Instructions for Isomap code, Release 1

---------------------------------------

Author: Josh Tenenbaum (jbt@psych.stanford.edu) [Dijkstra code by Mark Steyvers (msteyver@psych.stanford.edu)]

Date: December 22, 2000

Website for updates: http://isomap.stanford.edu

0. Copyright notice

Isomap code -- (c) 1998-2000 Josh Tenenbaum

This code is provided as is, with no guarantees except that bugs are almost surely present. Published reports of research using this code (or a modified version) should cite the article that describes the algorithm: J. B. Tenenbaum, V. de Silva, J. C. Langford (2000). A global geometric framework for nonlinear dimensionality reduction. Science 290 (5500): 2319-2323, 22 December 2000.

Comments and bug reports are welcome. Email to jbt@psych.stanford.edu. I would also appreciate hearing about how you used this code, improvements that you have made to it, or translations into other languages.

You are free to modify, extend or distribute this code, as long as this copyright notice is included whole and unchanged.

1. Contents

This package of Matlab (version 5.3) code implements the Isomap algorithm of Tenenbaum, de Silva, and Langford (2000) [TdSL]. The contents of the file "IsomapR1.tar" are:

这个Matlab（5.3版本）代码实现了Tenenbaum，de Silva和Langford（2000）[TdSL]的Isomap算法。 文件“IsomapR1.tar”的内容是：

Isomap.m

IsomapII.m

L2\_distance.m

Readme

dfun.m

dijk.m

dijkstra.cpp

dijkstra.dll (Windows binary produced by "mex -O dijkstra") Windows“二进制”由“mex -O dijkstra”

dijkstra.m

swiss\_roll\_data.mat

The data file "swiss\_roll\_data" contains the input-space coordinates ("X\_data") and the known low-dimensional manifold coordinates ("Y\_data") for 20,000 data points on the Swiss roll. 数据文件“swiss\_roll\_data”包含瑞士卷上20,000个数据点的输入空间坐标（“X\_data”）和已知的低维歧管坐标（“Y\_data”）。

Separately available at isomap.stanford.edu is a large data file of synthetic face images, "face\_data.mat.Z". The face data file contains images of 698 faces ("images"), the projections of those faces onto the first 240 (scaled) principal components ("image\_pcs"), and the known pose and lighting parameters for each face ("poses", "lights").

isomap.stanford.edu分别提供了一个合成人脸图像“face\_data.mat.Z”的大型数据文件。 脸部数据文件包含698个脸部（“图像”）的图像，这些脸部投影到第一个240（缩放）主要成分（“image\_pcs”）上，以及每个脸部的已知姿势和照明参数（“姿势”“灯”）。

2. Getting started (Isomap.m)

Isomap.m implements the basic version of the algorithm described in [TdSL]. The algorithm takes as input the distances between points in a high-dimensional observation space, and returns as output their coordinates in a low-dimensional embedding that best preserves their intrinsic geodesic distances. Isomap算法，实现了基本版] [ TDSL。该算法以高维观测空间中的点之间的距离作为输入，并在低维嵌入中输出它们的坐标，最好保留其内在测地距离。

Try it out on N=1000 points from the "Swiss roll" data set:

>> load swiss\_roll\_data

>> D = L2\_distance(X\_data(:,1:1000), X\_data(:,1:1000), 1);

To run Isomap with K = 7 neighbors/point, and produce embeddings

in dimensions 1, 2, ..., 10, type these commands:

>> options.dims = 1:10;

>> [Y, R, E] = Isomap(D, 'k', 7, options);

The other options for the code are explained in the header of Isomap.m. The only subtle option is "option.comp", which specifies which connected component to embed in the final step of the algorithm when more than one component has been detected in the neighborhood graph. The default is to embed the largest component.

This code should work reasonably well for data sets with 1000 or fewer points. For larger data sets, consider using the advanced code described below.

3. Advanced code (IsomapII.m)

IsomapII.m implements a more advanced version of the algorithm that exploits the sparsity of the neighborhood graph in computing shortest-path distances, and can also exploit the redundancy of the distances in constructing a low-dimensional embedding.

IsomapII uses Dijkstra's algorithm to compute graph distances. IsomapII works optimally when the file "dijkstra.cpp" (which uses Fibonacci heap data structures) has been compiled (with the command "mex -O dijkstra.cpp") to produce "dijkstra.dll". If IsomapII can't find a file called dijkstra.dll, it will default to a much slower Matlab implementation of Dijkstra's algorithm in dijk.m.

Alternatively, setting "option.dijkstra = 0" tells IsomapII to use a Matlab implementation of Floyd's algorithm (also used in Isomap.m), which is generally faster than dijk.m for small-to-medium-size data sets but does not exploit sparsity to achieve better time and space efficiency for large data sets. Both dijk.m and the Floyd algorithm are much much slower than dijkstra.dll, so the latter should be used if at all possible. This code package includes a version of dijkstra.dll suitable for running on Windows platforms.

For very large data sets, it is impractical to store in memory a full N x N distance matrix, as Isomap produces after step 2 (computing shortest-path distances), or to calculate its eigenvectors, as Isomap does in step 3 (constructing a low-dimensional embedding). However, in many cases where the data lie on a low-dimensional manifold, the distances computed in step 2 are heavily redundant and most of them can be ignored with little effect on the final embedding. IsomapII constructs embeddings that, rather than trying to preserve distances between all pairs of points, preserve only the distances between all points and a subset of "landmark" points.

The user specifies which points to use as landmarks (in the options.landmarks field). Setting "options.landmarks = 1:N" (i.e. the entire data set) makes step 3 of IsomapII equivalent to classical multidimensional scaling (MDS); this is just as in Isomap.m, and it is the default mode for IsomapII. Choosing a much smaller set of landmarks (e.g. by sampling randomly, or by using some subset of the data that is known a priori to be representative) can often be quite a good approximation. Try this version of the Swiss roll example with N=1000 data points but only 50 (random) landmark points:

>> load swiss\_roll\_data

>> D = L2\_distance(X\_data(:,1:1000), X\_data(:,1:1000), 1);

>> options.dims = 1:10;

>> options.landmarks = 1:50;

>> [Y, R, E] = IsomapII(D, 'k', 7, options);

We do not know of any prior studies of the use of landmark points with classical MDS, and we have only just begun to explore this approach as an extension to Isomap. A more detailed discussion of the use of landmark points in Isomap and classical MDS will be presented in Steyvers, de Silva, and Tenenbaum (in preparation, at http://isomap.stanford.edu). The use of landmark points in nonmetric MDS for data visualization is discussed briefly in a paper by Buja, Swayne, Littman, and Dean ("XGvis: Interactive Data Visualization with Multidimensional scaling", J. Comp. & Graph. Statistics, http://www.research.att.com/areas/stat/xgobi/#xgvis-paper).

To facilitate working with large data sets, IsomapII can take as input

distances in one of three formats:

Mode 1: A full N x N matrix

Mode 2: A sparse N x N matrix

Mode 3: A distance function (see sample: "dfun.m")

Mode 1 is equivalent to Isomap.m (except for the use of dijkstra.dll,

which will usually be much faster than Isomap.m).

Mode 2 is designed for cases where the distances between nearby points are known, but the differences between faraway points are not known. The sparse input matrix is assumed to contain distances between each point and a set of neighboring points, from which the graph neighbors will be chosen using the standard K or epsilon methods. Any missing entries in the sparse input matrix are effectively assumed to be infinite (NOT zero) -- these represents non-neighboring pairs of points. Except for the sparsity of the input matrix, Mode 2 does not differ in any noticeable way from Mode 1.

Mode 3 is designed for cases where the input-space distances have not already been computed explicitly. The user provides a distance function, which takes as input one argument, i, and returns as output a row vector containing the input-space distances from all N points to point i. A sample distance function, "dfun.m", is provided. Note that the distance function may have to use a global variable in order to encode the information necessary to compute the appropriate distances. For example, dfun.m assumes that the coordinates of points in the high-dimensional input space are encoded in the global variable X. It then uses these coordinates to compute Euclidean distances in input space. Try this example:

>> load swiss\_roll\_data

>> global X

>> X = X\_data(:,1:1000);

>> options.dims = 1:10;

>> options.landmarks = 1:size(X,2);

>> [Y, R, E] = IsomapII('dfun', 'k', 7, options);

This should give exactly the same results (subject to numerical error and sign changes) as the first example in Section 2, but much more quickly! Using a distance function rather than a distance matrix allows IsomapII to handle much larger data sets (we have tested it up to N=20,000) than the simple Isomap code can handle. Also, note that the distance function need not be Euclidean. Depending on the application, domain-specific knowledge may be useful for designing a more sophisticated distance function (as in Fig. 1B of [TdSL]).