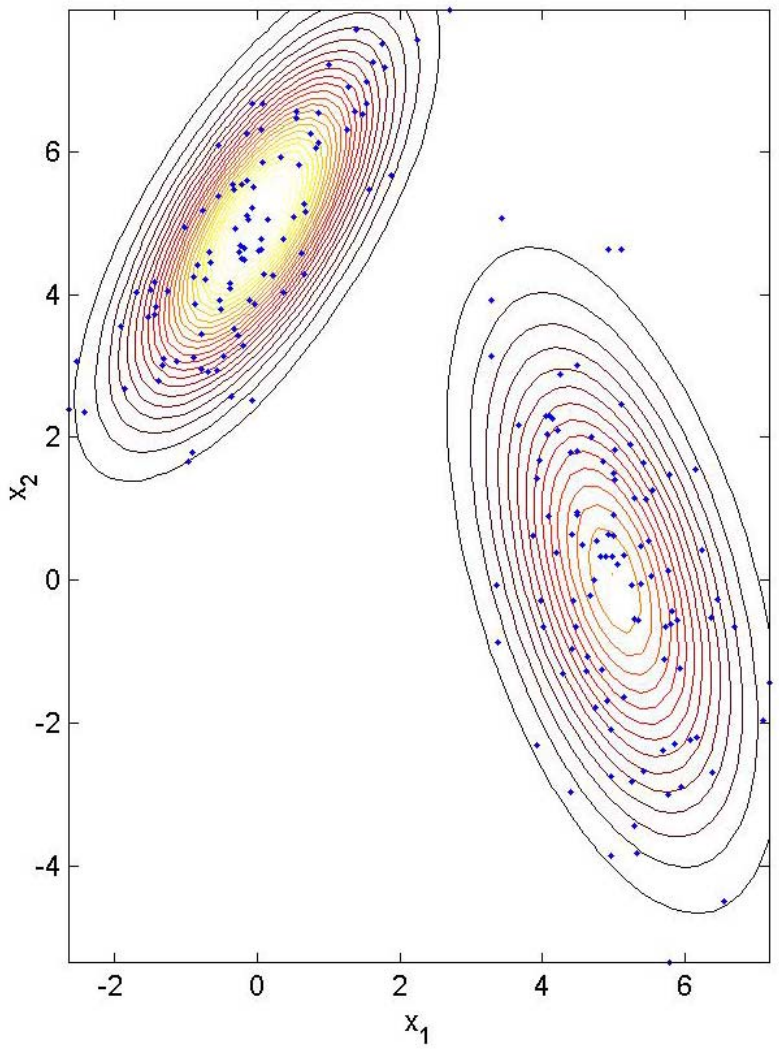
$$\text{with} \begin{cases} \mu_1 = [0 \;\; 5]^T & \Sigma_1 = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \\ \mu_2 = [5 \;\; 0]^T & \Sigma_2 = \begin{bmatrix} 1 & -1 \\ -1 & 4 \end{bmatrix} \end{cases}$$



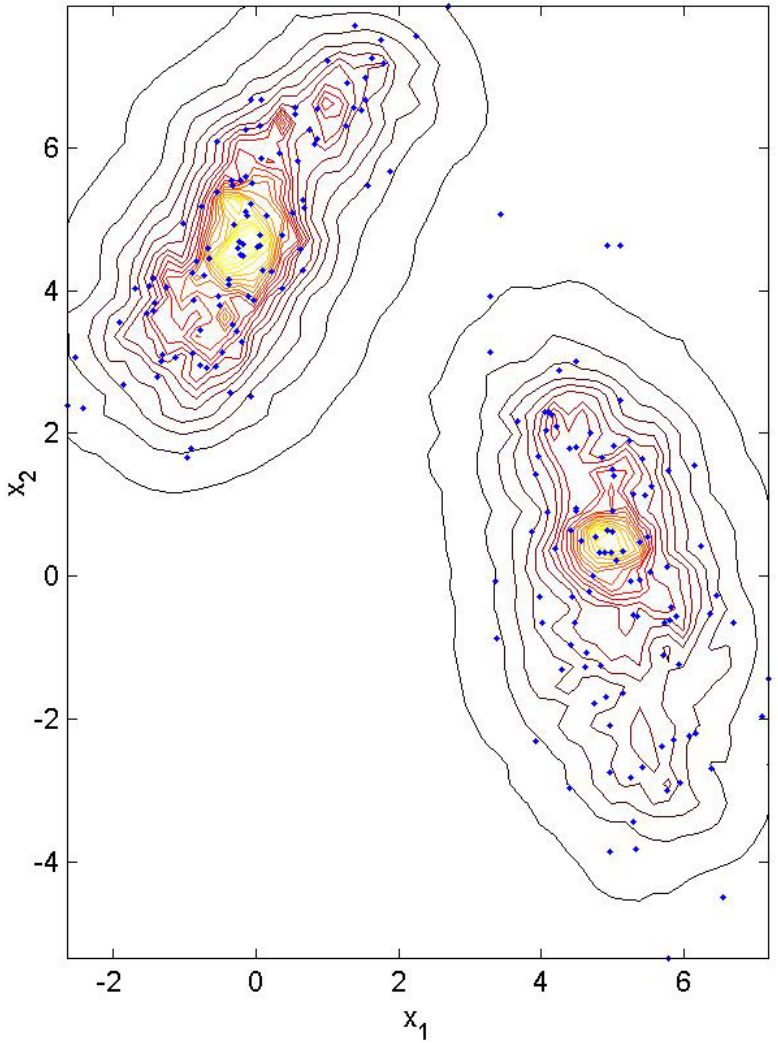
  - The bottom figure shows the density estimate for k=10 neighbors and N=200 examples
  - In the next slide we show the contours of the two distributions overlapped with the training data used to generate the estimate

kNN Density Estimation, example 2 (b)

True density contours



kNN density estimate contours



kNN Density Estimation as a Bayesian classifier

- **The main advantage of the kNN method is that it leads to a very simple approximation of the (optimal) Bayes classifier**
  - Assume that we have a dataset with N examples, N_i from class ω_i, and that we are interested in classifying an unknown sample x_u
    - We draw a hyper-sphere of volume V around x_u. Assume this volume contains a total of k examples, k_i from class ω_i.
  - We can then approximate the likelihood functions using the kNN method by:

$$P(x|\omega_i) = \frac{k_i}{N_i V}$$

  - Similarly, the unconditional density is estimated by

$$P(x) = \frac{k}{NV}$$

  - And the priors are approximated by

$$P(\omega_i) = \frac{N_i}{N}$$

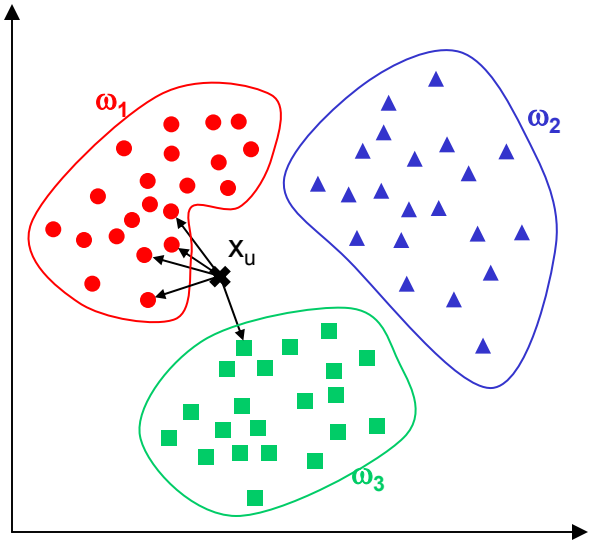
  - Putting everything together, the Bayes classifier becomes

$$P(\omega_i|x) = \frac{P(x|\omega_i)P(\omega_i)}{P(x)} = \frac{\frac{k_i}{N_i V}\cdot\frac{N_i}{N}}{\frac{k}{NV}} = \frac{k_i}{k}$$

From [Bishop, 1995]

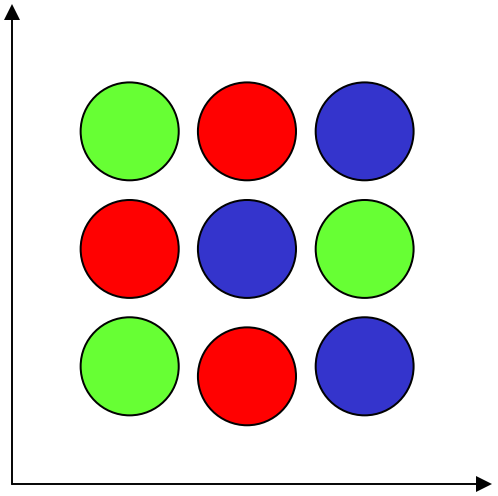
The k Nearest Neighbor classification rule

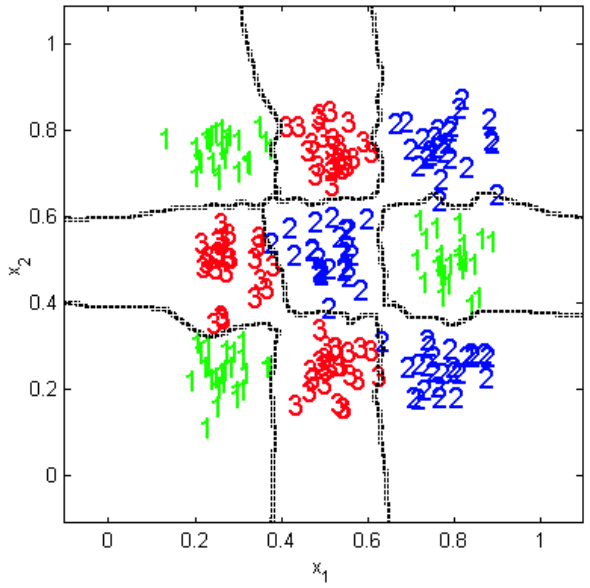
- **The K Nearest Neighbor Rule (kNN) is a very intuitive method that classifies unlabeled examples based on their similarity to examples in the training set**
  - **For a given unlabeled example $x_u \in \Re^D$, find the k "closest" labeled examples in the training data set and assign x_u to the class that appears most frequently within the k-subset**
- **The kNN 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 x_u is assigned to ω_1, the predominant class

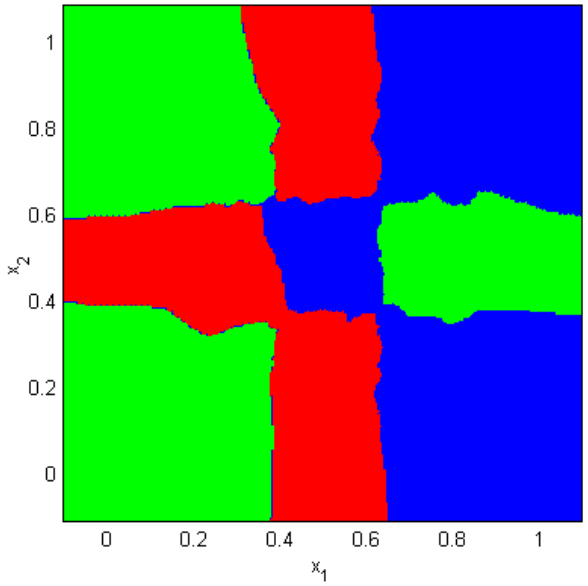


kNN in action: example 1

- **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 kNN rule with**
  - k = 5
  - The Euclidean distance as a metric
- **The resulting decision boundaries and decision regions are shown below**

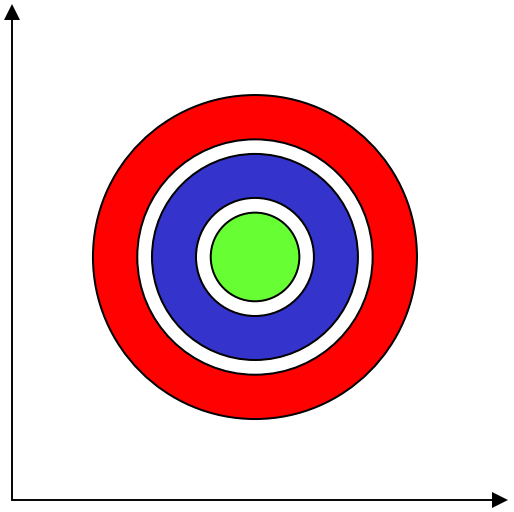


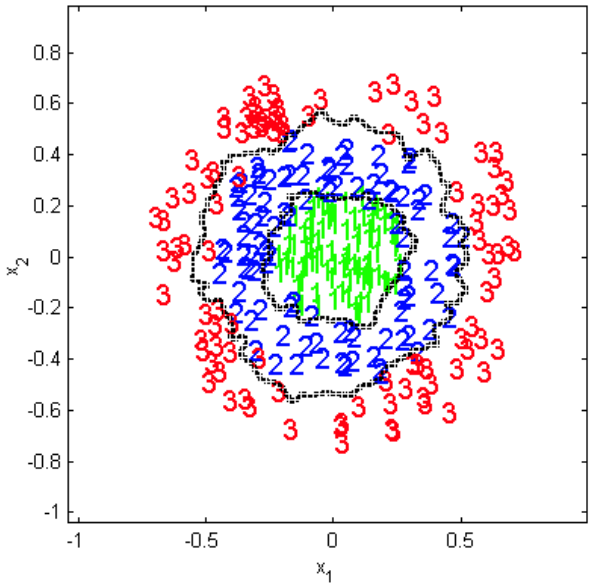


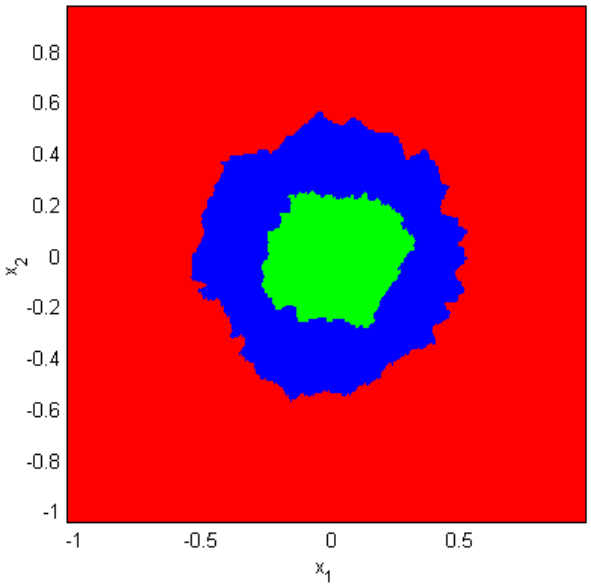


kNN in action: example 2

- 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 kNN rule with
  - k = 5
  - The Euclidean distance as a metric
- The resulting decision boundaries and decision regions are shown below







kNN as a lazy (machine learning) algorithm

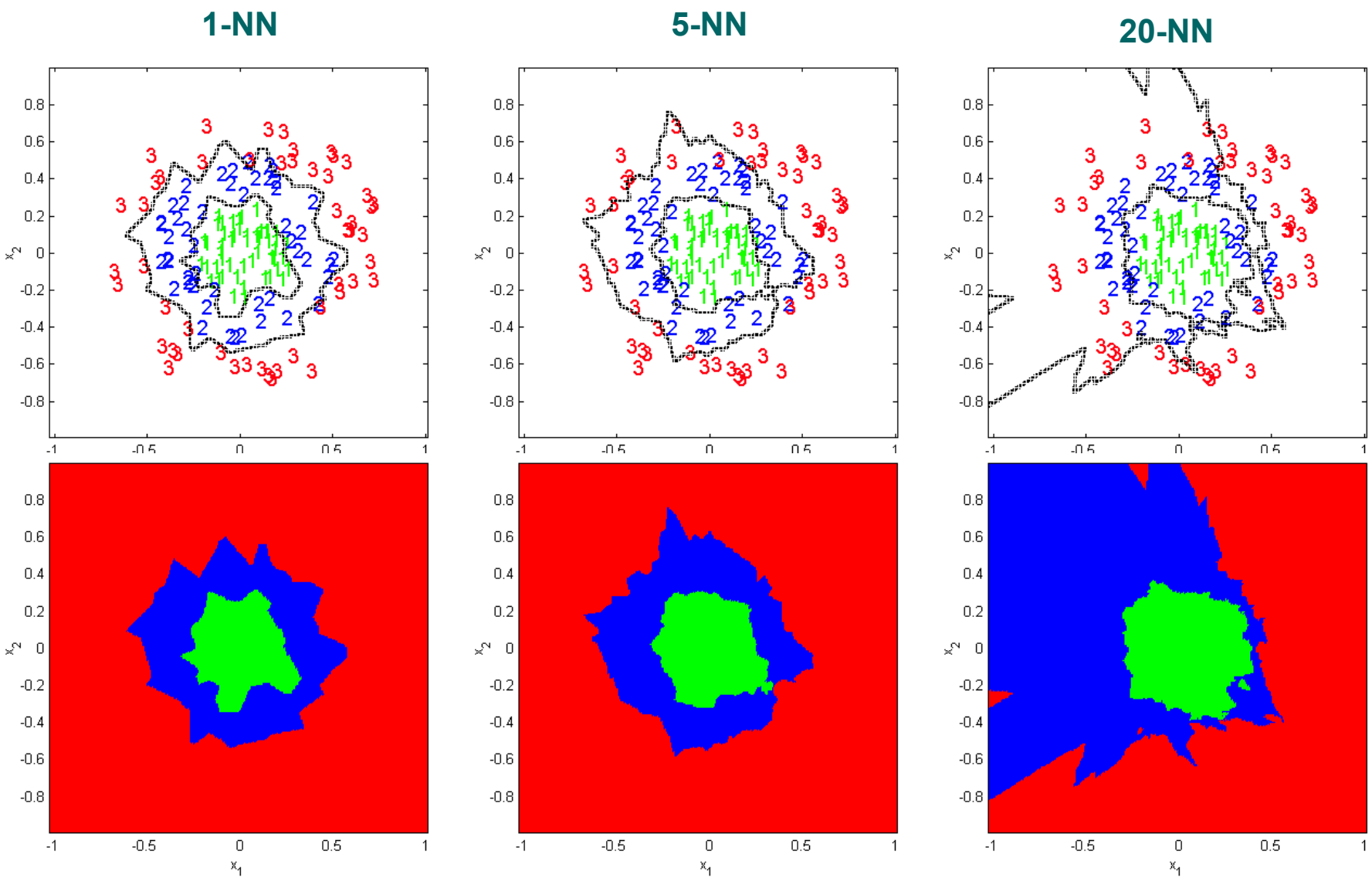
- **kNN is considered a lazy learning algorithm**
  - **Defers** data processing until it receives a request to classify an unlabelled example
  - Replies to a request for information by **combining** its stored training data
  - **Discards** the constructed answer and any intermediate results
- **Other 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

From [Aha, 1997]

Characteristics of the kNN classifier

- **Advantages**
  - Analytically tractable
  - Simple implementation
  - Nearly optimal in the large sample limit ($N \rightarrow \infty$)
    - $P[error]_{Bayes} < P[error]_{1NN} < 2P[error]_{Bayes}$
  - 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
- **1NN versus kNN**
  - 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 a value of k is detrimental
    - It destroys the locality of the estimation since farther examples are taken into account
    - In addition, it increases the computational burden

kNN versus 1NN



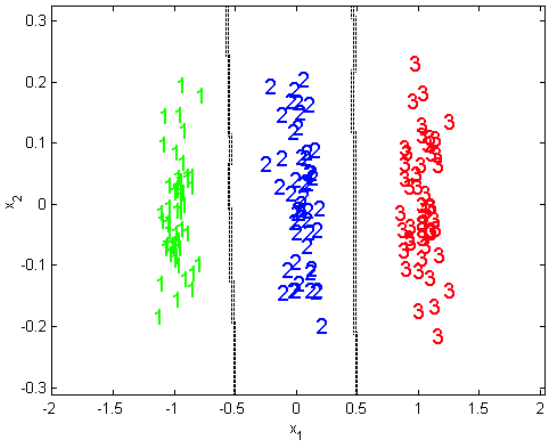
Optimizing storage requirements

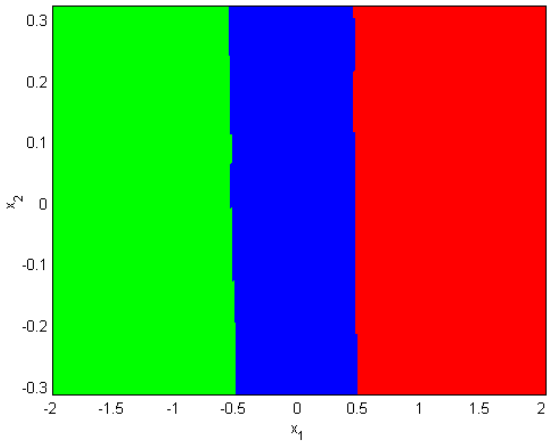
- **The basic kNN algorithm stores all the examples in the training set, creating high storage requirements (and computational cost)**
  - However, the entire training set need not be stored since the examples may contain information that is highly redundant
    - A degenerate case is the earlier example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

kNN and the problem of feature weighting

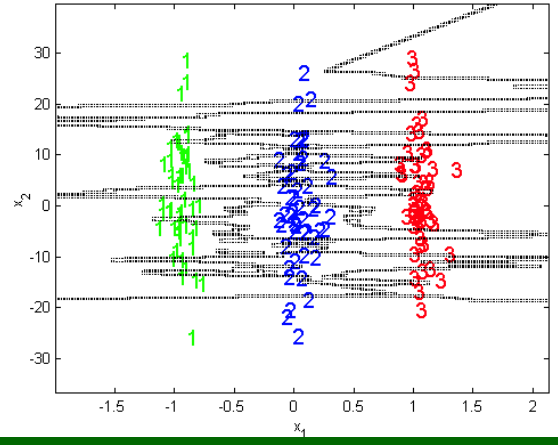
- **The basic kNN rule computes the similarity measure based on the Euclidean distance**
  - This metric makes the kNN rule very sensitive to noisy features
  - As an example, we have created a data set with three classes and two dimensions
    - The first axis contains all the discriminatory information.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

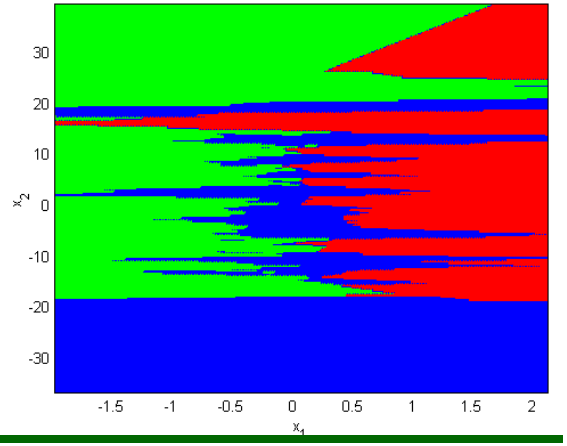
- **In the first example, both axes are scaled properly**
  - The kNN (k=5) finds decision boundaries fairly close to the optimal





- **In the second example, the magnitude of the second axis has been increased two order of magnitudes (see axes tick marks)**
  - The kNN 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 kNN 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 if 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**

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

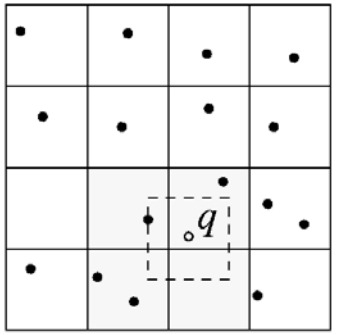
  - Note that this procedure is identical to performing a linear transformation where the transformation matrix is diagonal with the weights placed in the diagonal elements
    - From this perspective, feature weighting can be thought of as a special case of feature extraction where the different features are not allowed to interact (null off-diagonal elements in the transformation matrix)
    - Feature subset selection, which will be covered later in 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 kNN variant that weights the contribution of each of the k nearest neighbors according to their distance to the unlabeled example
    - Distance-weighting distorts the kNN estimate of $P(\omega_i|x)$ and is NOT recommended
    - Studies have shown that distance-weighting DOES NOT improve kNN 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-determined function that measures the information content of each feature (i.e., mutual information and correlation between each feature and the class label)
  - These 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

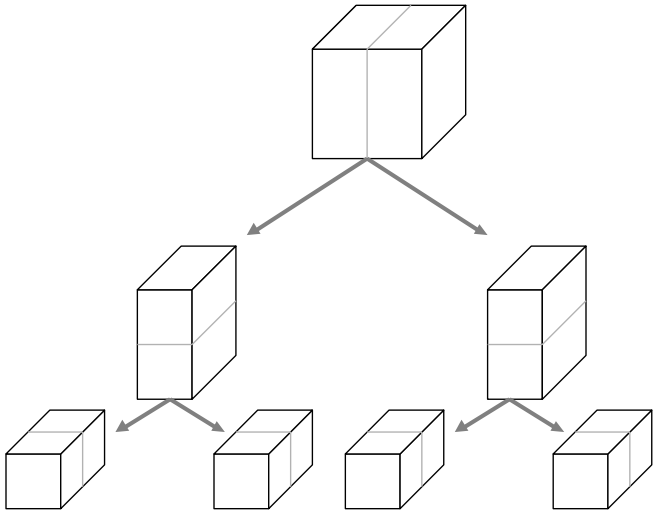
- **The problem of nearest neighbor can be stated as follows**
  - Given a set of N points in D-dimensional space and an unlabeled example $x_u \in \Re^D$, find the point that minimizes the distance to x_u
  - The naïve approach of computing a set of N distances, and finding the (k) smallest becomes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