# CS5489
# Lecture 8.2: Kernel Principal Component Analysis

Kede Ma

City University of Hong Kong (Dongguan)

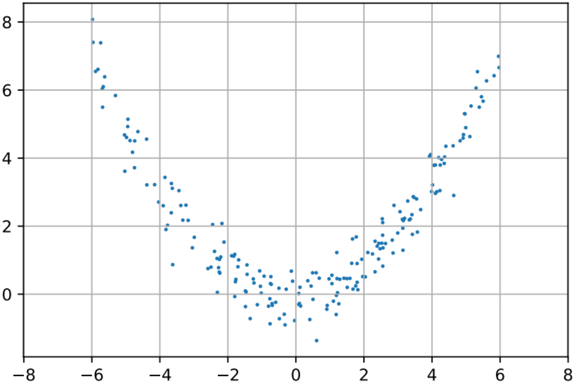
香港城市大學（東莞）
City University of Hong Kong (Dongguan)

Slide template by courtesy of Benjamin M. Marlin

# Outline

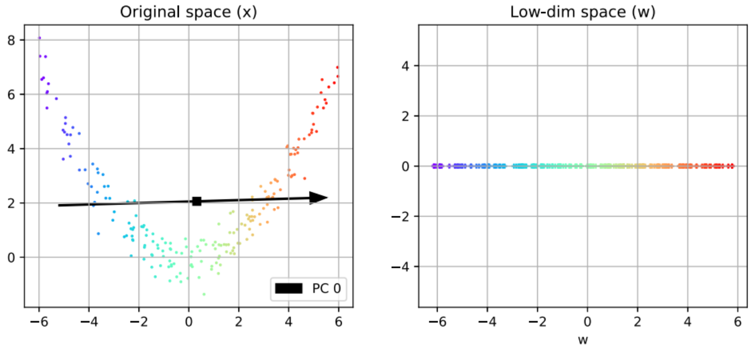
# Limitations of Linear Dimensionality Reduction

- **Question:** What if the data "lives" on a non-flat surface?



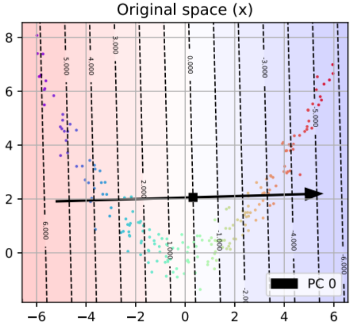
# Limitations of Linear Dimensionality Reduction

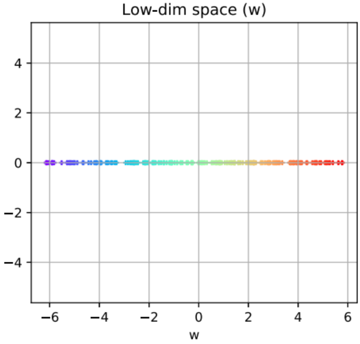
- PCA can't capture the curvature of the data
  - Purple points are close together
  - Red points are close together



# Limitations of Linear Dimensionality Reduction

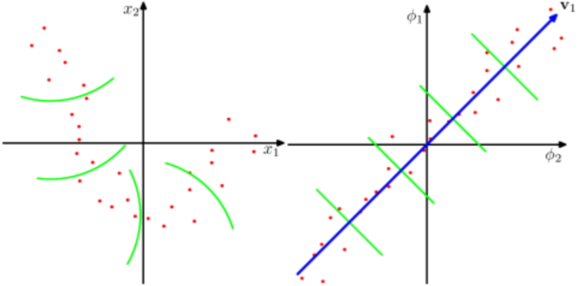
- Iso-contours of PCA projection
  - Points on the same dashed line are projected to the same PCA coefficient



# Feature Mapping

- How to project to a non-linear surface?
  - Apply a high-dimensional feature transformation to the data
    - $\mathbf{x}^{(i)} \to \phi(\mathbf{x}^{(i)})$
  - Project high-dim data to a linear surface
    - I.e., run PCA on $\phi(\mathbf{X})$
  - In the original space, the projection will be non-linear



# Feature Mapping + SVD

- Given a data set $\mathbf{X} \in \mathbb{R}^{M\times N}$ and a feature mapping function $\phi : \mathbb{R}^N \mapsto \mathbb{R}^L$ for $L > N$, we obtain the following SVD-based algorithm:
  1. Compute $\mathbf{U}, \mathbf{S}, \mathbf{V} = \mathrm{SVD}(\phi(\mathbf{X}))$
  2. Return $\boldsymbol{Z} = \mathbf{US}$

# Feature Mapping + PCA

- Given a data set $\mathbf{X} \in \mathbb{R}^{M \times N}$ and a feature mapping function $\phi : \mathbb{R}^N \mapsto \mathbb{R}^L$ for $L > N$, we obtain the following PCA-based algorithm:
  1. Compute $\mathbf{\Sigma} = \frac{1}{M} \sum_i (\phi(\mathbf{x}^{(i)}) - \boldsymbol{\mu})(\phi(\mathbf{x}^{(i)}) - \boldsymbol{\mu})^T$ where $\boldsymbol{\mu} = 1/M \sum_i \phi(\mathbf{x}^{(i)})$
  2. Compute the $K$ leading eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_K$ of $\mathbf{\Sigma}$ where $\mathbf{v}_j \in \mathbb{R}^{L \times 1}$ for $j = 1, \ldots, K$
  3. Stack the eigenvectors together to form $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_K]$, where $\mathbf{V} \in \mathbb{R}^{L \times K}$
  4. Project the matrix $\phi(\mathbf{X})$ into the rank-$K$ subspace of maximum variance by computing the matrix product $\mathbf{Z} = \phi(\mathbf{X})\mathbf{V}$

# Kernel PCA

- As in classification, it becomes very expensive to use an explicit feature function to map data into a high-dimensional space
- In the basic SVD-based algorithm, there's no way to avoid this problem
- In the PCA-based algorithm, we are able to take advantage of the **kernel** trick
  - $\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$

# Kernel PCA

- Given $\phi : \mathbb{R}^N \mapsto \mathbb{R}^L$, we compute the covariance matrix in the new feature space

$$\mathbf{\Sigma} = \frac{1}{M}\sum_{j=1}^{M}\phi(\mathbf{x}^{(j)})\phi(\mathbf{x}^{(j)})^T$$

- Eigendecomposition of $\mathbf{\Sigma}$ is given by

$$\mathbf{\Sigma}\mathbf{v}_k = \frac{1}{M}\sum_{j=1}^{M}\phi(\mathbf{x}^{(j)})\phi(\mathbf{x}^{(j)})^T\mathbf{v}_k = \lambda_k\mathbf{v}_k, \forall k = 1,\ldots,L$$

- It is not hard to see that $\mathbf{v}_k$ can be expressed as

$$\mathbf{v}_k = \sum_{j=1}^{M} w_k^{(j)}\phi(\mathbf{x}^{(j)}),\ \text{where } w_k^{(j)} = \frac{1}{M\lambda_k}\phi(\mathbf{x}^{(j)})^T\mathbf{v}_k$$

# Kernel PCA

- Copy: $\mathbf{v}_k = \sum_{j=1}^{M} w_k^{(j)} \phi(\mathbf{x}^{(j)})$, where $w_k^{(j)} = \frac{1}{M\lambda_k} \phi(\mathbf{x}^{(j)})^T \mathbf{v}_k$
  - Kernel PC is a linear combination of high-dim vectors
  - $w_k^{(j)}$ are weights to be determined
- Left multiplying $\phi(\mathbf{x}^{(i)})^T$ to both sides, we have

$$\phi(\mathbf{x}^{(i)})^T \mathbf{v}_k = \sum_{j=1}^{M} w_k^{(j)} \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)}) = M\lambda_k w_k^{(i)}$$

- Defining the kernel matrix $\mathbf{K} \in \mathbb{R}^{M \times M}$
  - $K_{ij} = \mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$
- Then,

$$\sum_{j=1}^{M} K_{ij} w_k^{(j)} = M\lambda_k w_k^{(i)}$$

# Kernel PCA

- If we consider $i = 1, \ldots, M$, the above scalar equation becomes the $i$th component of the following vector equation

$$\mathbf{K}\mathbf{w}_k = M\lambda_k \mathbf{w}_k$$

  - $\mathbf{w}_k = [w_k^{(1)}, \ldots, w_k^{(M)}]^T$ is the $k$-th eigenvector of $\mathbf{K}$
  - $M\lambda_k$ is the eigenvalue of $\mathbf{K}$, which is proportional to the eigenvalue $\lambda_k$ of the covariance matrix $\mathbf{\Sigma}$ in the feature space
- Therefore, PCA on $\mathbf{\Sigma}$ is equivalent to PCA on $\mathbf{K}$
- For a new point $\mathbf{x}^*$, the $k$-th kernel PC can be obtained by projecting $\phi(\mathbf{x}^*)$ on the $k$-th eigenvector $\mathbf{v}_k$ of $\mathbf{\Sigma}$

$$\phi(\mathbf{x}^*)^T\mathbf{v}_k = \sum_{i=1}^{M} w_k^{(i)}\phi(\mathbf{x}^*)^T\phi(\mathbf{x}^{(i)}) = \sum_{i=1}^{M} w_k^{(i)}\mathcal{K}(\mathbf{x}^*, \mathbf{x}^{(i)})$$

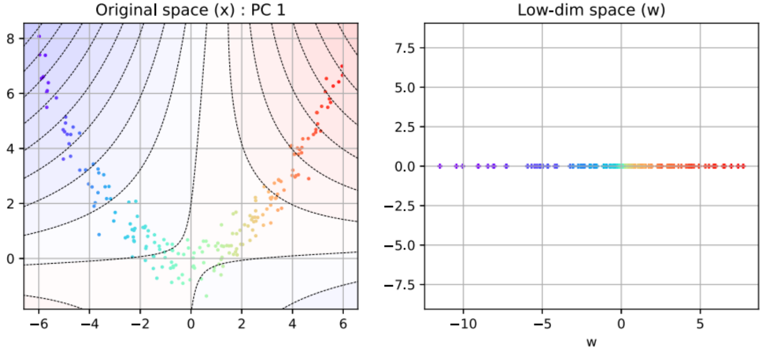
# Kernel PCA Algorithm

- Given a data set $\mathbf{X} \in \mathbb{R}^{M \times N}$ and a kernel function $\mathcal{K}$, kernel PCA can be computed as follows:
  1. Compute $K_{ij} = \mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ for all $i, j$
  2. Compute $\mathbf{K}' = (\mathbf{I} - \mathbf{1}_M)\mathbf{K}(\mathbf{I} - \mathbf{1}_M)$ where $\mathbf{1}_M$ is an $M \times M$ matrix where every entry is $1/M$
     - The goal is to zero center data points in the feature space
  3. Compute the $K$ leading eigenvectors $\mathbf{w}_1, \ldots, \mathbf{w}_K$ of $\mathbf{K}'$ along with their eigenvalues $M\lambda_1, \ldots, M\lambda_K$
  4. Compute the $k$-th PC of the projected data vector $\mathbf{z} \in \mathbb{R}^{K \times 1}$

$$z_k = \sum_{i=1}^{M} w_k^{(i)} \mathcal{K}(\mathbf{x}, \mathbf{x}^{(i)})$$

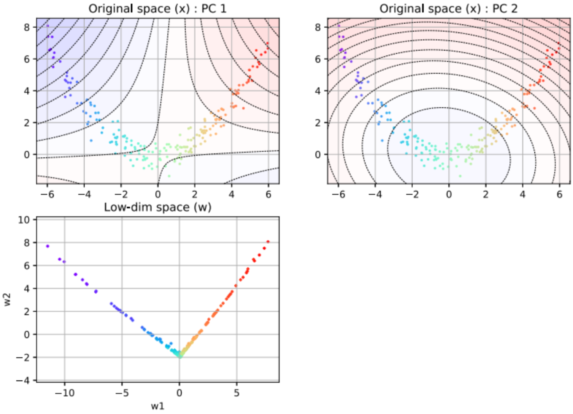
# Example

- Example using polynomial kernel
  - Purple points are further apart
  - PC coefficient corresponds to location along the data curve



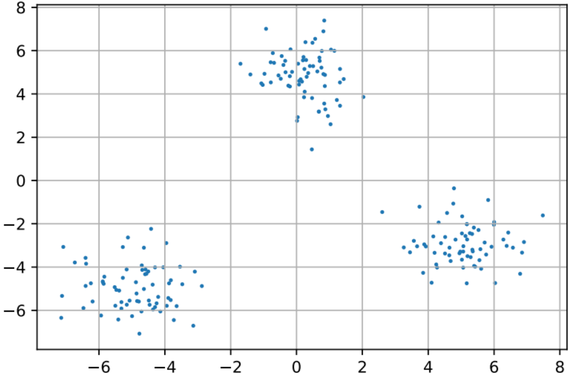
# Example

- Example: 2 PCs
  - 2nd PC corresponds to the distance from the center



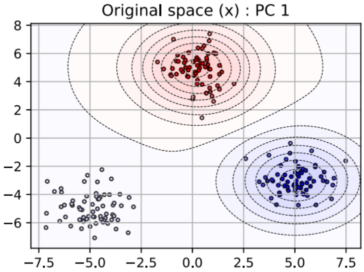
# RBF Kernel

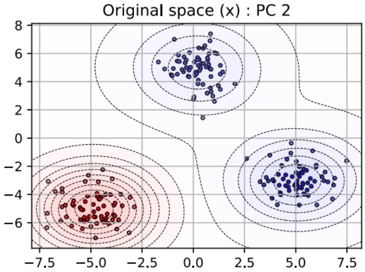
- Principal components separate the data into clusters
- Coefficient is distance to clusters
- Example: Data with 3 clusters



# Example

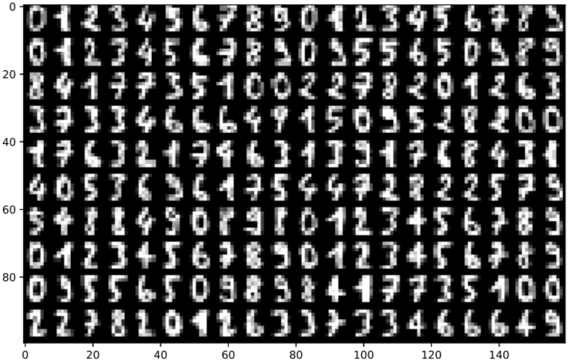
- The first 2 PCs can split the data into 3 clusters
  - The color of the data point corresponds to the coefficient value



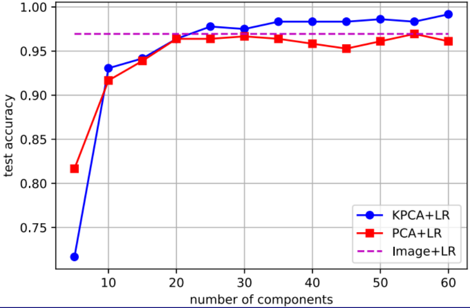
# Example on Digit Images

- $8 \times 8$ images $\rightarrow$ 64-dim vector



# Classification Experiment

- Use KPCA coefficients as the new representation
  - Train a logistic regression classifier
  - Try different numbers of components
    - Can do this efficiently by selecting a subset of KPCA components
  - Classification results on test set
    - KPCA improves the performance, compared with PCA and raw image



# Summary

- Kernel PCA uses kernel trick to perform PCA in high-dimensional space
    - Coefficients are based on a non-linear projection of the data
    - The type of projection is based on the kernel function selected
- Using RBF kernel, KPCA can split the data into clusters
- Kernel PCA can provide an effective pre-processing step for clustering methods as well as linear classification and regression methods
- However, exact computation of kernel PCA can be expensive because the size of the matrix to be decomposed is $M \times M$