

Nonparametric Density Estimation
Nearest Neighbors , KNN

k-Nearest Neighbors

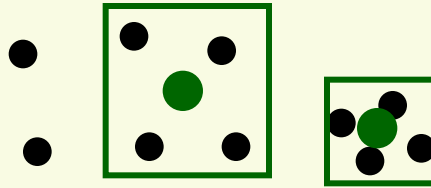
- Recall the generic expression for density estimation

$$p(\mathbf{x}) \approx \frac{k/n}{V}$$

- In Parzen windows estimation, we fix V and that determines k , the number of points inside V
- In k-nearest neighbor approach we fix k , and find V that contains k points inside

k-Nearest Neighbors

- kNN approach seems a good solution for the problem of the “best” window size
 - Let the cell volume be a function of the training data
 - Center a cell about x and let it grow until it captures k samples
 - k are called the k nearest-neighbors of x

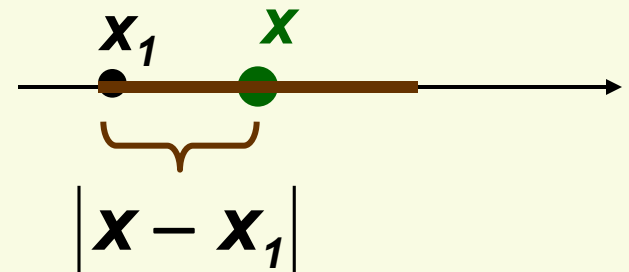


- 2 possibilities can occur:
 - Density is high near x ; therefore the cell will be small which provides a good resolution
 - Density is low; therefore the cell will grow large and stop until higher density regions are reached

k-Nearest Neighbor

- Of course, now we have a new question
 - How to choose k ?
- A good “rule of thumb” is $k = \sqrt{n}$
 - Can prove convergence if n goes to infinity
 - Not too useful in practice, however
- Let’s look at 1-D example
 - we have one sample, i.e. $n = 1$

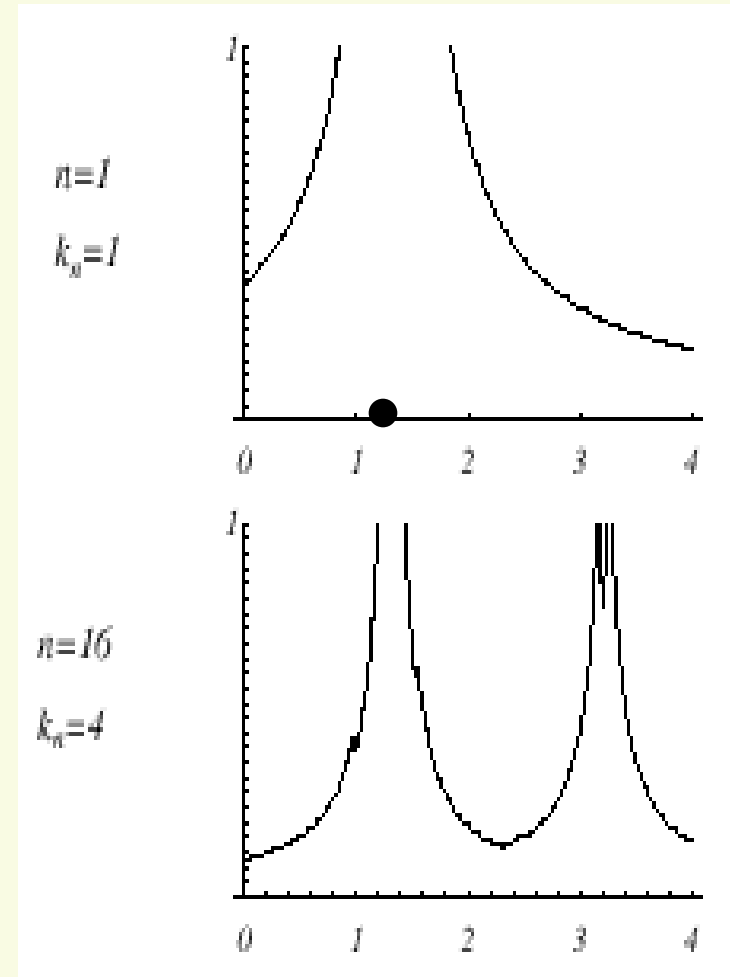
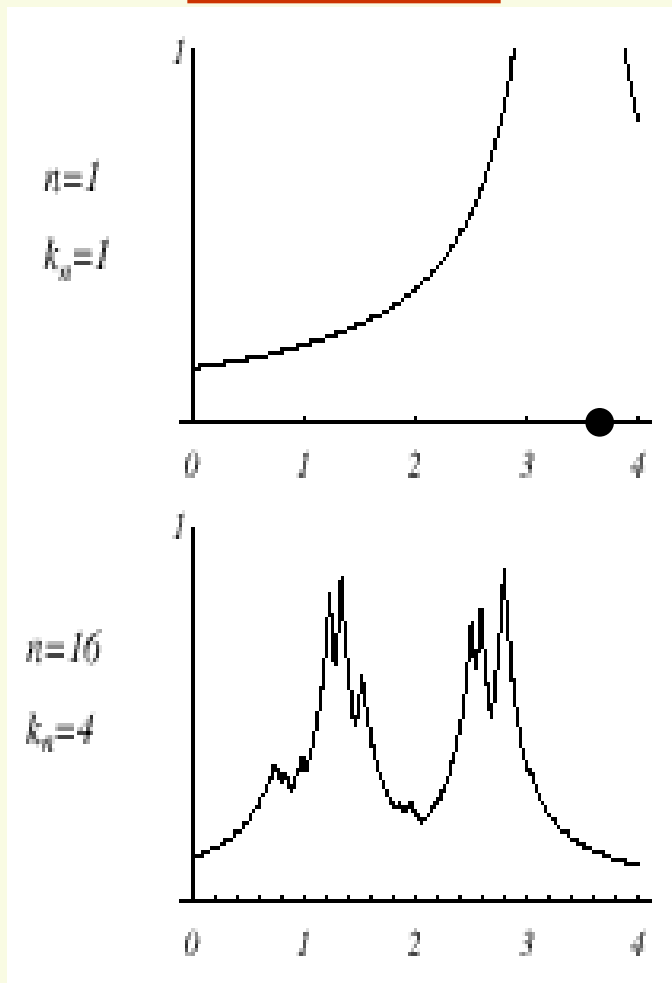
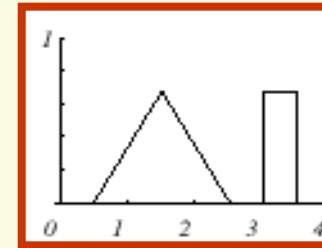
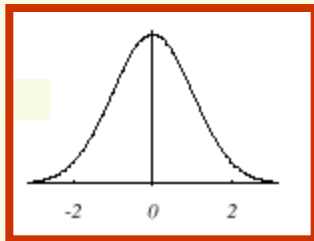
$$p(\mathbf{x}) \approx \frac{k/n}{V} = \frac{1}{2|\mathbf{x} - \mathbf{x}_1|}$$



- But the estimated $p(\mathbf{x})$ is not even close to a density function:

$$\int_{-\infty}^{\infty} \frac{1}{2|\mathbf{x} - \mathbf{x}_1|} d\mathbf{x} = \infty \neq 1$$

k-Nearest Neighbor: Density estimation



k-Nearest Neighbor

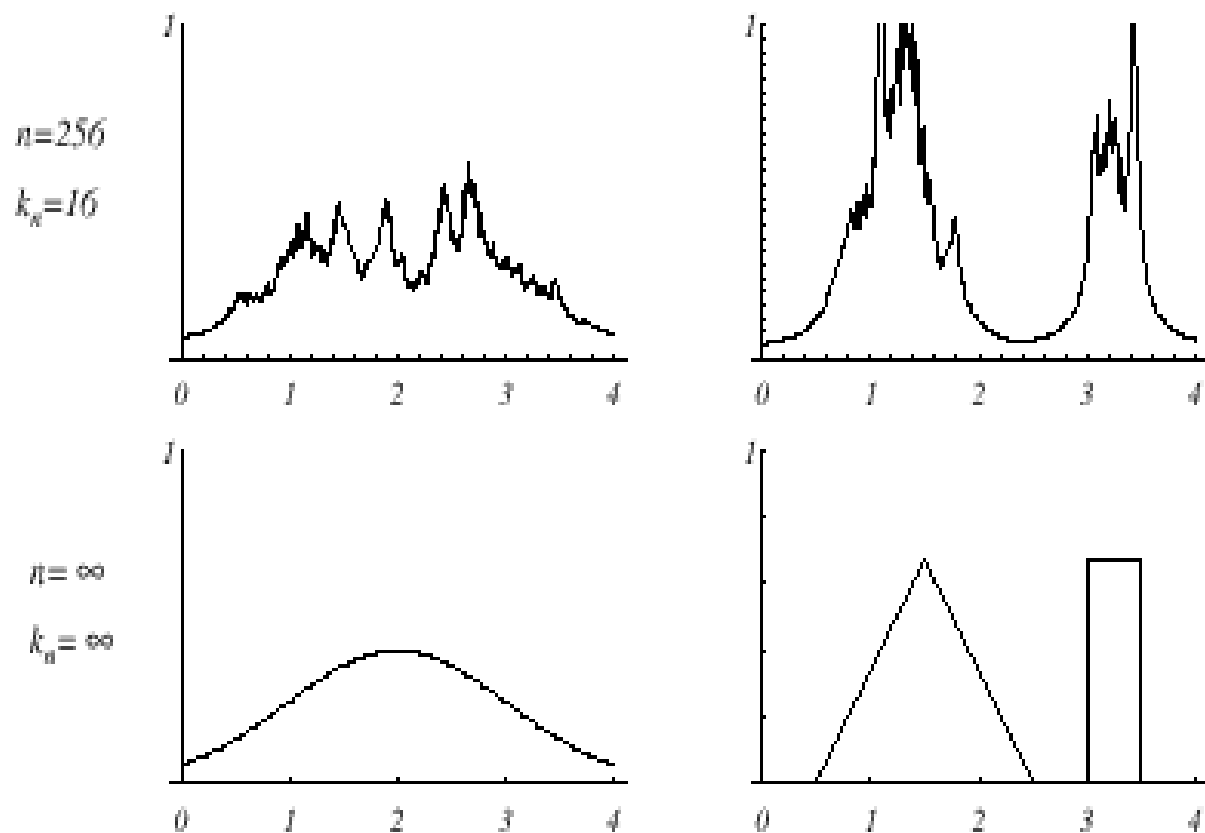


FIGURE 4.12. Several k -nearest-neighbor estimates of two unidimensional densities: a Gaussian and a bimodal distribution. Notice how the finite n estimates can be quite “spiky.” From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

k-Nearest Neighbor

- Thus straightforward density estimation $p(\mathbf{x})$ does not work very well with kNN approach because the resulting density estimate
 1. Is not even a density
 2. Has a lot of discontinuities (looks very spiky, not differentiable)
 3. Even for large regions with no observed samples the estimated density is far from zero (tails are too heavy)
- *Notice in the theory, if infinite number of samples is available, we could construct a series of estimates that converge to the true density using kNN estimation. However this theorem is not very useful in practice because the number of samples is always limited*

k-Nearest Neighbor

- However we shouldn't give up the nearest neighbor approach yet
- Instead of approximating the density $\mathbf{p}(\mathbf{x})$, we can use kNN method to approximate the posterior distribution $\mathbf{P}(\mathbf{c}_i|\mathbf{x})$
 - We don't need $\mathbf{p}(\mathbf{x})$ if we can get a good estimate on $\mathbf{P}(\mathbf{c}_i|\mathbf{x})$

k-Nearest Neighbor

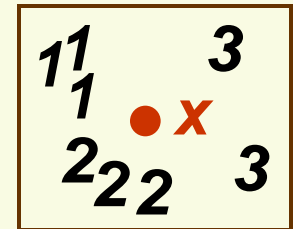
- How would we estimate $P(\mathbf{c}_i | \mathbf{x})$ from a set of n labeled samples?

- Recall our estimate for density: $p(\mathbf{x}) \approx \frac{k/n}{V}$

- Let's place a cell of volume V around \mathbf{x} and capture k samples

- k_i samples amongst k labeled \mathbf{c}_i then:

$$p(\mathbf{c}_i, \mathbf{x}) \approx \frac{k_i/n}{V}$$



- Using conditional probability, let's estimate posterior:

$$p(\mathbf{c}_i | \mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{c}_i)}{p(\mathbf{x})} = \frac{p(\mathbf{x}, \mathbf{c}_i)}{\sum_{j=1}^m p(\mathbf{x}, \mathbf{c}_j)} \approx \frac{k_i/n}{V \sum_{j=1}^m \frac{k_j/n}{V}} = \frac{k_i}{\sum_{j=1}^m k_j} = \frac{k_i}{k}$$

k-Nearest Neighbor Rule

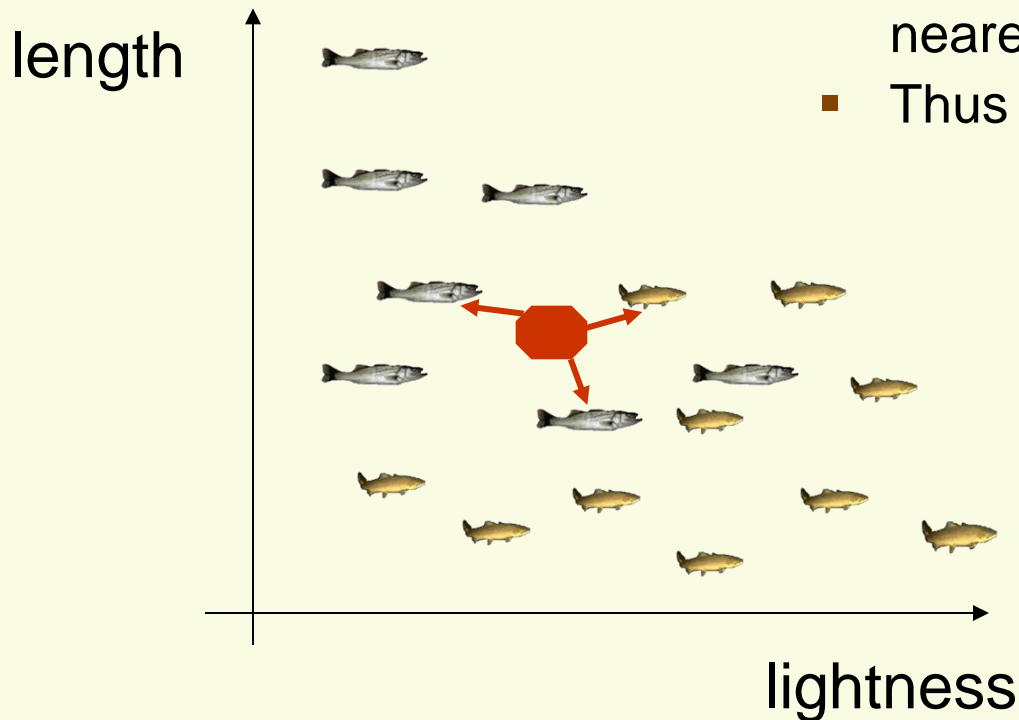
- Thus our estimate of posterior is just the fraction of samples which belong to class \mathbf{c}_i :

$$p(\mathbf{c}_i | \mathbf{x}) = \frac{k_i}{k}$$

- This is a very simple and intuitive estimate
- Under the zero-one loss function (MAP classifier) just choose the class which has the largest number of samples in the cell
- Interpretation is: given an unlabeled example (that is \mathbf{x}), find k most similar labeled examples (closest neighbors among sample points) and assign the most frequent class among those neighbors to \mathbf{x}

k-Nearest Neighbor: Example

- Back to fish sorting
 - Suppose we have 2 features, and collected sample points as in the picture
 - Let $k = 3$



- 2 sea bass, 1 salmon are the 3 nearest neighbors
- Thus classify as sea bass

***k**NN: How Well Does it Work?*

- kNN rule is certainly simple and intuitive, but does it work?
- Assume we have an unlimited number of samples
- By definition, the best possible error rate is the Bayes rate E^*
- Nearest-neighbor rule leads to an error rate greater than E^*
- But even for $k=1$, as $n \rightarrow \infty$, it can be shown that nearest neighbor rule error rate is smaller than $2E^*$
- As we increase k , the upper bound on the error gets better and better, that is the error rate (as $n \rightarrow \infty$) for the **kNN** rule is smaller than cE^* , with smaller c for larger k
- If we have a lot of samples, the kNN rule will do very well !

1NN: Voronoi Cells

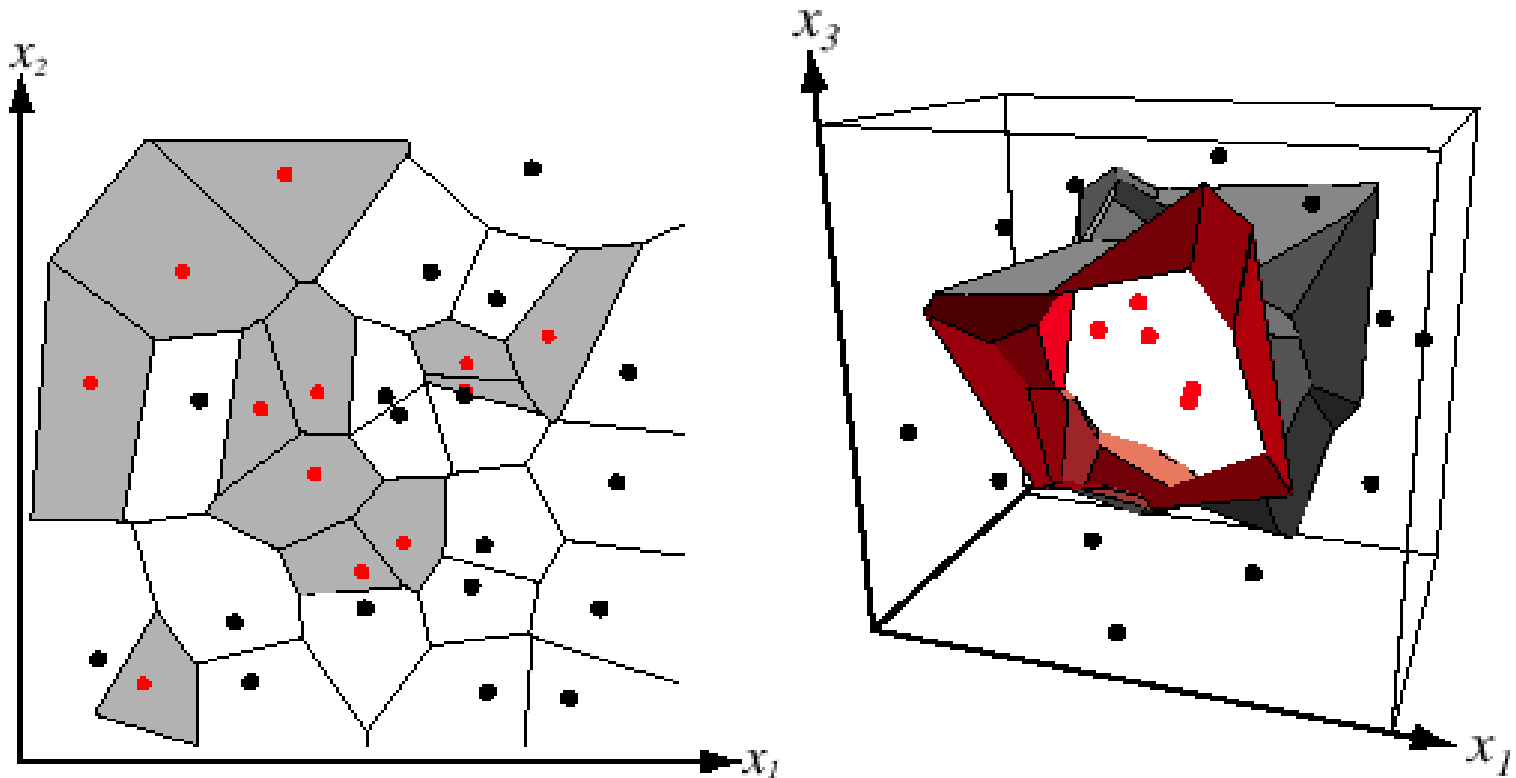
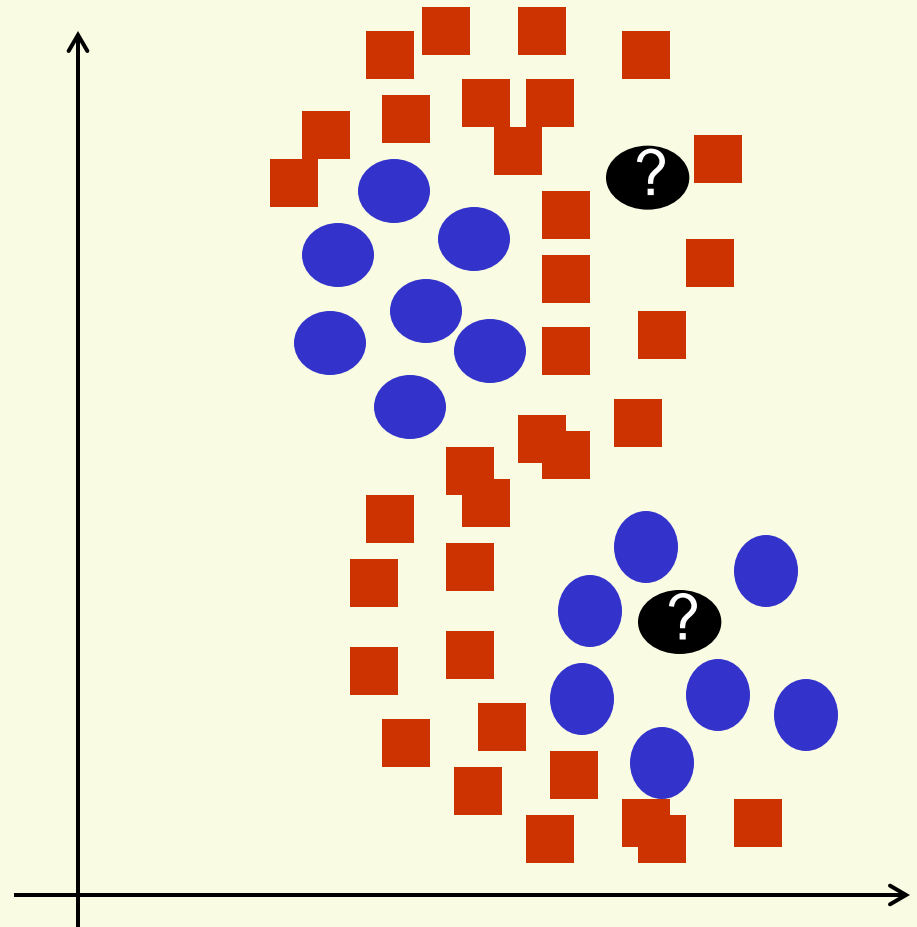


FIGURE 4.13. In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

*k*NN: Multi-Modal Distributions

- Most parametric distributions would not work for this 2 class classification problem:
- Nearest neighbors will do reasonably well, provided we have a lot of samples



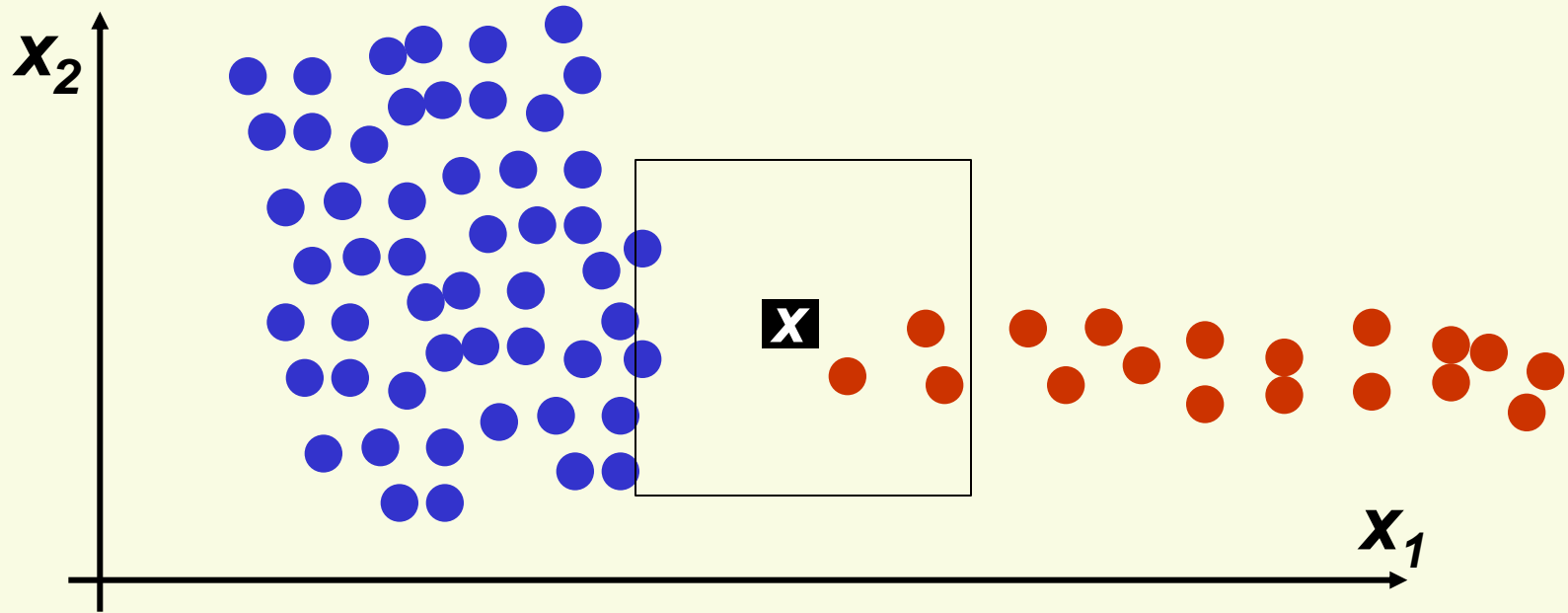
***k**NN: How to Choose **k**?*

- In theory, when the infinite number of samples is available, the larger the **k**, the better is classification (error rate gets closer to the optimal Bayes error rate)
- But the caveat is that all **k** neighbors have to be close to **x**
 - Possible when infinite # samples available
 - Impossible in practice since # samples is finite

***k**NN: How to Choose **k**?*

- In practice
 - 1.** **k** should be large so that error rate is minimized
 - **k** too small will lead to noisy decision boundaries
 - 2.** **k** should be small enough so that only nearby samples are included
 - **k** too large will lead to over-smoothed boundaries
- Balancing **1** and **2** is not trivial
 - This is a recurrent issue, need to smooth data, but not too much

*k*NN: How to Choose *k*?



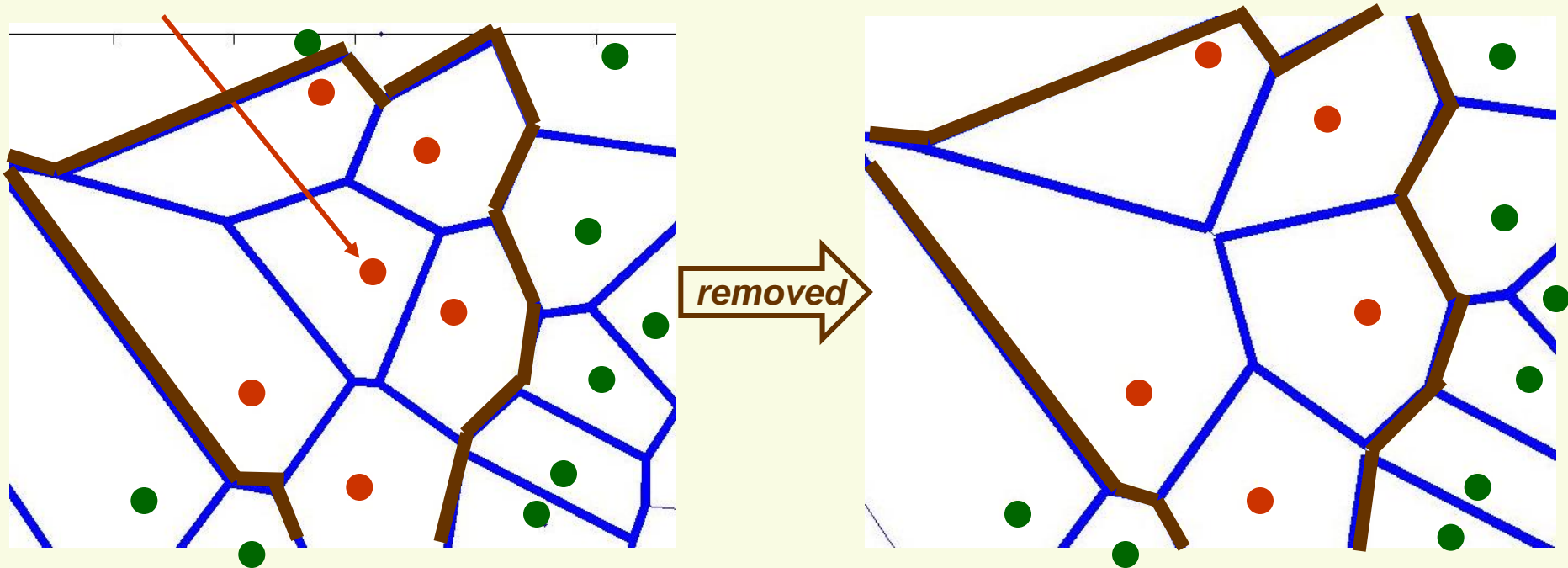
- For $k = 1, \dots, 5$ point x gets classified correctly
 - red class
- For larger k classification of x is wrong
 - blue class

kNN: Computational Complexity

- Basic ***kNN*** algorithm stores all examples. Suppose we have ***n*** examples each of dimension ***d***
 - $O(d)$ to compute distance to one example
 - $O(nd)$ to find one nearest neighbor
 - $O(knd)$ to find ***k*** closest examples
 - Thus complexity is $O(knd)$
- This is prohibitively expensive for large number of samples
- But we need large number of samples for ***kNN*** to work well!

Reducing Complexity: Editing 1NN

- If all voronoi neighbors have the same class, a sample is useless, we can remove it:



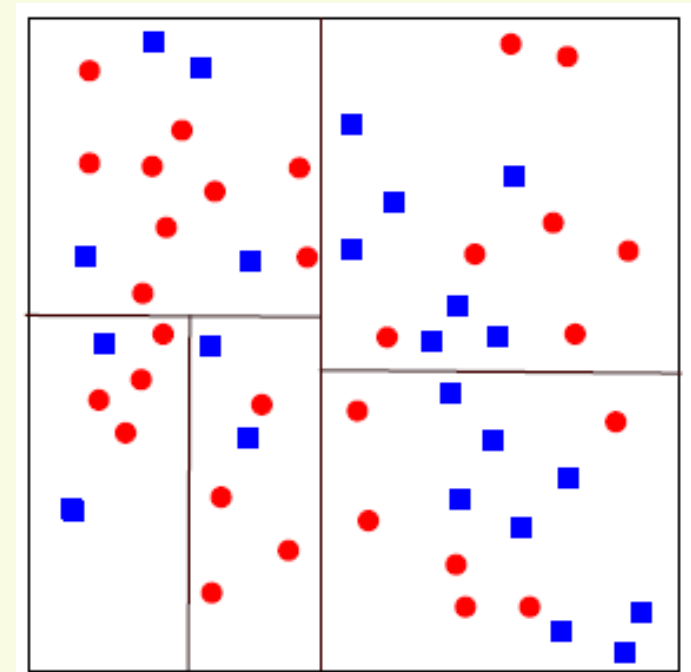
- Number of samples decreases
- We are guaranteed that the decision boundaries stay the same

Reducing the complexity of KNN

- **Idea:** Partition space recursively and search for NN only close to the test point
- **Preprocessing:** Done prior to classification process.

Axis-parallel tree construction:

1. Split space in direction of largest 'spread' into two equi-numbered cells
2. Repeat procedure recursively for each subcell, until some stopping criterion is achieved



Reducing the complexity of KNN

■ Classification:

1. Propagate a test point down the tree. Classification is based on NN from the final leaf reached.
2. If NN (within leaf) is further than nearest boundary - retrack

■ Notes:

- Clearly $\log n$ layers (and distance computations) suffice.
- Computation time to build tree: $O(dn \log n)$ (offline)
- Many variations and improvements exist (e.g. diagonal splits)
- Stopping criterion: often ad-hoc (e.g. number of points in leaf region is k , region size, etc.)

kNN: Selection of Distance

- So far we assumed we use Euclidian Distance to find the nearest neighbor:

$$D(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_k (\mathbf{a}_k - \mathbf{b}_k)^2}$$

- However some features (dimensions) may be much more discriminative than other features (dimensions)
- Euclidian distance treats each feature as equally important

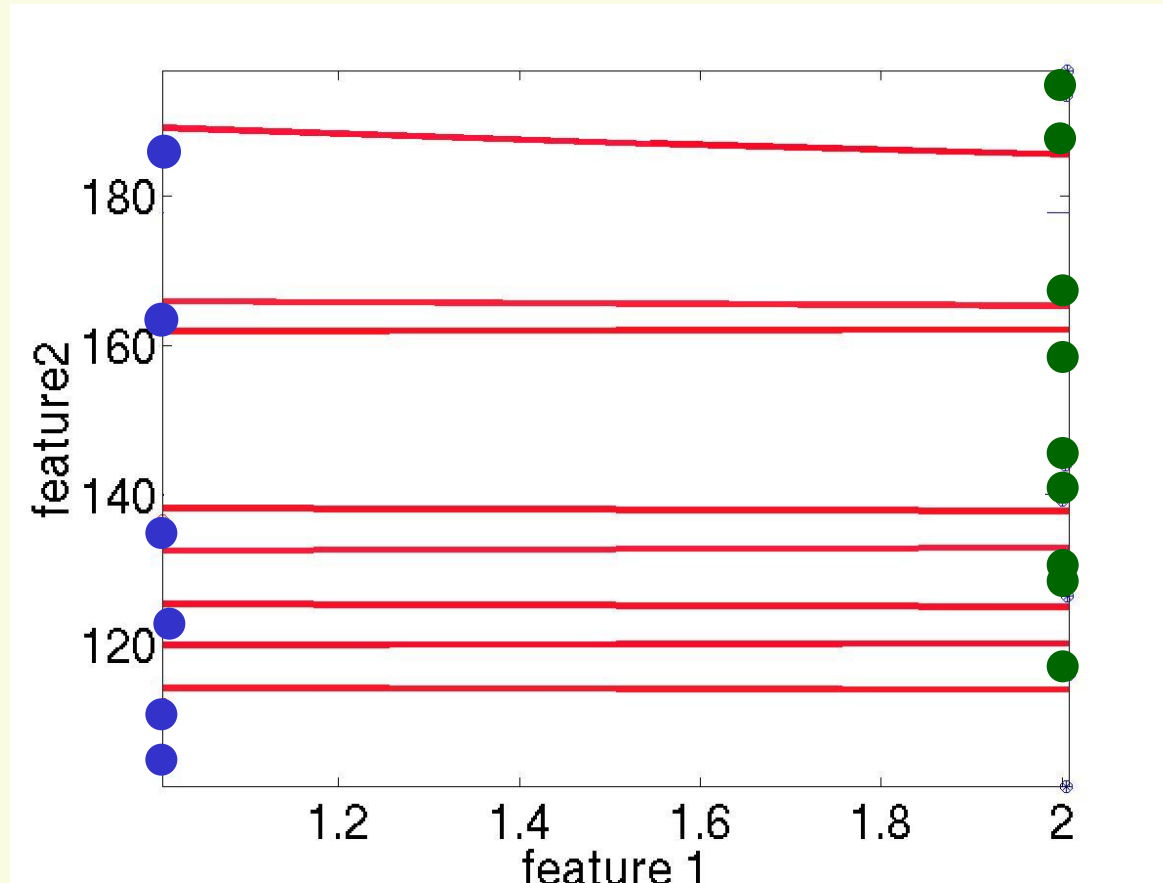
*k*NN: Selection of Distance

- Extreme Example
 - feature 1 gives the correct class: 1 or 2
 - feature 2 gives irrelevant number from 100 to 200
- Suppose we have to find the class of $x=[1 \ 100]$ and we have 2 samples $[1 \ 150]$ and $[2 \ 110]$

$$D\left(\begin{bmatrix} 1 \\ 100 \end{bmatrix}, \begin{bmatrix} 1 \\ 150 \end{bmatrix}\right) = \sqrt{(1-1)^2 + (100-150)^2} = 50 \quad D\left(\begin{bmatrix} 1 \\ 100 \end{bmatrix}, \begin{bmatrix} 2 \\ 110 \end{bmatrix}\right) = \sqrt{(1-2)^2 + (100-110)^2} = 10.5$$

- $x = [1 \ 100]$ is misclassified!
- The denser the samples, the less of the problem
 - But we rarely have samples dense enough

kNN: Extreme Example of Distance Selection

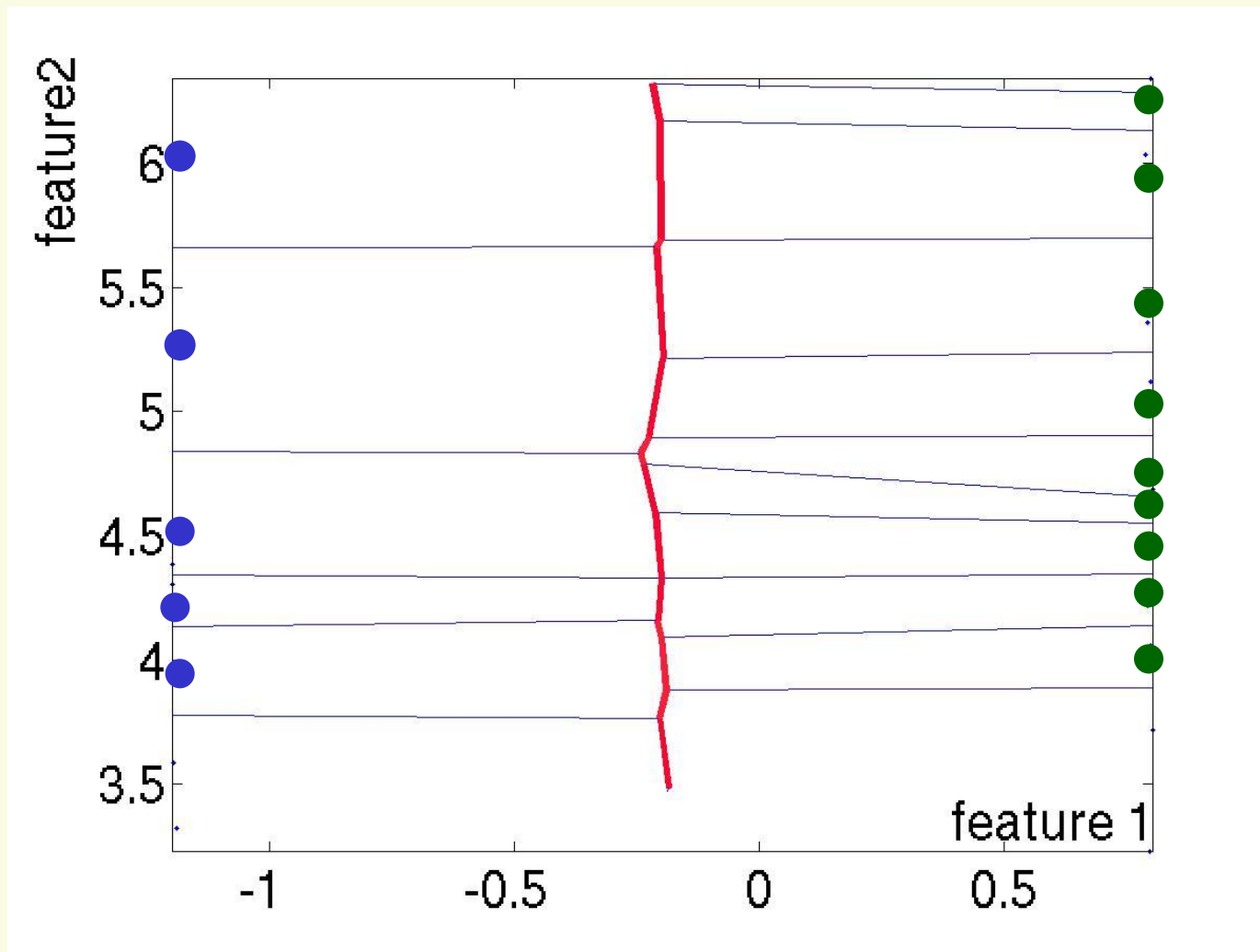


- decision boundaries for blue and green classes are in red
- These boundaries are really bad because
 - feature 1 is discriminative, but its scale is small
 - feature 2 gives no class information (noise) but its scale is large

*k*NN: Selection of Distance

- Notice the 2 features are on different scales:
 - feature 1 takes values between 1 or 2
 - feature 2 takes values between 100 to 200
- We could normalize each feature to be between of mean 0 and variance 1
- If \mathbf{X} is a random variable of mean μ and variance σ^2 , then $(\mathbf{X} - \mu)/\sigma$ has mean 0 and variance 1
- Thus for each feature vector \mathbf{x}_i , compute its sample mean and variance, and let the new feature be $[\mathbf{x}_i - \text{mean}(\mathbf{x}_i)]/\text{sqrt}[\text{var}(\mathbf{x}_i)]$
- Let's do it in the previous example

*k*NN: Normalized Features



- The decision boundary (in red) is very good now!

kNN: Selection of Distance

- However in high dimensions if there are a lot of irrelevant features, normalization will not help

$$D(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_k (\mathbf{a}_k - \mathbf{b}_k)^2} = \sqrt{\underbrace{\sum_i (\mathbf{a}_i - \mathbf{b}_i)^2}_{\text{discriminative feature}} + \underbrace{\sum_j (\mathbf{a}_j - \mathbf{b}_j)^2}_{\text{noisy features}}}$$

- If the number of discriminative features is smaller than the number of noisy features, Euclidean distance is dominated by noise

kNN: Feature Weighting

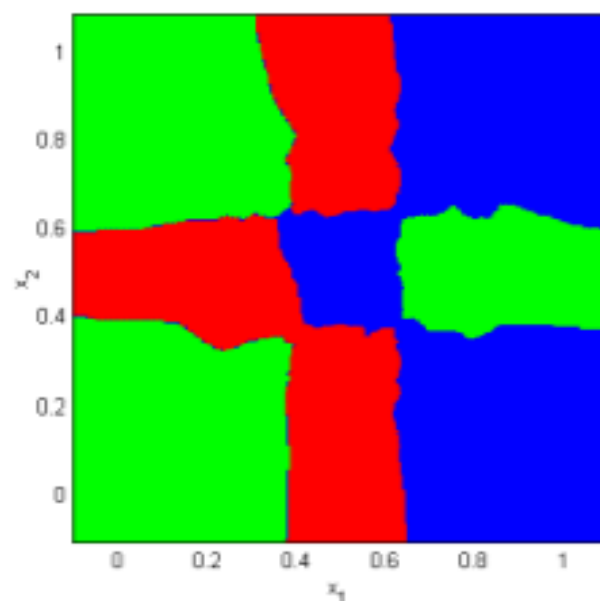
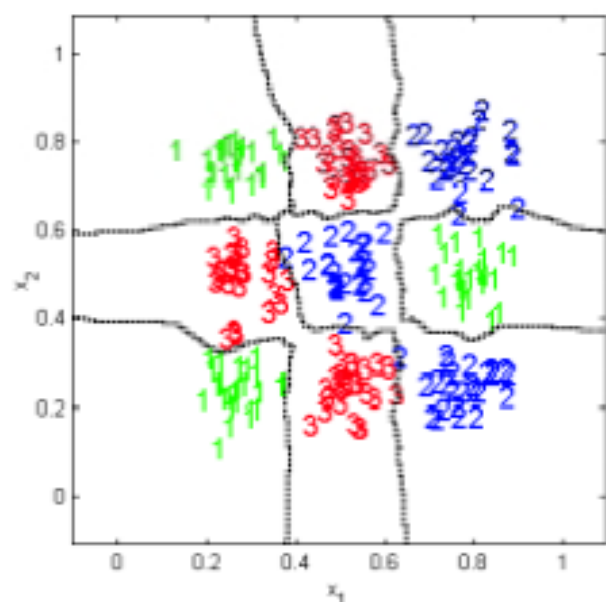
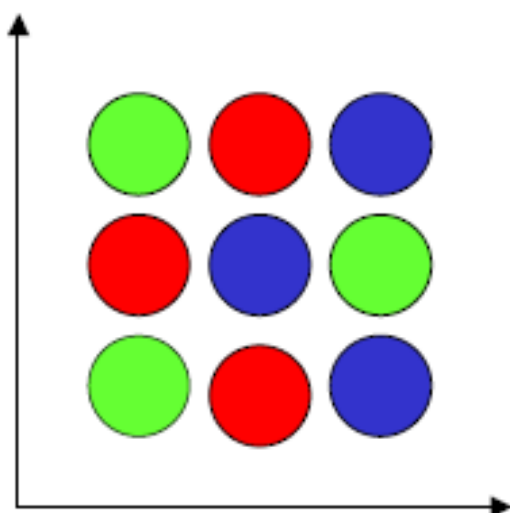
- Scale each feature by its importance for classification

$$D(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_k \mathbf{w}_k (\mathbf{a}_k - \mathbf{b}_k)^2}$$

- Can learn the weights \mathbf{w}_k from the validation data
 - Increase/decrease weights until classification improves

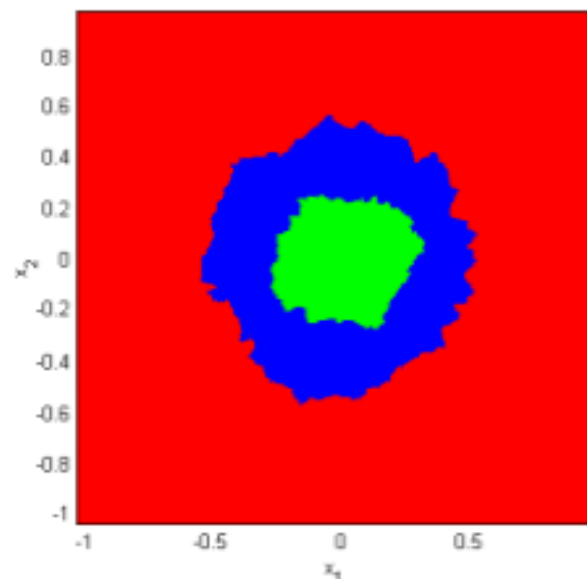
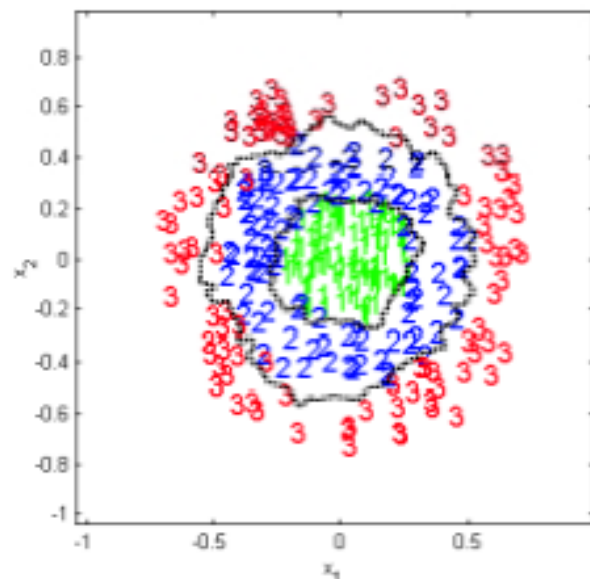
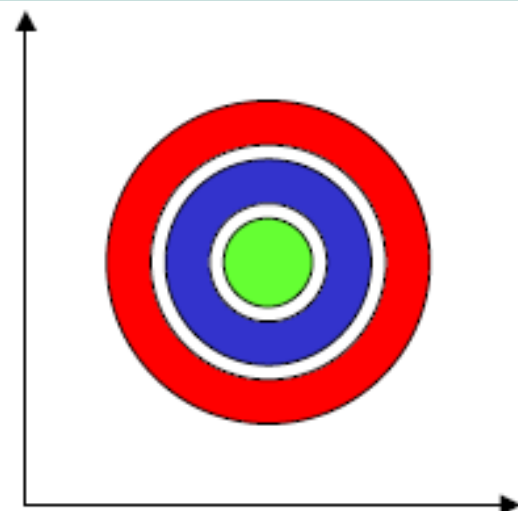
k-NNR in action: example 1

- 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



kNN Summary

- Advantages
 - Can be applied to the data from any distribution
 - Very simple and intuitive
 - Good classification if the number of samples is large enough
- Disadvantages
 - Choosing best k may be difficult
 - Computationally heavy, but improvements possible
 - Need large number of samples for accuracy
 - Can never fix this without assuming parametric distribution