引言：

机器学习领域中所谓的降维就是指采用某种映射方法，将原高维空间中的数据点映射到低维度的空间中。降维的本质是学习一个映射函数 f : x->y，其中x是原始数据点的表达，目前最多使用向量表达形式。 y是数据点映射后的低维向量表达，通常y的维度小于x的维度（当然提高维度也是可以的）。f可能是显式的或隐式的、线性的或非线性的。

目前大部分降维算法处理向量表达的数据，也有一些降维算法处理高阶张量表达的数据。之所以使用降维后的数据表示是因为在原始的高维空间中，包含有冗余信息以及噪音信息，在实际应用例如图像识别中造成了误差，降低了准确率；而通过降维,我们希望减少 冗余信息 所造成的误差,提高识别（或其他应用）的精度。又或者希望通过降维算法来寻找数据内部的本质结构特征。

在很多算法中，降维算法成为了数据预处理的一部分，如PCA。事实上，有一些算法如果没有降维预处理，其实是很难得到很好的效果的。

注：我写的东西有一些口语化，而且受限于网页blog的编辑功能，很多地方可能有一些简单。

主成分分析算法（PCA）

Principal Component Analysis(PCA)是最常用的线性降维方法，它的目标是通过某种线性投影，将高维的数据映射到低维的空间中表示，并期望在所投影的维度上数据的方差最大，以此使用较少的数据维度，同时保留住较多的原数据点的特性。

通俗的理解，如果把所有的点都映射到一起，那么几乎所有的信息（如点和点之间的距离关系）都丢失了，而如果映射后方差尽可能的大，那么数据点则会 分散开来，以此来保留更多的信息。可以证明，PCA是丢失原始数据信息最少的一种线性降维方式。（实际上就是最接近原始数据，但是PCA并不试图去探索数 据内在结构）

设 n 维向量w为目标子空间的一个坐标轴方向（称为映射向量），最大化数据映射后的方差，有：

$$\max_{\mathbf{w}} \frac{1}{m-1}\sum_{i=1}^{m}\left(\mathbf{w}^T(\mathbf{x}_i-\bar{\mathbf{x}})\right)^2$$

其中 m 是数据实例的个数， xi是数据实例 i 的向量表达， x拔是所有数据实例的平均向量。定义W为包含所有映射向量为列向量的矩阵，经过线性代数变换，可以得到如下优化目标函数：

$$\min_{\mathbf{W}} tr(\mathbf{W}^T\mathbf{A}\mathbf{W}),\ \text{s.t.}\ \mathbf{W}^T\mathbf{W}=\mathbf{I}$$

其中tr表示矩阵的迹，

$$\mathbf{A}=\frac{1}{m-1}\sum_{i-1}^{m}(\mathbf{x}_i-\bar{\mathbf{x}})(\mathbf{x}_i-\bar{\mathbf{x}})^T;$$

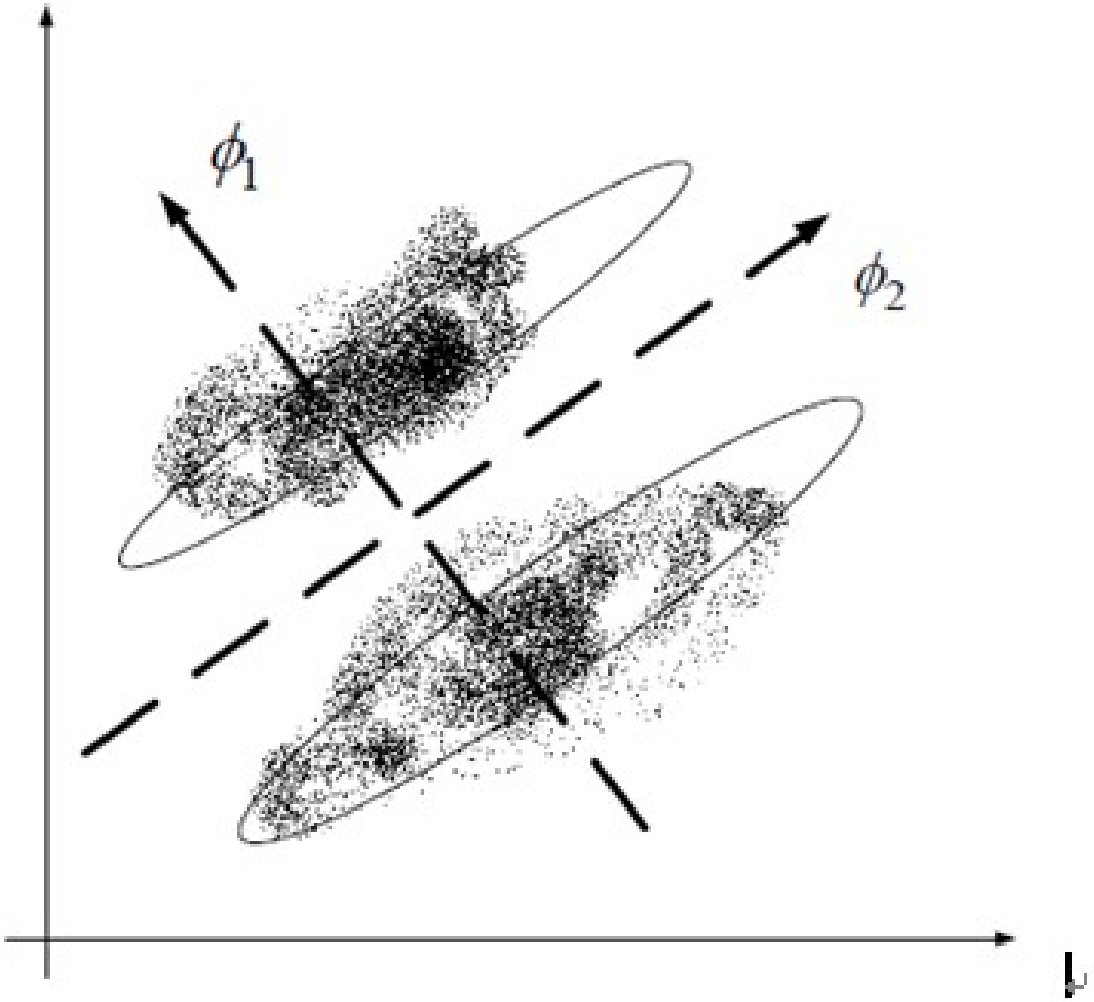
A是数据协方差矩阵。

容易得到最优的W是由数据协方差矩阵前 k 个最大的特征值对应的特征向量作为列向量构成的。这些特征向量形成一组正交基并且最好地保留了数据中的信息。

PCA的输出就是Y = W‘X，由X的原始维度降低到了k维。

PCA追求的是在降维之后能够最大化保持数据的内在信息，并通过衡量在投影方向上的数据方差的大小来衡量该方向的重要性。但是这样投影以后对数据 的区分作用并不大，反而可能使得数据点揉杂在一起无法区分。这也是PCA存在的最大一个问题，这导致使用PCA在很多情况下的分类效果并不好。具体可以看 下图

所示，若使用PCA将数据点投影至一维空间上时，PCA会选择2轴，这使得原本很容易区分的两簇点被揉杂在一起变得无法区分；而这时若选择1轴将会得 到很好的区分结果。



Discriminant Analysis所追求的目标与PCA不同，不是希望保持数据最多的信息，而是希望数据在降维后能够很容易地被区分开来。后面会介绍LDA的方法，是另一 种常见的线性降维方法。另外一些非线性的降维方法利用数据点的局部性质，也可以做到比较好地区分结果，例如LLE，Laplacian Eigenmap等。以后会介绍。

LDA

Linear Discriminant Analysis(也有叫做Fisher Linear Discriminant)是一种有监督的（supervised）线性降维算法。与PCA保持数据信息不同，LDA是为了使得降维后的数据点尽可能地容易被区分！

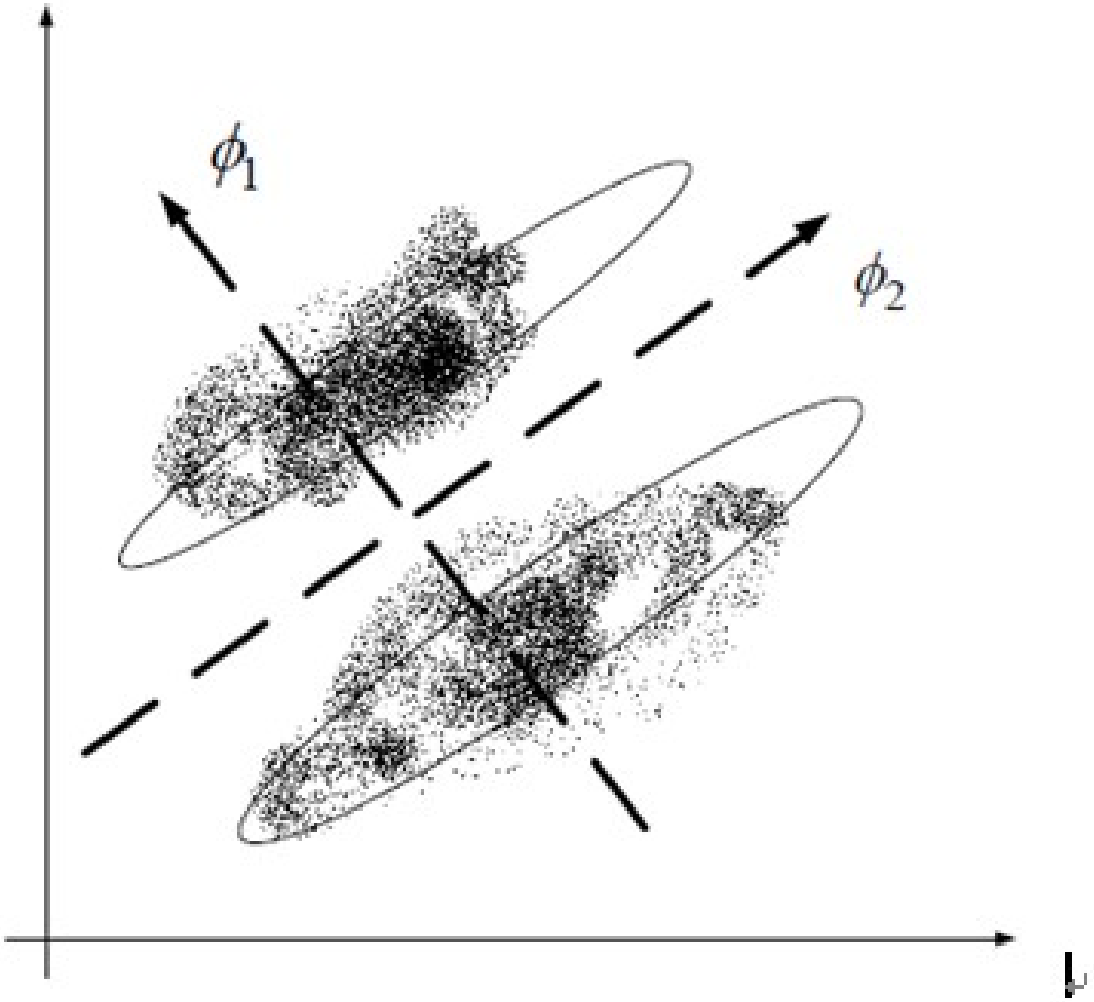
假设原始数据表示为X，（m*n矩阵，m是维度，n是sample的数量）

既然是线性的，那么就是希望找到映射向量a， 使得 a‘X后的数据点能够保持以下两种性质：

1、同类的数据点尽可能的接近（within class）

2、不同类的数据点尽可能的分开（between class）

所以呢还是上次PCA用的这张图，如果图中两堆点是两类的话，那么我们就希望他们能够投影到轴1去（PCA结果为轴2），这样在一维空间中也是很容易区分的。



接下来是推导，因为这里写公式很不方便，我就引用Deng Cai老师的一个ppt中的一小段图片了：

- **Two Classes ω_1, ω_2**

$$z = \boldsymbol{a}^T \boldsymbol{x}$$

$$\tilde{\mu}_i = \frac{1}{n_i} \sum_{z \in \omega_i} z$$

$$\boldsymbol{\mu}_i = \frac{1}{n_i} \sum_{\boldsymbol{x} \in \omega_i} \boldsymbol{x}, \tilde{\mu}_i = \boldsymbol{a}^T \boldsymbol{\mu}_i$$

$$|\tilde{\mu}_1 - \tilde{\mu}_2| = |\boldsymbol{a}^T (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)|$$

$$\tilde{s}_i^2 = \sum_{z \in \omega_i} (z - \tilde{\mu}_i)^2$$

$$\frac{1}{n} (\tilde{s}_1^2 + \tilde{s}_2^2)$$

$$J(\boldsymbol{a}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

思路还是非常清楚的，目标函数就是最后一行J（a)， μ（一飘）就是映射后的中心用来评估类间距，s（一瓢）就是映射后的点与中心的距离之和用来评估类内距。J(a)正好就是从上述两个性质演化出来的。

因此两类情况下：

加上a’a=1的条件（类似于PCA）

Two Classes $$J(\boldsymbol{a}) = \frac{\boldsymbol{a}^T S_B \boldsymbol{a}}{\boldsymbol{a}^T S_W \boldsymbol{a}}$$

$$S_B \boldsymbol{a} = \lambda S_W \boldsymbol{a}$$

可以拓展成多类：

Multi-classes $$J(\boldsymbol{a}) = \frac{\boldsymbol{a}^T S_B \boldsymbol{a}}{\boldsymbol{a}^T S_W \boldsymbol{a}}$$

$$S_W = \sum_{i=1}^{c} S_i = \sum_{i=1}^{c} \sum_{x \in \omega_i} (\boldsymbol{x} - \boldsymbol{\mu}_i)(\boldsymbol{x} - \boldsymbol{\mu}_i)^T$$

$$S_B = \sum_{i=1}^{c} n_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T$$

$$S_B \boldsymbol{a} = \lambda S_W \boldsymbol{a}$$

$$S_B \boldsymbol{a} = \lambda S_T \boldsymbol{a}$$

以上公式推导可以具体参考pattern classification书中的相应章节，讲fisher discirminant的

OK，计算映射向量a就是求最大特征向量，也可以是前几个最大特征向量组成矩阵A=[a1,a2,….ak]之后，就可以对新来的点进行降维了： y = A’ X （线性的一个好处就是计算方便！）

可以发现，LDA最后也是转化成为一个求矩阵特征向量的问题，和PCA很像，事实上很多其他的算法也是归结于这一类，一般称之为谱（spectral）方法。

线性降维算法我想最重要的就是PCA和LDA了，后面还会介绍一些非线性的方法。

局部线性嵌入　（LLE）

Locally linear embedding（LLE）[1] 是一种非线性降维算法，它能够使降维后的数据较好地保持原有　流形结构　。LLE可以说是流形学习方法最经典的工作之一。很多后续的流形学习、降维方法都与LLE有密切联系。

见图1，使用LLE将三维数据（b）映射到二维（c）之后，映射后的数据仍能保持原有的数据流形（红色的点互相接近，蓝色的也互相接近），说明LLE有效地保持了数据原有的流行结构。

但是LLE在有些情况下也并不适用，如果数据分布在整个封闭的球面上，LLE则不能将它映射到二维空间，且不能保持原有的数据流形。那么我们在处理数据中，首先假设数据不是分布在闭合的球面或者椭球面上。

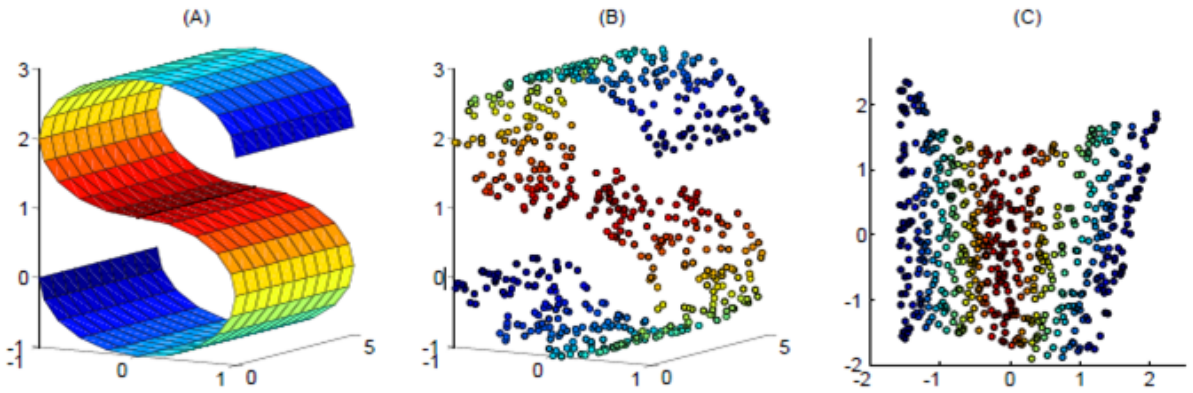


图1 LLE降维算法使用实例

LLE算法认为每一个数据点都可以由其近邻点的线性加权组合构造得到。算法的主要步骤分为三步：(1)寻找每个样本点的k个近邻点；（2）由每个 样本点的近邻点计算出该样本点的局部重建权值矩阵；（3）由该样本点的局部重建权值矩阵和其近邻点计算出该样本点的输出值。具体的算法流程如图2所示：

1. Compute the neighbors of each data point, $\vec{X}_i$.

2. Compute the weights W_{ij} that best reconstruct each data point $\vec{X}_i$ from its neighbors, minimizing the cost in Equation (1) by constrained linear fits.

3. Compute the vectors $\vec{Y}_i$ best reconstructed by the weights W_{ij}, minimizing the quadratic form in Equation (2) by its bottom nonzero eigenvectors.

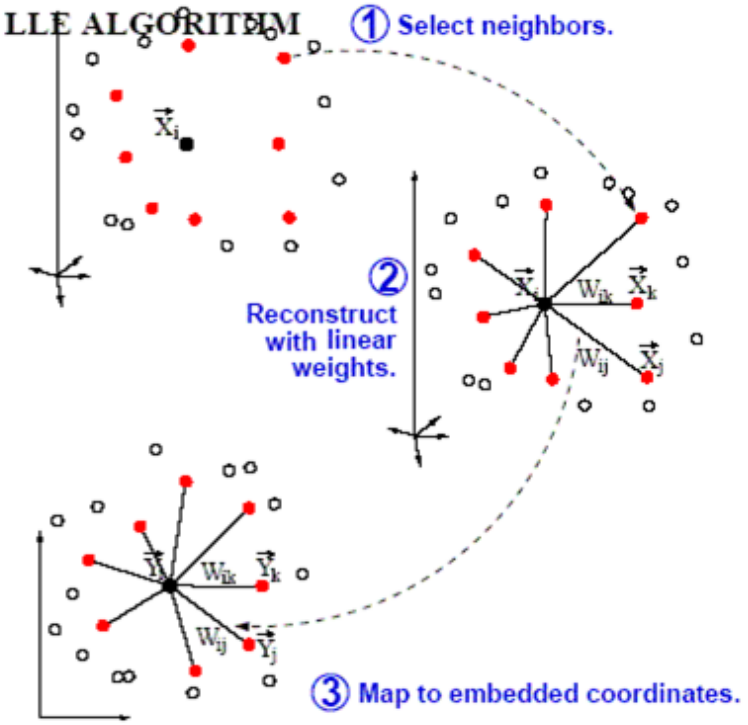


图 2 LLE算法步骤

步骤 1：

算法的第一步是计算出每个样本点的 k 个近邻点。例如采用 KNN 的策略，把相对于所求样本点距离（常用欧氏距离）最近的 k 个样本点规定为所求样本点的 个近邻点，k 是一个预先给定值。

步骤 2：

计算出样本点的局部重建权值矩阵 W，首先定义重构误差：

$$\varepsilon(W)=\sum_i\left|\vec{X}_i-\sum_j W_{ij}\vec{X}_j\right|^2$$

以及局部协方差矩阵 C：

$$C_{jk}=\left(\vec{x}-\vec{\eta}_j\right)\cdot\left(\vec{x}-\vec{\eta}_k\right)$$

其中 x 表示一个特定的点，它的的 K 个近邻点用 η 表示。

于是，目标函数：

最小化：$\varepsilon(W)=\sum_i\left|\vec{X}_i-\sum_j W_{ij}\vec{X}_j\right|^2$

$$\text{其中}\sum_j w_j=1$$

得到：

$$w_j=\frac{\sum_k C_{jk}^{-1}}{\sum_{lm}C_{lm}^{-1}}$$

核心：这里可以直接计算得到 w，然后再步骤 3 中假设用数据重构得到的权重 w 和降维空间中的重构权重 w 是共享的（相同的）。

步骤 3：

将所有的样本点映射到低维空间中。映射条件满足如下所示：

$$\min_{Y} \Phi(Y) = \sum_{i} \left| \vec{Y}_i - \sum_j W_{ij} \vec{Y}_j \right|^2$$

上式可以转化为：

$$\Phi(Y) = \sum_{ij} M_{ij} \left(\vec{Y}_i \cdot \vec{Y}_j \right)$$

其中：$M = (I - W)^T (I - W)$

再加上限制条件：$\sum_i \vec{Y}_i = \vec{0}$（中心化），$\frac{1}{N} \sum_i \vec{Y}_i \vec{Y}_i^T = I$（单位协方差）

可以得到最终解的是这样一个问题：$MY = \lambda Y$

标准的特征分解问题，即取 Y 为 M 的最小 m 个非零特征值所对应的特征向量。在处理过程中，将 M 的特征值从小到大排列，第一个特征值几乎接近于零，那么舍去第一个特征值。通常取第 2 到 m+1 间的特征值所对应的特征向量组成列向量，作为输出结果，即一个 N*m 的数据表达矩阵 Y，假设有 N 个数据点。

Laplacian Eigenmaps 拉普拉斯特征映射

继续写一点经典的降维算法，前面介绍了PCA,LDA，LLE，这里讲一讲Laplacian Eigenmaps。其实不是说每一个算法都比前面的好，而是每一个算法都是从不同角度去看问题，因此解决问题的思路是不一样的。这些降维算法的思想都很 简单，却在有些方面很有效。这些方法事实上是后面一些新的算法的思路来源。

Laplacian Eigenmaps[1] 看问题的角度和LLE有些相似，也是用局部的角度去构建数据之间的关系。

它的直观思想是希望相互间有关系的点（在图中相连的点）在降维后的空间中尽可能的靠近。Laplacian Eigenmaps可以反映出数据内在的流形结构。

Laplacian Eigenmaps也通过构建相似关系图（对应的矩阵为 W ）来重构数据流形的局部结构特征，Laplacian Eigenmaps算法的主要思想是，如果两个数据实例i和j很相似，那么i和j在降维后目标子空间中应该尽量接近，设数据实例的数目为n，目标子空间的维度为m。定义 $n \times m$ 大小的矩阵 Y ，其中每一个行向量 y_i^T 是数据实例i在目标m维子空间中的向量表示，Laplacian Eigenmaps要优化的目标函数如下

$$\min \sum_{i,j} W_{ij} \|y_i - y_j\|^2$$

定义对角矩阵 D ，对角线上 (i,i) 位置元素等于矩阵 W 的第i行之和，经过线性代数变换，上述优化问题可以用矩阵向量形式表示如下：

$$\min tr\left(Y^T L Y\right), \text{ s.t. } Y^T D Y = I$$

其中矩阵 $L = D - W$ 是图拉普拉斯矩阵。限制条件 $Y^T D Y = I$ 保证优化问题有解，并且保证映射后的数据点不会被“压缩”到一个小于m维的子空间中。使得公式最小化的Y的列向量是以下广义特征值问题的m个最小非0特征值（包括重根）对应的特征向量：

$$Ly = \lambda Dy$$

使用时算法具体步骤为：

步骤1：构建图

使用某一种方法来将所有的点构建成一个图，例如使用KNN算法，将每个点最近的K个点连上边。K是一个预先设定的值。

步骤2：确定权重

确定点与点之间的权重大小，例如选用热核函数来确定，如果点i和点j相连，那么它们关系的权重设定为：

$$W_{ij} = e^{-\frac{\|x_i - x_j\|^2}{t}}$$

另外一种可选的简化设定是 $W_{ij} = 1$ 如果点i，j相连，否则 $W_{ij} = 0$ 。

步骤3：特征映射

计算拉普拉斯矩阵L的特征向量与特征值： $Ly = \lambda Dy$

其中D是对角矩阵，满足 $D_{ii} = \sum_j W_{ji}$ ， $L = D - W$ 。

使用最小的m个非零特征值对应的特征向量作为降维后的结果输出。

前面提到过，Laplacian Eigenmap具有区分数据点的特性，可以从下面的例子看出：

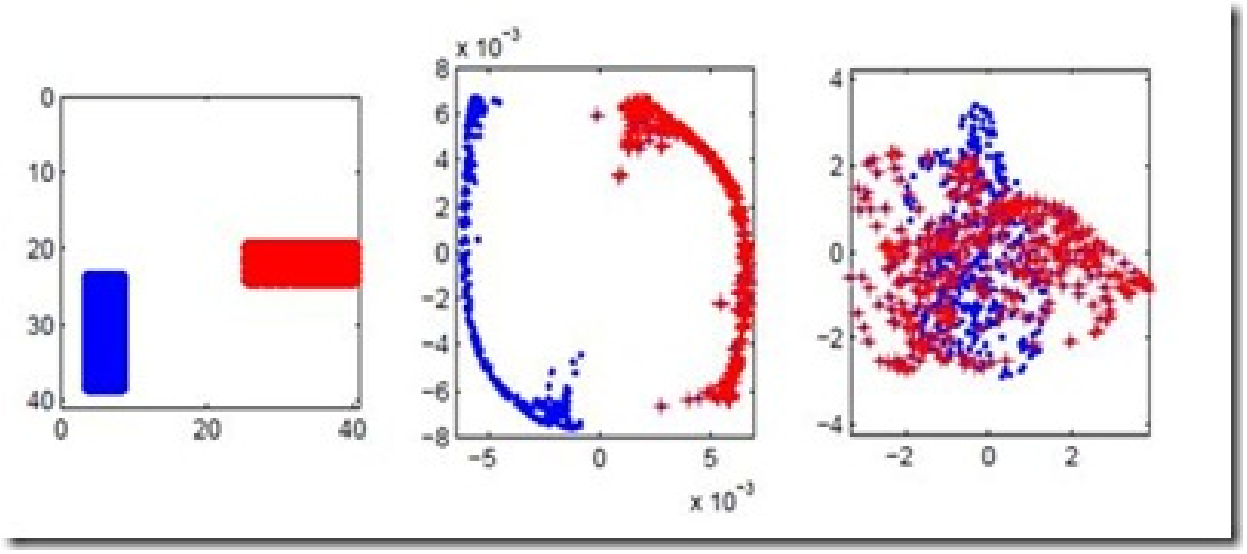


图1 Laplacian Eigenmap实验结果

见图1所示，左边的图表示有两类数据点（数据是图片），中间图表示采用Laplacian Eigenmap降维后每个数据点在二维空间中的位置，右边的图表示采用PCA并取前两个主要方向投影后的结果，可以清楚地看到，在此分类问题 上，Laplacian Eigenmap的结果明显优于PCA。

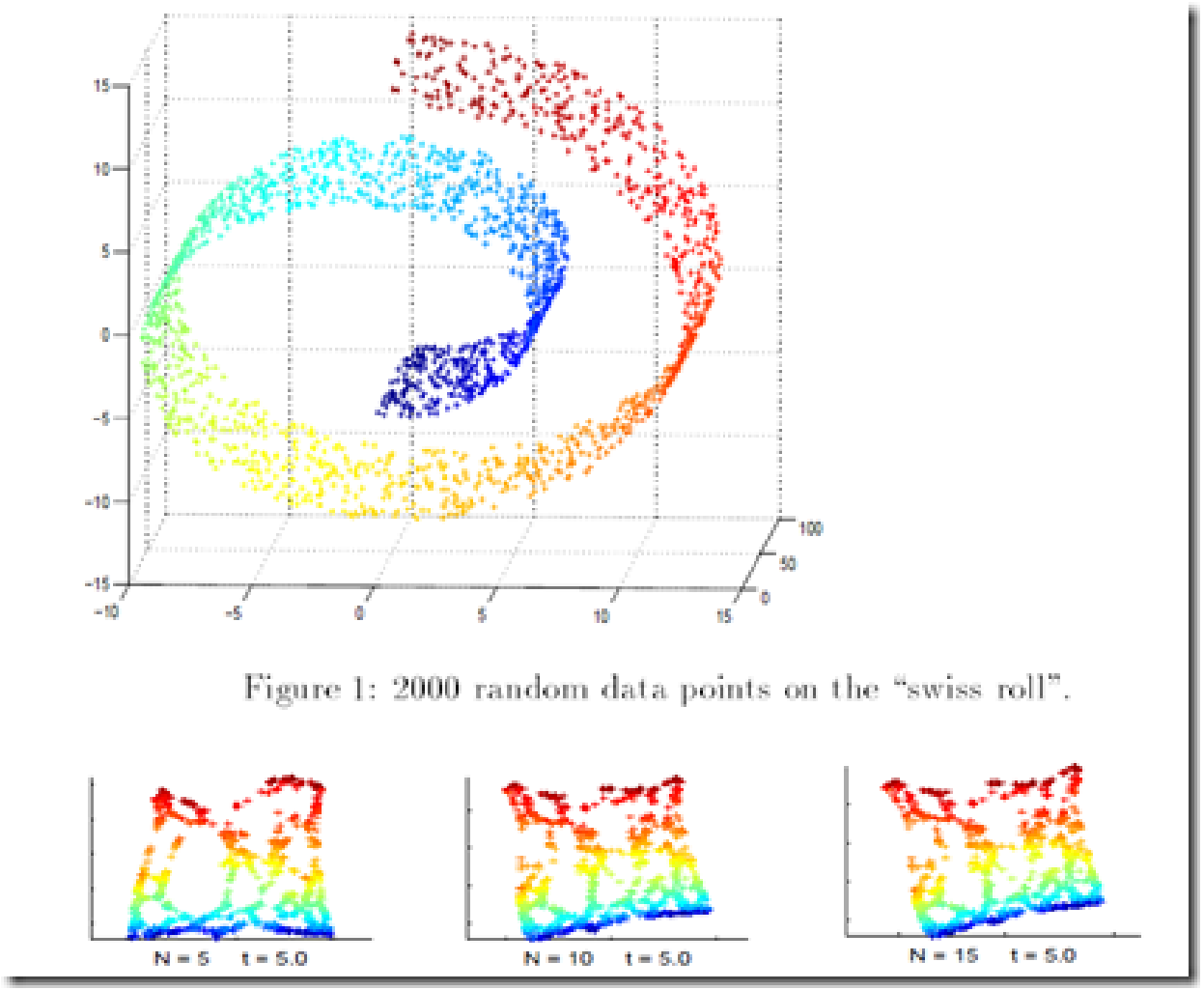


图2 roll数据的降维

图2说明的是，高维数据（图中3D）也有可能是具有低维的内在属性的（图中roll实际上是2D的），但是这个低维不是原来坐标表示，例如如果要保持局部关系，蓝色和下面黄色是完全不相关的，但是如果只用任何2D或者3D的距离来描述都是不准确的。

下面三个图是Laplacian Eigenmap在不同参数下的展开结果（降维到2D），可以看到，似乎是要把整个带子拉平了。于是蓝色和黄色差的比较远。

文章出处：http://blog.csdn.net/xbinworld?viewmode=contents