Lecture 5

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

Nearest Neighbours

and

Non-parametric Density Estimation

Remember..

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** $p(\mathbf{x} \mid \omega_i)$.

What is this optimal classifier?

Remember..

It is a decision rule based on the **Likelihood Ratio Test**:

$$\text{Class}\,(\mathbf{x}) = \begin{cases} \omega_1 & \text{if } \frac{p(\mathbf{x}\,|\,\omega_1)}{p(\mathbf{x}\,|\,\omega_2)} \geq \frac{P(\omega_2)}{P(\omega_1)} \\ \omega_2 & \text{if } \frac{p(\mathbf{x}\,|\,\omega_1)}{p(\mathbf{x}\,|\,\omega_2)} < \frac{P(\omega_2)}{P(\omega_1)} \end{cases}$$

This classifier minimizes the probability of error: $P(\text{error})$.

Potential stumbling block

Unfortunately, we rarely have complete knowledge of these class-conditional densities or the prior probabilities.

$$p(\mathbf{x} \,|\, \omega_i) = ?? \qquad P(\omega_i) = ??$$

However, we can often find training data that include particular representatives of the patterns we want to classify. Can obtain

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \text{ sampled from } p(\mathbf{x} \,|\, \omega_i)$$

Density estimation

Given

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \text{ sampled from } p(\mathbf{x} \mid \omega_i)$$

Want to estimate $p(\mathbf{x} \mid \omega_i)$.

HOW ?

Density estimation

Approach I:

Parametric Assume some parametric form for the conditional densities.

For example assume each one is a multivariate Gaussian:

$$p(\mathbf{x} \mid \omega_i) = \mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$$

Estimate its parameters, $(\hat{\boldsymbol{\mu}}_i \; \hat{\Sigma}_i)$, from the training examples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$.

Then use the resulting estimates as if they were the true values and perform classification using the Bayesian decision rule.
That is set: $\boldsymbol{\mu}_i = \hat{\boldsymbol{\mu}}_i$ and $\Sigma_i = \hat{\Sigma}_i$

Technical Interlude: MLE

Maximum Likelihood Estimation is a fundamental part of data analysis. It is used frequently used for parameter estimation.

Suppose you have $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ (i.i.d) with each $\mathbf{x}_i \sim p(\mathbf{x} \,|\, \boldsymbol{\theta})$. Then the MLE of $\boldsymbol{\theta}$ is defined as

$$
\begin{aligned}
\boldsymbol{\theta}^{\text{MLE}} &= \arg\max_{\boldsymbol{\theta}} \; p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \,|\, \theta) \\
&= \arg\max_{\boldsymbol{\theta}} \; \prod_{i=1}^{n} p(\mathbf{x}_i \,|\, \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \; \sum_{i=1}^{n} \log p(\mathbf{x}_i \,|\, \boldsymbol{\theta})
\end{aligned}
$$

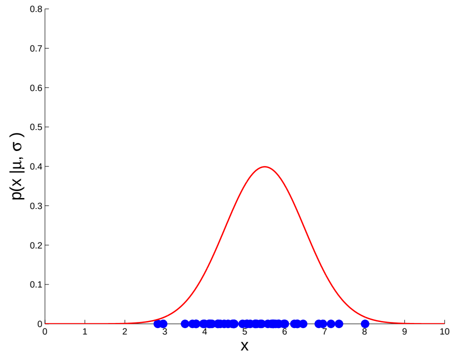
Will quickly review the general approach when x is univariate.

Learning univariate Gaussians from data

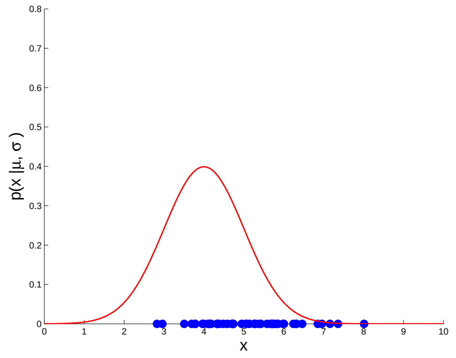
Suppose you have $x_1, x_2, \ldots, x_n$ (i.i.d) with each $x_i \sim \mathcal{N}(\mu, \sigma^2)$.

If you know σ but not μ then: $\mu^{\text{MLE}} = \arg\max_{\mu} \, p(x_1, x_2, \ldots, x_n \mid \mu, \sigma^2)$

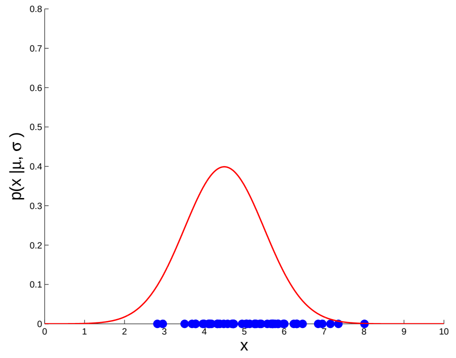
Log-likelihood scores for 3 different μ values



$\mu_{\text{try}} = 5.5$
$\mathcal{L} = -80.4257$



$\mu_{\text{try}} = 4$
$\mathcal{L} = -109.7614$



$\mu_{\text{try}} = 4.5$
$\mathcal{L} = -87.4829$

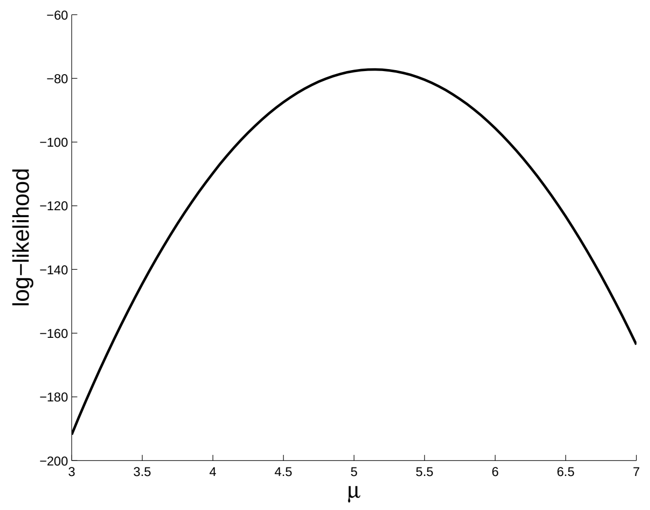
Learning univariate Gaussians from data

Suppose you have $x_1, x_2, \ldots, x_n$ (i.i.d) with each $x_i \sim \mathcal{N}(\mu, \sigma^2)$.

Say you know σ but not μ, then

$$\mu^{\mathsf{MLE}} = \arg\max_{\mu} p(x_1, x_2, \ldots, x_n \,|\, \mu, \sigma^2)$$

Graph of log-likelihood scores for different μ's



General MLE strategy for a scalar

Task Find MLE θ assuming known form of $p(\text{Data} \,|\, \theta, \text{stuff})$

1. Write down the log-likelihood of the data

$$\mathcal{L} = \log p(\text{Data} \,|\, \theta, \text{stuff}) \quad \left(= \sum_{i=1}^{n} \log p(x_i \,|\, \theta, \text{stuff})\right)$$

2. Work out $\frac{\partial \mathcal{L}}{\partial \theta}$

3. Set $\frac{\partial \mathcal{L}}{\partial \theta} = 0$ to find the maximum, creating an equation in θ and solve for θ.

4. Check you've found a maximum rather than a minimum or saddle-point and be careful if θ is constrained.

Learning univariate Gaussians from data

$$
\begin{aligned}
\mu^{\mathsf{MLE}} &= \arg\max_{\mu} \; p(x_1, x_2, \ldots, x_n \mid \mu, \sigma^2) \\
&= \arg\max_{\mu} \; \prod_{i=1}^{n} p(x_i \mid \mu, \sigma^2) \\
&= \arg\max_{\mu} \; \sum_{i=1}^{n} \log p(x_i \mid \mu, \sigma^2) \\
&= \arg\max_{\mu} \; - n \log \sigma - \frac{n}{2} \log 2\pi + \sum_{i=1}^{n} -\frac{(x_i - \mu)^2}{2\sigma^2} \\
&= \arg\min_{\mu} \; \sum_{i=1}^{n} (x_i - \mu)^2
\end{aligned}
$$

The MLE μ

$$0 = \frac{\partial \mathcal{L}}{\partial \mu} = \frac{\partial}{\partial \mu} \sum_{i=1}^{n} (x_i - \mu)^2 = -\sum_{i=1}^{n} 2\,(x_i - \mu)$$

Thus

$$\mu^{\text{MLE}} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

The best estimate of the mean of a distribution is the mean of the sample!

General MLE strategy

Suppose $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_p)^T$ is a vector of parameters

Task Find MLE $\boldsymbol{\theta}$ assuming known form of $p(\text{Data} \,|\, \boldsymbol{\theta}, \text{stuff})$

1. Write down the log-likelihood of the data

$$\mathcal{L} = \log p(\text{Data} \,|\, \boldsymbol{\theta}, \text{stuff}) (= \sum_{i=1}^{n} \log p(x_i \,|\, \boldsymbol{\theta}, \text{stuff}))$$

2. Calculate $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left(\frac{\partial \mathcal{L}}{\partial \theta_1}, \quad \frac{\partial \mathcal{L}}{\partial \theta_2}, \quad \cdots, \quad \frac{\partial \mathcal{L}}{\partial \theta_p} \right)^T$
3. Solve the set of simultaneous equations

$$\frac{\partial \mathcal{L}}{\partial \theta_1} = 0, \ \frac{\partial \mathcal{L}}{\partial \theta_2} = 0, \cdots, \frac{\partial \mathcal{L}}{\partial \theta_p} = 0$$

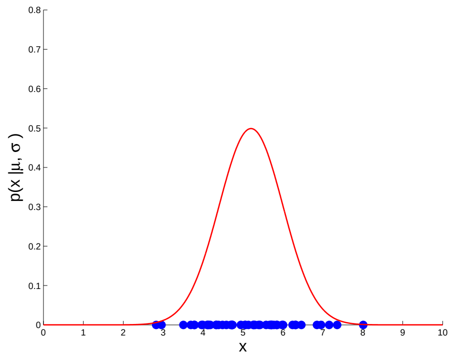
4. Check you've found a maximum.

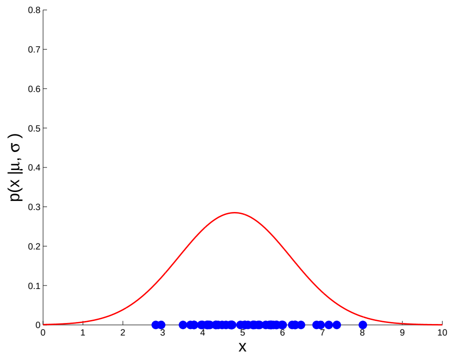
Learning univariate Gaussians from data

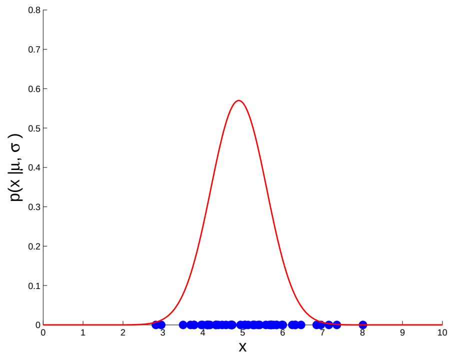
Suppose you have $x_1, x_2, \ldots, x_n$ (i.i.d) with each $x_i \sim \mathcal{N}(\mu, \sigma^2)$.

You don't know μ or σ:

Log-likelihood scores for 3 different (μ, σ) values







$\mu_{\text{try}} = 5.2, \sigma_{\text{try}} = .8$
$\mathcal{L} = -83.7677$

$\mu_{\text{try}} = 4.8, \sigma_{\text{try}} = 1.4$
$\mathcal{L} = -80.2036$

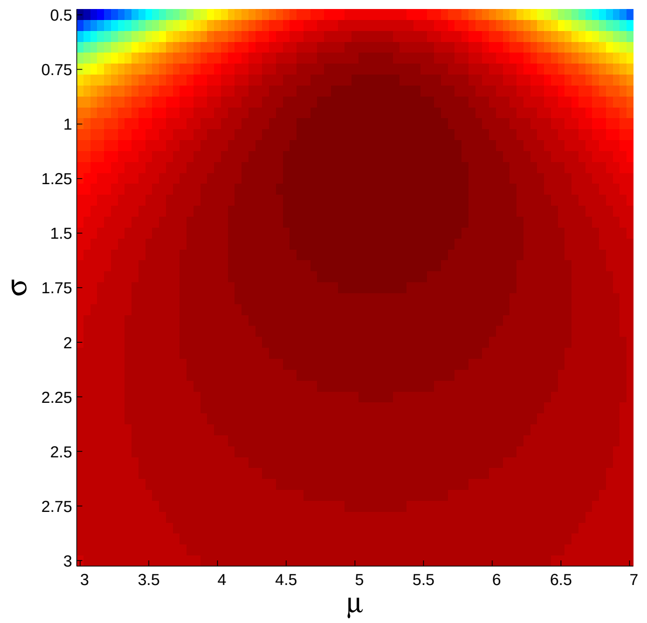
$\mu_{\text{try}} = 4.9, \sigma_{\text{try}} = .7$
$\mathcal{L} = -94.8745$

Learning univariate Gaussians from data

Suppose you have $x_1, x_2, \ldots, x_n$ (i.i.d) with each $x_i \sim \mathcal{N}(\mu, \sigma^2)$.

You don't know μ or σ. Find it by maximizing the log-likelihood.

Log-likelihood scores as μ and σ vary



Learning univariate Gaussians from data

Suppose you have $x_1, x_2, \ldots, x_n$ (i.i.d) with each $x_i \sim \mathcal{N}(\mu, \sigma^2)$.

You don't know μ or σ. Find it by maximizing the log-likelihood.

In this case can solve the optimization problem analytically:

The log-likelihood

$$\begin{aligned}\mathcal{L} = \log p(x_1, x_2, \ldots, x_n \mid \mu, \sigma^2) &= \sum_{i=1}^{n} \log p(x_i \mid \mu, \sigma^2) \\ &= -n \log \sigma - \frac{n}{2} \log 2\pi - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2\end{aligned}$$

Optimization of the log-likelihood

$$\frac{\partial \mathcal{L}}{d\mu} = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu) = 0 \qquad \Longrightarrow \qquad \mu^{\mathsf{MLE}} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\frac{\partial \mathcal{L}}{d\sigma} = -\frac{n}{\sigma} + \frac{2}{2\sigma^3}\sum_{i=1}^{n}(x_i - \mu)^2 = 0 \qquad \Longrightarrow \sigma^2_{\mathsf{MLE}} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu^{\mathsf{MLE}})^2$$

Density estimation

Approach II:

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.

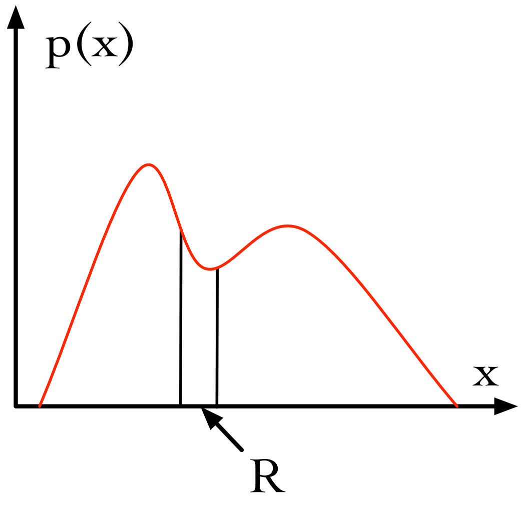
Non-parametric density estimation: How?

Given: Suppose n samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are drawn i.i.d. (independently and identically distributed) from a probability density function $p(\mathbf{x})$.

Probability of a sample landing in a region

The probability, $P_{\mathcal{R}}$, a vector $\mathbf{x} \sim p$ will fall in a region $\mathcal{R}$ is given by

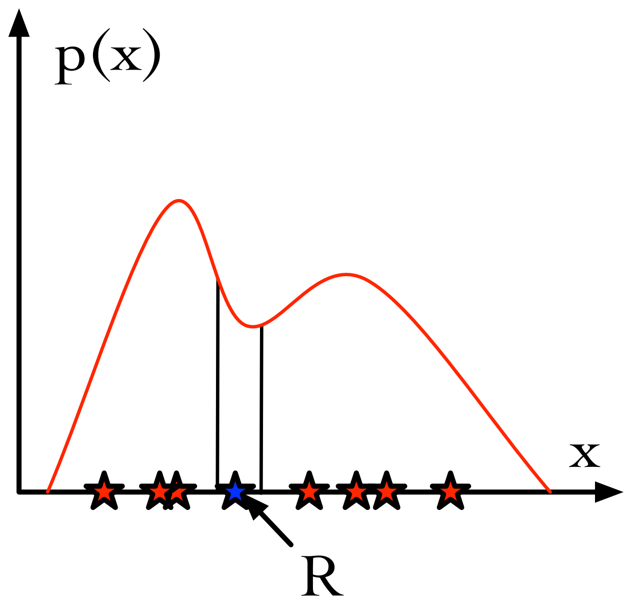
$$P_{\mathcal{R}} = \int_{\mathbf{x}' \in \mathcal{R}} p(\mathbf{x}')\, d\mathbf{x}'$$



Probability of k samples landing in a region

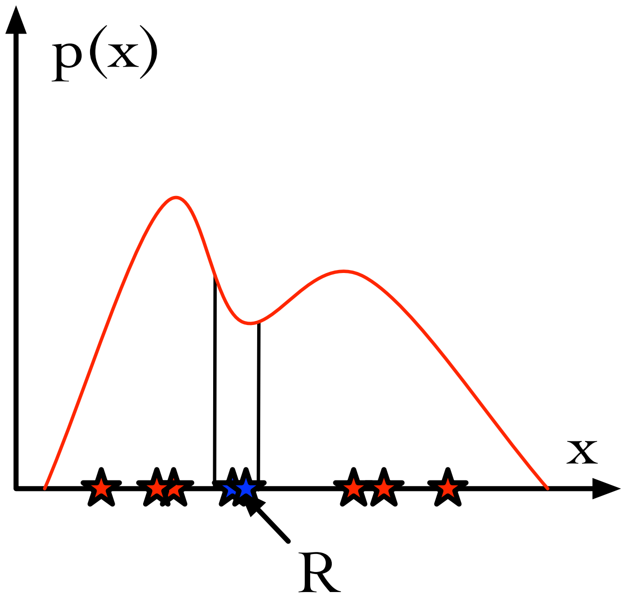
The probability 1 of the n training points will fall in $\mathcal{R}$ is

$$P_{\mathcal{R}}^{(1)} = \sum_{i=1}^{n} \underbrace{P_{\mathcal{R}}\,(1 - P_{\mathcal{R}})^{n-1}}_{\Downarrow} = n\,P_{\mathcal{R}}\,(1 - P_{\mathcal{R}})^{n-1}$$



The probability 2 of the n training points will fall in $\mathcal{R}$ is

$$P_{\mathcal{R}}^{(2)} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \underbrace{P_{\mathcal{R}}^{2} \left(1 - P_{\mathcal{R}}\right)^{n-2}}_{\Downarrow} = \frac{1}{2} n(n-1)\, P_{\mathcal{R}}^{2} \left(1 - P_{\mathcal{R}}\right)^{n-2}$$



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

$$P_{\mathcal{R}}^{(k)} = \binom{n}{k} P_{\mathcal{R}}^{k} \left(1 - P_{\mathcal{R}}\right)^{n-k}$$

Expected number of samples landing in region $\mathcal{R}$

The expected value of k is

$$\begin{aligned}
\mathrm{E}\left[k\right] &= \sum_{k=0}^{n} k\, P_{\mathcal{R}}^{(k)} = \sum_{k=0}^{n} k \binom{n}{k} P_{\mathcal{R}}^{k} \left(1 - P_{\mathcal{R}}\right)^{n-k} = \sum_{k=1}^{n} k \binom{n}{k} P_{\mathcal{R}}^{k} \left(1 - P_{\mathcal{R}}\right)^{n-k} \\
&= n\, P_{\mathcal{R}} \sum_{k'=0}^{n-1} \binom{n-1}{k'} P_{\mathcal{R}}^{k'} \left(1 - P_{\mathcal{R}}\right)^{n-1-k'} \\
&= n\, P_{\mathcal{R}} \left(P_{\mathcal{R}} + \left(1 - P_{\mathcal{R}}\right)\right)^{n-1}, \quad \text{as } (x+y)^n = \textstyle\sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k} \\
&= n\, P_{\mathcal{R}}
\end{aligned}$$

If k samples land in $\mathcal{R}$ what is a good estimate for $P_{\mathcal{R}}$?

$$P(k \text{ samples land in } \mathcal{R}) = \binom{n}{k} P_{\mathcal{R}}^{k} \left(1 - P_{\mathcal{R}}\right)^{n-k}$$

The MLE for $P_{\mathcal{R}}$ is calculated via the constraint

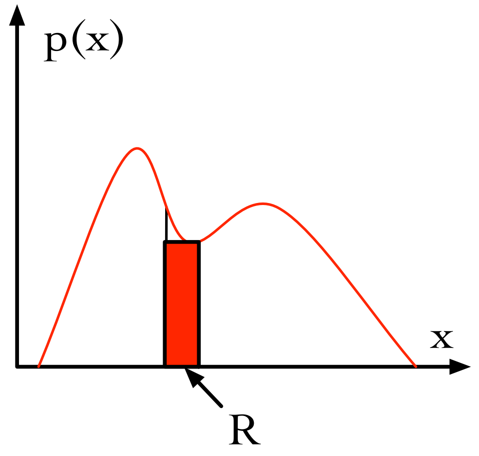
$$\begin{aligned}
\frac{d\, P(k \text{ samples land in } \mathcal{R})}{dP_{\mathcal{R}}} &= \binom{n}{k} k\, P_{\mathcal{R}}^{k-1} \left(1 - P_{\mathcal{R}}\right)^{n-k} + \binom{n}{k} P_{\mathcal{R}}^{k} \left(n - k\right) \left(1 - P_{\mathcal{R}}\right)^{n-k-1} \\
&= \binom{n}{k} P_{\mathcal{R}}^{k-1} \left(1 - P_{\mathcal{R}}\right)^{n-k-1} \left(k \left(1 - P_{\mathcal{R}}\right) + \left(n - k\right) P_{\mathcal{R}}\right) \\
&= \binom{n}{k} P_{\mathcal{R}}^{k-1} \left(1 - P_{\mathcal{R}}\right)^{n-k-1} \left(k - n\, P_{\mathcal{R}}\right) = 0
\end{aligned}$$

Solving this constraint we get

$$\hat{P}_{\mathcal{R}} = \frac{k}{n}$$

Non-parametric density estimation: How?

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



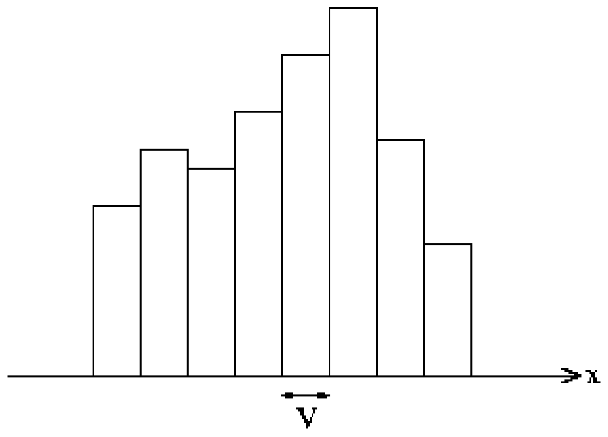
$$\begin{aligned} P_{\mathcal{R}} &= \int_{\mathbf{x}'\in\mathcal{R}} p(\mathbf{x}')\, d\mathbf{x}' \\ &\approx p(\mathbf{x}) \int_{\mathbf{x}'\in\mathcal{R}} d\mathbf{x}' \quad \text{for some } \mathbf{x} \in \mathcal{R} \\ &= p(\mathbf{x})\, V \end{aligned}$$

If we approximate $P_{\mathcal{R}}$ with $\frac{k}{n}$ 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

Have computed the general expression for non-parametric density estimation:

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

Approach I to computing this estimate:

Fix the volume V and count the number k of data points inside V.

This is the histogram method.

Non-parametric density estimation

Have computed the general expression for non-parametric density estimation:

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

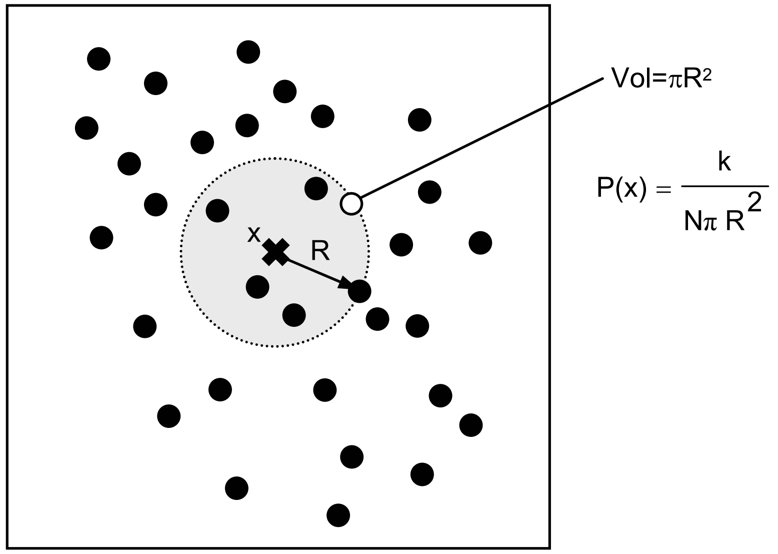
Approach II to computing this estimate:

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 Neighbour (kNN) approach

kNN density estimation

In the kNN method grow the volume surrounding the estimation point $\mathbf{x}$ until it encloses a total of k data points



The density estimate then becomes

$$p(\mathbf{x}) \cong \frac{k}{nV} = \frac{k}{n\, c_d\, R_k^d(\mathbf{x})}$$

where

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

$$c_d = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)}$$

  (Remember $\Gamma(n+1) = n\,\Gamma(n)$ and $\Gamma(1) = 1, \Gamma(\frac{1}{2}) = \sqrt{\pi}$)

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

kNN density estimation

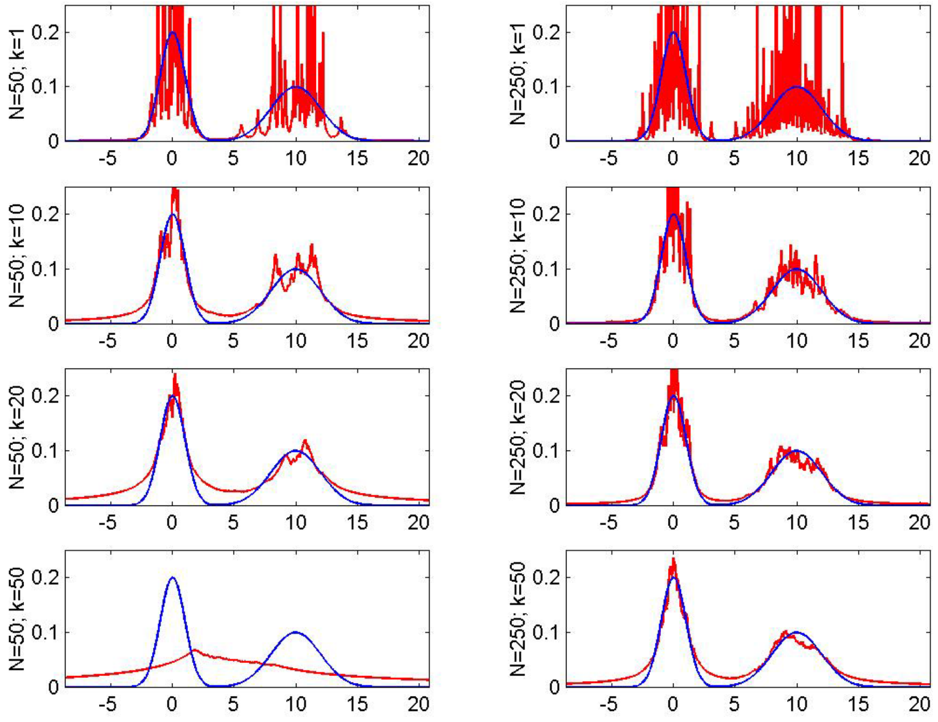
Unfortunately, the estimates obtained with the kNN method are frequently not very satisfactory.

Why do you think is the case?

These properties are illustrated in the next few slides.

kNN density estimation, example 1

A kNN estimate for a mixture of two Gaussians: $p(x) = \frac{1}{2}\mathcal{N}(0,1) + \frac{1}{2}\mathcal{N}(10,4)$ using several values of n and k.



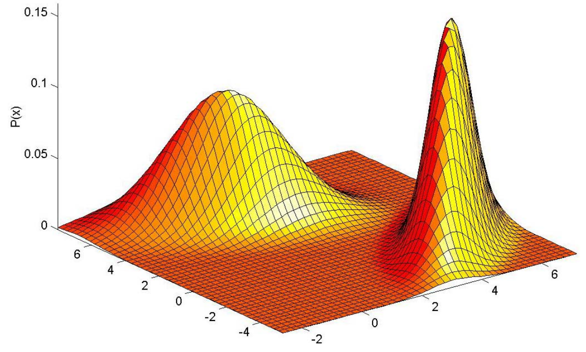
$n = 50$ $n = 250$

kNN density estimation, example 2

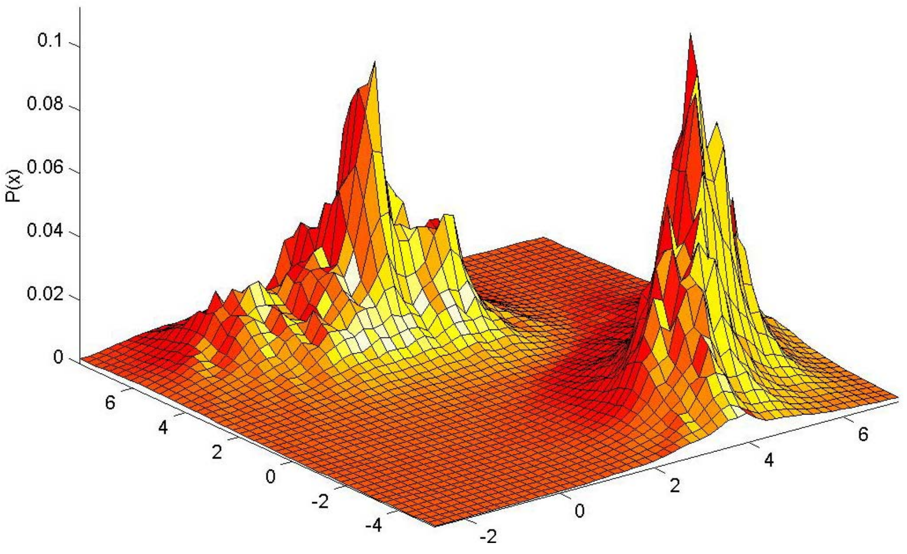
A two dimensional example

Below is the true density, a mixture of two bivariate Gaussians

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

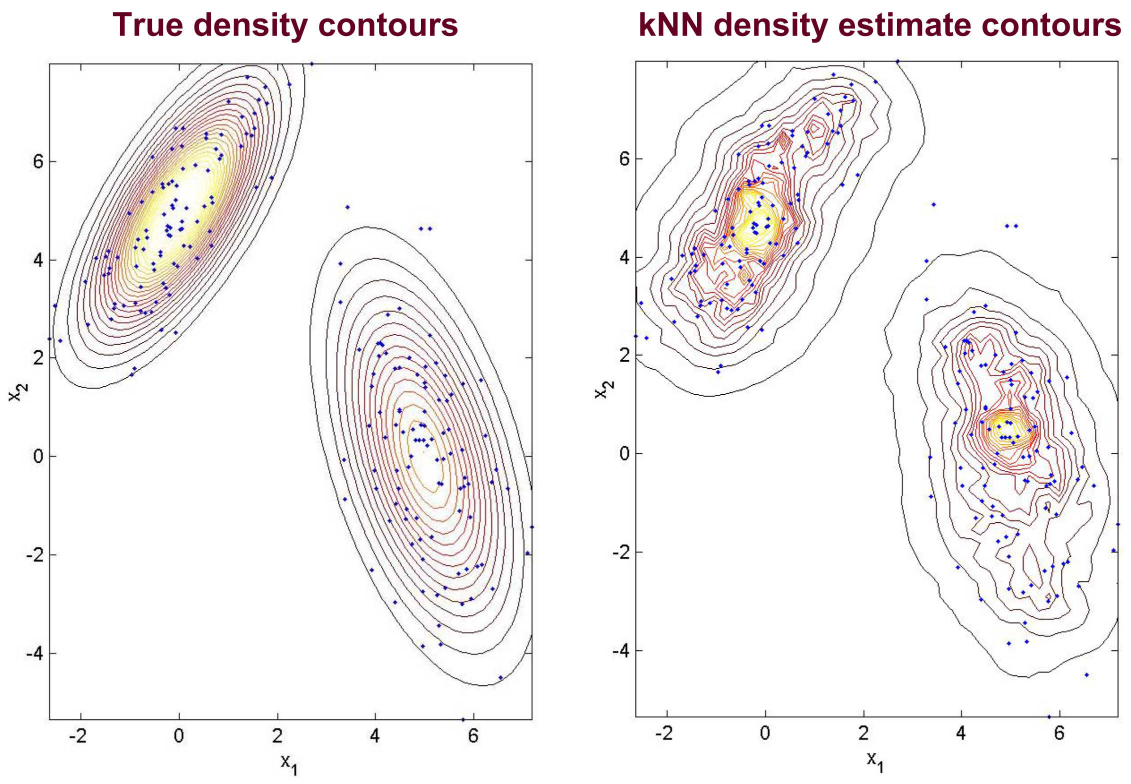


$$\boldsymbol{\mu}_1 = (0, 5)^T,\ \Sigma_1 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}, \quad \boldsymbol{\mu}_2 = (5, 0)^T,\ \Sigma_2 = \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix}$$



The density estimate for $k = 10$ neighbours and $n = 200$ examples

kNN density estimation, example 2



Contours of the true and estimated distribution and training data.

kNN density estimation

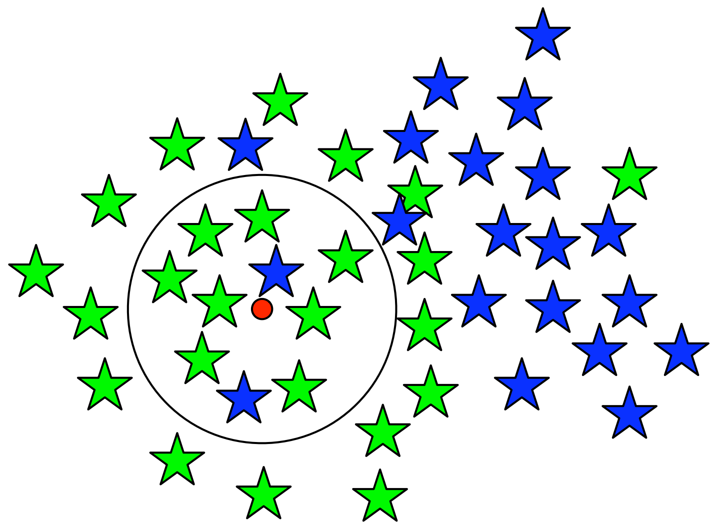
Common problems with the estimates obtained with the kNN method:

- 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.

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, from Bishop 1996.

Derivation: Assume we have a dataset with n examples, n_i from class ω_i, and we want to classify an unknown sample $\mathbf{x}_u$.



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

Can approximate the likelihood functions using the kNN method by:

$$p(\mathbf{x}_u \,|\, \omega_i) = \frac{k_i}{n_i V}$$

The priors are approximated by

$$P(\omega_i) = \frac{n_i}{n}$$

The unconditional density is estimated by

$$p(\mathbf{x}_u) = \sum_i p(\mathbf{x}_u \,|\, \omega_i)\, P(\omega_i) = \sum_i \frac{k_i}{n_i V}\frac{n_i}{n} = \frac{1}{n\, V}\sum_i k_i = \frac{k}{nV}$$

Putting everything together, the Bayes classifier becomes

$$P(\omega_i \,|\, \mathbf{x}_u) = \frac{p(\mathbf{x}_u \,|\, \omega_i)\, P(\omega_i)}{p(\mathbf{x}_u)} = \frac{\frac{k_i}{n_i V}\frac{n_i}{n}}{\frac{k}{n\, V}} = \frac{k_i}{k}$$

Kernel density estimation in 1D

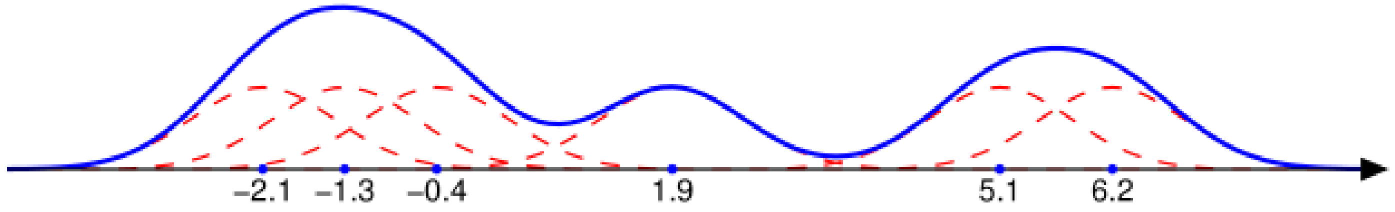
This is another popular non-parametric method for estimating $p(x)$ from a set of training examples $x_1, x_2, \ldots, x_n$. It is also known as **Parzen Windows**.

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

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.

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

An example



Six Gaussians (red) and their sum (blue). The bandwidth, h, of the estimate was set to 0.5. The kernel density estimate $\hat{p}_h(x)$ is obtained by dividing 6 $\times$ .5.

Note:

– 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

Properties:

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

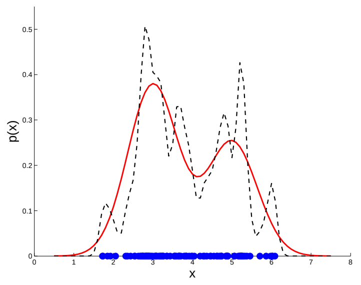
$$\hat{p}_h(x) \geq 0 \; \forall x \quad \text{and} \quad \int_x \hat{p}_h(x)\,dx = 1$$

Major parameter choice:

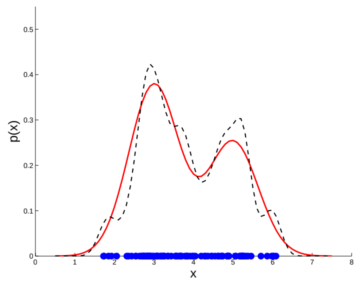
Choice of h (bandwidth). If **too large** then the density estimate $\hat{p}_h(x)$ will be very smooth and *out-of-focus*. If **too small** then $\hat{p}_h(x)$ will be noisy and wiggly.

Effect of varying bandwidth

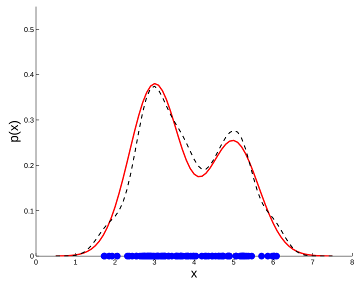
The true density (**red curve**) $.6\,\mathcal{N}(3,.4^2)+.4\,\mathcal{N}(5,.4^2)$ is approximated with kernel density estimation (**dashed curve**) with bandwidths varying from $h=.1$ to $h=.6$ using 100 training observations.



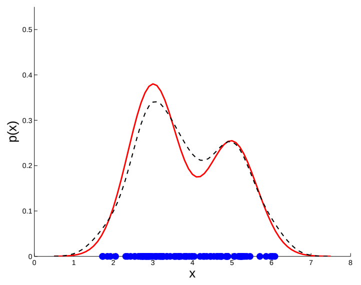
h=.1



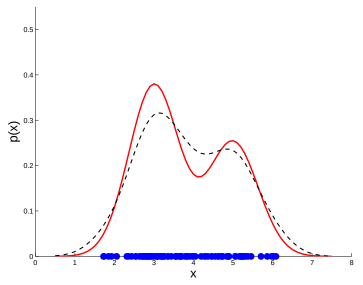
h=.2



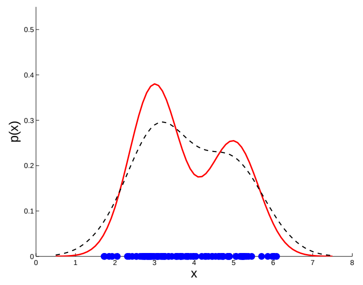
h=.3



h=.4



h=.5



h=.6

Rule of thumbs for bandwidth estimation

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.)

Kernel density estimation in dD

Kernel density can also be applied on d dimensional data. Estimate $p(\mathbf{x})$ from a set of training examples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$.

$$\hat{p}_{\mathbf{h}}(\mathbf{x}) = \frac{1}{n\, h_1\, h_2 \ldots h_d} \sum_{i=1}^{n} K\left(H^{-1}\left(\mathbf{x} - \mathbf{x}_i\right)\right)$$

where

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

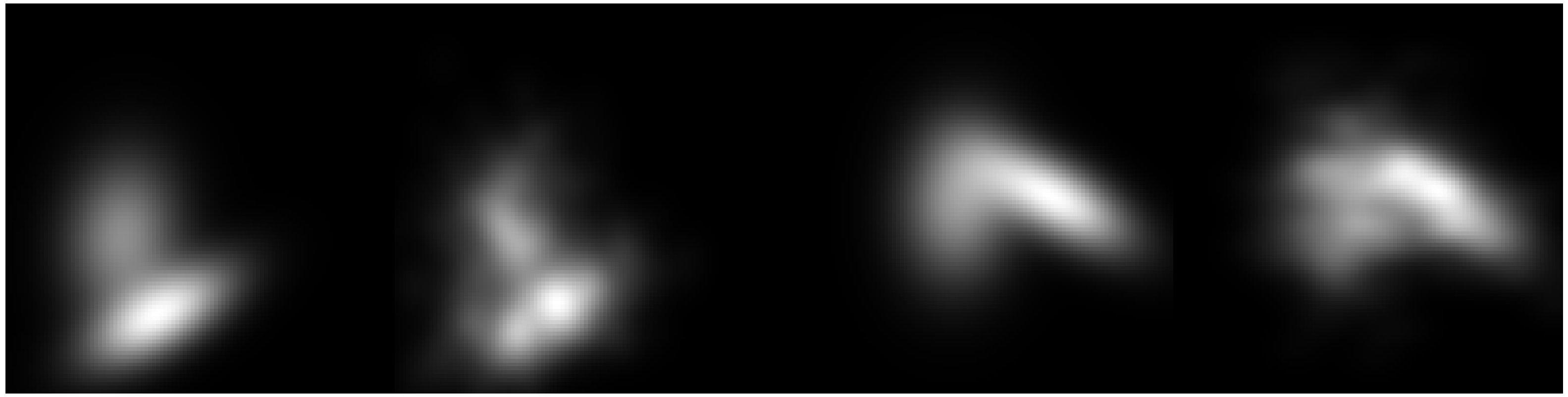
and

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

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

Revisit Lecture 4 example

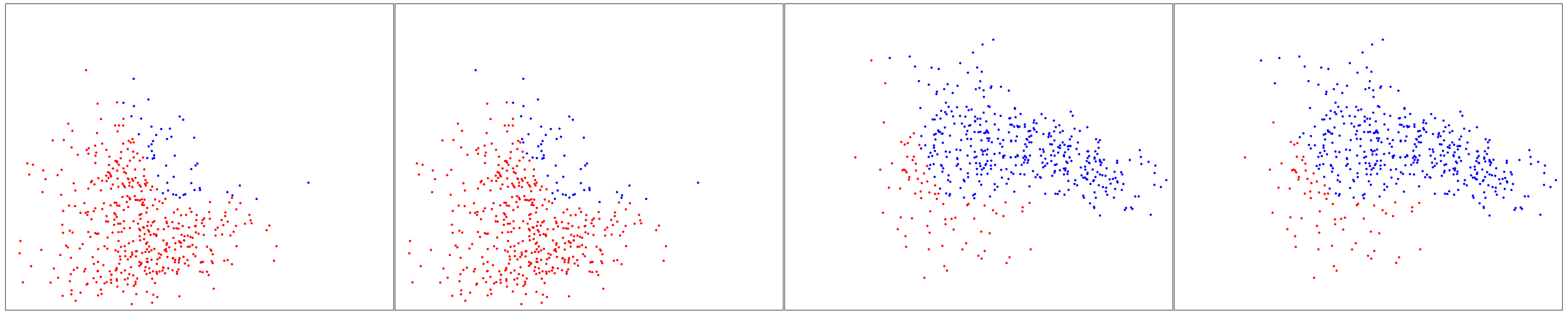
Estimate the distributions using kernel density estimation with the bandwidth set by the **rule of thumb 1** and 1000 training examples for each class.



true $p(\mathbf{x} \mid \omega_1)$ **estimated** $p(\mathbf{x} \mid \omega_1)$ **true** $p(\mathbf{x} \mid \omega_2)$ **estimated** $p(\mathbf{x} \mid \omega_2)$

Classification results

Test points from class ω_1 **Test points from class ω_2**



true $p(\mathbf{x} \mid \omega_1)$ **estimated** $p(\mathbf{x} \mid \omega_1)$ **true** $p(\mathbf{x} \mid \omega_2)$ **estimated** $p(\mathbf{x} \mid \omega_2)$

Cross-validation could also be used to find good values of the bandwidth.

Nearest Neighbours and Non-Bayesian Classification

Nearest Neighbours

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

The k Nearest Neighbour classification rule

The k **Nearest Neighbour Rule** (kNN) is a very intuitive method that classifies unlabeled examples based on their similarity to examples in the training set.

Exact steps:

For a given unlabeled example $\mathbf{x}_u \in R^d$:

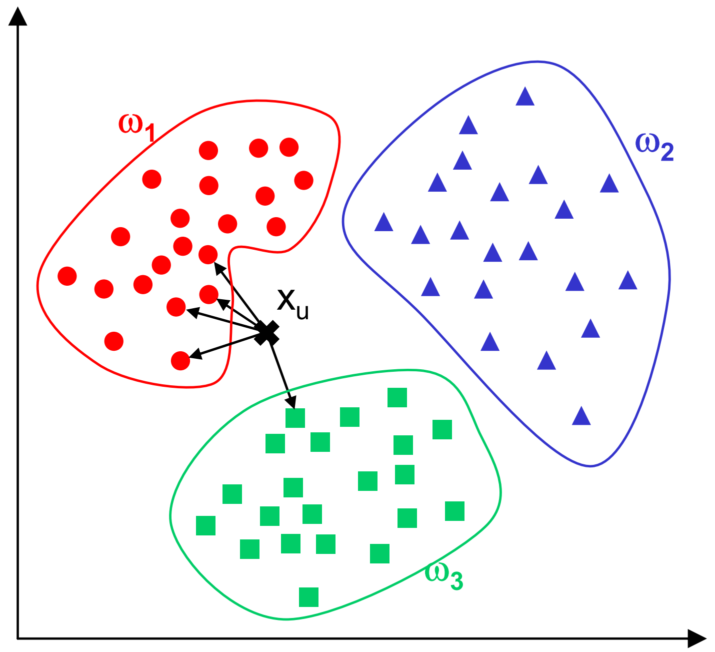
- Find the k *closest* labeled examples in the training data set.
- Assign $\mathbf{x}_u$ to the class that appears most frequently within the k-subset.

The k Nearest Neighbour classification rule

The kNN requires

- An integer k.
- A set of labeled examples (training data).
- A metric to measure *closeness*.

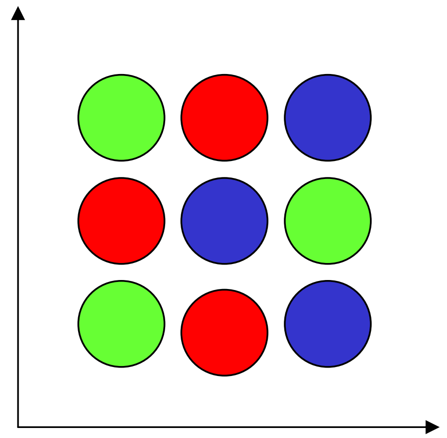
Example



- 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 $k = 5$ neighbours.
- Of the 5 closest neighbours, 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

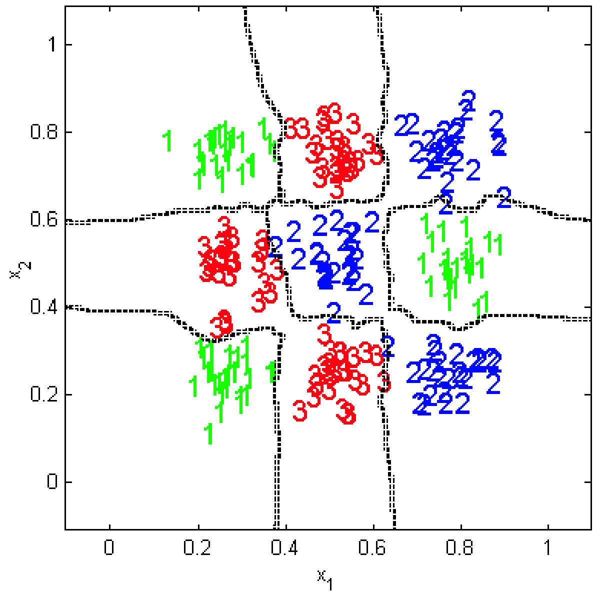
Have generated data for a 2-dimensional 3-class problem, where the class-conditional densities are multi-modal, and non-linearly separable as shown.

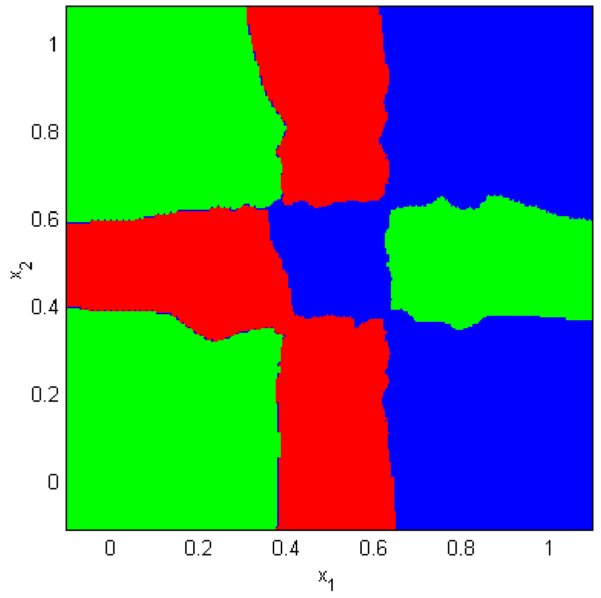


Solution:

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

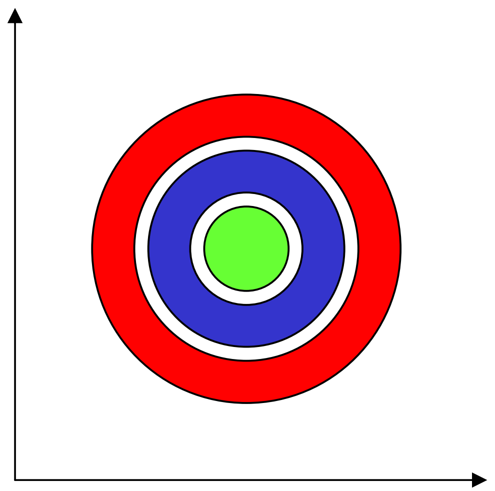
The resulting decision boundaries and regions are shown below





kNN in action: example 2

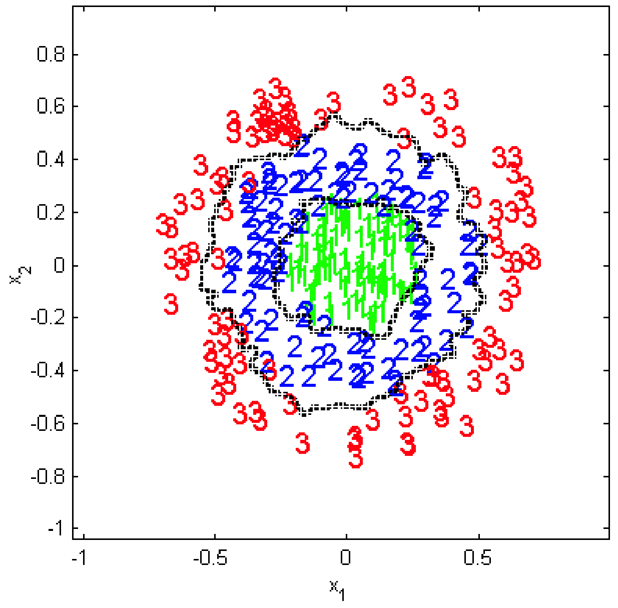
Have generated data for a 2-dimensional 3-class problem, where the class-conditional densities are unimodal and are distributed in rings around a common mean. These classes are also non-linearly separable as illustrated in the figure below

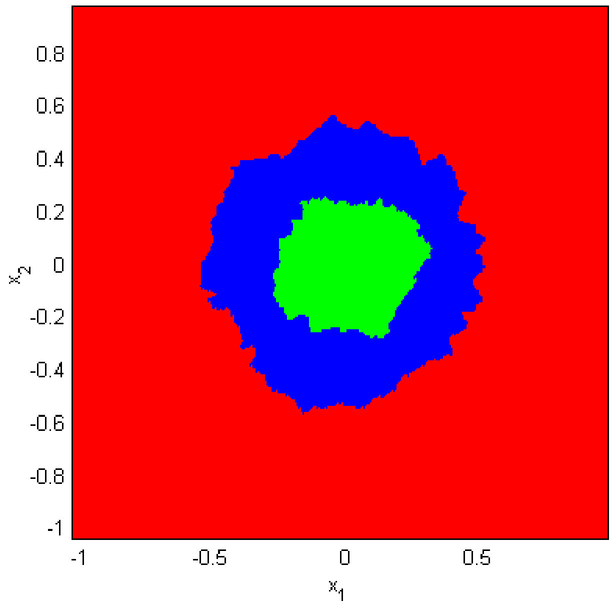


Solution:

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

The resulting decision boundaries and regions are shown below





Distance functions

The nearest neighbour 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.

Euclidean distance is the L_2 norm

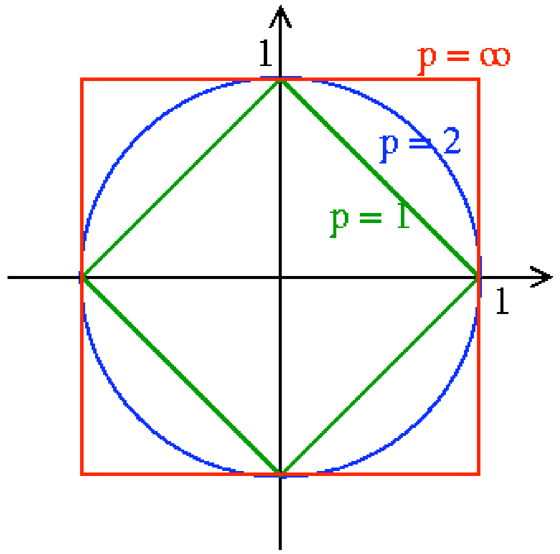
$$L_2(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^2 \right)^{\frac{1}{2}}$$

Manhattan/city block distance is the L_1 norm

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

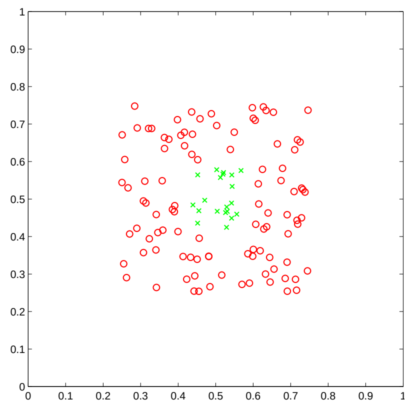
Infinity norm is the L_∞ norm

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

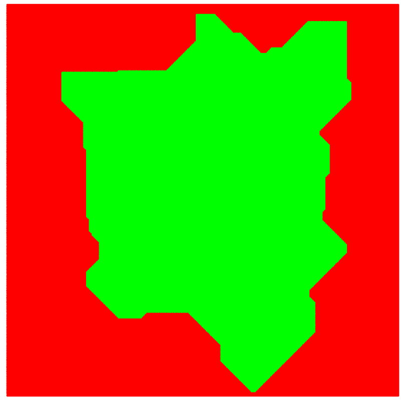


Each colored curve shows points at a distance 1.0 from the origin, measured using different values of p in the Minkowski L_p metric.

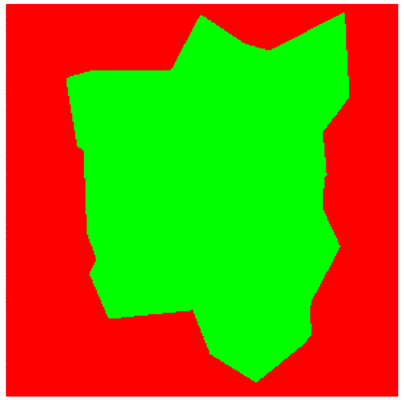
Decision boundary depends on distance fn



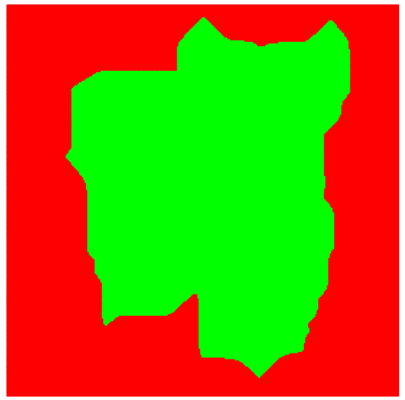
Training data



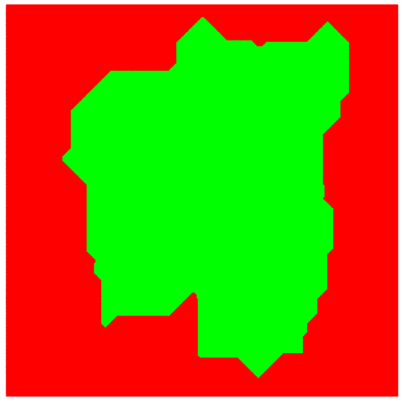
City block distance



Euclidean distance



Minkowski $p = 10$



Infinity distance

Nearest Neighbours

- Nearest Neighbours density estimation
- The k Nearest Neighbours 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

As opposed to

Eager learning algorithms

Compiles its data into a compressed description or model such as

a **density estimate** or **density parameters**
a **graph structure** and associated weights

In these cases the algorithms

- **discard** the training data **after compilation** of the **model**
- **classify** incoming patterns using the **induced model**, which is retained for future requests

Which one to choose:

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 Neighbours

- Nearest Neighbours density estimation
- The k Nearest Neighbours 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 $n \to \infty$

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

- 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

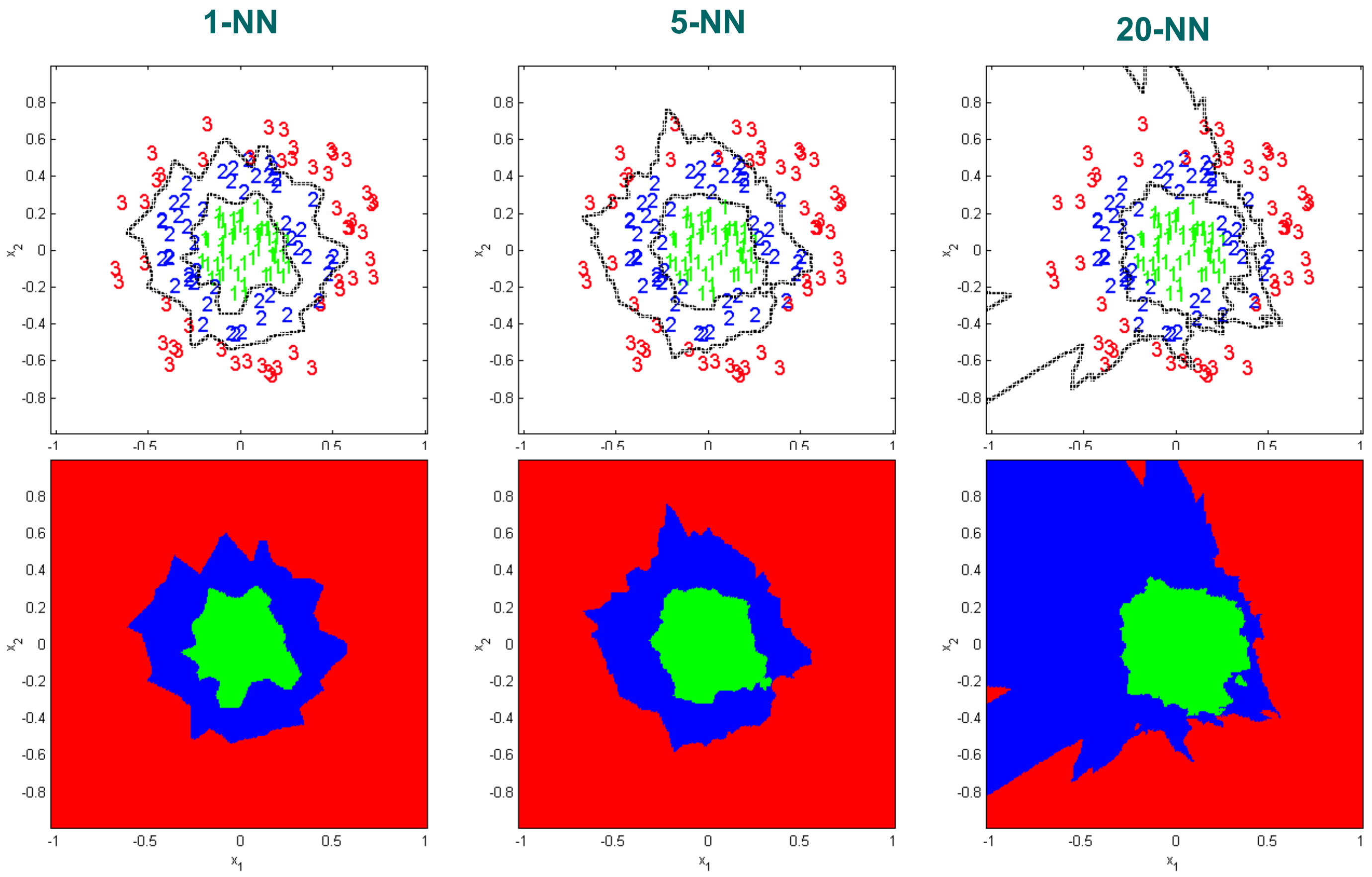
- Yields smoother decision regions.
- Provides probabilistic information

  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 far away examples are taken into account.
- Additionally it increases the computational burden.

kNN versus 1NN

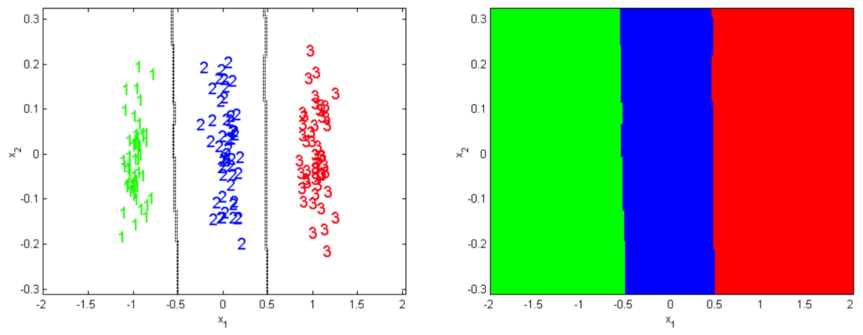


kNN and the problem of feature weighting

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

An example

Have a data set with 3-classes and 2 dimensions:

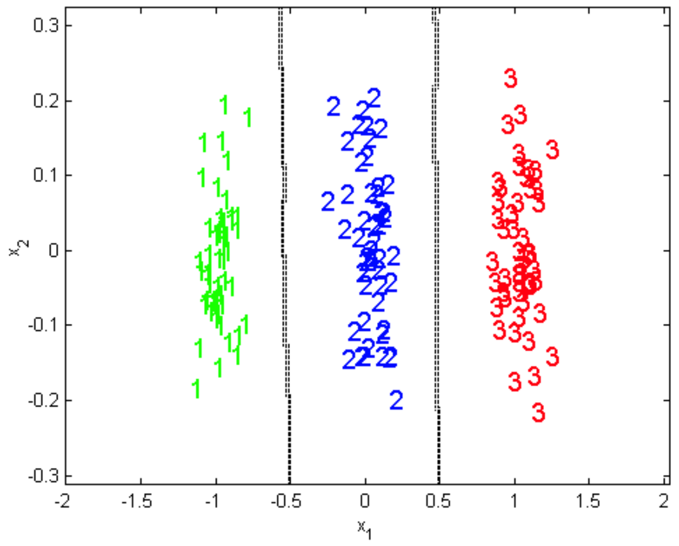


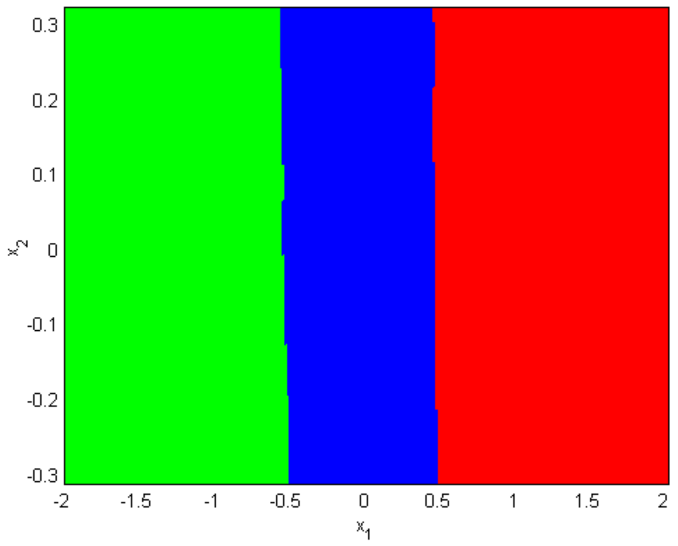
The first axis contains the discriminatory information. Class separability is excellent.

The second axis is white noise and, thus, does not contain classification information.

Example 1

If both axes are scaled properly, then kNN (k=5) finds decision boundaries fairly close to the optimal.

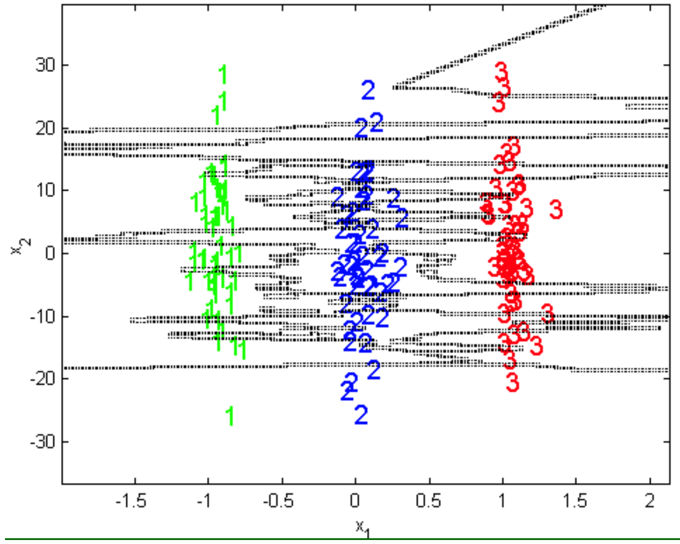


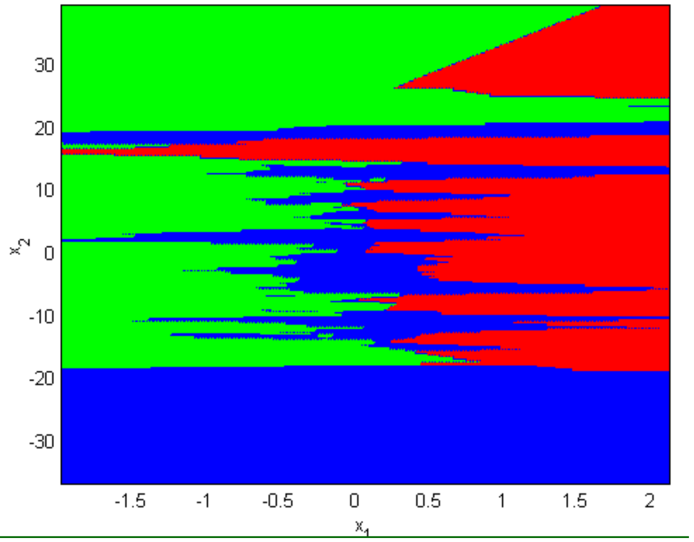


Example 2

We increase the magnitude of the second axis by two order of magnitudes (see axes tick marks).

The kNN is now biased by the large values of the second axis and its performance is very poor.





Feature weighting

The kNN classifier is sensitivity to noisy axes.

Problem:

Examine the Euclidean distance:

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

This metric can become very noisy if

- Different dimensions have different scalings,
- Most of the dimensions are irrelevant to the classification task even if all dimensions have the same scale. Explain..

Feature weighting

Solution to unequal scaling:

Normalize each dimension of the feature to $\mathcal{N}(0,1)$.

Solution to irrelevant dimensions:

Modify the Euclidean metric by a set of non-negative weights that represent the information content or **goodness** of each feature

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

Note: this procedure is identical to performing a linear transformation where the transformation matrix is diagonal with the weights placed in the diagonal elements.

Feature weighting

Feature weighting can be thought as:

Special case of feature extraction
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)

Like feature subset selection
Feature subset selection can be viewed as a special case of feature weighting where the weights can only take binary $\{0, 1\}$ values

Feature weighting

An aside:

Do not confuse feature-weighting with distance-weighting, a kNN variant that weights the contribution of each of the k nearest neighbours 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

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 quickly

Nearest Neighbours

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

Improving the nearest neighbour search procedure

The problem of nearest neighbour 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.

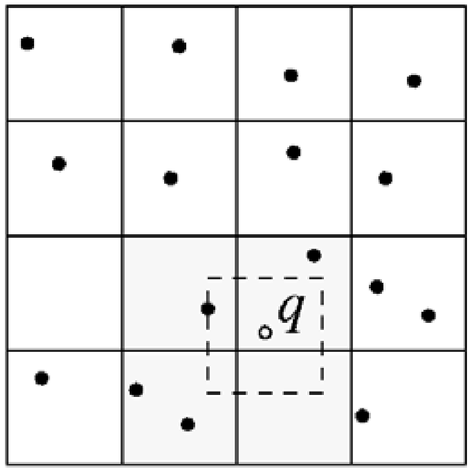
Improving the nearest neighbour search procedure

There are two classical algorithms that speed up the nearest neighbour search

- Bucketing (a.k.a Elias's algorithm) [Welch 1971]
- k-d trees [Bentley, 1975; Friedman et al, 1977]

The Bucketing algorithm

1. The space is divided into identical cells and for each cell the data points inside it are stored in a list

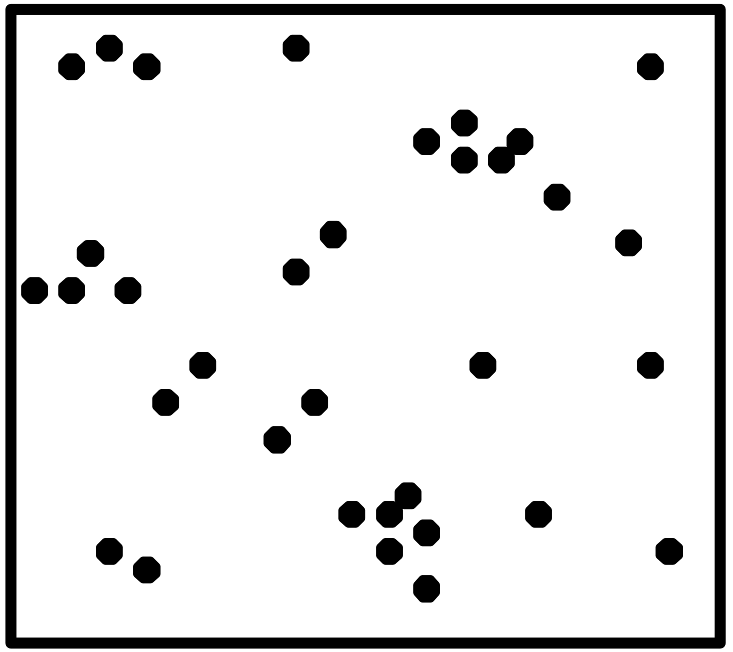


2. Each cell's minimum possible distance to the query point is computed (fast operation).

3. Cells are examined in increasing order of these minimum distances and for each cell the distance is computed between its internal data points and the query point.

4. The search ends when the minimum distance from the query point to the cell exceeds the current minimum distance to a point.

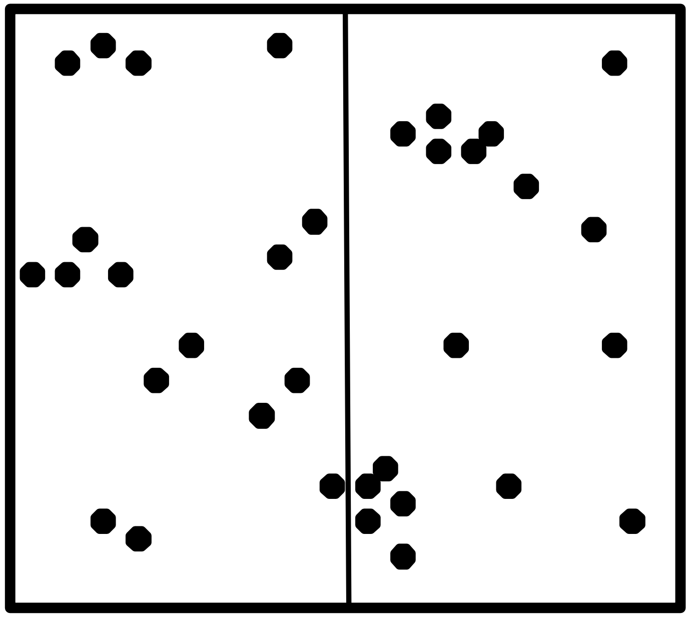
KD-tree construction

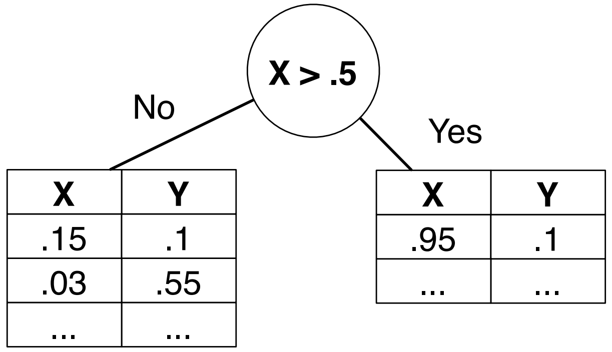


X	Y
.15	.1
.03	.55
.95	.1
...	...

Start with a list of d-dimensional points

KD-tree construction



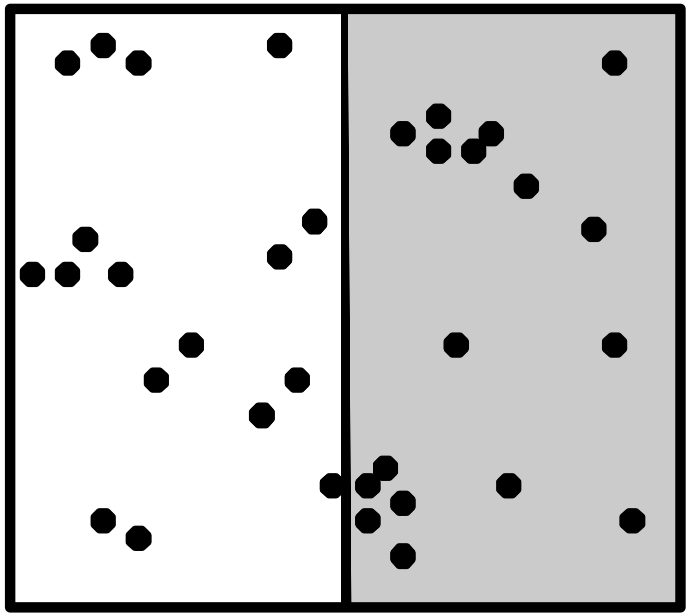


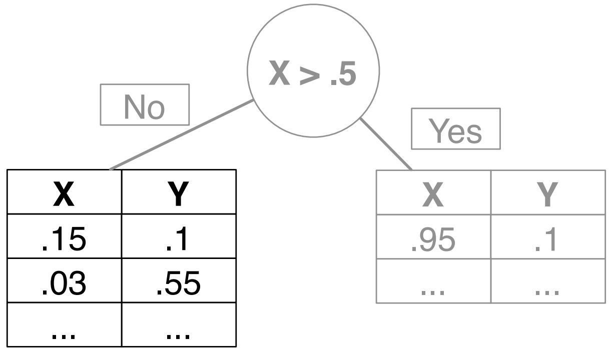
X	Y
.15	.1
.03	.55
...	...

X	Y
.95	.1
...	...

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



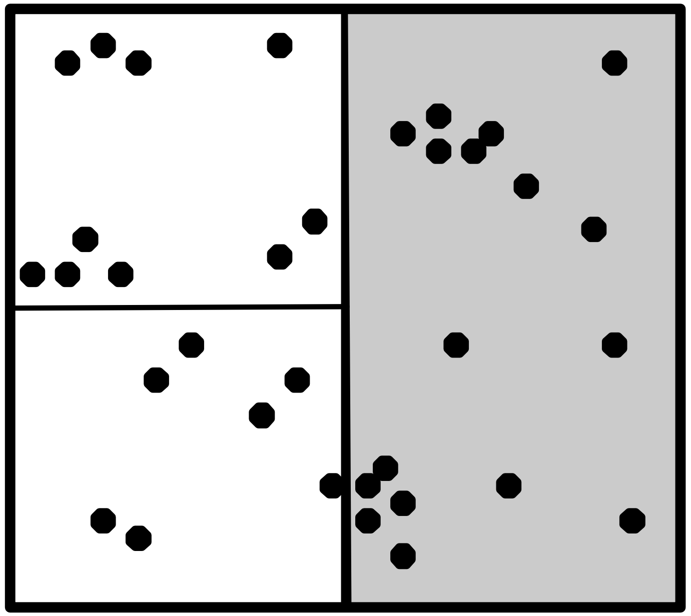


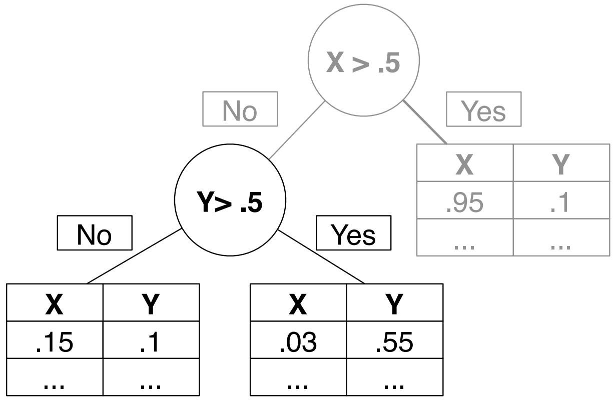
X	Y
.15	.1
.03	.55
...	...

X	Y
.95	.1
...	...

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

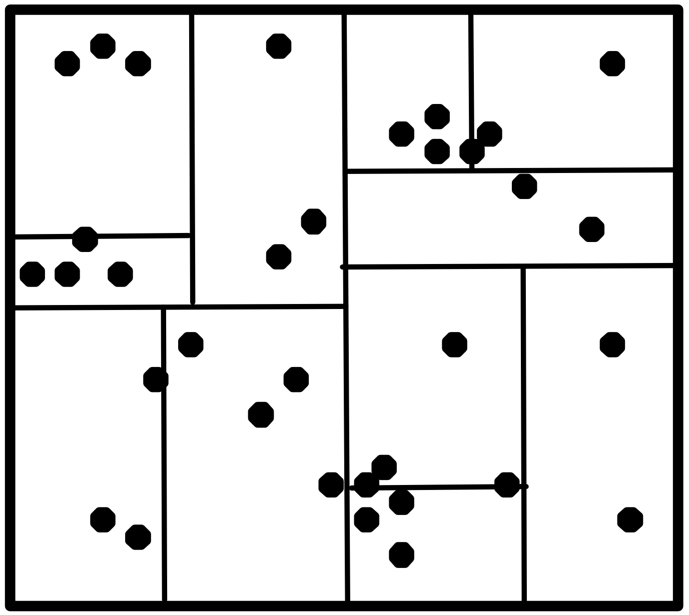
KD-tree construction

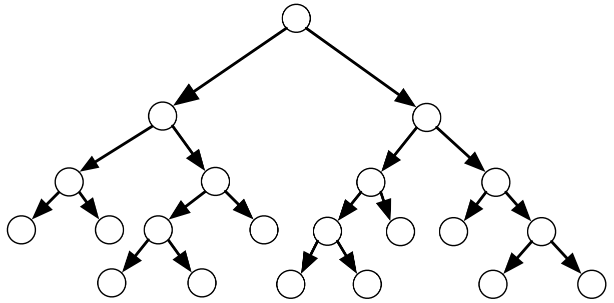




Consider each group separately and possibly split again (along same/different dimension).

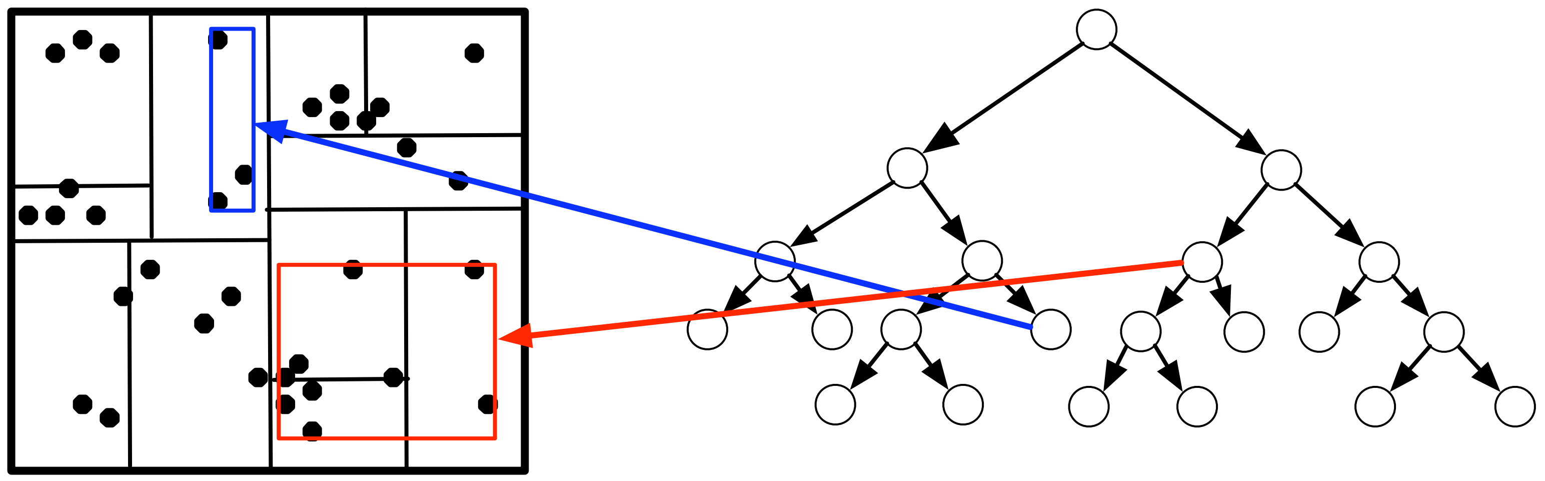
KD-tree construction





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.

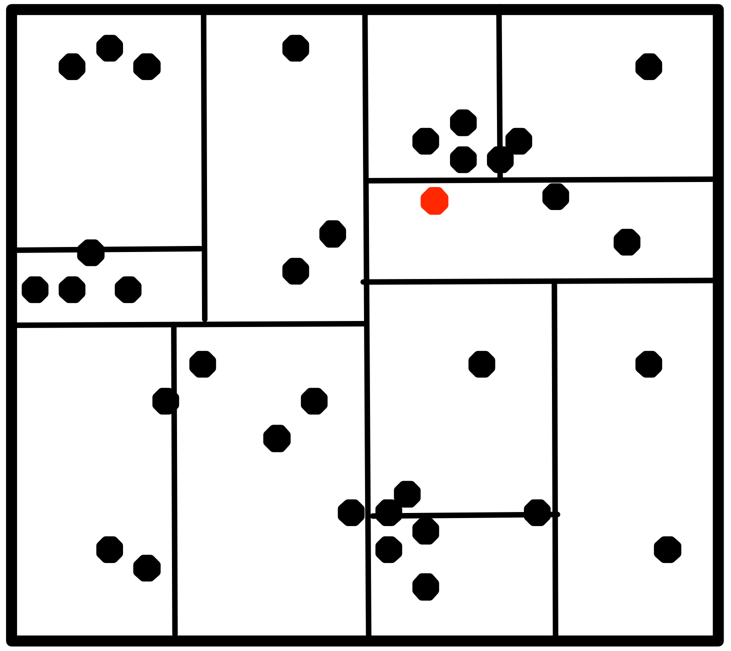
KD-tree construction in words

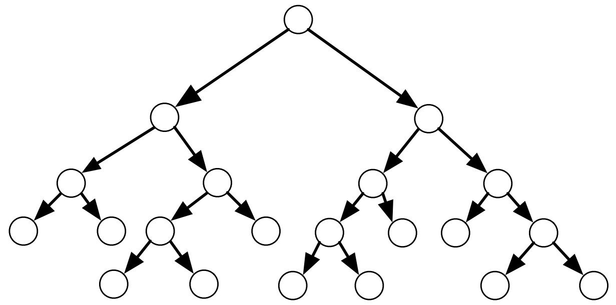
A k-d tree is a generalization of a binary search tree in high dimensions

- 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 k-d tree partitions 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.

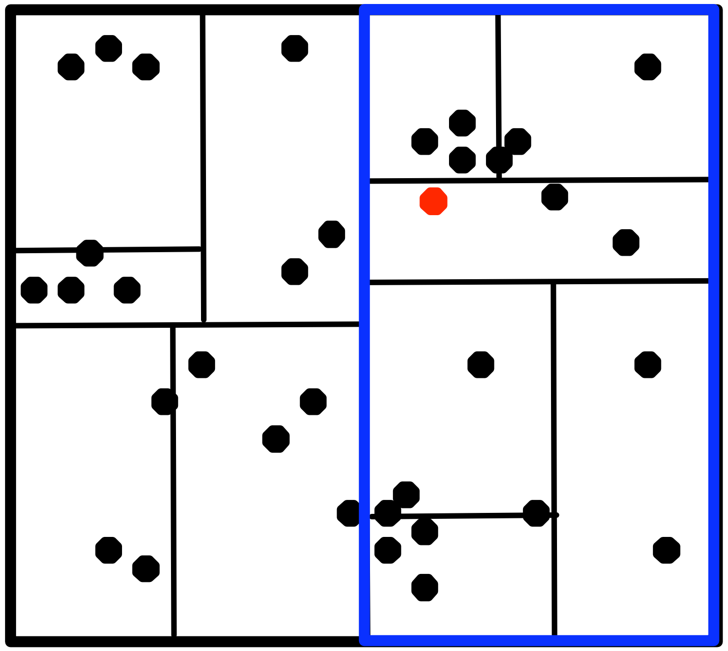
Nearest neighbour with KD-trees

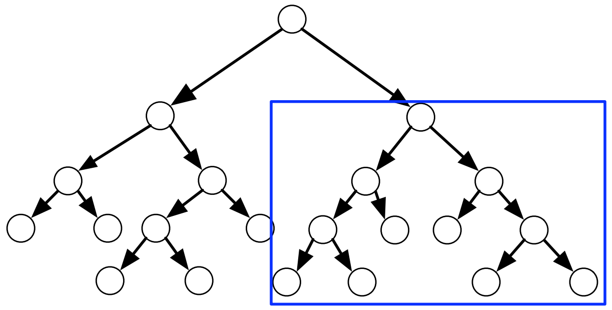




Traverse the tree looking for the nearest neighbor of the query point.

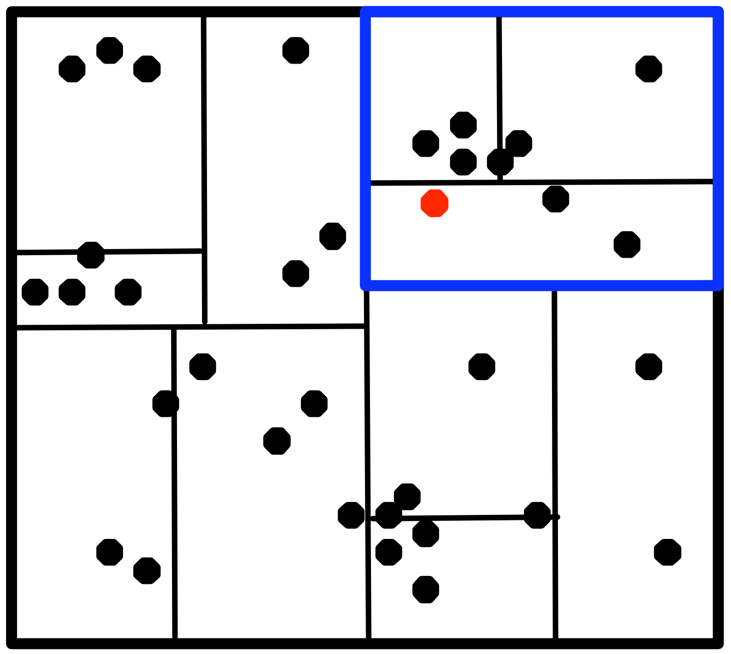
Nearest neighbour with KD-trees

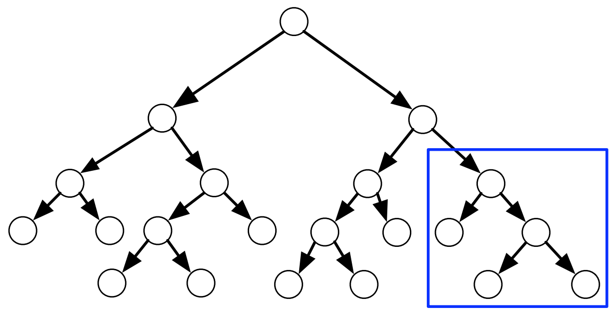




Examine nearby points first: Explore the branch of the tree that is closest to the query point first. Descend to query point's leaf node.

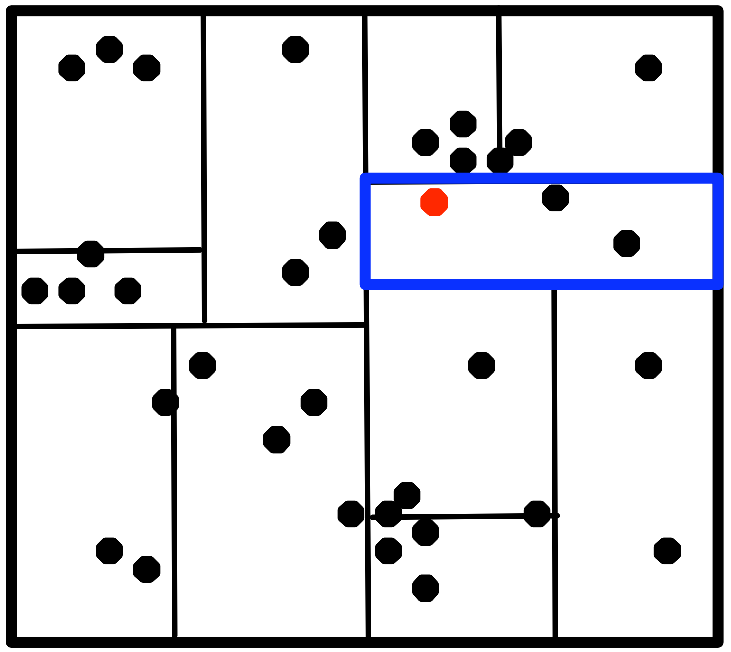
Nearest neighbour with KD-trees

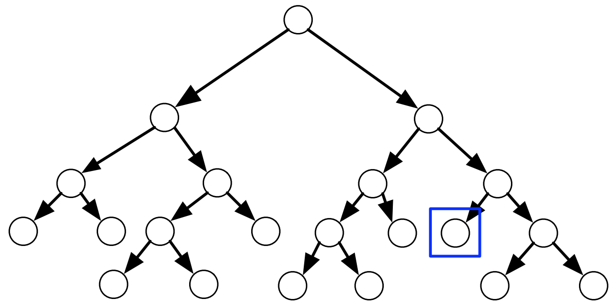




Examine nearby points first: Explore the branch of the tree that is closest to the query point first. Descend to query point's leaf node.

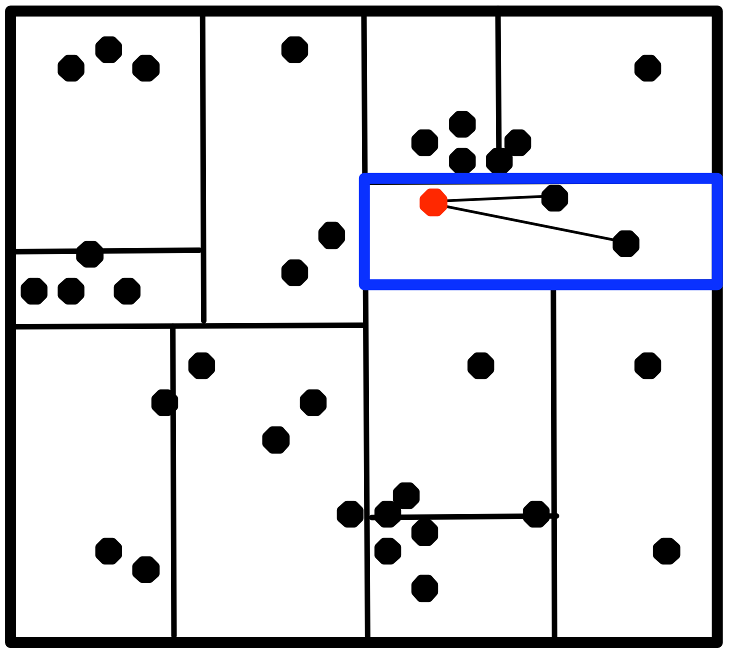
Nearest neighbour with KD-trees

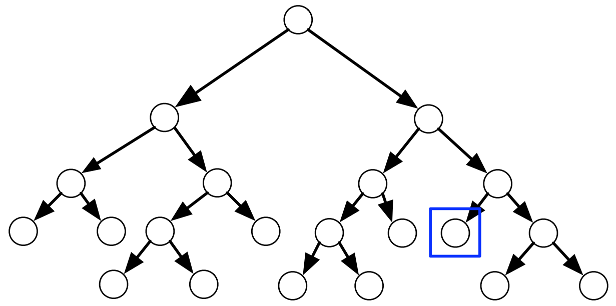




When leaf node is reached: compute the distance to each point in the node.

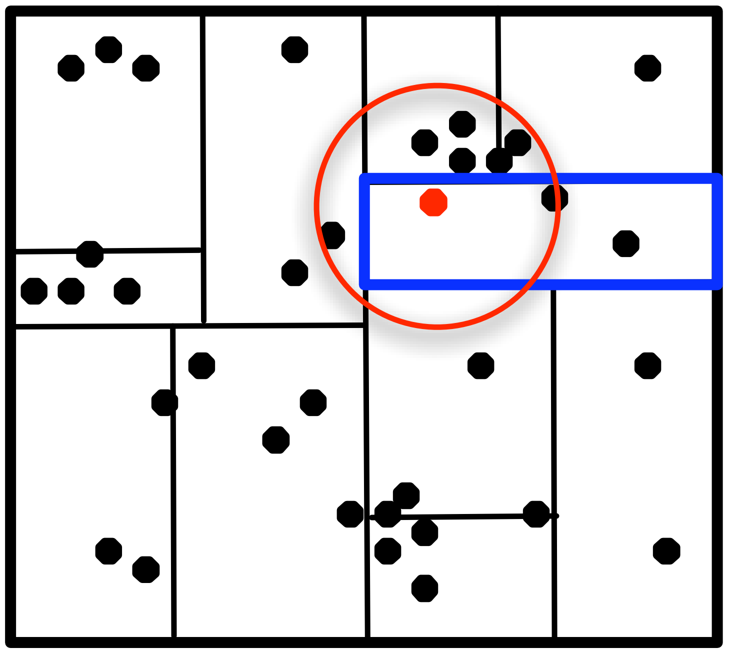
Nearest neighbour with KD-trees

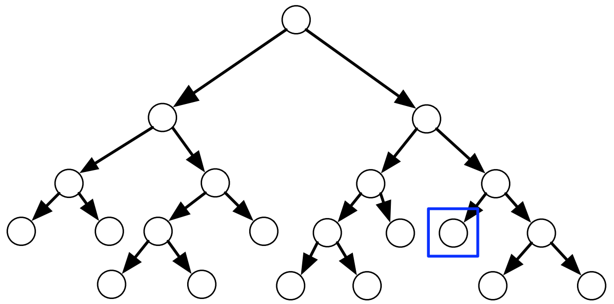




When a leaf node is reached: compute the distance to each point in the node.

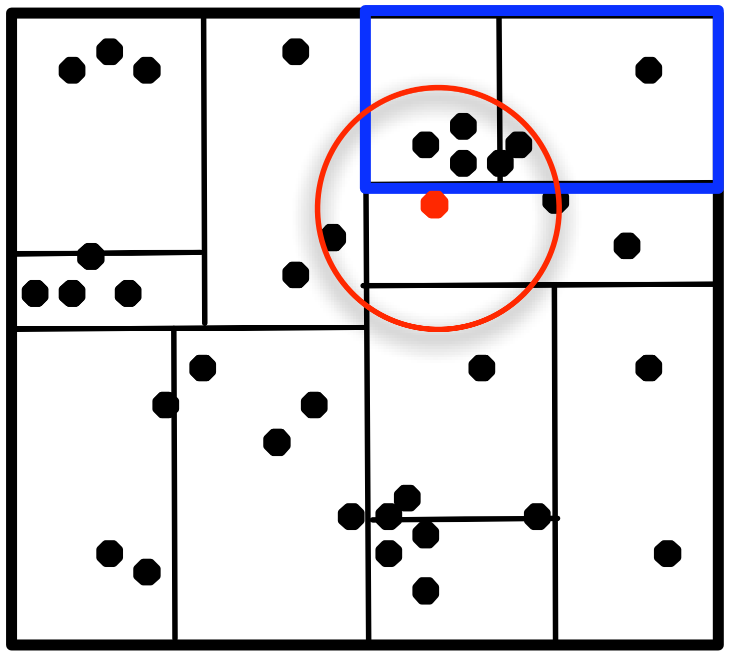
Nearest neighbour with KD-trees

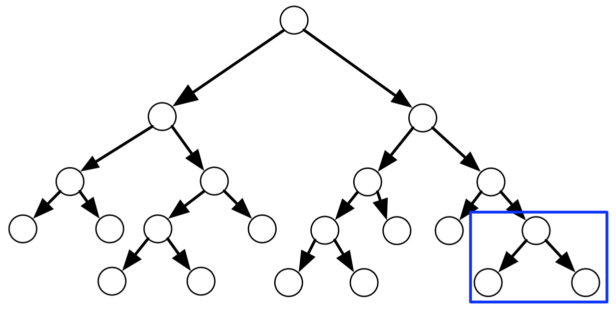




When a leaf node is reached: compute the distance to each point in the node.

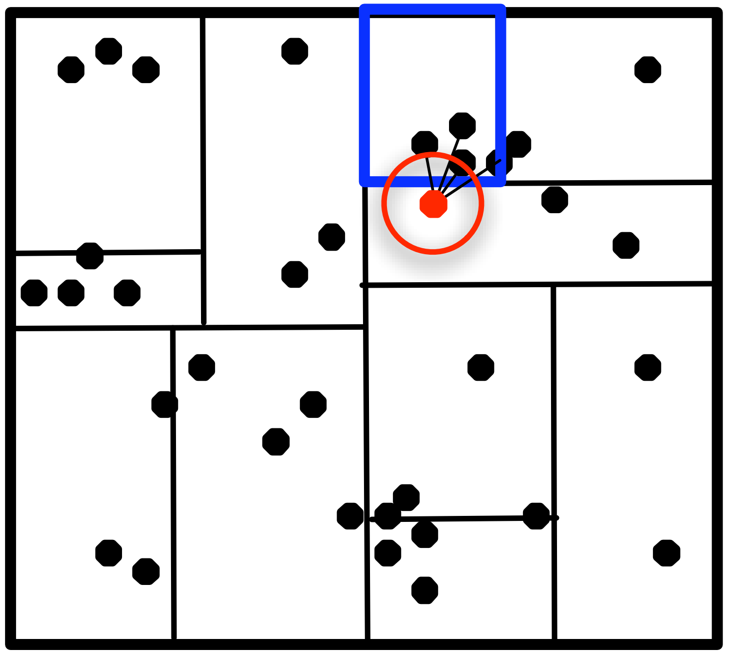
Nearest neighbour with KD-trees

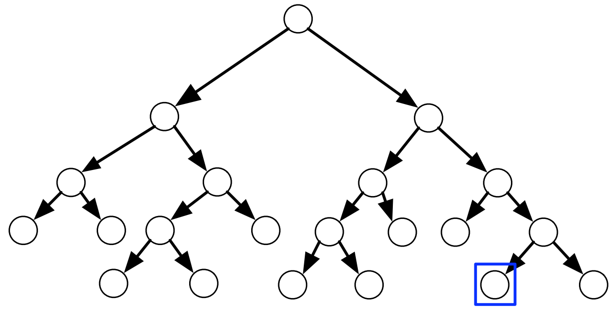




Then we can backtrack and try the other branch at each node visited.

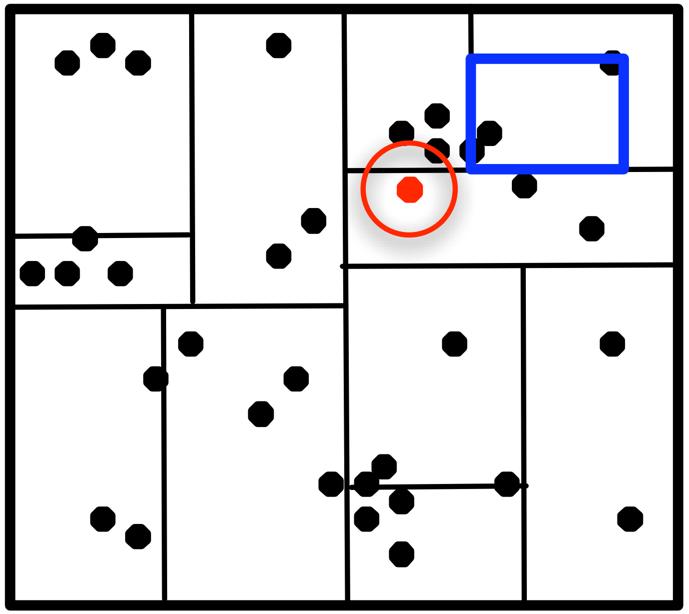
Nearest neighbour with KD-trees

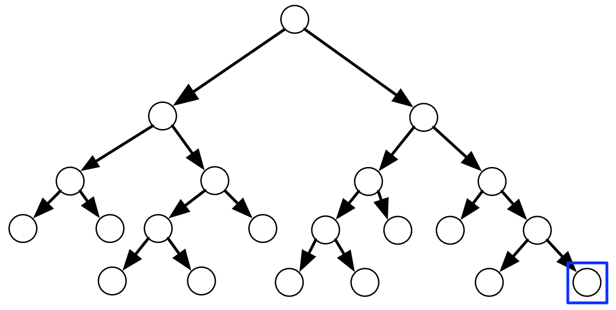




Each time a new closest node is found, we can update the distance bounds.

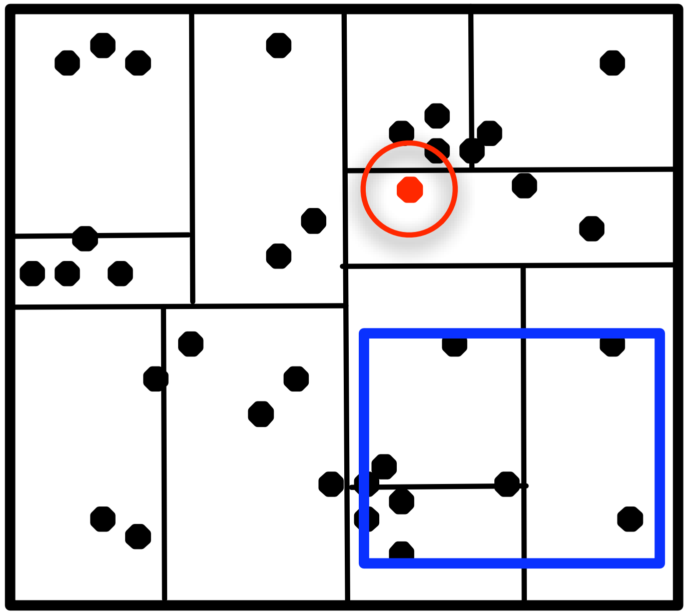
Nearest neighbour with KD-trees

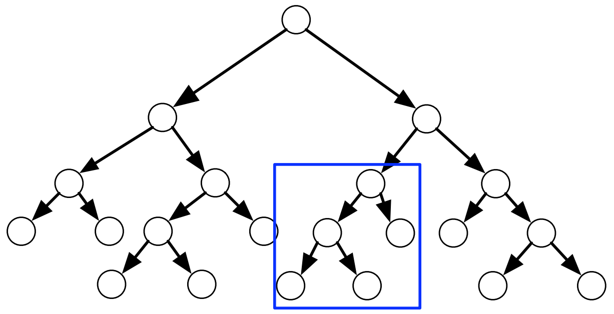




Using the current smallest distance and the bounds of the data below each node, prune parts of the tree that can **NOT** include the nearest neighbour.

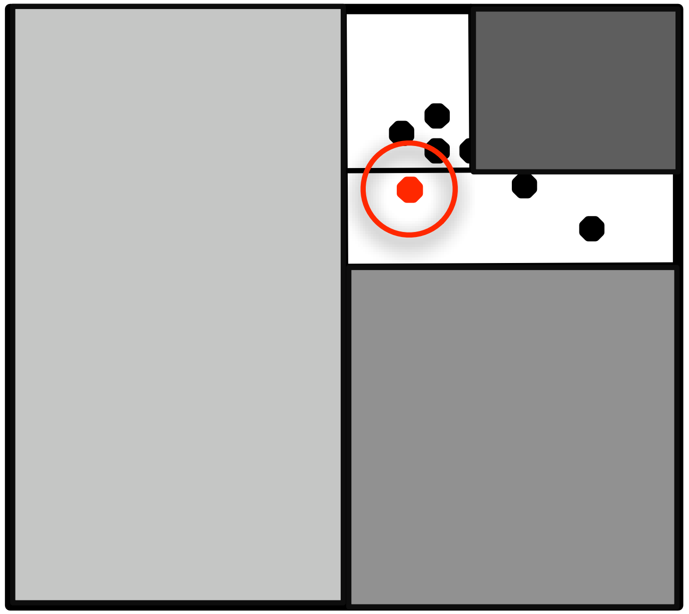
Nearest neighbour with KD-trees

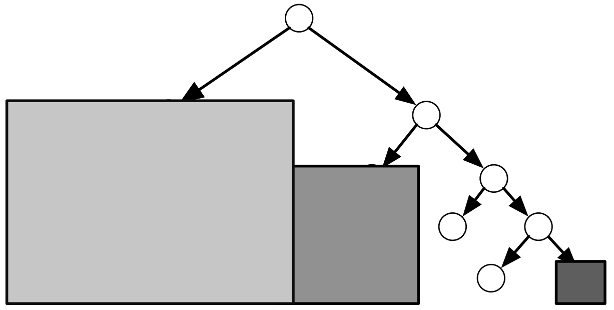




Using the current smallest distance and the bounds of the data below each node, prune parts of the tree that can **NOT** include the nearest neighbour.

Nearest neighbour with KD-trees





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

Nearest Neighbour serch with KD-trees

Advantages

- Finding the nearest point is $O(\log n)$ operation in the case of n randomly distributed points. As opposed to $O(n)$ for an exhaustive search.
- Can easily be converted to find an approximate nearest neighbour and hence to run much faster.

Disadvantages

- kd-trees are not suitable for efficiently finding the nearest neighbour in high dimensional spaces.
- **Rule of thumb** if the dimensionality is d and the number of data points, n, is $n >> 2d$, then using kd-trees will generally be better than an exhaustive search. Otherwise, it will not be.

Pen & Paper Assignment

- Details available on the course website.
- You will perform some simple nearest neighbour classifications.
- Mail me about any errors you spot in the Exercise notes.
- I will notify the class about errors spotted and corrections via the course website and mailing list.