多元统计 – 多维标度法

彭 真[*]

2018 年 3 月 26 日

写在前面

本次多元统计的选题是多维标度法 (Multi-dimensional Scaling, MDS)，我将从理论和应用两方面简要介绍。

该报告主要参考了周志华老师的**机器学习** [1] 和北大数院 *Yuan Yao* 老师的专著 **A Mathematical Introduction to Data Science**[1][2]、经典教材 **The Elements of Statistical Learning**[3] 以及相关的专业书籍。

1 MDS 理论基础

MDS 是一组用于分析数据中的相似点或相异点的多元数据分析方法，MDS 的一个很好的特点是它允许我们在一对低维空间中表示物体对点之间的距离相似性 (或不相似性)。换句话说，MDS 允许我们在低维空间可视化相似性 (或不相似性)，以便进行探索和检查。MDS 是一种将多维空间的研究对象简化到低维空间进行定位、分析和归类，同时又保留对象间原始关系的数据分析方法。

1.1 MDS

经典的 MDS 或等距欧氏嵌入的问题：给定数据点之间的成对距离，我们可以找到一个欧几里德坐标的系统，这些点的成对距离满足给定的约束？

考虑一个简单的问题：根据样本点计算样本之间距离。给定 n 个样本 $x_1, x_2, \ldots, x_n \in R^p$，令 $X = [x_1, x_2, \ldots, x_n]^{p\times n}$，两样本点 x_i 和 x_j 间的欧式距离 d_{ij}(*dissimilarity measure*) 满足

$$d_{ij}^2 = \|x_i - x_j\|^2 = (x_i - x_j)^T (x_i - x_j) = x_i^T x_i + x_j^T x_j - 2x_i^T x_j$$

现在我们考虑这个问题的逆问题:根据样本之间距离估计样本点。如果给定 d_{ij},我们能否找到 $x_1, x_2, \cdots, x_n$ 满足上式。显然这个问题有解且解不唯一。我们可以最小化以下问题 (被称为 *stress function*) 来寻找待估计的样本点：

$$S_{MDS}(x_1, x_2, \ldots, x_n) = \sum_{i\neq j} (d_{ij} - \|x_i - x_j\|)^2$$

[*]学号：SY1709129，Email：pengzhen@buaa.edu.cn，TEL：18810993590

[1]preprint http://www.math.pku.edu.cn/teachers/yaoy/reference/book05.pdf

类似的，基于上述形式平方最小的缩放变体，*Sammon Mapping* 被提出：

$$S_{SM}\left(x_1, x_2, \ldots, x_n\right) = \sum_{i \neq j} \frac{\left(d_{ij} - \|x_i - x_j\|\right)^2}{d_{ij}}$$

根据特征向量这些问题存在显式解，传统的 MDS 算法的中心思想是：

1. 将距离矩阵 $D = d_{ij}^2$ 转换为内积表示
2. 对该内积做特征值分解

接下来我们将看 MDS 算法如何利用 D。

MDS 推导 [2]

令 K 为内积矩阵

$$K = X^T X$$

其中，$k = \operatorname{diag}\left(K_{ii}\right)$。故

$$D = \left(d_{ij}^2\right) = k \cdot \mathbf{1}^T + \mathbf{1} \cdot k^T - 2K$$

其中 $\mathbf{1} = (1, 1, \ldots, 1)^T \in R^p$。而样本去中心化处理：

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{1}{n} X \cdot \mathbf{1}$$

$$\tilde{x}_i = x_i - \hat{\mu}_n = x_i - \frac{1}{n} X \cdot \mathbf{1} \Rightarrow \tilde{X} = X - \frac{1}{n} X \cdot \mathbf{1} \cdot \mathbf{1}^T$$

易得

$$\begin{aligned} \tilde{K} &= \tilde{X}^T \tilde{X} \\ &= K - \tfrac{1}{n} K \cdot \mathbf{1} \cdot \mathbf{1}^T - \tfrac{1}{n} \mathbf{1} \cdot \mathbf{1}^T \cdot K + \tfrac{1}{n^2} \mathbf{1} \cdot \mathbf{1}^T \cdot K \cdot \mathbf{1} \cdot \mathbf{1}^T \end{aligned}$$

令

$$B = -\frac{1}{2} H \cdot D \cdot H^T$$

其中，$H = I - \frac{1}{n} \mathbf{1} \cdot \mathbf{1}^T$ 为居中矩阵 (centering matrix，规范化)。因此

$$\begin{aligned} B &= -\tfrac{1}{2} H \cdot D \cdot H^T \\ &= -\tfrac{1}{2} H \cdot \left(k \cdot \mathbf{1}^T + \mathbf{1} \cdot k^T - 2K\right) \cdot H^T \\ &= H \cdot K \cdot H^T \\ &= K - \tfrac{1}{n} K \cdot \mathbf{1} \cdot \mathbf{1}^T - \tfrac{1}{n} \mathbf{1} \cdot \mathbf{1}^T \cdot K + \tfrac{1}{n^2} \mathbf{1} \cdot \mathbf{1}^T \cdot K \cdot \mathbf{1} \cdot \mathbf{1}^T \\ &= \tilde{K} \end{aligned}$$

也就是说

$$B = -\frac{1}{2} H \cdot D \cdot H^T = \tilde{X}^T \tilde{X}$$

以上显示，我们可以利用给定的矩阵向量 $D = d_{ij}^2$ 转换为内积矩阵 $B = -\frac{1}{2} H \cdot D \cdot H^T$，对 B 应用特征值分解可以得到以原地为中心样本点坐标。在现实应用中为了有效降维，往往仅需要降维后的矩阵与原

始空间中距离尽可能接近，而不必严格相等。此时可取 B 中前 $k(<p)$ 个非零特征值对应的特征向量，得到样本的低维坐标，从而达到降维的目的。

Algorithm 1 经典 MDS 算法 [2]

```
Input: 距离矩阵 $D^{n\times n}$ 其中 $D_{ij}=d_{ij}^2$。
1: 计算内积矩阵 $B=-\frac{1}{2}HDH^T$，其中 $H$ 为居中矩阵 (centering matrix，规范化)；
2: 计算特征分解 $B=U\Lambda U^{\mathrm{T}}$，其中对角矩阵 $\Lambda=diag\left(\lambda_1,\cdots,\lambda_n\right)$；
3: 取前 $k$ 个最大的非零特征值及其相应的特征向量，
       $$\tilde{X}_k=U_k\Lambda_k^{1/2}$$
   其中，$U_k=\left[u_1,\cdots,u_k\right],\Lambda_k=diag\left(\lambda_1,\cdots,\lambda_k\right)$
```

1.2 MDS 与 PCA 关系

首先回顾一下主成分分析 (Principle Component Analysis, PCA)，其目标是将数据从高维数据投影到低维数据，从而使数据的低维方差最大化。PCA 计算协方差矩阵的奇异值分解 (Singular value decomposition, SVD)：$\hat{\Sigma}_n=\frac{1}{n-1}\tilde{X}\tilde{X}^T=\frac{1}{n}US^2U^T$，其中，$\tilde{X}=USV^T$，并取前 k 个左奇异向量作为嵌入的坐标表示为主成分。

然后重新考虑 MDS 的模型：

$$\min_{x_i\in R^k}\sum_{i,j}\left(\left\|x_i-x_j\right\|^2-d_{ij}^2\right)^2$$

由前面的推导，构造核矩阵 $B=-\frac{1}{2}H\cdot D\cdot H^T$，其中，$H=I-\frac{1}{n}\mathbf{1}\cdot\mathbf{1}^T$, $D=d_{ij}^2$，上式等价于

$$\min_{X\in R^{k\times n}}\sum_{i\neq j}\left\|X^TX-B\right\|_F^2$$

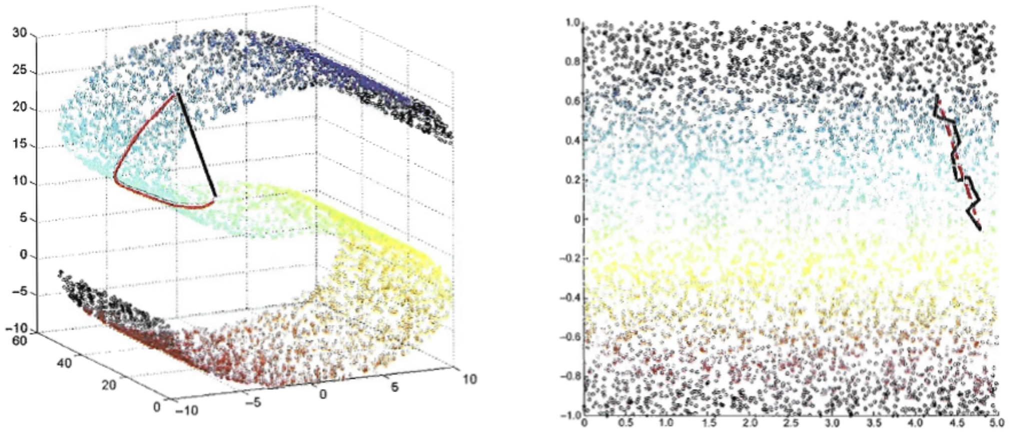
当 B 为**正半定矩阵**时，B 可以表示为内积形式，因此 X 可表示为 $B=\tilde{X}^T\tilde{X}$ 的 k 个最大特征值对应的特征向量，或者说，$\tilde{X}=USV^T$ 中前 k 个右奇异向量。从中心化的数据矩阵进行 SVD 分解这个角度来说，MDS 和 PCA 是统一的，当成对距离是欧几里得距离时，本质上说，MDS 是 PCA 的对偶问题。

比较 MDS 与 PCA 的异同发现：两者都是把观察的数据用较少的维数来表达；不同之处在于，MDS 利用的是成对样本间相似性，目的是利用这个信息去构建合适的低维空间，样本在此空间的距离和在高维空间中的样本间的相似性尽可能的保持一致。

1.3 推广到 ISOMAP

Isomap[4] 认为低维流形嵌入到高维空间之后，直接在高维空间中计算直线距离具有误导性，因为高维空间中的直线距离在低维嵌入流形上式不可达的。图1为高维数据分布图，蓝色虚线为高维空间中的直线距离，红色实线为相同两点的测地线距离。

利用流形可以进行欧氏距离计算的性质，对每个点基于欧氏距离找出其近邻点，于是可以建立一个近邻连接图，图中近邻点之间存在连接，而非近邻点不存在连接，于是计算两点之间测地线距离的问题就转变为计算近邻连接图上两点之间的最短路径问题。



(a) 测地线距离与高维直线距离 (b) 测地线距离与近邻距离

图 1: 测地线距离

Algorithm 2 Isomap 算法 [1]

```
Input: 样本集 $D = x_1, x_2, ..., x_n$，近邻参数 $k$，低维空间维数 $d'$。
1: for $i = 1$ to $m$ do
2:     确定 $x_i$ 的 $k$ 近邻；
3:     $x_i$ 与 $k$ 近邻点之间的距离设置为欧氏距离，与其他点的距离设置为无穷大；
4: end for
5: 调用最短路径算法计算两样本点之间的距离 $dist(x_i, x_j)$；
6: 将 $dist(x_i, x_j)$ 最为 MDS 算法的输入；
7: return MDS 算法的输出；
Output: 样本集 $D$ 在低维空间的投影 $Z = (z_1, z_2, \cdots, z_m)$
```

Isomap 算法的基本特征可以描述为：我们获取数据的低维嵌入，保证附近的点被映射到附近，距离较远的点被映射到距离较远。换句话说，Isomap 算法对数据距离有全局控制，因此这个方法是一个全局方法。Isomap 算法试图保持近邻样本之间的距离不变，是 **MDS 算法在高维流形上的推广，即将样本点之间的欧氏距离用黎曼距离替代**。其主要缺点在于其计算复杂性，根源在于全矩阵特征向量分解。关于收敛性，在流形上的稠密样本和正则 (dense-sample and regularity) 条件下，Isomap 算法保证收敛以保持流形上的测地距离。

2 MDS 应用

2.1 MDS 的 python 实现

该部分主要是利用非常成熟的 Python 开发的机器学习库 Scikit-Learn[2][5]，其中包含大量机器学习算法、数据集，是数据挖掘方便的工具，因此我就对其中四个自带的案例代码运行一遍并注释，结果见附

[2]http://scikit-learn.org/stable/, https://github.com/scikit-learn/scikit-learn

录A，具体可看 *MDS in python* 文件夹下相关文件，*MDS_python.ipynb* 为 python 的源码，*MDS_python*，实验环境是 Jupyter Notebook。

2.2 MDS 的 R 实现

该部分主要用 R 中自带的 cmdscale() 函数应用到数据集 eurodist 中，实现 21 个城市间距离的可视化[3]，结果见附录B，具体可看 *MDS in R* 文件夹下相关文件，*MDS_R_tex.Rmd* 生成 tex 的源码，*MDS_R.Rmd* 为生成 html 的源码。实验环境是 RStudio。

3 总结

这次多元统计学习的方法是多维标度法（MDS），可以用于数据降维以及可视化，虽然没有 PCA 或 LDA 方法有名，但是这个方法可以说是给了 ISOMAP 算法一个直接的灵感，即将欧氏距离改为欧氏距离，从而衍生了流形学习这一非线性降维的分支。

这次学习分别从理论分析和代码实现两方面对 MDS 这一方法介绍，第一部分包括：理论推导过程借鉴文献 [2]，讨论 MDS 与 PCA 的对偶关系，并介绍 MDS 算法到流形上的推广；第二部分包括用 python 和 R 两种语言实现 MDS 算法，详细报告见附件以及文件夹中的文件。

值得注意的是，Python 和 R 里面求解 MDS 算法都不是简单的对内积矩阵做特征值分解，均使用了 SMACOF 算法，这个算法应该是类似于对 *stree function*（见公式1.1）做梯度下降算法。

最后做个总结：MDS 算法形式上很简单，其中的数学理论比较深（可见文献 [2] 第一章第三节、第三章、第四章等），因此在多元统计与机器学习的运用广泛，但是随着大数据时代，获得的数据更多的展现出非线性结构，线性方法如 PCA、LDA、MDS 方法在处理曲面结构的数据性能比不上非线性降维方法，因此基于核的方法将线性方法非线性化，使其具有能力处理非线性结构。

参考文献

[1] 周志华. 机器学习 (Machine learning)[M]: 清华大学出版社, 2016.

[2] YAO Y. A mathematical introduction to data science[J], .

[3] HASTIE T, TIBSHIRANI R, FRIEDMAN J. The Elements of Statistical Learning[J]. Journal of the Royal Statistical Society, 2009, 167(1): 267–268.

[4] TENENBAUM J B, DE SILVA V, LANGFORD J C. A global geometric framework for nonlinear dimensionality reduction[J]. science, 2000, 290(5500): 2319–2323.

[5] PEDREGOSA F, VAROQUAUX G, GRAMFORT A, et al. Scikit-learn: Machine Learning in Python[J]. Journal of Machine Learning Research, 2011, 12: 2825–2830.

[6] I. BORG P J F G. Springer Series in Statistics: Modern Multidimensional Scaling: Theory and Applications[M]. 2nd: Springer, 2005.

[3] http://www.gastonsanchez.com/visually-enforced/how-to/2013/01/23/MDS-in-R/

Python 实现多维标度法

彭 真

1 MDS in Python

Multidimensional scaling (MDS) seeks a low-dimensional representation of the data in which the distances respect well the distances in the original high-dimensional space.

In general, is a technique used for analyzing similarity or dissimilarity data. MDS attempts to model similarity or dissimilarity data as distances in a geometric spaces. The data can be ratings of similarity between objects, interaction frequencies of molecules, or trade indices between countries.

There exists two types of MDS algorithm: metric and non metric. In the scikit-learn, the class MDS implements both. In Metric MDS, the input similarity matrix arises from a metric (and thus respects the triangular inequality), the distances between output two points are then set to be as close as possible to the similarity or dissimilarity data. In the non-metric version, the algorithms will try to preserve the order of the distances, and hence seek for a monotonic relationship between the distances in the embedded space and the similarities/dissimilarities.

Let S be the similarity matrix, and X the coordinates of the n input points. Disparities $\hat{d}_{ij}$ are transformation of the similarities chosen in some optimal ways. The objective, called the stress, is then defined by $\sum\limits_{i<j} d_{ij}(X) - \hat{d}_{ij}(X)$

The simplest metric MDS model, called absolute MDS, disparities are defined by $\hat{d}_{ij} = S_{ij}$. With absolute MDS, the value S_{ij} should then correspond exactly to the distance between point i and j in the embedding point.

Most commonly, disparities are set to $\hat{d}_{ij} = bS_{ij}$.

1.1 MDS by SMACOF algorithm

sklearn.manifold.MDS

Computes multidimensional scaling using the SMACOF algorithm.

The SMACOF (Scaling by MAjorizing a COmplicated Function) algorithm is a multidimensional scaling algorithm which minimizes an objective function (the *stress*) using a majorization technique. Stress majorization, also known as the Guttman Transform, guarantees a monotone convergence of stress, and is more powerful than traditional techniques such as gradient descent.

The SMACOF algorithm for metric MDS can summarized by the following steps: 1. Set an initial start configuration, randomly or not. 2. Compute the stress 3. Compute the Guttman Transform 4. Iterate 2 and 3 until convergence.

1.2 References:

- "Modern Multidimensional Scaling - Theory and Applications" Borg, I.; Groenen P. Springer Series in Statistics (1997)
- "Nonmetric multidimensional scaling: a numerical method" Kruskal, J. Psychometrika, 29 (1964)
- "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis" Kruskal, J. Psychometrika, 29, (1964)

2 Examples using sklearn.manifold.MDS

2.1 Multi-dimensional scaling

An illustration of the metric MDS on generated noisy data.

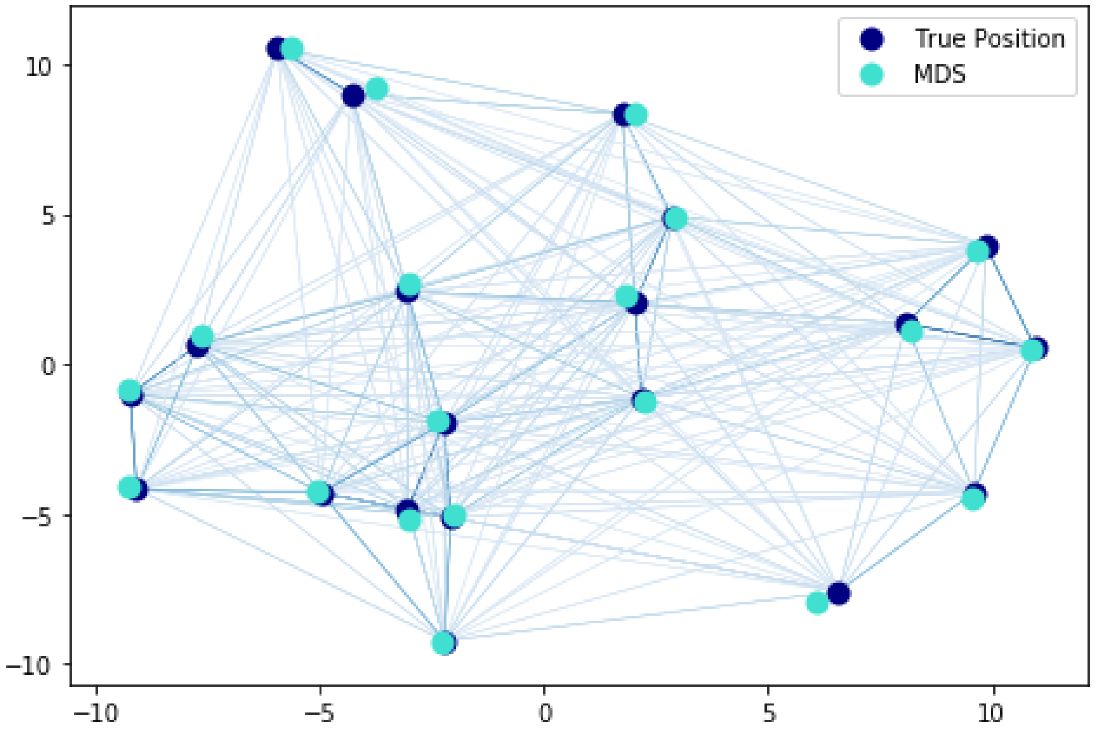
The reconstructed points using the metric MDS and non metric MDS are slightly shifted to avoid overlapping.

plot_mds.ipynb

```
In [1]: # Author: Nelle Varoquaux <nelle.varoquaux@gmail.com>
        # License: BSD
        % matplotlib inline
        print(__doc__)
        # 载入包
        import numpy as np

        from matplotlib import pyplot as plt
        from matplotlib.collections import LineCollection

        from sklearn import manifold
        from sklearn.metrics import euclidean_distances
        from sklearn.decomposition import PCA

        n_samples = 20
```

```python
seed = np.random.RandomState(seed=3)
X_true = seed.randint(0, 20, 2 * n_samples).astype(np.float)
X_true = X_true.reshape((n_samples, 2))
# 数据中心化
X_true -= X_true.mean()

similarities = euclidean_distances(X_true)

# 对距离矩阵加噪
noise = np.random.rand(n_samples, n_samples)
noise = noise + noise.T
noise[np.arange(noise.shape[0]), np.arange(noise.shape[0])] = 0
similarities += noise

# MDS 算法求出低维嵌入
mds = manifold.MDS(n_components=2, max_iter=3000, eps=1e-9,     #manifold.MDS
                   random_state=seed, dissimilarity="precomputed", n_jobs=1)
pos = mds.fit(similarities).embedding_

# 调整数据
pos *= np.sqrt((X_true ** 2).sum()) / np.sqrt((pos ** 2).sum())

# 旋转数据
clf = PCA(n_components=2)
X_true = clf.fit_transform(X_true)
pos = clf.fit_transform(pos)

# 绘图
fig = plt.figure(1)
ax = plt.axes([0., 0., 1., 1.])

s = 100
plt.scatter(X_true[:, 0], X_true[:, 1], color='navy', s=s, lw=0,
            label='True Position')
plt.scatter(pos[:, 0], pos[:, 1], color='turquoise', s=s, lw=0, label='MDS')
plt.legend(scatterpoints=1, loc='best', shadow=False)
```

```
similarities = similarities.max() / similarities * 100
similarities[np.isinf(similarities)] = 0

# 画边
start_idx, end_idx = np.where(pos)
# a sequence of (*line0*, *line1*, *line2*), where::
#             linen = (x0, y0), (x1, y1), ... (xm, ym)
segments = [[X_true[i, :], X_true[j, :]]
            for i in range(len(pos)) for j in range(len(pos))]
values = np.abs(similarities)
lc = LineCollection(segments,
                    zorder=0, cmap=plt.cm.Blues,
                    norm=plt.Normalize(0, values.max()))
lc.set_array(similarities.flatten())
lc.set_linewidths(0.5 * np.ones(len(segments)))
ax.add_collection(lc)

plt.show()
```



2.2 Manifold learning on handwritten digits

An illustration of various embeddings on the digits dataset.

The RandomTreesEmbedding, from the sklearn.ensemble module, is not technically a manifold embedding method, as it learn a high-dimensional representation on which we apply a dimensionality reduction method. However, it is often useful to cast a dataset into a representation in which the classes are linearly-separable.

t-SNE will be initialized with the embedding that is generated by PCA in this example, which is not the default setting. It ensures global stability of the embedding, i.e., the embedding does not depend on random initialization.

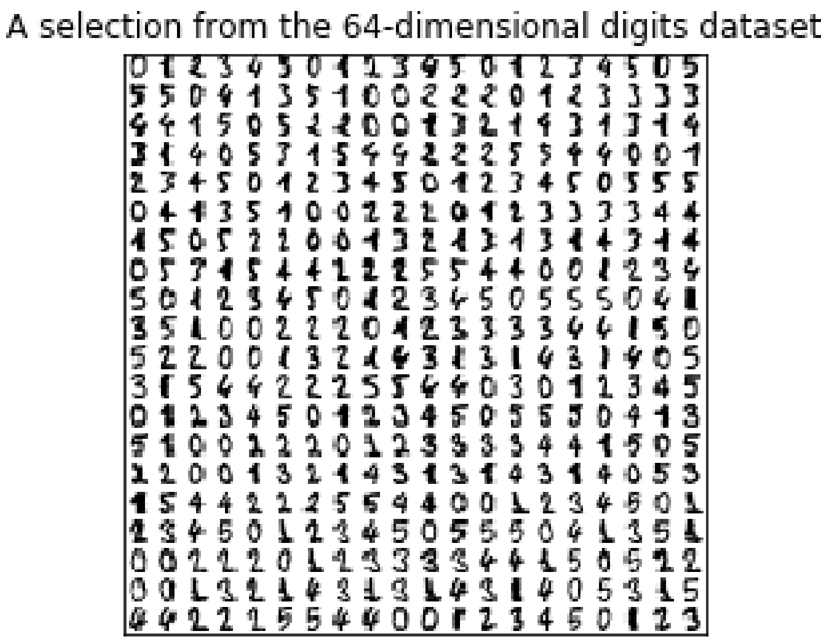
plot_lle_digits.ipynb

```
In [2]: # Authors: Fabian Pedregosa <fabian.pedregosa@inria.fr>
        #          Olivier Grisel <olivier.grisel@ensta.org>
        #          Mathieu Blondel <mathieu@mblondel.org>
        #          Gael Varoquaux
        # License: BSD 3 clause (C) INRIA 2011
        % matplotlib inline
        print(__doc__)
        # 载入包
        from time import time

        import numpy as np
        import matplotlib.pyplot as plt
        from matplotlib import offsetbox
        from sklearn import (manifold, datasets, decomposition, ensemble,
                             discriminant_analysis, random_projection)

        digits = datasets.load_digits(n_class=6)
        X = digits.data
        y = digits.target
        n_samples, n_features = X.shape
        n_neighbors = 30

        #----------------------------------------------------------------------
        # 定义函数对低维嵌入绘图
        def plot_embedding(X, title=None):
            x_min, x_max = np.min(X, 0), np.max(X, 0)
```

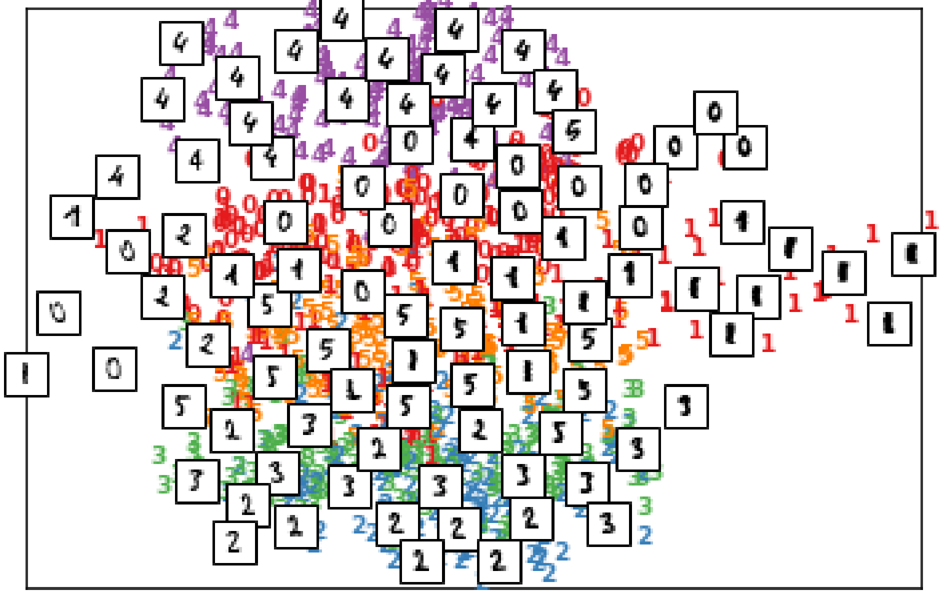
```
    X = (X - x_min) / (x_max - x_min)

    plt.figure()
    ax = plt.subplot(111)
    for i in range(X.shape[0]):
        plt.text(X[i, 0], X[i, 1], str(digits.target[i]),
                 color=plt.cm.Set1(y[i] / 10.),
                 fontdict={'weight': 'bold', 'size': 9})

    if hasattr(offsetbox, 'AnnotationBbox'):
        # only print thumbnails with matplotlib > 1.0
        shown_images = np.array([[1., 1.]])  # just something big
        for i in range(digits.data.shape[0]):
            dist = np.sum((X[i] - shown_images) ** 2, 1)
            if np.min(dist) < 4e-3:
                # don't show points that are too close
                continue
            shown_images = np.r_[shown_images, [X[i]]]
            imagebox = offsetbox.AnnotationBbox(
                offsetbox.OffsetImage(digits.images[i], cmap=plt.cm.gray_r),
                X[i])
            ax.add_artist(imagebox)
    plt.xticks([]), plt.yticks([])
    if title is not None:
        plt.title(title)

#----------------------------------------------------------------------
# 绘制手写数字图像
n_img_per_row = 20
img = np.zeros((10 * n_img_per_row, 10 * n_img_per_row))
for i in range(n_img_per_row):
    ix = 10 * i + 1
    for j in range(n_img_per_row):
        iy = 10 * j + 1
        img[ix:ix + 8, iy:iy + 8] = X[i * n_img_per_row + j].reshape((8, 8))

plt.imshow(img, cmap=plt.cm.binary)
plt.xticks([])
plt.yticks([])
plt.title('A selection from the 64-dimensional digits dataset')

#----------------------------------------------------------------------
# PCA: 前 2 个主成分的投影

print("Computing PCA projection")
t0 = time()
X_pca = decomposition.TruncatedSVD(n_components=2).fit_transform(X)   # PCA
plot_embedding(X_pca,
               "Principal Components projection of the digits (time %.2fs)" %
               (time() - t0))

#----------------------------------------------------------------------
# LDA: 前 2 个线性判别分量的投影

print("Computing Linear Discriminant Analysis projection")
X2 = X.copy()
X2.flat[::X.shape[1] + 1] += 0.01  # Make X invertible
t0 = time()
X_lda = discriminant_analysis.LinearDiscriminantAnalysis(   #LDA
    n_components=2).fit_transform(X2, y)
plot_embedding(X_lda,
               "Linear Discriminant projection of the digits (time %.2fs)" %
               (time() - t0))

#----------------------------------------------------------------------
# 数字数据集的 MDS 嵌入

print("Computing MDS embedding")
clf = manifold.MDS(n_components=2, n_init=1, max_iter=100)   # manifold.MDS
t0 = time()
X_mds = clf.fit_transform(X)
print("Done. Stress: %f" % clf.stress_)
plot_embedding(X_mds,
               "MDS embedding of the digits (time %.2fs)" %
```

```
                    (time() - t0))

    #----------------------------------------------------------------------
    # 数字数据集的 Isomap 嵌入

    print("Computing Isomap embedding")
    t0 = time()
    X_iso = manifold.Isomap(n_neighbors, n_components=2).fit_transform(X)   # Isomap
    print("Done.")
    plot_embedding(X_iso,
                   "Isomap projection of the digits (time %.2fs)" %
                   (time() - t0))

    plt.show()
```

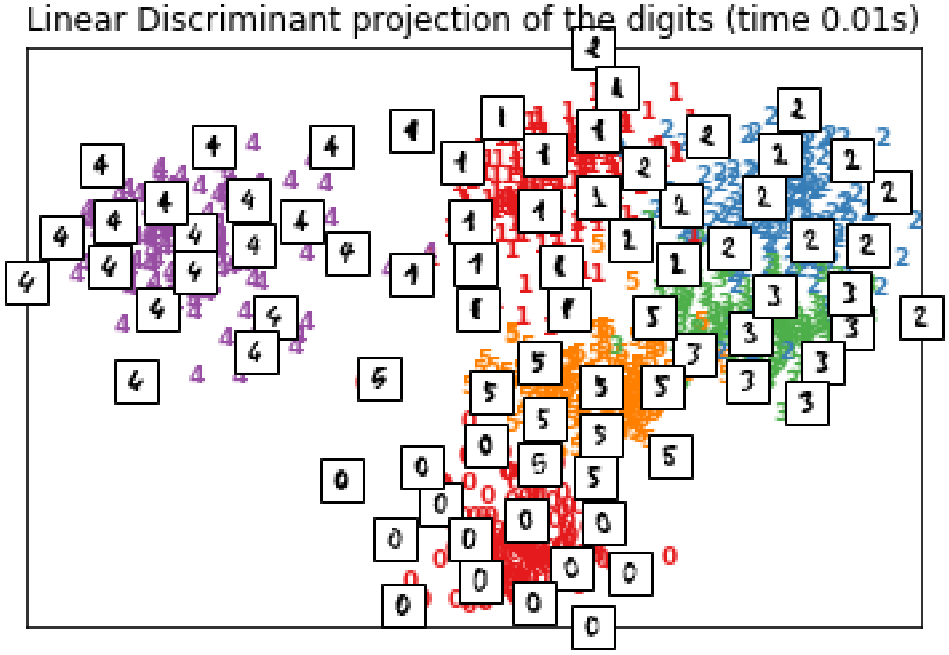
```
Automatically created module for IPython interactive environment
Computing PCA projection
Computing Linear Discriminant Analysis projection
Computing MDS embedding
Done. Stress: 142114952.026530
Computing Isomap embedding
Done.
```

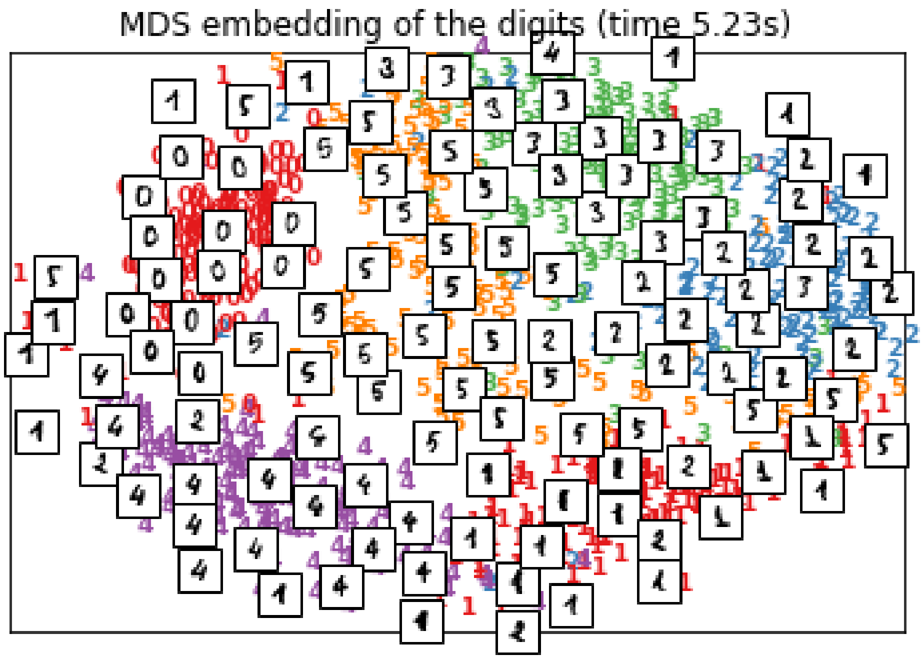


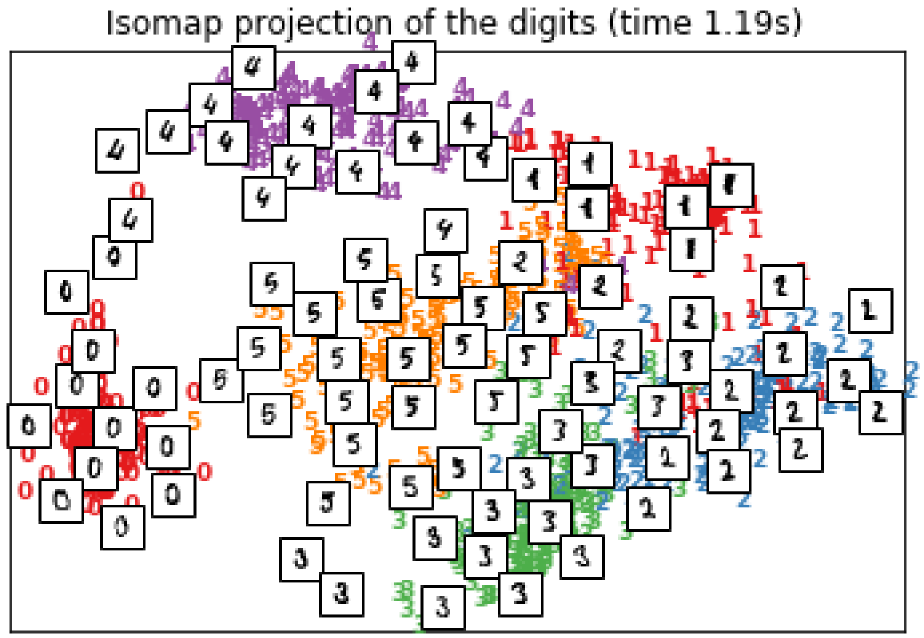
Principal Components projection of the digits (time 0.00s)



Linear Discriminant projection of the digits (time 0.01s)







2.3 Comparison of Manifold Learning methods

An illustration of dimensionality reduction on the S-curve dataset with various manifold learning methods.

For a discussion and comparison of these algorithms, see the manifold module page

For a similar example, where the methods are applied to a sphere dataset, see Manifold Learning methods on a severed sphere

Note that the purpose of the MDS is to find a low-dimensional representation of the data (here 2D) in which the distances respect well the distances in the original high-dimensional space, unlike other manifold-learning algorithms, it does not seeks an isotropic representation of the data in the low-dimensional space.

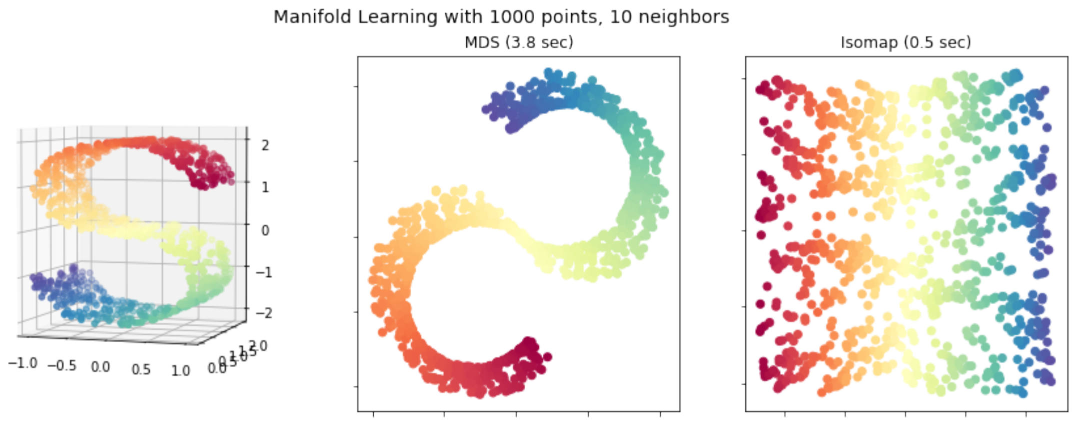
plot_compare_methods.ipynb

```
In [3]: # Author: Jake Vanderplas -- <vanderplas@astro.washington.edu>
        % matplotlib inline
        print(__doc__)
        # 载入包
        from time import time

        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        from matplotlib.ticker import NullFormatter

        from sklearn import manifold, datasets

        # pyflakes 包画 3D 图
        Axes3D

        n_points = 1000
        X, color = datasets.samples_generator.make_s_curve(n_points, random_state=0)
        n_neighbors = 10
        n_components = 2

        # 构造一个大图，每个结果作为子图
        fig = plt.figure(figsize=(15, 5))
        plt.suptitle("Manifold Learning with %i points, %i neighbors"
                     % (1000, n_neighbors), fontsize=14)
```

```
# 画三维 S 曲面
ax = fig.add_subplot(131, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap=plt.cm.Spectral)
ax.view_init(4, -72)

# S 曲面的 MDS 嵌入
t0 = time()
mds = manifold.MDS(n_components, max_iter=100, n_init=1)   # manifold.MDS
Y = mds.fit_transform(X)
t1 = time()
print("MDS: %.2g sec" % (t1 - t0))

# 低维 MDS 嵌入绘图
ax = fig.add_subplot(132)
plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral)
plt.title("MDS (%.2g sec)" % (t1 - t0))
ax.xaxis.set_major_formatter(NullFormatter())
ax.yaxis.set_major_formatter(NullFormatter())
plt.axis('tight')

# S 曲面的 Isomap 嵌入
t0 = time()
Y = manifold.Isomap(n_neighbors, n_components).fit_transform(X)   # Isomap
t1 = time()
print("Isomap: %.2g sec" % (t1 - t0))

# 低维 Isomap 嵌入绘图
ax = fig.add_subplot(133)
plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral)
plt.title("Isomap (%.2g sec)" % (t1 - t0))
ax.xaxis.set_major_formatter(NullFormatter())
ax.yaxis.set_major_formatter(NullFormatter())
plt.axis('tight')

plt.show()
```

```
Automatically created module for IPython interactive environment
```

```
MDS: 3.8 sec
Isomap: 0.5 sec
```



2.4 Manifold Learning methods on a severed sphere

An application of the different Manifold learning techniques on a spherical data-set. Here one can see the use of dimensionality reduction in order to gain some intuition regarding the manifold learning methods. Regarding the dataset, the poles are cut from the sphere, as well as a thin slice down its side. This enables the manifold learning techniques to 'spread it open' whilst projecting it onto two dimensions.

For a similar example, where the methods are applied to the S-curve dataset, see Comparison of Manifold Learning methods

Note that the purpose of the MDS is to find a low-dimensional representation of the data (here 2D) in which the distances respect well the distances in the original high-dimensional space, unlike other manifold-learning algorithms, it does not seeks an isotropic representation of the data in the low-dimensional space. Here the manifold problem matches fairly that of representing a flat map of the Earth, as with map projection

plot_manifold_sphere.ipynb

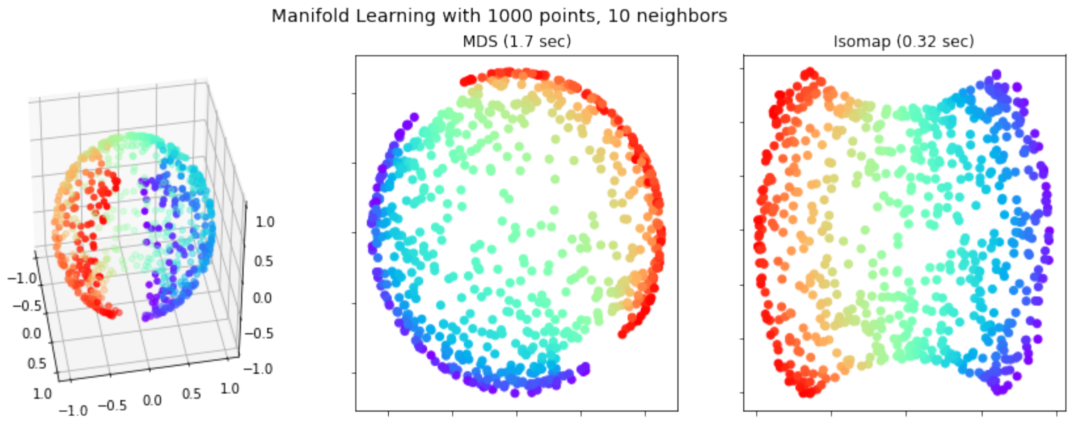
```
In [4]: # Author: Jaques Grobler <jaques.grobler@inria.fr>
        # License: BSD 3 clause
        % matplotlib inline
        print(__doc__)
        # 载入包
        from time import time
```

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import NullFormatter

from sklearn import manifold
from sklearn.utils import check_random_state

# pyflakes 包画 3D 图
Axes3D

# 变量个数
n_neighbors = 10
n_samples = 1000

# 构造球.
random_state = check_random_state(0)
p = random_state.rand(n_samples) * (2 * np.pi - 0.55)
t = random_state.rand(n_samples) * np.pi

# 球体上去掉两级
indices = ((t < (np.pi - (np.pi / 8))) & (t > ((np.pi / 8))))
colors = p[indices]
x, y, z = np.sin(t[indices]) * np.cos(p[indices]), \
    np.sin(t[indices]) * np.sin(p[indices]), \
    np.cos(t[indices])

# 构造一个大图, 每个结果作为子图
fig = plt.figure(figsize=(15, 5))
plt.suptitle("Manifold Learning with %i points, %i neighbors"
             % (1000, n_neighbors), fontsize=14)

# 画三维原始球
ax = fig.add_subplot(131, projection='3d')
ax.scatter(x, y, z, c=p[indices], cmap=plt.cm.rainbow)
ax.view_init(40, -10)
```

```
    sphere_data = np.array([x, y, z]).T

    # 球的 MDS 嵌入
    t0 = time()
    mds = manifold.MDS(2, max_iter=100, n_init=1)
    trans_data = mds.fit_transform(sphere_data).T
    t1 = time()
    print("MDS: %.2g sec" % (t1 - t0))

    # 低维 MDS 嵌入绘图
    ax = fig.add_subplot(132)
    plt.scatter(trans_data[0], trans_data[1], c=colors, cmap=plt.cm.rainbow)
    plt.title("MDS (%.2g sec)" % (t1 - t0))
    ax.xaxis.set_major_formatter(NullFormatter())
    ax.yaxis.set_major_formatter(NullFormatter())
    plt.axis('tight')

    # 球的 Isomap 嵌入
    t0 = time()
    trans_data = manifold.Isomap(n_neighbors, n_components=2)\
        .fit_transform(sphere_data).T
    t1 = time()
    print("%s: %.2g sec" % ('ISO', t1 - t0))

    # 低维 Isomap 嵌入绘图
    ax = fig.add_subplot(133)
    plt.scatter(trans_data[0], trans_data[1], c=colors, cmap=plt.cm.rainbow)
    plt.title("%s (%.2g sec)" % ('Isomap', t1 - t0))
    ax.xaxis.set_major_formatter(NullFormatter())
    ax.yaxis.set_major_formatter(NullFormatter())
    plt.axis('tight')

    plt.show()
```

```
Automatically created module for IPython interactive environment
MDS: 1.7 sec
```

```
ISO: 0.32 sec
```



Manifold Learning with 1000 points, 10 neighbors
MDS (1.7 sec)
Isomap (0.32 sec)
−1.0
−0.5
0.0
0.5
1.0

R 实现多维标度法

彭真

多维标度 (MDS) 是一组用于分析数据中的相似点或相异点的多元数据分析方法。MDS 的一个很好的特点是它允许我们在一对低维空间中表示物体对点之间的距离 (不) 相似性。换句话说，MDS 允许我们在低维空间可视化 (不) 相似性，以便进行探索和检查。MDS 是一种将多维空间的研究对象简化到低维空间进行定位、分析和归类，同时又保留对象间原始关系的数据分析方法。

1 距离

设想一下如果我们在欧氏空间中已知一些点的座标，由此可以求出欧氏距离。那么反过来，已知距离应该也能得到这些点之间的关系。这种距离可以是古典的欧氏距离，也可以是广义上的 “距离”。MDS 就是在尽量保持这种高维度 “距离” 的同时，将数据在低维度上展现出来。从这种意义上来讲，主成分分析也是多维标度分析的一个特例。

多元分析中常用有以下几种距离，即绝对值距离、欧氏距离 (euclidean)、马氏距离 (manhattan)、两项距离 (binary)、明氏距离 (minkowski)。在 R 中通常使用 disk() 函数得到样本之间的距离。MDS 就是对距离矩阵进行分析，以展现并解释数据的内在结构。

2 已有的方法

R 有很多函数来实现度量 MDS。下面的列表显示了 7 个函数以及相应 packages：

- cmdscale() (stats by R Development Core Team)
- smacofSym() (smacof by Jan de Leeuw and Patrick Mair)

- wcmdscale() (vegan by Jari Oksanen et al)
- pco() (ecodist by Sarah Goslee and Dean Urban)
- pco() (labdsv by David W. Roberts)
- pcoa() (ape by Emmanuel Paradis et al)
- dudi.pco() (ade4 by Daniel Chessel et al)

3 简单应用

在经典 MDS 中，距离是数值数据表示，将其看作是欧氏距离。在 R 中 stats 包的 cmdscale( ) 函数实现了经典 MDS。它是根据各点的欧氏距离，在低维空间中寻找各点座标，而尽量保持距离不变。

3.1 安装包

```
# 安装包
install.packages(c("vegan", "ecodist", "labdsv", "ape", "ade4", "smacof"))
```

3.2 导入包

```
# 导入包
library(vegan)
library(ecodist)
library(labdsv)
library(ape)
library(ade4)
library(smacof)
```

3.3 导入数据

我们导入 R 自带的数据集 eurodist：欧洲 21 个城市间的公路里程距离，单位为公里。

```
# 将 eurodist 转为矩阵形式
euromat = as.matrix(eurodist)

# 抽取前五个城市的距离矩阵
euromat[1:5, 1:5]
```

```
##             Athens Barcelona Brussels Calais Cherbourg
## Athens           0      3313     2963   3175      3339
## Barcelona     3313         0     1318   1326      1294
## Brussels      2963      1318        0    204       583
## Calais        3175      1326      204      0       460
## Cherbourg     3339      1294      583    460         0
```

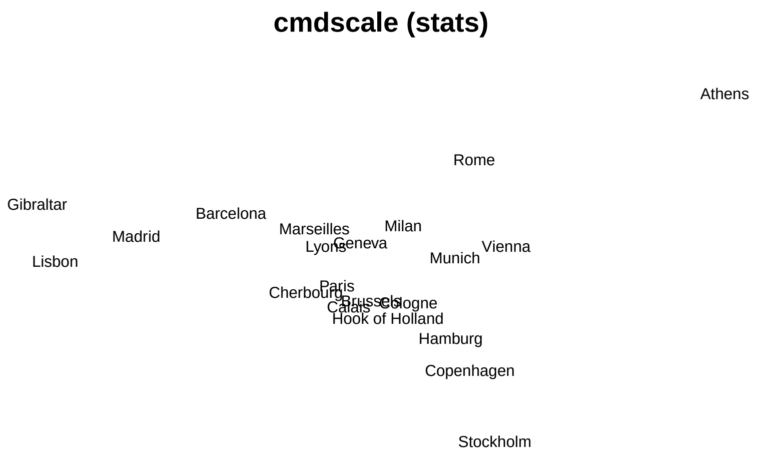
3.4 MDS with cmdscale()

执行度量 MDS 最常用的函数是 cmdscale()（R 中默认的函数）。它的一般用法有以下形式：

```
cmdscale(d, k = 2, eig = FALSE, add = FALSE, x.ret = FALSE)
```

我们的目标是应用 cmdscale() 来获得欧洲城市之间距离的可视表示。

```
# 应用 cmdscale()
mds1 = cmdscale(eurodist, k = 2)

# 画图
plot(mds1[,1], mds1[,2], type = "n", xlab = "", ylab = "", axes = FALSE,
     main = "cmdscale (stats)")
# 标记
text(mds1[,1], mds1[,2], labels(eurodist), cex = 0.7, xpd = TRUE)
```



3.5 结论

所获得的图形在二维空间中表示城市之间的距离，然而这张图与欧洲的地理图不一样：**Athens 在北方，Stockholm 在南方**。这种“反常”反映了一个事实：在二维空间满足距离约束的解不是唯一的；如果想得到符合实际的地理图，我们需要反转垂直轴。

4 参考文献

[1] http://www.gastonsanchez.com/visually-enforced/how-to/2013/01/23/MDS-in-R/