

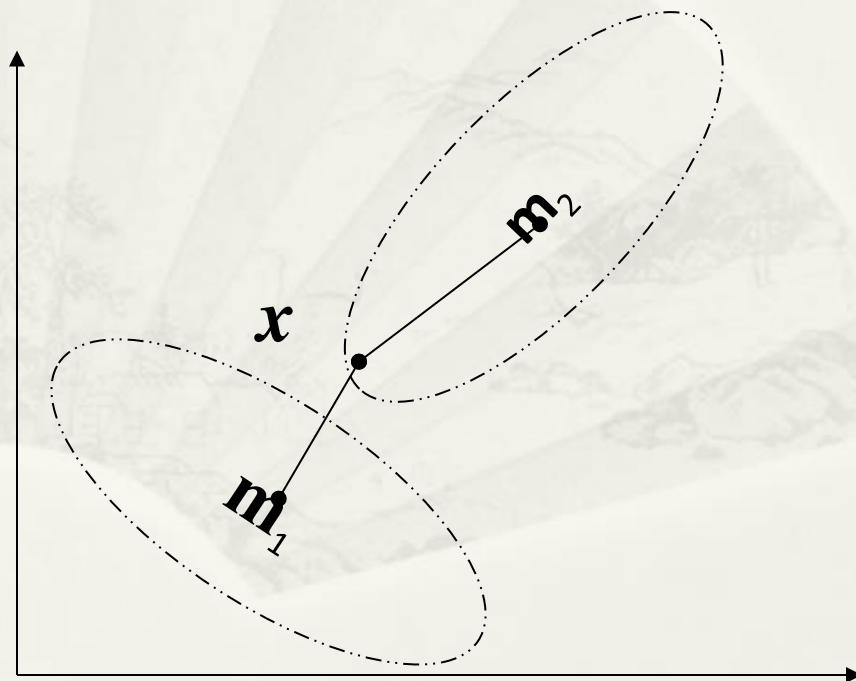
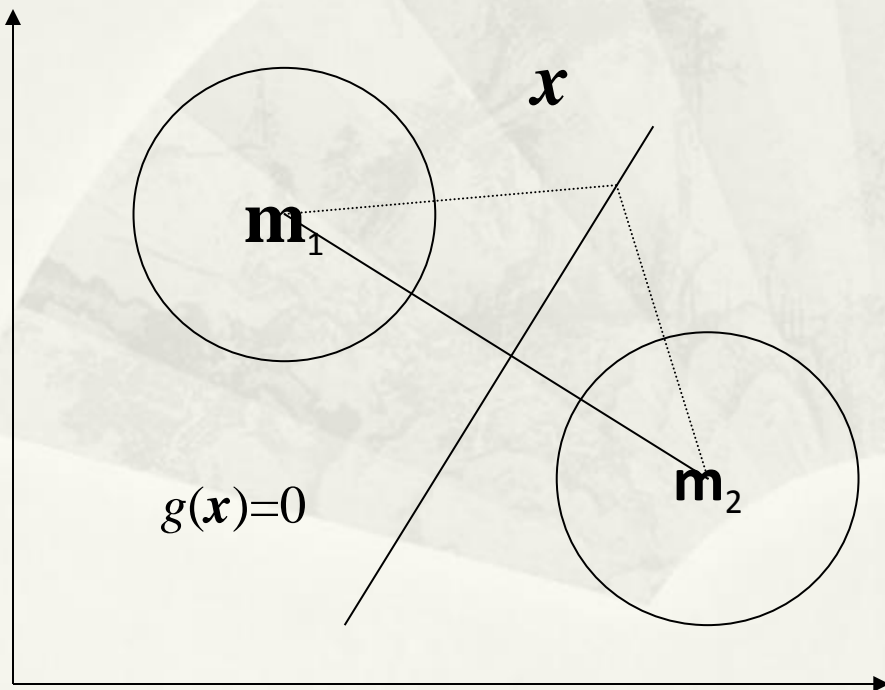
# 其他模型

基于非参数回归的分类算法

# 最近邻估计算法及其改进

# 引言

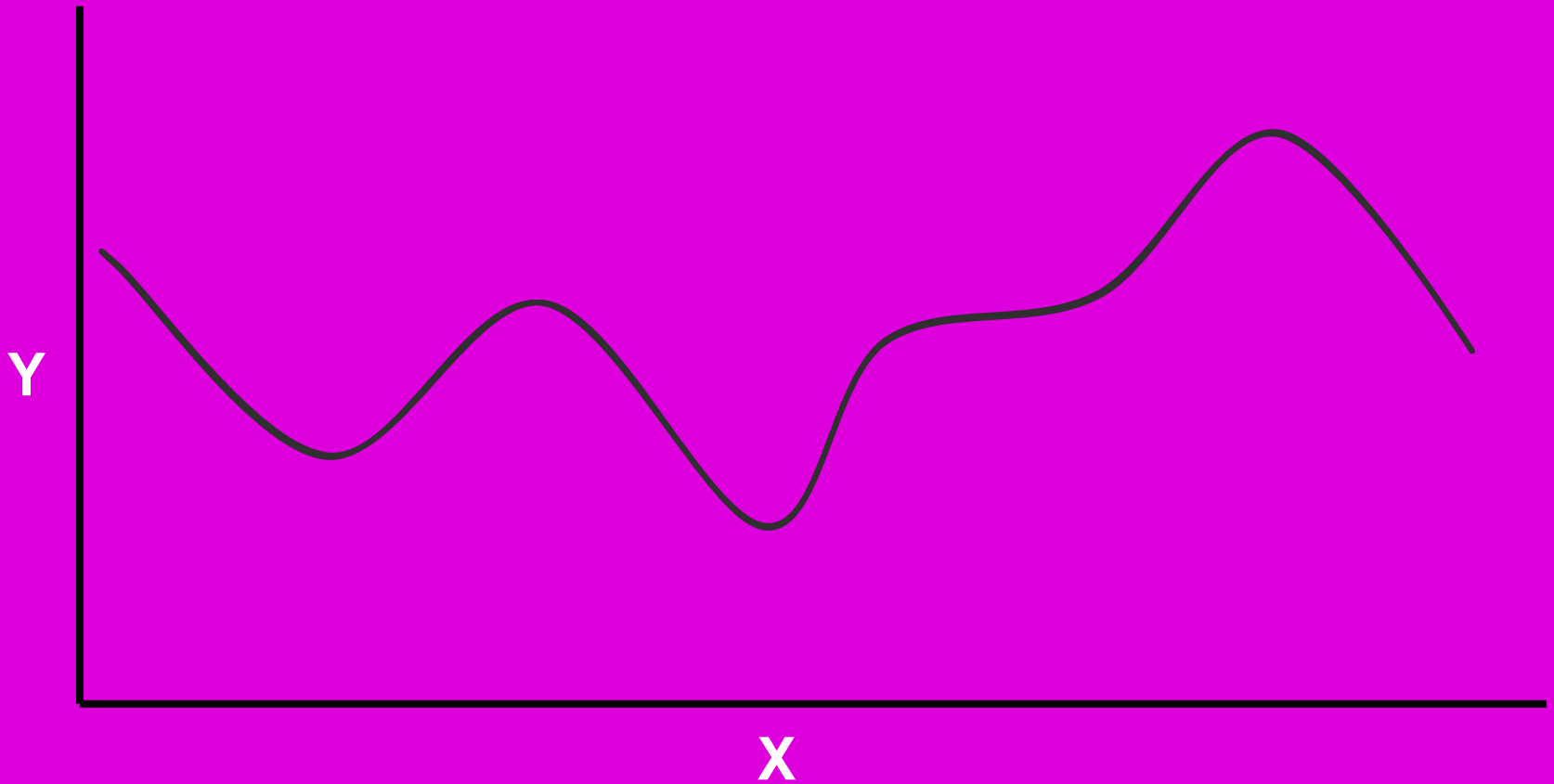
- \* **最小距离分类器**: 它将各类训练样本划分成若干子类, 并在**每个子类中确定代表点**。未知样本的类别则以其与这些代表点距离最近作决策。该方法的缺点是所选择的代表点并不一定能很好地代表各类, 其后果将使错误率增加。



# Idea of Local Regression

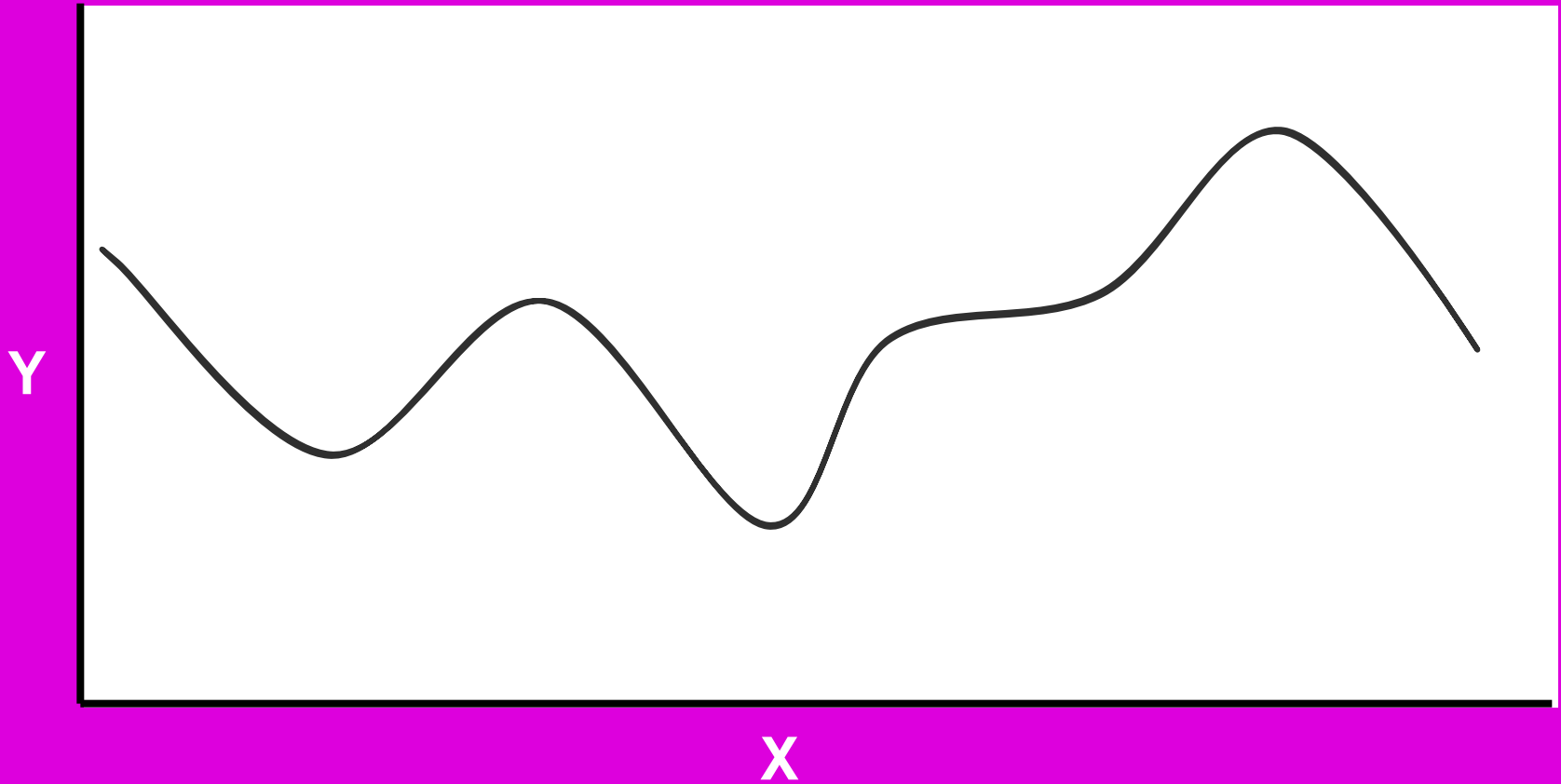
---

It is difficult to find an appropriate parametric



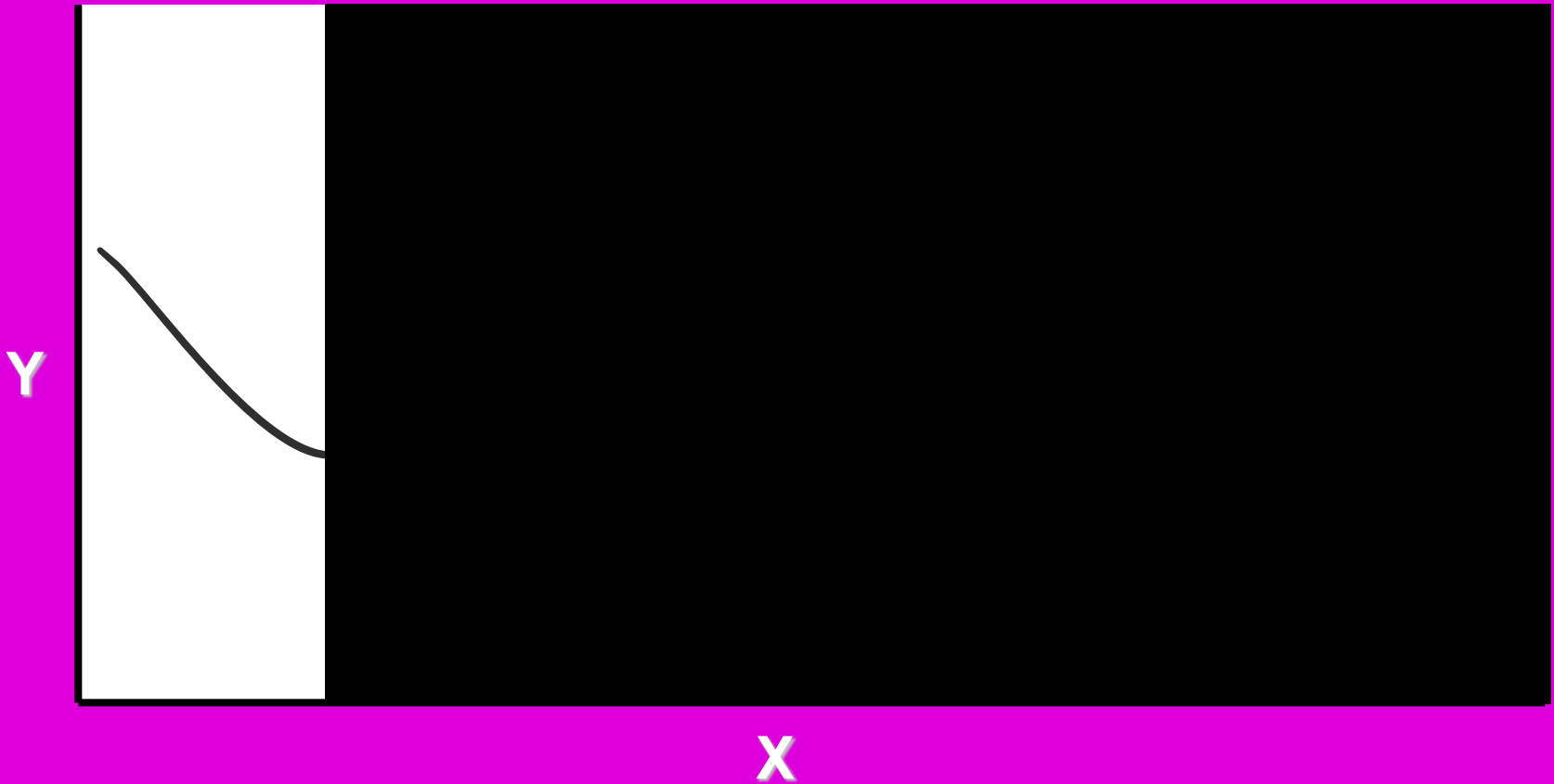
# Idea of Local Regression

Locally such curves can be well approximated



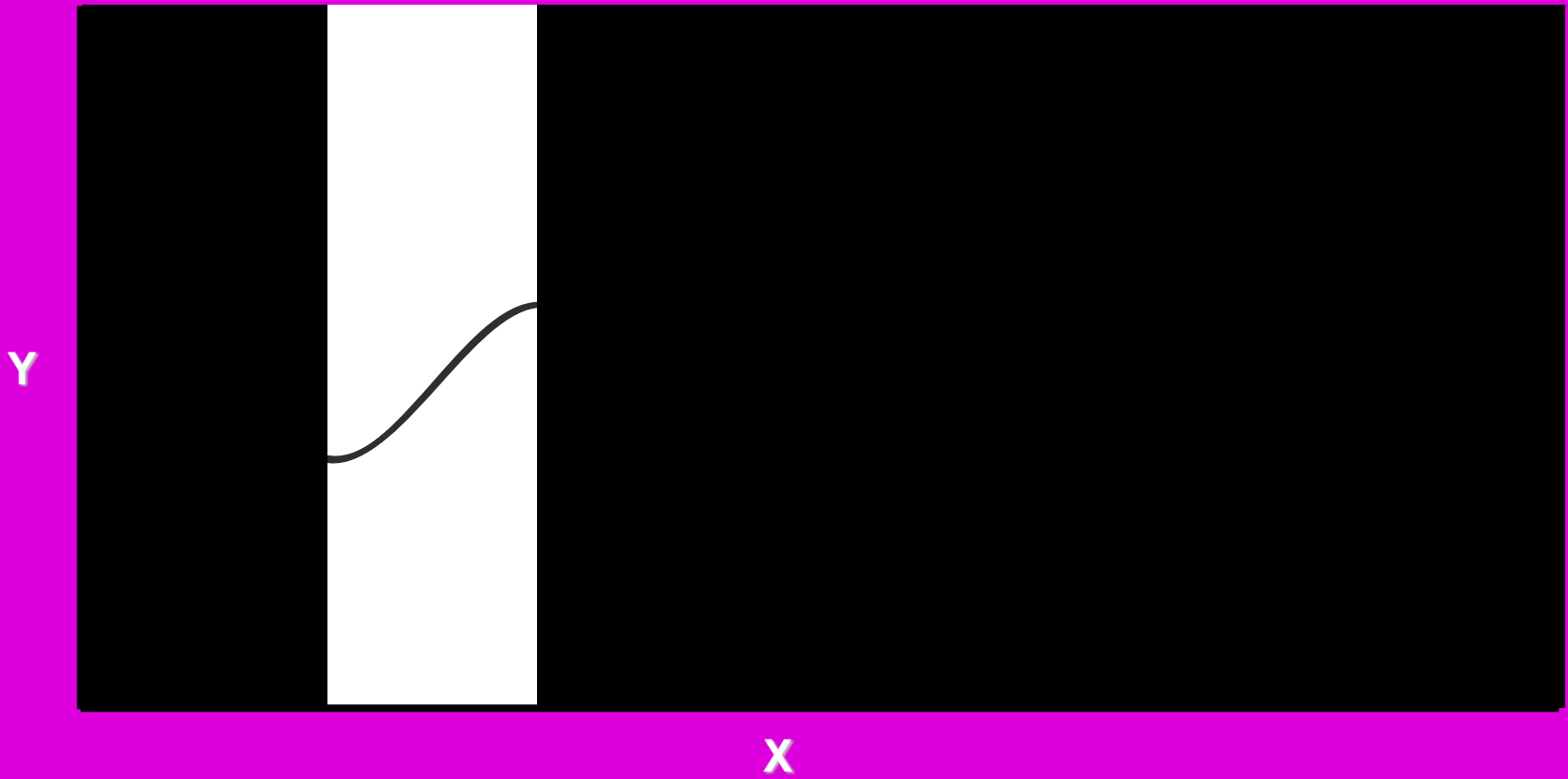
# Idea of Local Regression

Locally such curves can be well approximated



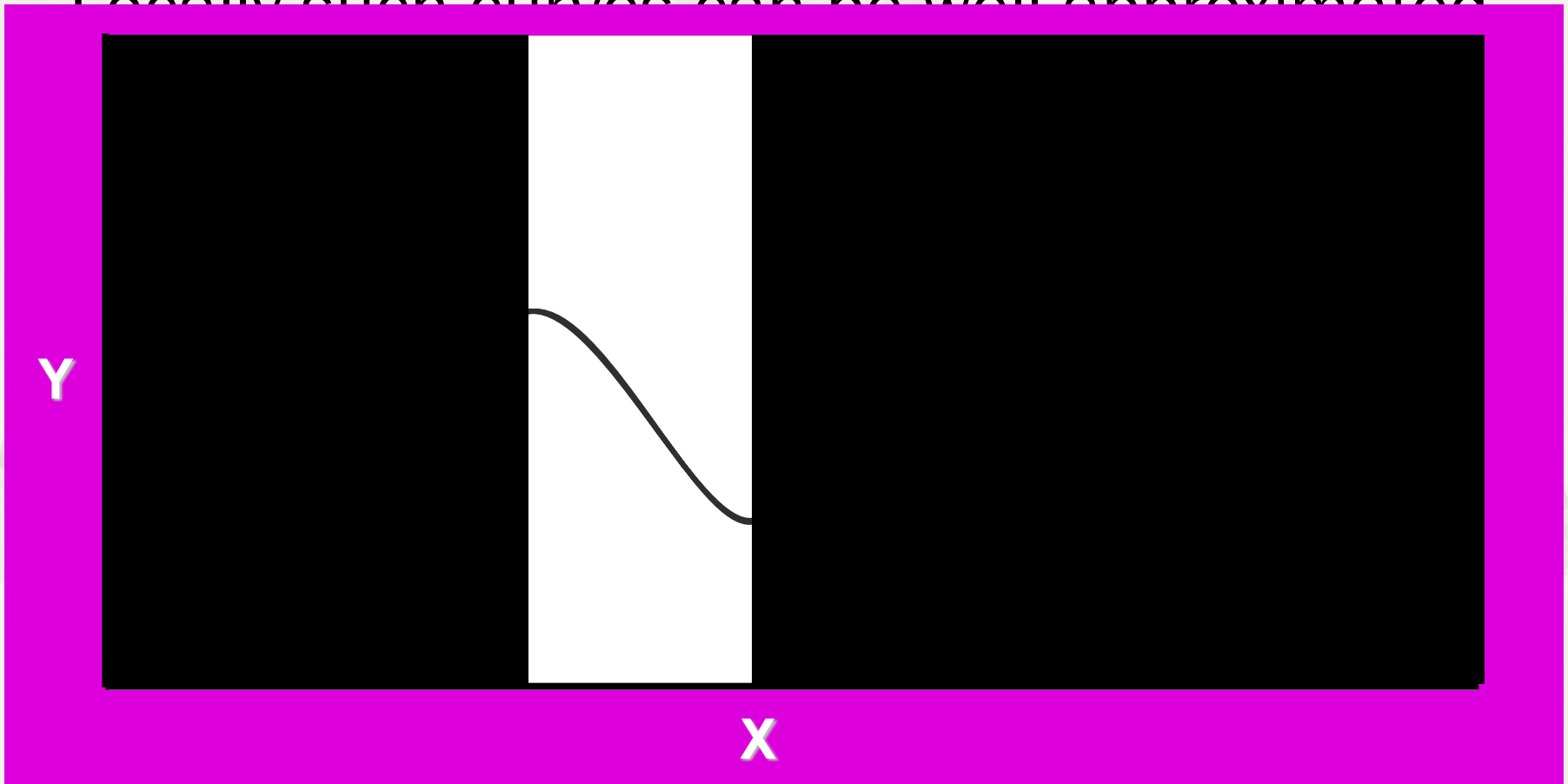
# Idea of Local Regression

Locally such curves can be well approximated



# Idea of Local Regression

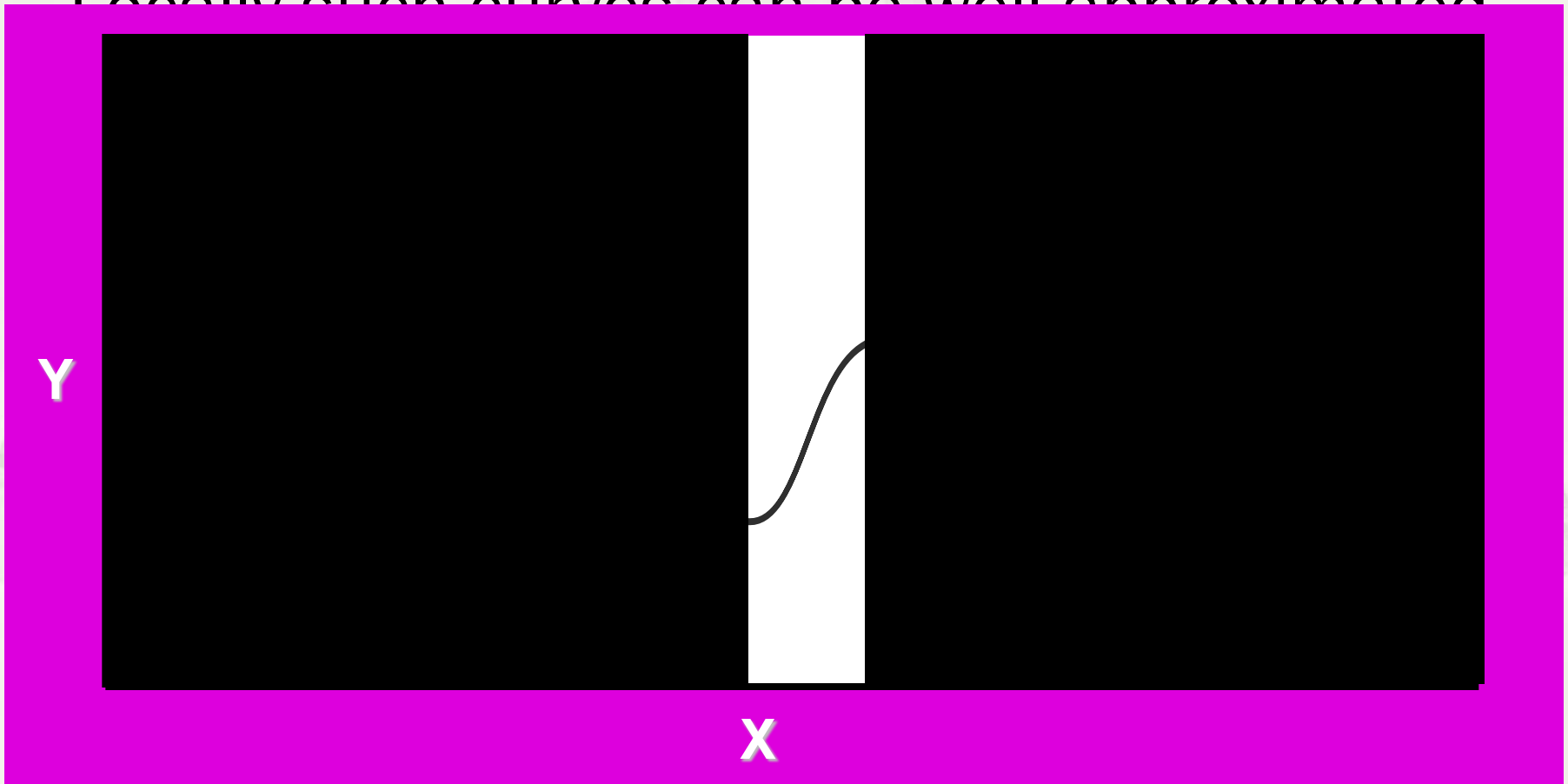
Locally such curves can be well approximated





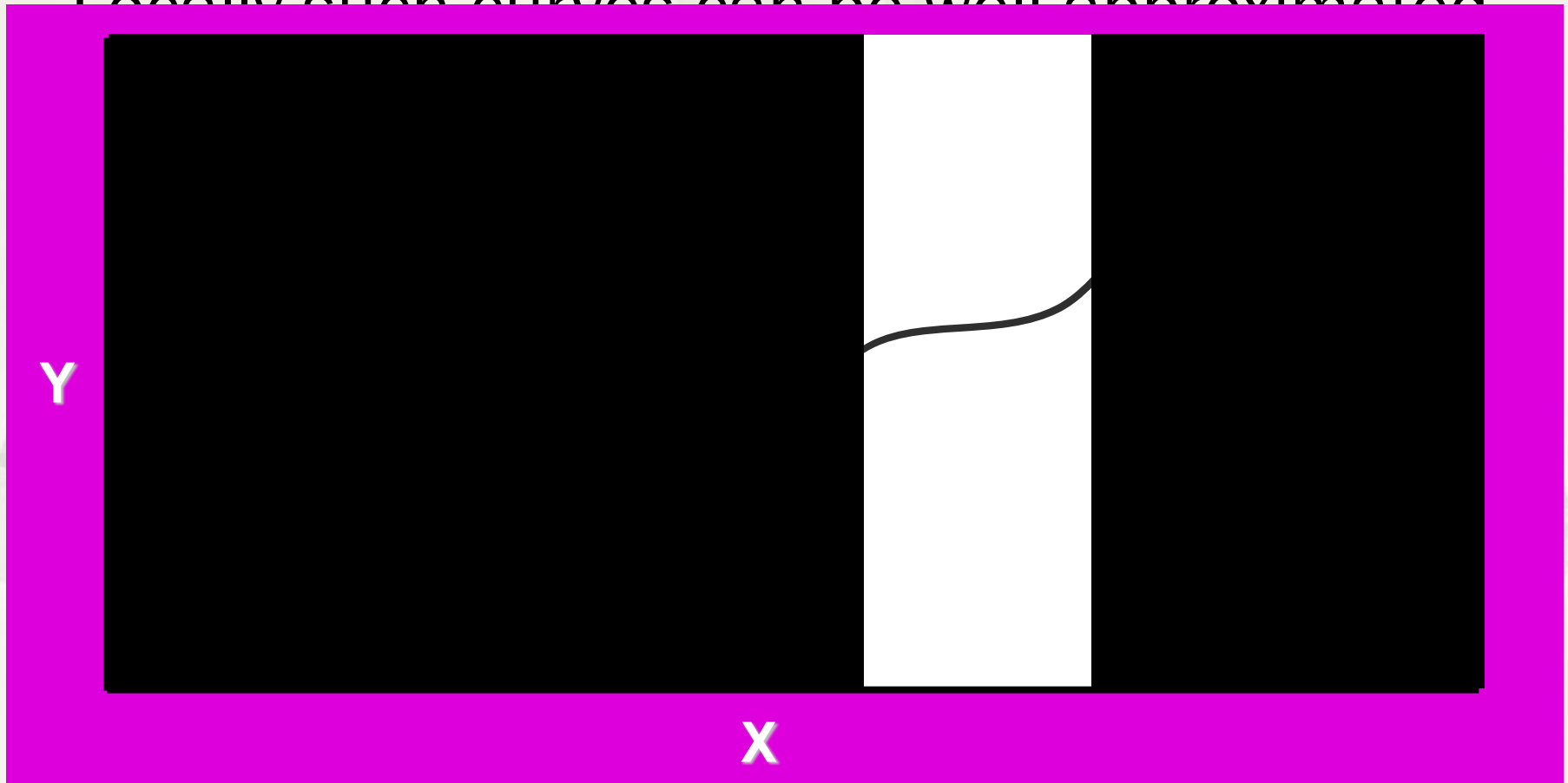
# Idea of Local Regression

Locally such curves can be well approximated



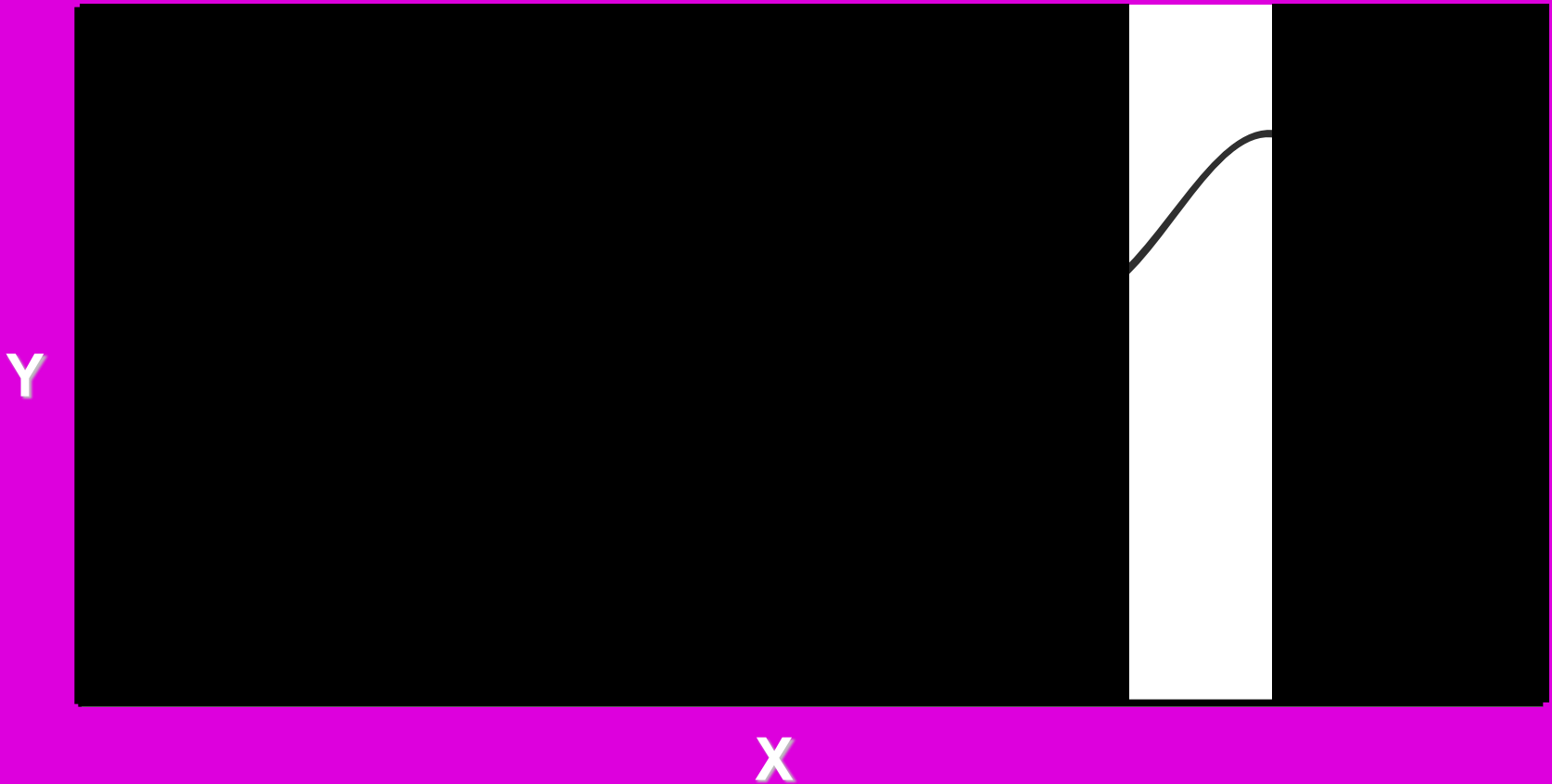
# Idea of Local Regression

Locally such curves can be well approximated



# Idea of Local Regression

Locally such curves can be well approximated

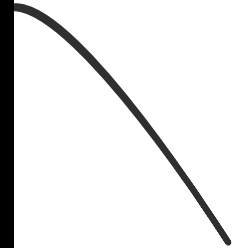


# Idea of Local Regression

Locally such curves can be well approximated

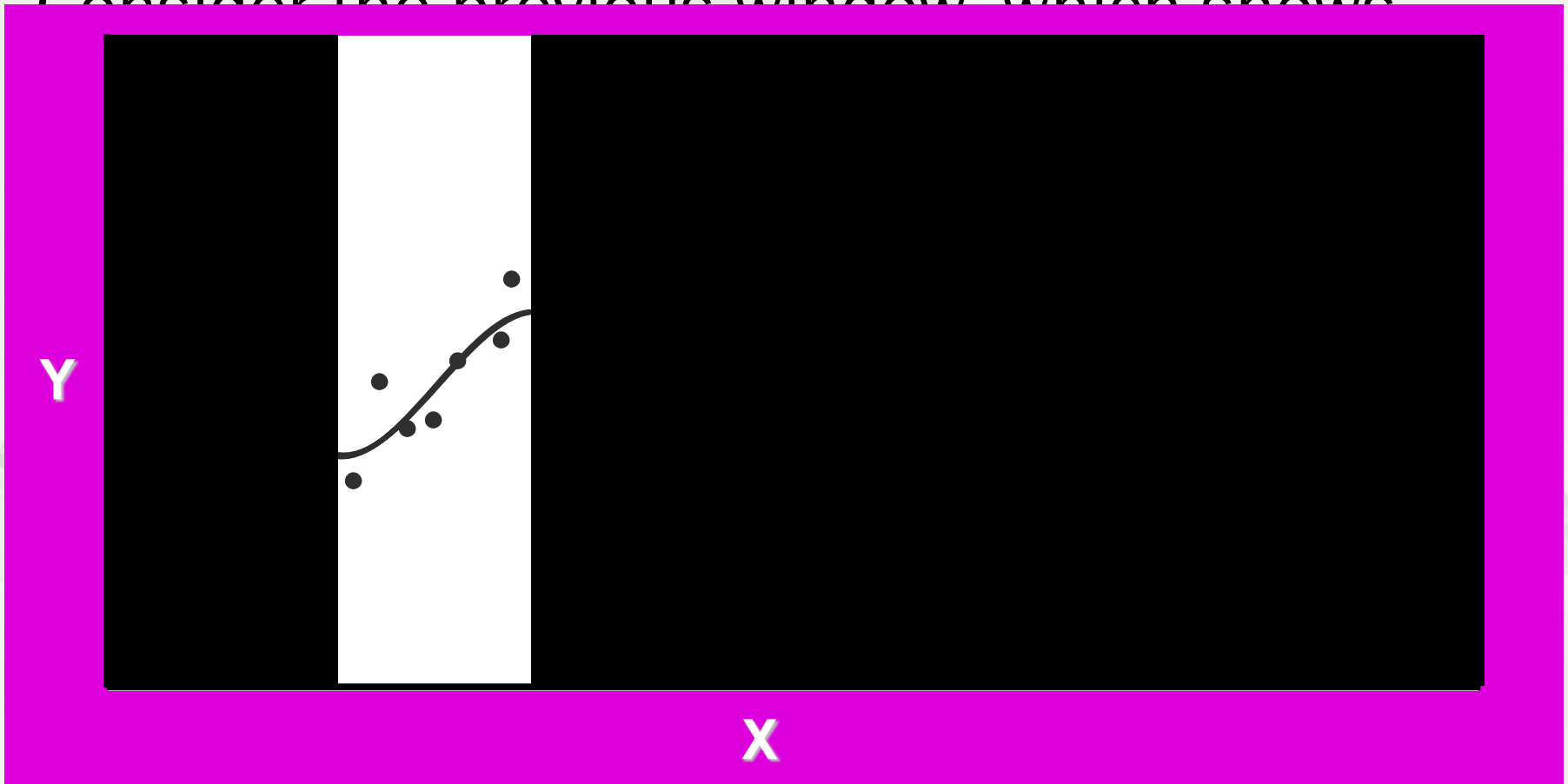
Y

X



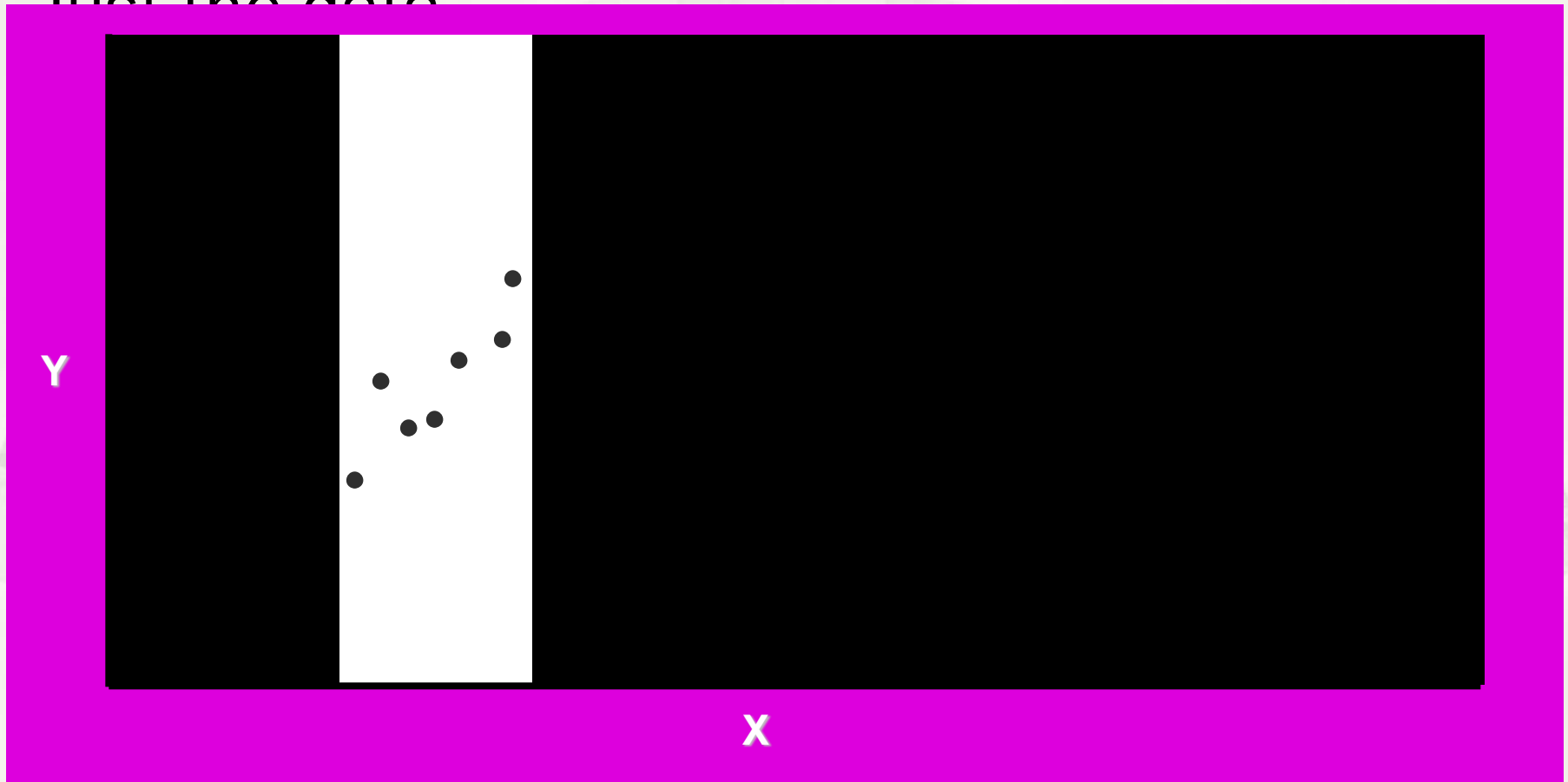
# Idea of Local Regression

Consider the previous window, which shows



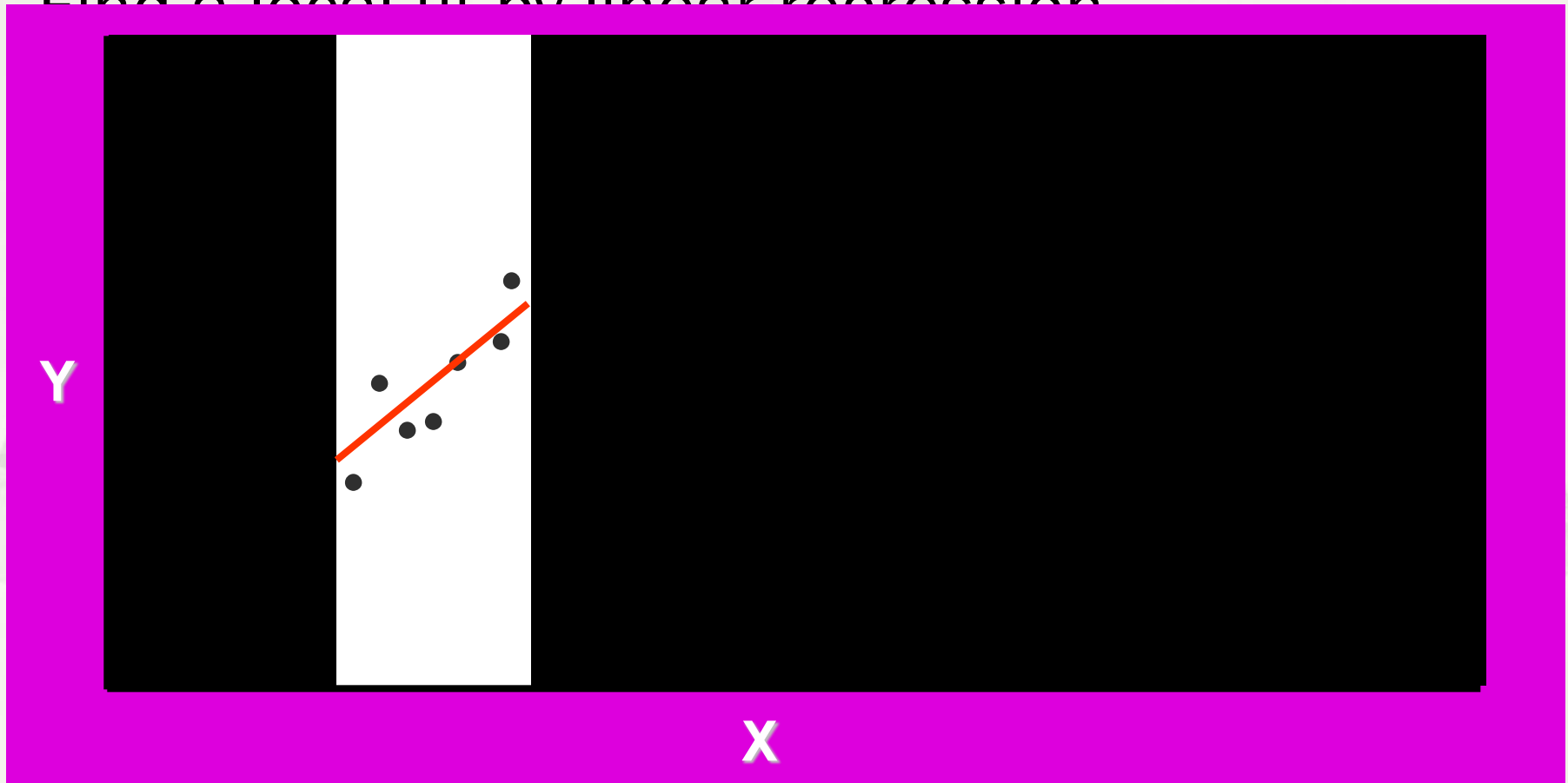
# Idea of Local Regression

Just the data



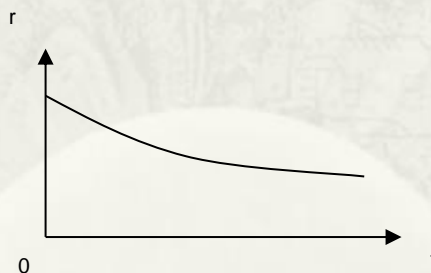
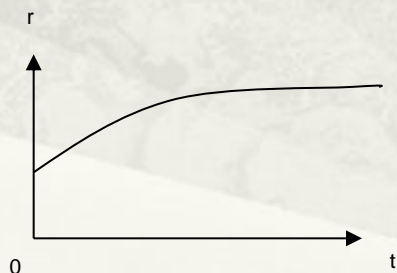
# Idea of Local Regression

Find a local fit by linear regression



# 利率期限结构理论

- \* 一般而言收益率曲线形状主要有三种：收益率曲线是在以期限长短为横坐标，以收益率为纵坐标的直角坐标系上显示出来。主要有三种类型：第一类是正收益曲线（或称上升收益曲线），其显示的期限结构特征是短期国债收益率较低，而长期国债收益率较高。第二类是反收益曲线（或称下降收益曲线），其显示的期限结构特征是短期国债收益率较高，而长期国债收益率较低。这两种收益率曲线转换过程中会出现第三种形态的收益曲线，称水平收益曲线，其特征是长短期国债收益率基本相等。





# 收益率曲线的拟合方法

## \* 1.样条法

### (1) 多项式样条法

多项式样条法是由麦克库隆茨（Mc Culloch）于1971年提出的, 它的主要思想是将贴现函数用分段的多项式函数来表示。在实际应用中, 多项式样条函数的阶数一般取为三, 从而保证贴现函数及其一阶和二阶导数都是连续的。于是我们用下式表示期限为t的贴现函数:

$$B(t) = \begin{cases} B_0(t) = d_0 + c_0 t + b_0 t^2 + a_0 t^3, t \in [0, n] \\ B_n(t) = d_1 + c_1 t + b_1 t^2 + a_1 t^3, t \in [n, m] \\ B_m(t) = d_2 + c_2 t + b_2 t^2 + a_2 t^3, t \in [m, 20] \end{cases}$$

# 收益率曲线的拟合方法

## \* (2) 指数样条法

- \* 指数样条法则是考虑到贴现函数基本上是一个随期限增加而指数下降的函数，它是瓦西塞克（Vasicek）和弗隆戈（Fong）在1982年提出的，该方法将贴现函数用分段的指数函数来表示。同样为了保证曲线的连续性和平滑性，通常采用三阶的指数样条函数，其形式如下：

$$B(t) = \begin{cases} B_0(t) = d_0 + c_0 e^{-ut} + b_0 e^{-2ut} + a_0 e^{-3ut}, & t \in [0, n] \\ B_n(t) = d_1 + c_1 e^{-ut} + b_1 e^{-2ut} + a_1 e^{-3ut}, & t \in [n, m] \\ B_m(t) = d_2 + c_2 e^{-ut} + b_2 e^{-2ut} + a_2 e^{-3ut}, & t \in [m, 20] \end{cases}$$

# 收益率曲线的拟合方法

## \* 2. 尼尔森-辛格尔 (Nelson-Siegel) 模型

- \* 尼尔森和辛格尔在1987年提出了一个用参数表示的瞬时 (即期限为零的) 远期利率函数。

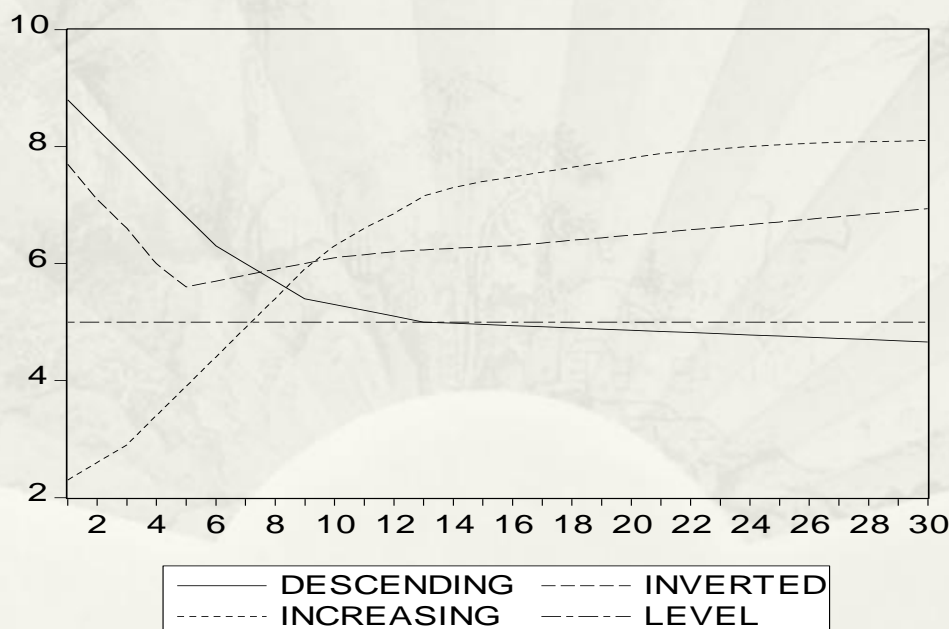
$$f(t) = \beta_0 + \beta_1 \exp\left(-\frac{t}{\tau_1}\right) + \beta_2 \left(\frac{t}{\tau_1}\right) \exp\left(-\frac{t}{\tau_1}\right)$$

- \* 由此我们可以求得即期利率的函数形式：

$$R(t) = \frac{\int_0^t f(s) ds}{t} = \beta_0 + \beta_1 \left[ \frac{1 - \exp\left(-\frac{t}{\tau_1}\right)}{\frac{t}{\tau_1}} \right] + \beta_2 \left[ \frac{1 - \exp\left(-\frac{t}{\tau_1}\right)}{\frac{t}{\tau_1}} - \exp\left(-\frac{t}{\tau_1}\right) \right]$$

# 收益率曲线的拟合方法

- \* 这个模型中只有四个参数, 即  $\beta_0, \beta_1, \beta_2, \tau_1$ , 根据式中的即期利率, 我们可以得到相应的贴现函数, 从而计算债券的模型价值用以拟合市场数据。虽然参数的个数不多, 但这样的函数形式已经有足够的灵活度来拟合收益率曲线的标准形状, 递增的、递减的、水平和倒置的形状, 如图所示。



# 收益率曲线的拟合方法

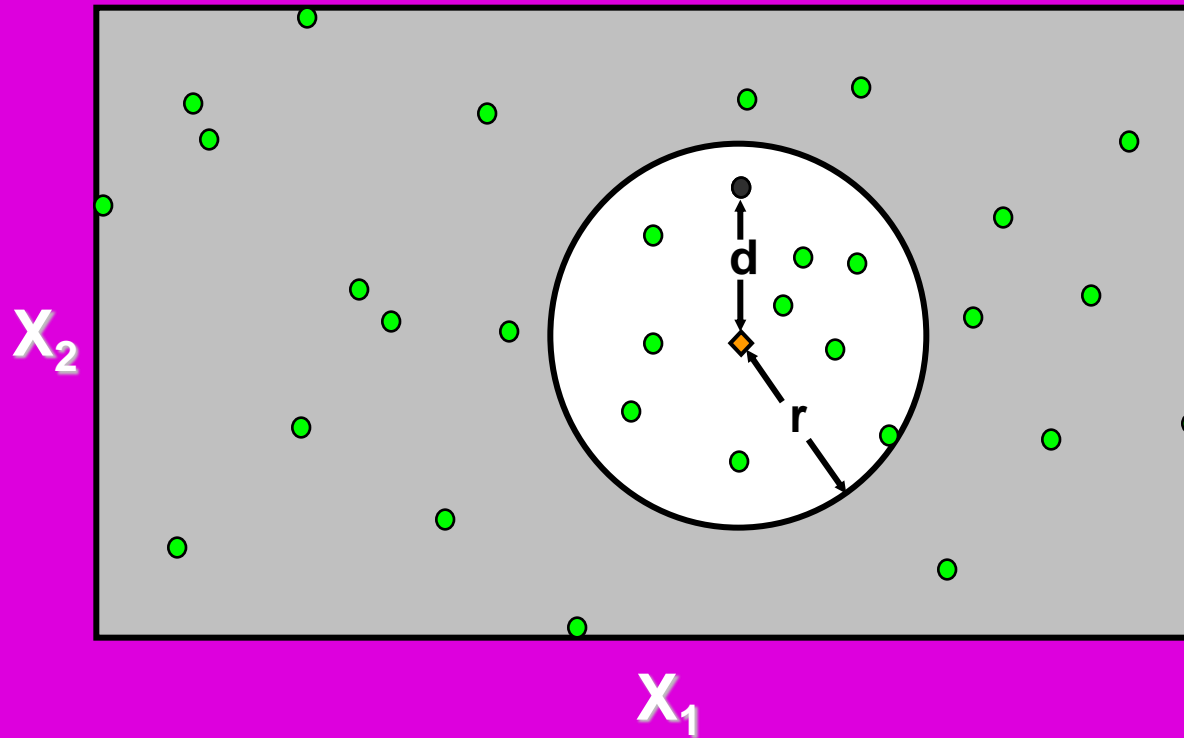
## \* 3. 斯文森（Svensson）模型

- \* 斯文森将Nelson-Siegel 模型作了推广, 引进了另外两个参数  $\beta_3, \tau_2$ , 而得到如下的即期利率函数:

$$R(t) = \beta_0 + \beta_1 \left[ \frac{1 - \exp\left(-\frac{t}{\tau_1}\right)}{\frac{t}{\tau_1}} \right] + \beta_2 \left[ \frac{1 - \exp\left(-\frac{t}{\tau_1}\right)}{\frac{t}{\tau_1}} - \exp\left(-\frac{t}{\tau_1}\right) \right] \\ + \beta_3 \left[ \frac{1 - \exp\left(-\frac{t}{\tau_2}\right)}{\frac{t}{\tau_2}} - \exp\left(-\frac{t}{\tau_2}\right) \right]$$

- \* 这个模型也被称为扩展的Nelson-Siegel 模型, 这一模型在计算短期债券价格时的灵活性大大增强。

# The Loess Method with Two Predictors

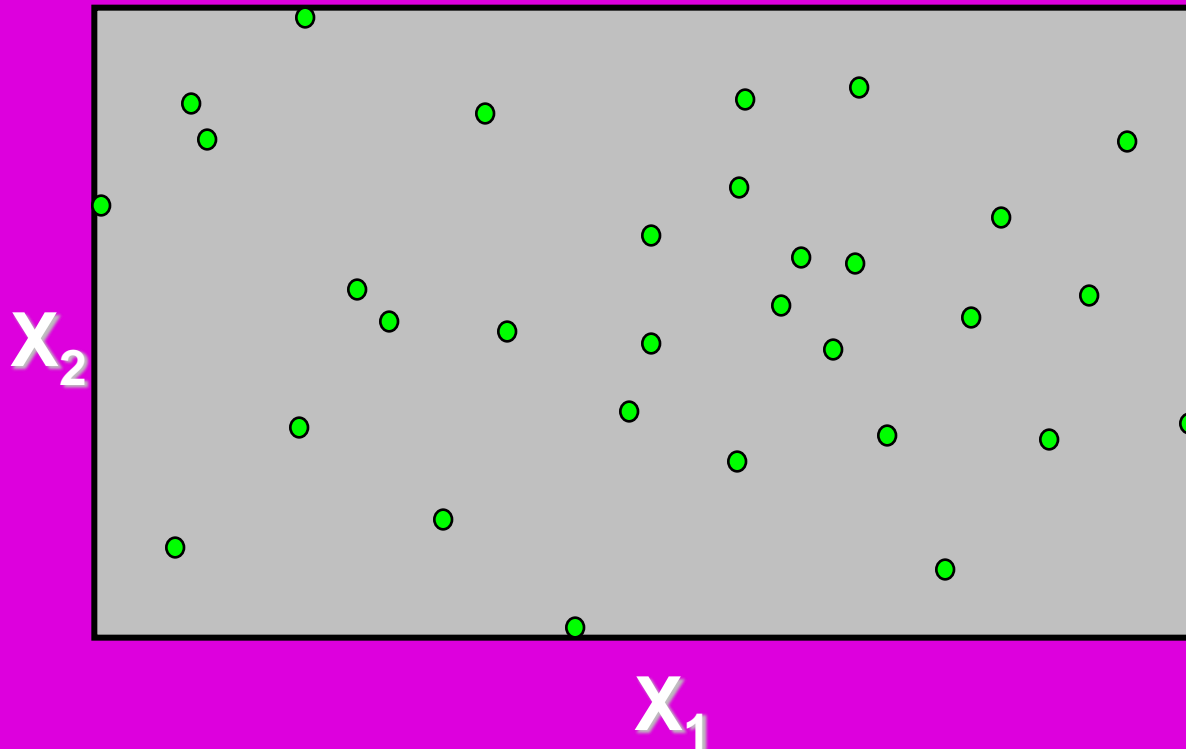


30 data points ●

Smoothing  
Parameter =  $1/3$

10 points per  
neighborhood

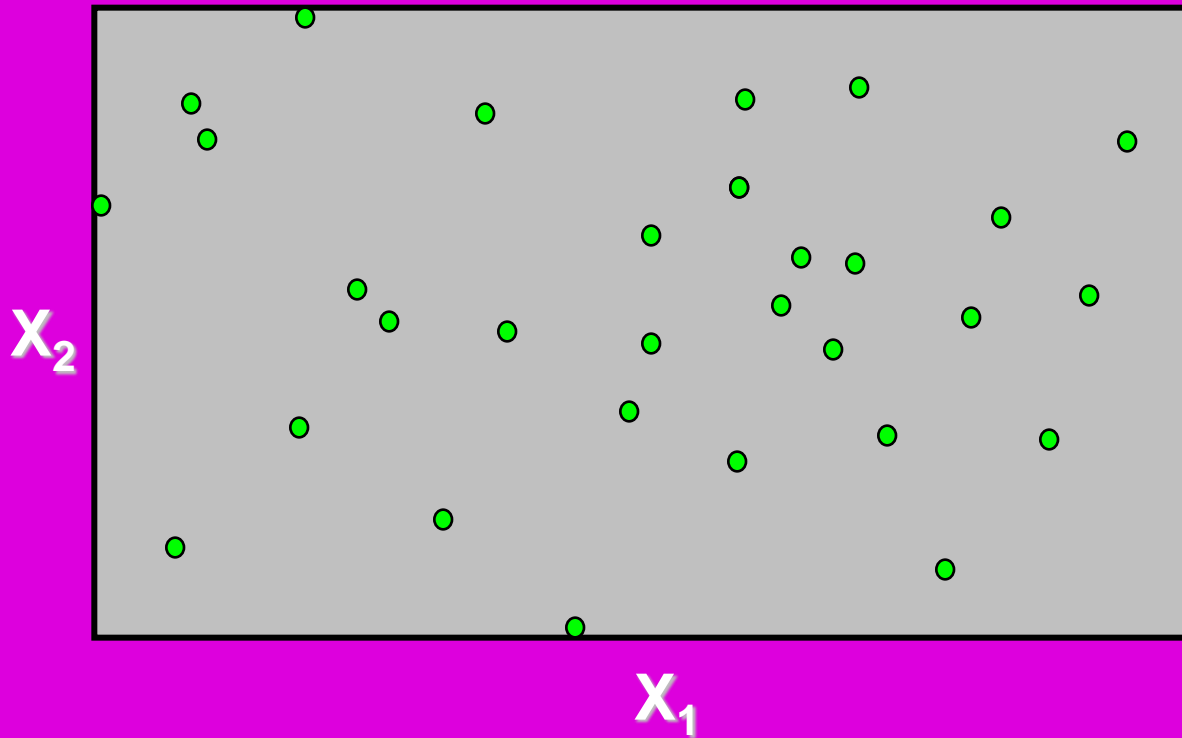
# The Loess Method with Two Predictors



30 data points ●

# The Loess Method with Two Predictors

Step 0: Choose a smoothing parameter. This determines

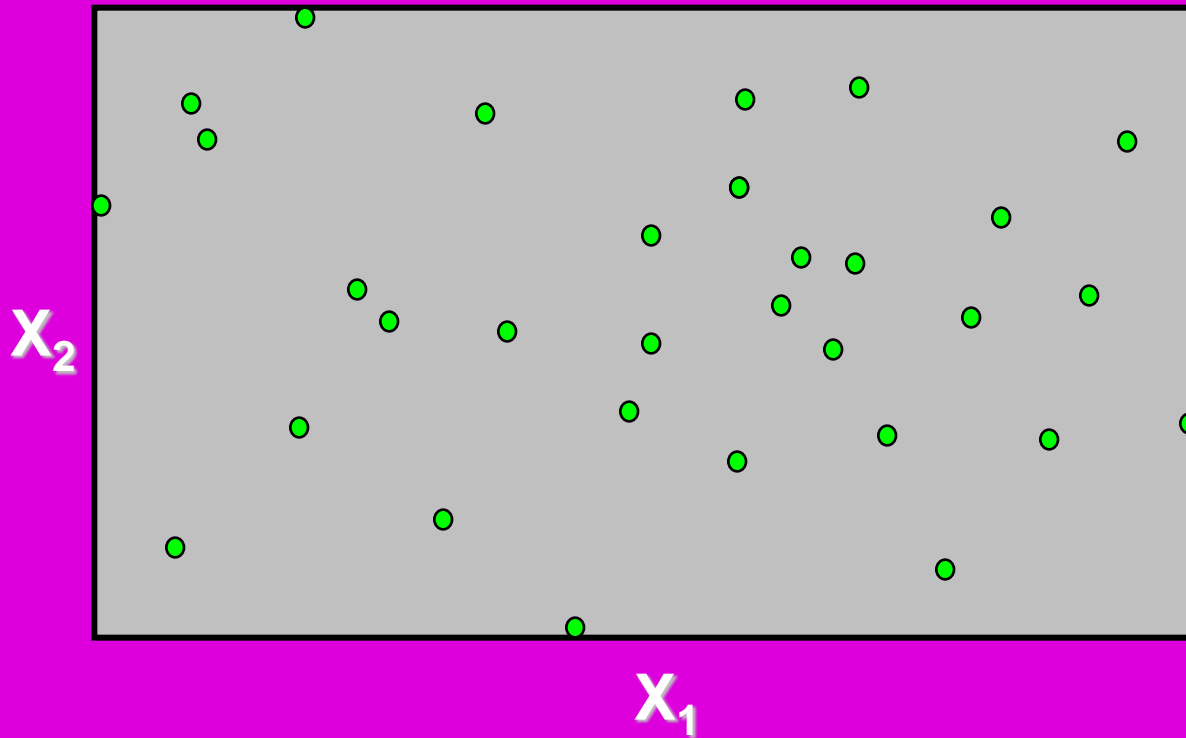


30 data points ●



# The Loess Method with Two Predictors

Step 0: Choose a smoothing parameter. This determines

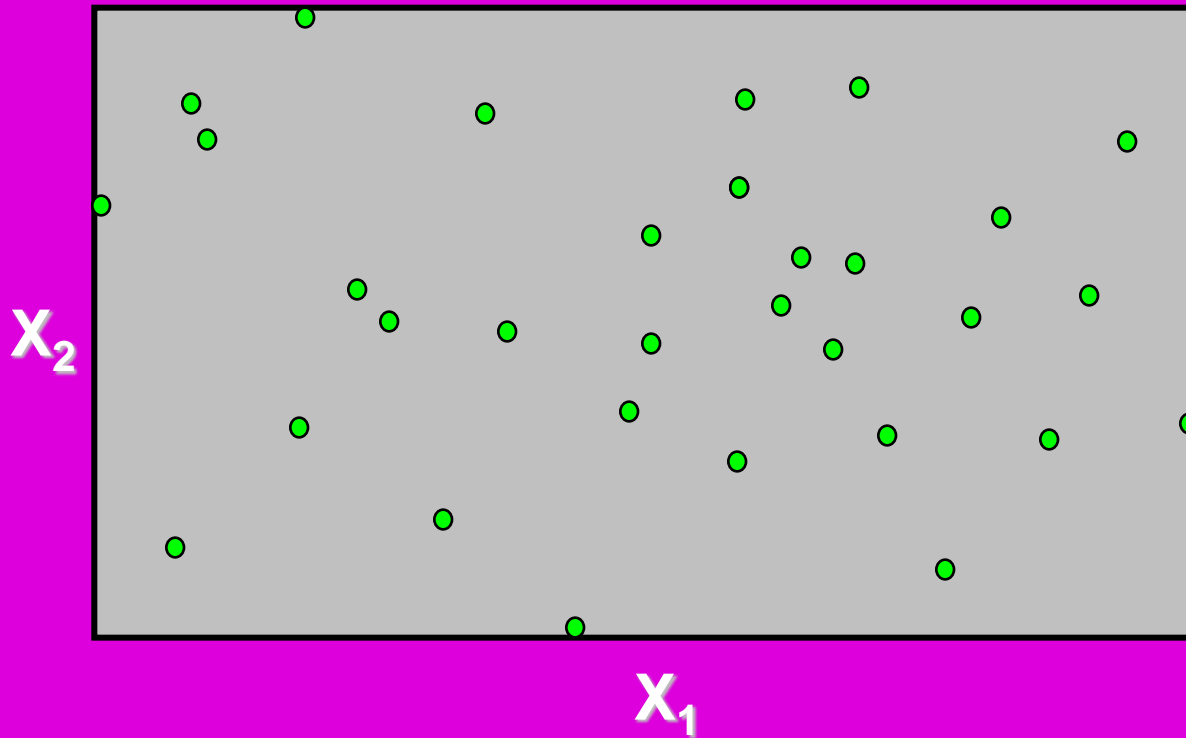


30 data points ●

Smoothing  
Parameter =  $1/3$

# The Loess Method with Two Predictors

Step 0: Choose a smoothing parameter. This determines



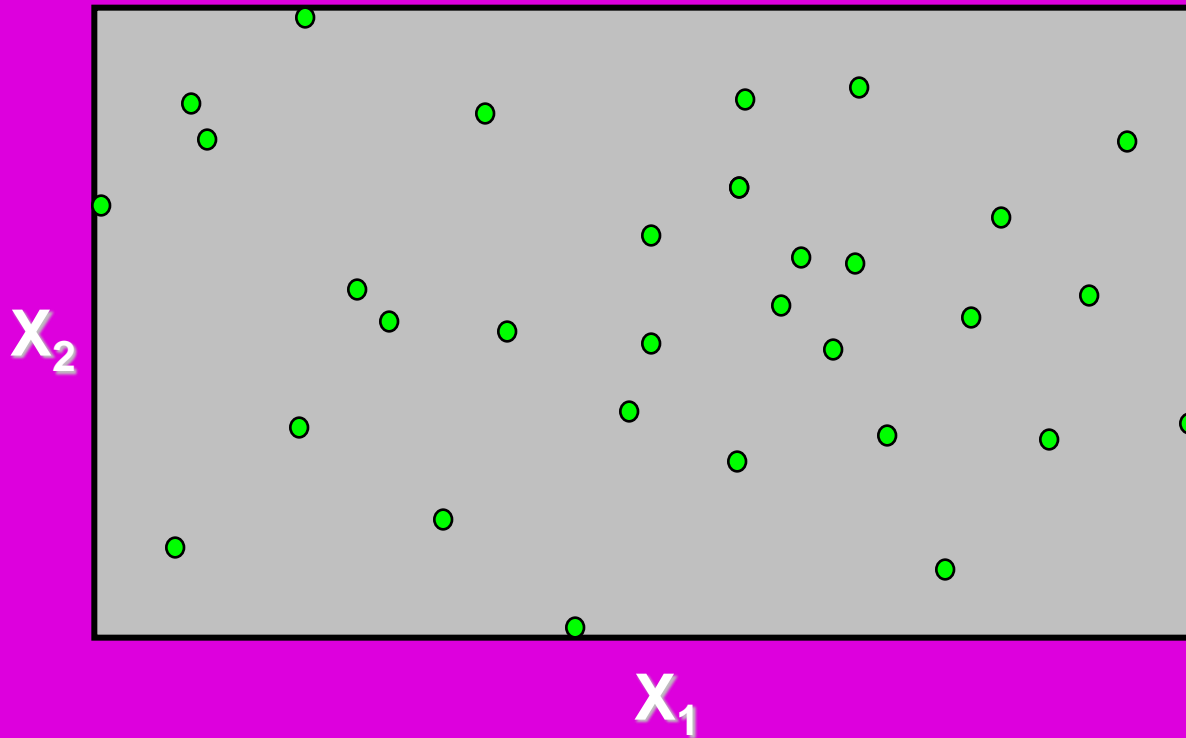
30 data points ●

Smoothing  
Parameter =  $1/3$

10 points per  
neighborhood

# The Loess Method with Two Predictors

Step 1: Choose a point at which you want a loess fit



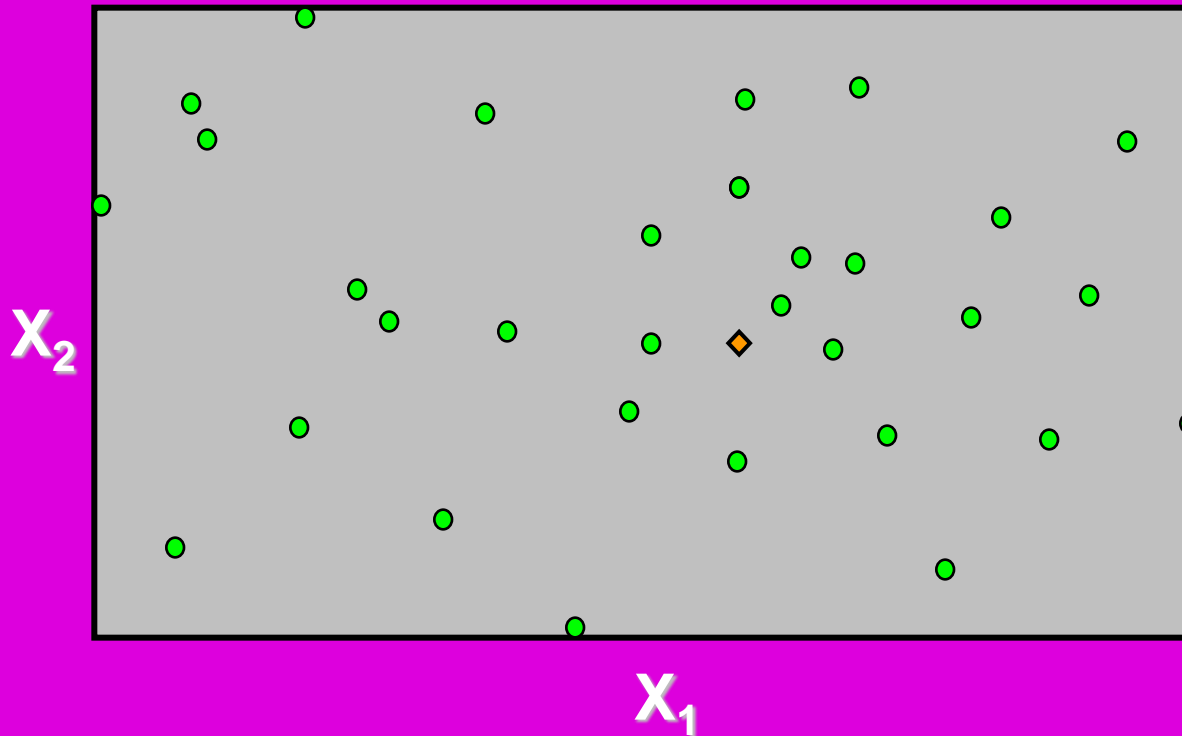
30 data points ●

Smoothing  
Parameter =  $1/3$

10 points per  
neighborhood

# The Loess Method with Two Predictors

- \* Step 1: Choose a point at which you want a loess fit



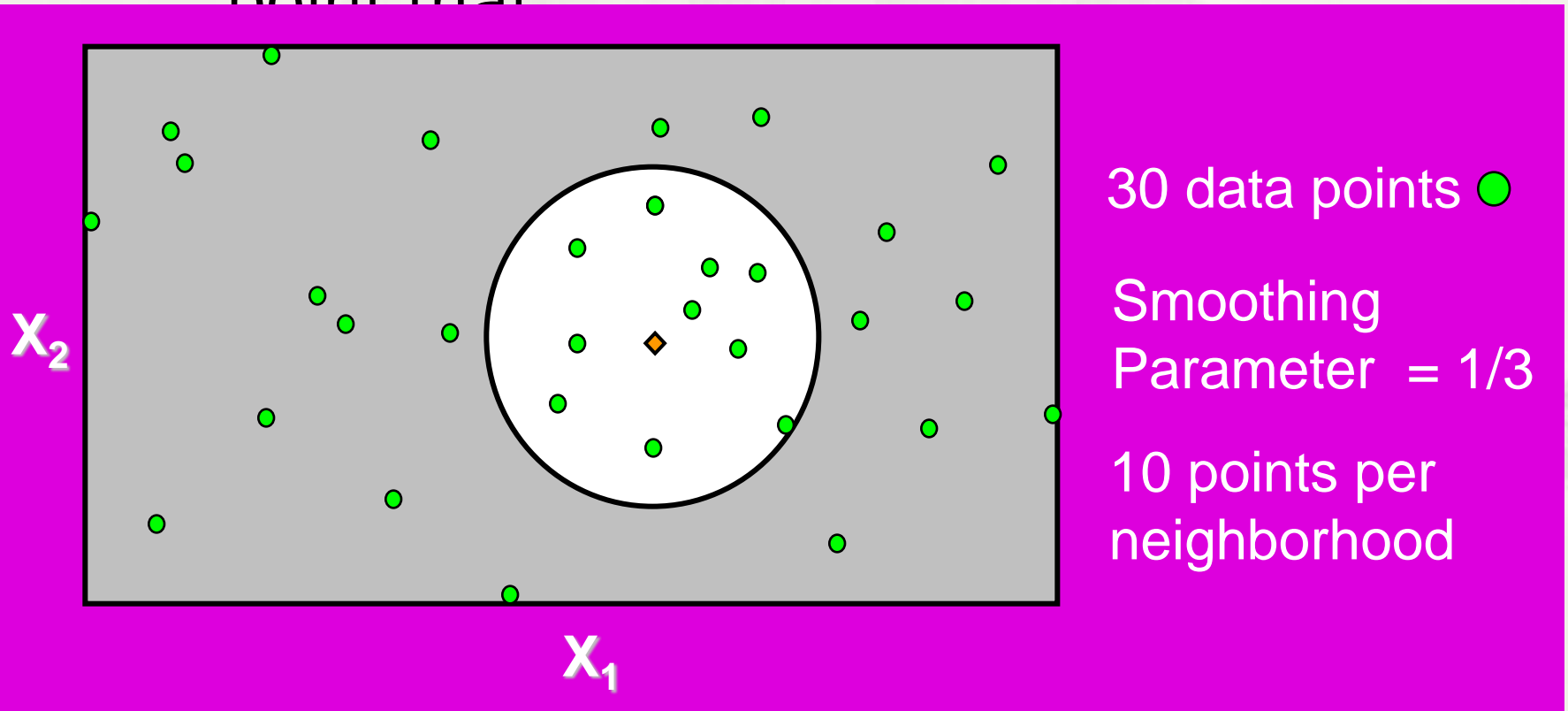
30 data points ●

Smoothing  
Parameter =  $1/3$

10 points per  
neighborhood

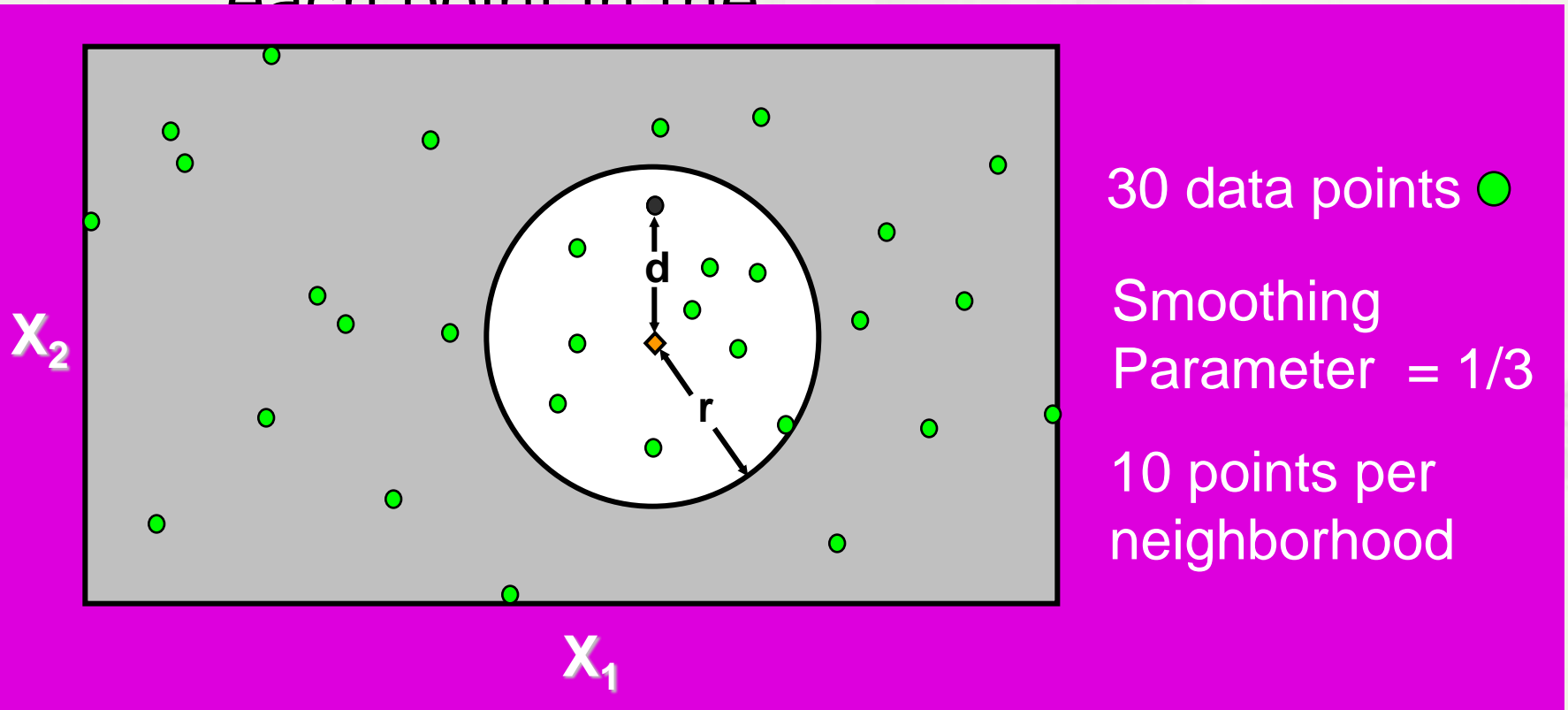
# The Loess Method with Two Predictors

Step 2: Find the smallest circle about the fit point that



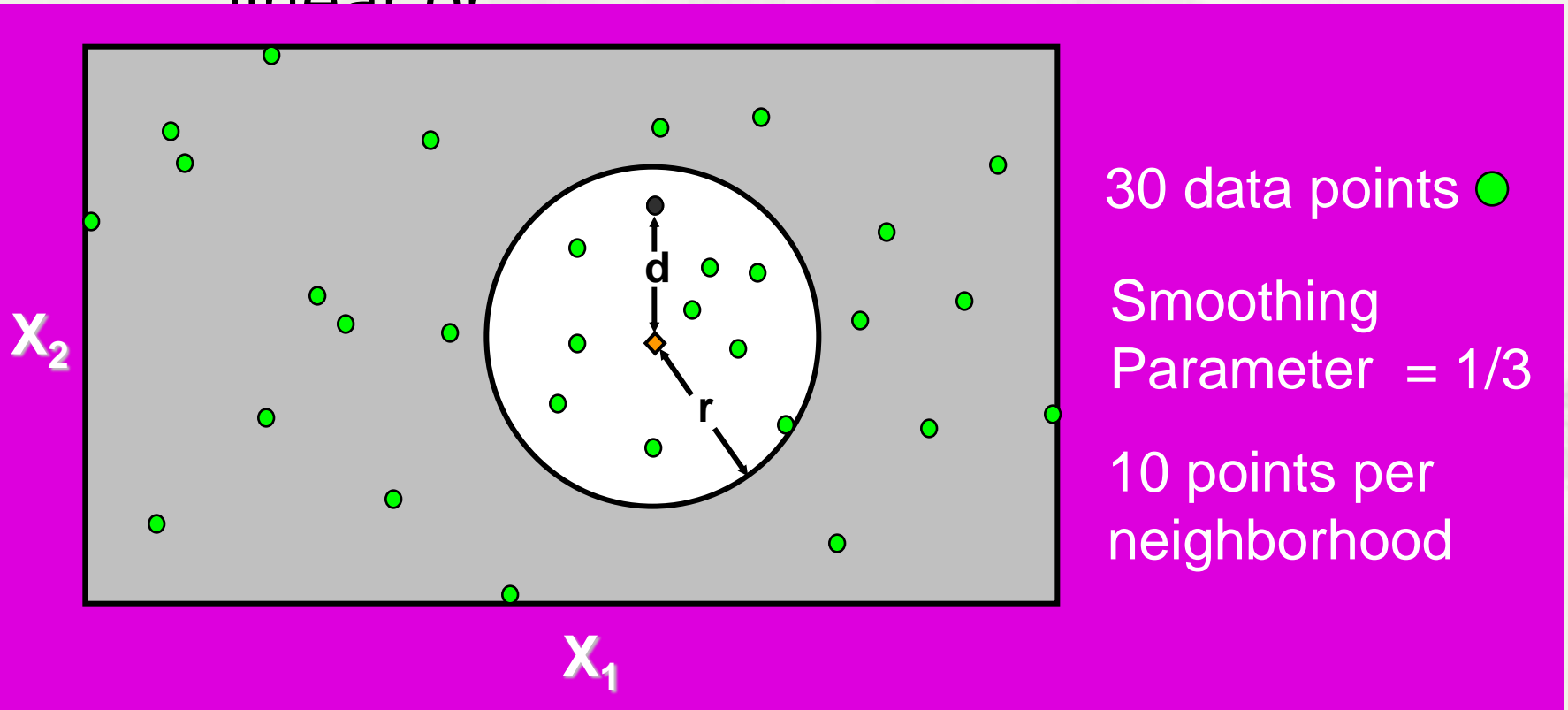
# The Loess Method with Two Predictors

- \* Step 3: Determine a local weight for each point in the neighborhood  $\left( \frac{d^3}{r^3} \right)^3$



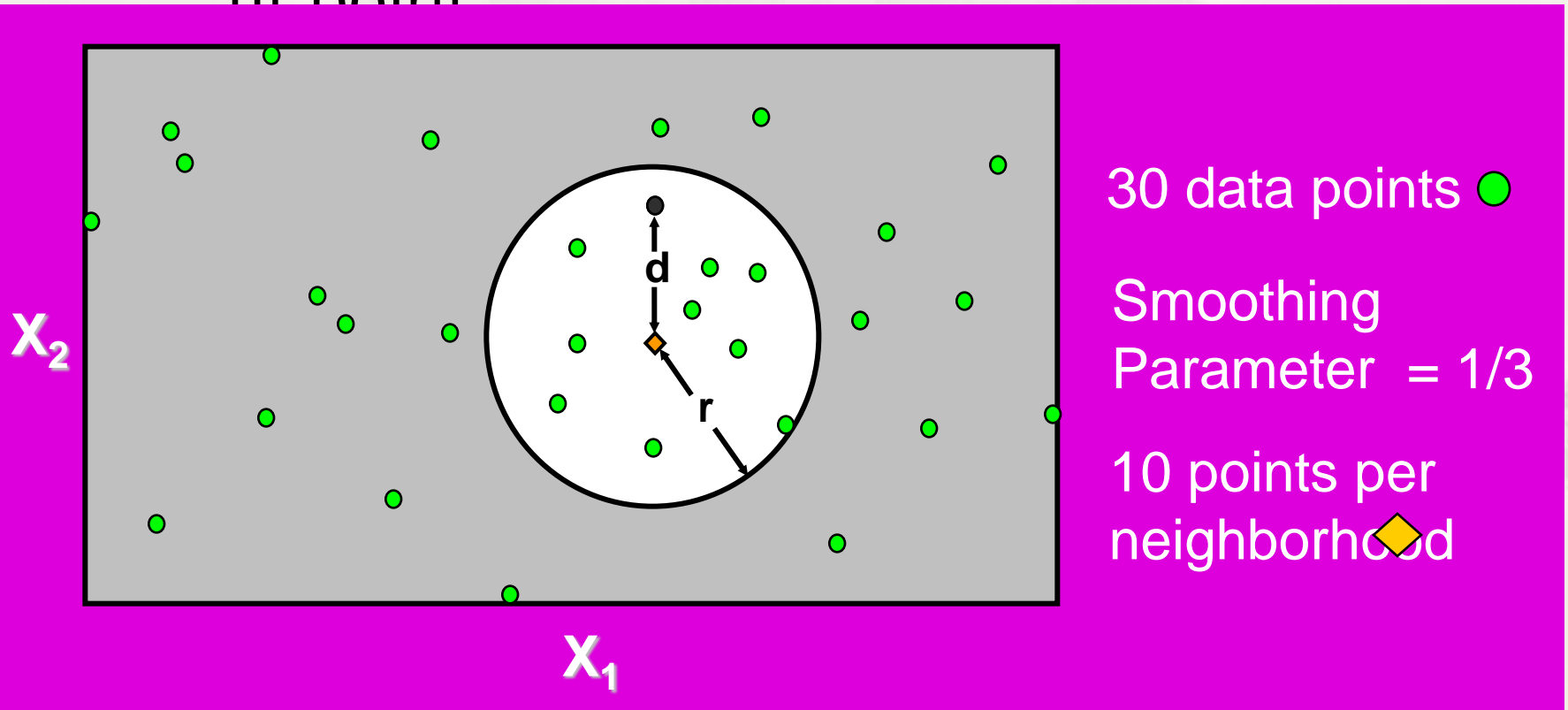
# The Loess Method with Two Predictors

Step 4: Locally fit a surface using weighted linear or



# The Loess Method with Two Predictors

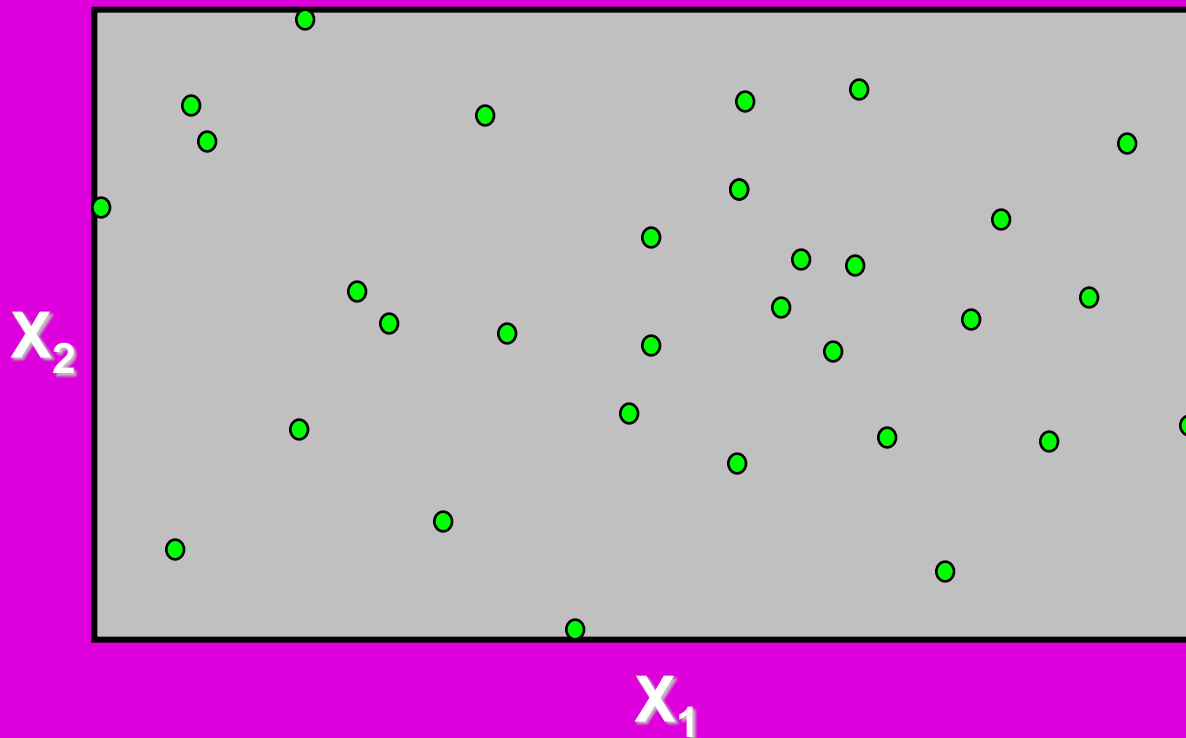
Step 5: Evaluate this regression surface at the fit point





# The Loess Method with Two Predictors

Step 6: Repeat this procedure for every point at which you



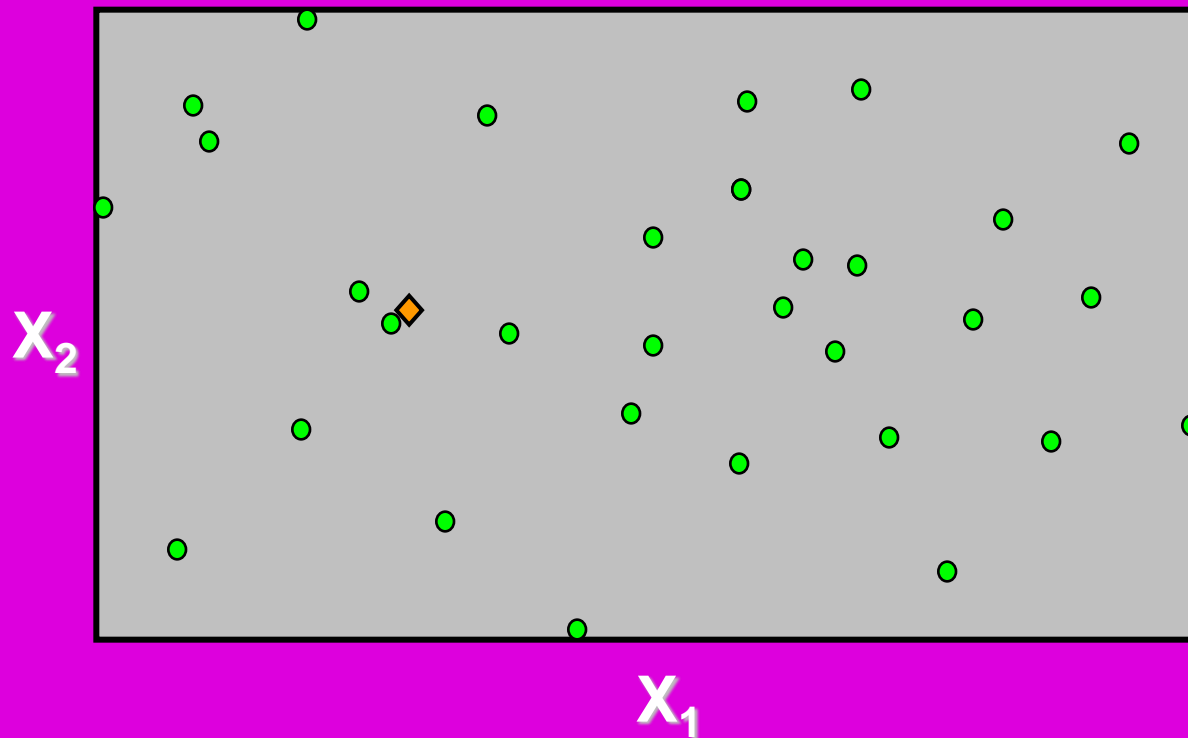
30 data points ●

Smoothing  
Parameter =  $1/3$

10 points per  
neighborhood

# The Loess Method with Two Predictors

Step 6: Repeat this procedure for every point at which you



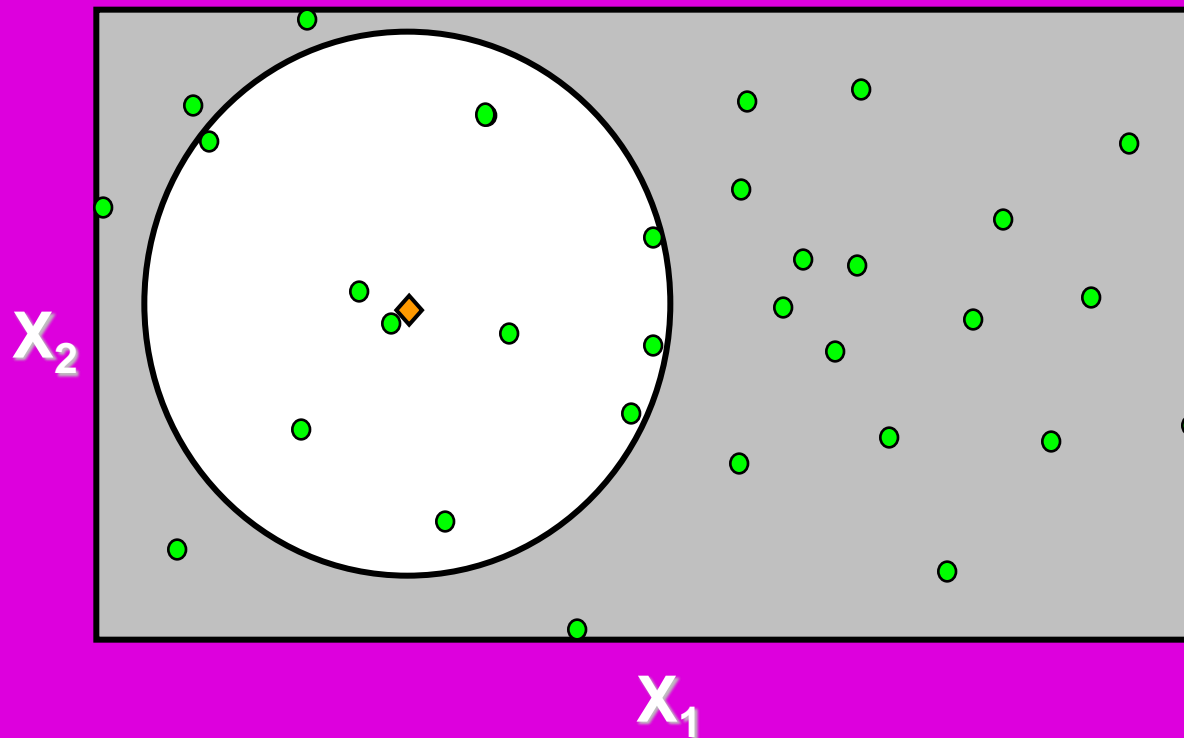
30 data points ●

Smoothing  
Parameter =  $1/3$

10 points per  
neighborhood

# The Loess Method with Two Predictors

Step 6: Repeat this procedure for every point at which you



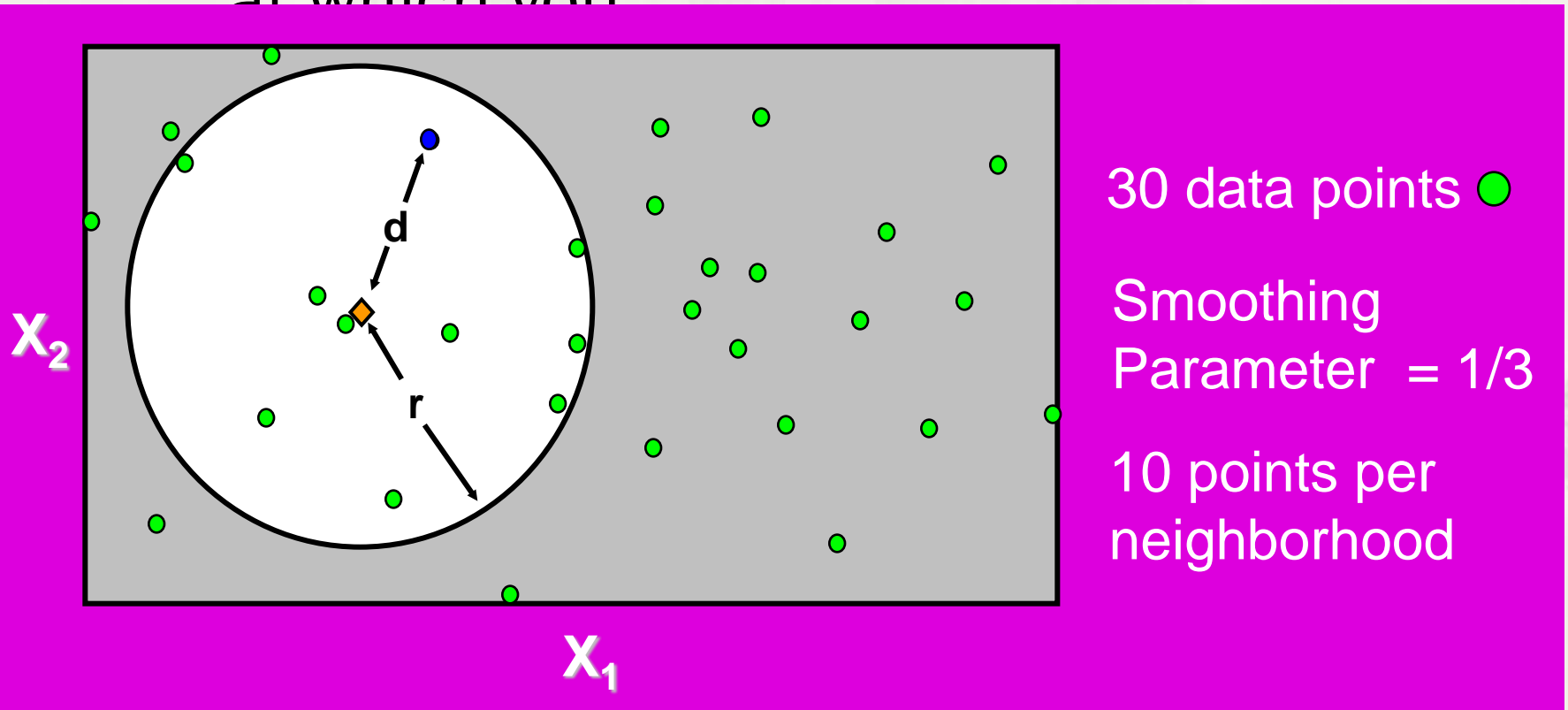
30 data points ●

Smoothing  
Parameter =  $1/3$

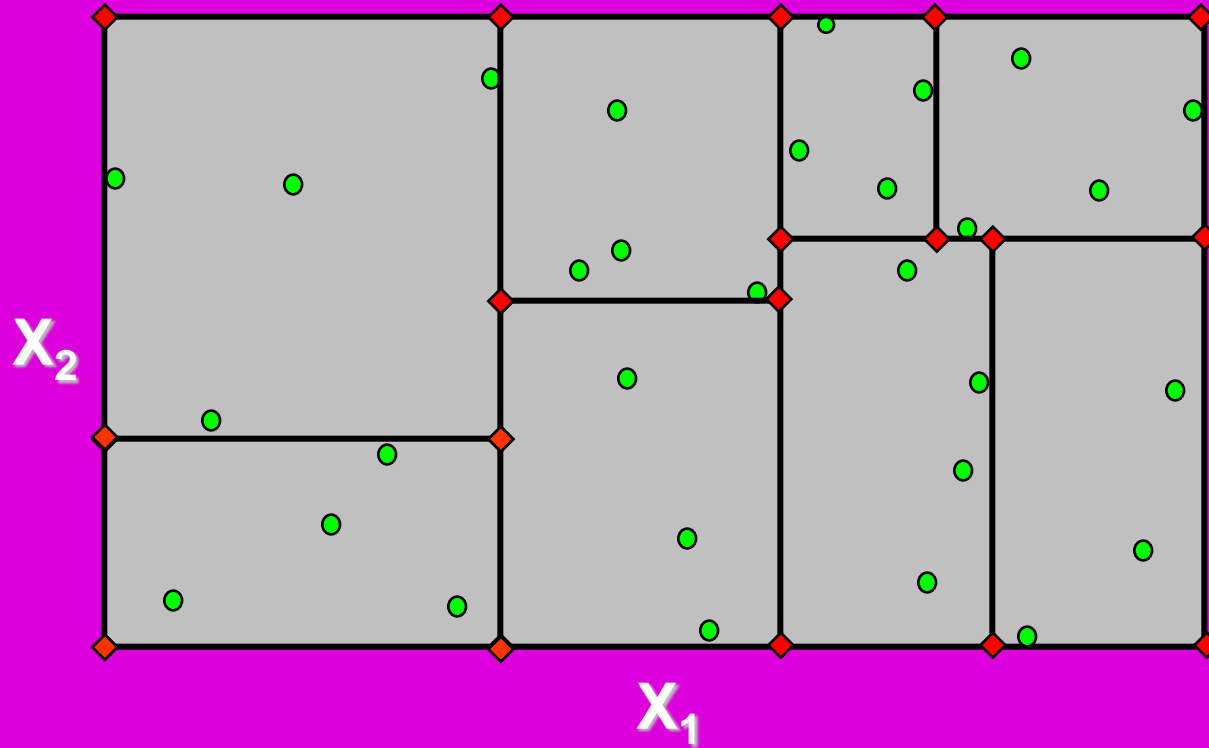
10 points per  
neighborhood

# The Loess Method with Two Predictors

Step 6: Repeat this procedure for every point at which you



# kd Tree Construction

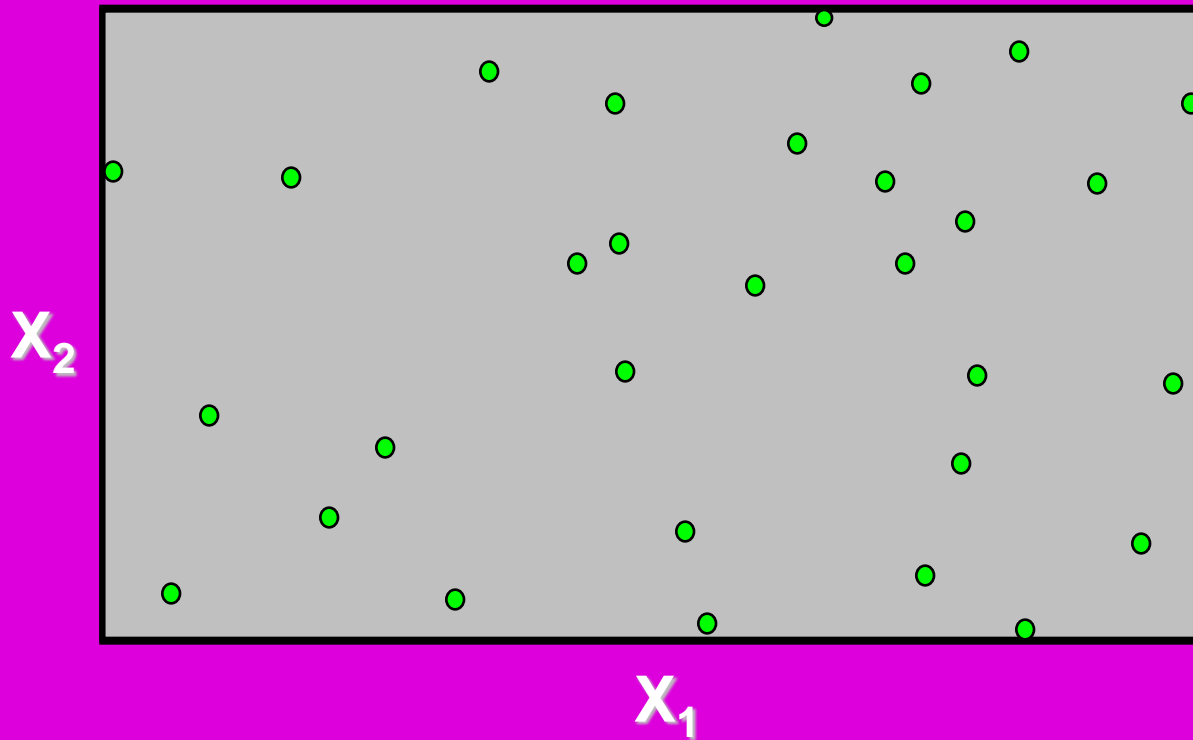


30 data points ●

Bucket size = 4

18 fit points ◆

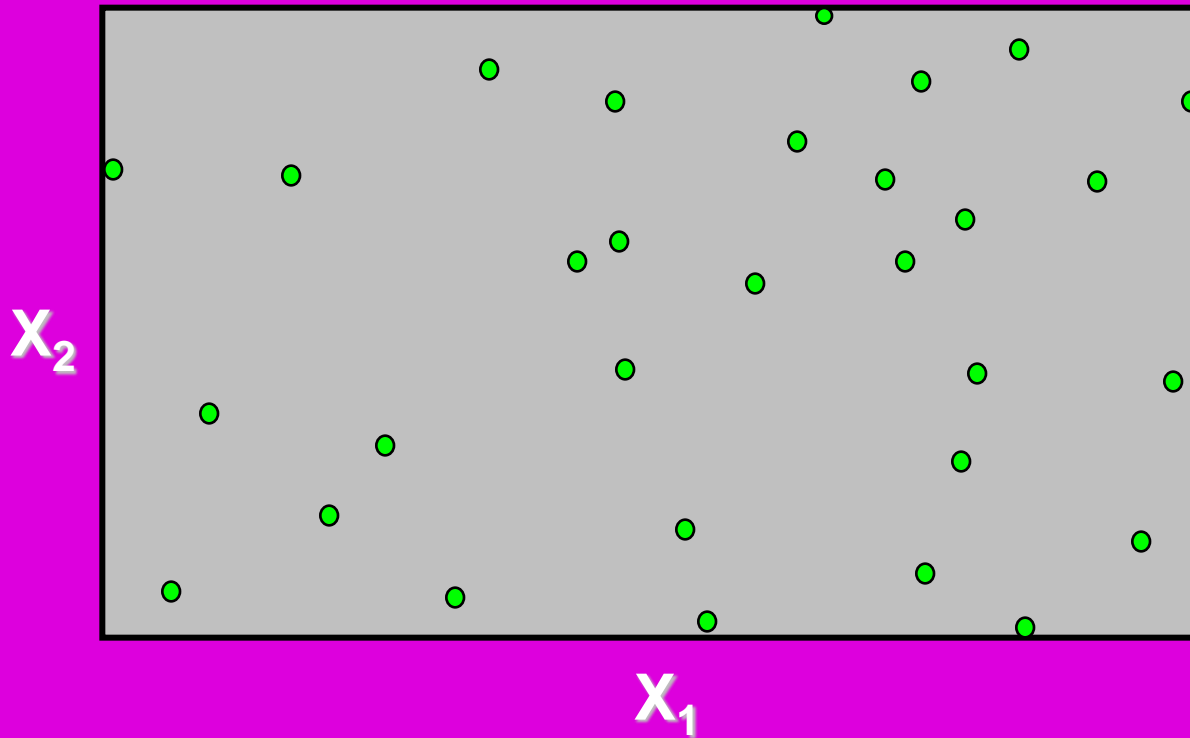
# kd Tree Construction



30 data points ●

# kd Tree Construction

Step 0: Select the bucket size. The default is  
(number of

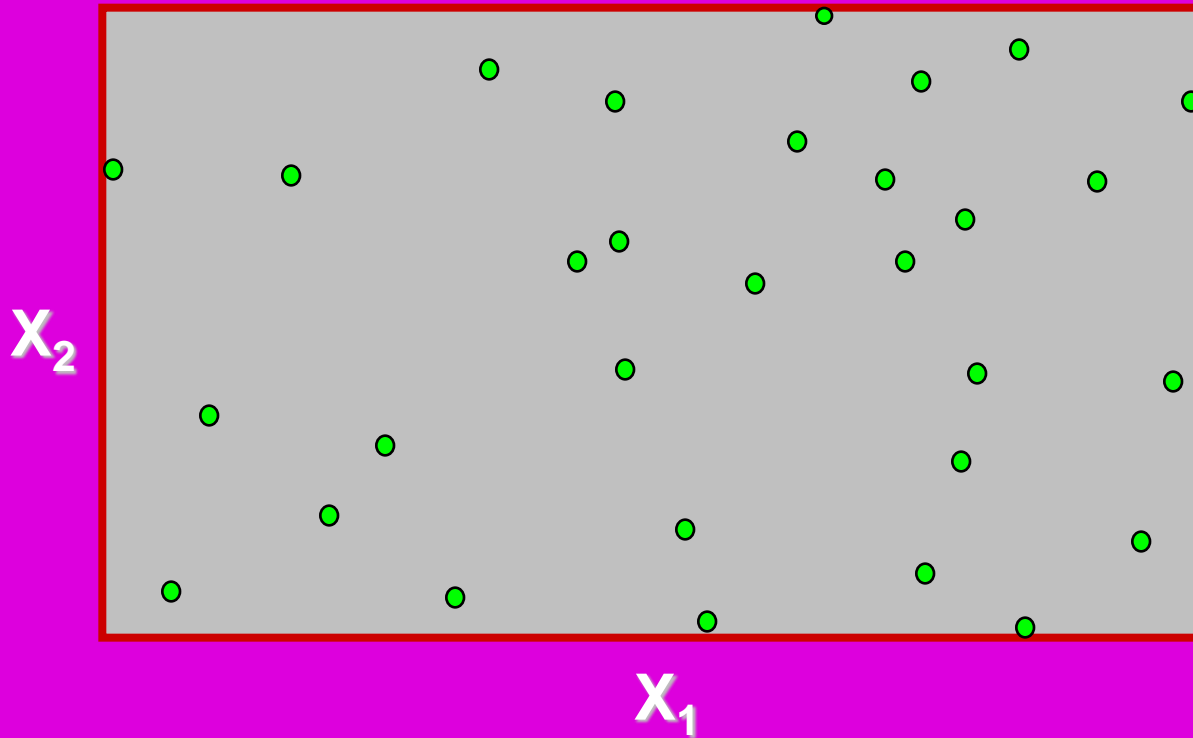


30 data points ●

Bucket size = 4

# kd Tree Construction

- \* Step 1: Start with the smallest cell (rectangle) enclosing the predictor data



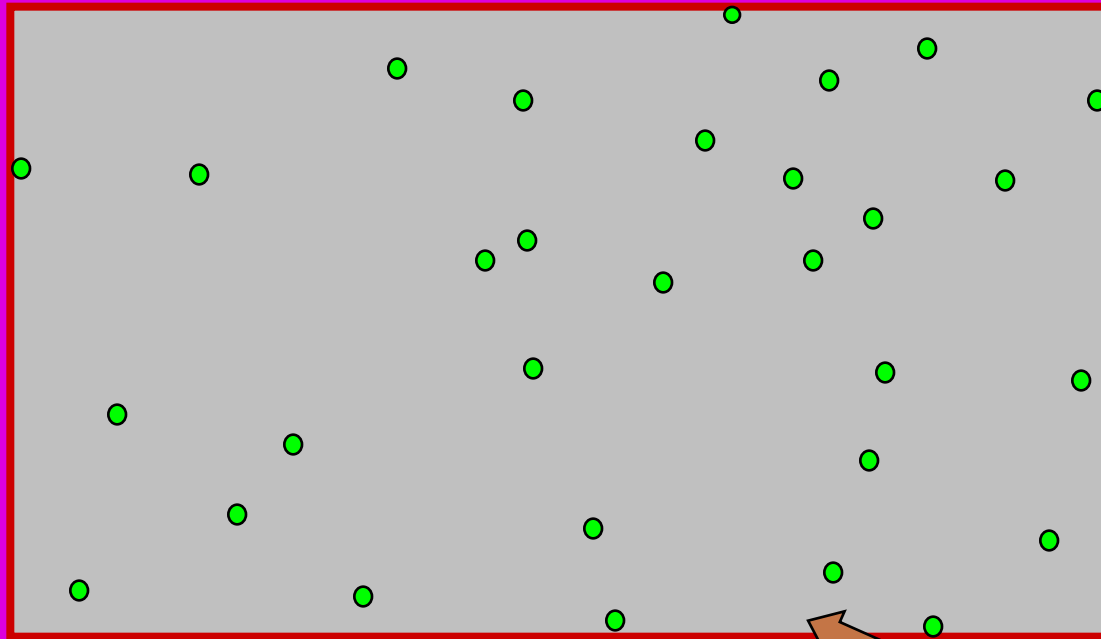
30 data points ●

Bucket size = 4



# kd Tree Construction

Step 2: Select the direction of the longest cell edge as the



$x_1$

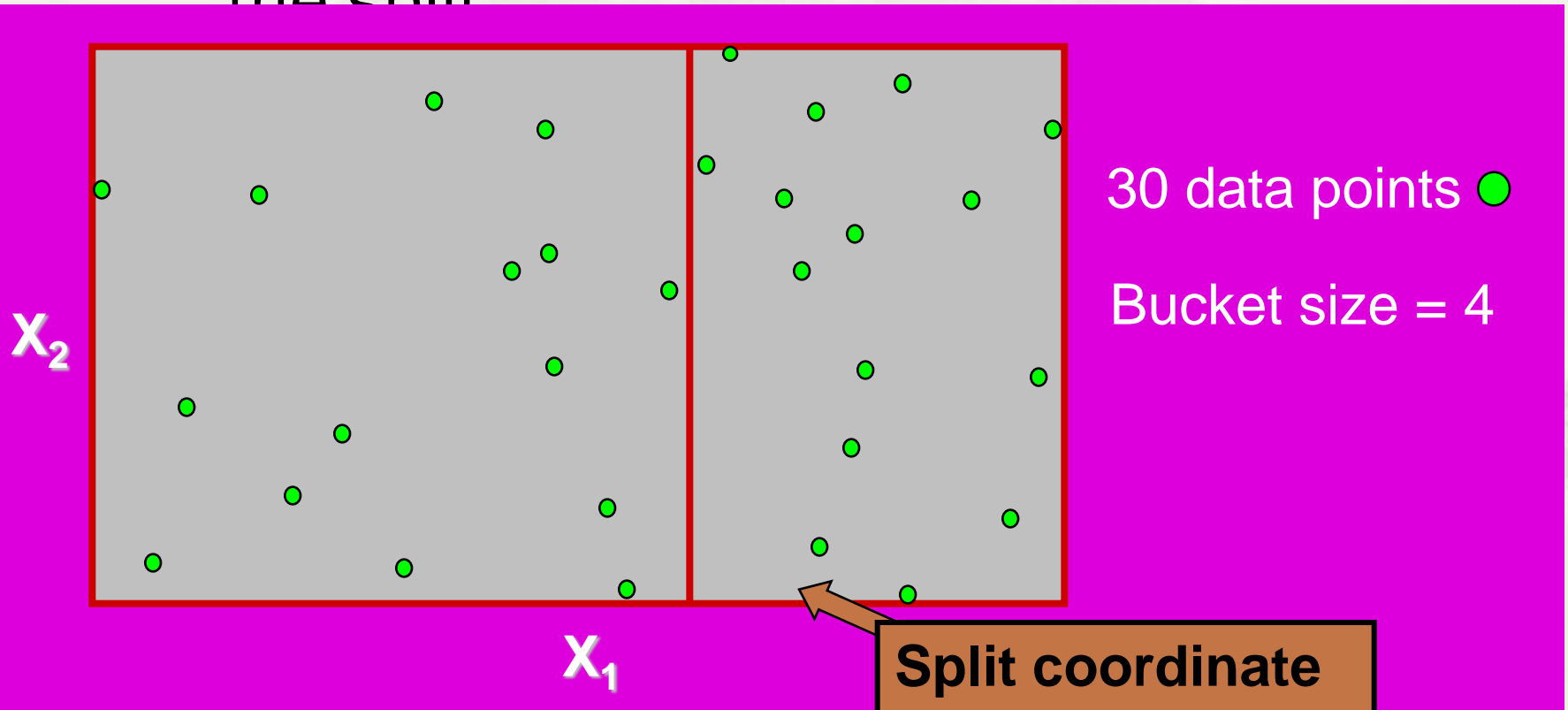
Split coordinate

30 data points ●

Bucket size = 4

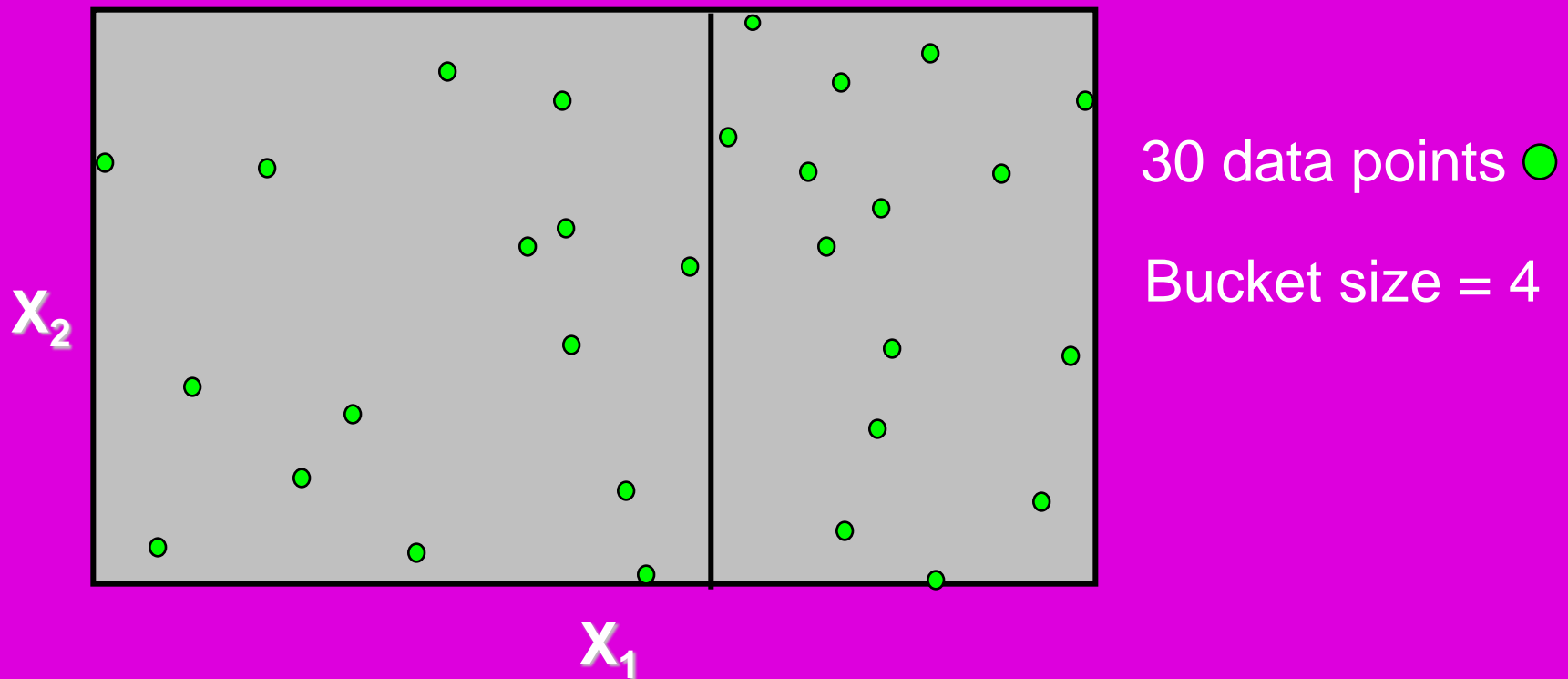
# kd Tree Construction

Step 3: Divide the cell in two at the median of the split



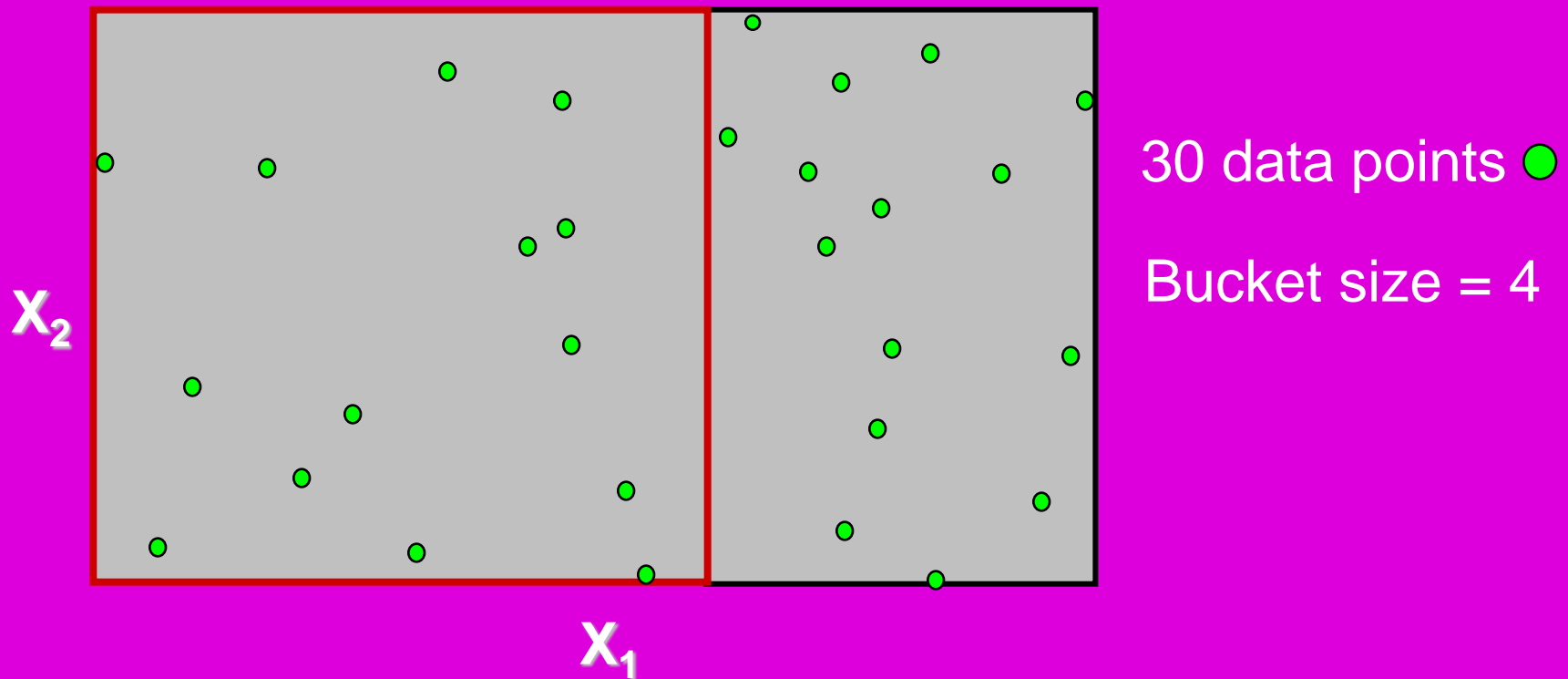
# kd Tree Construction

Step 4: Repeat process for each child cell until all cells have at most 4 points



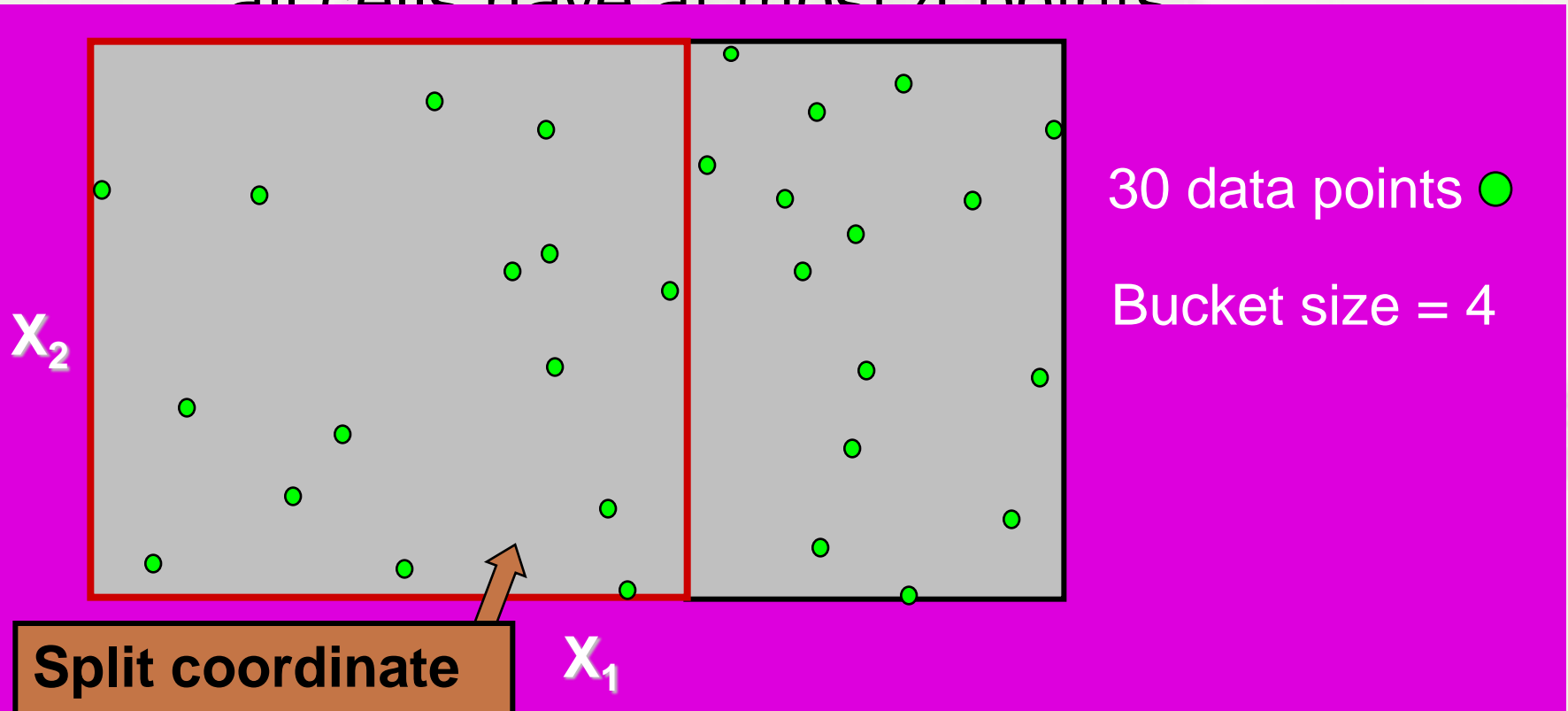
# kd Tree Construction

Step 4: Repeat process for each child cell until all cells have at most 4 points



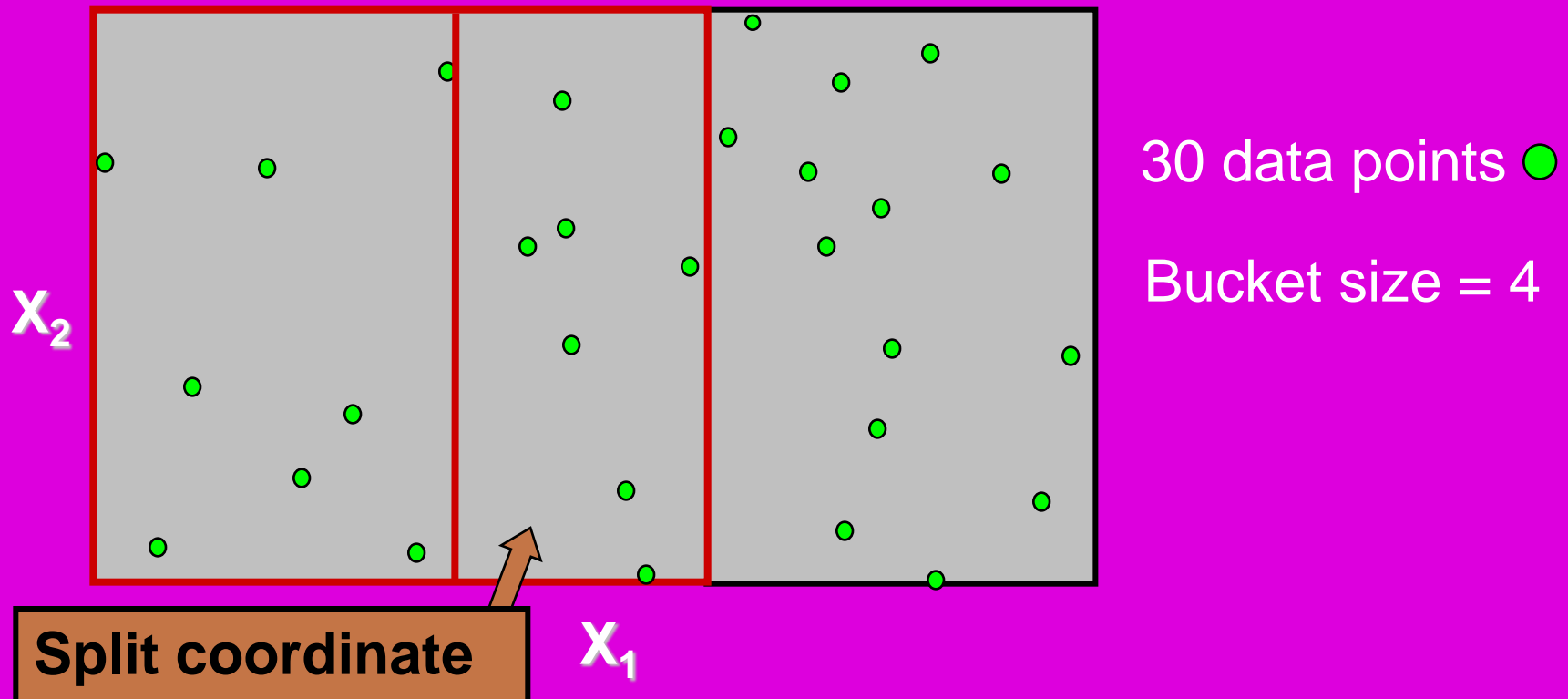
# kd Tree Construction

Step 4: Repeat process for each child cell until all cells have at most 4 points



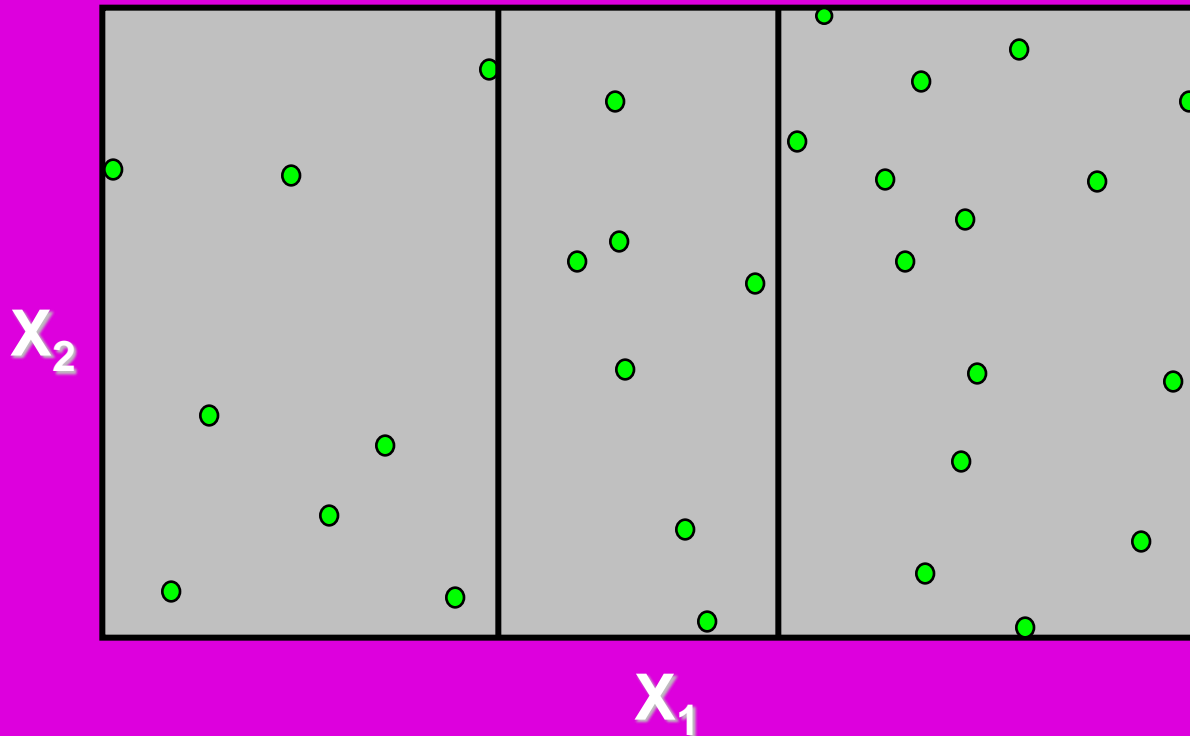
# kd Tree Construction

- \* Step 4: Repeat process for each child cell until all cells have at most 4 points



# kd Tree Construction

Step 4: Repeat process for each child cell until all cells have at most 4 points

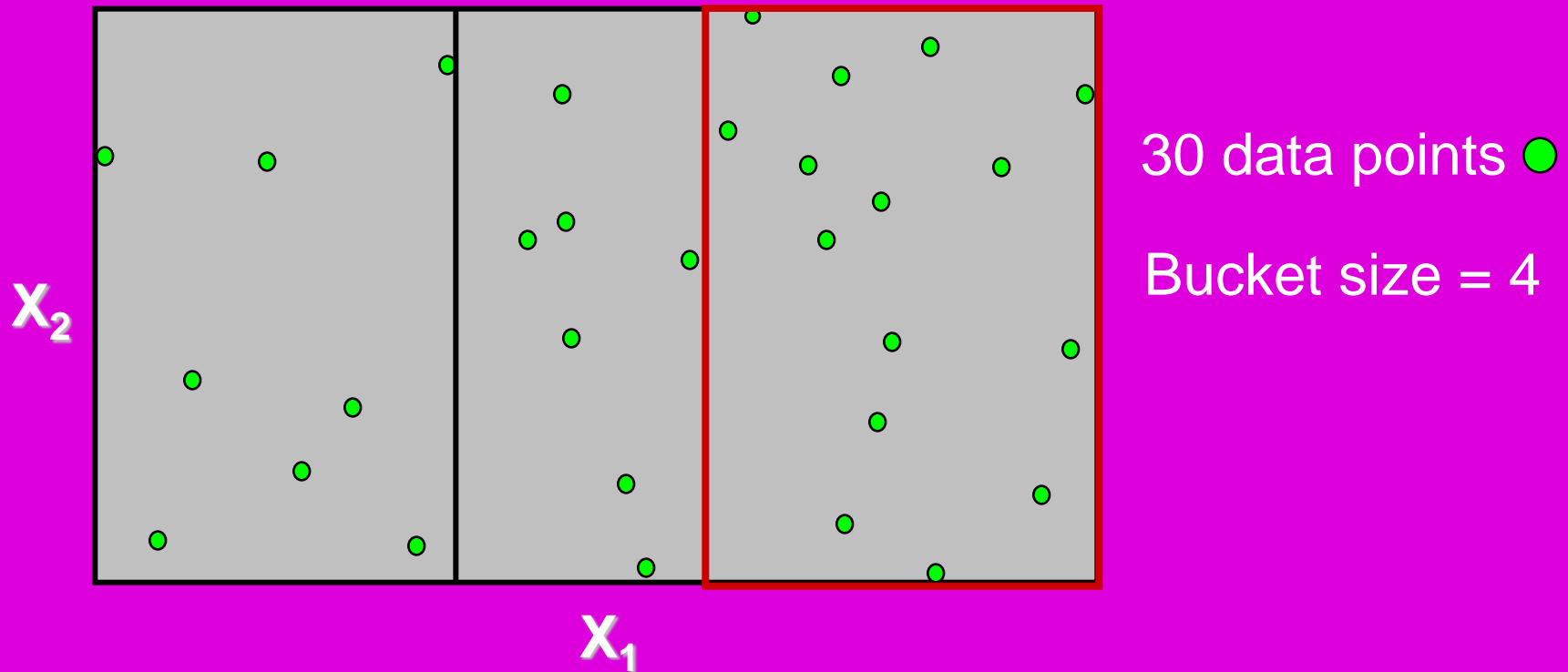


30 data points ●

Bucket size = 4

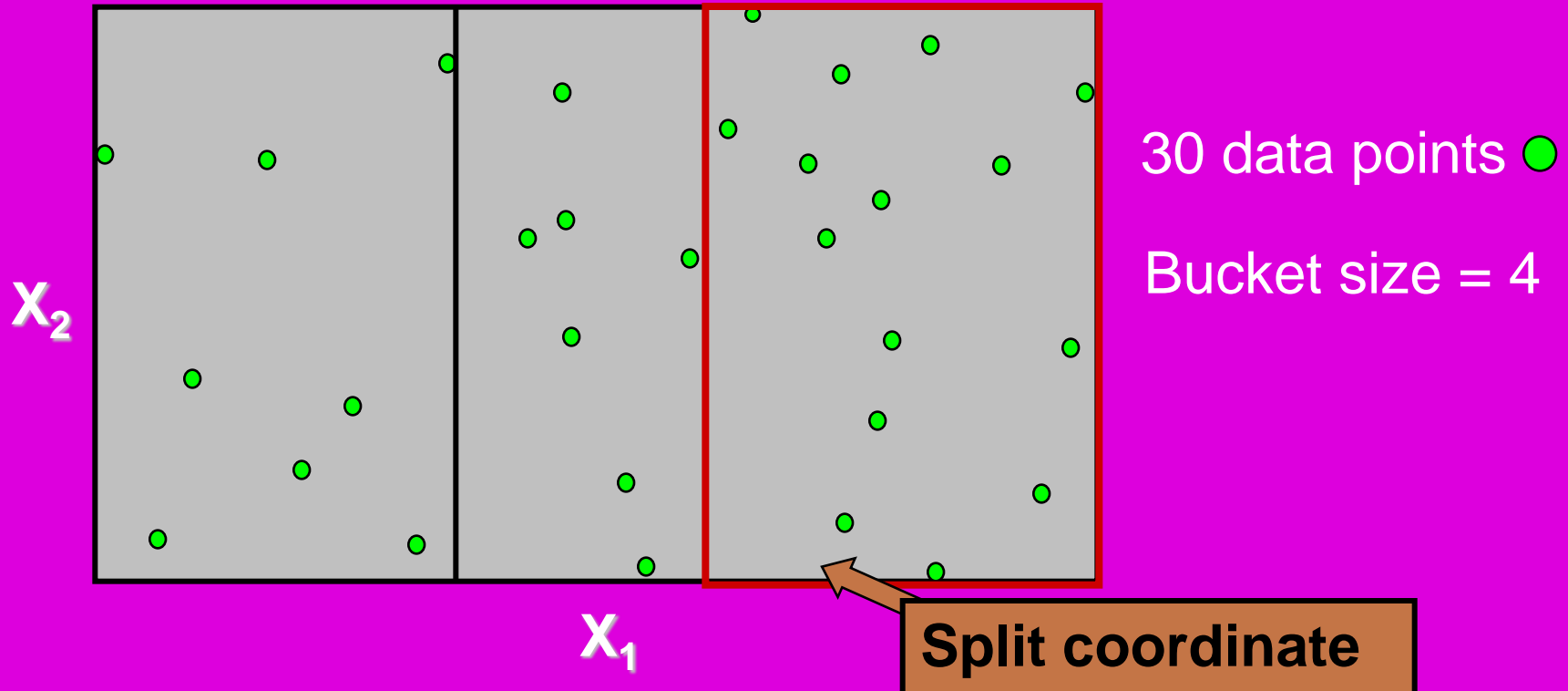
# kd Tree Construction

- \* Step 4: Repeat process for each child cell until all cells have at most 4 points

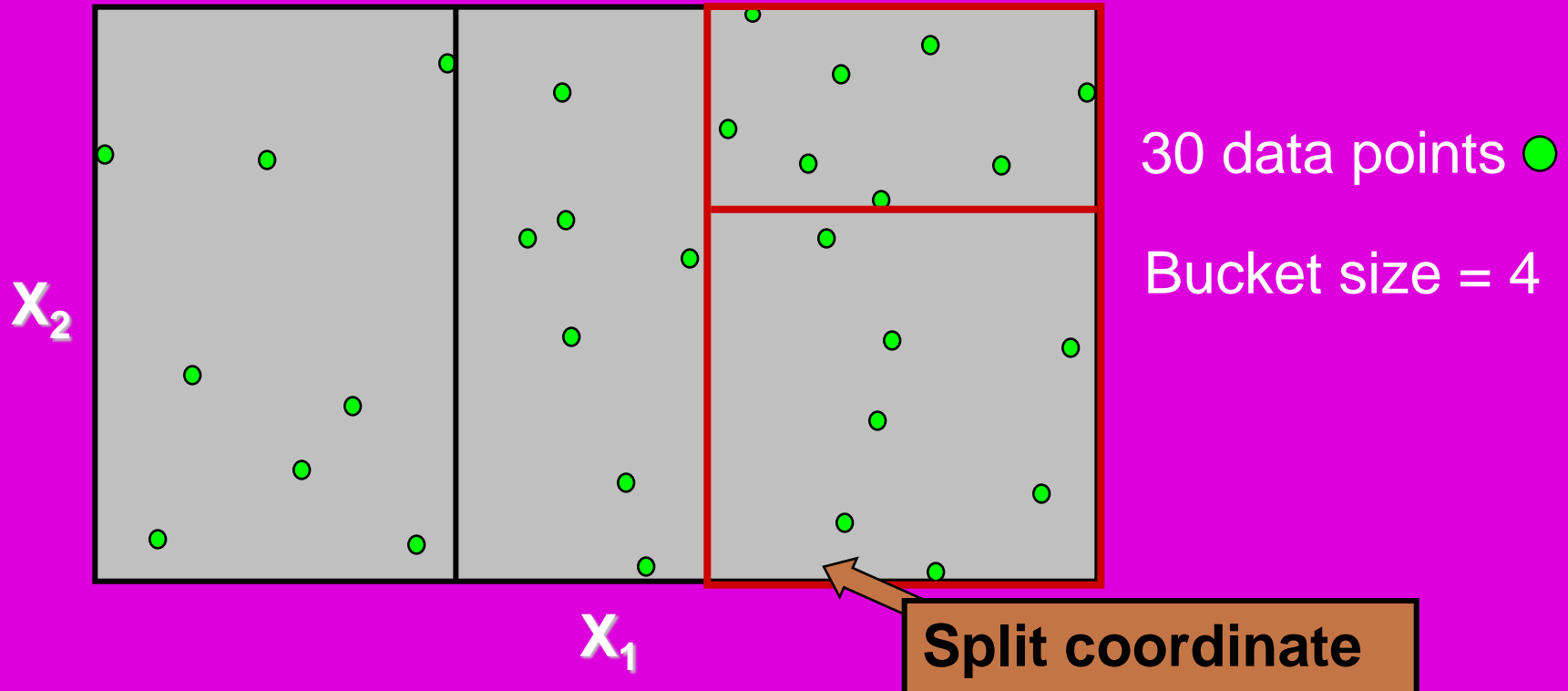




# kd Tree Construction

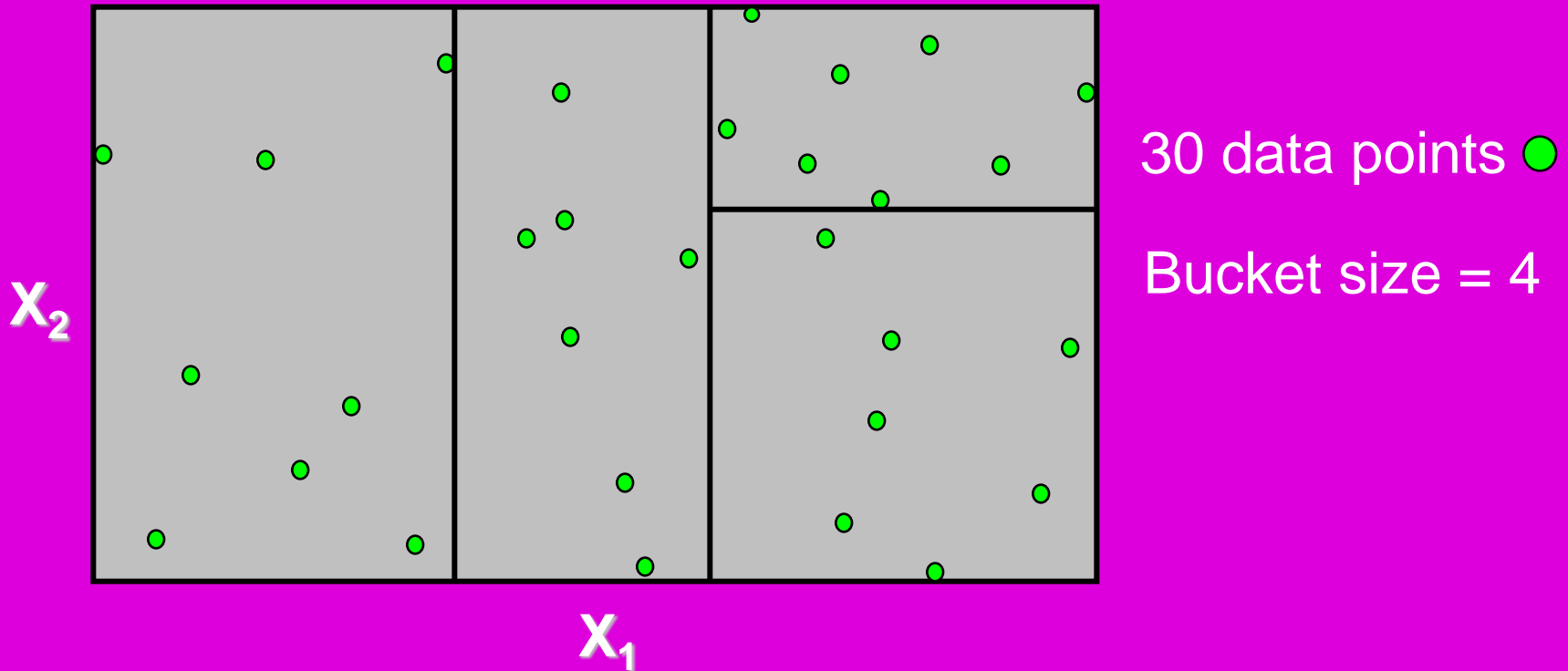


# kd Tree Construction



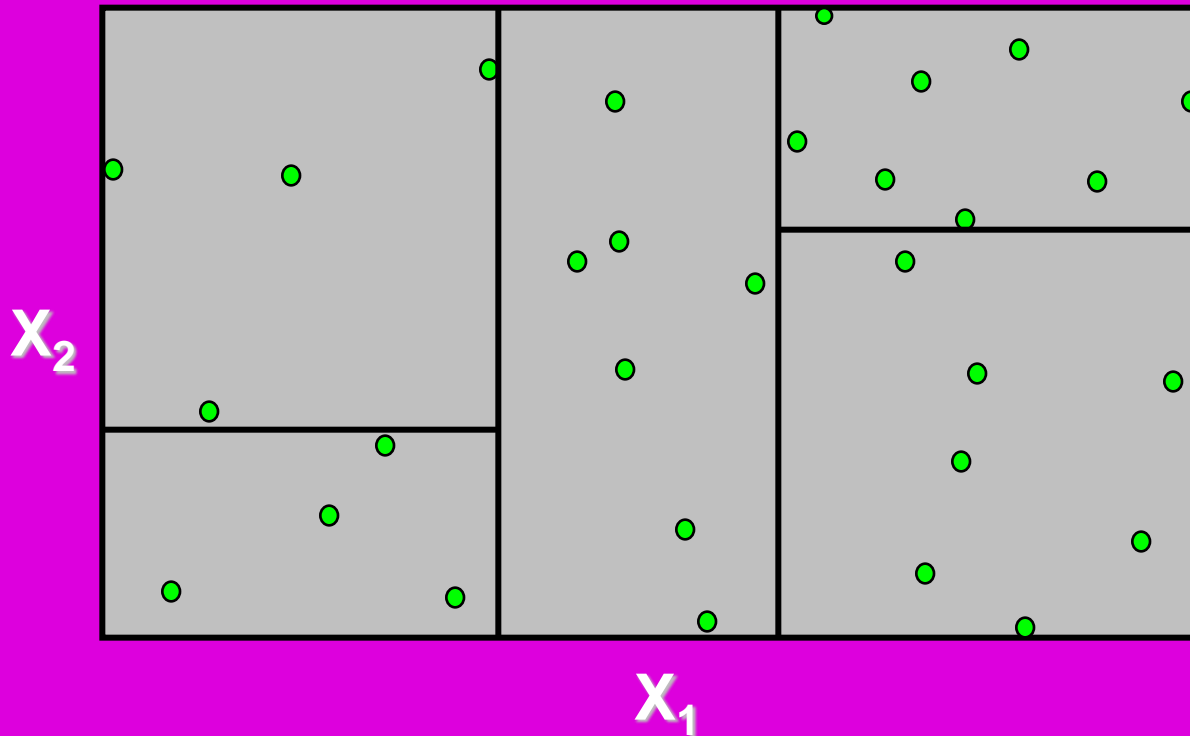
# kd Tree Construction

Step 4: Repeat process for each child cell until all cells have at most 4 points



# kd Tree Construction

- \* Step 4: Repeat process for each child cell until all cells have at most 4 points

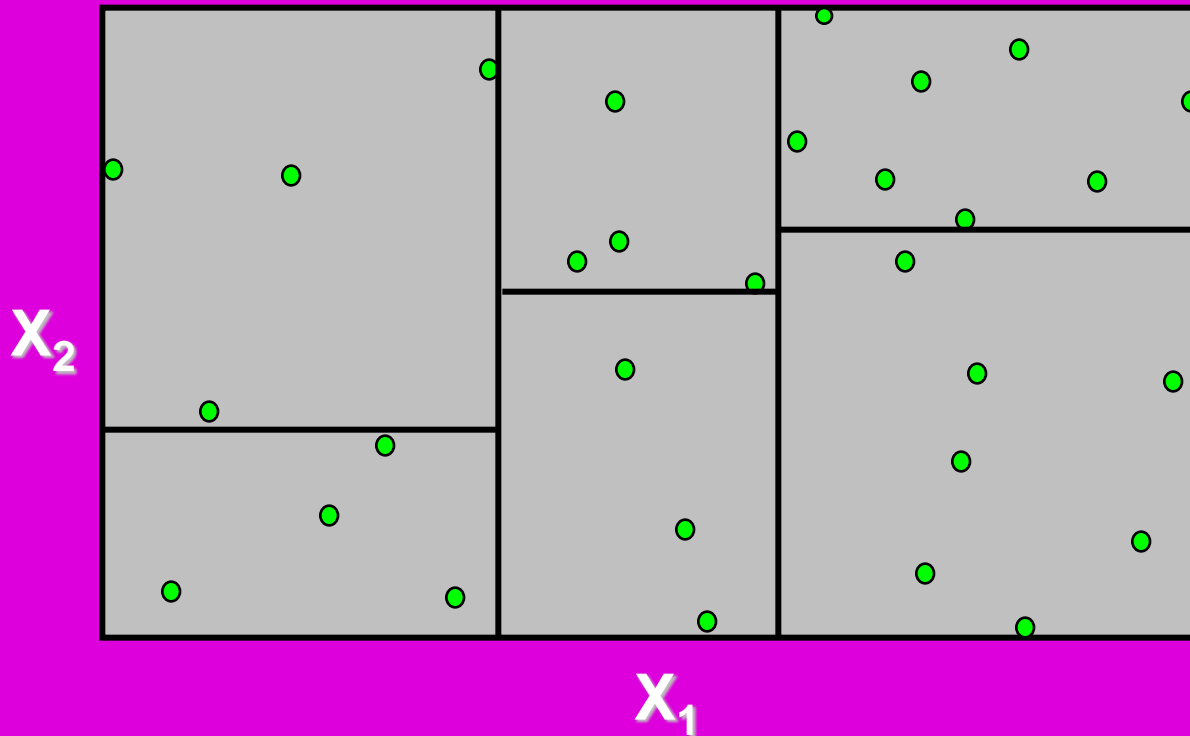


30 data points ●

Bucket size = 4

# kd Tree Construction

Step 4: Repeat process for each child cell until all cells have at most 4 points

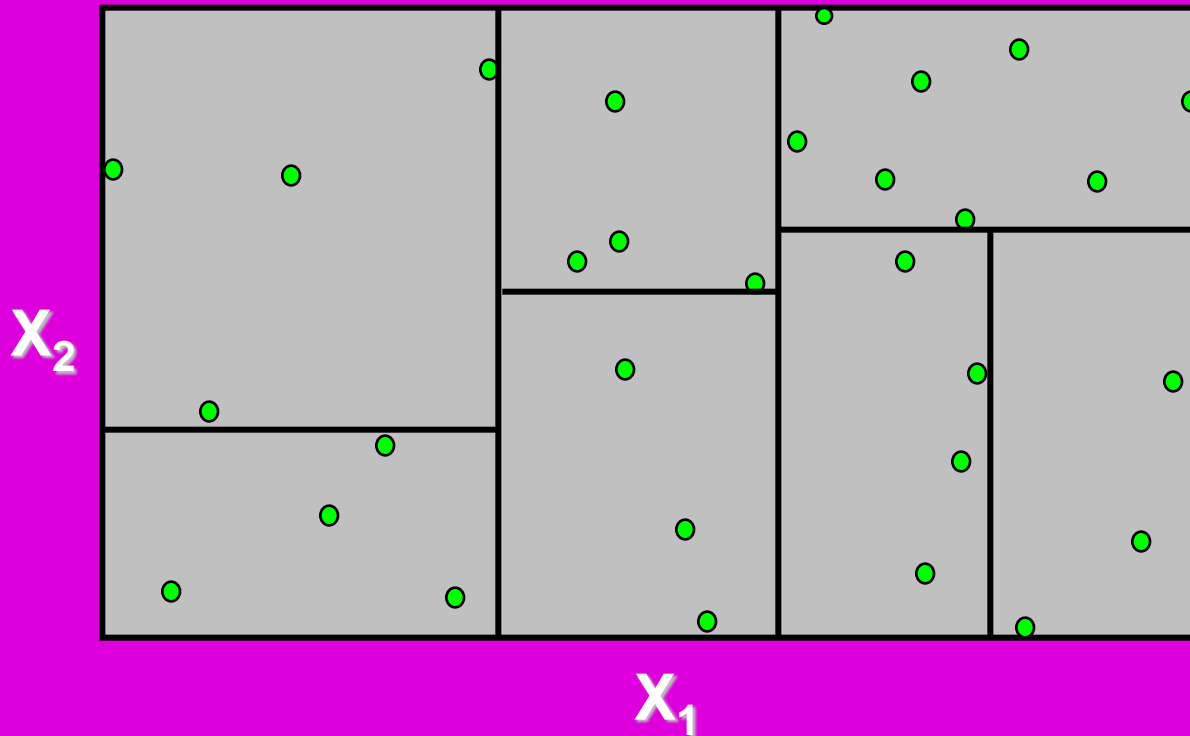


30 data points ●

Bucket size = 4

# kd Tree Construction

Step 4: Repeat process for each child cell until all cells have at most 4 points

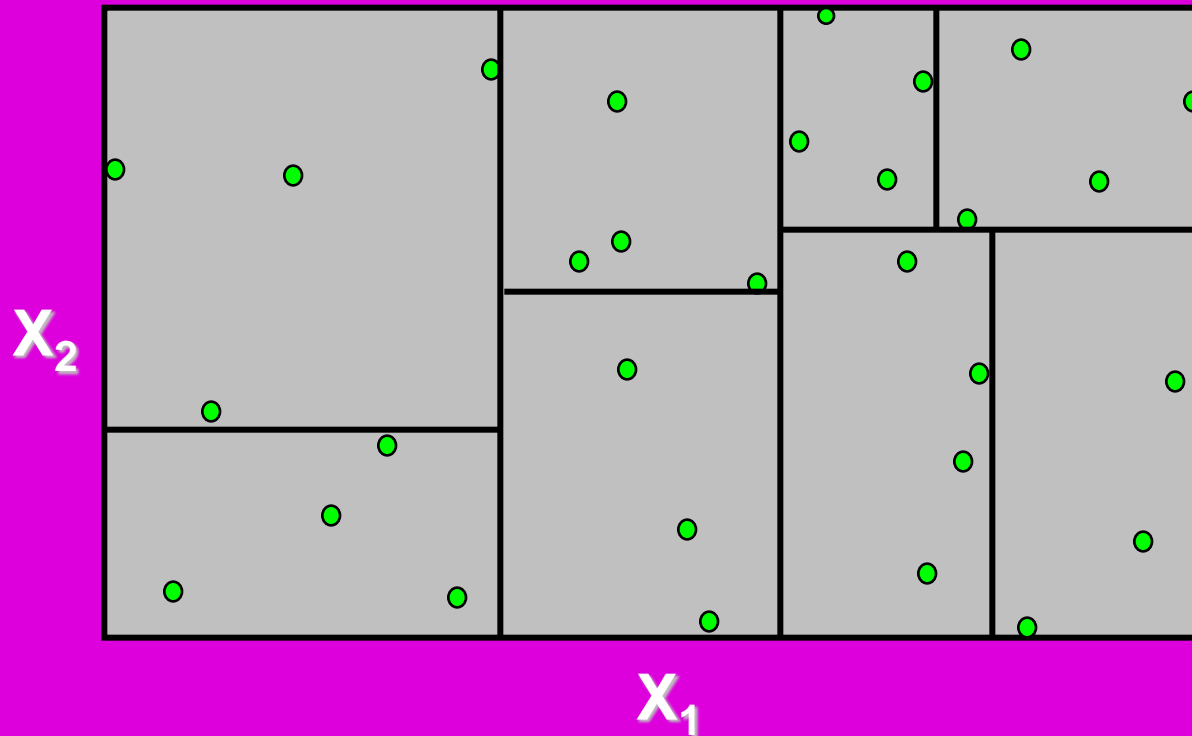


30 data points ●

Bucket size = 4

# kd Tree Construction

Step 4: Repeat process for each child cell until all cells have at most 4 points

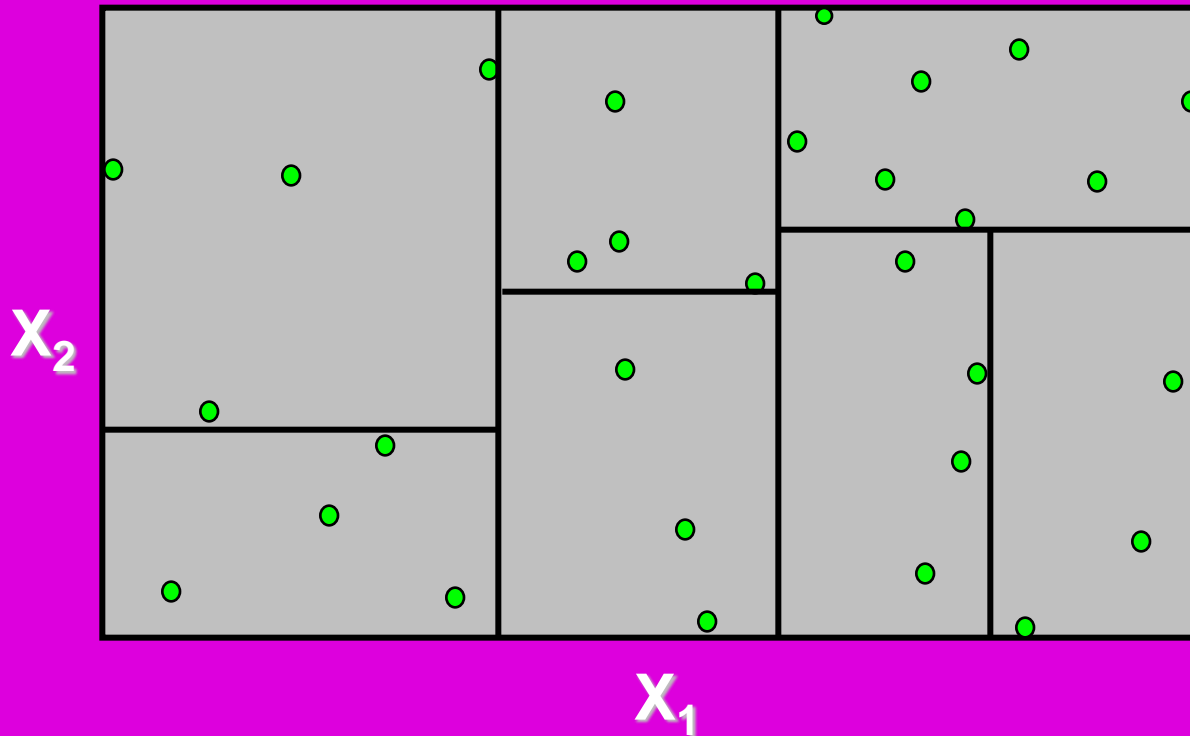


30 data points ●

Bucket size = 4

# kd Tree Construction

Step 4: Repeat process for each child cell until all cells have at most 4 points



30 data points ●

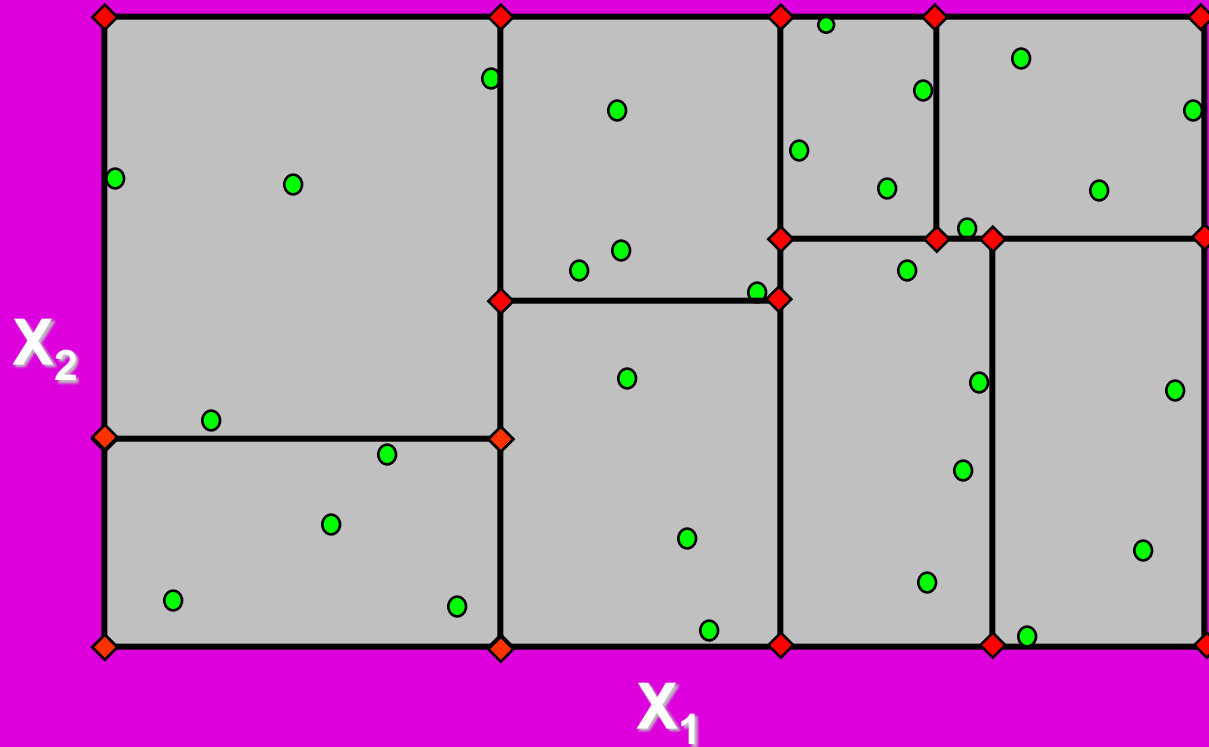
Bucket size = 4

18 fit points ◆



# kd Tree Construction

Step 5: Corners of the cells of the final kd tree are used as the fit points



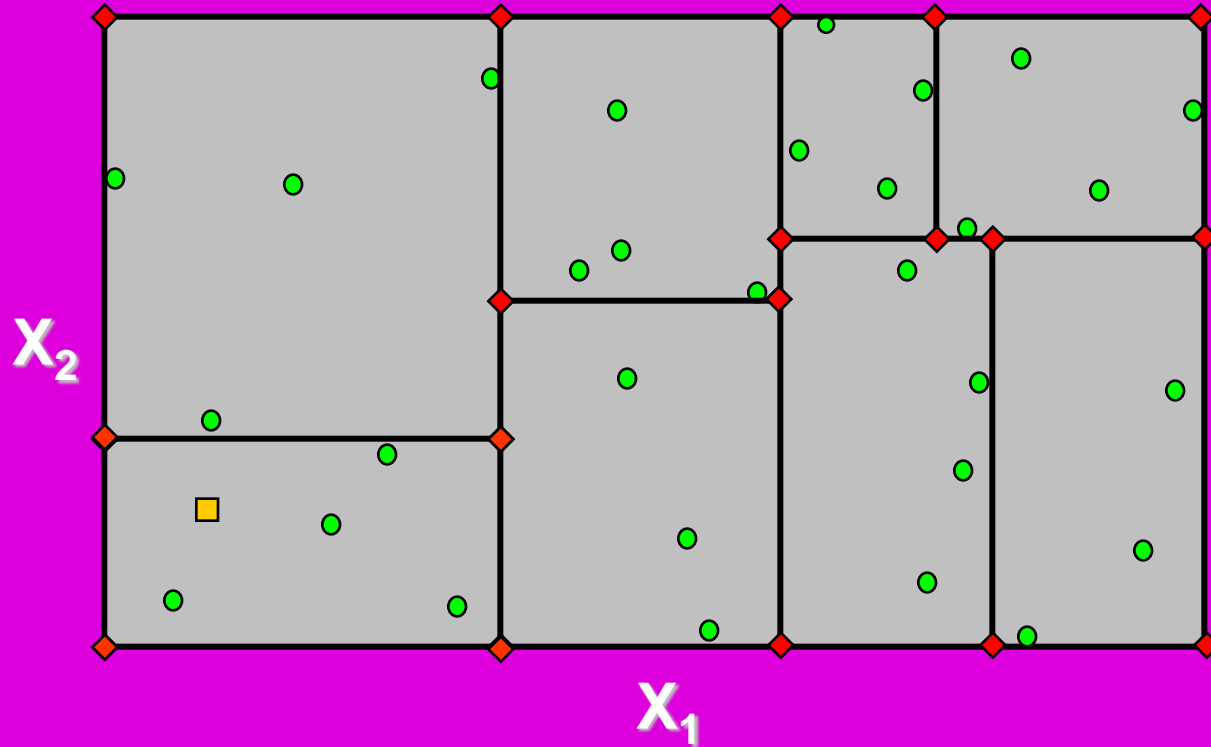
30 data points ●

Bucket size = 4

18 fit points ◆

# kd Tree Construction

Step 6: The fit at any other point is found by interpolation



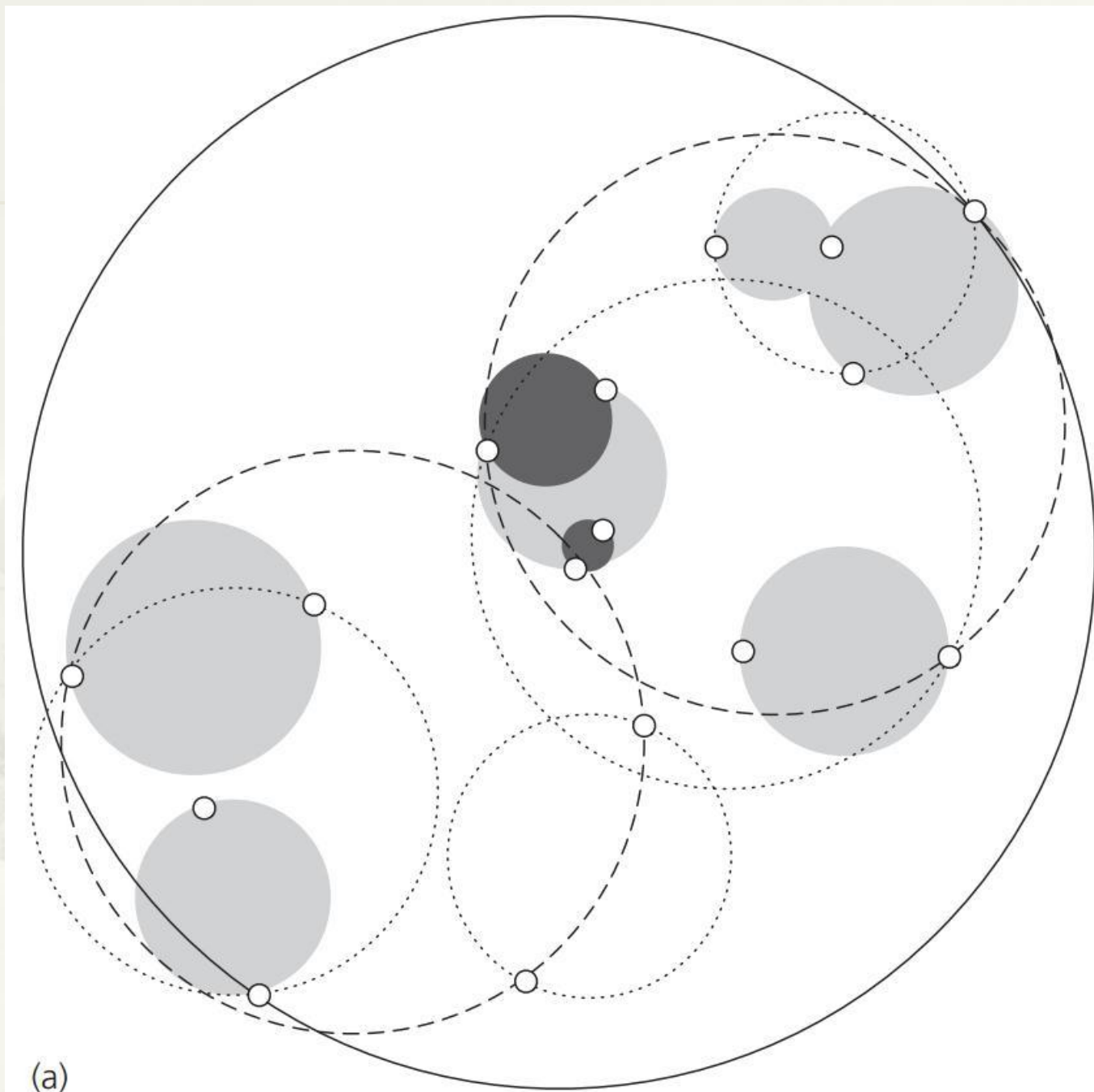
30 data points ●

Bucket size = 4

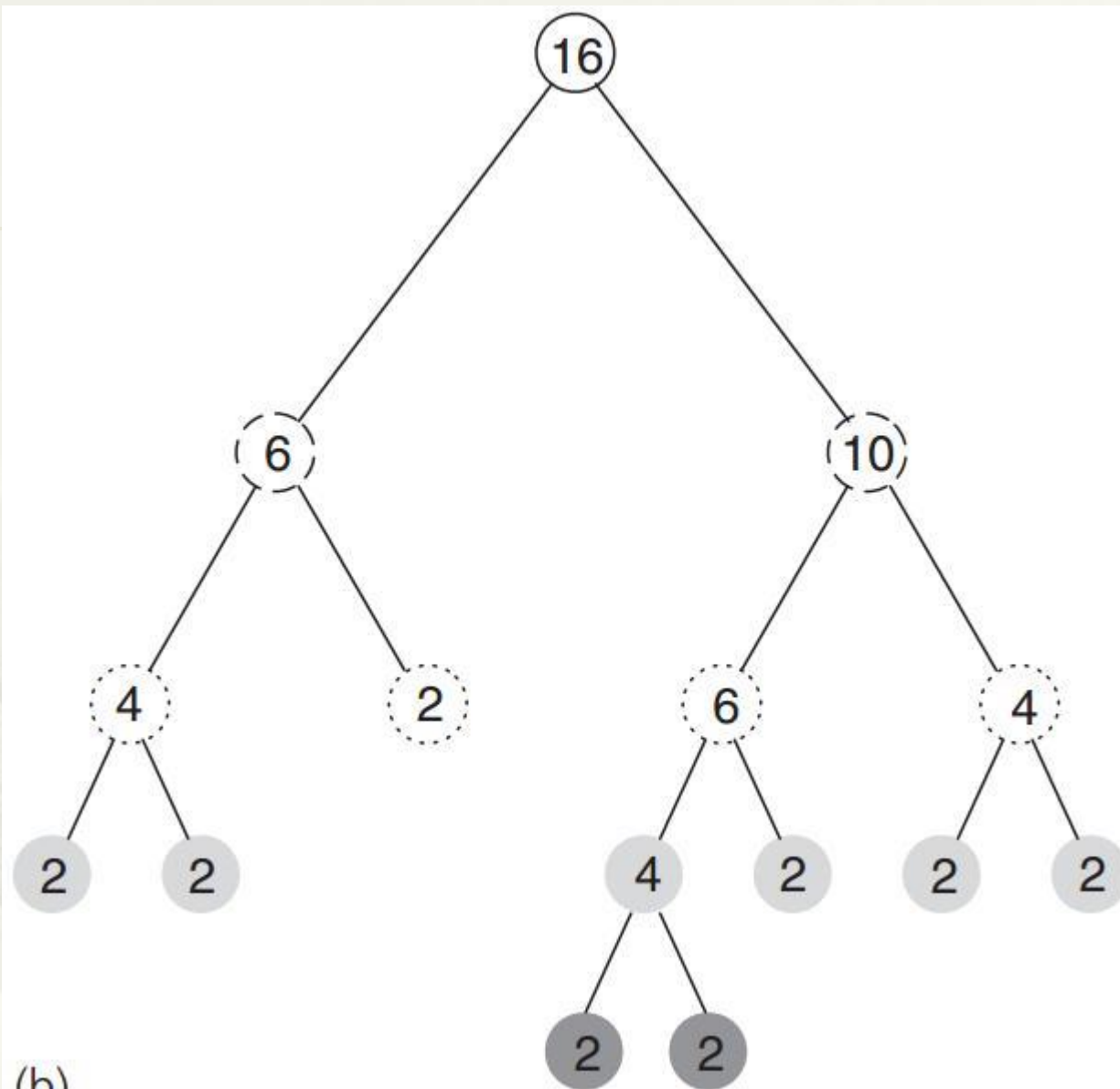
18 fit points ◆

# Ball tree

- \* 解决上面问题的方案就是使用超球面而不是超矩形划分区域。使用球面可能会造成球面间的重叠，但却没有关系。ball tree就是一个 $k$ 维超球面来覆盖这些观测点，把它们放到树里面。图7 (a)显示了一个2维平面包含16个观测实例的图,图7 (b)是其对应的ball tree，其中结点中的数字表示包含的观测点数。



- \* 选择一个距离当前圆心最远的观测点  $i_1$ ，和距离  $i_1$  最远的观测点  $i_2$ ，将圆中所有离这两个点最近的观测点都赋给这两个簇的中心，然后计算每一个簇的中心点和包含所有其所属观测点的最小半径。对包含  $n$  个观测点的超圆进行分割，只需要线性的时间。



\* 不同层次的圆被用不同的风格画出。树中的每个结点对不应一个圆，结点的数字表示该区域包含的观测点数，但并不一定就是图中该区域囊括的点数，因为重叠的情况，并且一个观测点只能属于一个区域。实际的ball tree的结点保存圆心和半径。叶子结点保存它包含的观测点。使用ball tree时，先自上而下找到包含target的叶子结点，从此结点中找到离它最近的观测点。这个距离就是最近邻的距离的上界。检查它的兄弟结点中是否包含比这个上的界更小的观测点。方法是：如果目标点的距离兄弟结点的圆心的距离大于这个圆的圆心的加上前面的上界的值，则这个兄弟结点不可能是包含所要的观测点。（如图）否则，检查这个兄弟结点是否包含符合条件的观测点。

# Python代码

```
In [2]: from numpy import *
import operator

def createDataSet():
    group = array([[1.0,1.1],[1.0,1.0],[0,0],[0,0.1]])
    labels = ['A','A','B','B']
    return group,labels

def classify0(inX,dataSet,labels,k):
    #返回“数组”的行数，如果shape[1]返回的则是数组的列数
    dataSetSize = dataSet.shape[0]
    #两个“数组”相减，得到新的数组
    diffMat = tile(inX,(dataSetSize,1))- dataSet
    #求平方
    sqDiffMat = diffMat **2
    #求和，返回的是一维数组
    sqDistances = sqDiffMat.sum(axis=1)
    #开方，即测试点到其余各个点的距离
    distances = sqDistances **0.5
    #排序，返回值是原数组从小到大排序的下标值
    sortedDistIndicies = distances.argsort()
    #定义一个空的字典
    classCount = {}
    for i in range(k):
        #返回距离最近的k个点所对应的标签值
        voteIlabel = labels[sortedDistIndicies[i]]
        #存放到字典中
        classCount[voteIlabel] = classCount.get(voteIlabel,0)+1
    #排序 classCount.iteritems() 输出键值对 key代表排序的關鍵字 True代表降序
    sortedClassCount = sorted(classCount.iteritems(),key = operator.itemgetter(1),reverse = True)
    #返回距离最小的点对应的标签
    return sortedClassCount[0][0]
```

In [ ]:

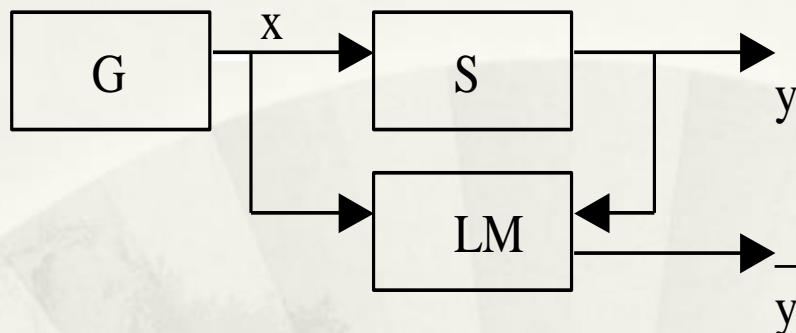


# Python代码

- \* 调用方式：打开notebook;
- \* 进入kNN文件所在的目录;
- \* 依次输入import kNN      group, labels =  
kNN.createDataSet()      kNN.classify0([0,0],g  
roup,lables,3)

# 支持向量机svm

# 学习问题



- \* 产生器 (G)，随机产生向量  $x \in R^n$ ，它带有一定但未知的概率分布函数  $F(x)$
- \* 训练器 (S)，条件概率分布函数  $F(y|x)$ ，期望响应  $y$  和输入向量  $x$  关系为  $y=f(x,v)$
- \* 学习机器 (LM)，输入-输出映射函数集  $y=f(x, w)$ ， $w \in W$ ， $W$  是参数集合。

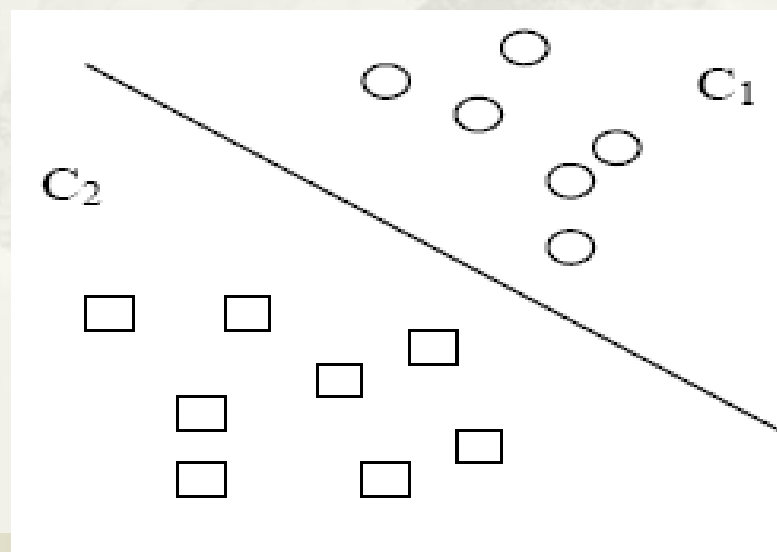
# 线性判别函数和判别面

- \* 一个线性判别函数(discriminant function)是指由 $x$ 的各个分量的线性组合而成的函数

$$g(x) = w^T x + w_0$$

- \* 两类情况：对于两类问题的决策规则为

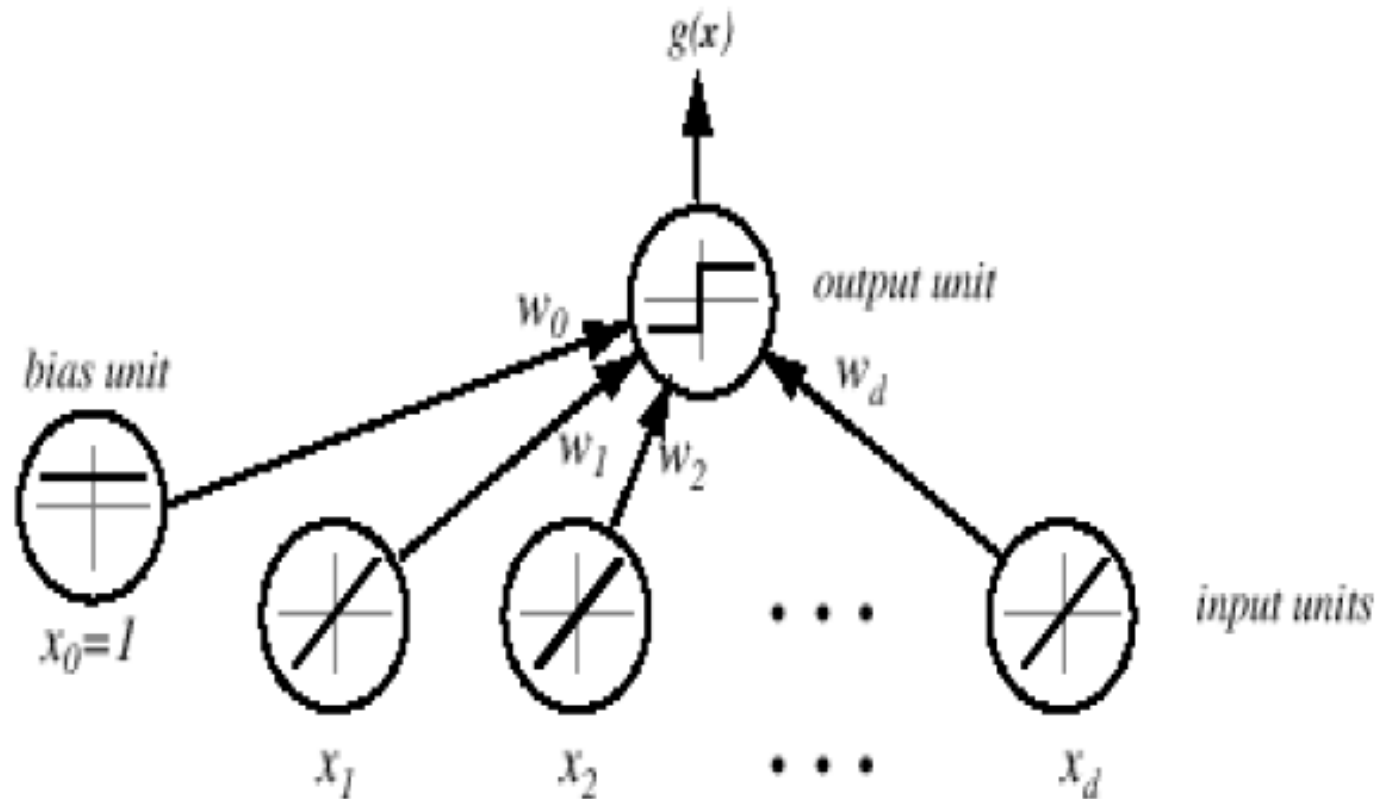
- \* 如果 $g(x) > 0$ ，则判定 $x$ 属于 $C_1$ ，
- \* 如果 $g(x) < 0$ ，则判定 $x$ 属于 $C_2$ ，
- \* 如果 $g(x) = 0$ ，则可以将 $x$ 任意分到某一类或者拒绝判定。



# 线性判别函数

- \* 下图表示一个简单的线性分类器，具有d个输入的单元，每个对应一个输入向量在各维上的分量值。该图类似于一个神经网络。

$$g(x) = w^T x + w_0$$



# 超平面

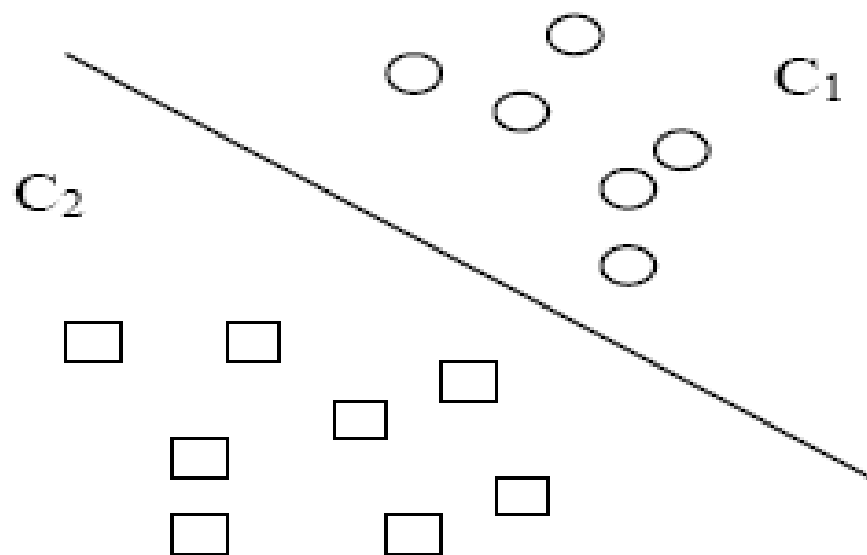
- \* 方程 $g(x)=0$ 定义了一个判定面，它把归类于 $C_1$ 的点与归类于 $C_2$ 的点分开来。
- \* 当 $g(x)$ 是线性函数时，这个平面被称为“超平面”(hyperplane)。
- \* 当 $x_1$ 和 $x_2$ 都在判定面上时，

$$w^T x_1 + w_0 = w^T x_2 + w_0$$

$$\text{或者 } w^T (x_1 - x_2) = 0$$

- \* 这表明 $w$ 和超平面上任意向量正交，并称 $w$ 为超平面的法向量。

注意到： $x_1 - x_2$ 表示超平面上一个向量



判别函数 $g(x)$ 是特征空间中某点 $x$ 到超平面的距离的一种代数度量

从下图容易看出

$$x = x_p + r \frac{w}{\|w\|}$$

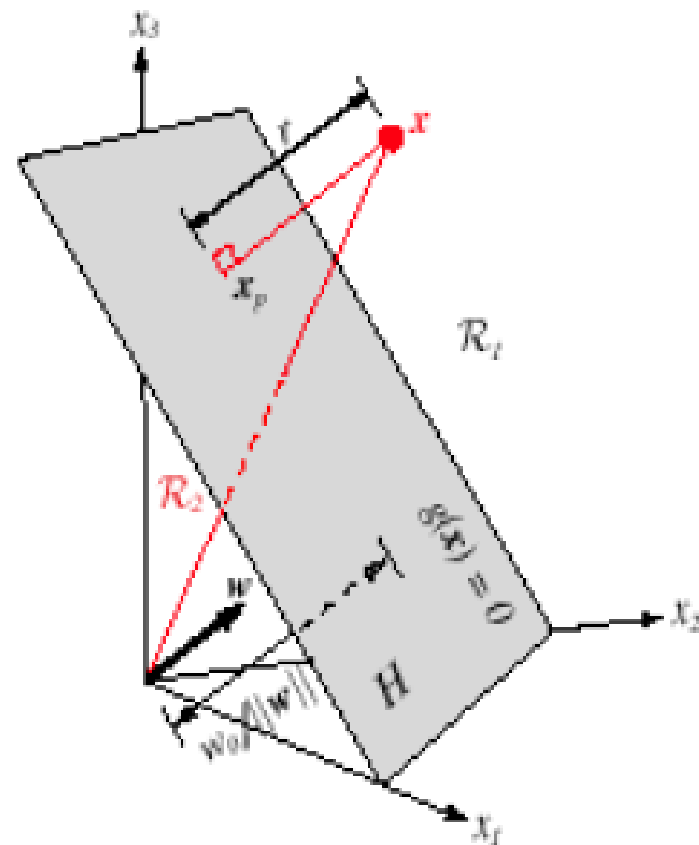
将  $x = x_p + r \frac{w}{\|w\|}$  代入  $g(x) = w^T x + w_0$  中, 我们有

$$g(x) = w^T x + w_0$$

$$= w^T \left( x_p + r \frac{w}{\|w\|} \right) + w_0$$

$$= w^T x_p + w_0 + w^T r \frac{w}{\|w\|}$$

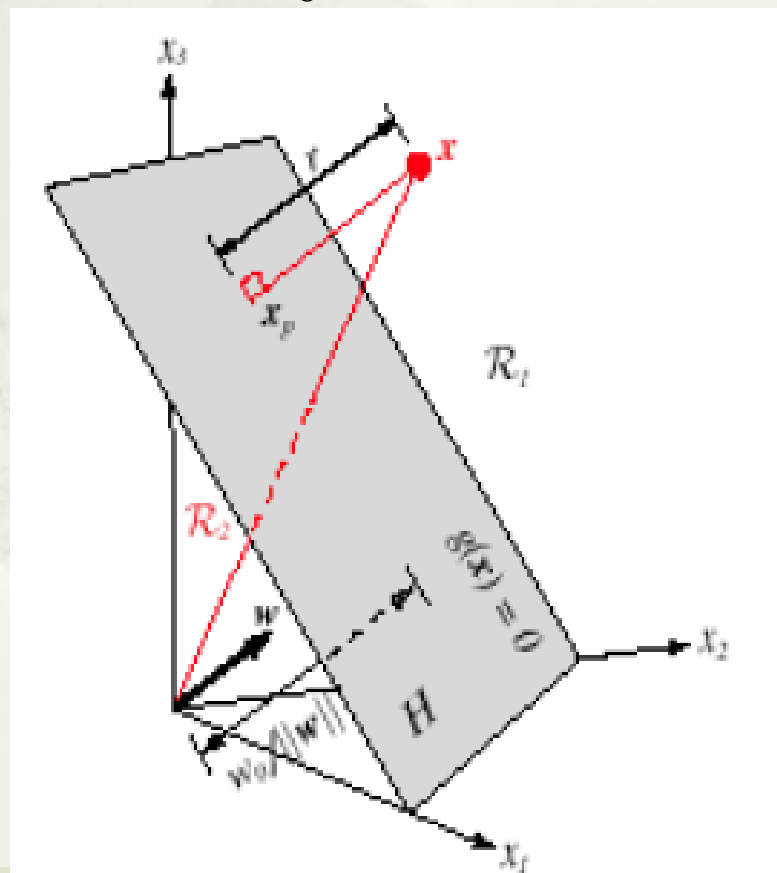
$$= r \|w\|$$



- \* 上式也可以表示为:  $r = g(x)/\|w\|$ 。当 $x=0$ 时, 表示原点到超平面的距离,  $r_0 = g(0)/\|w\| = w_0/\|w\|$ , 标示在上图中。

总之:

- 线性判别函数利用一个超平面把特征空间分隔成两个区域。
- 超平面的方向由法向量 $w$ 确定, 它的位置由阈值 $w_0$ 确定。
- 判别函数 $g(x)$ 正比于 $x$ 点到超平面的代数距离(带正负号)。当 $x$ 点在超平面的正侧时,  $g(x) > 0$ ; 当 $x$ 点在超平面的负侧时,  $g(x) < 0$



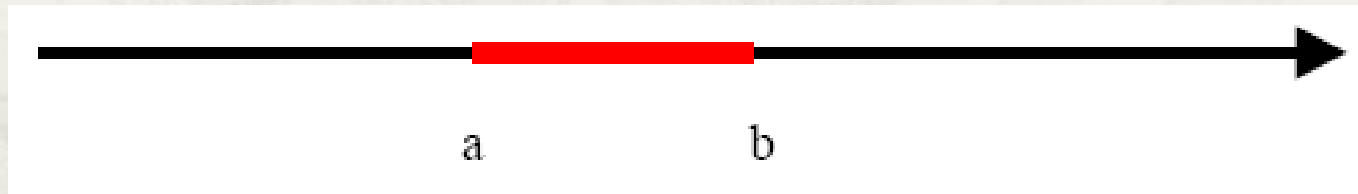


# 多类的情况

- \* 利用线性判别函数设计多类分类器有多种方法。例如
  - \* 可以把 $k$ 类问题转化为 $k$ 个两类问题，其中第 $i$ 个问题是利用线性判别函数把属于 $C_i$ 类与不属于 $C_i$ 类的点分开。
  - \* 更复杂一点的方法是用 $k(k-1)/2$ 个线性判别函数，把样本分为 $k$ 个类别，每个线性判别函数只对其中的两个类别分类。

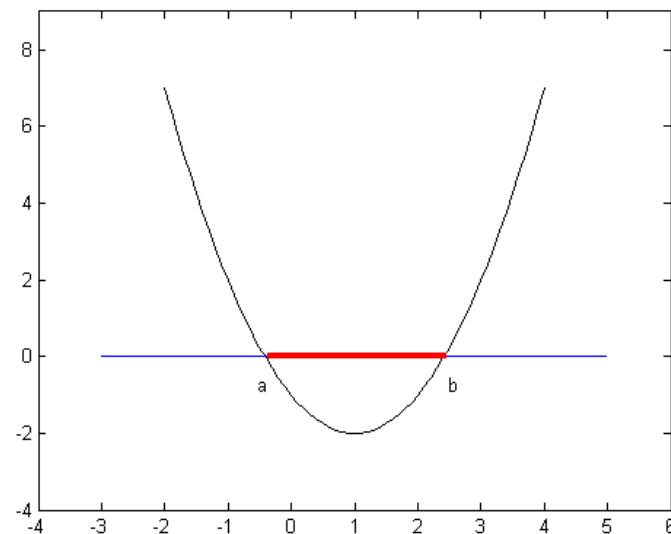
# 广义线性判别函数

在一维空间中，没有任何一个线性函数能解决下述划分问题（黑红各代表一类数据），可见线性判别函数有一定的局限性。



# 广义线性判别函数

- \* 如果建立一个二次判别函数 $g(x)=(x-a)(x-b)$ ，则可以很好的解决上述分类问题。
- \* 决策规则仍是：如果 $g(x)>0$ ，则判定 $x$ 属于 $C_1$ ，如果 $g(x)<0$ ，则判定 $x$ 属于 $C_2$ ，如果 $g(x)=0$ ，则可以将 $x$ 任意分到某一类或者拒绝判定。



# 广义线性判别函数

二次判别函数  $g(x)=(x-a)(x-b)$ 可写成如下的一般形式:

$$g(x) = c_0 + c_1x + c_2x^2$$

如果选择  $x \rightarrow y$  的映射, 则可以把二次判别函数化为关于  $y$  的线性函数

$$g(x) = a^T y = \sum_{i=1}^3 a_i y_i$$

其中,

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}, \quad a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}$$

称  $g(x) = a^T y$  为广义线性判别函数,  $a$  叫做广义权向量

# 广义线性判别函数

一般地，对于任意高次判别函数  $g(\mathbf{x})$ ，都可以通过适当的变换，化为广义线性判别函数来处理， $a^T \mathbf{y}$  不是  $\mathbf{x}$  的线性函数，但却是  $\mathbf{y}$  的线性函数。 $a^T \mathbf{y} = 0$  在  $\mathbf{Y}$  空间确定了一个通过原点的超平面。此处的  $g(\mathbf{x})$  也可以看作任意判别函数作级数展开，然后取其截尾部分的逼近。

这样我们就可以利用线性判别函数的简单性来解决复杂问题。同时带来的问题是，维数大大增加了，这将使问题很快陷入所谓的“维数灾难”。

# 设计线性分类器

所谓设计线性分类器，就是利用训练样本集建立线性判别函数

$g(x) = w^T x + w_0$  或者广义线性判别函数  $g(x) = a^T y$ 。

设计线性分类器的过程实质寻找较好的 $w$ 和 $w_0$ 或 $a$ 的过程。最好的结果往往出现在准则函数的极值点上。这样，设计线性分类器的问题就转化为利用训练样本集寻找准则函数的极值点 $w^*$ 和 $w_0^*$ 或 $a^*$ 的问题了。

常见的准则函数有：

- Fisher 准则函数

- 感知准则函数

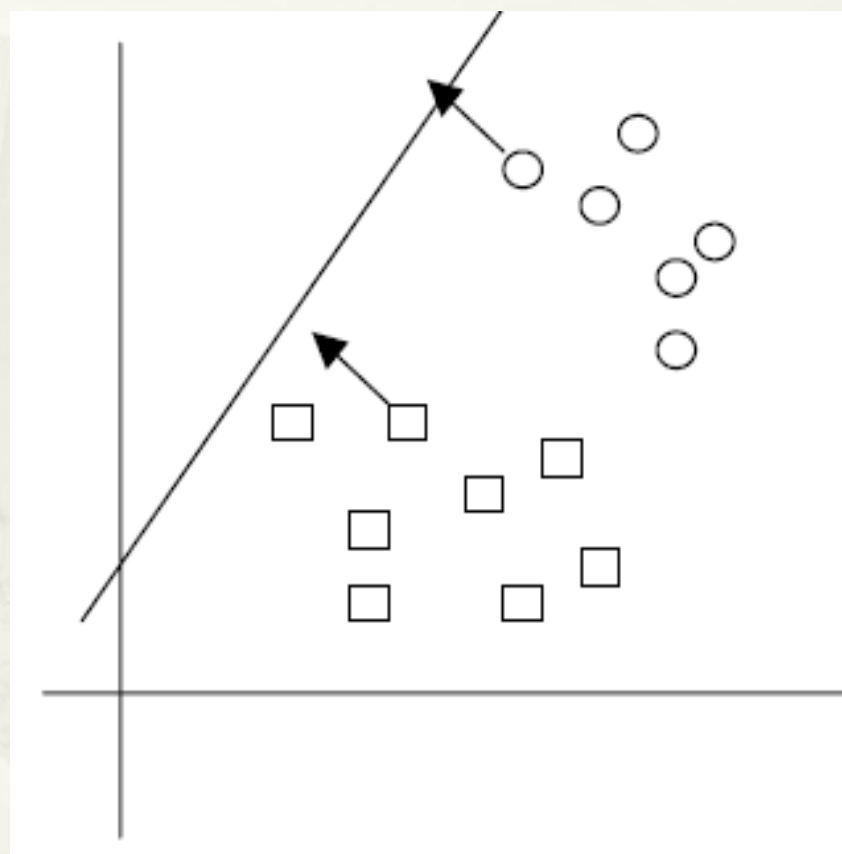
- 最小错分样本数准则函数

- 最小平方误差准则函数

- 随机最小错误率线性判别准则函数

# Fisher线性判别方法

- \* 如：Fisher线性判别方法，主要解决把 $d$ 维空间的样本投影到一条直线上，形成一维空间，即把维数压缩到一维。
- \* 然而在 $d$ 维空间分得很好的样本投影到一维空间后，可能混到一起而无法分割。
- \* 但一般情况下总可以找到某个方向，使得在该方向的直线上，样本的投影能分开的最好。
- \* 目的是降维,在低维空间中分割



# 最优分类面

- \* SVM 是从线性可分情况下的最优分类面发展而来的，基本思想可用图2的两维情况说明。

图中，方形点和圆形点代表两类样本， $H$  为分类线， $H_1$ ， $H_2$ 分别为过各类中离分类线最近的样本且平行于分类线的直线，它们之间的距离叫做**分类间隔** (margin)。

所谓最优分类线就是要求分类线不但能将两类正确分开(训练错误率为0)，而且使分类间隔最大。

推广到高维空间，最优分类线就变为**最优分类面**。

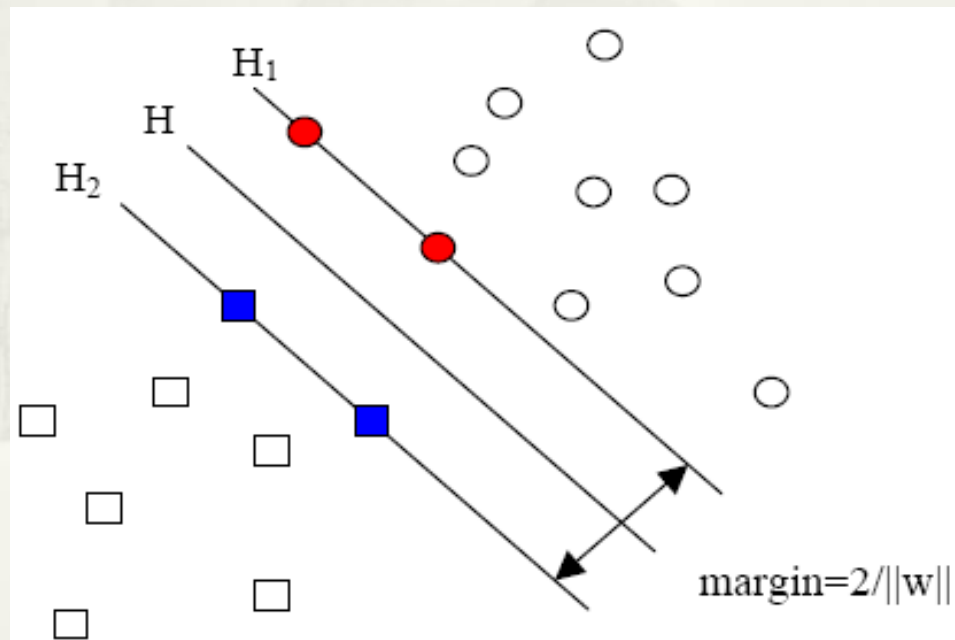


图2 线性可分情况下的最优分类线



# 最优分类面

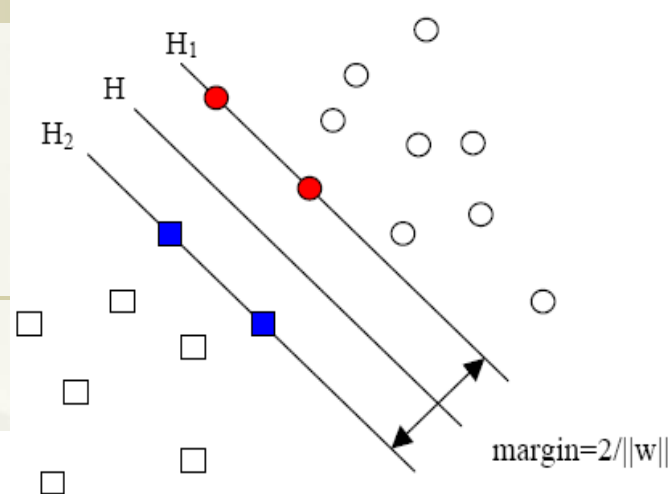


图2 线性可分情况下的最优分类线

设线性可分的样本集  $(x_i, y_i)$  ,  $i=1, \dots, n$ ,  $x \in \mathbb{R}^d$  ,  $y \in \{+1, -1\}$ 。  $d$  维空间中的线性判别函数:  $g(x) = wx + b$ , 分类面方程为  $wx + b = 0$

我们可以对它进行归一化, 使得所有样本都满足  $|g(x)| \geq 1$ , 即离分类面最近的样本满足  $|g(x)| = 1$ , 这样分类间隔就等于  $2/||w||$ 。 因此要求分类间隔最大, 就是要求  $||w||$  (或  $||w||^2$ ) 最小。 而要求分类面对所有样本正确分类, 就是要求满足

$$y_i[wx_i + b] - 1 \geq 0 \quad , \quad i=1, \dots, n, \quad (1)$$

因此, 满足上面公式且使  $||w||^2$  最小的分类面就是最优分类面。 过两类样本中离分类面最近的点且平行于最优分类面的超平面  $H_1, H_2$  上的训练样本, 就是使上式等号成立的样本称作支持向量。

# 如何求最优分类面

求最优分类面问题可以转化为如下的约束优化问题：

$$\begin{array}{ll}\text{minise} & \frac{1}{2} \|w\|^2 \\ \text{subject to} & y_i[wx_i + b] - 1 \geq 0 \quad i = 1, 2, \dots, l\end{array}$$

这是一个二次凸规划问题，由于目标函数和约束条件都是凸的，根据最优化理论，这一问题存在唯一全局最小解。

应用Lagrange乘子法并满足KKT条件（Karush-kuhn-Tucher）：

$$\alpha_i \{y_i[wx_i + b] - 1\} = 0 \quad (5)$$

最后可得到解上述问题的最优分类函数为：

$$f(x) = \text{sgn}\{w^* \cdot x + b^*\} = \text{sgn}\left\{\sum_{i=1}^k \alpha_i^* y_i (x_i \cdot x) + b^*\right\}$$

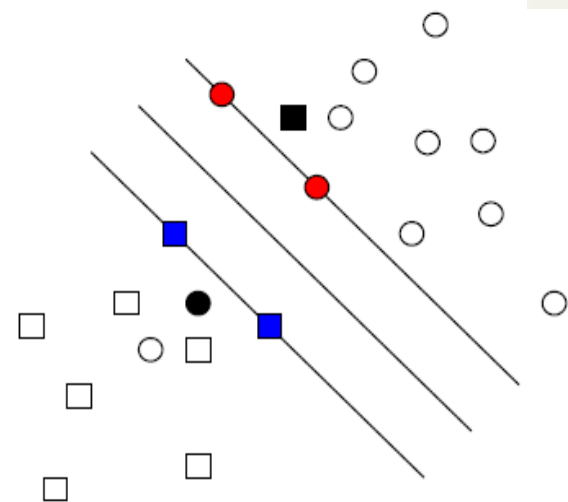
其中  $\alpha^*$  ,  $b^*$  为确定最优划分超平面的参数， $(x_i \cdot x)$  为两个向量的点积。

由式(5) 知：非支持向量对应的  $\alpha_i$  都为零，求和只对少数支持向量进行。

# 最优分类面

在线性不可分的情况下，可以在条件  $y_i[wx_i + b] - 1 \geq 0$  中增加一个松弛项  $\xi_i$   
 $\geq 0$  成为

$$y_i[wx_i + b] - 1 + \xi_i \geq 0$$



将目标改为求  $\frac{1}{2}\|w\|^2 + C(\sum_{i=1}^n \xi_i)$  最小，即折衷考虑最少错分样本和最大分类间隔，就得到广义最优分类面。其中， $C > 0$  是一个常数，它控制对错分样本惩罚的程度。

# 支持向量机

上面所得到的最优分类函数为：

$$f(x) = \text{sgn}\{w^* \cdot x + b^*\} = \text{sgn}\left\{\sum_{i=1}^k \alpha_i^* y_i (x_i \cdot x) + b^*\right\}$$

- \* 该式只包含待分类样本与训练样本中的支持向量的内积运算，可见，要解决一个特征空间中的最优线性分类问题，我们只需要知道这个空间中的内积运算即可。
- \* 对非线性问题，可以通过非线性变换转化为某个高维空间中的线性问题，在变换空间求最优分类面。这种变换可能比较复杂，因此这种思路在一般情况下不易实现。

# 支持向量机

根据泛函的有关理论, 只要一种核函数 $K(x_i, x_j)$  满足Mercer 条件, 它就对应某一变换空间中的内积. 因此, 在最优分类面中采用适当的内积函数 $K(x_i, x_j)$  就可以实现某一非线性变换后的线性分类, 而计算复杂度却没有增加.

相应的分类函数也变为

$$f(x) = \text{sgn}\{\sum_{i=1}^k \alpha_i^* y_i K(x_i \cdot x) + b^*\}$$

这就是支持向量机.

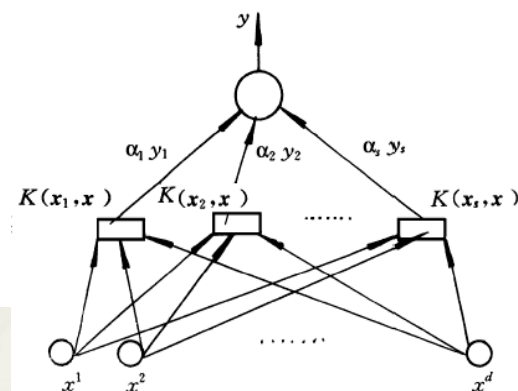


图3 支持向量机示意图

概括地说, 支持向量机就是首先通过用内积函数定义的非线性变换将输入空间变换到一个高维空间, 在这个空间中求最优分类面.

SVM 分类函数形式上类似于一个神经网络, 输出是中间节点的线性组合, 每个中间节点对应一个输入样本与一个支持向量的内积, 因此也被叫做支持向量网络。

# 核函数技术

---

- \* 特征空间与核函数
- \* 核函数构造



# 特征空间与核函数

- \* 乘积特征

- \* 模式:  $\mathbf{x} \in \mathbf{R}^N$

- \* 大部分信息包含在d-阶乘积中:

$$x_{j_1} \cdot \dots \cdot x_{j_d}, \quad j_1, \dots, j_d \in \{1, \dots, N\}$$

- \* 提取所有乘积特征，把输入模式映射到所有d-阶乘积的特征空间F中，在这个空间中构造算法。

# 特征空间与核函数

\* 二维  $\Phi: \mathbf{R}^2 \rightarrow F = \mathbf{R}^3$   
 $(x_1, x_2) \mapsto (x_1^2, x_2^2, x_1 x_2)$

\* N维 
$$N_F = \frac{(N + d - 1)!}{d!(N - 1)!}$$

16x16的图像， $d=5$ ，则 $N_F$ :  $10^{10}$

\* 如何处理这么高的维数？



# 特征空间与核函数

- \* 核函数导出的多项式特征空间
  - \* 如何在输入空间中计算特征空间中的点积？

$$k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}))$$

二维

$$C_2 : (x_1, x_2) \mapsto (x_1^2, x_2^2, x_1x_2, x_2x_1)$$

$$(C_2(\mathbf{x}) \cdot C_2(\mathbf{y})) = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 = (\mathbf{x} \cdot \mathbf{y})^2$$

# 特征空间与核函数

- \* 命题 1.1:

- \* 映射  $C_d$  把  $\mathbf{x} \in \mathbf{R}^N$  映射到向量  $C_d(\mathbf{x})$ ,  $C_d(\mathbf{x})$  的项是  $\mathbf{x}$  的项的所有有序  $d$  阶乘积。则有:

$$k(\mathbf{x}, \mathbf{y}) = (C_d(\mathbf{x}) \cdot C_d(\mathbf{y})) = (\mathbf{x} \cdot \mathbf{y})^d$$

- \* 直到  $d$  阶的所有乘积项:

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^d$$

# 特征空间与核函数

## \* 核函数的定义

核是一个函数 $k$ ，对所有 $x, y \in X$ ，

$$k(x, y) = \phi(x) \cdot \phi(y)$$

成立。其中 $\phi: X \rightarrow R^n$ 是从输入空间 $X$ 到特征空间 $F$ (Hilbert空间)的映射。

# 特征空间与核函数

- \* Mercer核

- \* 多项式核

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^d$$

- \* 高斯径向基函数核

$$k(\mathbf{x}, \mathbf{y}) = \exp \left( -\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2 \sigma^2} \right)$$

- \* Sigmoid核

$$k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \Theta)$$

(只在部分参数值情况下才满足核函数的定义)

# 特征空间与核函数

## \* 正定核

\* 命题1.2:  $X$ 是一个集合,  $K: X \times X \rightarrow R$ ,  
如果 $K$ 是对称的, 并且 $K$ 是正定的, 也就是  
对于任何  $x_1, \dots, x_N \in X$   $K_{ij} = K(x_i, x_j)$   
定义的矩阵 $K$ 是正定的, 也就是对所有  
 $c_1, \dots, c_N \in R$ , 有  $\sum_{ij} c_i c_j K_{ij} \geq 0$   
那么, 可以构造到特征空间 $F$ 的映射  $\phi$ , 使得  
 $K(x_i, x_j) = (\phi(x_i) \cdot \phi(x_j))$

\* 逆命题也成立

# 特征空间与核函数

- \* 作为两两相似性的核函数值

- \* 上述命题不需要 $X$ 是向量空间中的元素的集合。它们可以是任何对象的集合，只要这个集合对于某个函数 $k$ (可以看作是对对象的相似性量度)产生一个正定矩阵。基于两两距离或者相似性的方法最近引起关注。这种方法的好处是适用于很难把数据表示成向量的情况(例如，文本聚类).

# 特征空间与核函数

- \* 点积定义了输入空间的几何性质：

$$\|x\|^2 = (x, x) \quad \cos \theta = \frac{(x, y)}{\sqrt{(x, x) \cdot (y, y)}}$$

$$\|x - y\|^2 = (x, x) + (y, y) - 2(x, y)$$

- \* 核函数定义了特征空间的几何性质，是输入空间上的伪距离：

$$D(x, y) = K(x, x) + K(y, y) - 2K(x, y)$$

# 核函数的选择

用不同核函数  $k(\mathbf{x}, \mathbf{x}_i)$  可以构造实现输入空间中不同类型的非线性决策面的学习机,

从而导致不同的支持向量算法。在实际问题中,通常是直接给出核函数。常用的核函数有

1) 线性核函数 (linear kernel)  $k(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x} \cdot \mathbf{x}_i)$

2) 多项式核函数 (polynomial kernel)  $k(\mathbf{x}, \mathbf{x}_i) = (s(\mathbf{x} \cdot \mathbf{x}_i) + c)^d$ , 其中  $s, c, d$  为参数。显然, 线性核函数可以看作多项式核函数的一种特殊情况。


3) 径向基核函数 (radical basis function, rbf)  $k(\mathbf{x}, \mathbf{x}_i) = \exp(-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2)$ , 其中  $\gamma$  为参数。

4) Sigmoid 核函数 (Sigmoid tanh)  $k(\mathbf{x}, \mathbf{x}_i) = \tanh(s(\mathbf{x} \cdot \mathbf{x}_i) + c)$ , 其中  $s, c$  为参数。



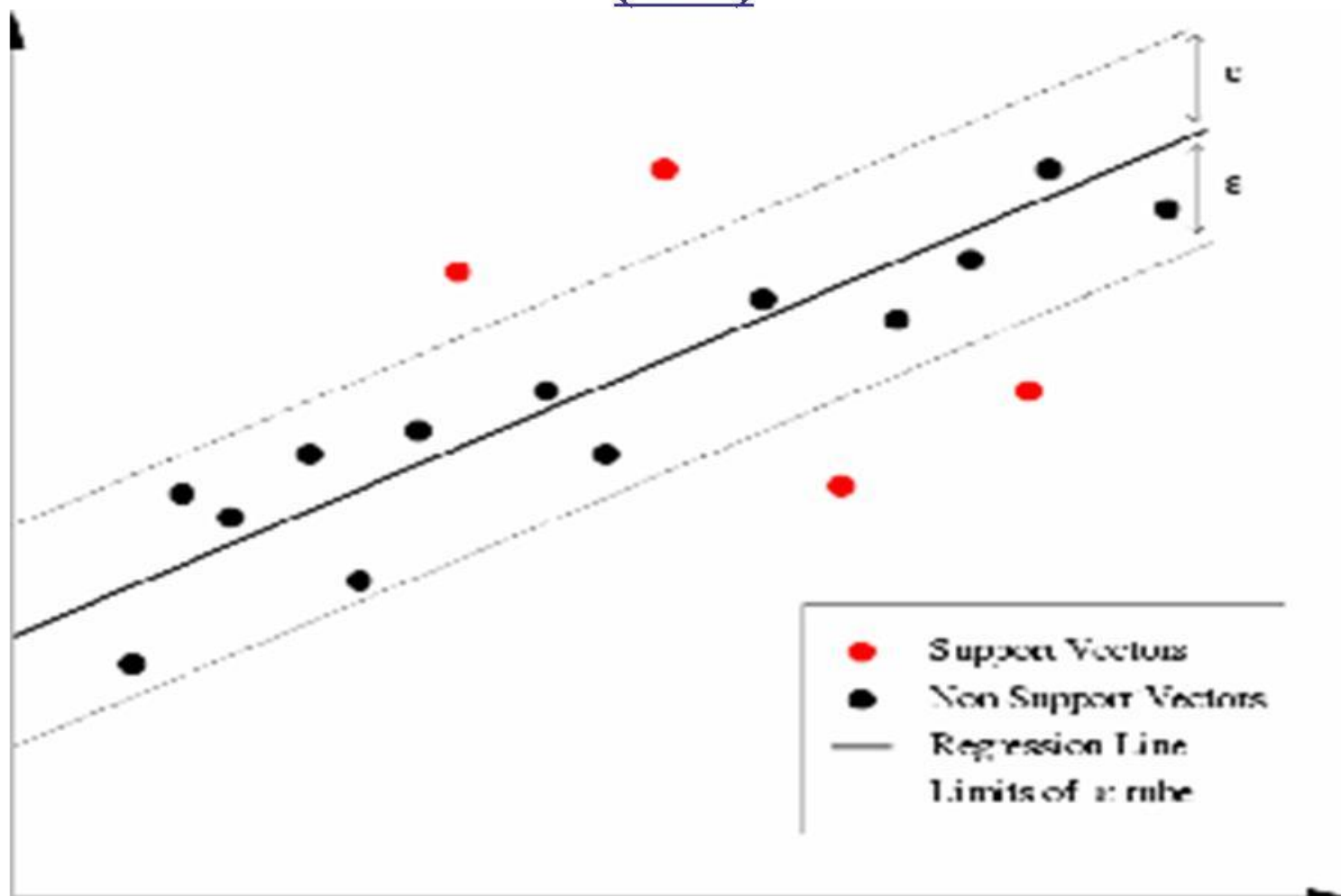
# 支持向量机其他形式

---

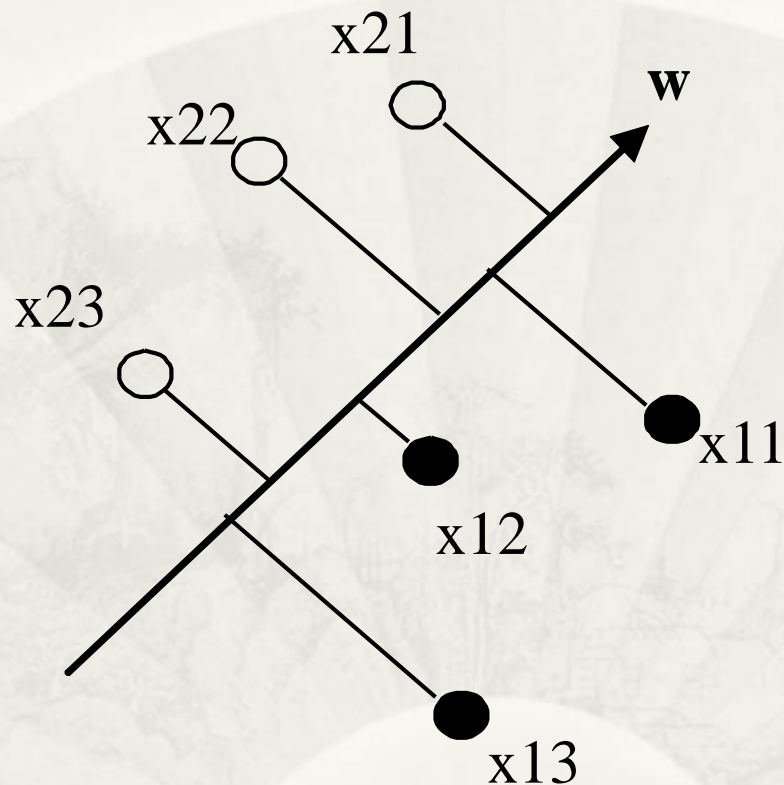
- \* 回归支持向量机
  - \* 排序支持向量机
- 
- A decorative background graphic featuring a large, light-colored fan. The fan's surface is covered with a traditional Chinese landscape painting, showing mountains, trees, and a body of water. The fan is positioned diagonally, with its handle at the bottom center and its edges extending towards the top left and top right corners of the slide.

# 回归支持向量机.(SVR)

## Principle of Support Vector Regression (SVR)



# 排序支持向量机(Ranking SVM)



# SVM 应用

- \* 近年来SVM 方法已经在图像识别、信号处理和基因图谱识别等方面得到了成功的应用, 显示了它的优势。
- \* SVM 通过核函数实现到高维空间的非线性映射, 所以适合于解决本质上非线性的分类、回归和密度函数估计等问题。
- \* 支持向量方法也为样本分析、因子筛选、信息压缩、知识挖掘和数据修复等提供了新工具。

# 案例分析：约会成功概率预测

---

