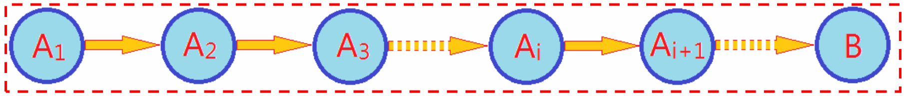
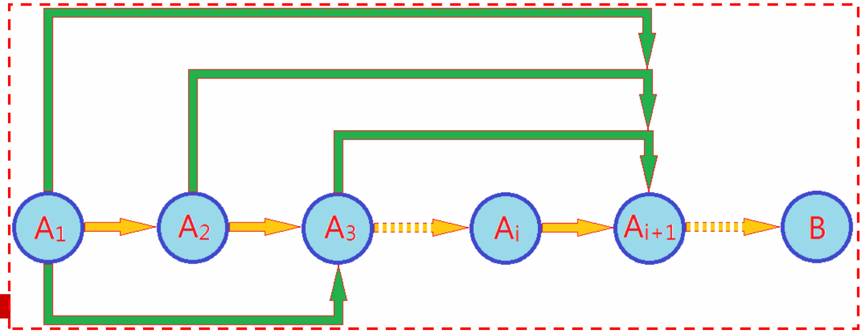
概率、分治、机器学习

七月算法 邹博

2015年5月7日



动态规划总结



- 相对于前面使用存储结构来划分章节：“数组”、“字符串”、“树”、“图”(它们是世界观)，动态规划是方法论，是解决一大类问题的通用思路。事实上，前面章节论述的很多内容都可以归结为动态规划的思想。
  - 如：KMP中求next数组的过程：已知next[0…i-1]，求next[i]；
- 何时可以考虑使用动态规划：
  - 初始规模下能够方便的得出结论
    - 空串、长度为0的数组、自身等
  - 能够得到问题规模增大导致的变化
    - 递推式——状态转移方程
- 事实上，动态规划还有“无后效性”的要求
  - 一旦计算得到了A[0…i-1]，那么，计算A[i]时只可能读取A[0…i-1]，而不会更改它们的值——过去发生的，只能承认，不能改变；
  - 一旦计算得到了A[0…i-1]，那么，计算A[i]时只需要读取A[0…i-1]的值即可，不需要事先知道A[i+1…n-1]的值——未来的事情，完全未知。
- 在实践中往往忽略无后效性这一要求：
  - 要么问题本身决定了它是成立的：格子取数问题；
  - 要么通过更改计算次序，可以达到该要求：矩阵连乘I问题。

卡塔兰数 Catalan

- n个矩阵连乘，可以分解成i个矩阵连乘和(n-i)个矩阵连乘，最后，再将这两个矩阵相乘。故：

$$P(n)=\begin{cases} 1 & n=1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n>1 \end{cases} \Rightarrow P(n)=\Omega\left(\frac{4^n}{\sqrt{\pi} * n^{3/2}}\right)$$

$$P(n)=\frac{1}{n}C_{2n-2}^{n-1}$$

- 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452……

卡塔兰数 Catalan $H(n)=\frac{1}{n+1}C_{2n}^{n}$

- 有N个节点的二叉树共有多少种情形？
- 一个栈(无穷大)的进栈序列为1,2,3,..n,有多少个不同的出栈序列？
- 凸多边形三角化：将一个凸多边形划分成三角形区域的方法有多少种？
- 由左而右扫描由n个1和n个0组成的2n位二进制数，要求在任何时刻，1的累计数不小于0的累计数。求满足这样条件的二进制数的个数。

计算机中的概率计算

- 伪随机数
  - 选择方便获取的“随机”事件
    - 读取当前系统时间？
    - 读取某经常变化的系统文件长度？
  - 数学公式
    - Hash杂凑？
    - 可以获得几乎和均匀分布性质相当的实践结果。

Code

```
/***
*int rand() - returns a random number
*
*Purpose:
*       returns a pseudo-random number 0 through 32767.
*
*Entry:
*       None.
*
*Exit:
*       Returns a pseudo-random number 0 through 32767.
*
*Exceptions:
*
*******************************************************************************/

int __cdecl rand (
        void
        )
{
#ifdef _MT

        _ptiddata ptd = _getptd();

        return( ((ptd->_holdrand = ptd->_holdrand * 214013L
            + 2531011L) >> 16) & 0x7fff );

#else  /* _MT */
        return(((holdrand = holdrand * 214013L + 2531011L) >> 16) & 0x7fff);
#endif  /* _MT */
}
```

辅助结构

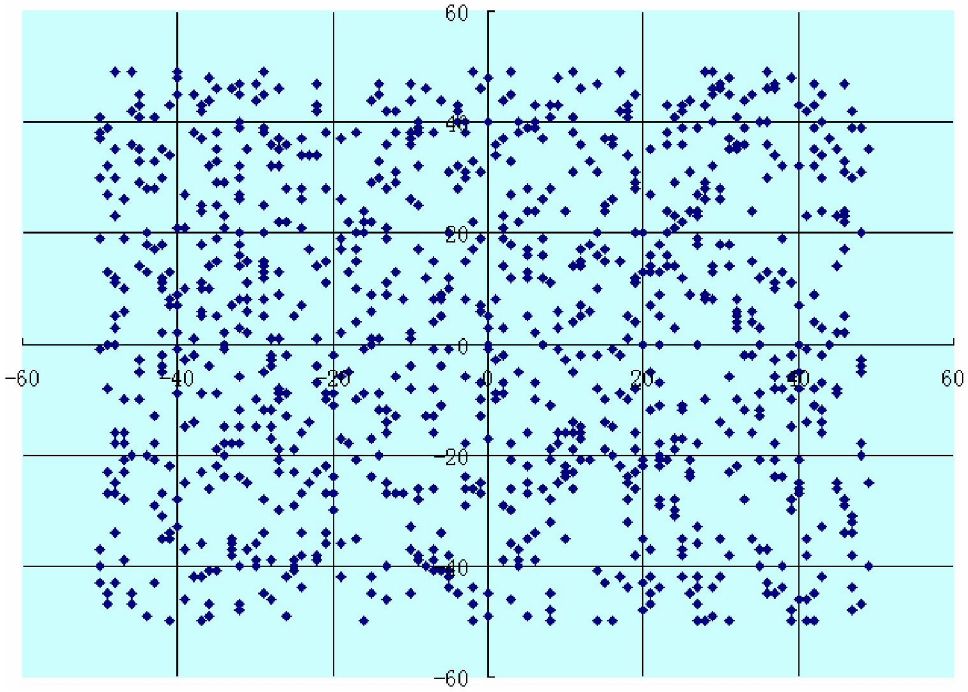
```
#ifndef _MT
static long holdrand = 1L;
#endif  /* _MT */

/***
*void srand(seed) - seed the random number generator
*
*Purpose:
*       Seeds the random number generator with the int given.  Adapted from the
*       BASIC random number generator.
*
*Entry:
*       unsigned seed - seed to seed rand # generator with
*
*Exit:
*       None.
*
*Exceptions:
*
*******************************************************************************/

void __cdecl srand (
        unsigned int seed
        )
{
#ifdef _MT

        _getptd()->_holdrand = (unsigned long)seed;

#else  /* _MT */
        holdrand = (long)seed;
#endif  /* _MT */
}
```

产生二维随机数

- 给定区间$[a_x,b_x]\times[a_y,b_y]$，使得二维随机点(x,y)落在等概率落在区间的某个点上。
- 分析：因为两个维度是独立的，分别生成两个随机数即可。

代码与效果

```
int rand50()
{
    return rand() % 100 - 50;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ofstream oFile;
    oFile.open(_T("D:\\rand.txt"));
    int x,y;
    for(int i = 0; i < 1000; i++)
    {
        x = rand50();
        y = rand50();
        oFile << x << '\t' << y << '\n';
    }
    oFile.close();
    return 0;
}
```

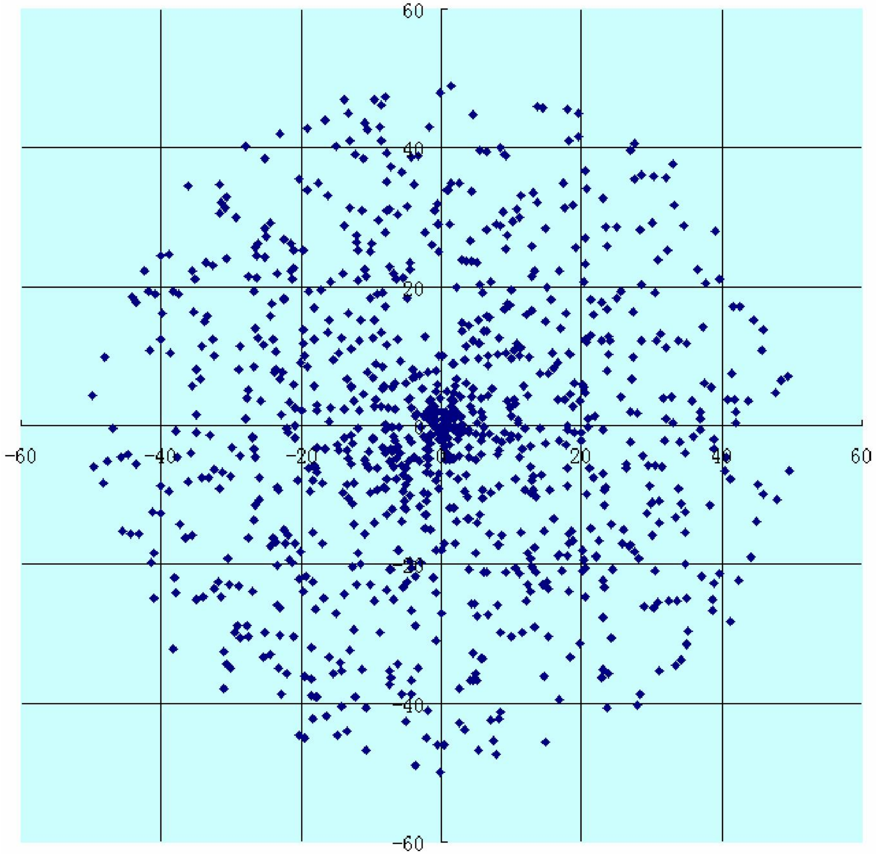


圆内均匀取点

- 给定定点$O(x_0,y_0)$和半径r，使得二维随机点(x,y)等概率落在圆内。
- 分析
  - 因为均匀分布的数据是具有平移不变性，生成半径为r，定点为圆心的随机数(x_1,y_1)，然后平移得到(x_1+x_0,y_1+y_0)即可。
  - 直接使用$x=r*\cos\theta$，$y=r*\sin\theta$是否可以呢？
    - 具体试验一下。

代码与效果

```
int rand50()
{
    return rand() % 100 - 50;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ofstream oFile;
    oFile.open(_T("D:\\rand.txt"));
    double r,theta;
    double x,y;
    for(int i = 0; i < 1000; i++)
    {
        r = rand50();
        theta = rand();
        x = r*cos(theta);
        y = r*sin(theta);
        oFile << x << '\t' << y << '\n';
    }
    oFile.close();
    return 0;
}
```

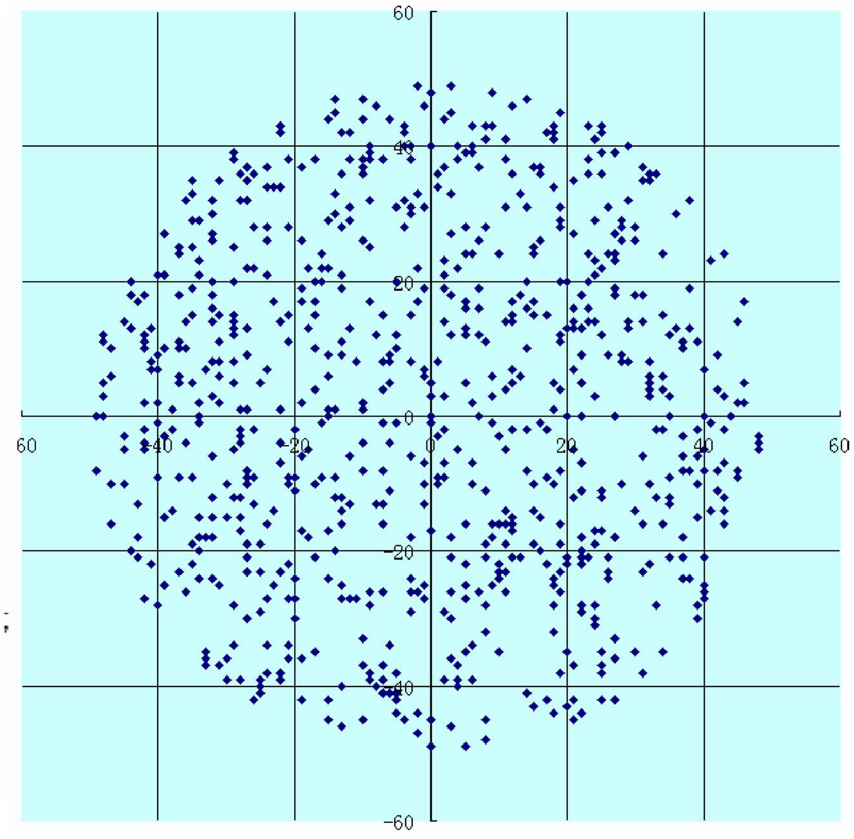


代码与效果

- 显然上述做法是不对的。但可以使用二维随机点的做法，若落在圆外，则重新生成点。结果如下。

代码与效果

```
int rand50()
{
    return rand() % 100 - 50;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ofstream oFile;
    oFile.open(_T("D:\\rand.txt"));
    int x,y;
    for(int i = 0; i < 1000; i++)
    {
        x = rand50();
        y = rand50();
        if(x*x + y*y < 2500)
            oFile << x << '\t' << y << '\n';
    }
    oFile.close();
    return 0;
}
```



思考

- 不是每次生成随机数都能退出该算法
  - 有一定的接受率。
  - 请问：
    - 以多大的概率1次退出：接受率是多少？
    - 得到随机数的需要的平均次数(期望)是多少？
- 这个做法简洁、有效，值得推荐；
  - 许多相关问题，往往可以如此解决。

例题

- 已知有个rand7()的函数，返回1到7随机自然数，让利用这个rand7()构造rand10() 随机1~10。

分析

- 因为rand7仅能返回1~7的数字，少于rand10的数目。因此，多调用一次，从而得到49种组合。超过10的整数倍部分，直接丢弃。

Code

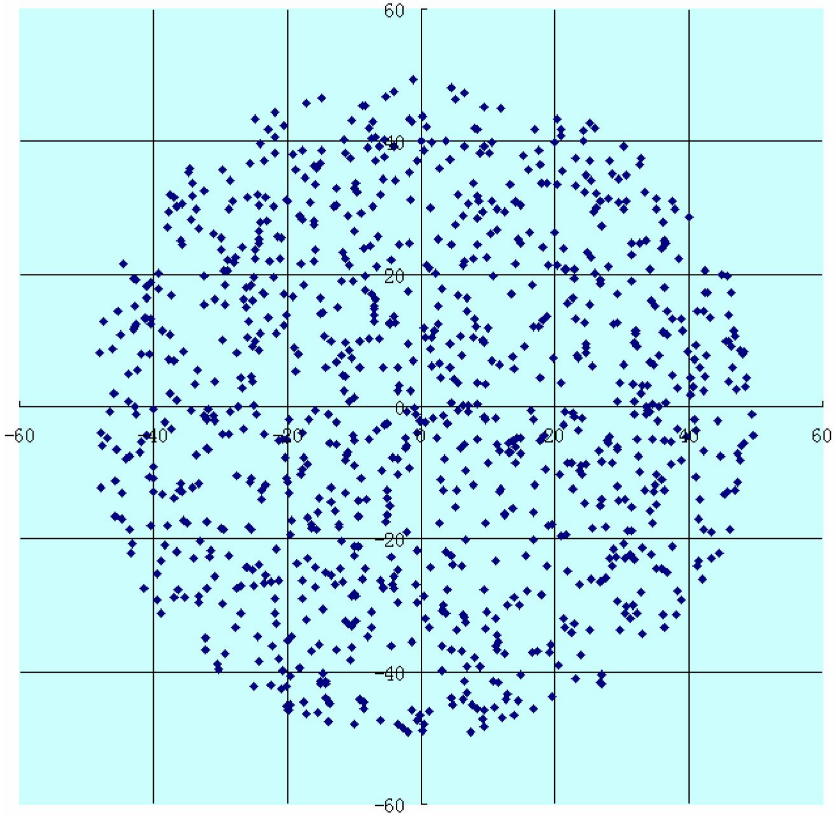
```
int rand10()
{
	int a1, a2, r;
	do
	{
		a1 = rand7() - 1;
		a2 = rand7() - 1;
		r = a1 * 7 + a2;
	} while (r >= 40);
	return r / 4 + 1;
}
```

圆内均匀取点的1次成功算法

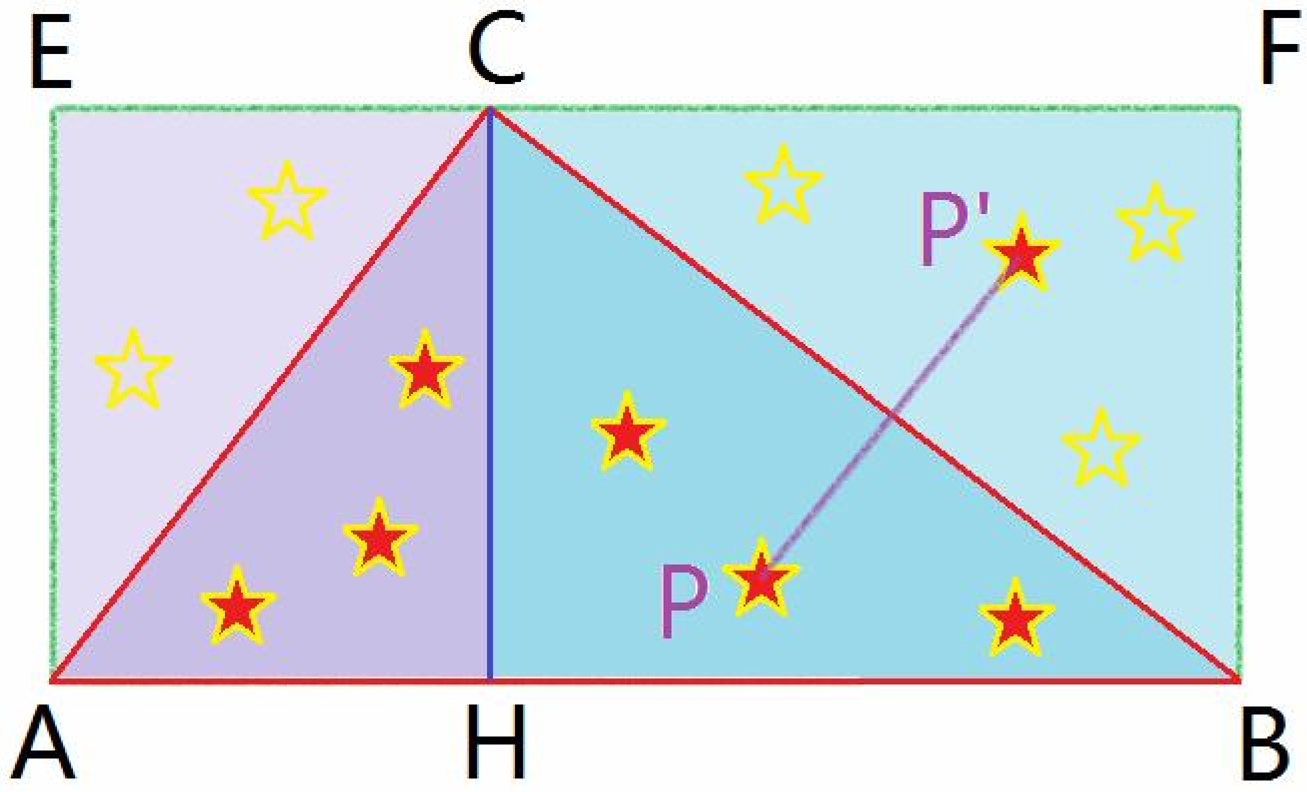
- 问题分析：把随机点看做面积很小的区域，圆内均匀取点意味着随机点P的面积与圆的面积成正比。
- $S_p=kS$
- $S=\pi r^2$，与半径的平方成正比
- 从而，$S_p(r)=k\pi r^2$
- 将均匀生成的随机数x取平方根赋值给r；则$S_p(r)$即为均匀分布。
- 同时，是与角度θ无关的，即：取均匀分布的随机数θ作为旋转角即可。

代码与效果

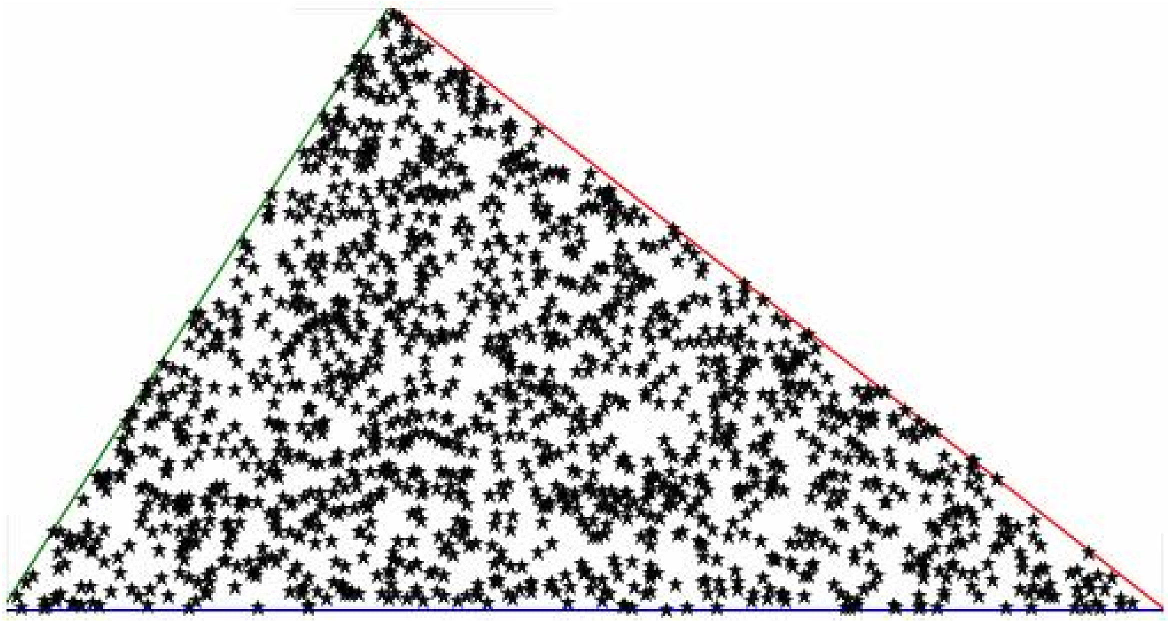
```
double rand2500()
{
    return rand() % 2500;
}

int _tmain(int argc, _TCHAR* argv[])
{
    ofstream oFile;
    oFile.open(_T("D:\\rand.txt"));
    double r,theta;
    double x,y;
    for(int i = 0; i < 1000; i++)
    {
        r = sqrt(rand2500());
        theta = rand();
        x = r*cos(theta);
        y = r*sin(theta);
        oFile << x << '\t' << y << '\n';
    }
    oFile.close();
    return 0;
}
```



三角形内的均匀点



实践效果



多边形内均匀取点

- 使用圆内取点的思想：计算多边形的外包围盒，生成外包围盒(矩形)内的二维点，若点在多边形内，则退出；否则，继续探测。
- 将多边形剖分成三角形集合，然后调用三角形内均匀取点算法。
  - 按照面积为权重，选择某个三角形；
  - 生成该三角形内的随机点。
  - 思考：如何将多边形快速剖分成三角形？
    - Delaunay三角剖分

思考题

- 若某函数rand()以概率p(p≠0.5)返回数字0，以概率1-p返回数字1，如何利用该函数返回等概率的0和1？

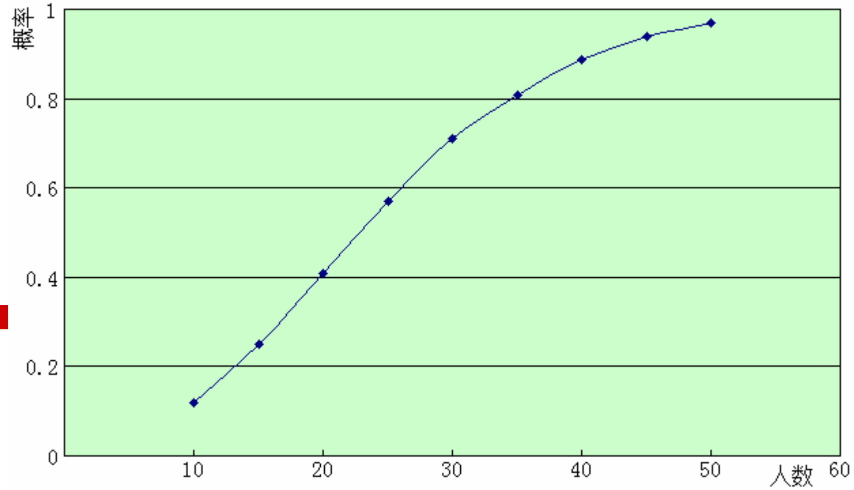
古典概型

- 将n个不同的球放入N(N≥n)个盒子中，假设盒子容量无限，求事件A={每个盒子至多有1个球}的概率。

解 $P(A)=\frac{P_N^n}{N^n}$

- 基本事件总数：
  - 第1个球，有N种放法；
  - 第2个球，有N种放法；
  - ……
  - 共：N^n种放法。
- 每个盒子至多放1个球的事件数：
  - 第1个球，有N种放法；
  - 第2个球，有N-1种放法；
  - 第3个球，有N-2种放法；
  - ……
  - 共：$N(N-1)(N-2)\cdots(N-n+1)=P_N^n$

实际问题



- 某班上有50位同学，至少有2人生日相同的概率是多少？

n	10	15	20	25	30	35	40	45	50
P	0.12	0.25	0.41	0.57	0.71	0.81	0.89	0.94	0.97

装箱问题

- 将12件正品和3件次品随机装在3个箱子中。每箱中恰有1件次品的概率是多少?

解

- 将15件产品装入3个箱子，每箱装5件，共有15!/(5!5!5!)种装法；
- 先把3件次品放入3个箱子，有3!种装法。对于这样的每一种装法，把其余12件产品装入3个箱子，每箱装4件，共有12!/(4!4!4!)种装法；
- P(A)= (3!*12!/(4!4!4!)) / (15!/(5!5!5!)) = 25/91

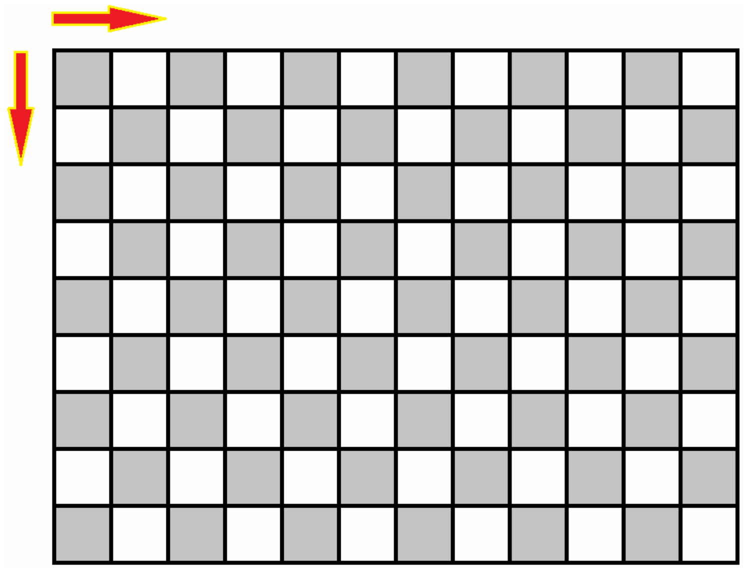
与组合数的关系

- 把n个物品分成k组，使得每组物品的个数分别为n1,n2…nk，(n=n1+n2+…+nk)，则不同的分组方法有$\frac{n!}{n_1!n_2!n_3!\cdots n_k!}$种。

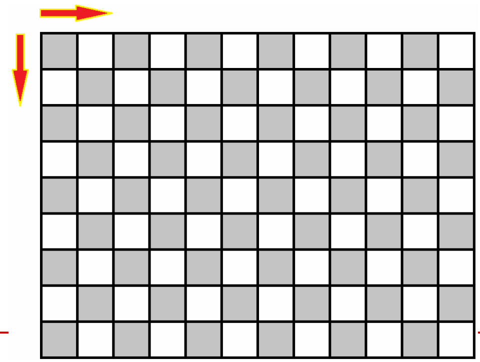
- 上述问题的简化版本，即n个物品分成2组，第一组m个，第二组n-m个，则分组方法有$\frac{n!}{m!(n-m)!}$，即：C_n^m。

走棋盘

- 给定m*n的矩阵，从左上角开始行走，每次只能朝右和下走，走到右下角，求一共有多少种走法？



分析



- 数学公式：一共要走m+n–2步，其中(m–1)步向右，(n-1)步向下。
  - 即：在m+n–2个步数中，选择m–1个作为“右行”，即：题目的解为：

$$N = C_{m+n-2}^{m-1} = C_{m+n-2}^{n-1}$$

  - 令dp(x,y)为当前位于(x,y)时有多少种可行路径，则：dp(x,y)=dp(x-1,y)+dp(x,y-1)

通过动态规划递推式，可以得到什么？

$$dp(x,y)=dp(x-1,y)+dp(x,y-1)$$

$$\xrightarrow{\text{增加“两个坐标值加和”}} dp(x+y,x,y)=dp(x+y-1,x-1,y)+dp(x+y-1,x,y-1)$$

$$\xrightarrow{\text{删除最后一维}} dp(x+y,x)=dp(x+y-1,x-1)+dp(x+y-1,x)$$

$$\xrightarrow{\text{令}t=x+y} dp(t,x)=dp(t-1,x-1)+dp(t-1,x)$$

$$\xrightarrow{\text{换个表达方式}} C_t^x=C_{t-1}^{x-1}+C_{t-1}^x$$

$$\xrightarrow{\text{令}n=t,m=x} C_n^m=C_{n-1}^{m-1}+C_{n-1}^m$$

分治法

- 给定实数x和整数n，求x^n。
- 分析：令pow(x,n)= x^n，如果能够求出y=pow(x,n/2)，只需要返回y*y即可，节省一半的时间。因此，可以递归下去。
  - 时间复杂度O(NlogN)
  - 需要考虑的：如果n是奇数呢?
  - 如果n是负数呢?

Code

```
class Solution {
public:
    double pow(double x, int n) {
        if (n < 0) return 1.0 / power(x, -n);
        else return power(x, n);
    }
private:
    double power(double x, int n) {
        if (n == 0) return 1;
        double v = power(x, n / 2);
        if (n % 2 == 0) return v * v;
        else return v * v * x;
    }
};
```

矩阵的乘法

- A为m×s阶的矩阵，B为s×n阶的矩阵，那么，C=A×B是m×n阶的矩阵，其中，

$$c_{ij} = \sum_{k=1}^{s} a_{ik} b_{kj}$$

- 根据定义来计算C=A×B，需要m*n*s次乘法。
  - 即：若A、B都是n阶方阵，C的计算时间复杂度为$O(n^3)$
  - 问：可否设计更快的算法？

分治

- 矩阵分块

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\begin{cases} C_{11} = A_{11}B_{11} + A_{12}B_{21} \\ C_{12} = A_{11}B_{12} + A_{12}B_{22} \\ C_{21} = A_{21}B_{11} + A_{22}B_{21} \\ C_{22} = A_{21}B_{12} + A_{22}B_{22} \end{cases}$$

- 按照定义：计算n/2阶矩阵的乘积：$O(n^3/8)$
- 这里需要计算8个：总时间复杂度：$O(n^3)$
  - 没有任何效果。

Strassen矩阵乘法：由8到7

- 目标

$$
\begin{cases}
C_{11} = A_{11}B_{11} + A_{12}B_{21} \\
C_{12} = A_{11}B_{12} + A_{12}B_{22} \\
C_{21} = A_{21}B_{11} + A_{22}B_{21} \\
C_{22} = A_{21}B_{12} + A_{22}B_{22}
\end{cases}
$$

$$
\begin{cases}
P = (A_{11} + A_{22})(B_{11} + B_{22}) \\
Q = (A_{21} + A_{22})B_{11} \\
R = A_{11}(B_{12} - B_{22}) \\
S = A_{22}(B_{21} - B_{11}) \\
T = (A_{11} + A_{12})B_{22} \\
U = (A_{21} - A_{11})(B_{11} + B_{12}) \\
V = (A_{12} - A_{22})(B_{21} + B_{22})
\end{cases}
\Longrightarrow
\begin{cases}
C_{11} = P + S - T + V \\
C_{12} = R + T \\
C_{21} = Q + S \\
C_{22} = P + S - Q + U
\end{cases}
$$

说明

- 时间复杂度降为$O(n^{\log 7})=O(n^{2.81})$
  - 理论意义：由定义出发直接得出的算法未必是最好的。
- Hopcroft与Kerr已经证明，两个2×2矩阵相乘必须要用7次乘法，如果需要进一步改进，应考虑3×3、4×4或者更高阶数的分块子矩阵——或者采用其他设计策略。
- 当n很大时，实际效果比直接定义求解的$O(n^3)$好。
- 根据矩阵乘法的定义$c_{ij}=\sum_{k=1}^{s} a_{ik}b_{kj}$可知，Cij只与A的第i行、B的第j列相关，在实践中若遇到大矩阵，应考虑并行计算。

思考

- 将矩阵分治乘法的思想，用于两个大整数的乘法呢？
- 根据定义，两个大整数A、B相乘，应该遍历B从低到高的数字，依次与大整数A相乘，最后将这些值相加。
  - 时间复杂度$O(n^2)$。
  - 可否将A、B写成两个规模小一半的整数A1,A2,B1,B2，然后计算它们的积呢？

大整数乘法

- 取大整数x和y的长度较大者的一般，记为k，则有：

$$\begin{cases} x = x_1 M^k + x_0 \\ y = y_1 M^k + y_0 \end{cases}$$

$$\Rightarrow xy = (x_1 M^k + x_0)(y_1 M^k + y_0)$$

$$= x_1 y_1 M^{2k} + (x_1 y_0 + x_0 y_1) M^k + x_0 y_0$$

- 计算长度为n/2的两个数的积，需要乘法次数为$O(n^2/4)$，而上面的式子需要4次乘法，总时间复杂度为$O(n^2)$，没有效果。因此，需要考虑改进。

大整数乘法：Karatsuba算法

- 事实上：$\begin{cases} x = x_1 M^k + x_0 \\ y = y_1 M^k + y_0 \end{cases}$

$$\Rightarrow xy = (x_1 M^k + x_0)(y_1 M^k + y_0)$$

$$xy = x_1 y_1 M^{2k} + (x_1 y_0 + x_0 y_1) M^k + x_0 y_0$$

$$= x_1 y_1 M^{2k} + ((x_1 + x_0)(y_1 + y_0) - x_1 y_1 + x_0 y_0) M^k + x_0 y_0$$

- 上式只需要3次乘法(配合若干次加法和移位)即可完成，时间复杂度为O($n^{\log 3}$)=O($n^{1.585}$)。

机器学习若干概念

- 交叉验证
- 泛化能力
- 监督学习
- 无监督学习
- 强化学习

机器学习算法的分类

- 监督
  - K近邻
  - 回归
  - SVM
  - 决策树
  - 朴素贝叶斯
  - BP神经网络
- 非监督
  - 聚类
  - Apriori
  - FP-growth

交叉验证

- 交叉验证(Cross-validation)也称为交叉比对，主要用于建模应用中。在给定的建模样本中，拿出大部分样本进行建模型，留小部分样本用刚建立的模型进行预报，并求这小部分样本的预报误差，记录它们的平方加和。这个过程一直进行，直到所有的样本都被预报了一次而且仅被预报一次。把每个样本的预报误差平方加和，称为PRESS(predicted Error Sum of Squares)。
- 交叉验证是常用的精度测试方法，其目的是为了得到可靠稳定的模型。例如10折交叉验证(10-fold cross validation)，将数据集分成十份，轮流将其中9份做训练1份做测试，10次的结果的均值作为对算法精度的估计，一般还需要进行多次10折交叉验证求均值，例如：10次10折交叉验证，以求更精确一点。

交叉验证的形式

- Holdout 验证
  - 通常来说，Holdout 验证并非一种交叉验证，因为数据并没有交叉使用。随机从最初的样本中选出部分，形成交叉验证数据，而剩余的就当做训练数据。一般来说，少于原本样本三分之一的数据被选做验证数据。
- K-fold cross-validation
  - K折交叉验证，初始采样分割成K个子样本，一个单独的子样本被保留作为验证模型的数据，其他K-1个样本用来训练。交叉验证重复K次，每个子样本验证一次，平均K次的结果或者使用其它结合方式，最终得到一个单一估测。这个方法的优势在于，同时重复运用随机产生的子样本进行训练和验证，每次的结果验证一次，10折交叉验证是最常用的。
- 留一验证
  - 意指只使用原本样本中的一项来当做验证资料，而剩余的则留下来当做训练资料。这个步骤一直持续到每个样本都被当做一次验证资料。事实上，这等同于 K-fold 交叉验证是一样的，其中K为原本样本个数。

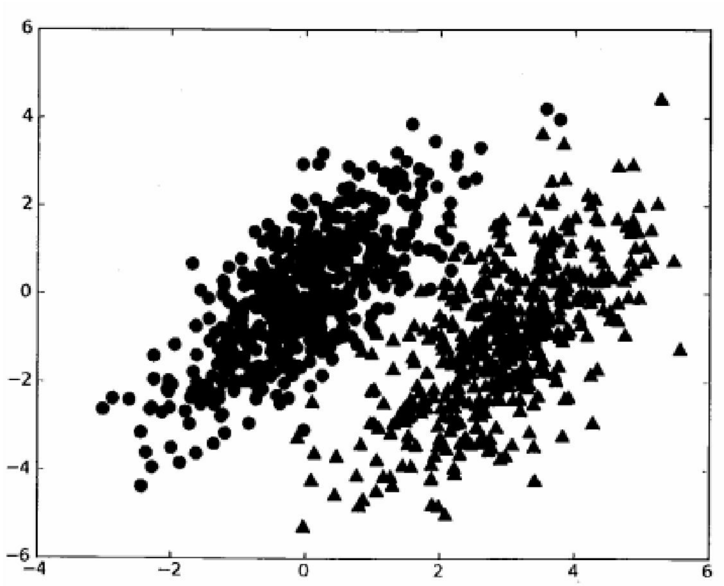
泛化能力

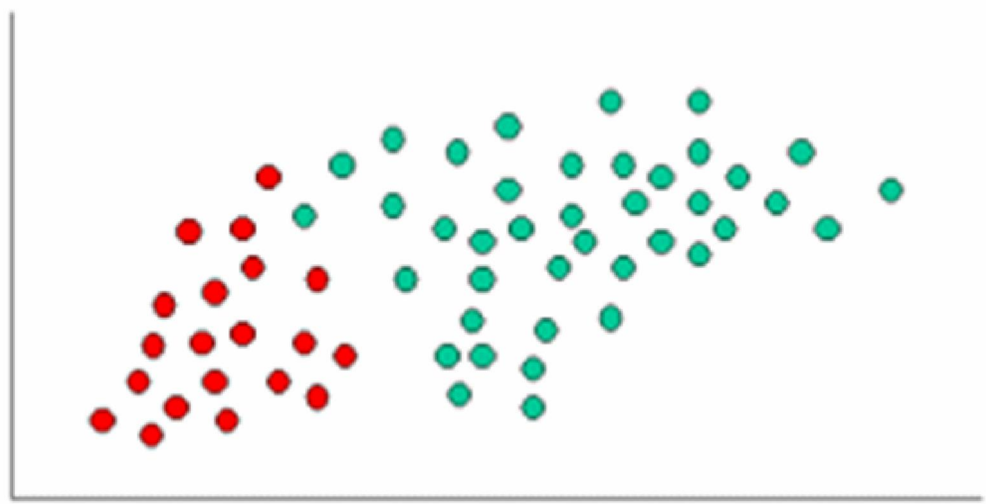
- 概括地说，所谓泛化能力（generalization ability）是指机器学习算法对新鲜样本的适应能力。学习的目的是学到隐含在数据对背后的规律，对具有同一规律的学习集以外的数据，经过训练的算法也能给出合适的输出，该能力称为泛化能力。
- 通常期望经训练样本训练的算法具有较强的泛化能力，也就是对新输入给出合理响应的能力。应当指出并非训练的次数越多越能得到正确的输入输出映射关系。算法的性能主要用它的泛化能力来衡量。

从下面几个问题入手机器学习

- k近邻
- 向量距离
- 聚类
- 线性回归
- 朴素贝叶斯

k近邻分类(属于有监督学习)





相似度/距离计算方法总结

- 闵可夫斯基距离Minkowski/欧式距离

$$dist(X,Y)=\left(\sum_{i=1}^{n}|x_i-y_i|^p\right)^{\frac{1}{p}}$$

- 杰卡德相似系数(Jaccard)

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}$$

- 余弦相似度(cosine similarity)

$$\cos(\theta)=\frac{a^T b}{|a|\cdot|b|}$$

- Pearson相似系数

$$\rho_{XY}=\frac{\operatorname{cov}(X,Y)}{\sigma_X\sigma_Y}=\frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X\sigma_Y}=\frac{\sum_{i=1}^{n}(X_i-\mu_X)(Y_i-\mu_Y)}{\sqrt{\sum_{i=1}^{n}(X_i-\mu_X)^2}\sqrt{\sum_{i=1}^{n}(Y_i-\mu_Y)^2}}$$

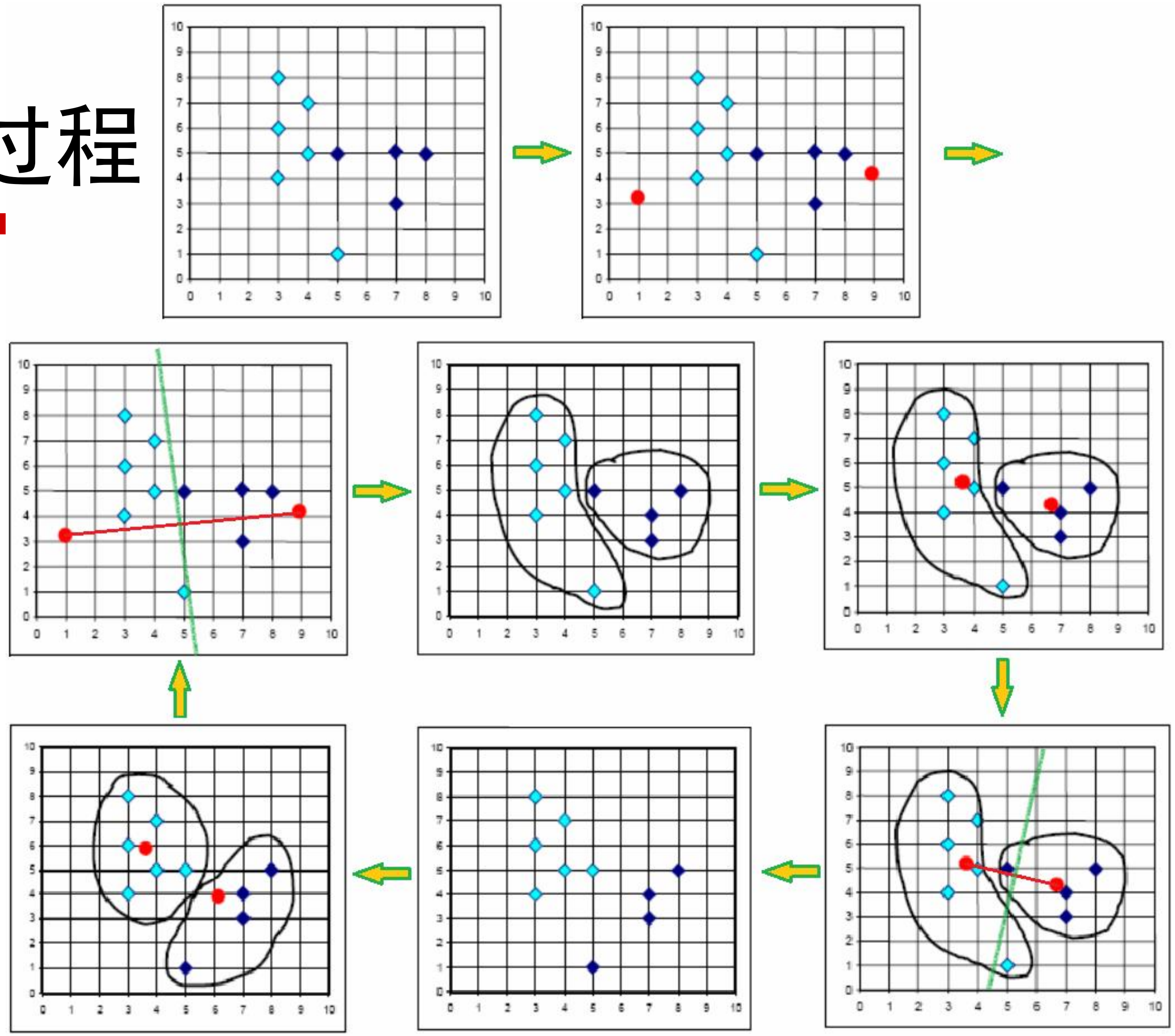
- 相对熵(K-L距离)

$$D(p\|q)=\sum_x p(x)\log\frac{p(x)}{q(x)}=E_{p(x)}\log\frac{p(x)}{q(x)}$$

k-均值聚类(属于无监督学习)

- 创建k个点作为起始质心(如：随机选择起始质心)
- 当任意一个点的簇分配结果发生改变时
  - 对数据集中的每个数据点
    - 对每个质心
      - 计算质心与数据点之间的距离
    - 将数据点分配到距其最近的簇
  - 对每个簇，计算簇中所有点的均值并作为质心

K-means过程

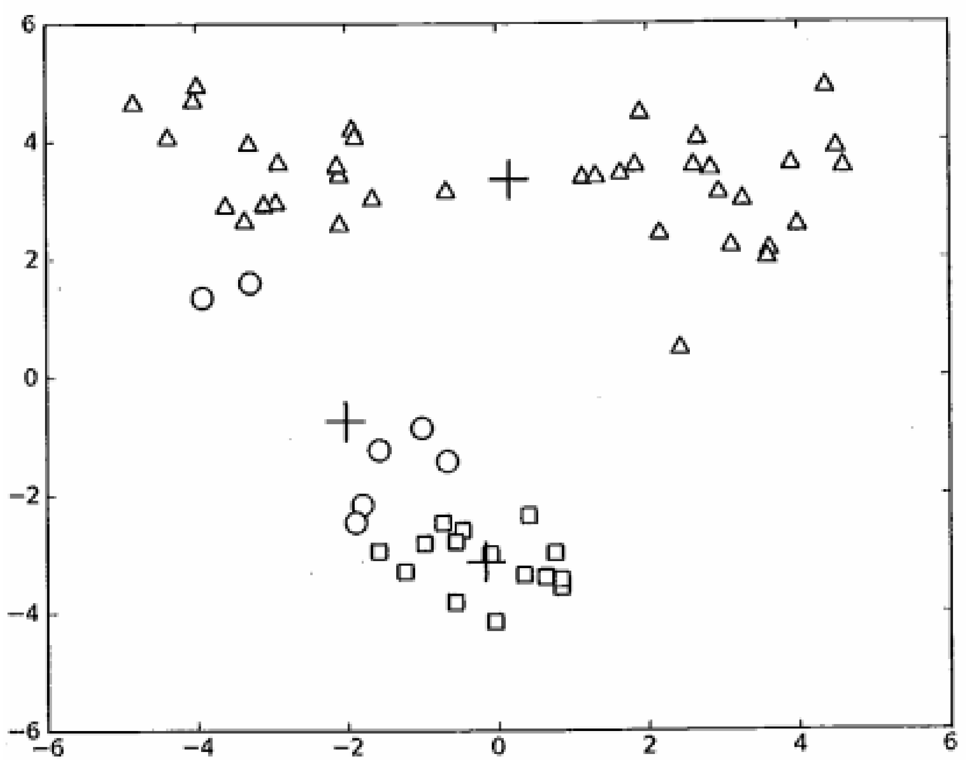


对K-Means的思考

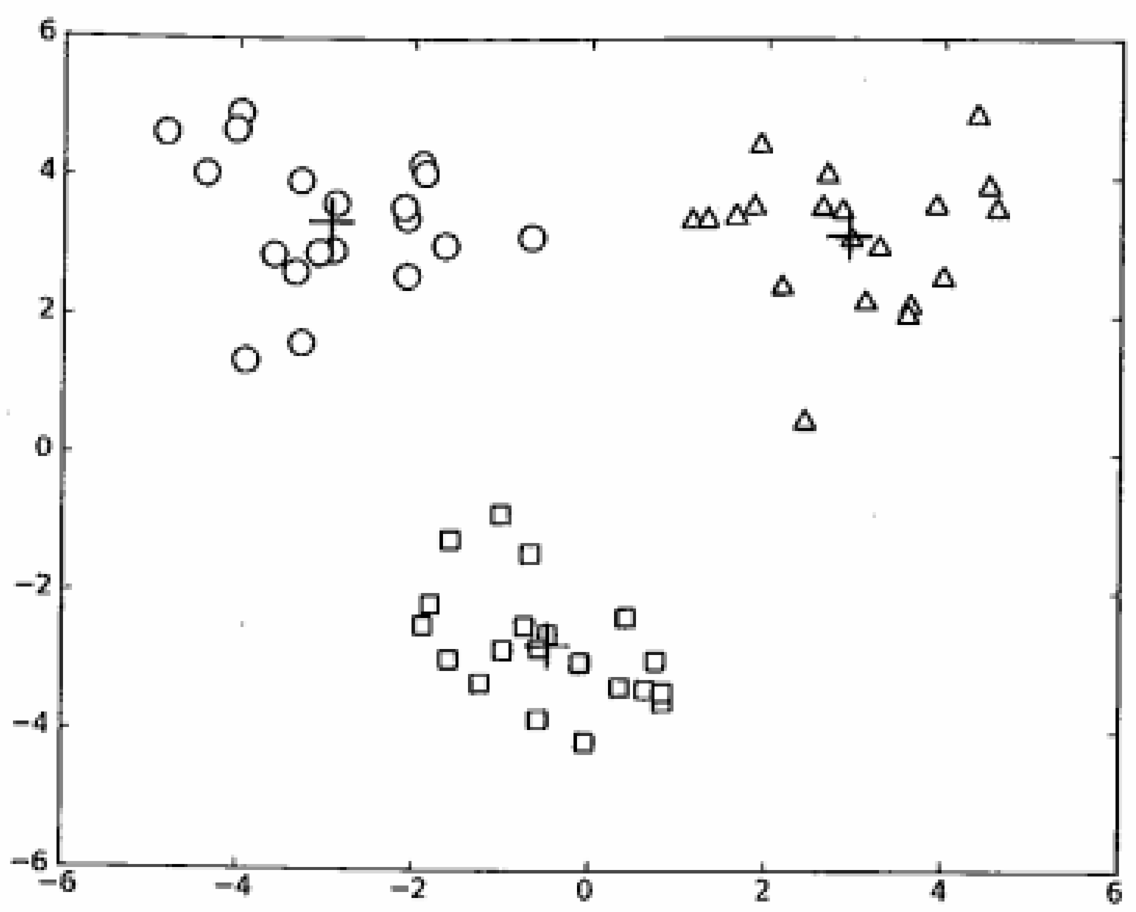
- K-Means将簇中所有点的均值作为新质心，若簇中含有异常点，将导致均值偏离严重。以一维数据为例：
  - 数组1、2、3、4、100的均值为22，显然距离“大多数”数据1、2、3、4比较远
  - 改成求数组的中位数3，在该实例中更为稳妥。
  - 这种聚类方式即K-Mediods聚类(K中值距离)
- 点的簇分配结果发生改变的标准如何判断？
  - 实践中可以选择误差的平方和最小
- 初值的选择，对聚类结果有影响吗？
  - 如何避免？

利用SSE进行聚类后处理

- SSE: Sum of Squared Error 误差平方和

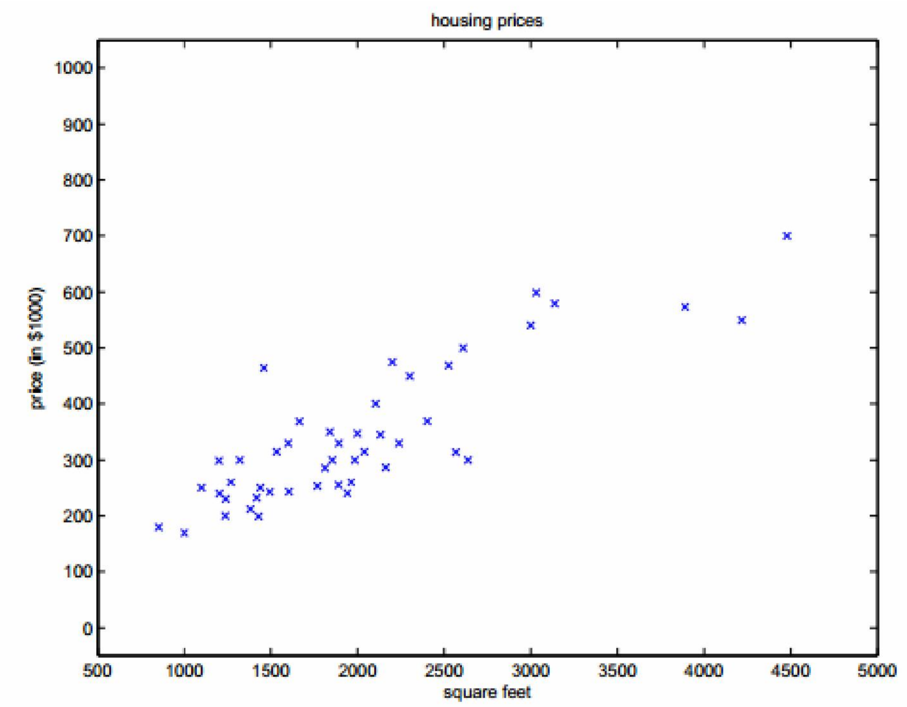


二分k-均值聚类后的结果

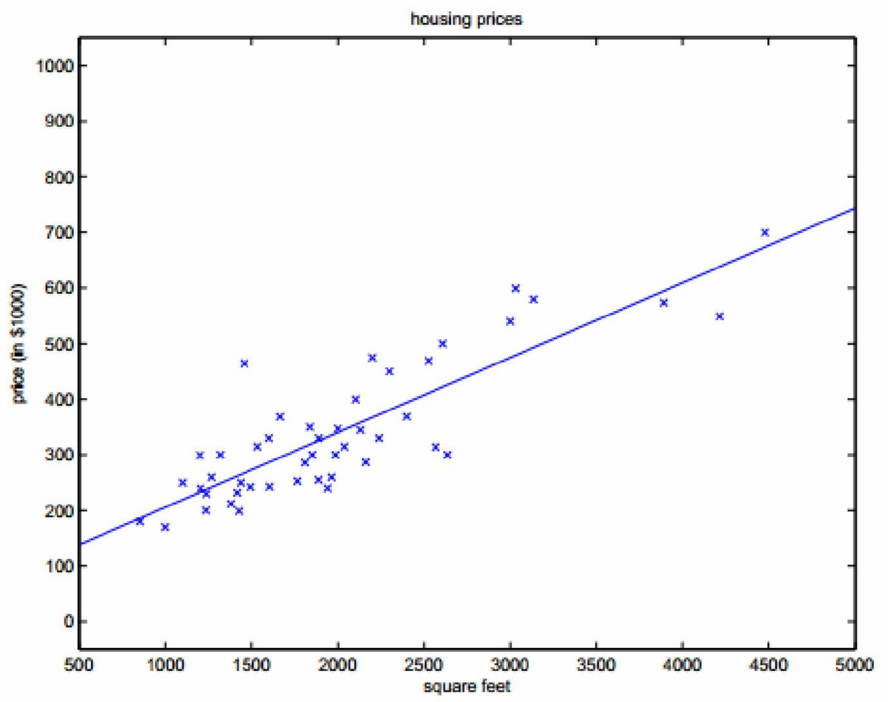


线性回归

- y=ax+b



housing prices
price (in $1000)
square feet



housing prices
price (in $1000)
square feet

多个变量的情形

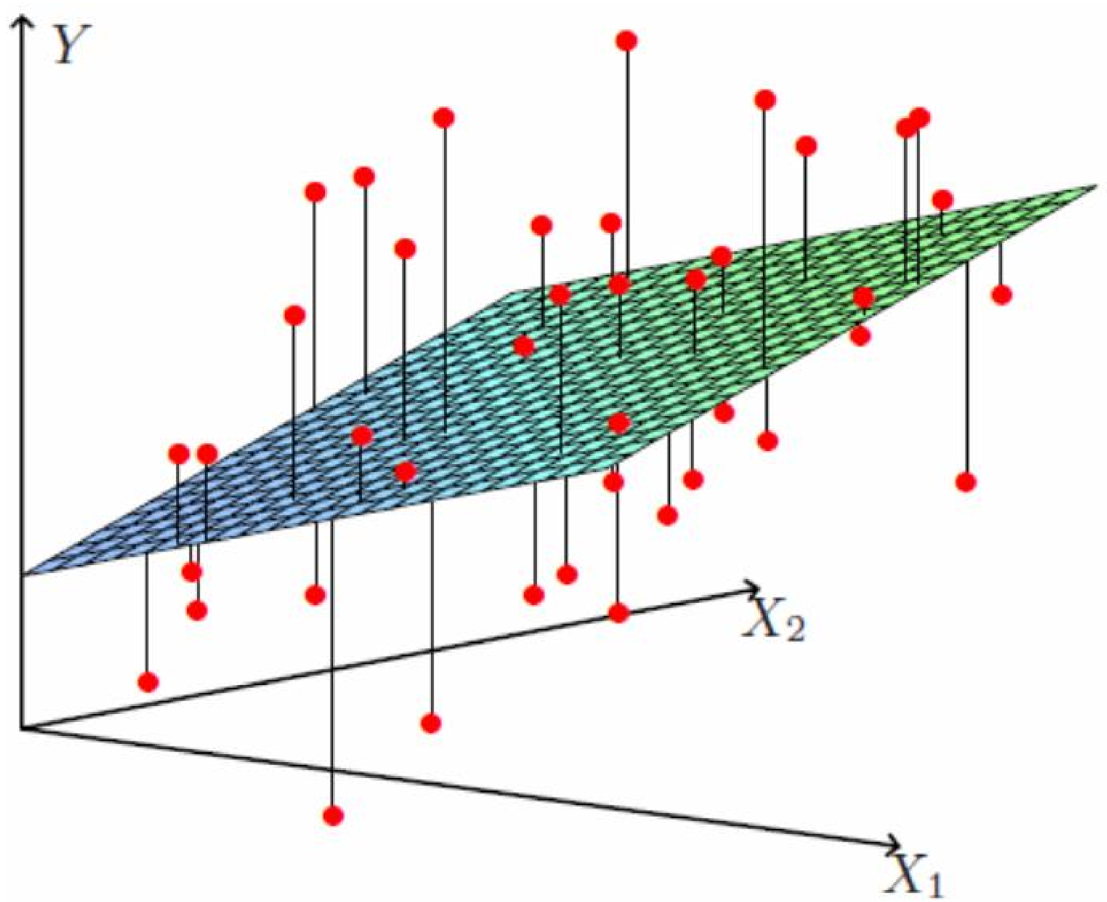
- 考虑两个变量

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

多个变量的情形



最小二乘的目标函数

- m为样本个数，则一个比较"符合常理"的误差函数为：

$$J(\theta)=\frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$$

- 思考：如何解释和定义"符合常理"?

使用极大似然估计解释最小二乘

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

the $\epsilon^{(i)}$ are distributed IID (independently and identically distributed) according to a Gaussian distribution (also called a Normal distribution) with mean zero and some variance σ^2

似然函数

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}\right)$$

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}\right) \end{aligned}$$

对数似然

$$
\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2
\end{aligned}
$$

计算极大似然函数的最优解

$$\boxed{\nabla_\theta(X\theta) = X^T}$$

$$\begin{aligned}\frac{1}{2}(X\theta-\vec{y})^T(X\theta-\vec{y}) &= \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2 \\ &= J(\theta)\end{aligned}$$

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta-\vec{y})^T(X\theta-\vec{y}) \\ &= \frac{1}{2}\nabla_\theta\left(\theta^T X^T X\theta - \theta^T X^T\vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right) \\ &= \frac{1}{2}\left(X^T X\theta + X^T X\theta - 2X^T\vec{y}\right) \\ &= X^T X\theta - X^T\vec{y}\end{aligned}$$

“简便”方法记忆结论

$$X\theta = y$$

$$X^T X\theta = X^T y$$

$$\theta = \left(X^T X\right)^{-1} X^T y$$

最小二乘意义下的参数最优解

- 参数的解析式

$$\theta = \left(X^T X\right)^{-1} X^T y$$

- 若X^TX不可逆，上式不可使用

$$\theta = \left(X^T X + \lambda I\right)^{-1} X^T y$$

加入λ扰动后

- X^TX半正定：对于任意的非零向量u

$$uX^TXu = (Xu)^T Xu \xrightarrow{\text{令}v=Xu} v^Tv \geq 0$$

- 所以，对于任意的实数λ>0，$X^TX + \lambda I$ 正定，从而可逆。保证回归公式一定有意义。

$$\theta = \left(X^TX + \lambda I\right)^{-1} X^T y$$

对线性回归的思考

- 若目标y与观测向量X不是线性关系，怎么处理？
- 局部线性回归
  - 非参数方法
- 广义线性回归
  - 对数线性回归
  - Logistic回归

概率

- 条件概率：

$$P(A|B)=\frac{P(AB)}{P(B)}$$

- 全概率公式：

$$P(A)=\sum_i P(A|B_i)P(B_i)$$

- 贝叶斯(Bayes)公式：

$$P(B_i|A)=\frac{P(A|B_i)P(B_i)}{\sum_j P(A|B_j)P(B_j)}$$

贝叶斯公式的应用

- 8支步枪中有5支已校准过，3支未校准。一名射手用校准过的枪射击，中靶概率为0.8；用未校准的枪射击，中靶概率为0.3；现从8支枪中随机取一支射击，结果中靶。求该枪是已校准过的概率。

- 解：

$$P(G=1)=\frac{5}{8} \qquad P(G=0)=\frac{3}{8}$$

$$P(A=1|G=1)=0.8 \qquad P(A=0|G=1)=0.2$$

$$P(A=1|G=0)=0.3 \qquad P(A=0|G=0)=0.7$$

$$P(G=1|A=1)=?$$

$$P(G=1|A=1)=\frac{P(A=1|G=1)P(G=1)}{\sum_{i\in G}P(A=1|G=i)P(G=i)}=\frac{0.8\times\frac{5}{8}}{0.8\times\frac{5}{8}+0.3\times\frac{3}{8}}=0.8163$$

贝叶斯准则

- 条件概率公式
  - $P(x|y) = P(x,y) / P(y) \rightarrow P(x,y) = P(x|y) * P(y)$
  - $P(y|x) = P(x,y) / P(x) \rightarrow P(x,y) = P(y|x) * P(x)$
  - 则 $P(x|y) * P(y) = P(y|x) * P(x)$
- 从而： $P(x|y) = P(y|x) * P(x) / P(y)$

- 分类原则：在给定的条件下，哪种分类发生的概率大，则属于那种分类。

朴素贝叶斯的假设

- 一个特征出现的概率，与其他特征(条件)独立(特征独立性)
  - 其实是：对于给定分类的条件下，特征独立
- 每个特征同等重要(特征均衡性)

以文本分类为例

- 样本：1000封邮件，每个邮件被标记为垃圾邮件或者非垃圾邮件
- 分类目标：给定第1001封邮件，确定它是垃圾邮件还是非垃圾邮件
- 方法：朴素贝叶斯

分析

- 类别c：垃圾邮件c_1，非垃圾邮件c_2
- 词汇表，两种建立方法：
  - 使用现成的单词词典；
  - 将所有邮件中出现的单词都统计出来，得到词典。
  - 记单词数目为N
- 将每个邮件m映射成维度为N的向量**x**
  - 若单词w_i在邮件m中出现过，则$x_i=1$，否则，$x_i=0$。即邮件的向量化：$m \rightarrow (x_1,x_2 \ldots\ldots x_N)$
- 贝叶斯公式：$P(c|\mathbf{x})=P(\mathbf{x}|c)*P(c) / P(\mathbf{x})$
  - $P(c_1|\mathbf{x})=P(\mathbf{x}|c_1)*P(c_1) / P(\mathbf{x})$
  - $P(c_2|\mathbf{x})=P(\mathbf{x}|c_2)*P(c_2) / P(\mathbf{x})$
    - 注意这里**x**是向量

分解

- $P(c|\mathbf{x})=P(\mathbf{x}|c)*P(c) / P(\mathbf{x})$
- $P(\mathbf{x}|c)=P(x_1,x_2\ldots x_N|c)=P(x_1|c)*P(x_2|c)\ldots P(x_N|c)$
  - 特征条件独立假设
- $P(\mathbf{x})=P(x_1,x_2\ldots x_N)=P(x_1)*P(x_2)\ldots P(x_N)$
  - 特征独立假设
- 带入公式：$P(c|\mathbf{x})=P(\mathbf{x}|c)*P(c) / P(\mathbf{x})$

- 等式右侧各项的含义：
  - $P(x_i|c_j)$：在c_j(此题目，cj要么为垃圾邮件1，要么为非垃圾邮件2)的前提下，第i个单词x_i出现的概率
  - $P(x_i)$：在所有样本中，单词x_i出现的概率
  - $P(c_j)$：在所有样本中，邮件类别c_j出现的概率

拉普拉斯平滑

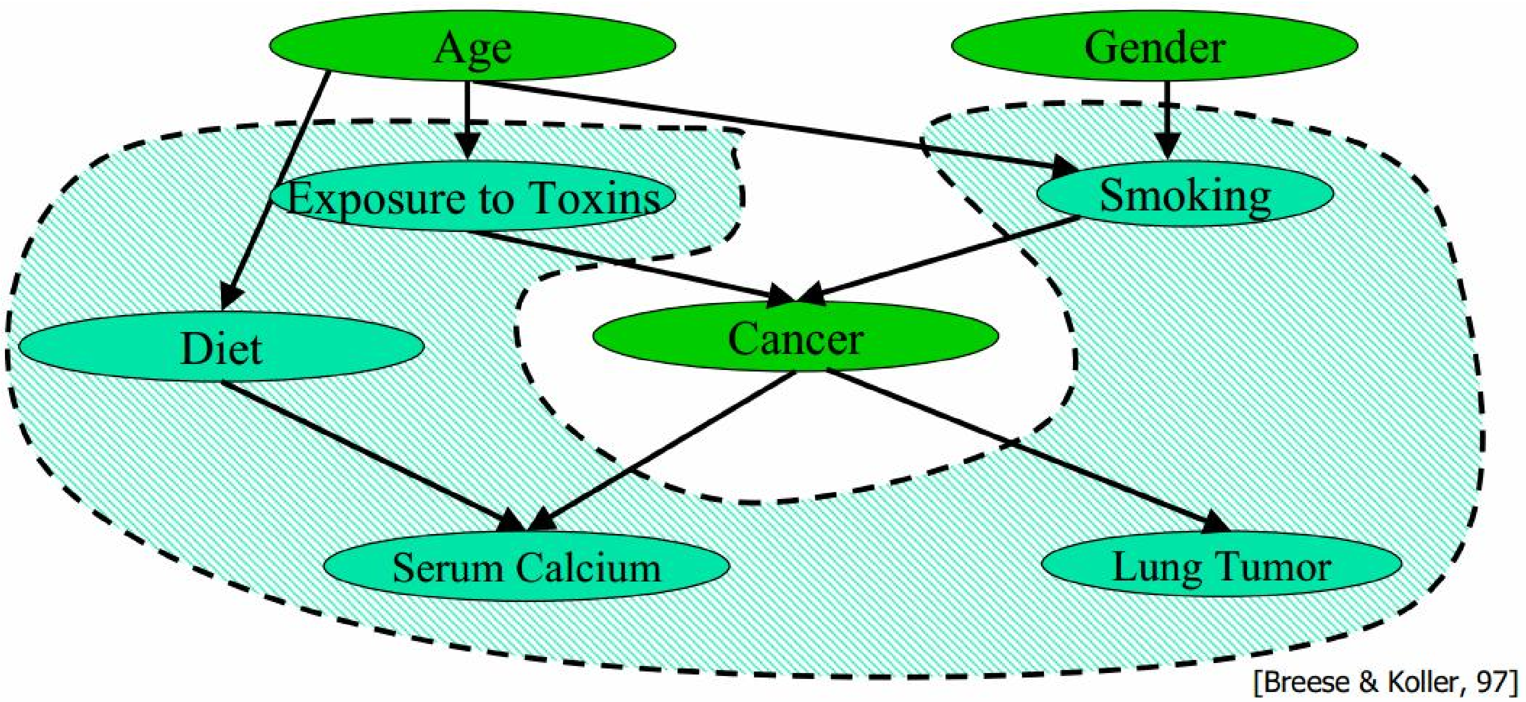
- $p(x_1|c_1)$是指的:在垃圾邮件c_1这个类别中，单词x1出现的概率。
  - x_1是待考察的邮件中的某个单词
- 定义符号
  - n_1：在所有垃圾邮件中单词x_1出现的次数。如果x_1没有出现过，则$n_1=0$。
  - n：属于c_1类的所有文档的出现过的单词总数目。
- 得到公式：$$p(x_1|c_1)=\frac{n_1}{n}$$
- 拉普拉斯平滑：$$p(x_1|c_1)=\frac{n_1+1}{n+N}$$
  - 其中，N是所有单词的数目。修正分母是为了保证概率和为1
- 同理，以同样的平滑方案处理$p(x_1)$

对朴素贝叶斯的思考

- 遇到生词怎么办？
  - 拉普拉斯平滑
- 编程的限制：小数乘积怎么办？
- 问题：一个词在样本中出现多次，和一个词在样本中出现一次，形成的词向量相同
  - 由0/1改成计数
- 如何判断两个文档的距离
  - 夹角余弦
- 如何判定该分类器的正确率
  - 样本中：K个生成分类器，1000-K个作为测试集
  - 交叉验证
- 若对象特征之间不独立，会演化成何种形式？

贝叶斯网络

背景知识：Serum Calcium(血清钙浓度)高于2.75mmo1/L即为高钙血症。许多恶性肿瘤可并发高钙血症。



阴影部分的结点集合，称为Cancer的“马尔科夫毯” (Markov Blanket)

参考文献

- Prof. Andrew Ng, Machine Learning, Stanford University
- Pattern Recognition and Machine Learning Chapter 8, M. Jordan, J. Kleinberg, ect, 2006
- http://www.cnblogs.com/TenosDoIt/p/4025221.html

我们在这里

- 更多算法面试题在 7 七月算法
  - http://www.julyedu.com/
    - 免费视频
    - 直播课程
    - 问答社区
- contact us：微博
  - @研究者July
  - @七月问答
  - @邹博_机器学习

感谢大家！

恳请大家批评指正！