Lecture 6

- **Nearest Neighbors and Non-parametric Density Estimation**
  - Simple approximation of any probability density function given training data.
- **Nearest Neighbors and Non-Bayesian Classification**
  - A classifier learned directly from labeled training data without estimating any probabilistic structure.

Nearest Neighbors and Non-parametric Density Estimation

Density estimation

Bayesian Decision Theory has shown us how to design an optimal classifier if we know the prior probabilities $P(\omega_i)$ and the class-conditional densities $P(\mathbf{x}|\omega_i)$.

- Unfortunately, we rarely have complete knowledge of these class-conditional densities or the prior probabilities.
- However, we can often find training data that include particular representatives of the patterns we want to classify.

Density estimation

There are two general approaches:

Parametric Assume some parametric form for the conditional densities (e.g., multi-variate Gaussian $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$) and estimate its parameters using training data. Then use the resulting estimates as if they were the true values and perform classification using the Bayesian decision rule.

Non-parametric Make no assumptions about the form of the underlying class-conditionals and estimate them completely from the training data.

Why non-parametric ?

- Common parametric forms do not always fit the densities actually encountered in practice.
- In addition, most of the classical parametric densities are unimodal, whereas many practical problems involve multi-modal densities.
- Non-parametric methods can be used with arbitrary distributions and without the assumption that the forms of the underlying densities are known.

Non-parametric density estimation

- Suppose N samples $\mathbf{x}_1, \cdots, \mathbf{x}_N$ are drawn i.i.d. (independently and identically distributed) from a probability density function $p(\mathbf{x})$.
- The probability, $P_{\mathcal{R}}$, that a vector $\mathbf{x}$ will fall in a region $\mathcal{R}$ is given by

$$P_{\mathcal{R}} = \int_{\mathbf{x}' \in \mathcal{R}} p(\mathbf{x}') d\mathbf{x}'$$

- The probability k of the N will fall in $\mathcal{R}$ is given by the binomial law

$$P_{\mathcal{R}}^{(k)} = \binom{N}{k} P_{\mathcal{R}}^{k} (1 - P_{\mathcal{R}})^{N-k}$$

- The expected value of k is $\mathrm{E}[k] = N P_{\mathcal{R}}$ and the MLE for $P_{\mathcal{R}}$ is $\frac{k}{N}$.

Non-parametric density estimation

- If we assume that $p(\mathbf{x})$ is continuous and $\mathcal{R}$ is small enough so that $p(\mathbf{x})$ does not vary significantly in it, we can approximate

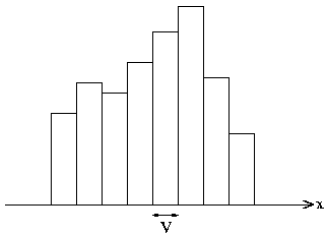
$$\int_{\mathbf{x}' \in \mathcal{R}} p(\mathbf{x}') d\mathbf{x}' \approx p(\mathbf{x}) V$$

  where $\mathbf{x} \in \mathcal{R}$ and V is the volume of $\mathcal{R}$.
- Then the density estimate becomes

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

Histogram Method

- A very simple method is to partition the space into a number of equally-sized cells (bins) and compute a histogram.



Histogram in one dimension.

- The estimate of the density at a point $\mathbf{x}$ becomes

$$p(\mathbf{x}) = \frac{k}{NV}$$

  where N is the total number of samples, k is the number of samples in the cell that includes $\mathbf{x}$, and V is the volume of that cell.

Histogram Method

- Although the histogram method is very easy to implement, it is usually not practical in high-dimensional spaces due to the number of cells.
- Many observations are required to prevent the estimate being zero over a large region.

Non-parametric density estimation

- Remember the general expression for non-parametric density estimation:

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

- Can compute this estimate:
  - Fix the volume V and count the number k of data points inside V
    - This is the approach used for the histogram method
  - Fix the value of k and determine the minimum volume V that encompasses k points in the dataset
    - This gives rise to the k Nearest Neighbor (kNN) approach

kNN Density Estimation

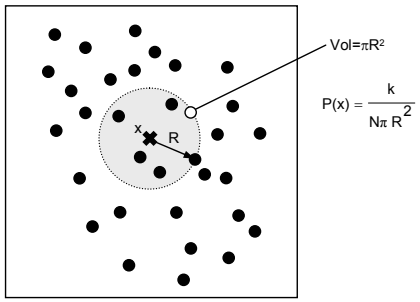
- In the kNN method we grow the volume surrounding the estimation point $\mathbf{x}$ until it encloses a total of k data points
- The density estimate then becomes

$$p(\mathbf{x}) \cong \frac{k}{NV} = \frac{k}{Nc_D R_k^D(\mathbf{x})}$$

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

$$c_D = \begin{cases} \frac{1}{(\frac{D}{2})!}\, \pi^{\frac{D}{2}} & \text{if } D \text{ is even} \\ \frac{1}{(\frac{D}{2})!}\, \pi^{\frac{D-1}{2}}\, 2^n\, (\frac{D-1}{2})! & \text{if } D \text{ is odd} \end{cases}$$

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



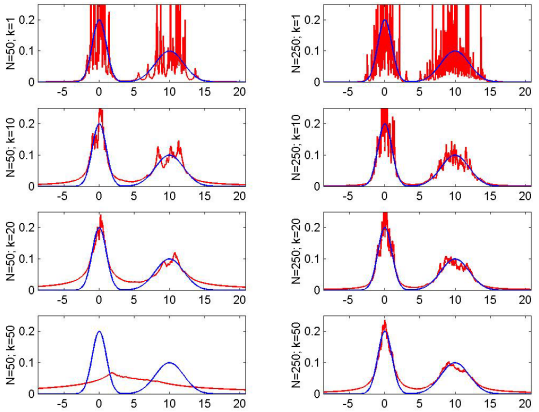
kNN Density Estimation

- 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 $R_k^D(\mathbf{x})$ 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: $P(\mathbf{x}) = \frac{1}{2}\mathcal{N}(0,1) + \frac{1}{2}\mathcal{N}(10,4)$ and several values of N and k.



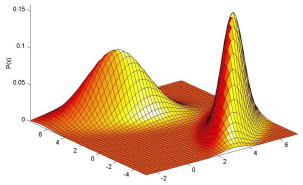
kNN Density Estimation, example 2(a)

- The performance of the kNN density estimation technique on two dimensions is now illustrated
  - Below is the true density, a mixture of two bivariate Gaussians

$$P(\mathbf{x}) = \frac{1}{2}\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1) + \frac{1}{2}\mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)$$

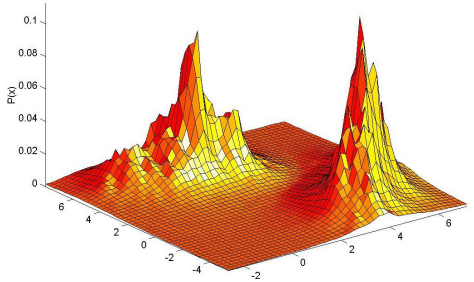
with

$$\boldsymbol{\mu}_1 = (0,5)^T,\ \Sigma_1 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}, \quad \boldsymbol{\mu}_2 = (5,0)^T,\ \Sigma_2 = \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix}$$

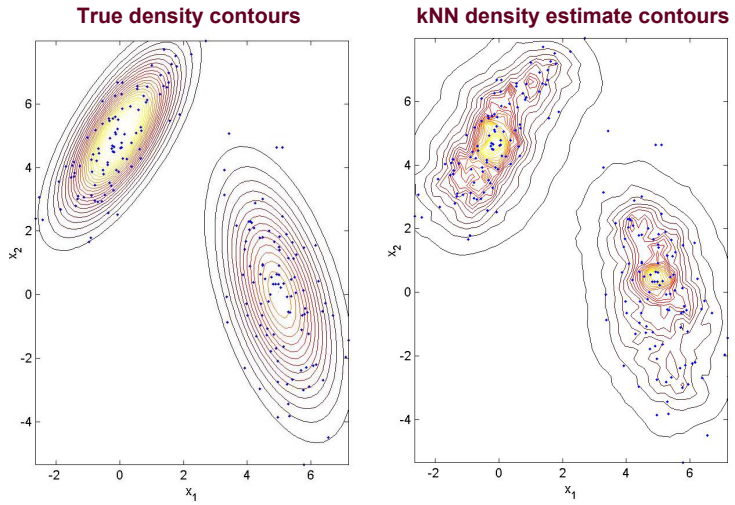


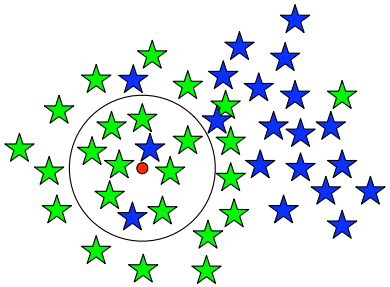
- The bottom figure shows the density estimate for $k = 10$ neighbors and $N = 200$ examples

- 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)





- We can then approximate the likelihood functions using the kNN method by:

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

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[1]

- 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 $\mathbf{x}_u$.
  - We draw a hyper-sphere of volume V around $\mathbf{x}_u$. Assume this volume contains a total of k examples, k_i from class ω_i

[1] From Bishop 1996

- Similarly, the unconditional density is estimated by

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

- And the priors are approximated by

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

- Putting everything together, the Bayes classifier becomes

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

Kernel Density Estimation in 1D

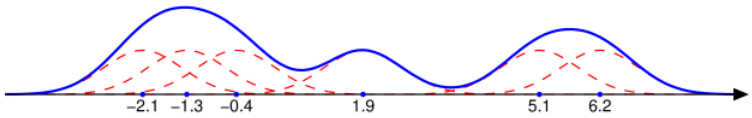
This is another popular non-parametric method for estimating $p(x)$ from a set of training examples $x_1, x_2, \ldots, x_N$. It is also known as **Parzen Windows**.

$$\hat{p}_h(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right)$$

where $K(\cdot)$ is some kernel (window) function and h is the bandwidth (smoothing parameter). Frequently, $K(\cdot)$ is chosen to the Gaussian function with mean 0 and variance 1.

$$K(\mathbf{x}) = \frac{1}{(2\pi)^{1/2}} \exp\left(-\frac{x^2}{2}\right)$$

An example



Six Gaussians (red) and their sum (blue). The kernel density estimate $\hat{p}_h(x)$ is obtained by dividing this sum by 6, the number of training examples. The bandwidth, h, of the estimate was set to 0.5.

Where the training points are denser the density estimate has higher values.

Each training sample contributes to $\hat{p}_h(x)$ in accordance with its distance from x.

Kernel Density Estimate Properties

- If $K(x) \geq 0 \ \forall x$ and $\int_x K(x)dx = h$ then

$$\hat{p}_h(x) \geq 0 \ \forall x \quad \text{and} \quad \int_x \hat{p}_h(x)dx = 1$$

- **Major Issue**: Choice of h. If **too large** then the density estimate $\hat{p}_h(x)$ will be very smooth and *out-of-focus*. If **too small** then $\hat{p}_h(x)$ will be noisy and wiggly.

  Rule of thumb 1:

  $$\hat{h} = 1.06\,\hat{\sigma}\,N^{-\frac{1}{5}}$$

  where $\hat{\sigma}^2$ is the sample variance of the points $x_1, \ldots, x_N$.

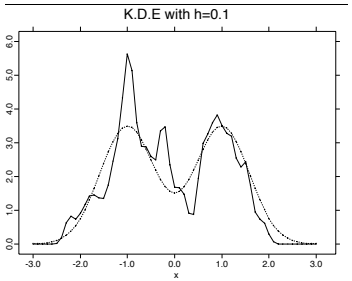
Rule of thumb 2:

$$\hat{h} = 1.06 \ \min\left(\hat{\sigma}, \frac{\hat{R}}{1.34}\right) N^{-\frac{1}{5}}$$

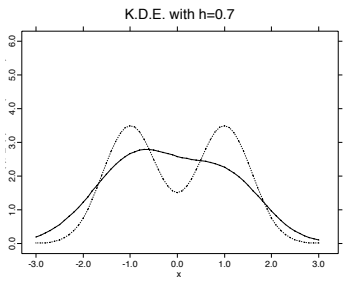
where $\hat{R}$ is an estimate of the interquartile range of the points $x_1, \ldots, x_N$. (This estimate is more robust to outliers.)

Effect of varying bandwidth

Kernel density estimates (solid line) with the Gaussian kernel and bandwidths of $h = .1$ and $h = .7$ for a sample of 100 observations from the density $.5N(-1, (\frac{4}{7})^2) + .5N(+1, (\frac{4}{7})^2)$ (dotted line).



too little smoothing



too much smoothing

Which estimate is better ?

and

$$K(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{\mathbf{x}^T\mathbf{x}}{2}\right)$$

Estimate the bandwidth in each dimension, $h_1, h_2, \ldots, h_d$ independently using the **rule of thumb** for one dimension.

Kernel Density Estimation in dD

This is another popular non-parametric method for estimating $p(\mathbf{x})$ from a set of training examples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$.

$$\hat{p}_{\mathbf{h}}(\mathbf{x}) = \frac{1}{Nh_1h_2\ldots h_d} \sum_{i=1}^{N} K\left(H^{-1}(\mathbf{x} - \mathbf{x}_i)\right)$$

where

$$H = \begin{pmatrix} h_1 & 0 & 0 & \ldots & 0 \\ 0 & h_2 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & h_d \end{pmatrix}$$

Nearest Neighbors

and

Non-Bayesian Classification

Nearest Neighbors

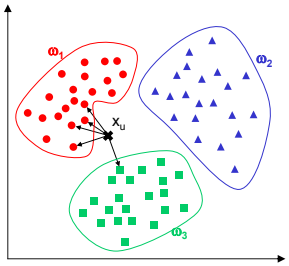
- Nearest Neighbors density estimation
- The k Nearest Neighbors classification rule

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 $\mathbf{x}_u \in R^d$, find the k *closest* labeled examples in the training data set and assign $\mathbf{x}_u$ 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*

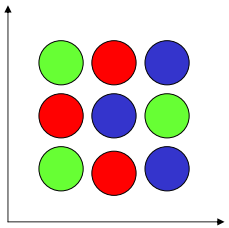
Example:

- In the example below we have three classes and the goal is to find a class label for the unknown example $\mathbf{x}_u$
- 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_2, so $\mathbf{x}_u$ is assigned to ω_1, the predominant class



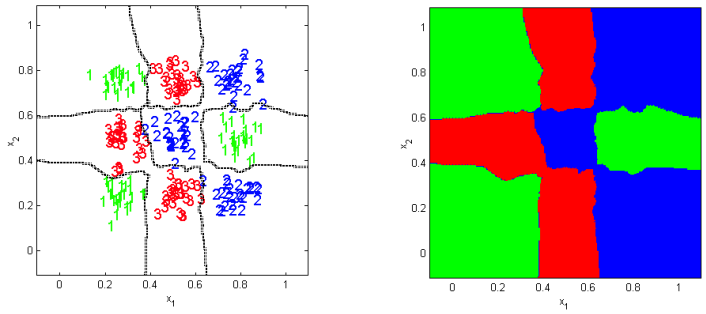
kNN in action: example 1

Have generated data for a 2-dimensional 3-class problem, where the class-conditional densities are multi-modal, and non-linearly separable as shown.



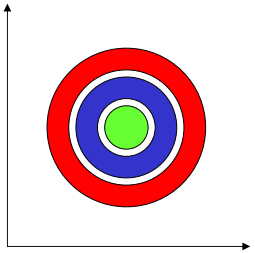
Solution:
Use the kNN rule with $k = 5$ and the Euclidean distance as the distance metric

The resulting decision boundaries and decision regions are shown below



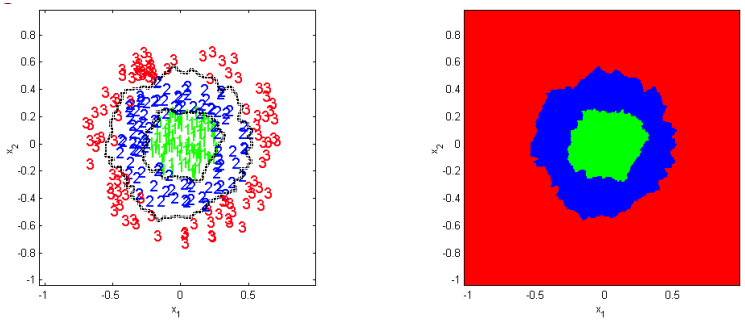
kNN in action: example 2

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 below



Solution:
Use the kNN rule with $k = 5$ and the Euclidean distance as a metric

The resulting decision boundaries and decision regions are shown below



Distance Functions

- The nearest neighbor classifier relies on a metric or a distance function between points.
- For all points $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$, a metric $D(\cdot, \cdot)$ must satisfy the following properties:
  - Nonnegativity: $D(\mathbf{x}, \mathbf{y}) \geq 0$.
  - Reflexivity: $D(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}$.
  - Symmetry: $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$.
  - Triangle inequality: $D(\mathbf{x}, \mathbf{y}) + D(\mathbf{y}, \mathbf{z}) \geq D(\mathbf{x}, \mathbf{z})$.

Distance Functions

- A general class of metrics for d-dimensional patterns is the Minkowski metric

$$L_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

  also referred to as the L_p norm.

- The Euclidean distance is the L_2 norm

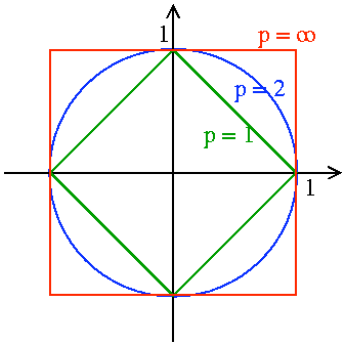
$$L_2(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^2 \right)^{\frac{1}{2}}$$

- The Manhattan or city block distance is the L_1 norm

$$L_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} |x_i - y_i|$$

- The L_∞ norm is the maximum of the distances along individual coordinate axes

$$L_\infty(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|$$



Each colored shape consists of points at a distance 1.0 from the origin, measured using different values of p in the Minkowski L_p metric.

Nearest Neighbors

- Nearest Neighbors density estimation
- The k Nearest Neighbors classification rule
- kNN as a lazy learner

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

Nearest Neighbors

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

Characteristics of the kNN classifier

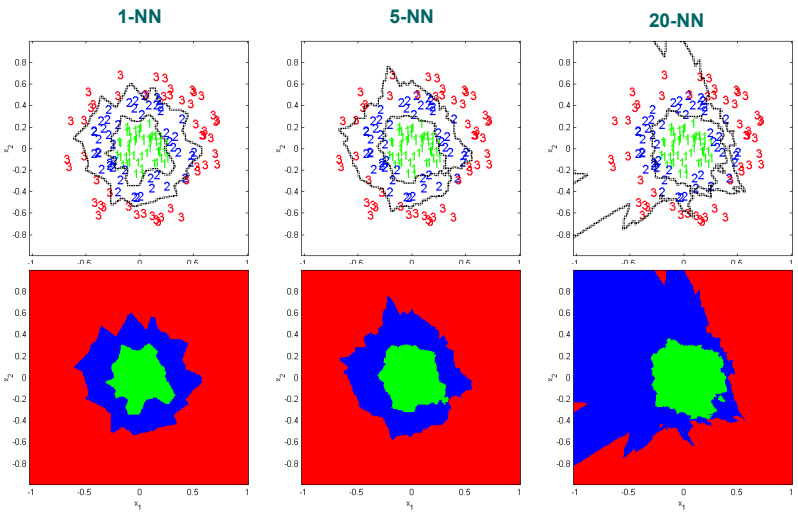
- **Advantages**
  - Analytically tractable
  - Simple implementation
  - Nearly optimal in the large sample limit, as $N \rightarrow \infty$

$$P(\text{error})_{\text{Bayes}} < P(\text{error})_{1\text{NN}} < 2P(\text{error})_{\text{Bayes}}$$

  - 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 Vs 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



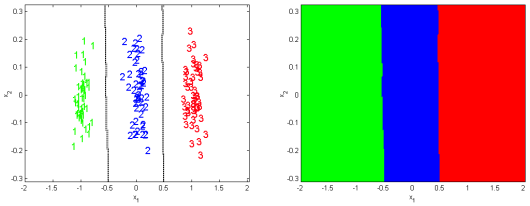
kNN and the problem of feature weighting

The basic kNN rule's similarity measure is based on the **Euclidean distance** which makes the kNN rule very sensitive to noisy features.

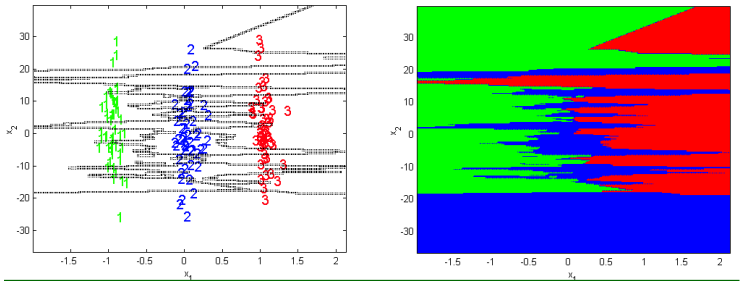
As an example, have a data set with 3-classes and 2 dimensions:

- The first axis contains the discriminatory information. Class separability is excellent.
- The second axis is white noise and, thus, does not contain classification information.

Example 1 Both axes are scaled properly. Then kNN (k=5) finds decision boundaries fairly close to the optimal.



Example 2 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 $\mathcal{N}(0,1)$
  - 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.

$$D(\mathbf{x}_u, \mathbf{x}) = \sqrt{\sum_{k=1}^{d} (x_{uk} - x_k)^2}$$

- 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

$$D(\mathbf{x}_u, \mathbf{x}) = \sqrt{\sum_{k=1}^{d} (w_k (x_{uk} - x_k))^2}$$

- 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 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 $P(\omega_i|\mathbf{x})$ 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 predetermined 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
- In Feature subset selection (FSS) the performance bias methods are called **wrappers** and preset bias methods are called **filters**

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

Improving the nearest neighbor search procedure

- The problem of nearest neighbor can be stated as follows:

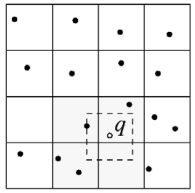
  Given a set of N points in d-dimensional space and an unlabeled example $\mathbf{x}_u \in R^d$, find the point that minimizes the distance to $\mathbf{x}_u$.

  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

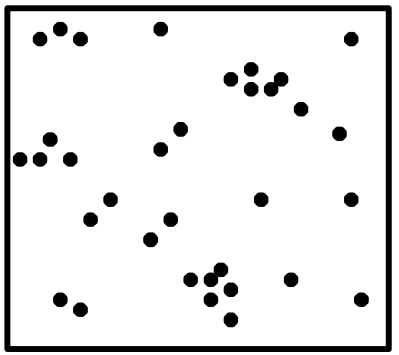
- In the Bucketing algorithm
  - The space is divided into identical cells and for each cell the data points inside it are stored in a list
  - The cells are examined in order of increasing distance from the query point and for each cell the distance is computed between its internal data points and the 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

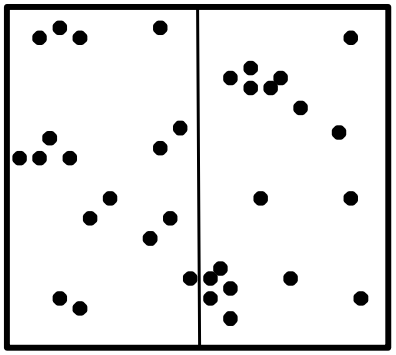
KD-tree construction

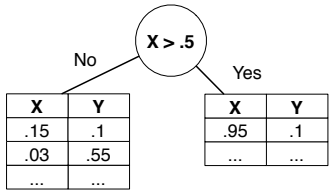


X	Y
.15	.1
.03	.55
.95	.1
...	...

Start with a list of n-dimensional points

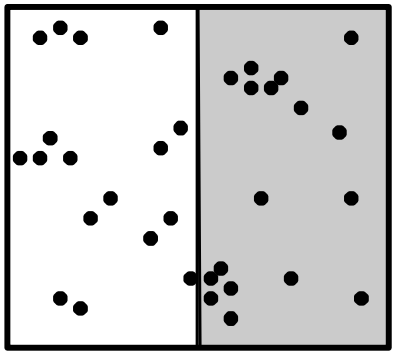
KD-tree construction

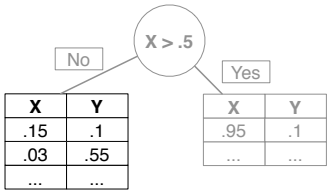




Split the points into 2 groups by choosing a dimension X and values v and separating the points into $x > v$ and $x \geq v$.

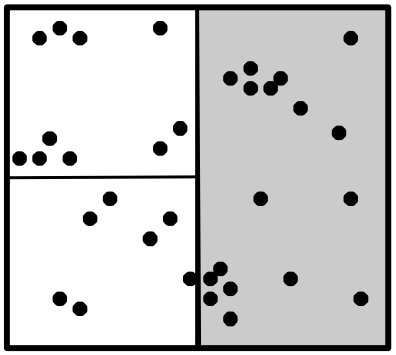
KD-tree construction

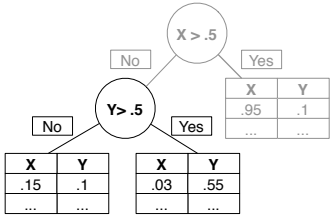




Consider each group separately and possibly split again (along same/different dimension).

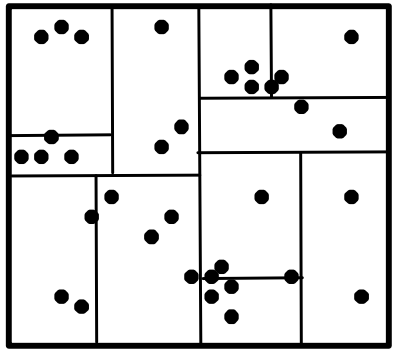
KD-tree construction

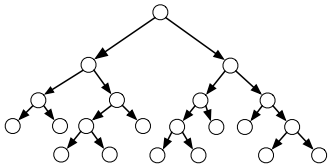




Consider each group separately and possibly split again (along same/different dimension).

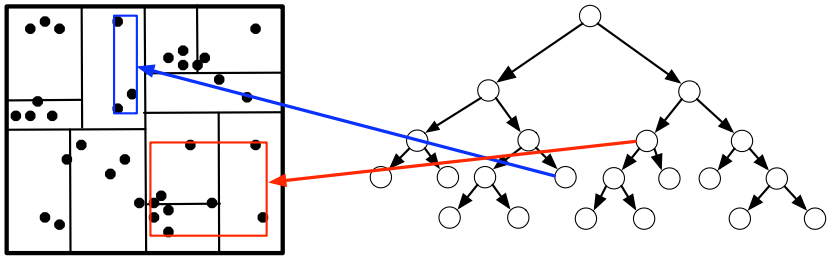
KD-tree construction





Keep splitting the points in each set to create a tree structure. Each node with no children (leaf node) contains a list of points.

KD-tree construction



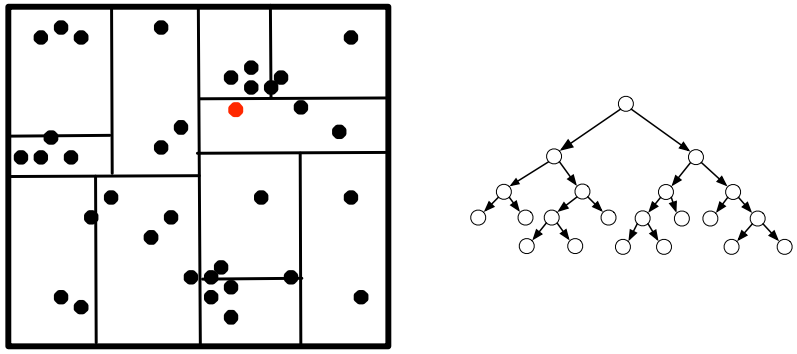
Will keep around one additional piece of information at each node. The (tight) bounds of the points at or below this node.

KD-tree construction

Use heuristics to make splitting decisions:

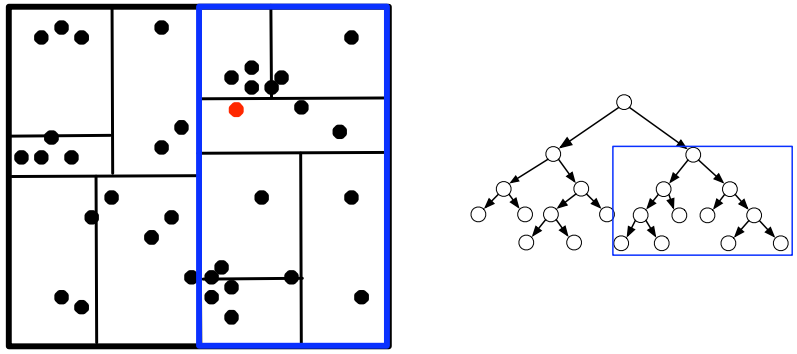
- Which dimension do we split along ?
  Widest
- Which value do we split at ?
  Median of value of the split dimension for the points.
- When do we stop ?
  When there are fewer then m points left **OR** the box has hit some minimum width.

Nearest neighbour with KD-trees



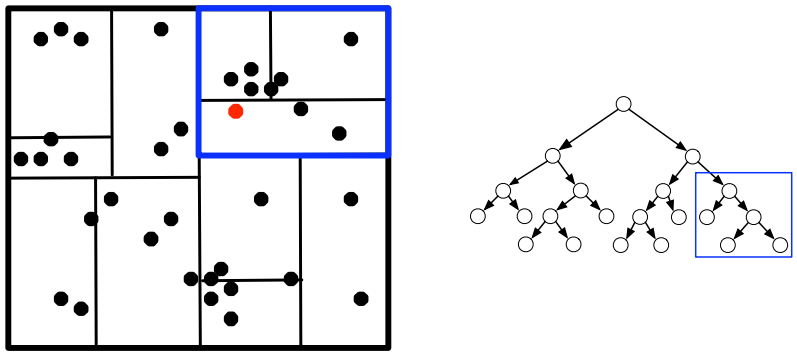
Traverse the tree looking for the nearest neighbor of the query point.

Nearest neighbour with KD-trees



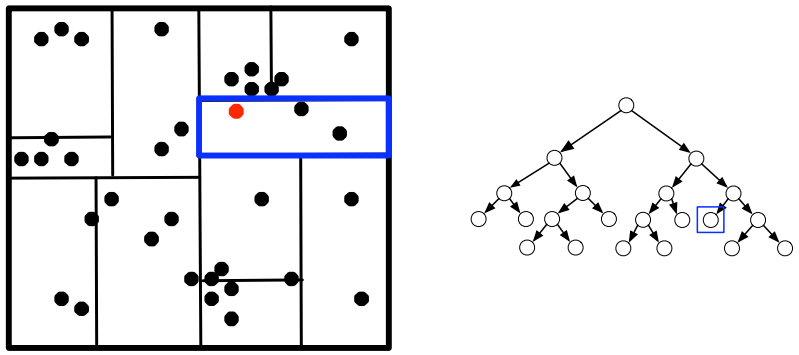
Examine nearby points first: Explore the branch of the tree that is closest to the query point first.

Nearest neighbour with KD-trees



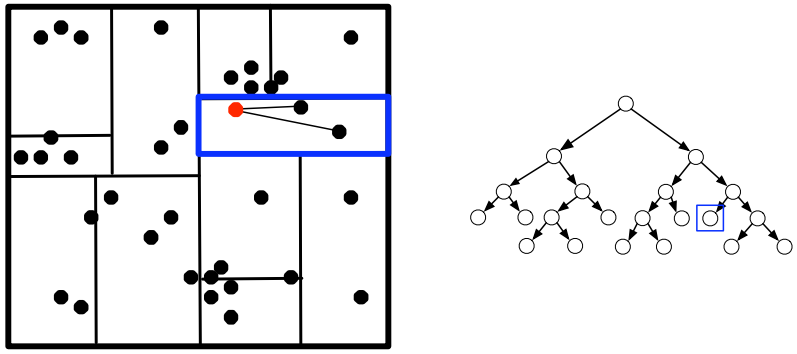
Examine nearby points first: Explore the branch of the tree that is closest to the query point first.

Nearest neighbour with KD-trees



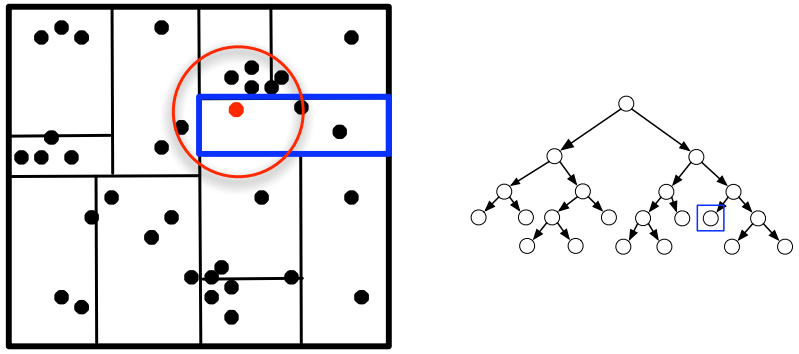
When a leaf node is reached: compute the distance to each point in the node.

Nearest neighbour with KD-trees



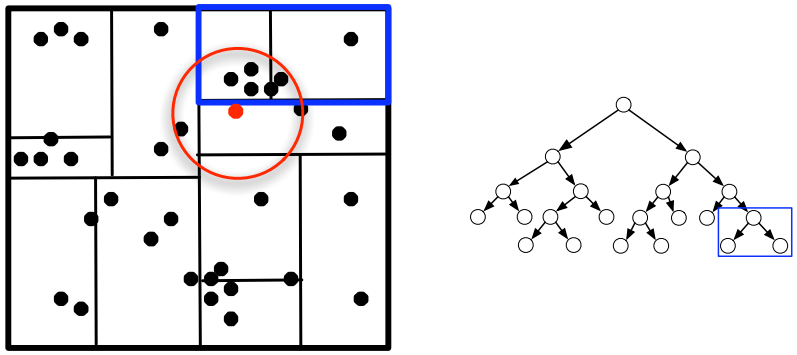
When a leaf node is reached: compute the distance to each point in the node.

Nearest neighbour with KD-trees



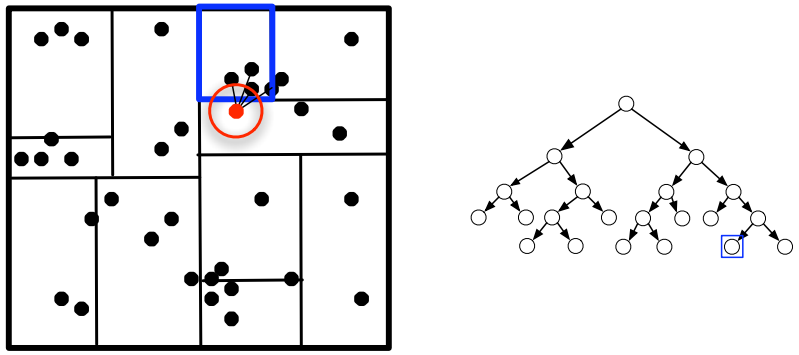
When a leaf node is reached: compute the distance to each point in the node.

Nearest neighbour with KD-trees



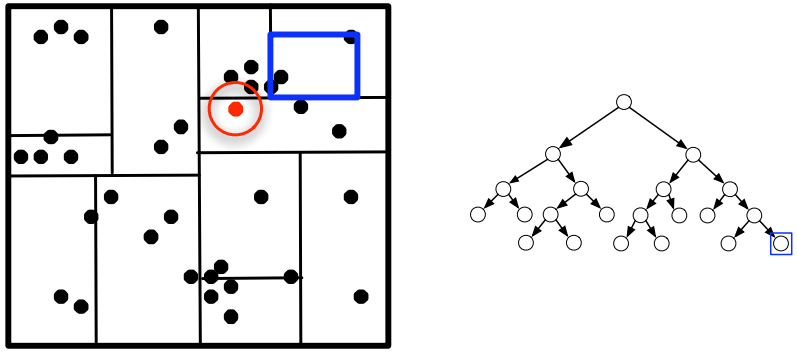
Then we can backtrack and try the other branch at each node visited.

Nearest neighbour with KD-trees



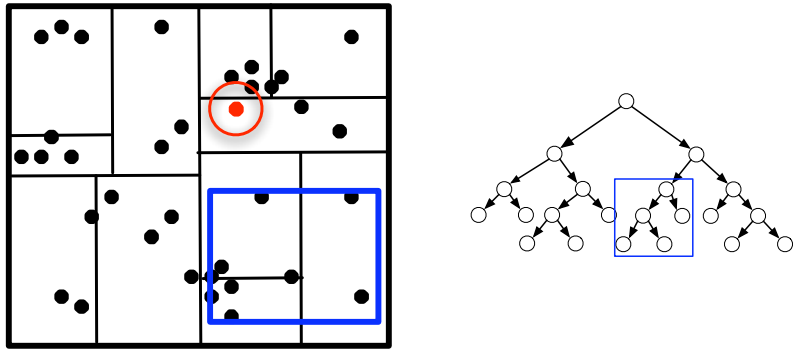
Each time a new closest node is found, we can update the distance bounds.

Nearest neighbour with KD-trees



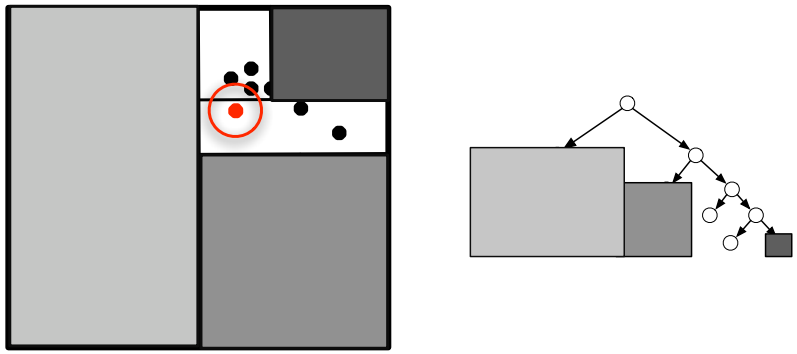
Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could **NOT** include the nearest neighbour.

Nearest neighbour with KD-trees



Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could **NOT** include the nearest neighbour.

Nearest neighbour with KD-trees



Using the distance bounds and the bounds of the data below each node, we can prune parts of the tree that could **NOT** include the nearest neighbour.