Analysis of the Kernel Spectral Curvature Clustering (KSCC) algorithm

Maria Nazari & Hung Nguyen

Department of Mathematics
San Jose State University

Abstract

High dimensional data sets are common in machine learning. The recurring problem is that the datasets are not distributed uniformly in the ambient space. The data may be located close to a union of low dimensional manifolds. Uncovering these low dimensional structures is key to tackling the tasks of clustering, dimensionality reduction, and classification for multi-manifold data. This paper focuses on the main ideas from the Kernel Spectral Curvature Clustering (KSCC) algorithm developed by G. Chen, S. Atev and G. Lerman [1, 2]. The algorithm uses kernels at two levels to allow a manifold modeling to be converted to hybrid linear modeling in an embedded space. KSCC is an extension of the spectral curvature clustering (SCC) algorithm; the main differences between the two algorithms will be discussed. We will demonstrate the efficiency of the KSCC algorithm on a range of different datasets.

Introduction

The Spectral Curvature Clustering (SCC) was developed to handle clustering a collection of multi-subspace data. The main idea is to analyze d+2 combinations of the data points and find possibility that these points come from the same subspace. The algorithm captures the curvature of the set of these points; the value(s) will be zero when all the points are located in the same subspace. The results are inputted into an affinity matrix. Once the matrix is created, spectral clustering can be applied. The algorithm’s asymptotic time complexity is $O(N^{d+2})$ [1].
In order to handle noisy data, SCC requires prior knowledge of the number and dimensions of the subspaces. SCC’s main drawback is required that all the subspaces must have the same dimension d. For example, the algorithm will perform well on the dataset shown in figure 1, but it will perform poorly for the data set in figure 2 because of the failure to comply with the equal subspace dimension requirement.

![Figure 1](image1.png)  ![Figure 2](image2.png)

**Kernel Spectral Curvature Clustering (KSCC)**

Kernel Spectral Curvature Clustering (KSCC) is designed to perform well on manifolds that are not limited to linear subspaces by using a kernel trick. Generally, a kernel is used to map data into a higher dimensional space in the assumption that the data will become more easily separated in this new dimension. The idea of KSCC is to convert multi-manifold modeling into hybrid linear modeling by using a kernel, resulting in the parametric surfaces becoming flatter [2].

When it is necessary to use terms dot products between the data points to express the hybrid linear modeling algorithm, the kernel function can be used to avoid explicit embedding [2].
In KSCC, we can replace the dot product by a kernel, represented as:

\[ k(x, y) = \langle \Phi(x), \Phi(y) \rangle, \text{ for all } x, y \in \mathbb{R}^D \]

where \( \Phi : \mathbb{R}^D \to F \) and \( F \) is a Hilbert space, a vector space closed under dot products. The kernel matrix

\[ K := \{ k(x_i, x_j) \}_{1 \leq i, j \leq N}, \text{ for any } N \text{ points } x_1, \ldots, x_n \text{ in } \mathbb{R}^D \]

is symmetric and positive semi-definite, meaning that the kernel has only positive Eigenvalues.

KSCC is equivalent to the SCC algorithm, but performed in a certain feature space.

Similarly to SCC, the KSCC computes a polar curvature \( c_p^2(l) \) for any \( d + 2 \) points in the feature space via the Kernel trick

\[ c_p^2(l) = \frac{1}{l+2} \max_{i \in \mathbb{I}} \left( \text{det}(K_{i,l} + K_{j,j} - 2K_{i,j}) \right) \Sigma_{i \in \mathbb{I}} \frac{\text{det}(K_{i,l+1})}{\prod_{j \neq i, j \in \mathbb{I}} K_{i,j}^{l+1}} \cdot K_{i,l} := (K_{ij})_{l,j \in \mathbb{I}}, \text{ where } K_{i,l} := (K_{ij})_{l,j \in \mathbb{I}} \]

The affinity matrix is \( A_p(l) = e^{-\frac{c_p^2(l)}{2\sigma^2}} \) where \( \sigma > 0 \) is a tuning parameter, and otherwise 0.

The curvature function will generate large outputs (close to 1) values for points from the same parametric surface and small values (close to 0) for points that fall on different spaces. The next step in the KSCC algorithm is calculating the pairwise weights \( W = AA' \). Spectral clustering is then applied to find K clusters through the process of iterative sampling by using total kernel least square errors, \( e^2_{KLS} \). The function sums least square errors of d-flats approximations to the clusters [2].

**The Kernel Spectral Curvature Clustering (KSCC) Scheme**

The main scheme of the kernel spectral curvature clustering is summarized below:

1. Compute the polar curvature of \( d + 2 \) tuples of \( c \) subsets for the dataset, creating a an \( N^{c(d+2)} \) tensor

2. Flatten tensor into a matrix A
3. Compute weights $W = AA'$

4. Apply spectral clustering to these weights

**Kernel Spectral Curvature Clustering (KSCC) Algorithm Important Details**

- **Input:** Dataset $X$, kernel matrix $K$, maximal dimension $d$ (in feature space), number of manifolds $K$, and number of sampled $(d+2)$-tuples $c$ (The default value is $100K$).
- **Output:** $K$ disjoint clusters $C_1, \ldots, C_K$.
- KSCC performs faster than SCC in the embedded spaces (with large dimensions).
- However, there are important issues in order to successfully apply KSCC.
  1. The choice of kernel function
  2. The dimension of the flats in the feature space is often quiet large. For example, two circles in $R^3$ will be mapped into $R^8$
  3. A more careful examination of the situation when the data is corrupted with noise.
  4. Carefully examination of the performance on data set contaminated with outliers.

**How to pick a Kernel:**

There is not a clear mechanism in place to help guide the selection. However, Tom Howley and Michael Madden have proposed an automatic kernel selection in their paper “An Evolutionary Approach to Automatic Kernel Construction [3].”

The solution to selecting the proper kernel depends on what we are trying to model. For example, radial basis functions allows to pull out circles and linear kernels allows to pull out lines.

There are many different kernels available for use, some of the most commonly used include:

- **Linear:** $K(x_i, x_j) = x_i^T x_j$
- **The polynomial:** $K(x_i, x_j) = (x_i^T x_j + c)^d$ where $c \geq 0$
• Spherical Kernel: \( K(x_i, x_j) = x_i \cdot x_j + \|x_i\|^2 * \|x_j\|^2 \)

When it is not clear which kernel should be selected, the default kernel to choose is the Gaussian kernel.

\[ k(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) \]

It should be noted that sigma plays an important role in how well the kernel performs, and the parameter needs to be carefully tuned. A large sigma value may cause the exponential to lose its non-linear power by behaving linearly. A small sigma value may cause the kernel function to lose regularization. Fine tuning the parameter can become tedious and cumbersome [4].

**Experiments Performed**

In this paper, the KSCC algorithm was applied to three datasets. The Gaussian (also known as RBF) kernel was used to run the KSCC algorithm in all three cases.

**Crescent & Full Moon dataset**

For the crescent & full moon data set (figure 3), KSCC successfully split the two clusters using sigma=1.5 (figure 4). The algorithm had not trouble separating the two clusters with a with a perfect accuracy rate.
**Half Kernel dataset**

The KSCC algorithm was then applied to the half kernel dataset (figure 5). After several attempts at running KSCC with different sigma values (figure 6), the algorithm came very close to accurately separate the two clusters. Using sigma= 1.62542221, KSCC was able to achieve about 90 percent accuracy in clustering the two groups.

![Figure 5: Raw data](image)

![Figure 6: Clusters obtained by KSCC using different sigmas](image)

**Spiral dataset**

Finally, the KSCC algorithm was applied to a spiral data set represented by figure 7. After several attempts at fine tuning the parameter sigma and the d value (representing the dimensional
plane in the feature space induced by the kernel Gaussian kernel applied, we were unsuccessful at separating the two natural clusters using KSCC as shown in figure 8.

![Figure 7: Raw Data](image1)

![Figure 8: Clusters obtained by KSCC](image2)

When using the Gaussian kernel, it can be very difficult to predict the correct value for $d$ used in the KSCC algorithm.

**Conclusion**

In this paper, we analyzed KSCC and found the experiment results for using the algorithm on different datasets. Although KSCC is a powerful tool for multi-manifold modeling, we found that KSCC can be successful by using the proper kernel and knowing the dimension in the feature space. In our future work, we plan on exploring the efficiencies of using other kernels, especially on the spiral dataset.
References


Data obtained from: http://www.mathworks.com/matlabcentral/fileexchange/41459-6-functions-for-generating-artificial-datasets/content/datasetsdemo.m
Appendix

Matlab Code:

**Crescent and Moon Script:**

```matlab
halfmoon = crescentfullmoon();
scatter(halfmoon(:,1), halfmoon(:,2), dotsize, halfmoon(:,3)); axis equal;
title('Crescent & Full Moon');
```

**Crescent Full Moon Function:**

```matlab
function data = crescentfullmoon(N, r1, r2, r3)
    if nargin< 1
        N = 1000;
    end
    if mod(N,4) ~= 0
        N = round(N/4) * 4;
    end
    if nargin< 2
        r1 = 5;
    end
    if nargin< 3
        r2 = 10;
    end
    if nargin< 4
        r3 = 15;
    end

    N1 = N/4;
    N2 = N-N1;

    phi1 = rand(N1,1) * 2 * pi;
    R1 = sqrt(rand(N1, 1));
    moon = [cos(phi1) .* R1 * r1 sin(phi1) .* R1 * r1 zeros(N1,1)];

    d = r3 - r2;
    phi2 = pi + rand(N2,1) * pi;
    R2 = sqrt(rand(N2,1));
    crescent = [cos(phi2) .* (r2 + R2 * d) sin(phi2) .* (r2 + R2 * d) ones(N2,1)];

    data = [moon; crescent];
```

**Gaussian Function:**

```matlab
function K = rbfK(X)
n=size(X,1);
X_tilde=X-repmat(mean(X,1),n,1);
for i=1:n
    for j=i+1:n
        dist(i,j)=sqrt(sum((X_tilde(i,:)-X_tilde(j,:)).^2));
    end
end
% dist=dist + dist';
% mean_sim=mean(sort(dist,1),2)
% sigma =mean_sim(20);
```
sigma=1.5
K=zeros(n,n);
for i=1:n
for j=i:n
    K(i,j)= RBF(X_tilde(i,:),X_tilde(j,:),sigma);
end
end
K=K + K'-diag(diag(K));% compute K from the upper triangular K matrix.

K=rbfK(halfmoon);
[sampleLabels,averageL2Error] = ksc(halfmoon, K, 1, 2);
gcplot(halfmoon,sampleLabels)

**Crescent and Moon Script:**
Used following sigmas for RBF function (posted above):
TL: simga= 1.65
TR: sigma=1.625421
BL: sigma=1.62545
BR: sigma=1.6254221

K = rbfK(halfkernal);
[sampleLabels,averageL2Error] = ksc(halfkernal, K, 5, 2);
gcplot(halfkernal,sampleLabels);title('Clusters Obtained by KSCC')

**Halfkernal Script:**

halfkernal = halfkernel();
sscatter(halfkernal(:,1), halfkernal(:,2), dotsize, halfkernal(:,3)); axis equal;
title('Half-kernel');

**Halfkernal Function:**
function data = halfkernel(N, minx, r1, r2, noise, ratio)

if nargin< 1
    N = 1000;
end
if mod(N,2) ~= 0
    N = N + 1;
end
if nargin< 2
    minx = -20;
end
if nargin< 3
    r1 = 20;
end
if nargin< 4
    r2 = 35;
end
if nargin< 5
    noise = 4;
end
if nargin< 6
    ratio = 0.6;
phi1 = rand(N/2,1) * pi;
inner = [minx + r1 * sin(phi1) - .5 * noise + noise * rand(N/2,1) r1 * ratio * cos(phi1) - .5 * noise + noise * rand(N/2,1) ones(N/2,1)];

phi2 = rand(N/2,1) * pi;
outer = [minx + r2 * sin(phi2) - .5 * noise + noise * rand(N/2,1) r2 * ratio * cos(phi2) - .5 * noise + noise * rand(N/2,1) zeros(N/2,1)];

data = [inner; outer];
end

Spiral Data
Used following sigmas in RBF function:
Sigma=.93
Sigma=.8

K = rbfK(spirals2);
[sampleLabels,averageL2Error] = kscc(spirals2, K, 2);
geplot(spirals2,sampleLabels);title('Clusters Obtained by KSCC')

Data Script:
spirals2 = twospirals(5000,300,60,0);
scatter(spirals2(:,1), spirals2(:,2), dotsize, spirals2(:,3)); axis equal;
title('Two spirals');

Spiral Data Script:
function data = twospirals(N, degrees, start, noise)
% Generate "two spirals" dataset with N instances.
% degrees controls the length of the spirals
% start determines how far from the origin the spirals start, in degrees
% noise displaces the instances from the spiral.
% 0 is no noise, at 1 the spirals will start overlapping
if nargin< 1
    N = 2000;
end
if nargin< 2
    degrees = 570;
end
if nargin< 3
    start = 90;
end
if nargin< 4
    noise = 0.2;
end
deg2rad = (2*pi)/360;
start = start * deg2rad;

N1 = floor(N/2);
N2 = N-N1;
n = start + sqrt(rand(N1,1)) * degrees * deg2rad;
d1 = [-cos(n).*n + rand(N1,1)*noise sin(n).*n+rand(N1,1)*noise zeros(N1,1)];

n = start + sqrt(rand(N1,1)) * degrees * deg2rad;
d2 = [cos(n).*n+rand(N2,1)*noise -sin(n).*n+rand(N2,1)*noise ones(N2,1)];

data = [d1;d2];
end