Advanced Machine Learning
Practical 2: Manifold Learning + Clustering (Spectral Clustering and Kernel K-Means)

Professor: Aude Billard
Assistants: Bernardo Fichera, Roland Schwan
`E-mails:` aude.billard@epfl.ch,
bernardo.fichera@epfl.ch, roland.schwan@epfl.ch

Spring Semester 2021

1 Introduction

During this week's practical we will cover two more manifold learning algorithms, namely (i) Laplacian Eigenmaps and (ii) Isomaps, and compare their projections to kernel PCA. We will then perform *Spectral Clustering*, which involves applying K-means on the projections of these non-linear embeddings. Finally, we will evaluate the kernel K-Means algorithm, which clusters and extracts the non-linear embedding simultaneously.

2 ML_toolbox

`ML_toolbox` contains a set of methods and examples for easily learning and testing machine learning methods on your data in MATLAB. It is available in the following link:

`https://github.com/epfl-lasa/ML_toolbox`

From the course Moodle webpage (or the website), the student should download and extract the .zip file named `TP2-Manifold+Clustering.zip` which contains the following files:

Code	Code
`TP2_proj_methods.m`	`TP2_kernel_kmeans.m`
`TP2_spectral_clustering.m`	`setup_TP2.m`

Before proceeding make sure that `./ML_toolbox` is at the same directory level as the TP directory `./TP2-Manifold+Clustering`. You must also place `setup_TP2.m` at the same level. Now, to add these directories to your search path, type the following in the MATLAB command window:

```
>> setup_TP2
```

NOTE: To test that all is working properly with ML_Toolbox you can try out some examples of the toolbox; look in the **examples** sub-directory.

3 Manifold Learning Techniques

In *classical* dimensionality reduction techniques, such as **PCA** and **CCA**, given a training dataset $\mathbf{X} \in \mathbb{R}^{N \times M}$ composed of M datapoints with N-dimensions, we seek to find a lower-dimensional embedding $\mathbf{Y} \in \mathbb{R}^{p \times M}$, through a mapping function $f(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^N \rightarrow \mathbf{y} \in \mathbb{R}^p$ where $p \leq N$. In *non-linear* dimensionality reduction techniques, we may lift the data into a higher dimensional space (i.e. $> N$) so as to find structure in the dataset. In order to find a non-linear embedding of the dataset, all of the kernel techniques seen in class (i.e. **kPCA**, **kCCA** and **Spectral Clustering**) rely on projecting the data-points onto a set of vectors whose dimension can be $p \leq M$. Reduction of the dimensionality is hence done on the size of the dataset ($N \times M$) and not necessarily on the size of the original space in which the datapoints are embedded ($\mathbb{R}^N$).

In the previous practical we learned about **kernel PCA**. In this practical we will evaluate two other non-linear embedding methods, namely (i) **Laplacian Eigenmaps** and (ii) **Isomap**. All of these techniques are based on a spectral decomposition of a matrix. This matrix differs depending on the technique of choice. In kernel PCA, it is the Gram (**K**ernel) matrix. In Spectral Clustering, it is the graph **L**aplacian which we construct from a **S**imilarity matrix. The **S**imilarity matrix can be built from using kernels as similarity measures, however, is not restricted to them. In fact, other similarity measures can also be used, as long as they hold specific properties, such as positive definitiveness, symmetry, among others.

3.1 Laplacian Eigenmaps

Given the dataset $\mathbf{X} \in \mathbb{R}^{N \times M}$, the Laplacian Eigenmaps [1] algorithm proceeds as follows:

1. Build *Adjacency* Graph: We begin by building an *adjacency graph* with M nodes, and a set of edges connecting neighboring points. An edge is placed between the i-th and j-th nodes if $\mathbf{x}^i$ is among the k-nearest neighbors of $\mathbf{x}^j$. The k nearest neighbors are selected under the Euclidean distance $||\mathbf{x}^i - \mathbf{x}^j||$.

2. Compute *Similarity* matrix: We then compute the **S**imilarity matrix $\mathbf{S} \in \mathbb{R}^{MxM}$ of the graph, in this case, we use the Gaussian kernel as a measure of similarity:

$$S_{ij} = \begin{cases} e^{-\frac{||\mathbf{x}^i - \mathbf{x}^j||^2}{2\sigma^2}}, & \text{if edge exists between } i\text{-th and } j\text{-th nodes} \\ 0, & \text{elsewhere} \end{cases} \tag{1}$$

3. Construct the *Eigenmaps*: Given $\mathbf{S}$, we can construct the diagonal weighting matrix $\mathbf{D}$, whose entries are column sums of $\mathbf{S}$; i.e. $D_{ii} = \sum_j S_{ji}$. Once the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{S}$ is computed, we solve for the following generalized eigenvalue problem:

$$\begin{aligned} \mathbf{L}\mathbf{e} &= \lambda \mathbf{D}\mathbf{e} \\ (\mathbf{I} - \mathbf{D}^{-1}\mathbf{S})\mathbf{e} &= \lambda \mathbf{e} \end{aligned} \tag{2}$$

We then order the Eigenvectors wrt. $0 = \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_M$; i.e. $\mathbf{U} = [\mathbf{e}^0, \mathbf{e}^1, \ldots, \mathbf{e}^M]$ for $\mathbf{e}^i \in \mathbb{R}^M$ column-vectors. By selecting the first p eigenvectors, we construct the datapoints on the p-dimensional manifold $\mathbf{Y} \in \mathbb{R}^{p \times M}$ as $\mathbf{Y} = [\mathbf{e}^1, \ldots, \mathbf{e}^p]^T$

This mapping ensures minimal distortion while preventing arbitrary scaling. One must, however, take into consideration the hyper-parameters k (to generate the adjacency graph via the k-nearest neighbors) and σ (the width of the Gaussian kernel) in order to achieve the desired embedding.

3.2 Isomap (Isometric Mapping)

The Isomap algorithm [2], short for Isometric Feature Mapping, is a generalization of Multi-Dimensional Scaling (MDS), which uses geodesic rather than Euclidean distances to generate the similarity matrix $\mathbf{S}$. The Isomap algorithm proceeds as follows:

1. Build *Neighborhood* Graph: We begin by building a *neighborhood graph* with M nodes, and a set of edges connecting neighboring points. An edge is placed between the i-th and j-th nodes if $\mathbf{x}^i$ is among the k-nearest neighbors of $\mathbf{x}^j$. The k-nearest neighbors are selected under the Euclidean distance $||\mathbf{x}^i - \mathbf{x}^j||$.

2. Compute *Similarity* matrix: We then compute the similarity matrix $\mathbf{S} \in \mathbb{R}^{MxM}$ of the graph, by applying the following equation to each pair of points:

$$S_{ij} = \begin{cases} ||\mathbf{x}^i - \mathbf{x}^j||^2, & \text{if edge exists between } i\text{-th and } j\text{-th nodes} \\ \min\limits_{k-NN} ||\mathbf{x}^i - \mathbf{x}^j||^2, & \text{elsewhere} \\ 0, & \text{not in the same connected component} \end{cases} \tag{3}$$

   In other words, for nodes in connected components, if an edge exists between two nodes we use the squared Euclidean distance. On the other hand, if there is no edge between the nodes but they belong to the same connected component in the graph, we compute the shortest path in the weighted point-graph using the Dijkstra algorithm. This yields a matrix with the shortest path distances between all pairs of points.

3. Apply MDS to $\mathbf{S}$: Given $\mathbf{S}$, we must then compute the centered similarity matrix $\mathbf{S}'$ of square Euclidean distances as follows:

$$S'_{ij} = -\frac{1}{2}\left(S_{ij} - \frac{1}{M}D_i - \frac{1}{M}D_j + \frac{1}{M^2}D_{i,j}\right) \tag{4}$$

   where $D_i = \sum_{j=1} S_{ij}$, $D_j = \sum_{i=1} S_{ij}$ and $D_{i,j} = \sum_{i,j=1} S_{ij}$ correspond to sum of each row, sum of each column and sum of all elements of S, respectively. After performing eigenvalue decomposition on $\mathbf{S}' = \mathbf{U}\Lambda\mathbf{U}^T$, we select the p eigenvectors with positive real eigenvalues (as in PCA) and generate the scaled projections as: $y_i^j = \sqrt{\lambda_i} e_j^i$ for $i = 1, \ldots, p$ and $j = 1, \ldots, M$.

3.3 Comparison of Non-linear Embedding Techniques

TASK 1: Try Kernel PCA, Laplacian Eigenmaps and Isomap to find the best representation of 3D Datasets composed of 1000 points each.
In this task, you must try to determine which is the most suitable non-linear projection method to uncover the 'true' low-dimensional representation of different 3D datasets. The datasets in question are depicted in Figure 1 and 2. In order to address this, you should ask yourself the following questions:

- What is the underlying manifold that we want to uncover?
  `Hint: both datasets have an underlying 2D manifold.`
- Once we know what low-dimensional representation to expect, which method seems most suitable?
- For each method, what are the necessary hyper-parameters I have to tune and how can I select them?

Once you have answered these questions, load the datasets by running sub-blocks `1(a)` and `1(b)` of the accompanying MATLAB script: `TP2_proj_methods.m` and find a good projection for each method. As a baseline, we provide PCA in code block `(2)`. The following code blocks `(3-5)` contain the necessary code snippets to run kernel PCA `(3)`, Laplacian Eigenmaps `(4)` and Isomaps `(5)`. In each code block, we provide a description of the parameters to modify for each method. For example, in code block `(4)` the Laplacian Eigenmap method is defined as follows:

```
%% Compute Laplacian Eigenmap with ML_toolbox
options = [];
options.method_name       = 'Laplacian';
options.nbDimensions      = 4;   % Number of Eigenvectors to compute.
options.neighbors         = 10;  % Number of k-NN for Adjacency Graph
options.sigma             = 1;   % Sigma for Similarity Matrix
[proj_LAP_X, mappingLAP]  = ml_projection(X',options);
```

For each method, the generated projections will be computed automatically and stored in the `proj_METHOD_X` variable. The dimensionality of these projections is defined in `options.nbDimensions`. The `mappingLAP` variable is a data structure containing all the necessary variables computed for each technique; e.g. eigenvectors, eigenvalues, kernel matrix, similarity matrix, etc. For all methods covered in this practical, we can generate different embeddings depending on the eigenvector or eigenvector pairs we choose. To visualize these different embeddings, we can simply define the pair or set of projections that we wish to see in a `vector` as follows:

```
% Plot result of Laplacian Eigenmaps Projection
...
if exist('h4','var') && isvalid(h4), delete(h4);end
h4 = ml_plot_data(proj_LAP_X(:,[1 4]),plot_options);
```

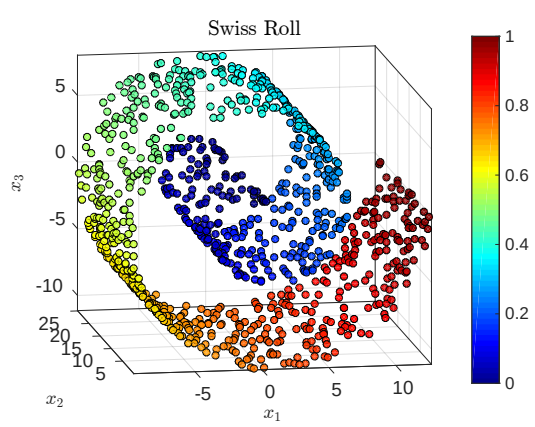


Figure 1: 3D Swiss Roll Dataset.

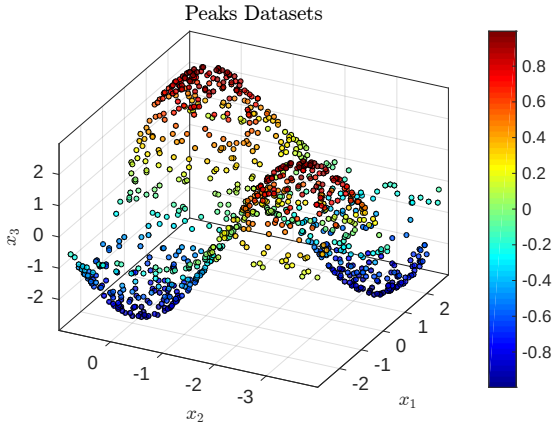


Figure 2: 3D Twin Peaks Dataset.

4 Spectral Clustering

These non-linear embedding methods are not only popular because they can transform data to its 'true' manifold, but also because the eigenvalues of the **S**imilarity, **L**aplacian and **K**ernel matrices can be used to determine the number of K clusters of a dataset. In fact, when non-linearly separable datasets are involved, we can apply a clustering algorithm (i.e. K-means) to the embedded space; where the data is assumed to be less non-linearly separable. This procedure is called *Spectral Clustering*. In this practical, we will focus on comparing results of applying the following *Spectral Clustering* variants, to non-linearly separable datasets:

1. kPCA + K-means
2. Laplacian Eigenmaps + K-means
3. Isomap + K-means

Following we will introduce some of the metrics used to evaluate the performance of K-means clustering.

4.1 Evaluation Metrics for K-Means Clustering

Clustering is a process of partitioning a set of data (or objects) in a set of meaningful sub-classes, called clusters. The K-means algorithm is a widely used clustering technique that seeks to minimize the average squared distance between points in the same cluster (Figure 3). In other words, given a dataset $\mathbf{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^M\}$ where $\mathbf{x}^i \in \mathbb{R}^N$ we can describe it with K clusters, each represented by a centroid $\mu^k \in \mathbb{R}^N$, by minimizing the total squared distance between each point and its closest centroid with the following cost function,

$$J(\mu_1, \ldots, \mu_k) = \sum_{k=1}^{K} \sum_{\mathbf{x}^i \in C^k} ||\mathbf{x}^i - \mu^{\mathbf{k}}||^2 \tag{5}$$

where $C^k \in \{1, \ldots, K\}$ is the cluster label. This is an NP-hard problem, however, the K-means algorithm provides a local search solution for (5) through an iterative procedure proposed in the 1980's by Stuart P. Lloyd [3].

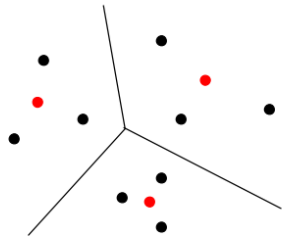


Figure 3: Illustration of k-means clustering. We have a dataset $\mathbf{X}$ with $N = 2$ and $M = 9$, by setting $K = 3$ we would like to estimate the red points corresponding to the $\mu = \{\mu_1, \mu_2, \mu_3\}$ and the decision boundaries describing the data partition (depicted by the black lines between clusters).

In practice, one can encounter two types of datasets in clustering:

1. Datasets where the number of clusters K is visually apparent, or datasets which are already linearly separable, and hence, can be easily partitioned, as shown in Figure 4. In this dataset, clearly $K = 4$.

2. Datasets where we either cannot visualize the data or simply don't know the correct number of clusters, such as the dataset in Figure 5. This dataset is composed of two classes with overlapping points. However, it is not clear how many K clusters to use, as clustering does **NOT** consider the labels. If we apply K-means on this dataset we can set K to different values, and they will all converge to a reasonable partition (see Figure 6, 7 and 8).

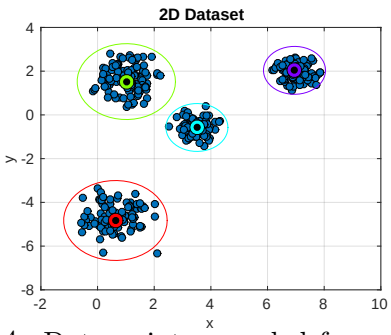


Figure 4: Data-points sampled from a $K = 4$ component GMM. Colored points/ellipses correspond to the parameters of each component.

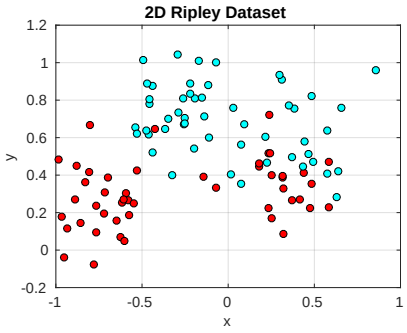


Figure 5: The 2D Ripley Dataset, which is composed of two classes with overlapping points.

So, how do we determine the best clustering?
For this, we can use a set of metrics which are used to evaluate clustering performance based on Equation 5. Following we provide the equations of the **three** clustering metrics that will be used in this practical:

- **RSS:** The **R**esidual **S**um of **S**quares is, in fact, the cost function that K-means is trying to minimize (Equation 5.), hence

$$RSS = \sum_{k=1}^{K} \sum_{\mathbf{x}^i \in c_k} |\mathbf{x}^i - \mu^{\mathbf{k}}|^2 \tag{6}$$

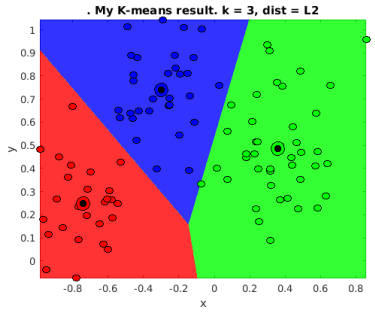


Figure 6: Decision Boundary of $K = 3$-Means.

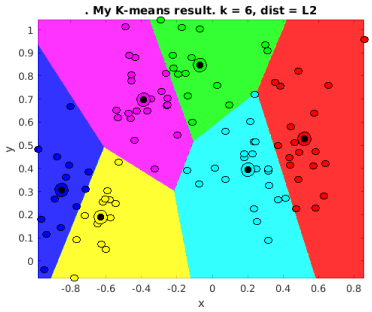


Figure 7: Decision Boundary of $K = 6$-Means.

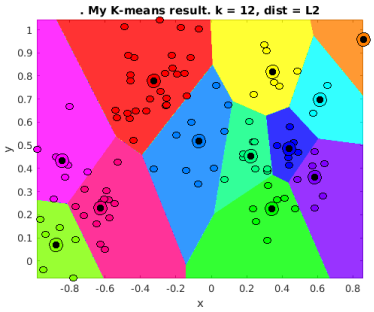


Figure 8: Decision Boundary of $K = 12$-Means.

- **AIC:** The AIC metric is a maximum-likelihood measure that penalizes for model complexity $AIC = -2\ln\mathcal{L} + 2B$ where B is the number of parameters and $\mathcal{L}$ the likelihood of the model. Even though the K-means algorithm does not provide a likelihood estimate of the model it can be formulated as a metric based on RSS as follows:

$$AIC_{RSS} = RSS + 2B \tag{7}$$

  where $B = (K * N)$ for K clusters and N dimensions.

- **BIC:** The BIC metric goes even further and penalizes for number of datapoints as well with the following equation $BIC = -2\ln\mathcal{L} + \ln(M)B$. As in AIC_{RSS}, we can formulate a BIC value based on the RSS as follows:

$$BIC_{RSS} = RSS + \ln(M)B \tag{8}$$

  where B is as before and M is the total number of datapoints.

Choosing the Optimal K

To choose the optimal K for a dataset, one can run K-means by increasing monotonically the number of clusters and plotting the results. Since the output of K-means is not deterministic, we should run K-means for `repeats` times, generally 10, and take the **mean** or *best* of those runs as the result, in this implementation we will take the mean. For the RSS metric, the 'elbow' or 'plateau' method can be reliable for certain datasets. This method finds the optimal K at the elbow of the curve;i.e. when the values stop decreasing rapidly Figure 9). For the AIC and BIC metrics, the optimal K is found as the minimum value in the curve (Figure 10).

4.2 Spectral Clustering

The approach for finding the optimal K presented in the previous sub-section can be misguided if each data cluster is not centered around a origin. Figure 13(c) illustrates an example in 2D in which the optimal number of cluster ($K = 6$) does not match our intuition for two clusters. Spectral clustering provides a more expressive alternative. By projecting the points into a non-linear embedding and analyzing the eigenvalue spectra of the **K**ernel matrix (in Kernel PCA), the **L**aplacian matrix (in Laplacian Eigenmaps),

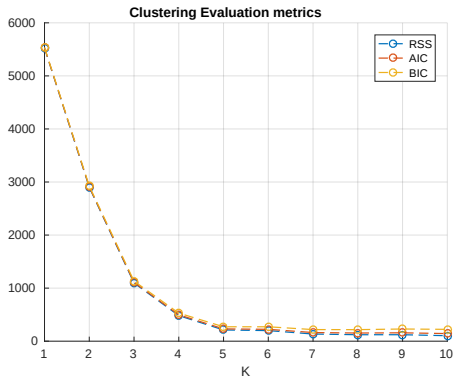


Figure 9: Clustering Evaluation metrics for the **2D-GMM** dataset. Optimal $K = 4$,`repeats=10`

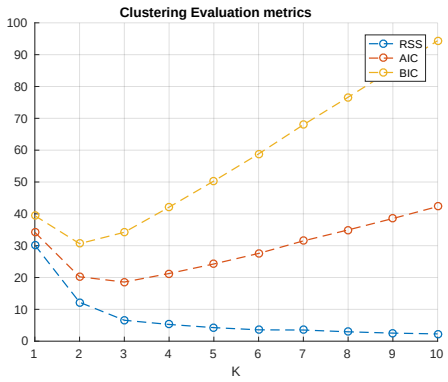


Figure 10: Clustering Evaluation metrics for the **2D-Ripley** dataset. Optimal $K = 3$, `repeats=10`.

or the **S**imilarity matrix (in Isometric Mapping), one can model a larger class of data distributions.

In addition, it turns out that one can deduce the number of clusters present in the data from the eigenvalues. Thus we avoid the expensive exhaustive search required to find the optimal K. This expensive search which usually goes by the name of *Model Selection* is not completely avoided though, since we still need to choose the hyperparameters of the kernel in spectral clustering. The next sections includes tools and build intuition for this selection process.

4.2.1 Using kernel methods to determine the number of clusters

We will study how Kernel PCA can be used to help to determine the number of clusters present in a dataset. To recall, all kernel methods are based on the computation of the so-called Gram matrix $\mathbf{K} \in \mathbb{R}^{M \times M}$,where M is the number of samples in your dataset. So be careful, if you use a kernel method on large dataset, it might take a very long time to compute this matrix. The way you can use matrix $\mathbf{K}$ to determine the number of clusters is through inspecting its eigenvalues.

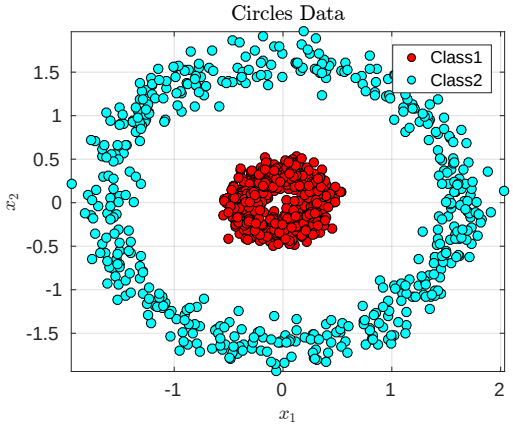
Gram eigenvalues The Gram matrix is symmetric positive semi-definite (PSD), and as you have seen in class, it can be decomposed into its eigenvalues and eigenvectors,

$$\mathbf{K} = \mathbf{V}\Lambda\mathbf{V}^{\mathrm{T}} \tag{9}$$

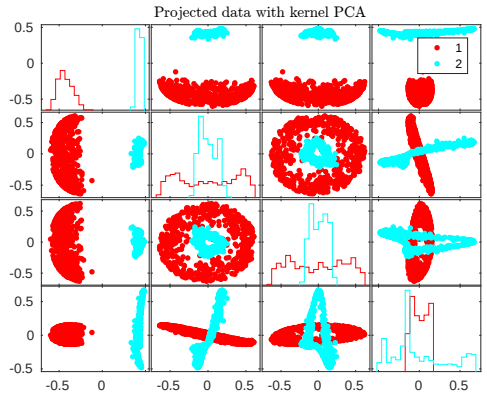
where $\mathbf{V} \in \mathbb{R}^{M \times M}$ are the eigenvectors α_i and $\Lambda \in \mathbb{R}^{M \times M}$ is diagonal matrix containing the eigenvalues λ_i. In regular PCA the eigenvalues conveyed how much of the variance in the original signal is preserved in the projected space. Usually one would keep as many eigenvectors as necessary to explain at least 90% of the variance of the original data. Hopefully, this will result in a significant dimensionality reduction. What about the eigenvalues of the Gram matrix? They have exactly the same interpretation as standard eigenvalues.

Properties of (Kernel) PCA: If we use a kernel that yields a PSD kernel matrix, we know that we are in fact performing (linear) PCA on the features $\phi(\mathbf{x}) \in \mathbb{R}^F$. Consequently, all mathematical and statistical properties of PCA (see, e.g., Jolliffe, 1986; Diamantaras & Kung, 1996) carry over to Kernel PCA, with the difference that they become statements concerning the feature space $\mathbb{R}^F$ rather than the original space $\mathbb{R}^N$

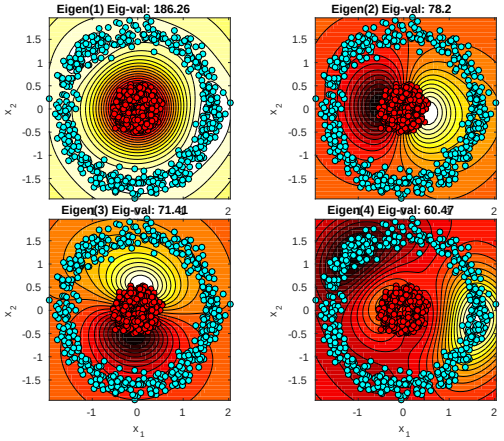
`Illustrative example:` Open the **TP2_spectral_clustering.m** MATLAB script and load the circles dataset shown in Figure 11(a), by running code sub-block `(1a)`. Then perform Kernel PCA with a chosen kernel, hyper-parameters and number of eigenvectors to retain with code sub-block `(2a)`. The result of the projection (on its first 4 eigenvectors) is illustrated in Figure 11(b). You can then inspect the eigenvalues and eigenvectors in Figures 11(c) and 11(d), which can be generated running code sub-block `(2b)`.



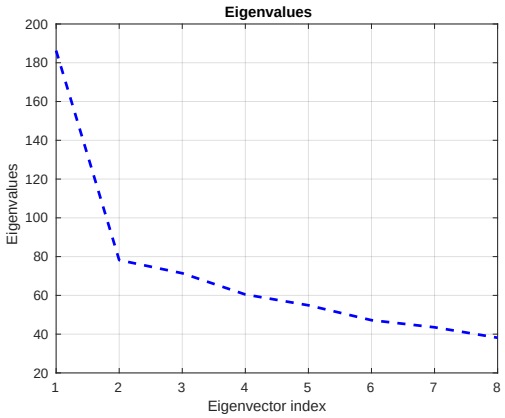
(a) Original Dataset



(b) KPCA Projections on first 4 Eigenvectors



(c) Dual Eigenvector Isolines

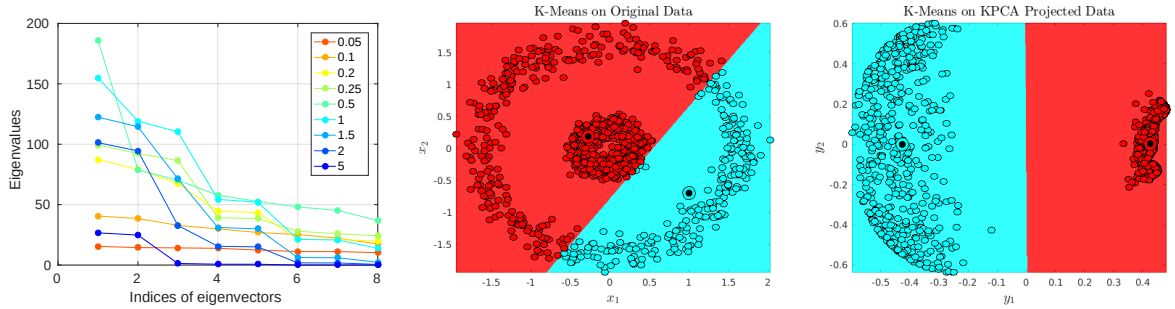


(d) Eigenvalues of first 8 Eigenvectors

Figure 11: Projected Circles Dataset with Kernel (RBF) PCA with $\sigma = 0.5$.

If Kernel PCA manages to find the appropriate number of clusters, there should then be exactly one eigenvector per cluster. To see if you are setting the hyperparameters correctly you can look at the isoline plots of different eigenvectors to see if the clusters are getting encoded by an eigenvector. As can be seen in Figure 11, the kernel type (RBF) and hyper-parameter $\sigma = 0.5$ were carefully chosen to yield the expected $K = 2$. However, if one does not choose this parameter appropriately, the result of Kernel PCA will not be satisfactory. To select the appropriate kernel parameter, one can do a grid search (as in the previous practical). For simplicity we only used the RBF kernel function, see Figure 12(a). This can be reproduced by running code sub-block `(2c)`. The best parameter is the one that yields a *dominant* set of eigenvectors. In the case of the RBF

kernel, if σ is too big or too small, all the eigenvectors have an equal amount of variance associated to them. For this specific dataset, we can see that a suitable range for the width is $\sigma = [0.25 - 2]$. In this range it is quite clear that a dip always occurs after the 2nd and 3rd eigenvector. However, the steepest dip is when $\sigma = 0.5$, which corresponds to the parameter that was carefully selected by visual analysis of the dataset. From this information we can gather that possibly $K = 2$ with $\sigma = 0.5$.



(a) Grid Search RBF Kernel (b) K(2)-Means on Original Datasets (c) K(2)-Means on Projected Datasets

Figure 12: (a) Grid Search of RBF Kernel, (b) $K(2)$-Means on Original Dataset and (c) $K(2)$-Means on KPCA Projected ($V^1 \times V^2$) Dataset

By running code sub-block `(5a)` and `(5b)`, one can perform K-means with the selected K on the original dataset and on the projected dataset, respectively. This should produce decision boundaries similar to those depicted in Figure 12(b) and 12(c).

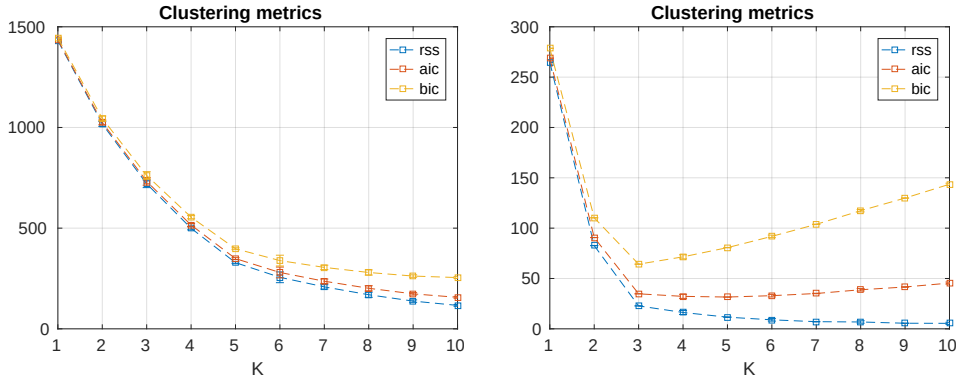
How does this compare to doing model selection? You can run a model selection procedure with RSS, AIC and BIC on the original dataset and the projected dataset, by running code sub-block `5c`:

```
%% 5c) Perform kmeans Model Selection (RSS, BIC, AIC)
cluster_options              = [];
cluster_options.method_name = 'kmeans';
repeats                      = 10;
Ks                           = 1:10;
[mus, stds]  = ml_clustering_optimise(X',Ks,repeats,...);
```

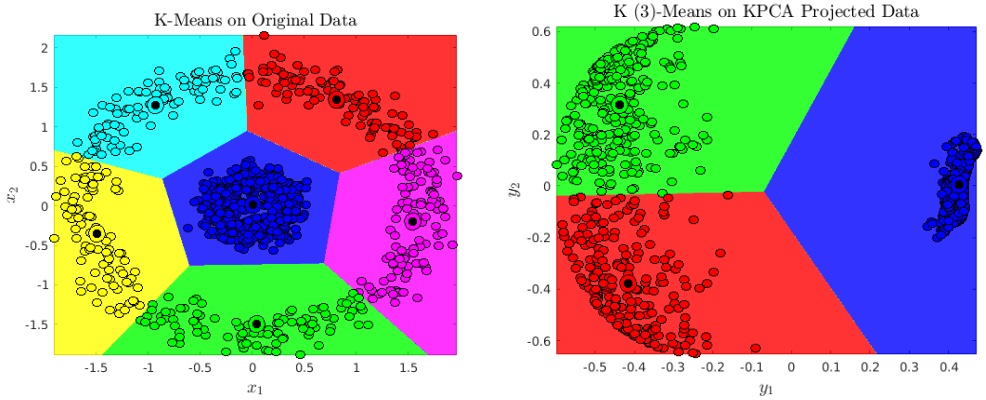
where `X` denotes the original dataset, `repeats` is the number of times each K will be repeated and `Ks` is the range of K to evaluate. If one wants to apply this on the projected dataset simply substitute `X` for `Y`. When modified properly, this code block should generate Figure 13(a) for model selection on `X` and Figure 13(b) for model selection on `Y`. Where `Y` can be defined in the same code block by choosing a projection:

```
%%% Choose between KPCA - LAP - ISO
algo = 'KPCA';
% Selected Projected Data
if strcmp(algo, 'KPCA')
    Y = proj_KPCA_X(:,[1 2])';
end
```

On the original dataset the optimal number of K clusters is around 6-7, but on the Kernel PCA projected data it is around 3-4. The optimal clusters for each dataset can be seen in Figure 13(c) and 13(d). To conclude, neither model selection runs yield the expected clusters $K = 2$ which are easily determined through the eigenvalues of Kernel PCA.



(a) Model Selection Original Dataset (b) Model Selection Projected Dataset



(c) Model Selection Original Dataset $K(6)$ (d) Model Selection Projected Dataset $K(3)$

Figure 13: K-Means Model Selection on (a) Original Dataset and (b) Kernel PCA Projected Dataset with $\sigma = 0.5$. According to these model selection procedures (c) Optimal $K = 6$ for the original dataset and (d) Optimal $K = 3$ for the projected dataset.

4.2.2 Using spectral methods to determine the number of clusters

Another alternative to determining the number of clusters, K, is to use the eigenvalues of the graph Laplacian, given by Laplacian Eigenmaps. When the similarity graph is not fully connected, the multiplicity of the eigenvalue $\lambda = 0$ gives us an estimation of K.

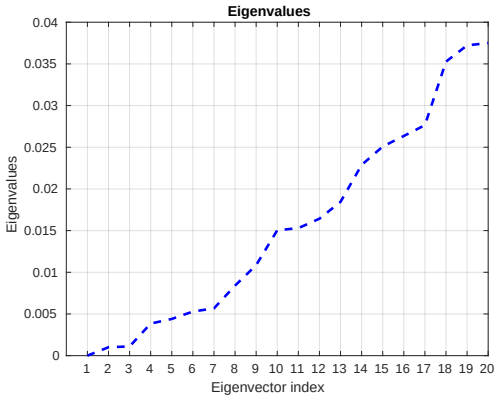
Laplacian eigenvalues The Laplacian Eigenmaps is a non-linear projection technique which is based on the eigenvalue decomposition of a scaled similarity matrix (or normalized graph Laplacian) $\mathbf{L} = \mathbf{D} - \mathbf{S}$. If we perform an eigenvalue decomposition of this matrix

$$\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^{\mathrm{T}}, \tag{10}$$

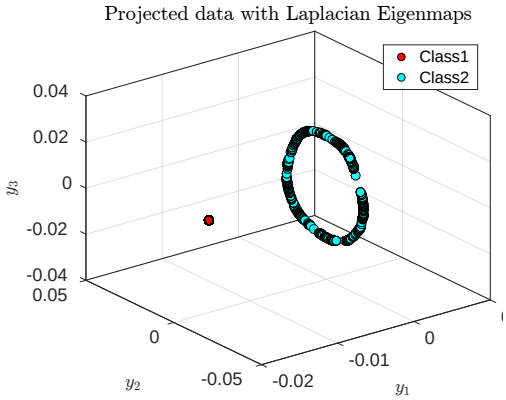
where $\mathbf{U} \in \mathbb{R}^{(M\times M)}$ are the eigenvectors and $\Lambda \in \mathbb{R}^{(M\times M)}$ is a diagonal matrix containing the eigenvalues, we can then order the eigenvalues by increasing order : $\lambda_0 = 0 \leq \lambda_2 \leq$

... $\leq \lambda_M$, where M is the number of datapoints. Looking at the multiplicity of eigenvalue $\lambda_0 = 0$ or more generally at the eigenvalues which are closest to 0 provides information about the partitioning of the similarity graph built on the data and so, an indication about the number of clusters K we could find.

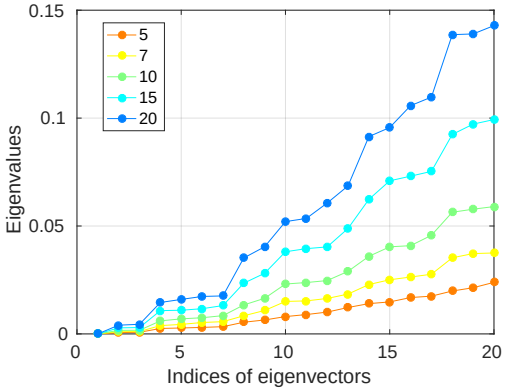
`Illustrative example:` In the MATLAB script: **TP2_spectral_clustering.m**, you will find the necessary code snippets to run the Laplacian Eigenmaps for clustering analysis. We begin by loading the same circles dataset as before (Figure 11(a)) with code sub-block `(1a)`. Then you proceed to run code sub-block `(3a)` which will compute the Laplacian Eigenmaps (with a chosen kernel width, neighborhood for Laplacian Eigenmaps and number of eigenvectors to retain). A plot of the eigenvalues of the graph Laplacian and the result of the projection with Laplacian Eigenmaps is shown in Figure 14(a) and 14(b). As in the previous case, the hyper-parameters $\sigma = 1$ and $k = 7$ (for k neighbors) were carefully selected to yield the desired projection. By looking at the eigenvalue plot (Figure 14(a)) we can see that the first 3 eigenvectors have eigenvalues very close to 0, this suggests that $K = 3$. This is not the ideal clustering we expected, however, given the projection (Figure 14(b)) it seems like the most plausible one. Hence, the problem is still the selection of the optimal hyper-parameters; i.e. the appropriate width of the



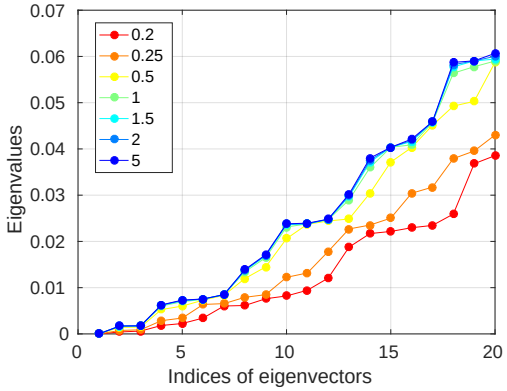
(a) Laplacian eigenvalues for $\sigma = 1$ and $k = 7$



(b) Laplacian Projections on first 3 Eigenvectors



(c) Grid Search on k-Neighbors with $\sigma = 1$



(d) Grid Search on σ with $k = 10$

Figure 14: Laplacian Eigenmaps Projection Analysis Circles Dataset.

kernel function σ and also the appropriate number of k neighbors to construct the graph. This can be addressed by doing a grid search over the width σ and neighborhood k. By running code sub-blocks `(3b)` and `(3c)`, we can generate the grid search results shown in Figure 14(c) and 14(d). In this case, the number of neighbors doesn't seem to have an influence. However, we need to have a sufficiently large (≥ 0.3) σ in order to observe a gap in the curve which enables to determine the number of clusters. If not, all the eigenvalues are close to 0 and it's hard to get K. Here we can estimate that the multiplicity of the eigenvalue $\lambda_0 = 0$ is three and so $K = 3$. Now, to apply clustering on this projected dataset, we can modify the `Y` variable in code sub-block `(5b)`:

```
%% 5b) Perform kmeans clustering on the projected data
%%% Choose between KPCA - LAP - ISO
% algo = 'KPCA';
algo = 'LAP';
% algo = 'ISO';

% Selected Projected Data
if strcmp(algo, 'KPCA')
    Y = proj_KPCA_X(:,[1:2])';
elseif strcmp(algo, 'LAP')
    Y = proj_LAP_X(:,[1:2])';
...
end
```

By changing `Y = proj_LAP_X(:,[1 2])';` we are selecting the first two projected dimensions of the Laplacian Eigenmap embeddings. By running the subsequent code, we can plot the decision boundaries in 2-D, if a 2-D projection was selected, as shown in Figure 15. Otherwise, one can plot the labeled dataset in N-D for an N-Dimensional projection, as shown in Figure 16.

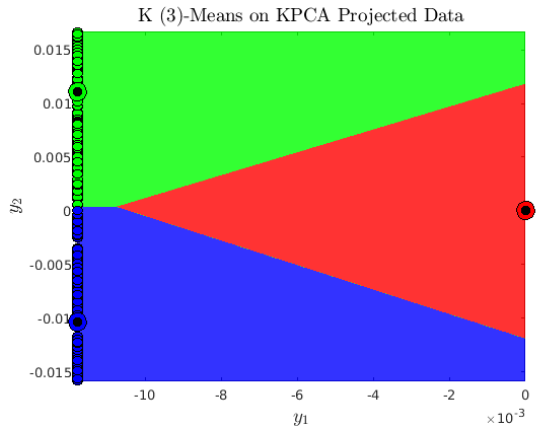


Figure 15: Circles Dataset Projected on first 2 Eigenvectors of Laplacian and clustered with $K = 3$.

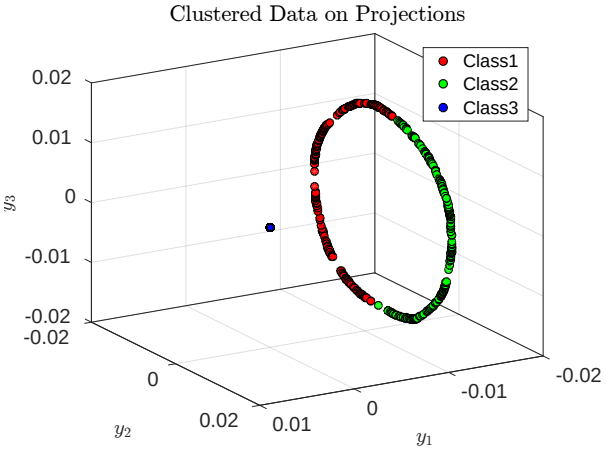


Figure 16: Circles Dataset Projected on first 3 Eigenvectors of Laplacian and clustered with $K = 3$.

Isomap eigenvalues To recall, the Isomap technique feeds a matrix of pair-wise distances $\mathbf{S}$ to the MDS algorithm. This matrix is, in fact, a Gram matrix. MDS then yields a low-dimensional output through the eigenvalue decomposition of this centered Gram matrix

$$\mathbf{S}' = \mathbf{U}\Lambda\mathbf{U}^T, \tag{11}$$

as before, $\mathbf{U} \in \mathbb{R}^{(M\times M)}$ are the eigenvectors and $\Lambda \in \mathbb{R}^{(M\times M)}$ is a diagonal matrix containing the eigenvalues. This matrix has the same rank and eigenvalues (up to constant factor) of the Covariance matrix in PCA (as in Kernel PCA); hence, a large gap between the *top* and the remaining eigenvalues, indicates a good approximation of the lower-dimensional subspace. If your data is already quite disconnected (as is in our illustrative example), one has to take special considerations when applying this algorithm. When the neighborhood graph is not completely connected, two things might happen depending on the Isomap implementations:

1. Only the largest connected component is embedded.
2. Each connected component is embedded separately.

This might seem like a drawback; however, if one is getting K disconnected components from the neighborhood graph, this is a clear indication that your data has *at least* K clusters. The necessary code for running the Isomap algorithm (and plotting the eigenvalues of the Gram matrix) is provided in code block `(4)` in the accompanying MATLAB script: **TP2_spectral_clustering.m**. When testing this on the circles dataset, you will see that only the largest connected component is being projected into a low-dimensional representation.

4.3 Comparison of Spectral Clustering Variants

External Clustering Metrics Comparing the results of different clustering algorithms is often difficult when the labels of the expected clusters (*classes*) are not given. However, when the labels are available one can use the $\mathcal{F}$-measure to compare different clustering results. This is the case for the datasets that will be used in the following task.

The $\mathcal{F}$-measure is a well-known classification metric which represents the harmonic mean between Precision ($P = \frac{TP}{TP+FP}$) and Recall ($R = \frac{TP}{TP+FN}$). In the context of clustering, Recall and Precision of the k-th cluster wrt. the j-th class are $R(s_j, c_k) = \frac{|s_j \cap c_k|}{|s_j|}$ and $P(s_j, c_k) = \frac{|s_j \cap c_k|}{|c_k|}$, respectively, where $\mathcal{S} = \{s_1, \ldots, s_J\}$ is the set of classes and $\mathcal{C} = \{c_1, \ldots, c_K\}$ the set of predicted clusters. s_j is the set of data-points in the j-th class, whereas c_k is the set of data-points belonging to the k-th cluster. The $\mathcal{F}$-measure of the k-th cluster wrt. the j-th class is,

$$\mathcal{F}_{j,k} = \frac{2P(s_j, c_k)R(s_j, c_k)}{P(s_j, c_k) + R(s_j, c_k)}, \tag{12}$$

and the F-measure for the overall clustering is then computed as,

$$\mathcal{F}(\mathcal{S}, \mathcal{C}) = \sum_{s_j \in \mathcal{S}} \frac{|s_j|}{|\mathcal{S}|} \max_k \{\mathcal{F}_{j,k}\}. \tag{13}$$

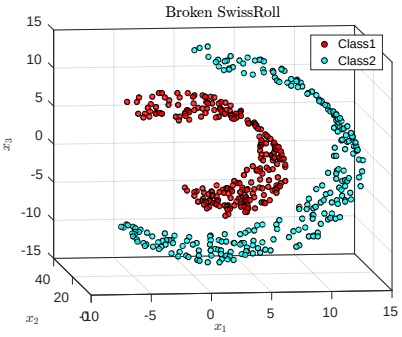
TASK 2: Compare Spectral Clustering Variants

Using the code provided in the MATLAB script: **TP2_spectral_clustering.m**, your task will be to find the best non-linear projections technique to cluster the following non-linearly separable datasets:

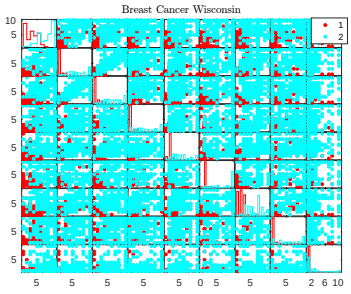
Broken Swiss Roll	:	3D data set of (2) non linearly separable clusters.
Breast-cancer-Wisconsin	:	Medical dataset taken from the UCI database: (2) classes / (9) dimensions.
Digits	:	8×8 digit images. (6) classes / (64) dimensions.

Before blindly testing all of the *Spectral Clustering* variants, you should analyze each dataset and try to deduce which algorithm would be the best fit for the dataset at hand. In order to address this task, try answering the following questions:

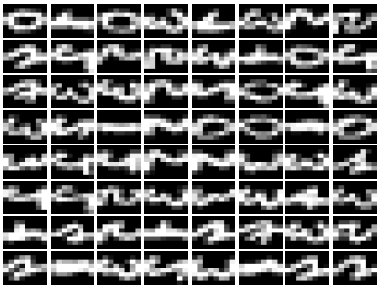
- How many clusters can you find with Kernel PCA?
- How many clusters can you find with Laplacian Eigenmaps?
- How many clusters can you find with Isomap?
- How can I find the optimal range of hyper-parameters for each method?
- How many clusters are suggested by the *Model Selection* procedure?
- Which method gives the best result according to the true class labels (i.e. $\mathcal{F}$-measure)?



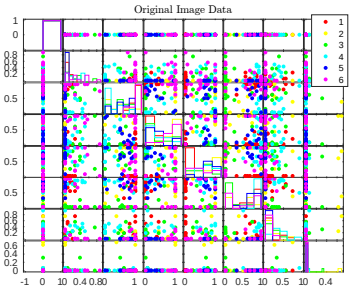
(a) Broken Swiss Roll Dataset



(b) Breast Cancer Dataset



(c) Samples of Digits Dataset



(d) 8 Dimensions of the Digits dataset

Figure 17: Non-linearly separable datasets for **TASK 2**.

To load each dataset, run code sub-block `(1b)` for the Broken Swiss Roll dataset (Figure 17(a)), sub-block `(1c)` for the Breast Cancer Dataset (Figure 17(b)) and sub-block `(1d)` for the Digits dataset (Figure 17(c) and 17(d)).

5 Kernel K-Means

Interestingly, an approach that tackles clustering of non-linearly separable datasets head on is Kernel K-means, where, prior to clustering, points $\mathbf{x}^i$ are mapped to a higher-dimensional feature space using a nonlinear function $\phi(\mathbf{x}^i)$, then classical K-means clustering with norm-2 is applied in feature space, yielding non-linear boundaries in the original space. As opposed to *Spectral clustering*, **kernel K-means** does clustering and non-linear embedding simultaneously. Both of these algorithms can be seen as two different approaches that solve the same objective. To recall, Kernel K-Means seeks to minimize the total squared distance between each point and its closest centroid in *feature space* with the following cost function,

$$J(\mu_1, \ldots, \mu_k) = \sum_{k=1}^{K} \sum_{\mathbf{x}^l \in C^k} \left|\left| \phi(\mathbf{x}^i) - \frac{\sum_{\mathbf{x}^l \in C^k} \phi(\mathbf{x}^l)}{M_k} \right|\right|^2 . \tag{14}$$

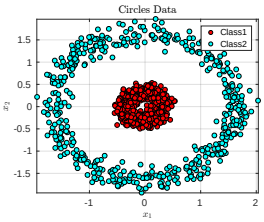
where C^k denotes the k-th cluster and M_k the number of points in the k-th cluster. The proposed solution to (14) is an iterative procedure as K-Means, however, in the expectation step, the data points are assigned to the closest centroid in *feature space* with the following distance:

$$\underset{k}{\text{argmin}} \quad d(\mathbf{x}^i, C^k) = \underset{k}{\text{argmin}} \left(k(\mathbf{x}^i, \mathbf{x}^i) - \frac{2\sum_{\mathbf{x}^j \in C^k} k(\mathbf{x}^i, \mathbf{x}^j)}{M_k} + \frac{\sum_{\mathbf{x}^j, \mathbf{x}^l \in C^k} k(\mathbf{x}^j, \mathbf{x}^l)}{(M_k)^2} \right) \tag{15}$$

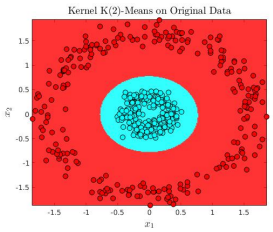
With such a distance metric one can cluster non-linearly separable datasets, like the circles dataset, shown in Figure 18(b).

In the accompanying MATLAB script: **TP2_kernel_kmeans.m** we provide the necessary code to run Kernel K-means on your preferred dataset, plot the non-linear decision boundaries and the isolines of the eigenvectors of the kernel matrix. In Figure 18(b) and 18(c) we illustrate the resulting decision boundaries of applying Kernel K-Means on the circles dataset with $\sigma = 0.5$ and $\sigma = 1$. In the case of $\sigma = 1$, one can see that even though the isolines of the eigenvectors seem to perfectly separate the two clusters (Figure 18(e)) the initial centroids in feature space misguide the partition; resulting in a poor clustering. However, if you run this code block multiple times, you will see how in some cases the expected partitioning is achieved. One can see that, the main advantage of Kernel K-means; i.e. its ability to model non-linear boundaries; is burdened by its sensitivity to initialization (which is random) and to the kernel's hyper-parameters (as in Kernel PCA).

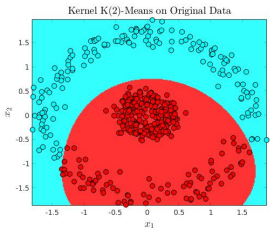
An important characteristic of Kernel K-Means is its relationship to *Spectral Clustering*. It has been shown, that the objective function of a modified kernel K-Means (i.e. weighted Kernel K-Means) is equivalent to a special case of *Spectral Clustering* (described through normalized-cuts) [4]. In fact, one can solve the Kernel K-Means objective function through Kernel-PCA, by initially embedding the data-points with the first K top eigenvectors and then applying K-Means on the low-dimensional embedding. This is what we have done in the previous task! The main difference is that in the *Spectral Clustering* approach, we had the freedom to choose the dimensionality, p, of the embedded space and the clusters, K, independently. In Kernel K-Means this is not the case.



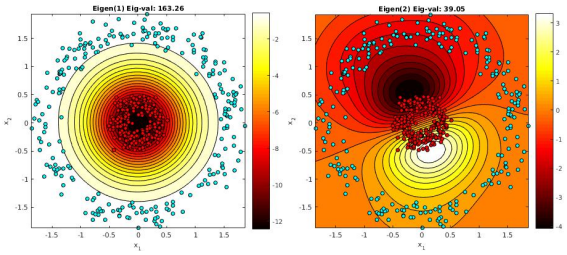
(a) Original Dataset



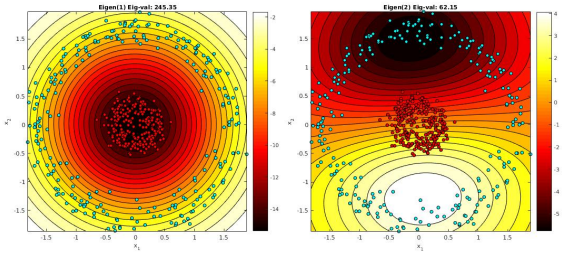
(b) Kernel $K(2)$-Means with $\sigma = 0.5$



(c) Kernel $K(2)$-Means with $\sigma = 1$



(d) Isolines for Kernel $K(2)$-Means with $\sigma = 0.5$



(e) Isolines for Kernel $K(2)$-Means with $\sigma = 1$

Figure 18: Kernel K-Means on Circles Dataset with $K = 2$

Moreover, in *Spectral Clustering* the embedded space doesn't necessarily have to be the first top eigenvectors, as in Kernel K-Means.

TASK 3: Evaluate Kernel K-Means on Different Datasets

In this task we will try to generate complex 2D datasets where Kernel K-Means clearly outperforms classical K-Means. By running code sub-block `1b` of the MATLAB script: **TP2_kernel_kmeans.m**, you can load the drawing GUI shown in Figure 19. After drawing your data, you should click on the `Store Data` button, this will store a data array in your MATLAB workspace. After running the following sub-blocks the data and labels will be stored in two different arrays: $\mathbf{X} \in \mathbb{R}^{2\times M}$ and $\mathbf{y} \in \mathbb{I}^{M}$, which can then be used to visualize and manipulate in MATLAB (Figure 20). **Try** to generate examples for both polynomial and RBF kernel.

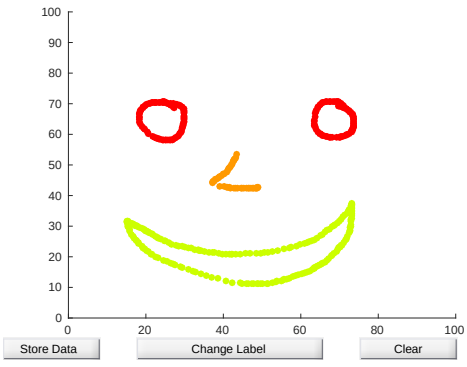


Figure 19: ML_toolbox 2D Drawing GUI.

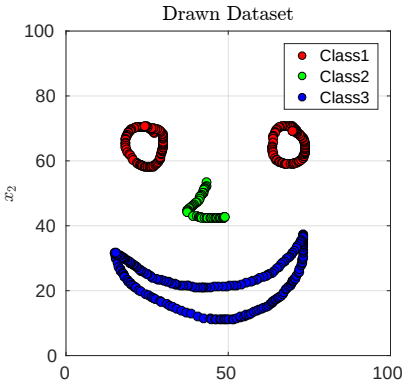


Figure 20: Data After Storing it in workspace.

References

[1] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 15(6):1373–1396, June 2003.

[2] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319, 2000.

[3] S. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 28(2):129–137, September 2006.

[4] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: Spectral clustering and normalized cuts. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 551–556, New York, NY, USA, 2004. ACM.