Homework Assignment 3
Due by end of the day Dec 7th via canvas

SDS 385 Statistical Models for Big Data
Answer

1. **Spectral Clustering**[1] In class we have seen k-means for estimating GMM's. Since your professor will not get a chance to actually cover her favorite clustering method aka spectral clustering, methinks this is a great opportunity to introduce you to it. There is a class of clustering algorithms, called spectral clustering algorithms, which has recently become quite popular. Many of these algorithms are quite easy to implement and perform well on certain clustering problems compared to more traditional methods like k-means. In this problem, we will try to develop some intuition about why these approaches make sense and implement one of these algorithms.

   Before beginning, we'll review a few basic linear algebra concepts you may find useful for some of the problems.

   - If A is a matrix, it has an v with eigenvalue λ if $Av = \lambda v$.
   - For any $m \times m$ symmetric matrix A, the *Singular Value Decomposition* of A yields a factorization of A into
     $$A = USU^T$$
     where U is an $m \times m$ orthogonal matrix (meaning that the columns are pairwise orthogonal). and $S = diag(|\lambda_1|, |\lambda_2|, \ldots, |\lambda_m|)$ where the λ_i are the eigenvalues of A.

   Given a set of m datapoints $x_1, \ldots, x_m$, the input to a spectral clustering algorithm typically consists of a matrix, A, of pairwise similarities between datapoints often called the *affinity matrix*. The choice of how to measure similarity between points is one which is often left to the practitioner. A very simple affinity matrix can be constructed as follows:

   $$A(i,j) = A(j,i) = \begin{cases} 1 & \text{if } d(x_i, x_j) < \Theta \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

   where $d(x_i, x_j)$ denotes the Euclidean distance between points x_i and x_j.

   The general idea of spectral clustering is to construct a mapping of the datapoints to an eigenspace of A with the hope that points are well separated in this eigenspace so that something simple like k-means applied to these new points will perform well.

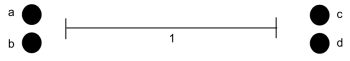


Figure 1: Simple dataset

[1] This problem was designed in collaboration with Jon Huang

As an example, consider forming the affinity matrix for the dataset in Figure 1 using Equation 1 with $\Theta = 1$. We have that

$$A = \left[\begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 0 & 0 \\ b & 1 & 1 & 0 & 0 \\ c & 0 & 0 & 1 & 1 \\ d & 0 & 0 & 1 & 1 \end{array}\right]$$

Now for this particular example, the clusters $\{a, b\}$ and $\{c, d\}$ show up as nonzero blocks in the affinity matrix. This is, of course, artificial, since we could have constructed the matrix A using any ordering of $\{a, b, c, d\}$. For example, another possibile affinity matrix for A could have been:

$$\tilde{A} = \left[\begin{array}{c|cccc} & a & c & b & d \\ \hline a & 1 & 0 & 1 & 0 \\ c & 0 & 1 & 0 & 1 \\ b & 1 & 0 & 1 & 0 \\ d & 0 & 1 & 0 & 1 \end{array}\right]$$

The key insight here is that the eigenvectors of matrices A and $\tilde{A}$ have the same entries (just permuted). The eigenvectors with nonzero eigenvalue of A are: $e_1 = (.7, .7, 0, 0)^T$, $e_2 = (0, 0, .7, .7)$. And the nonzero eigenvectors of $\tilde{A}$ are: $\tilde{e}_1 = (.7, 0, .7, 0)^T$, $\tilde{e}_2 = (0, .7, 0, .7)$. Spectral clustering embeds the original datapoints in a new space by using the coordinates of these eigenvectors. Specifically, it maps the point x_i to the point $(e_1(i), e_2(i), \ldots, e_k(i))$ where $e_1, \ldots, e_k$ are the top k eigenvectors of A. We refer to this mapping as the *spectral embedding*.

Algorithm description

Frequently, the affinity matrix is constructed as

$$A_{ij} = \begin{cases} 1 & \text{If } i, j \text{ are amongst } k \text{ nearest neighbors of each other} \\ 0 & \text{Otherwise} \end{cases} \tag{2}$$

The best that we can hope for in practice is a near block-diagonal affinity matrix. It can be shown in this case, that after projecting to the space spanned by the top k eigenvectors, points which belong to the same block are close to each other in a euclidean sense. We won't try to prove this, but using this intuition, you will implement one (of many) possible spectral clustering algorithms. This particular algorithm is described in

```
On Spectral Clustering: Analysis and an algorithm
Andrew Y. Ng, Michael I. Jordan, Yair Weiss (2001)
```

We won't try to justify every step, but see the paper if you are interested. The steps are as follows:

- Construct an affinity matrix A using Equation 2.
- Symmetrically 'normalize' the rows and columns of A to get a matrix N: such that $N(i, j) = \frac{A(i,j)}{\sqrt{d(i)d(j)}}$, where $d(i) = \sum_k A(i, k)$.

- Construct a matrix Y whose columns are the first k eigenvectors of N.
- Normalize each row of Y such that it is of unit length.
- Cluster the dataset by running k-means on the set of spectrally embedded points, where each row of Y is a datapoint.

(a) Generate a dataset with 10000 points, with 5000 coming from a Gaussian centered at $\mu_1(i) = 3/\sqrt{p}$ with $\Sigma_1 = I_p$ and the rest from a mean $\mu_2(i) = -3/\sqrt{p}$, for $i = 1, \ldots, p$, gaussian with $\Sigma_2 = I_p$. Create a k-nearest neighbor graph from this dataset and do Spectral clustering. For $p \in \{250, 500, 1000, 2000\}$, plot the time taken and the clustering accuracy of the following algorithms, averaged over 10 randomly generated datasets. You just have to write your own code for Spectral clustering, you can use available software like packages for kdtree for the rest. There are 3 algorithms, *Exact*, *JL+Exact*, *JL+KDtrees*. Use $k = 5$. You can also try out different k, its interesting to try this because too small or too large k can lead to a bad clustering accuracy. Remember that you can calculate the clustering accuracy because you generated the data and hence know the "latent" ground truth memberships.

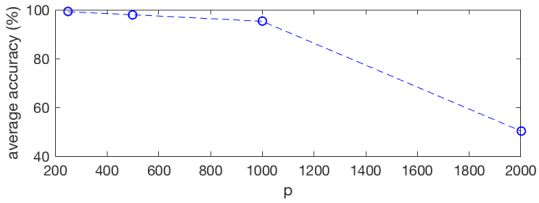
i. (Exact) Use the brute force k-nearest neighbor algorithm. If this is taking too long, you may omit the results, but please write explicitly how long it took, e.g. "my k-nn graph took over yyy hours to build when $p = xxx$ and so I gave up."

ii. (JL+Exact, JL+KDtrees) Use the Johnson Lindenstrauss lemma to reduce the dimensionality of the data. Remember, for n points JL lets you project the datapoints into a $O(\log n/\epsilon^2)$ dimensional space with a multiplicative distortion of ϵ of the distances. Try out different ϵ values to generate the curves. For small ϵ, you would have higher accuracy and longer processing time, whereas for larger ϵ you would have lower accuracy and less computation time. Now use the KDtree algorithm and exact k-nearest neighbors to build the knn tree. There is a builtin function in Matlab using knnsearch which has an option of using the KDtree.

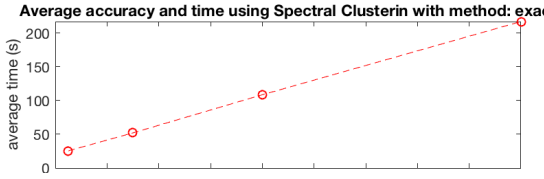
Solution. See "p1_spectral-cluster.html" for codes and results. The first 3 plots are average accuracy and time using spectral clustering with different methods to build affinity matrix: exact, JL_exact, JL_KDtree. For average accuracy, for all the four methods, as the dimension p increases, the accuracy of spectral clustering decreases. For running time, as the dimension p increases, brutal exact uses longer time; while that of JL_exact and JL_KDtree keep similar despite of different dimensions.

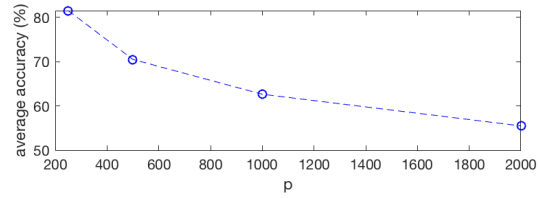
Among the four methods: brutal method takes longest time but results in highest accuracy; dimension reduction methods are very efficient but the accuracy decreases. We can choose small ϵ to get better accuracy with longer time. (See next two plots)

The following two plots show the effect of ϵ of JL related methods. As ϵ decreases small, the approximation of distances are accurate, so corresponding average accuracy increase while the average running time increase (but still keep similar for different p since the dimension is the same after projection).

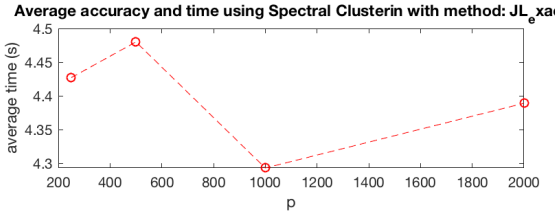


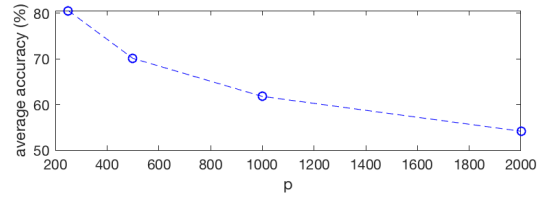
Average accuracy and time using Spectral Clusterin with method: exact



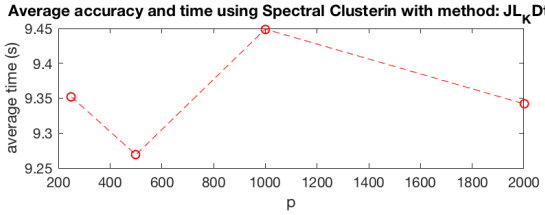


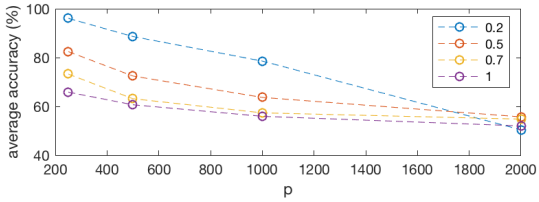
Average accuracy and time using Spectral Clusterin with method: JL$_e$xact



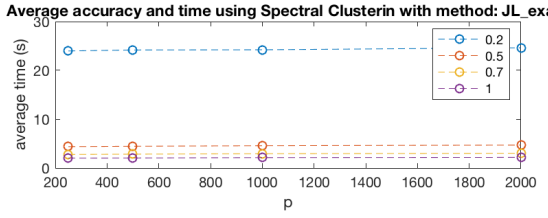


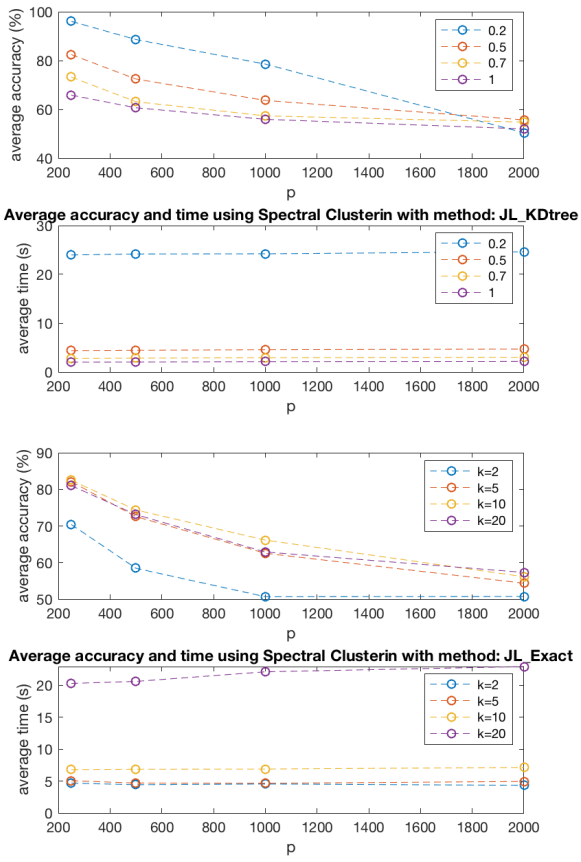
Average accuracy and time using Spectral Clusterin with method: JL$_K$Dtree





Average accuracy and time using Spectral Clusterin with method: JL_exact





The last plot shows the effect of k with JL_exact as an example. When k is too small ($k = 2$), the accuracy decreases. When k is too large ($k = 20$), the running time increases but the accuracy decreases. ■

2. We are going to learn and implement the power method in this problem. Let the eigenvalues of a square symmetric matrix $A \in \mathbb{R}^{n \times n}$ be given by $|\lambda_1| > |\lambda_2| > \ldots$. We will assume that the eigenvalues are all different for convenience of analysis. Start with some vector $\boldsymbol{q}^{(0)}$. Now compute the following:

$$\boldsymbol{z}^{(k)} = A\boldsymbol{q}^{(k-1)} \tag{3}$$

$$\boldsymbol{q}^{(k)} = \frac{\boldsymbol{z}^{(k)}}{\|\boldsymbol{z}^{(k)}\|} \tag{4}$$

$$\nu^{(k)} = (\boldsymbol{q}^{(k)})^T A\boldsymbol{q}^{(k)} \tag{5}$$

   (a) Prove that for $k > 1$,

$$\boldsymbol{q}^{(k)} = \frac{A^k\boldsymbol{q}^{(0)}}{\|A^k\boldsymbol{q}^{(0)}\|}.$$

   You can use induction to do this.

Proof. For $k=1$, $\boldsymbol{z}^{(1)} = A\boldsymbol{q}^{(0)}$, $\boldsymbol{q}^{(1)} = \frac{A\boldsymbol{q}^{(0)}}{\|A\boldsymbol{q}^{(0)}\|}$.
For $k>1$, suppose $\boldsymbol{q}^{(k-1)} = \frac{A^{k-1}\boldsymbol{q}^{(0)}}{\|A^{k-1}\boldsymbol{q}^{(0)}\|}$, then $\boldsymbol{z}^{(k)} = A\boldsymbol{q}^{(k-1)} = \frac{A^k\boldsymbol{q}^{(0)}}{\|A^{k-1}\boldsymbol{q}^{(0)}\|}$, hence,

$$\boldsymbol{q}^{(k)} = \frac{\boldsymbol{z}^{(k)}}{\|\boldsymbol{z}^{(k)}\|} = \frac{\frac{A^k\boldsymbol{q}^{(0)}}{\|A^{k-1}\boldsymbol{q}^{(0)}\|}}{\|\frac{A^k\boldsymbol{q}^{(0)}}{\|A^{k-1}\boldsymbol{q}^{(0)}\|}\|} = \frac{A^k\boldsymbol{q}^{(0)}}{\|A^k\boldsymbol{q}^{(0)}\|}$$

By induction, the equation holds for all $k>1$. □

(b) Now let the eigenvectors of A be $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and let

$$\boldsymbol{q}^{(0)} = \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i, \quad \boldsymbol{y}^{(k)} = \sum_{i=2}^{n} \frac{\alpha_i}{\alpha_1} \left(\frac{\lambda_i}{\lambda_1}\right)^k \boldsymbol{x}_i.$$

Show that

$$\|\boldsymbol{q}^{(k)} - \langle \boldsymbol{q}^{(k)}, \boldsymbol{x}_1\rangle \boldsymbol{x}_1\| \le C \left|\frac{\lambda_2}{\lambda_1}\right|^k$$

Proof. From (a), we have

$$A^k\boldsymbol{q}^{(0)} = A^k(\sum_{i=1}^{n}\alpha_i\boldsymbol{x}_i) = A^{k-1}(\sum_{i=1}^{n}\alpha_i A\boldsymbol{x}_i) = A^{k-1}(\sum_{i=1}^{n}\alpha_i\lambda_i\boldsymbol{x}_i) = \sum_{i=1}^{n}\alpha_i\lambda_i^k\boldsymbol{x}_i$$

$$\boldsymbol{q}^{(k)} = \frac{A^k\boldsymbol{q}^{(0)}}{\|A^k\boldsymbol{q}^{(0)}\|} = \frac{\sum_{i=1}^{n}\alpha_i\lambda_i^k\boldsymbol{x}_i}{\|\sum_{i=1}^{n}\alpha_i\lambda_i^k\boldsymbol{x}_i\|}$$

Plug in $\boldsymbol{q}^{(k)}$ and since $\{\boldsymbol{x}_i\}$ are eigenvectors, then $\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle = \delta_{ij}$, hence

$$\langle \boldsymbol{q}^{(k)}, \boldsymbol{x}_1\rangle = \frac{\langle\sum_{i=1}^{n}\alpha_i\lambda_i^k\boldsymbol{x}_i, \boldsymbol{x}_1\rangle}{\|\sum_{i=1}^{n}\alpha_i\lambda_i^k\boldsymbol{x}_i\|} = \frac{\alpha_1\lambda_1^k}{\|\sum_{i=1}^{n}\alpha_i\lambda_i^k\boldsymbol{x}_i\|}$$

Again, since $\{x_i\}_{i=1}^n$ are orthonormal, $\forall c_i \in \mathbb{R}$, we have $\|\sum_{i=1}^n c_i\boldsymbol{x}_i\| = \sqrt{\|\sum_{i=1}^n c_i\boldsymbol{x}_i\|_2} = \sqrt{\sum_{i=1}^n c_i^2}$, then

$$\|\boldsymbol{q}^{(k)} - \langle \boldsymbol{q}^{(k)}, \boldsymbol{x}_1\rangle \boldsymbol{x}_1\| = \frac{\|\sum_{i=2}^{n}\alpha_i\lambda_i^k\boldsymbol{x}_i\|}{\|\alpha_1\lambda_1^{(k)}\boldsymbol{x}_1 + \sum_{i=2}^{n}\alpha_i\lambda_i^k\boldsymbol{x}_i\|} = \frac{\|\sum_{i=2}^{n}\frac{\alpha_i}{\alpha_1}(\frac{\lambda_i}{\lambda_1})^k\boldsymbol{x}_i\|}{\|\boldsymbol{x}_1 + \sum_{i=2}^{n}\frac{\alpha_i}{\alpha_1}(\frac{\lambda_i}{\lambda_1})^k\boldsymbol{x}_i\|}$$
$$= \frac{\sqrt{\sum_{i=2}^{n}(\frac{\alpha_i}{\alpha_1})^2|\frac{\lambda_i}{\lambda_1}|^{2k}}}{\sqrt{1+\sum_{i=2}^{n}(\frac{\alpha_i}{\alpha_1})^2|\frac{\lambda_i}{\lambda_1}|^{2k}}} \le \sqrt{\sum_{i=2}^{n}(\frac{\alpha_i}{\alpha_1})^2}\cdot\left|\frac{\lambda_2}{\lambda_1}\right|^k \triangleq C\left|\frac{\lambda_2}{\lambda_1}\right|^k$$

where the last inequality is from (1) denominator: $\sqrt{1+p} \ge 1, \forall p \ge 0$; (2) numerator: since $\lambda_2 \ge \lambda_3 \ge \cdots \ge \lambda_n$, we have $\forall i > 2$, $|\frac{\lambda_i}{\lambda_1}|^k \le |\frac{\lambda_2}{\lambda_1}|^k$; (3) Let $C = \sqrt{\sum_{i=2}^{n}(\frac{\alpha_i}{\alpha_1})^2}$, it is a constant for any fixed $\boldsymbol{q}^{(0)}$. □

3. Last, but not the least, we will learn kernel PCA to deal with non-linear decision boundaries. Recall PCA? There you first centered your data to get $\tilde{X}$, computed covariance matrix, and then compute top K eigenvectors V_k of this and project a datapoint $\tilde{x}$ to get $\tilde{x}^T V_k$. Now we will directly compute these projections .

(a) Assume that you are mapping the datapoints to a different feature space $\phi(\boldsymbol{x}) \in \mathbb{R}^N$ and $\sum_i \phi(\boldsymbol{x}_i) = 0$. While we will not do this explicitly, let us follow through the steps of PCA applied on this new feature space. Create a new matrix $\phi(X) \in \mathbb{R}^{n \times N}$. Let $\boldsymbol{v}$ be the first eigenvector of the covariance matrix in this feature space, i.e.

$$\sum_{i=1}^{n} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T \boldsymbol{v} = \lambda \boldsymbol{v}. \tag{6}$$

Show that $\boldsymbol{v}$ can be written as

$$\boldsymbol{v} = \phi(X)^T \boldsymbol{a} \tag{7}$$

So finding $\boldsymbol{v}$ boils down to finding $\boldsymbol{a}$.

Proof. Let $a_i = \frac{\phi(\boldsymbol{x}_i)^T \boldsymbol{v}}{\lambda}, \forall i \in [n]$ and $\boldsymbol{a} = [a_i, \ldots, a_n]^T$, then from Eq (6),

$$\boldsymbol{v} = \frac{1}{\lambda} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T \boldsymbol{v} = \sum_{i=1}^{n} a_i \phi(\boldsymbol{x}_i) = \phi(X)^T \boldsymbol{a}$$

since $\phi(X)^T = [\phi(\boldsymbol{x}_1) \;\; \phi(\boldsymbol{x}_2) \;\; \cdots \;\; \phi(\boldsymbol{x}_n)] \in \mathbb{R}^{N \times n}$. □

(b) Now show that if you could get $\boldsymbol{a}$, the projection of the data (in the new space) on this direction is given by:

$$\Phi(X)\boldsymbol{v} = K\boldsymbol{a},$$

where $K(i,j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$. Now we will get $\boldsymbol{a}$.

Proof. Since from (a), $\boldsymbol{v} = \phi(X)^T \boldsymbol{a}$, then

$$\phi(X)\boldsymbol{v} = \phi(X)\phi(X)^T \boldsymbol{a} = K\boldsymbol{a}$$

where $K = \phi(X)\phi(X)^T$. Since

$$\phi(X) = \begin{bmatrix} — & \phi(\boldsymbol{x}_1)^T & — \\ — & \phi(\boldsymbol{x}_2)^T & — \\ & \vdots & \\ — & \phi(\boldsymbol{x}_n)^T & — \end{bmatrix}_{n \times N} \qquad \phi(X)^T = \begin{bmatrix} | & | & & | \\ \phi(\boldsymbol{x}_1) & \phi(\boldsymbol{x}_2) & \ldots & \phi(\boldsymbol{x}_n) \\ | & | & & | \end{bmatrix}_{N \times n}$$

we have $K(i,j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$. □

(c) Now plug in Eq (7) to Eq (6), and left multiply by $\phi(\boldsymbol{x}_k)^T$ to get:

$$K\boldsymbol{a} = \lambda \boldsymbol{a}.$$

You can assume that K is invertible.

Proof. plug in Eq (7) to Eq (6), since $\sum_{i=1}^{n} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T = \phi(X)^T\phi(X)$, we have

$$\sum_{i=1}^{n} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T \boldsymbol{v} = (\sum_{i=1}^{n} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T)\boldsymbol{v} = \phi(X)^T\phi(X)\phi(X)^T\boldsymbol{a} = \lambda\phi(X)^T\boldsymbol{a}$$

Then, left multiply by $\phi(X)$, since $K = \phi(X)\phi(X)^T$ and K is invertible, we have

$$\phi(X)\phi(X)^T\phi(X)\phi(X)^T\boldsymbol{a} = \lambda\phi(X)\phi(X)^T\boldsymbol{a}$$
$$\iff K^2\boldsymbol{a} = \lambda K\boldsymbol{a} \iff K\boldsymbol{a} = \lambda\boldsymbol{a}$$

□

(d) But typically we don't have centered features. So instead of $\phi(\boldsymbol{x}_i)$ we should work with $\psi(\boldsymbol{x}_i) = \phi(\boldsymbol{x}_i) - \dfrac{\sum_j \phi(\boldsymbol{x}_j)}{n}$. Show that the kernel matrix built from these centered feature vectors equal $\tilde{K} = K - K\mathbf{1}\mathbf{1}^T/n - \mathbf{1}\mathbf{1}^T/nK + \mathbf{1}^TK\mathbf{1}/n^2\mathbf{1}\mathbf{1}^T$

Proof. Rewrite $\psi(\boldsymbol{x}_i) = \phi(\boldsymbol{x}_i) - \dfrac{\sum_j \phi(\boldsymbol{x}_j)}{n} = \phi(\boldsymbol{x}_i) - \frac{1}{n}\phi(X)^T\mathbf{1}$, by the form of $\phi(X)^T$ in (b),

$$\psi(X)^T = \begin{bmatrix} | & | & & | \\ \psi(\boldsymbol{x}_1) & \psi(\boldsymbol{x}_2) & \dots & \psi(\boldsymbol{x}_n) \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \phi(\boldsymbol{x}_1) - \frac{1}{n}\phi(X)^T\mathbf{1} & \dots & \phi(\boldsymbol{x}_n) - \frac{1}{n}\phi(X)^T\mathbf{1} \end{bmatrix}$$
$$= \phi(X)^T - \frac{1}{n}\phi(X)^T\mathbf{1}\mathbf{1}^T$$

Then, plug in $\psi(X)$ into the definition of $\tilde{K}$, we have

$$\begin{aligned}
\tilde{K} &= \psi(X)\psi(X)^T \\
&= [\phi(X)^T - \frac{1}{n}\phi(X)^T\mathbf{1}\mathbf{1}^T]^T[\phi(X)^T - \frac{1}{n}\phi(X)^T\mathbf{1}\mathbf{1}^T] \\
&= [\phi(X) - \frac{1}{n}\mathbf{1}\mathbf{1}^T\phi(X)][\phi(X)^T - \frac{1}{n}\phi(X)^T\mathbf{1}\mathbf{1}^T] \\
&= \phi(X)\phi(X)^T - \frac{1}{n}\phi(X)\phi(X)^T\mathbf{1}\mathbf{1}^T - \frac{1}{n}\mathbf{1}\mathbf{1}^T\phi(X)\phi(X)^T + \frac{1}{n^2}\mathbf{1}\mathbf{1}^T\phi(X)\phi(X)^T\mathbf{1}\mathbf{1}^T \\
&= K - \frac{1}{n}K\mathbf{1}\mathbf{1}^T - \frac{1}{n}\mathbf{1}\mathbf{1}^TK + \frac{1}{n^2}\mathbf{1}(\mathbf{1}^TK\mathbf{1})\mathbf{1}^T
\end{aligned}$$

Since $\mathbf{1}^TK\mathbf{1}$ is a scalar, we have $\tilde{K} = K - \frac{1}{n}K\mathbf{1}\mathbf{1}^T - \frac{1}{n}\mathbf{1}\mathbf{1}^TK + \frac{(\mathbf{1}^TK\mathbf{1})}{n^2}\mathbf{1}\mathbf{1}^T$. □

(e) So the final algorithm is to build $\tilde{K}$ matrix from the data using your choice of a kernel, and then compute top eigenvector of this matrix and project K on that direction. Download the two parabola dataset. Plot the first principal component using PCA.

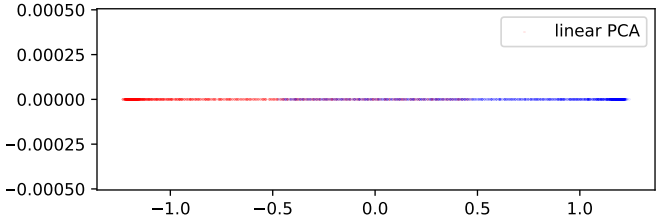


Figure 2: Plot for (e). First eigenvector using linear PCA

Solution. As Figure 2 shows, the two clusters are not separated well. ■

(f) Now do Kernel PCA with the RBF kernel $K(i,j) = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/2\sigma^2)$. Change your power method algorithm so that you do not need to compute the $\tilde{K}$ matrix, but only the K matrix. Provide the Pseudocode.

```
def calculate_square(X, Y):
    XX = (la.norm(X, axis=1)**2)[:, np.newaxis]
    YY = (la.norm(Y, axis=1)**2)[:, np.newaxis]
    XY = np.dot(X, Y.T)
    return XX + YY.T - 2*XY
```

```
def gaussian(G, sigma):
    return np.exp(G/(-2*sigma**2))
```

```
def kernel_dot(K, v):
    n = K.shape[0]
    bo = np.ones(n)

    return np.dot(K, v) - 1/n*np.dot(bo, v)*np.dot(K, bo)\
            - 1/n*np.dot(bo, np.dot(K, v))*bo \
            + 1/n/n*np.dot(bo, np.dot(K, bo))*np.dot(bo, v)*bo
```

```
def kernel_power_iteration(A, T):

    b_k = np.random.rand(A.shape[1])

    for _ in range(T):
        # calculate the kernel multiplication
        b_k1 = kernel_dot(A, b_k)
        # normalize the vector
        b_k = b_k1 / la.norm(b_k1)

    lam = np.dot(b_k, kernel_dot(A, b_k))

    return b_k, lam
```

Figure 3: Plot for (f). Code for Kernel PCA with power method.

Solution. The code is in Figure 3, and "pseudo code" description is

- calculate pairwise distance using matrix form (1st block)
- generate not centered Gaussian kernel matrix (2nd block)
- use "kernel dot" to do power iteration, using only matrix vector product with K and $\mathbf{1}$ (3rd block), then the rest routine is power method (4th block).

■

(g) Try different values of σ, and report the one which returns a first kernel PC such that the two classes are separated along this direction.

Solution. See "p4_kernel-pca.html" for codes and results. Figure 4 shows the results using kernel PCA with different σ. As it shows, when σ is between 0.1 and 0.2, the two classes are separated pretty well.

■

(h) What do you think is the relationship of Spectral Clustering with Kernel PCA? Give your answer in 4-5 lines.

Solution. **Similarity:** (1) Goal: Spectral Clustering and Kernel PCA are both trying to find a good nonlinear basis to the given dataset; (2)Eigen Decompostion: They both use first k eigenvectors. **Difference:** (1) Different matrices: Spectral Clustering uses affinity matrix based on pairwise distance while kernel PCA using Gaussian distance matrix; (2) Different Normalization methods.

■

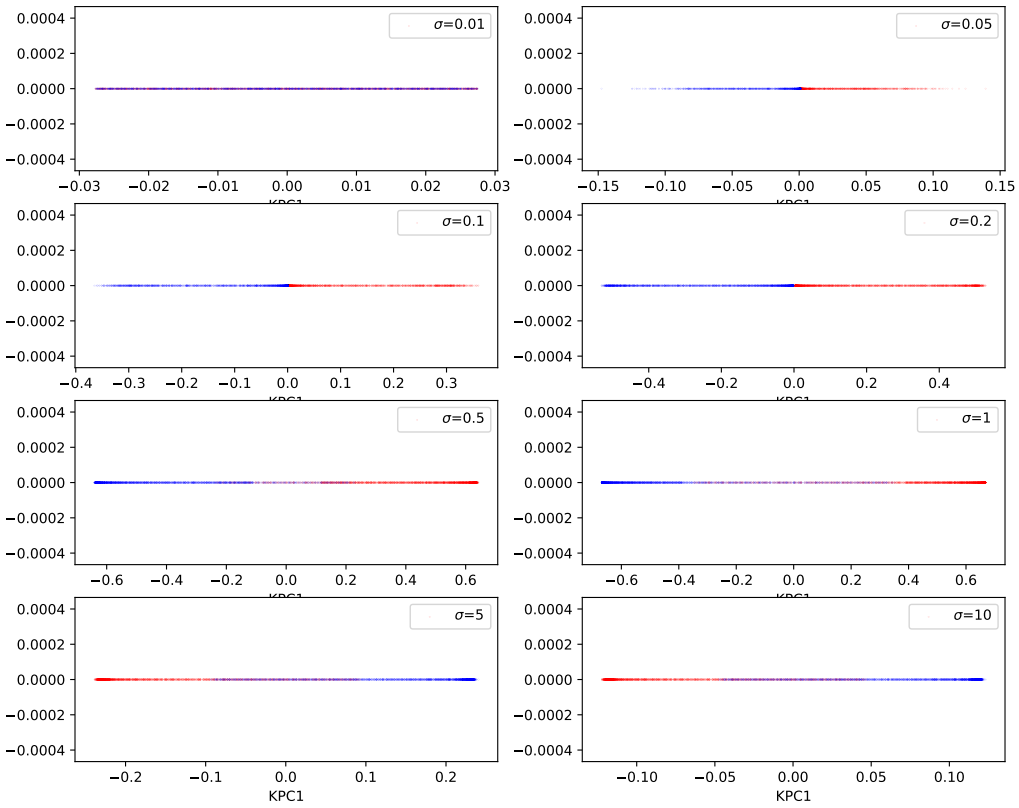


Figure 4: Plot for (g). First eigenvector using kernel PCA with gaussian kernel with differentσ values: $\sigma = 0.01, 0.05, 0.1, 0.2, 0.5, 1, 5, 10$.