Homework Assignment 3

Anurag, arp3969

1. **Spectral Clustering**[1]

   (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 E2LSH or packages for kdtree for the rest. There are four algorithms, *Exact*, *JL+Exact*, *JL+KDtrees* and *E2LSH*. 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.

   ***Ans*: Exact and Spectral Clustering**

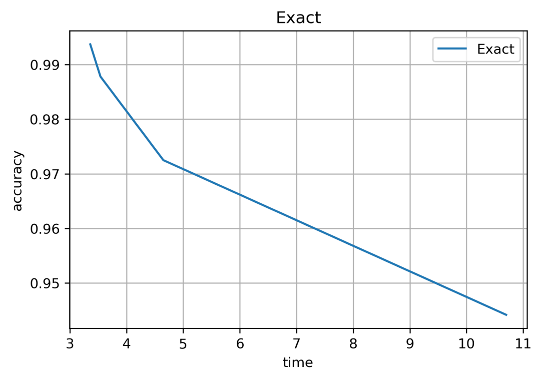

Figure 1: Time vs Accuracy for Exact

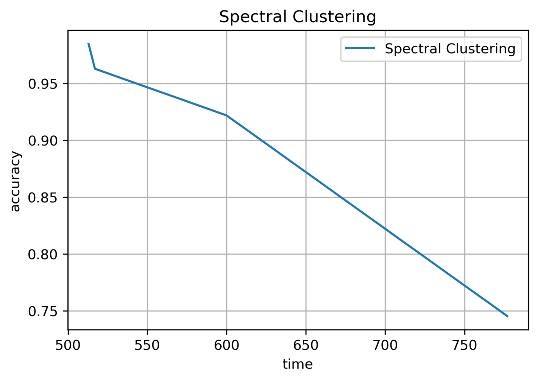

Figure 2: Time vs Accuracy for SC

Times are as shown in figure 1. For the Exact implementation the function KNeighborsClassifier with algorithm as brute from the sklearn python library was used. As the method uses an effective implementation of the algorithm the time for spectral clustering and Exact were very different. A fair comparision can be made with the other methods like JLExact and JLKDtree as the same library has been used for them as well. In general it can be seen the same trend for time vs accuracy holds true though the absolute values of time might be different.

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

Discussion of the bandwidth parameter:
The bandwidth parameter vs accuracy is as shown in the figure.

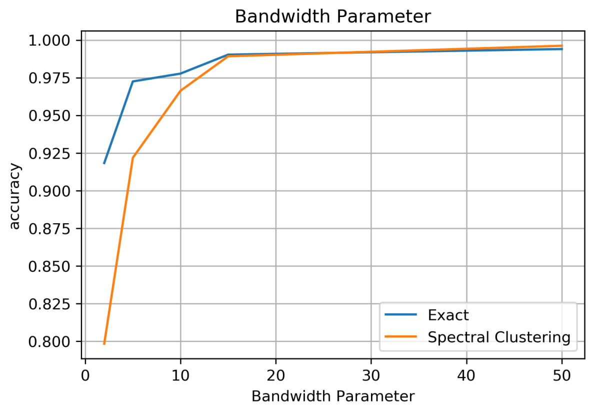

Figure 3: Bandwidth Parameter vs accuracy

As can be seen from the plot higher the values of the bandwidth parameter the accuracy increases while the time doesnt change much (Exact time = [7.3, 4.65, 7.43,7.58, 13.5], Spectral clustering time = [502 ,600,668,700,712]. But after a point change the value of bandwidth parameter doesn't change much. Hence, this curve flatens out and there is a good value of bandwidth parameter there is good accuracy and the time taken is not too much.

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."
Ans: The time for exact implementation without the library for 10 randomly generated run was 71 mins.

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.
Ans: **For JL+Exact:**

As can be seen from the figures 3 and 4, the required trend is observed for the largest dataset with $p = 2000$. A similar trend is observed for the smaller datasets as well. That is 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. For eps = [0.2, 0.3, 0.5,0.7, 0.8,0.9] the corresponding JL dimensions are [2125,1023, 442,281, 246,227]. Thus, for lesser dimensions like

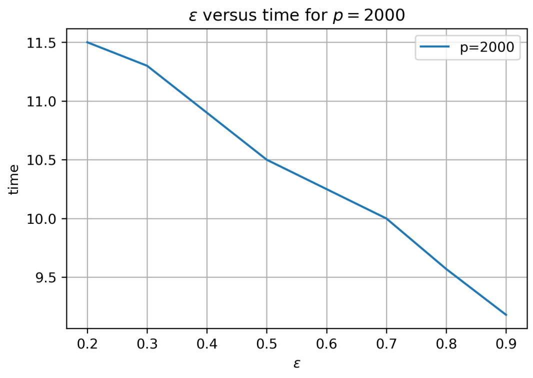

Figure 4: ϵ vs time for JL+Exact

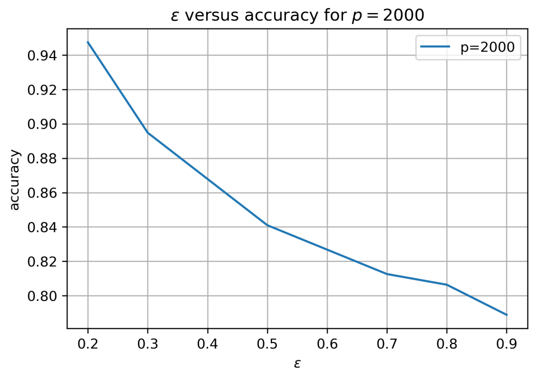

Figure 5: ϵ vs Accuracy for JL+Exact

250 and 500 the smaller epsilon values like 0.2, 0.3 result in inclusion of all the dimensions. Hence, for trend comparison $p = 2000$ was considered.

For JL+KD tree:

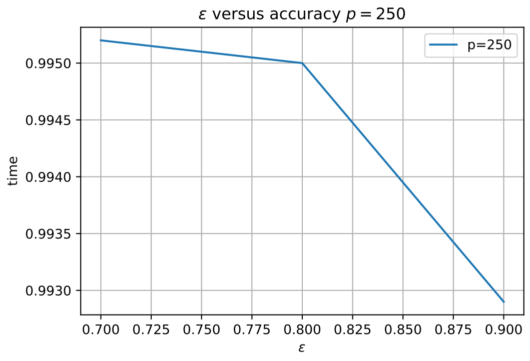

Figure 6: ϵ vs time for JL+KD tree

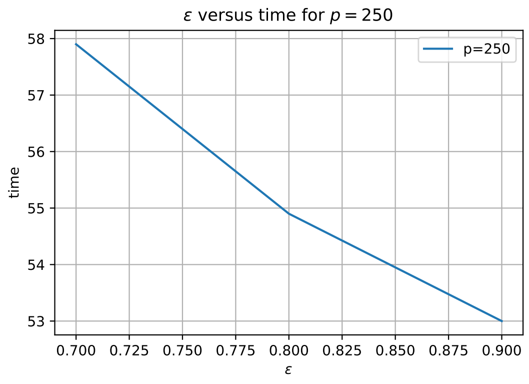

Figure 7: ϵ vs Accuracy for JL+KD tree

As can be seen from the previous discussion JL lemma for a high $\epsilon = 0.9$ results in 227 dimensions. As can be followed from the discussions in lectures KD trees don't work well for such high dimensions. The results for these are shown in figure 5 and 6. In order to analyse KD trees further, the dimesion for the KD tree were reduced as KDdim = [20, 30,50, 70,100]. The corresponding time and accuracy values are plotted in figures 7 and 8.

iii. (E2LSH) Use E2LSH to implement Locality sensitive hashing to obtain the nearest neighbor graph. You can find a matlab package here: `http://ttic.uchicago.edu/~gregory/download.html`.
***Ans*: E2LSH:**
Please see the submitted python notebook for implementation. The python package Falconn is used for this. Since, this is not from the library as used for

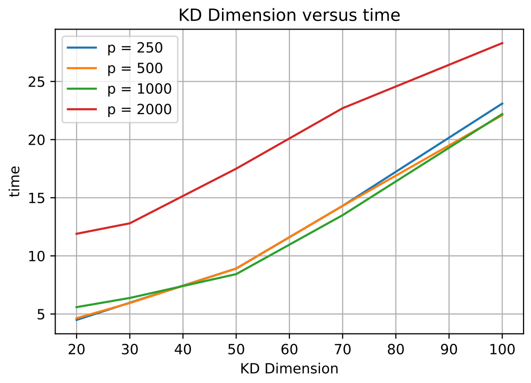

Figure 8: $KDdimensions$ vs time for JL+KD tree

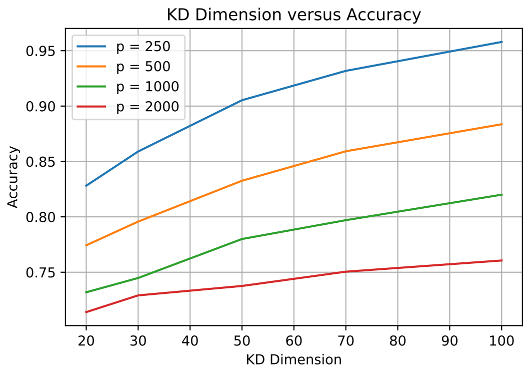

Figure 9: $KDdimensions$ vs Accuracy for JL+KD tree

other methods, the time comparison wont be fair. Nonetheless the accuracy vs time trend is as follows:

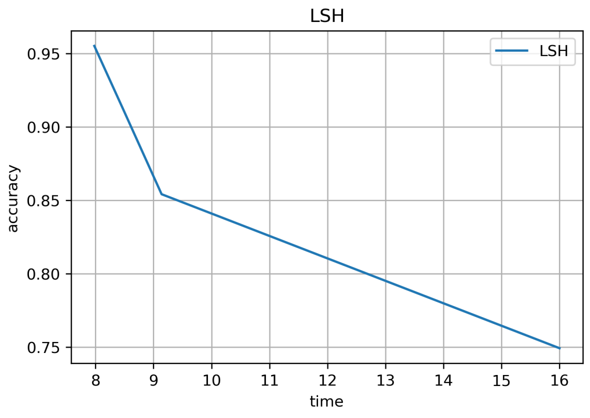

Figure 10: Time vs Accuracy for E2LSH

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 $|\lambda_1| > |\lambda_2| > \dots$. We will assume that the eigenvalues are all different for convenience of analysis. Start with some vector $\boldsymbol{q}^{(0)}$. Now compute the following:

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

$$\boldsymbol{q}^{(k)} = \frac{\boldsymbol{z}^{(k)}}{\|\boldsymbol{z}^{(k)}\|} \tag{2}$$

$$\nu^{(k)} = (\boldsymbol{q}^{(k)})^T A\boldsymbol{q}^{(k)} \tag{3}$$

(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.
Ans: Consider the initial value $\boldsymbol{q}^{(0)}$, for this,

$$\begin{aligned}
\boldsymbol{z}^{(1)} &= A\boldsymbol{q}^{(0)} \\
\boldsymbol{q}^{(1)} &= \frac{\boldsymbol{z}^{(1)}}{\|\boldsymbol{z}^{(1)}\|} \\
\boldsymbol{q}^{(1)} &= \frac{A^1\boldsymbol{q}^{(0)}}{\|A^1\boldsymbol{q}^{(0)}\|}.
\end{aligned}$$

Thus, it is valid for k=1.
Consider k = 2:

$$\begin{aligned}
\boldsymbol{z}^{(2)} &= A\boldsymbol{q}^{(1)} \\
\boldsymbol{q}^{(2)} &= \frac{\boldsymbol{z}^{(2)}}{\|\boldsymbol{z}^{(2)}\|} \\
\boldsymbol{q}^{(2)} &= \frac{A^2\boldsymbol{q}^{(0)}}{\|A^2\boldsymbol{q}^{(0)}\|}.
\end{aligned}$$

Now, assume that the equation is true for a arbitary k, considering this lets prove it for k+1;

$$\begin{aligned}
\boldsymbol{z}^{(k+1)} &= A\boldsymbol{q}^{(k-1)} \\
\boldsymbol{z}^{(k+1)} &= A\frac{A^k\boldsymbol{q}^{(0)}}{\|A^k\boldsymbol{q}^{(0)}\|} \\
\boldsymbol{q}^{(k+1)} &= \frac{\boldsymbol{z}^{(k+1)}}{\|\boldsymbol{z}^{(k+1)}\|} \\
\boldsymbol{q}^{(k+1)} &= \frac{A\frac{A^k\boldsymbol{q}^{(0)}}{\|A^k\boldsymbol{q}^{(0)}\|}}{A\frac{A^k\boldsymbol{q}^{(0)}}{\|A^k\boldsymbol{q}^{(0)}\|}} \\
q^{(k+1)} &= \frac{A^{k+1}\boldsymbol{q}^{(0)}}{\|A^{k+1}\boldsymbol{q}^{(0)}\|}
\end{aligned}$$

Thus, it is true for k+1. Hence, by principle of mathematical induction the give equation is proved.

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

Ans: We know that,

$$A = \sum_{i=1}^{n} \lambda_i \boldsymbol{x}_i \boldsymbol{x}_i^T \tag{4}$$

$$\boldsymbol{q}^{(k)} = \frac{A^k \boldsymbol{q}^{(0)}}{\|A^k \boldsymbol{q}^{(0)}\|} \tag{5}$$

$$\boldsymbol{q}^{(0)} = \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i \tag{6}$$

$$A\boldsymbol{x}_i = \lambda_i \boldsymbol{x}_i \tag{7}$$

$$\begin{aligned}
A^k\boldsymbol{q}^{(0)} &= \alpha_1 A^k \boldsymbol{x}_1 + \alpha_2 A^k \boldsymbol{x}_2 + ... + \alpha_n A^k \boldsymbol{x}_n \text{ [using equation 6]} \\
A^k\boldsymbol{q}^{(0)} &= \sum_{i=1}^{n} \alpha_i \lambda^k \boldsymbol{x}_i \\
A^k\boldsymbol{q}^{(0)} &= \alpha_1 \lambda_1^k \boldsymbol{x}_1 + \sum_{i=2}^{n} \alpha_i \lambda^k \boldsymbol{x}_i
\end{aligned}$$

$$A^k\boldsymbol{q}^{(0)} = \alpha_1 \lambda_1^k (\boldsymbol{x}_1 + \sum_{i=2}^{n} \frac{\alpha_i}{\alpha_1} \left(\frac{\lambda_i}{\lambda_1}\right)^k \boldsymbol{x}_i) \tag{8}$$

Now,

$$\begin{aligned}
\langle \boldsymbol{q}^{(k)}, \boldsymbol{x}_1 \rangle &= \boldsymbol{q}^{(k)T} \boldsymbol{x}_1 \\
\langle \boldsymbol{q}^{(k)}, \boldsymbol{x}_1 \rangle &= \frac{\lambda_1^k \alpha_1}{\|A^k \boldsymbol{q}^{(0)}\|}
\end{aligned}$$

that is the dot product between the two will only have component along the $_1$ direction.
Therefore,

$$\begin{aligned}
\|\boldsymbol{q}^{(k)} - \langle \boldsymbol{q}^{(k)}, \boldsymbol{x}_1 \rangle \boldsymbol{x}_1\| &= \|\boldsymbol{q}^{(k)} - \frac{\lambda_1^k \alpha_1}{\|A^k \boldsymbol{q}^{(0)}\|} \boldsymbol{x}_1\| \text{ [taking out the constant from the second term]} \\
&= \frac{\lambda_1^k \alpha_1}{\|A^k \boldsymbol{q}^{(0)}\|} \|\boldsymbol{q}^{(k)} \frac{\|A^k \boldsymbol{q}^{(0)}\|}{\lambda_1^k \alpha_1} - \boldsymbol{x}_1\| \text{ [using the equation(5)]} \\
&= \frac{\lambda_1^k \alpha_1}{\|A^k \boldsymbol{q}^{(0)}\|} \|\frac{A^k \boldsymbol{q}^{(0)}}{\lambda_1^k \alpha_1} - \boldsymbol{x}_1\| \text{ [using the equation(8)]} \\
&= \frac{\lambda_1^k \alpha_1}{\|A^k \boldsymbol{q}^{(0)}\|} \|(\boldsymbol{x}_1 + \sum_{i=2}^{n} \frac{\alpha_i}{\alpha_1} \left(\frac{\lambda_i}{\lambda_1}\right)^k \boldsymbol{x}_i) - \boldsymbol{x}_1\| \\
&= \frac{\lambda_1^k \alpha_1}{\|A^k \boldsymbol{q}^{(0)}\|} \| \sum_{i=2}^{n} \frac{\alpha_i}{\alpha_1} \left(\frac{\lambda_i}{\lambda_1}\right)^k \boldsymbol{x}_i)\| \text{ [norm of } \boldsymbol{x}_1 \text{ is one]} \\
&\leq C \left|\frac{\lambda_2}{\lambda_1}\right|^k \text{ [using cauchy schwarz inequality]}
\end{aligned}$$

3. Now, write your own routine for power iteration and apply it to compute the second eigenvector of the nearest neighbor graphs you have built in the last question. For each of them, show the error (you should adjust for signs since an eigenvector can be multiplied by 1 or -1 but it still remains an eigenvector) of your estimated vector and the second eigenvector computed using matlab or R.
Ans: The few elements of the error vector for A corresponding to $p = 250$ are as folllows array([0.01082991+0.j, 0.00981527+0.j, 0.0068765 +0.j, ...,-0.01082465+0.j, -0.00477892+0.j, -0.00885343+0.j]). All the error values are plotted in the figure 9.

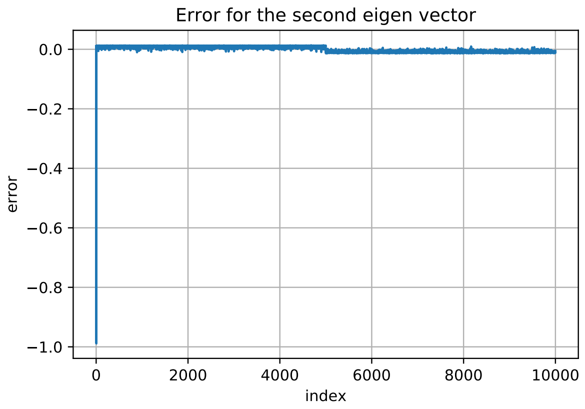

Figure 11: Error for the second eigen vector calculation

Similar results are observed for p = 500, 1000 and 2000 as seen in the attached python notebook.

4. 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.

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

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

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

   So finding $\boldsymbol{v}$ boils down to finding $\boldsymbol{a}$.
   Ans: The eigenvectors can be written as a linear combination for features:

$$\boldsymbol{v} = \sum_{i=1}^{n} \phi(\boldsymbol{x}_i) a_i \tag{9}$$

Hence, finding the first eigenvector is equivalent to finding the coefficients a_i, i = 1,...n. Therefore, $\boldsymbol{a} \in \mathbb{R}^{n\times 1}$. Hence, this can be return as

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

where, $\boldsymbol{v} \in \mathbb{R}^{N\times 1}$

(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 $\boldsymbol{a}$.
Ans: The projection of a point $\mathbf{x}$ onto the first eigenvector is given by:

$$y(\mathbf{x}) = \phi(x)^T\mathbf{v}$$

using the equation(9),

$$y(\mathbf{x}) = \sum_{i=1}^{n} a_i\phi(x)^T\phi(\boldsymbol{x}_i)$$
$$y(\mathbf{x}) = \sum_{i=1}^{n} a_i k(\mathbf{x},\mathbf{x}_i)$$

Thus, writing it in matrix form

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

(c) Now plug in Eq (9) to Eq (**??**), and left multiply by $\phi(\boldsymbol{x}_k)^T$ to get:

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

You can assume that K is invertible.
Ans:

$$\sum_{i=1}^{n}\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T\sum_{l=1}^{n}a_l\phi(\boldsymbol{x}_l) = \lambda\sum_{i=1}^{n}a_i\phi(\boldsymbol{x}_i)$$

The key step is now to express this in terms of kernel function $k(x_i,x_j) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$, which we do my multiplying both the sides by $\phi(\boldsymbol{x}_k)^T$ to give,

$$\sum_{i=1}^{n}\phi(\boldsymbol{x}_i)\sum_{l=1}^{n}a_l k(x_i,x_l) = \lambda\sum_{l=1}^{n}a_l\phi(\boldsymbol{x}_l)$$
$$\sum_{i=1}^{n}\phi(\boldsymbol{x}_k)^T\phi(\boldsymbol{x}_i)\sum_{l=1}^{n}a_l k(x_i,x_l) = \lambda\sum_{l=1}^{n}a_l\phi(\boldsymbol{x}_k)^T\phi(\boldsymbol{x}_l)$$
$$\sum_{i=1}^{n}k(x_k,x_i)\sum_{l=1}^{n}a_l k(x_i,x_l) = \lambda\sum_{l=1}^{n}a_l k(x_k,x_l)$$

This can be written in matrix notation as

$$K^2\boldsymbol{a} = \lambda K\boldsymbol{a}.$$

We remove a factor of K from both sides of matrix (this will only affect eigenvectors with eigenvalues 0, which will not be principle components anyway)

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

(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 $\tilde{K} = K - K\mathbf{1}\mathbf{1}^T/n - \mathbf{1}\mathbf{1}^T/nK + \mathbf{1}^T K\mathbf{1}/n^2\mathbf{1}\mathbf{1}^T$
Ans: The corresponding elements in the gram matrix are given by

$$\tilde{K}_{i,l} = \psi(\boldsymbol{x}_i)\psi(\boldsymbol{x}_l)$$
$$\tilde{K}_{i,l} = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_l) - \frac{\sum_j \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)}{n} - \frac{\sum_j \phi(\boldsymbol{x}_j)^T\phi(\boldsymbol{x}_l)}{n} + \frac{\sum_k\sum_j \phi(\boldsymbol{x}_k)^T\phi(\boldsymbol{x}_j)}{n^2}$$
$$\tilde{K}_{i,l} = k(x_i, x_l) - \frac{\sum_j k(x_i, x_j)}{n} - \frac{\sum_j k(x_j, x_l)}{n} + \frac{\sum_k\sum_j k(x_k, x_j)}{n^2}$$

This can be expressed in matrix form as
$\tilde{K} = K - K\mathbf{1}\mathbf{1}^T/n - \mathbf{1}\mathbf{1}^T/nK + \mathbf{1}^T K\mathbf{1}/n^2\mathbf{1}\mathbf{1}^T$

(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.
Ans:

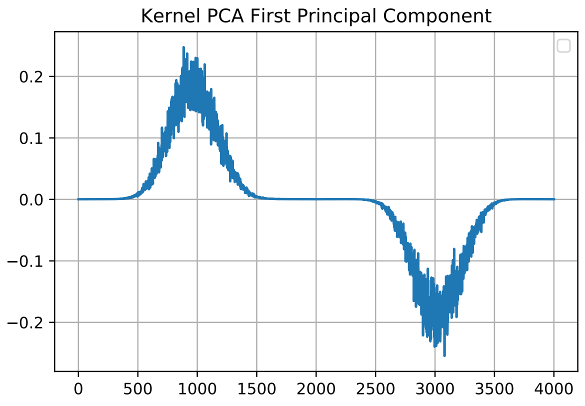

Figure 12: First principal component with Kernel PCA

(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.
Ans: The pseudocode is as follows:

- Intinitialize and store the frequently computed values.
  one = np.ones((N, N)) / N
  eigenvec = np.random.rand(N, 1)

```
OneDotK = one.dot(K)
KOneDot = K.dot(one)
```

- For iterations = 1000:
```
newEigenvec = K.dot(eigenvec) - OneDotK.dot(eigenvec) - KOneDot.dot(eigenvec)
+ OneDotK.dot(one).dot(eigenvec)
newEigenvec = newEigenvec/np.linalg.norm(newEigenvec)
eigenvec = newEigenvec
```

Here, the method is more efficient because in the update equation matrix vector updates are made as compared to matrix matrix as was the case earlier. Further more, to speed up the computation common reccuring mutliplications is K.one, etc are precomputed and stored.

(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.
Ans: The σ of 0.1 returns the first kernel PC such that the two classes are seperated. And σ of 0.01 returns the best seperation. The seperations into clusters are as show in the graph for different values of σ.

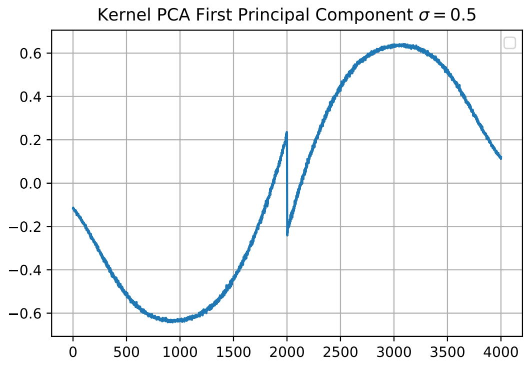

Figure 13: σ of 0.5

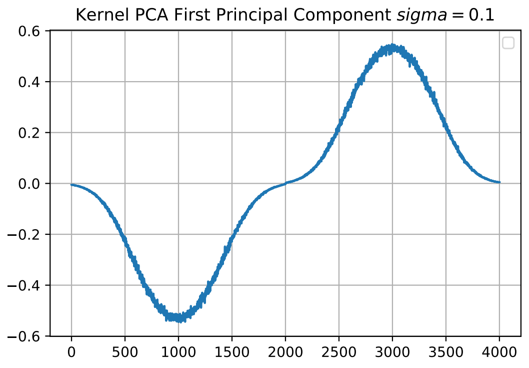

Figure 14: σ of 0.1

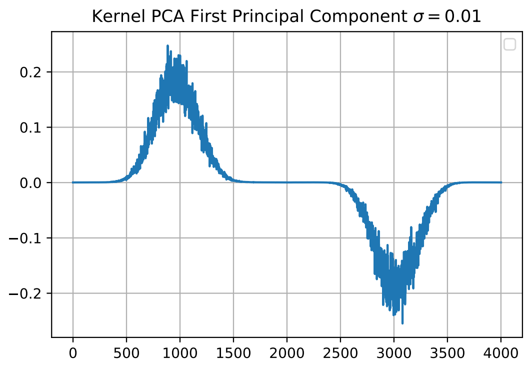

Figure 15: σ of 0.05

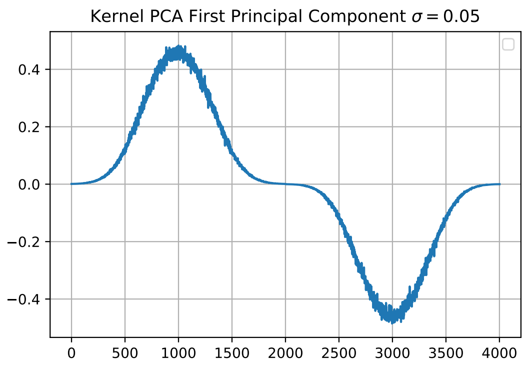

Figure 16: σ of 0.01

Thus from the plots, the clusters get bette seperated as σ value decreases. Thus

Validation, i.e if clusters are getting formed or not can be used to determine good kernel parameter(σ) values. This is expected since choice of σ completely changes the nature of the embedding.

(h) What do you think is the relationship of Spectral Clustering with Kernel PCA? Give your answer in 4-5 lines.
Ans: We have see how KPCA and spectral clustering are trying to find a good nonlinear basis to the dataset so that clustering will be easier in that space, but differ in the normalization applied to the Gram matrix. Both are using the eigenspace to find these clusters using reduced dimensions. Both types of methods are found to estimate eigenfunctions of a linear operator associated with the kernel and with the data. Both methods differ in the ease of tuning of their parameter, σ and the bandwidth parameter k. A comment on this could have been made had the same dataset been considered for both.