Logistic

解决问题

1. 两分类
2. 线性可回归问题

复习一下回归的定义 : 加入现在又一些数据点，我们用一条直线对这些点做拟合，期望得到最佳拟合直线，拟合的过程叫做回归

在拟合问题中，主要的思想在于我们拟合良好的数据预测回归公式

Logistic就是经典的一个拟合的实例，这是一个标准的二值型输出分类器 :

我们对于每一个数据集的特征都创建一个对应的回归系数，每一个特征对最后的分类的结果都存在贡献，我们目的就是对于拟合一组这样的系数使得我们对于分类的结果在训练集上很优秀，并在之后的数据集上泛化的很好

$$z = w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

在这里我们的X是分类器的输入数据，W是迭代找的最佳系数

Gradient descent

高数中我们都学习如何快速的求一个函数的最优值，我们通常使用的一个重要的工具叫做梯度

我们可以理解梯度就是我们朝最优值变化的一个趋势，沿这个趋势可以最快的到达目的地(当然这是微分上的概念，实际上我们还需要考虑步长(震荡)和局部最优的情况)

$$x_{k+1} = x_k + \rho_k \nabla^k$$

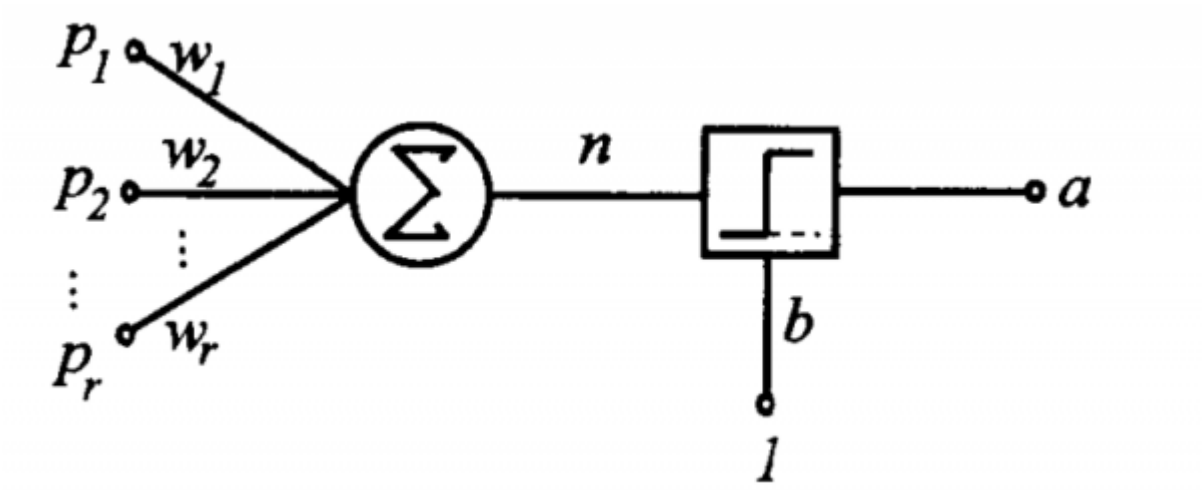
这里的公式就渗透了我们的利用梯度求解最优值的迭代过程

问题

1. 但是这里会引出来一个之后会显现的很重要的问题 : 步长(步长太短算法收敛的很慢，步长太大，算法容易发散并且会出现频繁的在最有值附近震荡的情况)，解决办法主要办法就是动态步长选取,这是后话了，之后会讲到
2. 迭代次数(迭代多少次可以处于良好的误差范围中)

如何利用梯度下降求解Logistic算法

1. 首先引入感知器神经元



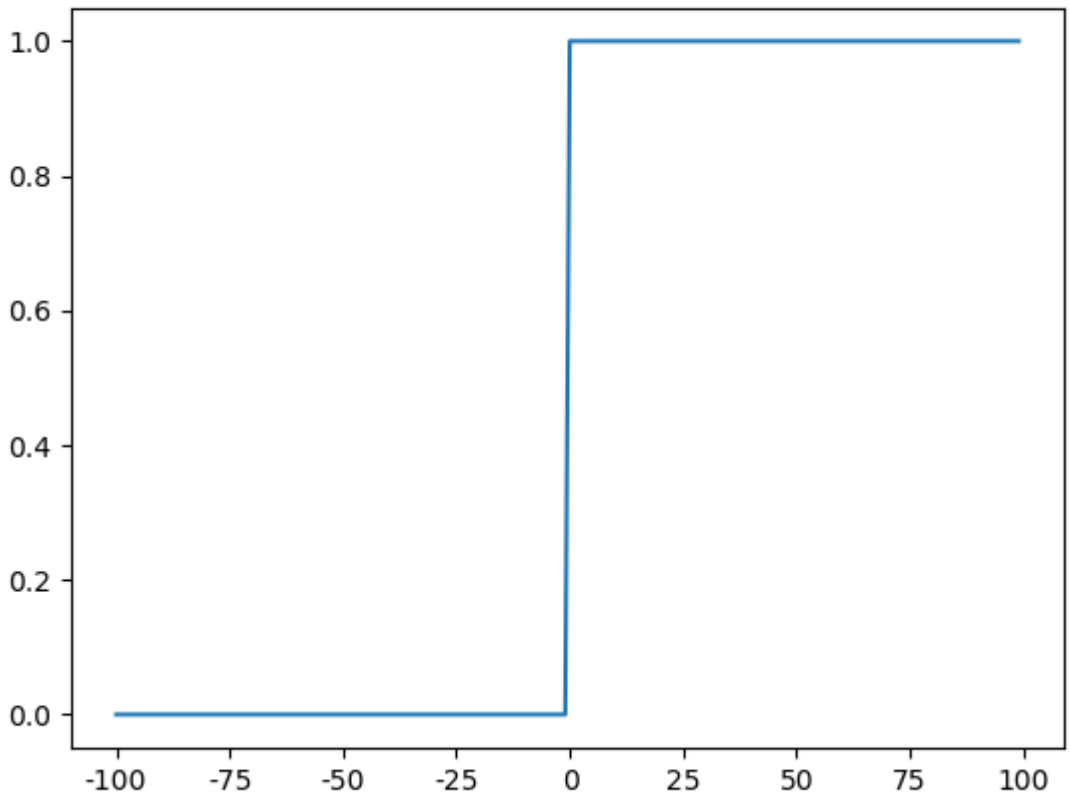
感知器神经元是一种现行分类器，包含一个算法框架和一个激活函数，在这里我们的算法框架就是我们的公式

$$z = w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

激活函数需要讲一下 ：

对于这个现行分类的问题，我们需要的目的是接收出入然后预测输出，因为是二值型的数值分类器我们要求是输出0,1代表两种不同的分类方式

一般来说我们对于这种输出0,1的函数(激活函数)最有印象的就是



这是一个0,1阶跃函数，我们又称是hardlim函数，针对我们的算法框架的输出，该函数可以生成对应的份额里标准，这好像就是我们的需要的答案了

算法的目标：

训练我们的参数，是的经过参数的迭代求出的z在经过我们的激活函数之后可以误差很小的将数据集正确的区分开来，之

后我们会更形象的看出来

这样的化，我们聚会很清楚的发现，这个问题就会变成基础的最优化问题，只不过这里的优化目标比较复杂而已，对于最优化问题，我们就可以考虑使用梯度的方法来解决

但是在这里，我们的梯度是什么?好像很难找到适用于该题目的梯度的概念

我们可以将梯度形象的理解成是我们当前的分类拟合出来的参数W针对我们的数据集的拟合出来的误差，我们的目标无

非就是要让误差下降

算法的思路有了：

1. 收集数据(特征， label标签(目标))
2. 确定迭代次数，和迭代的步长
3. 确定初始化的W参数向量
4. 迭代公式

   $$f(x) = w^T x$$

   $$error = label - hardlim(f(x))$$

   $$w_{k+1} = w_k + \alpha x^T error$$

   训练参数向量

Sigmod or H ardlim

Hardlim函数：可以看出来在0的附近是非常的剧烈的变化的，是阶跃的

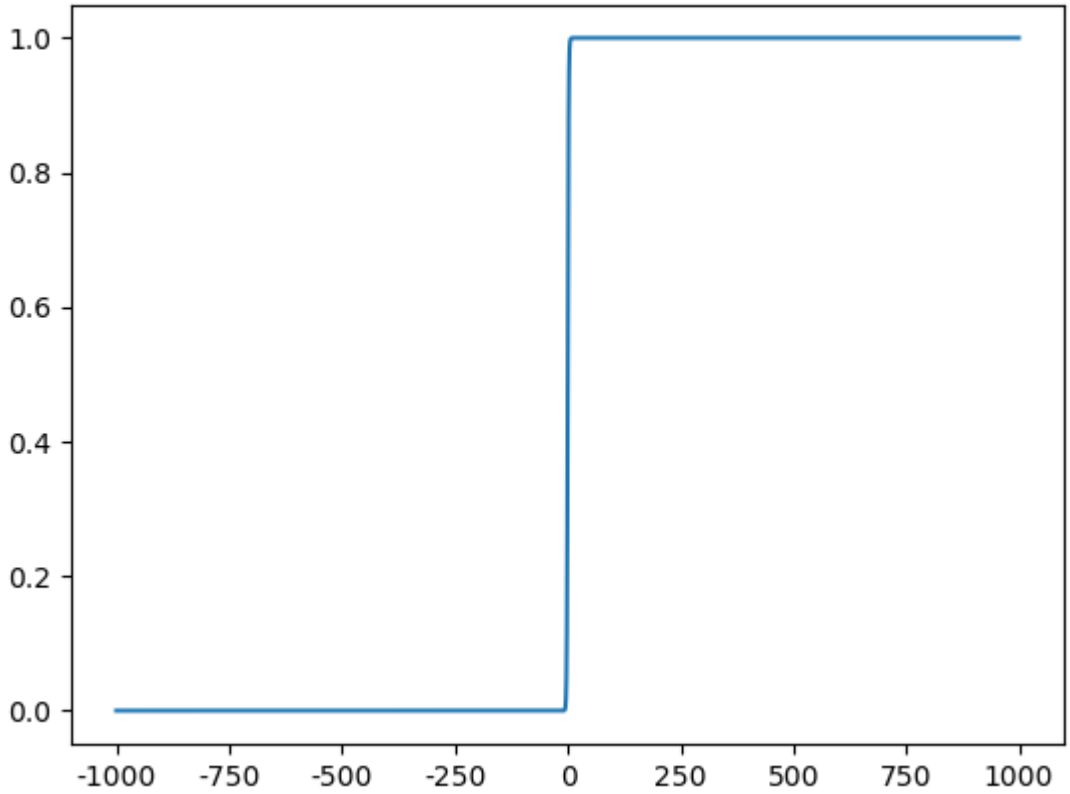
Sigmod函数 :

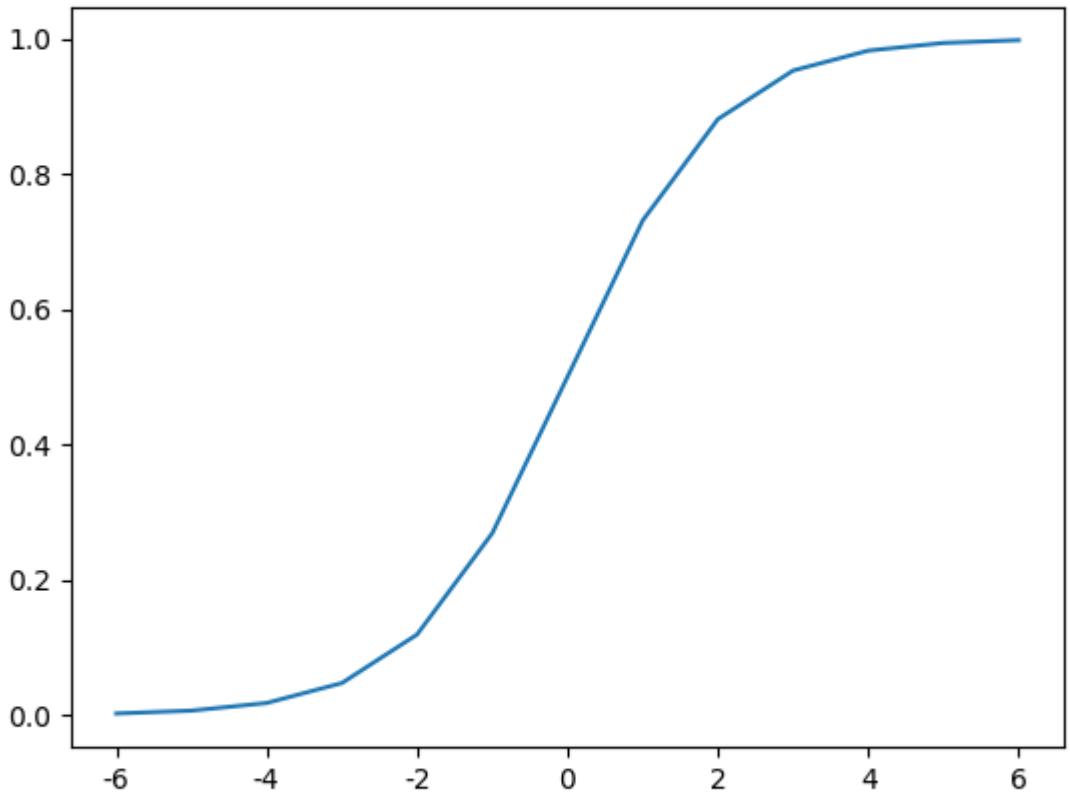
sigmod函数相对于hardlim更稳定的原因在于引入了概率，这样可以为算法提供一个平稳的分类的便捷，保证在迭代次数增大的同时我们的算法也会收敛的很好，更稳定

但是这样类似的曲线有很多，为什么一定要用sigmod函数呢

1. 我们之前应该还记的，我们对应这个最优化问题，使用功能的梯度的表达是是$w^T x$

   这个表达式的含义在于我们的对w向量和x向量求内积，这个表示我们的算法的计算结果，这个计算结果之后要带入我们的激活函数中去进行分类

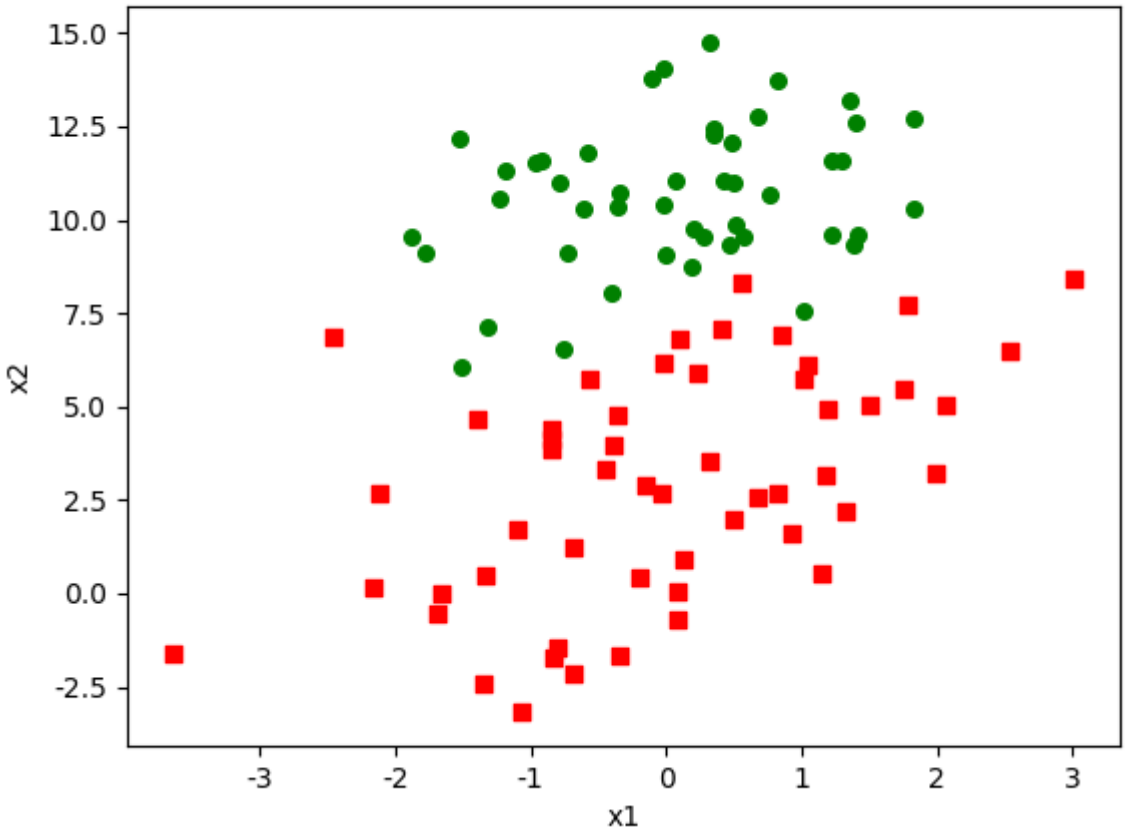

1.0
0.8
0.6
0.4
0.2
0.0
-1000
-750
-500
-250
0
250
500
750
1000


1.0
0.8
0.6
0.4
0.2
0.0
-6
-4
-2
0
2
4
6

在logistic和 Sigmod都讲完之后演示两张图表示出回归的效果的差距(同迭代次数下)

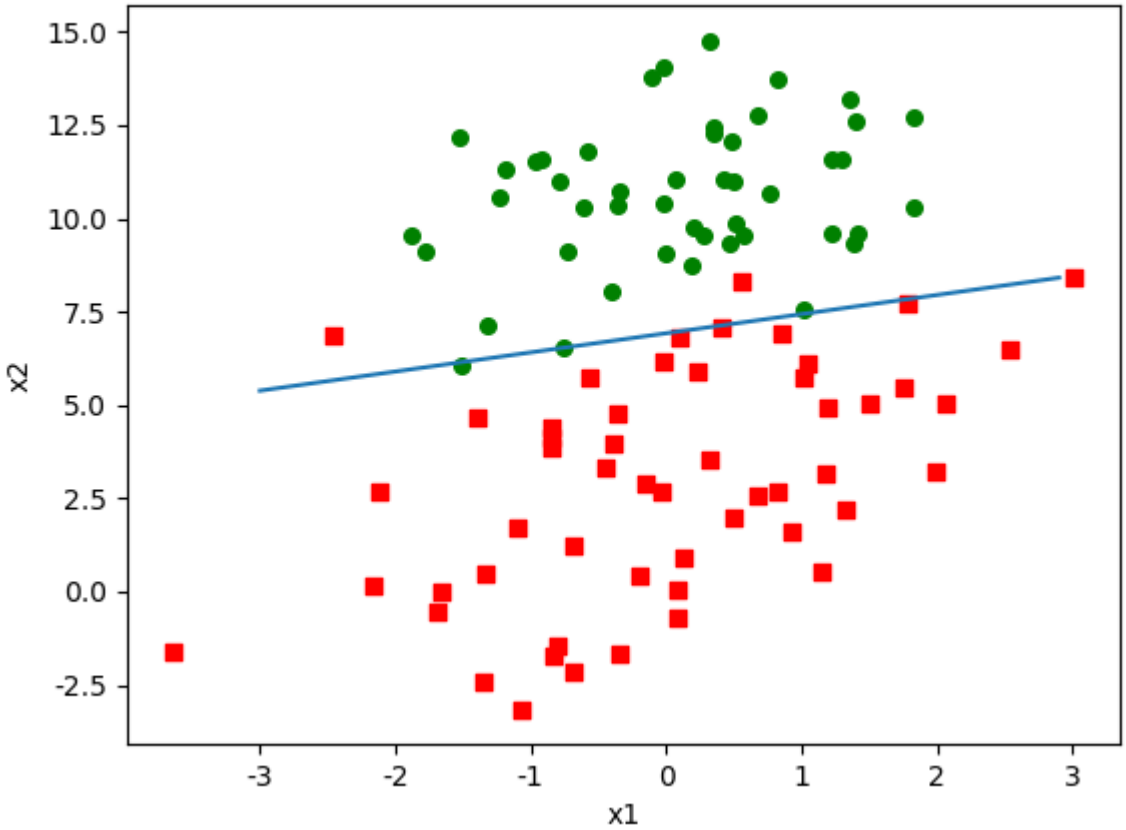
硬限符函数不是一个连续的函数，在一个很小的区间内，函数变化会非常的剧烈，目标函数会发生很剧烈的震荡，决策的便捷汇编的非常的狭窄，严重的而影响我们额分类精度和评估的结果

为了方便直观的说明问题，这里的数据有两个特征，100的数据量，我们的目的就是拟合出来一个参数向量可以将我们的两中类别尽可能的分类开



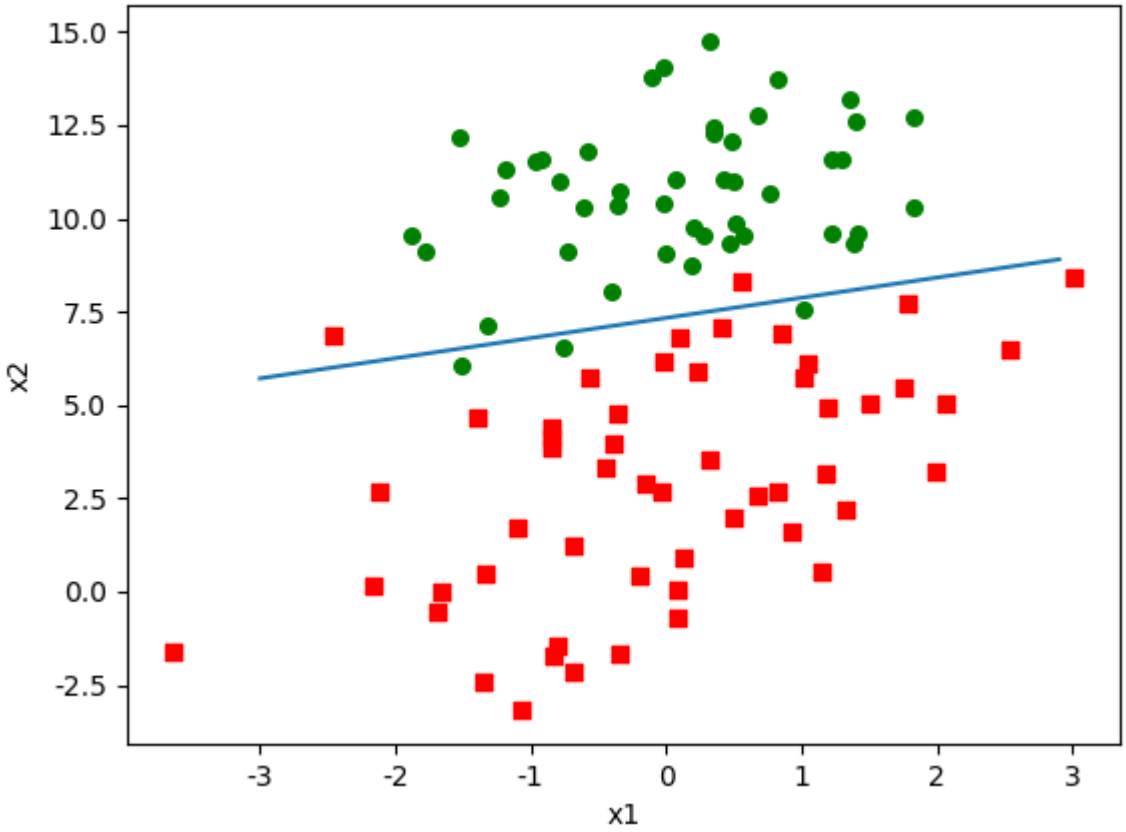
hardlim 500

错误分类 : 2



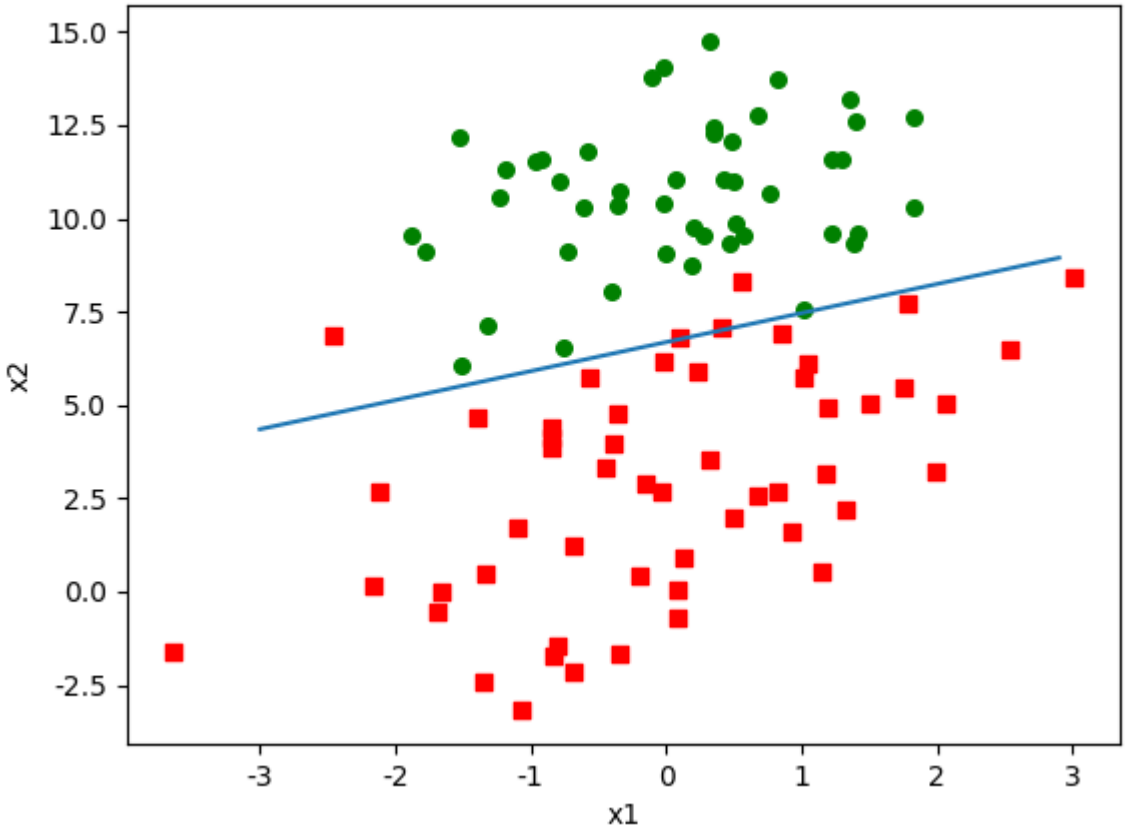
hardlim 501

错误分类 : 5



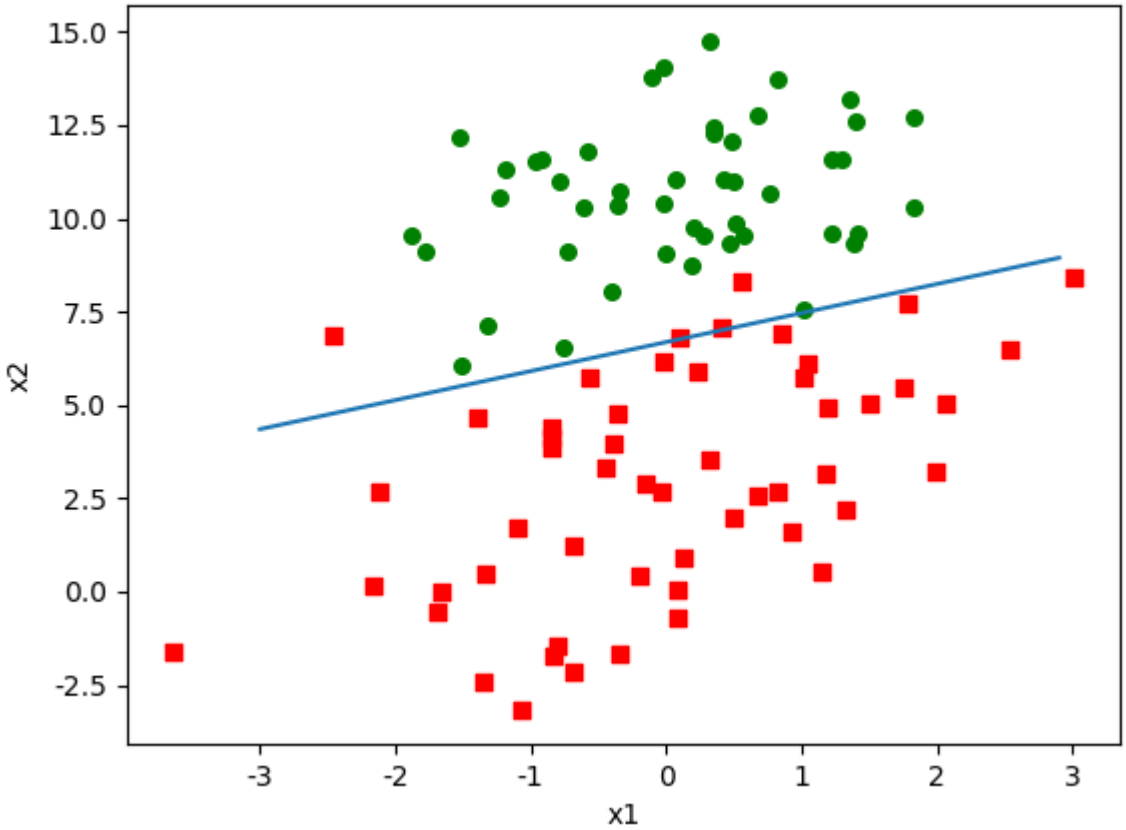
sigmod 500

错误分类 : 2



sigmod 501

错误分类 : 2



可以看出来分类的结果是非常的稳定的

ML

在我们继续讲解之前，需要引入最大似然估计的概念

有一组数据量是m的数据集，每一个数据都是有一个未知但是真实的概率分布独立的生成的，我们想要找到一个概率分布是的尽可能和我们的这个未知的真实分布是一致的就是最大似然法　需要解决的问题

最大似然的核心

因此我们可以将最大似然法看做是尝试的模型的分布尽可能的和经验分布一致匹配的一种尝试，理想的情况下，我们希望匹配真实数据的真实分布，但是我们知道真实分布是不可能的，因为我们对这个真实的分布一无所知

让我们来重新的看待这个问题，假设生成这个数据集的分布是一个我们未知的分布，我们现在想要模拟的向量$\boldsymbol{W}$实际上

就是在数据集$\boldsymbol{X}$上去找他的联合分布模型情况(一个特征就是一个概率分布，多个特征就是查找联合分布)

最大似然的优点

- 已经被证明，当使用的样本数目越多，我们的估计越准确，当样本数目足够多的时候，参数会收敛到参数的真实值

我们一开始假设的样本之间是相互独立的，所以联合分布就是独立分布的乘积

1. 第一个公式很好理解，带包的意思就是当对应的样本的label是1的时候选取前者计算我们的概率

   否则使用后者计算概率

2. 因为概率的乘积运算容易导数值下溢，并且乘法不好计算，所以我们通常使用对数似然转化成加法，并且显然，对数似然并不改变任何性质
3. 这里就凸显出logsitic函数好处,下面的计算公式在最大似然公式推到中会使用
4. 最小化目标函数(w的函数)等价于最大似然估计，求导计算最大似然估计
   求导计算梯度我们，要让后面的一部分就是sigmod函数，而不是其他的类似sigmod性质的其他函数，这样可以保证我们的最大似然法是最精确的

   L(w)是针对W的函数，我们的目的就是要让该函数的值最大，最大我们的估计，在求这个极值的过程中我们需要求导计算

$$P\{Y=1|x\}=p(x)=\frac{1}{1+e^{-w^Tx}}$$
$$P\{Y=0|x\}=p(x)=\frac{1}{1+e^{w^Tx}}$$
$$ln(\frac{P\{Y=1|x\}}{P\{Y=0|x\}})=ln\frac{p(x)}{1-p(x)}=w^Tx$$

$$l(w)=\Pi_{i=1}^{n}(P\{Y=1|x_i\})^{y_i}(1-P\{Y=1|x_i\})^{1-y_i}$$
$$L(w)=ln(l(w))=\Sigma_{i=1}^{n}y_iw^Tx_i-\Sigma_{i=1}^{n}ln(1+e^{w^Tx})$$
$$\frac{\partial L(w)}{\partial w}=\Sigma_{i=1}^{n}y_ix_i-\Sigma_{i=1}^{u}\frac{e^{w^Tx_i}}{1+e^{w^Tx_i}}x_i=\Sigma_{i=1}^{n}(y_i-\frac{1}{1+e^{-w^Tx_i}})x_i$$

为了让导数等于0求得极值$y=\frac{1}{1+e^{-w^Tx}}$

代码

```
def sigmod(inx):
    return 1.0 / (1 + exp(-inx))

def hardlim(inx):
    h = []
    for i in inx:
        if i < 0 : h.append(0)
        else : h.append(1)
    return mat(h).transpose()

def logistic(datamat , label):
    datamatrix = mat(datamat)
    labelmat = mat(label).transpose()
    m , n = shape(datamatrix)
    alpha = 0.001
    step = 501
    weights = ones((n,1))
    # 利用最大似然法的批量梯度下降
    for i in range(step):
        h = sigmod(datamatrix * weights)     # 计算后一项
        error = (labelmat - h)     # 计算误差，上面的yi-...，计算的结果是上式的最后一个表达式中的括号部分内容
        weights += alpha * datamatrix.transpose() * error     # 最后一个乘法代表的计算梯度，只不过是矩阵的形式而已
    return weights
```

显然hardlim或者其他的和logistic的性质类似的函数都没有办法做到和logistic函数一样的效果

BGD

批量梯度下降算法，是最原始的一种梯度下降算法，对于整个数据集进行遍历操作，可以尽可能的保证我们的回归系数的精度，但是带来的缺点显而易见

一旦我们的数据集的个数和我们的特征的数目变得异常的巨大，我们的计算量是非常的庞大的，甚至是不可接收的

SGD

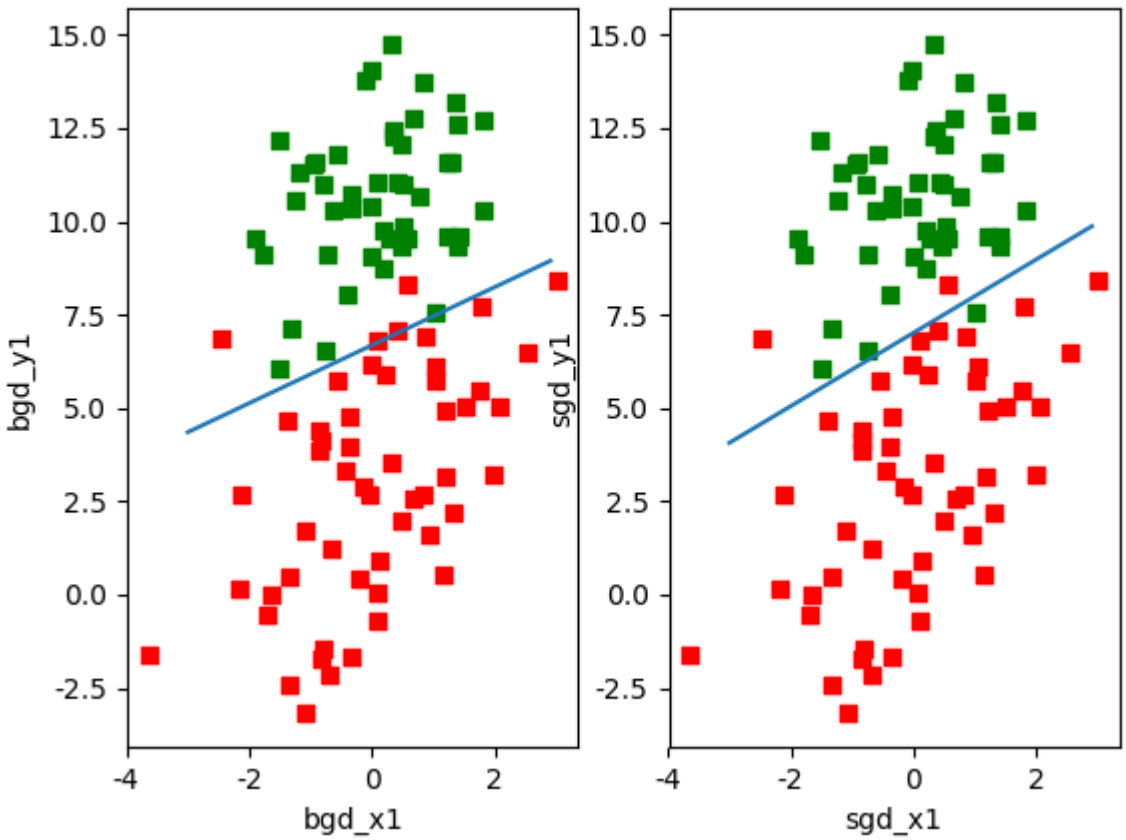
```
def soclogistic(datamat , label):
    import random
    datamatrix = array(datamat)
    m , n = shape(datamat)
    weights = ones(n)
    for j in range(500):
        dataindex = list(range(m))
        for i in range(m):
            alpha = 4 / (i + j + 1.0) + 0.01
            randomindex = random.randint(0,len(dataindex) - 1)
            h = sigmod(sum(datamatrix[randomindex] * weights))
            error = label[randomindex] - h
            weights = weights + alpha * error * datamatrix[randomindex]
            del(dataindex[randomindex])
        return weights
```

为了弥补BGD的缺陷，SGD的核心在于在数据量庞大的时候，我们随机抽取的数据集其实可以当做整体数据集影响的近似估计，我们从训练集中抽取出一小批量的样本作为本次迭代的数据集(该样本集的大小基本不随数据量的变大而变化)

假设现在有**100**个样本，**100**个特征**,100**次迭代，乘法次数：

- BGD : (100 * 100 + 100 * 100) * 100 = 2 * E6
- SGD : (100 + 100) * 100 * 50 = E5

  (该样本集的大小基本不随数据量的变大而变化),所以就算数据集是庞大的，我们的只要抽取的样本小就可以，而起而只要样本狗随机，可以达到和批量梯度下降差不多的拟合结果)



K-Means

聚类算法的目的是: 将想死的对象归为统一簇，将不相同的对象归类到不同的簇，想死的概念取决于我们的相似度计算方法

- 簇质心可以代表簇做出决策
- 算法的性能受举例计算公式的制约(距离代表相似性)，当然，出事的质心的选择也是影响算法效率的重要原因之一
- 初始的质心随机生成或者预处理之后生成，质心还要拨正必须在数据边界之内
- 误差：当前数据点到质心的距离(或者举例的平方和)，之后用来评判角力算法的效果

算法的核心思想

- 每一个点找最近的中心店，病分配至对应的簇里面
- 将簇的中心店更新成簇现有的所有的点的均值
- 算法到所有的点没有变更簇为止

Hyperparameter and validation set

在K聚类算法中，设计到一个东西，K这个K值是用户预先定义的参数，但是问题来了，用户实现是如何知道这堆数据点要分成多少类呢，怎么生成簇比较好?

换句话来说，这个算法本身就是很奇怪的，现实中没有偶一个单一的标准去度量我们的聚类的效果在真是世界如何(不想我们的编程题一样，对就是对错就是错)

，我们当前得到的聚类的分类信息都有可能是错误甚至是不完整的(丢失一些其他的隐藏的信息)

这些都属于超参数的范畴之内

超参数存在于大都数的机器学习算法中，用来控制机器学习算法的行为，超参数不像(逻辑回归的例子中提到的W)不是机器学习算法学习出来的值

比如前面王阁元提到的衰减系数，多项式的次数，还有这里的K聚类的K的值

超参数是很难或者是不是和去训练的但是控制算法行为一组或者一个变量，但是超参数的选取也非常重要，选取不成功还容易导致我们的过拟合或者欠拟合的发生

所以为了解决超参数的问题，我们引入了验证机和交叉验证的概念帮助我们确定一个算法的好坏(相对的来说)

- 验证集
    1. 验证集从何而来，因为我们的模型是在训练集上训练生成去应用在测试集上的，所以首先验证机不能来自于我们的测试集(甚至说，测试机不能对我们的额模型产生任何的影响)所以我们的验证集只能来自于我们的训练集中
    2. 数量 : 通常使用0.8做训练0.2做验证
    3. 验证集的目的 : 验证机是用来测试超参数的

Bisecting K-Means

当我们已经确保超参数已经优化完毕的时候，我们来回头看一下我们的 K 聚类算法，我们怎么评判我们的K聚类算法的效果呢(不是评判K聚类算法的好坏，这是涉及到g更换算法，我之后会提到的)

换句话来说，这个算法本身就是很奇怪的，现实中没有偶一个单一的标准去度量我们的聚类的效果在真是世界如何(不想我们的编程题一样，对就是对错就是错)

，我们当前得到的聚类的分类信息都有可能是错误甚至是不完整的(丢失一些其他的隐藏的信息)

所以我们需要认为在我们的现有的结构上去分析一些性质以进一步优化我们的算法

在这里我们之前定义的误差就有用了，这里我们引入一个定义聚类效果的指标**SSE** 误差平方和，之所以取平方视为了扩大里我们的簇质心元的那些点对我们的算法的影响，因为我们犯错的往往都是那些里我们现在分类的质心有出入的点

- 增加簇的个数(不对，指标不治本)
- 将SSE大的簇分成两个簇，在这个簇上重新运行K聚类算法
- 簇合并
    - 质心举例最近
    - 合并的SSE增幅最小或者降低

所以在这个算法基础之上，又出来一种新的思路，二分K均值聚类算法

我们可以看出来，我们的K均值聚类算法的算法运行结果和运行效率和我们的出事的质心是非常有关系，随机选取质心的缺点

- 迭代次数过大，停不下来
- 结果 收敛到局部最优

为了解决这两个问题，引入了二分K聚类算法

- 概算发一开始只有一个簇
- 选择可以最大降低SSE的簇进行二分簇(可以避免收敛到局部最优的情况，误差平方和可以定性的很好的分析我们当前的簇分类是否是最优的选择，尽可能避免陷入到局部最优的情况)，否则不分或者数目达到用户指定的超参数为止

KFoldXV

$$
\begin{aligned}
&Define\ KFoldXV(\mathbf{D}, A, L, K):\\
&Require:\ \mathbf{D}\ \ data\ set\\
&Require:\ A\ \ Meachine\ Learning\ Algorithm\\
&Require:\ L\ \ Lost function\\
&Require:\ k\\
&Algorithm\\
&for\ i\ from\ 1\ to\ k\ do\\
&\quad f_i = A(\mathbf{D}|\ \mathbf{D\_i})\\
&\quad for\ z^j\ in\ \mathbf{D\_i}\ do\\
&\quad\quad e_j = L(f_i, z^j)\\
&\quad end\ for\\
&end\ for\\
&return\ e
\end{aligned}
$$

但是这里的算法的优劣性只是我们理性定性的分析，实际上我们如何才能就真的确定后者一定比前者优秀呢，这是个很困难的问题甚至和测试机和训练数据还有很大额关系

所以在实际中，到底是选用哪一种需要有哦一个标准来帮助我们决策

在这里就要引入K折交叉验证算法，用来估算算法的泛化误差，安徽一个误差向量，误差响亮的额举止是我们的泛化误差的估计，但是我们并不是随便的就直接按照误差来进行判断，实际上是要使用误差的置信区间，当一个算法的置信区间完全的优秀另一个算法的时候我们才可以说要更优秀，有一些概率的成分在里面

学习了上述的机器学习算法我们感觉好像好像我们的学习算法可以处理很多的事情，但是如果我们要处理的事

1e9的样本数，1e5的特征数呢，按照上面的算法的话，恐怕计算机需要计算很长很航的时间才可以得到一个结果，而且这个结果未必优秀，在新的数据集上泛化的也不一定好

所以，这里就引出来了机器学习领域一个非常重要的困难问题，维数灾难

Difficulty

Curse of dimensionality

可以拿决策树的举例

维数灾难 :决策树的粒子其实引出来了一个位数灾难带来的很重要的困难的因素，统计样本的困难性，当面对一个非常高维的问题的时候，我们必须要手机足够多的样本以供我们可以正确的学习到所有的特征，但是这里又要说一下，一旦我们拿到了很多的数据，又要面对之前偶们提到的，大量的样本和大量的特征的计算困难性

下面为了应对这几个问题，我们就需要进行降维操作，我们需要引入一些算法和概念

降维的好处

- 数据方便使用，节省计算开销
- 去除噪声
- 结果显而易见

PCA

主成分分析可以说是很成功的一种降维的手段

在实际的问题中，虽然我们需要面对的是一个高高维的对象(举足球比赛的例子)的时候，其实我们值关心一些重要的数据特征，洽谈的数据特征我们可以理解成是噪音，对我们要分析的问题没有任何的帮助或者说帮助很小

降维的好处

举例二平面的例子

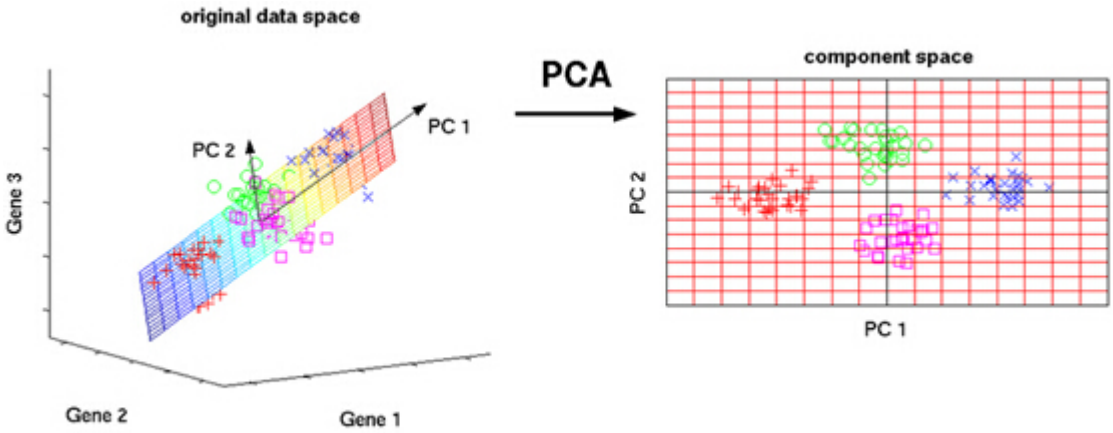
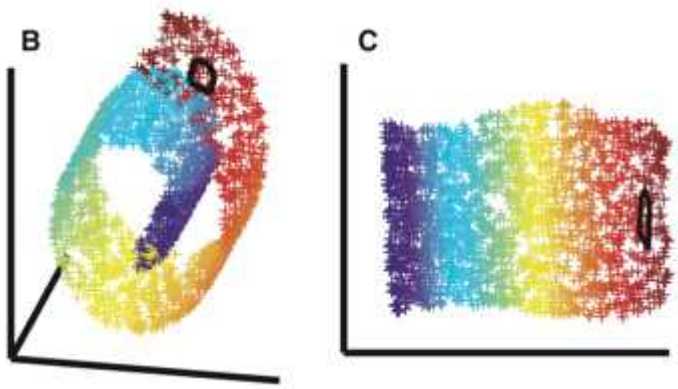


图 1

最大方差决定旋转坐标轴，选定第一个主成分，第二个主成分正交(学习将数据的可能现行相关的特征转化成现行不相关）并且方差最大，直达选取的主成分的数目已经足够秒数问题但是计算复杂性又不会过大

PCA我们可以把他看作是一种数据预处理的手段，之后对我们降维的数据在采用其他的机器学习算法进行学习就可以降低我们的算法的计算开销等等

Maniford



基于的假设：高维中存在的特征在低维下也会保存住，所以我们就可以在低维下研究高维的问题，低维已经足够反应高维的本质特征

在流形学习中，认为现实生活中在高维空间中的大多都是无效的输入或者是噪声，有意义的数据点都是在高位重保持一种特定的数据流向的(比如说视频或者是图片的话，数据都是非常的密集和有规律的，并不是那么多的像素随机组合就可以构成一幅合理的图片或者是视频，在自然语言中，并不是字母的所有组合构成的单词或者是句子都是有意义的，有意义的数据总是有一定的规律并且是非常的密集有序的)，并且这样的特征在低维足够表现的情况下，我们可以将高维的数据过滤掉不必要的噪声特征从而实现降维，但是我们需要注意到，流形学习可能不唯一，对于一个问题，我们可能有多个方式将其从高维压缩到低位，这取决于我们看待问题的角度