Wireless Communication and Mobile Networks Project Report
Dimensionality Reduction Based on Geodesic Distance

Hao Li, ID:515030910494 ; Yifan Shen, ID:515030910491

May 24, 2018

The work done by the two authors are listed below:
together: 1, 2, 3.1, 4; Hao Li: 3.4. 3.5, 3.6; Yifan Shen: 3.2, 3.3.
(actually same effort to this project)

1 Backgroud and Our Goal

Dimensionality reduction is an essential part of data preprocessing in machine learning tasks. Classic methods include Principle Component Analysis (PCA), Isometric Mapping (ISOMAP), Locally Linear Embedding (LLE), etc.

These existing approaches typically treat the local linearity of the data set as the basic assumption for their further approximation. In ISOMAP, for example, the key idea is to preserve the "true distance" between data points. In geometry, this "distance" is just the geodesic distance defined on a manifold where the data are sampled. Of course, since there is little a priori knowledge about what the actual manifold in most cases (otherwise the dimensionality reduction issue would be so much easier), it's hard to perform an exact computation of the geodesic distance between data points – and that's why the Dijkstra algorithm is used in the usual framework of ISOMAP. The idea is that, since manifolds are locally linear, so are the data if the samples are dense enough. And therefore a series of local Euclidean distances should be a good approximation for the geodesic distance since locally speaking, the Euclidean distance approximates the geodesic distance well.

But in many cases this approximation turns out to be not so satisfying. And people have been trying to work out a better method for computing or approximating the geodesic distance on a manifold for the sampled data. In our project, we focus on applying the Heat

Method, which is originally more of a method in the field of graphics, to ISOMAP. We basically want to replace he Dijkstra algorithm part with the Heat Method,the work of Crane et al.[1], which is actually dealing with the geodesic distance between given two data points, instead of approximating it simply using Euclidean distance. To accomplish that, a lot of survey, reading, discussion and coding were carried out by our little group, and most of our efforts are presented in the following content.

2 General Framework of the Heat Method

2.1 Introduction

Imagine touching a scorching hot needle to a single point on a surface. Over time heat spreads out over the rest of the domain and can be described by a function $k_{t,x}(y)$ called the heat kernel, which measures the heat transferred from a source x to a destination y after time t. A well-known relationship between heat and distance is Varadhan's formula, which says that the geodesic distance between any pair of points x, y on a Riemannian manifold can be recovered via a simple point wise transformation of the heat kernel:

$$\phi(x,y) = \lim_{t\to 0}\sqrt{-4t\log k_{t,x}(y)} \tag{2.1}$$

The intuition behind this behavior stems from the fact that heat diffusion can be modeled as a large collection of hot particles taking random walks starting at x: any particle that reaches a distant point y after a small time t has had little time to deviate from the shortest possible path. To date, however, this relationship has not been exploited by numerical algorithms that compute geodesic distance.

2.2 Algorithm

The method can be described purely in terms of operations on smooth manifolds. In order to fit the discrete properties,the algorithm has to be reformed which will be showed later. For now we will just show the basic steps to calculate the geodesic distance in the continuous time domain.

Let ∇ be the negative semi-definite Laplace-Beltrami operator acting on (weakly) differentiable real-valued functions over a manifold (M, g). The heat method consists of the following three basic steps:

1. Integrate the heat flow $\frac{\mathrm{d}u}{\mathrm{d}t} = \nabla u$ for some fixed time t.
2. Evaluate the vector field $X = -\frac{\nabla u}{|\nabla u|}$.
3. Solve the Poisson equation $\Delta\phi = \nabla \cdot X$.

The function ϕ approximates geodesic distance, approaching the true distance as t goes to zero. Note that the solution to step 3 is unique only up to an additive constant; final values simply need to be shifted such that the smallest distance is zero. Initial conditions $u_0 = \delta(x)$ (i.e., a Dirac delta) recover the distance to a single source point $x \in M$.

2.3 Time Discretization

We discretize the heat equation from step 1 of the algorithm in time using a single backward Euler step for some fixed time t. In practice, this means we simply solve the linear equation

$$(I - t\Delta)u_t = u_0 \tag{2.2}$$

over the entire domain M, where I is the identity. We can get a better understanding of solutions by considering the elliptic boundary value problem

$$\begin{aligned} (I - t\Delta)v_t &= 0 \quad \text{on } M\backslash\gamma \\ v_t &= 1 \qquad\quad \text{on } \gamma \end{aligned} \tag{2.3}$$

which for a point source yields a solution v_t equal to u_t up to a multiplicative constant. As established by Varadhan in his proof, v_t also has a close relationship with distance, namely,

$$\lim_{t\to 0} -\frac{\sqrt{t}}{2}\log v_t = \phi \tag{2.4}$$

away from the cut locus. This relationship ensures the validity of steps 2 and 3 since the transformation applied to v_t preserves the direction of the gradient.

3 Implementation of the Heat Method for Point Clouds

3.1 Introduction

One might have noticed that in the previous part, although the time discretization for the heat method is explained, no remarks are made about the spatial discretization, i.e., how all the operators, like Laplacian, gradient and divergence are computed on an unknown manifold given a discrete data set sampled from that manifold. In the paper of Crane, since it's under a graphic context, the spatial discretization is done based on the provided mesh grids of the data set. However, in terms of dimensionality reduction, as is mentioned before, there is no such a priori knowledge – the sampled data now form point clouds without any mesh-grid information. Of course, as Crane mentioned in his work, the Heat Method, in principle, can be applied to point clouds as well – he even provided a general framework for spatial discretization for point cloud and some experiment results for this. However, practice differs from theory all the time and this one is no exception. Spatial discretization of the Heat Method on point cloud data, especially in high dimensional space, can be of great trouble. In this section, we'll mainly introduce the spatial discretization framework of the Heat Method we built by survey, reading and discussion. Once the spatial discretization is done, the implementation of the whole method comes easily. This framework for spatial discretization of ours will be introduced step by step, with respect to the algorithm of the Heat Method given in the previous section.

3.2 Computing the Discrete LBO Operator (Δ)

At the very beginning of the Heat Method, a time-discretized Poisson Equation with boundary equation has to be solved to get the heat profile at time t. And to do this, we have to perform spatial discretization on the Laplace operator in that equation(or, in terms of manifolds, the Laplace-Beltrami operator) since we are working on point clouds.

One such discretization is given in the work of Mikhail Belkin et al.[2], in which the discrete LBO operation can be summarized as a matrix L whose elements are given as follows:

$$
\begin{aligned}
L_{ij} &= \frac{1}{t} l_{ij} \\
l_{ij} &= \begin{cases} 1 & \text{if } i = j \\ -\alpha e^{-\frac{||x_i - x_j||^2}{4t}} & \text{if } x_j \text{ is a neighbour of } x_j \\ 0 & \text{otherwise} \end{cases} \\
\text{where } \alpha &= \sum_{x_j,\ \text{the neighbours of } x_i} a
\end{aligned}
\tag{3.1}
$$

In practice, we find the neighbours by a simple knn algorithm, since there is no guarantee on the density of our data and thus we cannot specify an ϵ and a corresponding neighbourhood. And the parameter t in (3.1) is time for the heat kernel, which is fixed as a small constant or the square of the average of Euclidean distance between each data point and its neighbours.

3.3 Solving the Poisson Equation with Boundary Condition

With a given LBO operator on the manifold, where our data are sampled from, we can now solve the Poisson Equation (2.2) to yield the heat profile u_t at time t. This equation has an initial condition u_0, where the point whose geodesic distances to all other points are to be computed serves as the heat source and has an initial heat value, and also a boundary condition that the heat value of the heat source stays constant.

To solve this Poisson Equation with boundary condition, we figured out the following procedure by discussion.

Let L denotes the LBO matrix, n for the number of data points, and assume that point x_i is the heat source with constant heat value 1, then the Poisson Equation with boundary condition is:

$$
(I - tL)u_t = u_0 \text{where the } i\text{-th row in } u_t \text{ is 1, which equals that of } u_0 \tag{3.2}
$$

Since the $i-$th row of u_t stays constant, we notice that:

$$
\begin{aligned}
(I - tL)u_t = Au_t &= A'u'_t + a_i \\
\text{where } A = (I - tL) &= (a_1, a_2, ..., a_i, ..., a_n),\ (a_j \text{ is a column vector, } j = 1, 2, ..., n) \\
u_t &= (u_t^1, u_t^2, ..., u_t^{i-1}, 1, u_t^{i+1}, ..., u_t^n)^T \\
A' &= (a_1, a_2, ..., a_{i-1}, a_{i+1}, ..., a_n) \\
u'_t &= (u_t^1, u_t^2, ..., u_t^{i-1}, 1, u_t^{i+1}, ..., u_t^n)^T\ (u_t^j \text{ is the } j\text{-th element in } u_t)
\end{aligned}
\tag{3.3}
$$

And so by a little manipulation, the Poisson Equation with boundary condition reduces to:

$$A' u'_t = u_0 - a_i \tag{3.4}$$

Now note that although the size of A' is $n - by - (n-1)$, the rank of this matrix should be $(n-1)$, and so we can use SVD to solve this linear equation:

$$\begin{aligned} A' &= USV \qquad (\text{By } SVD) \\ USVu'_t &= u_0 - a_i \\ SVu'_t &= U^{-1} b_i \\ \text{where } b_i &= u_0 - a_i \end{aligned} \tag{3.5}$$

S is the n-by-$(n-1)$ rectangular matrix containing singular values of A'

Since the rank of A' is $(n-1)$, the last row of the singular value matrix S should be all zero, and thus (3.5) reduces to:

$$\begin{aligned} S'Vu'_t &= U^{-1} b'_i \\ \text{where } b'_i &\text{is } b_i \text{ with its last row removed} \\ \text{and } S' &\text{ is } S \text{ with its last row removed} \end{aligned} \tag{3.6}$$

And in (3.6), both S' and V should be invertible, and so we get the final solution of the Poisson Equation with boundary condition taken into account (again note that here the ith point is taken as heat source with constant heat value 1):

$$\begin{aligned} u'_t &= V^{-1} S'^{-1} U^{-1} b'_i \\ u^j_t &= \begin{cases} 1 & \text{if } j = i \\ u'^j_t & \text{if } j < i \\ u'^{j-1}_t & \text{if } j > i \end{cases} \end{aligned} \tag{3.7}$$

3.4 Computing the Gradient (∇) on a Manifold

3.4.1 General Scheme

Now that the heat profile u_t at time t is computed, according to the algorithm of the Heat Method, we now have to compute the gradient of u_t and perform normalization to it. Note that the gradient here is defined on the manifold. And to compute the gradient of u_t on the manifold which our data are sampled from, the general scheme is to first compute the gradient of u_t at each data point in the Euclidean space and the project the result onto the tangent space of each data point. So this section contains two subsection, one is for compute the gradient in Euclidean space, and the other for computing the tangent space at each data point and do projection.

3.4.2 COMPUTING THE GRADIENT IN EUCLIDEAN SPACE

According to the work of the Sayan Mukherjee et al.[3] We can compute the discrete gradient as shown below.

Before we start the algorithm we need to we state the matrices and vectors involved in the algorithm:

1. the kernel matrix K given the kernel function

$$K_{i,j} = K(x_i, x_j) = \frac{1}{(4\pi t)^{\frac{d}{2}}} e^{\frac{-|x_i - x_j|^2}{4t}} \qquad \text{for } i, j = 1, .., n \tag{3.8}$$

2. the elements of the weight matrix W given the parameter s

$$w_{i,j} = e^{\frac{-||x_i - x_j||^2}{2s^2}} \qquad \text{for } i, j = 1, .., n \tag{3.9}$$

3. the label vector computed from heat function for one point $y = (y_1, ..., y_m)^T = (u_{t1}, ..., u_{tn})^T$
4. the input train data set $M_x = [x_1 - x_n, x_2 - x_n, ..., x_{n-1} - x_n, x_n - x_n] \in R^{d \times n}$
5. $V = (v_1, v_2, ..., v_d)$ the d left eigenvectors of $M_x^T M_x$
6. $\beta_i = V^T(x_i - x_n)$ for $i = 1, .., n$
7. at iteration t we have the vector

$$\begin{aligned} &\eta_t = (\gamma_0^T, \gamma_1^T, ..., \gamma_n^T)^T \in R^{n(d+1)} \\ &\gamma_0 := \eta_t(1 : n) \\ &\gamma_i := \eta_t(n + (i-1)d + 1 : n + id) \\ &\gamma := (\gamma_1, ..., \gamma_n) \end{aligned} \tag{3.10}$$

8. at each iteration the matrix $a \in R^{m \times m}$ is defined by its components

$$a_{i,j} = w_{i,j}\phi'(y_i(k_j\gamma_0 + k_i\gamma^T(\beta_i - \beta_j))) \tag{3.11}$$

where k_i is the i-th column of the kernel matrix and

$$\phi(t) = \log(1 + e^{-t}) \tag{3.12}$$

9. at each iteration the matrix $A \in R^{m \times m}$ is defined by its components

$$A_{i,j} = w_{i,j}\phi''(y_i(k_j\gamma_0 + k_i\gamma^T(\beta_i - \beta_j))) \tag{3.13}$$

10. given the matrix a we define the vectors $b_0 = a^T y$ and

$$b_i = y_i \sum_{j=1}^{n} a_{i,j}(\beta_i - \beta_j) \qquad \text{where } i = 1, ..., m \tag{3.14}$$

11. given the matrix A we define the $m \times m$ matrix

$$K_0 = diag(Ae_m)K \qquad \text{where } e_m = (1, 1, ..., 1)T \tag{3.15}$$

12. from the matrices

$$K_1(j,l) = \sum_{i=1}^{n} A_{i,j} K(x_i, x_l)(\beta_i - \beta_j)^T \qquad \text{where } j,l = 1,..,n \tag{3.16}$$

construct the matrix

$$\tilde{K_1} = \begin{pmatrix} K_1(1,1) & \dots & K_1(1,n) \\ \vdots & \ddots & \vdots \\ K_1(n,1) & \dots & K_1(n,n) \end{pmatrix} \tag{3.17}$$

13. from the matrices

$$K_2(i,l) = \sum_{j=1}^{n} A_{i,j} K(x_j, x_l)(\beta_i - \beta_j)^T \qquad \text{where } i,l = 1,..,n \tag{3.18}$$

construct the matrix

$$\tilde{K_2} = \begin{pmatrix} K_2(1,1) & \dots & K_2(1,n) \\ \vdots & \ddots & \vdots \\ K_2(n,1) & \dots & K_2(n,n) \end{pmatrix} \tag{3.19}$$

14. from the matrices

$$B_i = \sum_{j=1}^{n} A_{i,j}(\beta_i - \beta_j)(\beta_i - \beta_j)^T \qquad \text{where } i = 1,..,n \tag{3.20}$$

construct the matrix

$$\tilde{K_3} = \begin{pmatrix} B_1K(x_1,x_1) & B_1K(x_1,x_2) & \dots & B_1K(x_1,x_n) \\ B_2K(x_2,x_1) & B_2K(x_2,x_2) & \dots & B_2K(x_2,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ B_nK(x_n,x_1) & B_nK(x_n,x_2) & \dots & B_nK(x_n,x_n) \end{pmatrix} \tag{3.21}$$

15. from the matrices

$$K_0 \qquad \tilde{K_1} \qquad \tilde{K_2} \qquad \tilde{K_3} \tag{3.22}$$

construct the matrix

$$\tilde{K} = \begin{pmatrix} K_0 & \tilde{K_1} \\ \tilde{K_2} & \tilde{K_3} \end{pmatrix} \tag{3.23}$$

With the matrices and vectors shown above we can start the algorithm. The basic steps of the algorithm are shown below:

Inputs:

$x = [x_1, x_2, ..., x_n]$ train data set

$y = [u_{t1}, u_{t2}, ..., u_{tn}]$ values of the heat function

Parameters:

s regularization parameter

ε threshold of the iteration

Algorithm:
$\eta_0 = 0$; stop=false; $t = 0$
repeat:
$u(\eta_t) = \frac{1}{m^2}(b_0^t, ...b_n^t)^T + \lambda\eta_t$;
$\nabla u(\eta_t) = \lambda I_{n(d+1)} + \frac{1}{m^2}\tilde{K}$;
$\Delta\eta_t = \nabla u(\eta_t)^{-1} u(\eta_t)$;
$\eta_{t+1} = \eta_t - \Delta\eta_t$;
$t = t + 1$;
If $||\Delta\eta_t|| < \varepsilon$ then stop=true;
until stop=true;
$\gamma_i = \eta_t(m + (i-1)d + 1 : m + id)$ for $i = 1, ..., m$;
$c_i = V\gamma_i$ for $i = 1, ..., m$;
$\nabla u_t(x) = \sum_{i=1}^{n} c_i K(x, x_i)$;

Here we have the final result of the gradient of the heat function u_t and then we could calculate the gradient of every point on the point cloud given the heat function from one point. The next step is to project it onto the tangent space.

3.4.3 Project the Gradient onto the Tangent Space

For computing the tangent space at each data point, the work of Tianhao Zhang et al.[4] is a good reference. The algorithm in their work mainly contains five steps: data preprocessing(omitted in our project), determining the neighbourhood, extracting local information, constructing alignment matrix and computing the final map for projection.

Determining the neighbourhood For each data point x_i, we find its k nearest neighbours $x_{i_j}, j = 1, 2, ..., k$. These neighbours play a major role in the approximation of the tangent space at point x_i since the approximation is based entirely on local information and optimization. These k neighbouring data of x_i forms a matrix $X_i = (x_{i_1}, ..., x_{i_k})$, where each x_{i_j} is a column vector.

Extracting local information For each data point x_i, X_i is first right multiplied with the matrix $H_k = I - e \cdot e^T / k$ where e is a column vector with k entries whose elements are all 1. And so for each column in $X_i H_k$, the mean of the neighbours are subtracted, compared with X_i. Then we do SVD on $X_i H_k$, i.e., $X_i H_k = U^i S^i V^i$. And we set V_i as the matrix made up by the d (d is the target dimension that we want the data to reduce to, and therefore it's the "assumed" dimension of the unknown manifold where the data are sampled form, and of course the dimension the tangent space at data point on that manifold) right singular vectors in V^i corresponding to the largest d singular values in S^i. And let $W_i = H_k(I - V_i V_i^T)$.

Constructing alignment matrix Form a matrix B using the neighbouring information computed in the previous step for each data point x_i in an iterative manner: $B(I_i, I_i) = B(I_i, I_i) + W_i \cdot W_i^T, i = 1, 2, ..., n$. Here the initial setting is $B = 0$, and $I_i = \{i_1, i_2, ..., i_k\}$ stores the indices of neighbours of point x_i – thus $B(I_i, I_i)$ is actually a $k - by - k$ matrix.

Computing the maps Solve the following generalized eigenvalue problem using the alignment matrix constructed previously: $XH_N BH_N X^T \alpha = \lambda XH_N X^T \alpha$, where H_N is again a matrix for removing mean except that its size is now n-by-n. Take out the$d$ eigenvalues and their

corresponding eigenvectors(one eigenvector for each eigenvalue of course). The eigenvalues are ordered in an ascending manner: $\lambda_1 < \lambda_2 < ... < \lambda_d$, and the matrix for the projection map consists of eigenvectors placed in bysame order: $A = (\alpha_1, \alpha_2, ..., \alpha_d)$. Finally, the projection is done by $X \to Y : Y = A^T X H_N$, where each column of X is a data point. And for the purpose of projecting the gradient at a certain point onto the tangent space of that point, we first compute A according to the data set X and then replace the X in the projection formula by the gradient matrix, with same form of course.

3.5 Computing the Divergence ($\nabla\cdot$) on a Manifold

For discrete divergence, different from other operators indicated before, we will use a quite simple way to estimate calculate. And this will be shown below.

From the definition of the divergence on the high dimensional space, if a function X in a d-dimension space is defined by

$$X(x_1, ..., x_d) = X_1(x_1)e_1 + ... + X_d(x_d)e_d = \sum_{i=1}^{d} X_i(x_i)e_i \tag{3.24}$$

where e_i is the normal vector on i-th dimension. Then we can obtain the divergence of X as shown below

$$\nabla \cdot X = \frac{\partial X_1(x_1)}{\partial x_1} + ... + \frac{\partial X_d(x_d)}{\partial x_d} = \sum_{i=1}^{d} \frac{\partial X_i(x_i)}{\partial x_i} \tag{3.25}$$

To obtain the discrete divergence of the function X. We first consider one dimension, say i-th dimension of the divergence, that is, to estimate $\frac{\partial X_i(x_i)}{\partial x_i}$. To achieve this we note that

$$\frac{\partial X_i(x_i)}{\partial x_i} = \lim_{x_i' \to x_i} \frac{X_i(x_i') - X_i(x_i)}{x_i' - x_i} \tag{3.26}$$

In order to obtain the value of the above expression, we use the following expression to estimate it on the point cloud

$$\frac{\partial X_i(x_i)}{\partial x_i} \approx \frac{X_i(x_i') - X_i(x_i)}{x_i' - x_i} \tag{3.27}$$

where x_i' is the i-th dimension of x' of the point which is closest to the original point x whose i-th dimension is x_i.

Then we can obtain the final estimation of the divergence of the target function:

$$\nabla \cdot X \approx \frac{X_1(x_1') - X_1(x_1)}{x_1' - x_1} + ... + \frac{X_d(x_d') - X_d(x_d)}{x_d' - x_d} = \sum_{i=1}^{d} \frac{X_i(x_i') - X_i(x_i)}{x_i' - x_i} \tag{3.28}$$

where $x' = (x_1, ..., x_d)$ is the closest point to point $x = (x_1, ..., x_d)$ and X_i is the projection of X to the i-th dimension normal vector.

3.6 Solve the Final Poisson Equation

Since there is no explicit boundary conditions here, we simply assume the LBO operator to be invertible and solve the final poisson equation below directly:

$$\Delta\phi = \nabla \cdot (-\frac{\nabla u_t}{|\nabla u_t|}) \tag{3.29}$$

$$\phi = L^{-1}[\nabla \cdot (-\frac{\nabla u_t}{|\nabla u_t|})] \tag{3.30}$$

where ϕ is the final geodesic distance between one point and others.

4 Summery

4.1 Total Steps of The Algorithm

With the theory all above, we can calculate the distance on the manifold using Heat Method and use this distance to run MDS algorithm which outputs the final result of the dimension reduction. Here we list the total steps of the all above.

1. compute the discrete LBO operator L
2. solve the poisson equation $(I - tL)u_t = u_0$ with boundary condition to get u_t
3. compute the gradient ∇u_t on a Manifold
4. project ∇u_t onto the tangent space
5. compute $X = -\frac{\nabla u_t}{|\nabla u_t|}$;
6. compute the divergence $\nabla \cdot X$
7. get the final geodesic distance $\phi = L^{-1}(\nabla \cdot X)$

4.2 Negative Results in Implementation on Matlab

We have implement the whole algorithm in MATLAB. But it turns out that the theory is still different from the practice. We have tried a simple Swiss roll in 3-dimension as a input train data set and use this algorithm to reduce the dimensionality of them. The result is quite unacceptable and it turns out that the algorithm dose not work.

Actually in manifold learning, there are some instinct reasons for negative results according to Yoshua Bengio, et al.[5] As for the manifold there are several characters that sentence the death of the manifold learning such as

Noise around the manifold Data are not exactly lying on the manifold. In the case of non-linear manifolds, the presence of noise means that more data around each pancake region will be needed to properly estimate the tangent directions of the manifold in that region.

High curvature of the manifold Local manifold learning methods basically approximate the manifold by the union of many locally linear patches. For this to work, there must be at least d close enough examples in each patch (more with noise). With a high curvature manifold, more-smaller-patches will be needed, and the number of required patches will grow exponentially with the dimensionality of the manifold such as image processing.

High intrinsic dimension of the manifold High manifold dimensionality d is hurtful because $O(d)$ examples are required in each patch and $O(r^d)$ patches (for some r depending on curvature and noise) are necessary to span the manifold.

Presence of many manifolds with little data per manifold In many real-world contexts there is not just one global manifold but a large number of manifolds which however share something about their structure. A simple example is the manifold of transformations (viewpoint, position, lighting,...) of 3D objects in 2D images. There is one manifold per object instance (corresponding to the successive application of small amounts of all of these transformations). If there are only a few examples for each such class then it is almost impossible to learn the manifold structures using only local manifold learning. However, if the manifold structures are generated by a common underlying phenomenon then a non-local manifold learning method could potentially learn all of these manifolds and even generalize to manifolds for which a single instance is observed.

4.3 Conclusion

In this project, we proposed a new method to calculate the geodesic distance used in MDS on manifold with the heat method. We mainly focus on how to implement the heat method in point cloud or discrete data set. And we implement the code on MATLAB. Also we analyzed the instinct reasons for negative result in manifold learning.

5 Acknowledgement

We mainly want to thank professor Xiaohua Tian and our senior Yinling Mao who offered us a lot of help during this whole project.

6 Reference

[1]Keenan Crane,Clarisse Weischedel, Max Wardetzky, "Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow"
[2]Mikhail Belkin, Partha Niyogi, "Laplacian Eigenmaps for Dimensionality Reduction and Data Representation"
[3]Sayan Mukherjee, Qiang Wu, "Estimation of Gradients and Coordinate Covariation in Classification"
[4]Tianhao Zhang, Jie Yang, Deli Zhao, Xinliang Ge, "Linear local tangent space alignment and application to face recognition"
[5]Yoshua Bengio, Martin Monperrus, "Non-Local Manifold Tangent Learning"