Nonlinear Component Analysis as a Kernel Eigenvalue Problem

Jifu Zhao

Huan Yan

Vikram Idiga

Pavan Kumar Nadiminti

Nov. 16, 2016, Urbana

Content

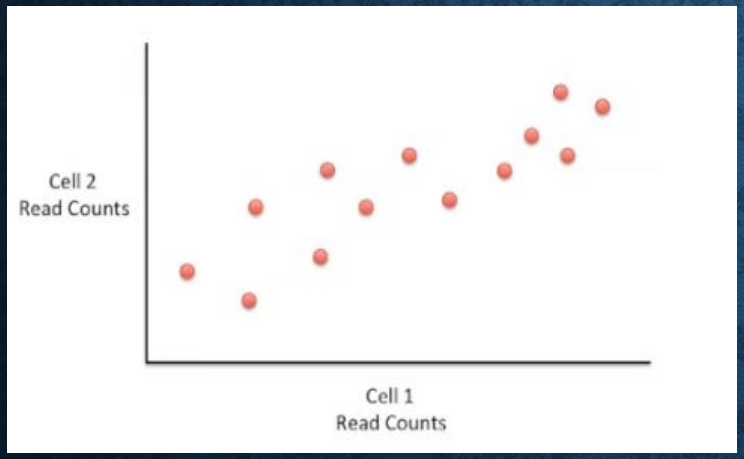
- Introduction
  - Motivation
  - PCA Recap
- Limitations of PCA
- Theoretical framework
- Kernel PCA
- Algorithms
- Examples
- Summary & Extension

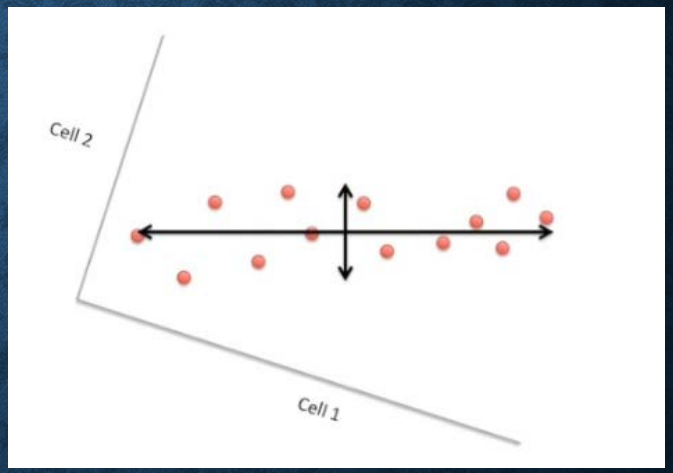
Motivation for Performing PCA

- **Dimensionality reduction**
  - When we have a very high dimensional data in which some dimensions are not really important we would want to ignore such dimensions
- **Feature extraction and data visualization**
  - We want to identify important features in a dataset, or maybe we want to visualize the data and want to know what are the best dimensions in which to visualize the data
- **Identifying major sources of variance**
  - We want to know what is the major sources of variance in our data set

PCA – a recap

- Invented by Karl Pearson, 1901*.
- Main intuition
  - Convert possibly correlated variables to linearly uncorrelated variables through orthogonal transformation





the dot products of the data points with the principal components give the projections along each of those components

* https://en.wikipedia.org/wiki/Principal_component_analysis
https://goo.gl/WbEQAb

What it means

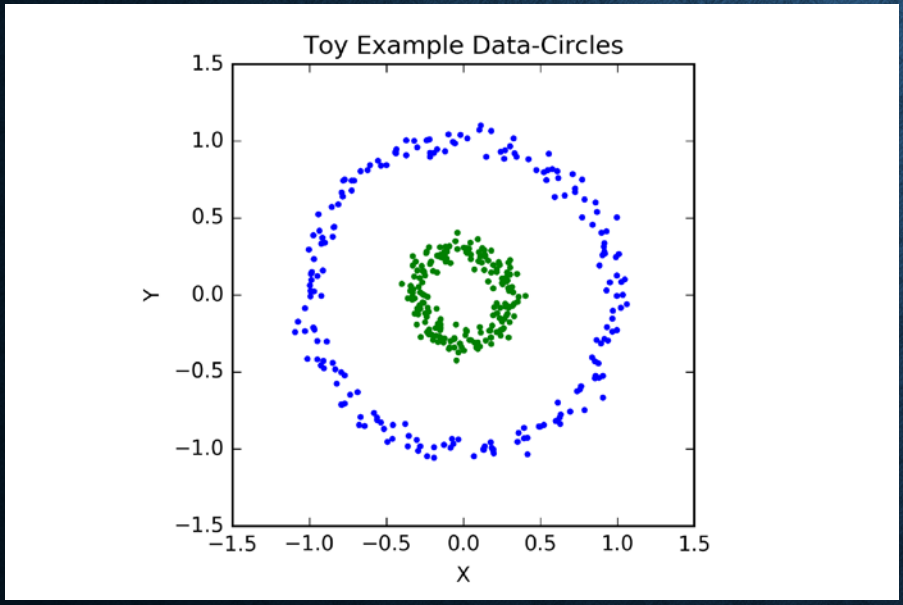
- Obtain a new set of orthogonal axes along which the data is represented better

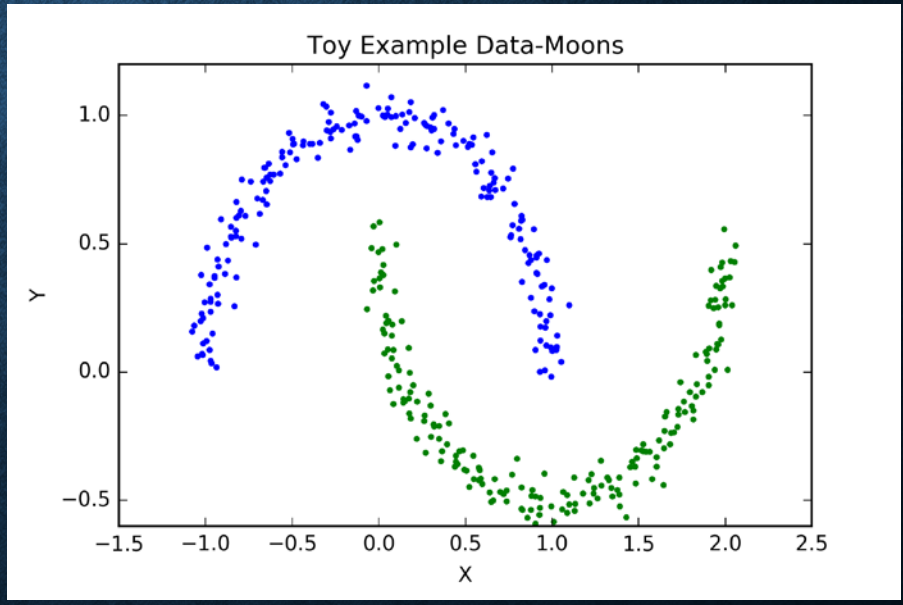
Algorithm 1 PCA in Feature Spaces

1: **procedure** PCA(X)
2: given input: $X_{n\times m} \leftarrow \left[\mathbf{x}_1; \mathbf{x}_2; \cdots; \mathbf{x}_n\right]^T$
3: de-mean (or standardize): $x_{ij} \leftarrow x_{ij} - \bar{x}_j$ *or* $x_{ij} \leftarrow \frac{x_{ij}-\bar{x}_j}{s_j}$
4: calculate covariance matrix: $Cov \leftarrow \frac{1}{n} X^T X$
5: singular value decomposition (SVD): $[U, S, V] \leftarrow svd(Cov)$
6: choose the first k eigenvectors: $E_{m\times k} \leftarrow \left[\mathbf{u}_1; \mathbf{u}_2; \cdots; \mathbf{u}_k\right]$
7: project the test data $\mathbf{x}$: $\mathbf{p} \leftarrow E^T \mathbf{x}$
8: **finish**

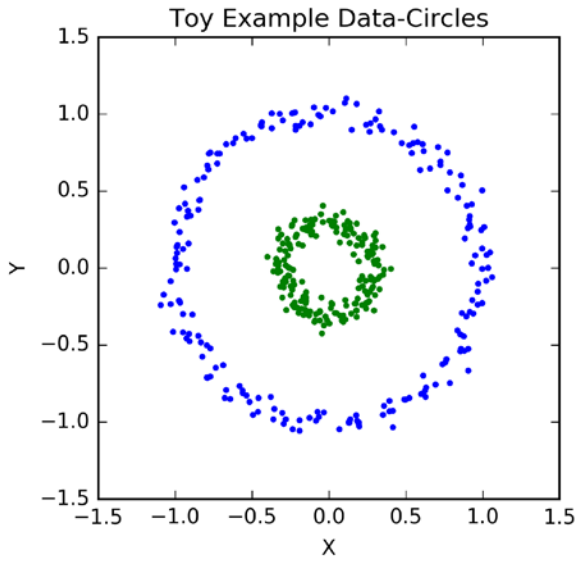
Limitations of linear PCA

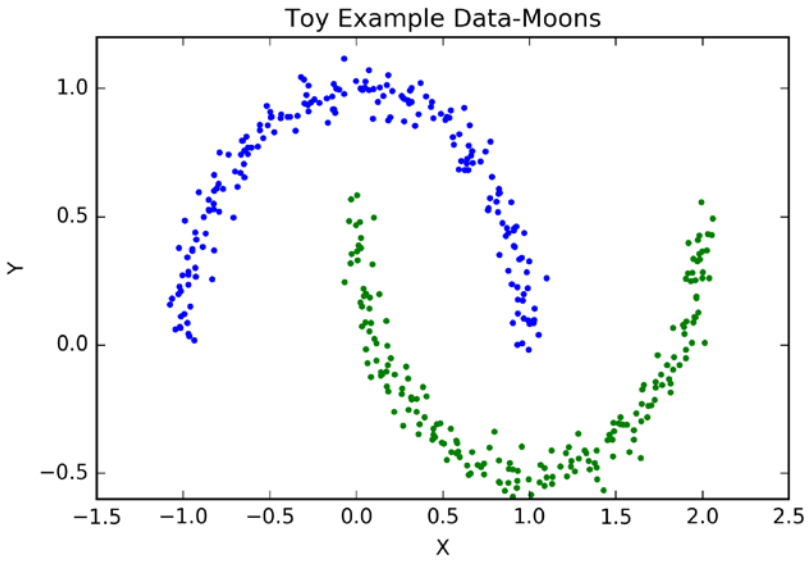
- If there are N dimensions and M data points , with M> N , then, we would be able to extract at most N features from the data set, even though there may be more features in the data.
- PCA cannot easily separate non – linear Data

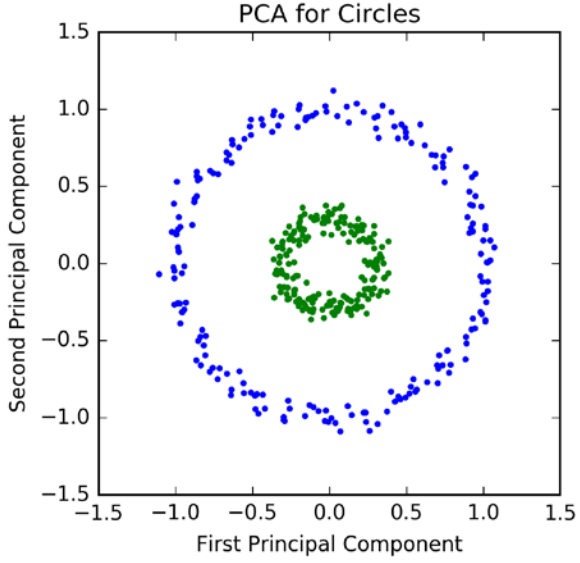


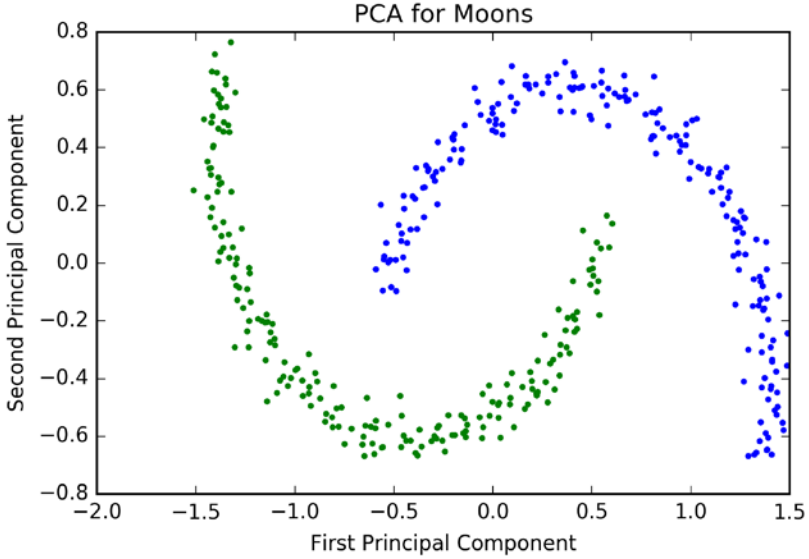


Why Kernel PCA ?

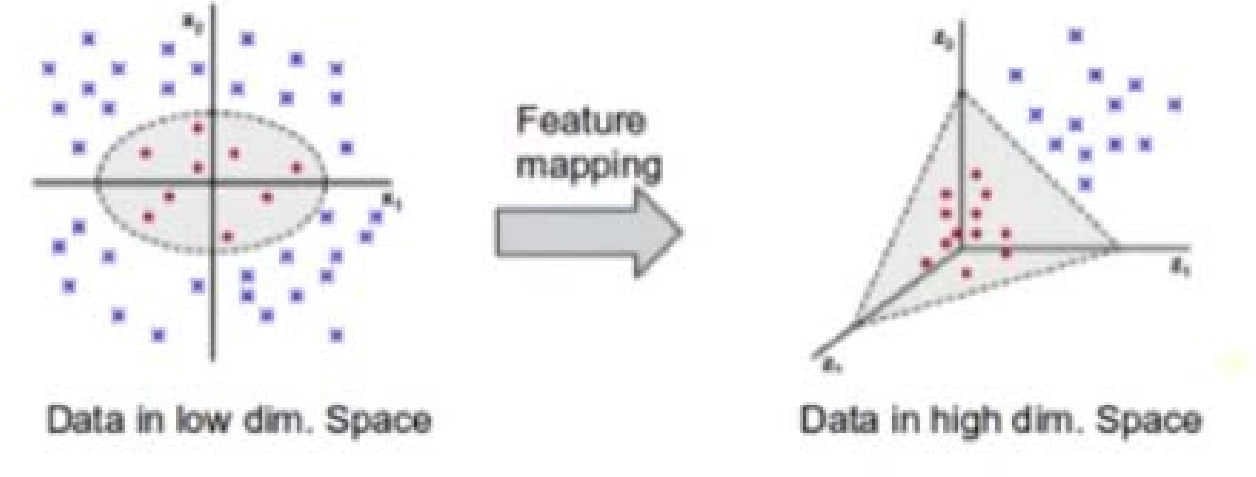








- There is no set of axes in the 2-D plane along which we can distinguish the data
- We need some non linear transformation to separate the features
- Key Idea is to be able to transform /map data into a higher dimensional space where data becomes separable in the new feature space



https://goo.gl/IVW0SM

Why kernel PCA

- Non-linear principal components result in better recognition rates for images
- Performance can be improved by using more components than is possible in the PCA
- Better, as this is essentially an eigen value problem and we are not solving any non –linear optimization problem

Theoretical framework

- First we develop the framework for normal PCA and then extend it to the non-linear case
- **It's all about the dot products**

Consider a centered data set $x_k \in RN$,

$$C = \frac{1}{M}\sum_{j=1}^{M} x_j x_j^T \qquad and \qquad \lambda \boldsymbol{v} = C\boldsymbol{v}$$

v lies in the same space spanned by x_k , as can be seen by

$$\boldsymbol{v} = \frac{1}{\lambda}(C\boldsymbol{v}) = \frac{1}{\lambda}\left(\frac{1}{M}\sum_{j=1}^{M} x_j x_j^T\, \boldsymbol{v}\right) = \frac{1}{\lambda M}\left(\sum_{j=1}^{M} x_j x_j^T\, \boldsymbol{v}\right) = \frac{1}{\lambda M}\left(\sum_{j=1}^{M} x_j\, (x_j \bullet \boldsymbol{v})\right)$$

Hence the projections (dot product) of each data point along each of the components v, is given by

$$\lambda(x_k \bullet \boldsymbol{v}) = (x_k \bullet \mathbf{C}\boldsymbol{v}) \quad for\ all\ k = 1, \ldots, M$$

- Now consider a non- linear map into another feature space :

$$\Phi : R^N \to F, \qquad x \mapsto \mathbf{X}$$

- We want to perform PCA in this new feature space.
- To perform PCA we first need to center the data in that space.
- We assume the data to be centered and later show how to deal with the case of non- centered data $\sum_{k=1}^{M} \Phi(x_k) = 0$

Repeating the steps, compute $\bar{C} = \frac{1}{n}\sum_{i=1}^{m} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T$

and $\lambda V = \bar{C} V$

the eigen value problem is equivalent to computing the projections

$$\lambda(\Phi(x_k) \bullet \boldsymbol{V}) = (\Phi(x_k) \bullet \bar{C}\boldsymbol{v}) \quad for \ all \ k = 1, \dots, M$$

- Since V is in the span of $\Phi(x_k)$, we can write $V = \sum_{i=1}^{M} \alpha_i \Phi(x_i)$
- So finding V amounts to finding the vector $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_M]^T$
- Substituting V in the projections equation , we get

$$\lambda \sum_{i=1}^{M} \alpha_i (\Phi(x_k) \bullet \Phi(x_k)) = \frac{1}{M} \sum_{i=1}^{M} \alpha_i \left(\Phi(x_k) \bullet \sum_{j=1}^{M} (\Phi(x_j)) \right) (\Phi(x_j) \bullet \Phi(x_i))$$

Now define a matrix K such that $K_{ij} := \Phi(x_i) \bullet \Phi(x_j)$

- Then the above equation transforms to

$$\mathbf{M\lambda K\alpha = K^2 \alpha \Leftrightarrow M\lambda\alpha = K\alpha}$$

- i.e the eigen value problem of finding V's has become eigen value problem of finding α^p 's the eigen vectors **α** of K .
- The eigen vectors **α** are scaled such that the eigen vectors V be come of length 1.

Kernel PCA

- Proposed by Scholkopf et al., 1998*
- Main idea
  - Expand the original feature space by non-linear transformations
  - Apply PCA in the transformed feature space
- Main drawback
  - Expanded feature space may have very high dimensions
  - Applying PCA is computationally expensive or even impossible
  - Solution -- Kernels

* Friedman, J., Hastie, T., & Tibshirani, R. *The elements of statistical learning.*

Expand the feature space

- Suppose we want to map $\vec{x}$ into a new space: $\phi(\vec{x})$
- $\vec{x}$ has d dimensions and $\phi(\vec{x})$ has m dimensions

$$m > d$$

- We need to calculate:
  - $\vec{x}\vec{x}^T$: complexity $O(d^2)$
  - $\phi(\vec{x})\phi(\vec{y})^T = \phi(\vec{x}) \bullet \phi(\vec{y})$: complexity $O(m^2)$
- What if $m \gg d$?
  - Solution: Kernels
  - k($\vec{x}, \vec{y}$) = $\phi(\vec{x}) \bullet \phi(\vec{y})$

“Kernel Trick”(1)

- **Example:**
  - $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$
  - $\phi(\vec{x}) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$ and $\phi(\vec{y}) = \begin{bmatrix} y_1^2 \\ \sqrt{2}y_1y_2 \\ y_2^2 \end{bmatrix}$
  - **Want to calculate:** $\phi(\vec{x}) \cdot \phi(\vec{y}) = x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_x^2$
  - **Define** $\mathrm{k}(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y})^2$
    - $\mathrm{k}(\vec{x}, \vec{y}) = x_1^2\, y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_x^2 = \phi(\vec{x}) \cdot \phi(\vec{y})$

“Kernel Trick”(1)

- **Example:**
  - $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$, $\phi(\vec{x}) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$ and $\phi(\vec{y}) = \begin{bmatrix} y_1^2 \\ \sqrt{2}y_1y_2 \\ y_2^2 \end{bmatrix}$
  - **Define** $\mathrm{k}(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y})^2$
    - $\mathrm{k}(\vec{x}, \vec{y}) = x_1^2\, y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_x^2 = \phi(\vec{x}) \cdot \phi(\vec{y})$
- Important finding:
  - Complexity for $\mathrm{k}(\vec{x}, \vec{y})$ is $O(d)$ instead of $O(m)$
- What’s next ?
  - Find some way to calculate $\mathrm{k}(\vec{x}, \vec{y})$ rather than $\phi(\vec{x})\phi(\vec{x})^T$

“Kernel Trick”(2)

- Covariance matrix:

$$C = \frac{1}{n}\sum_{i=1}^{m} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T$$

- Find eigenvalues and eigenvectors for:

$$\lambda \boldsymbol{v} = C\boldsymbol{v}$$

$$\lambda(\phi(\boldsymbol{x_k}) \cdot \boldsymbol{v}) = (\phi(\boldsymbol{x_k}) \cdot C\boldsymbol{v}) \text{ and } \boldsymbol{v} = \sum_{i=1}^{n} \alpha_i\, \phi(\boldsymbol{x}_i)$$

- For kernel matrix K

$$K_{ij} = \left(\phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j)\right)$$

$$n\lambda K\boldsymbol{\alpha} = K^2\boldsymbol{\alpha} \text{ or } n\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha}$$

- Projection for any given $\boldsymbol{x}$ on ith principal component

$$p_i(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_{ij} k(\boldsymbol{x}, \boldsymbol{x}_j)$$

Kernel PCA Algorithm

- Instead of finding covariance matrix, we need to calculate the kernel matrix K
- Instead of finding the eigenvalues and eigenvectors for covariance matrix, we find them for kernel matrix K

* Bishop, C. M. *Pattern Recognition and Machine Learning.*

Kernel PCA Algorithm

- To centering the data in high dimensional space:
- $\tilde{\phi}(x_i) = \phi(x_i) - \frac{1}{M}\sum_{i=1}^{M}\phi(x_i)$
- $\widetilde{K_{ij}} = (\tilde{\Phi}(x_i) \bullet \tilde{\Phi}(x_j)$ is the correct covariance matrix to use.
- However, we need to avoid explicitly calculate $\phi(x_i)$
- Instead, we directly find the corresponding expression for the centering kernel matrix K:
- $\widetilde{K_{ij}} = [\phi(x_i) - \frac{1}{N}\sum_{i=1}^{N}\phi(x_i)] \bullet [\phi(x_j) - \frac{1}{N}\sum_{i=1}^{N}\phi(x_i)]$

  $= (K - 1_N K - K 1_N + 1_N K 1_N)_{ij}.$

Kernel PCA Algorithm

Remember that eigenvector is the linear combination of feature vectors:

$$\boldsymbol{v} = \sum_{i=1}^{n} \alpha_i \, \phi(\boldsymbol{x}_i)$$

Algorithm 2 Kernel PCA

procedure K-PCA(X)
- given input: $X_{n\times m} \leftarrow \left[\mathbf{x}_1; \mathbf{x}_2; \cdots; \mathbf{x}_n\right]^T$
- calculate kernel matrix $K_{n\times n}$: $k_{ij} \leftarrow k(\mathbf{x}_i, \mathbf{x}_j)$
- centralize K : $K' \leftarrow K - \mathbb{I}_n K/n - K\mathbb{I}_n/n + \mathbb{I}_n K \mathbb{I}_n/n^2$
- calculate eigenvector $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \cdots, \boldsymbol{\alpha}_d$ according to: $n\lambda\boldsymbol{\alpha} = K'\boldsymbol{\alpha}$
- normalize eigenvector according to: $n\lambda_i \boldsymbol{\alpha}_i^T \boldsymbol{\alpha}_i = 1$
- project the test data $\mathbf{x}$: $p_i(\mathbf{x}) \leftarrow \sum_{j=1}^{n} \alpha_{ij} k(\mathbf{x}, \mathbf{x}_j)$

finish

Note: $\mathbb{I}_n$ stands for $n \times n$ matrix with all values equal to 1.

* Bishop, C. M. *Pattern Recognition and Machine Learning.*

Computational Complexity

- The kernel PCA doesn't necessarily reduce or increase the computational complexity. Let's consider three cases.
- Note that: $x_i \in \mathbb{R}^m,\ \phi(x_i) \in \mathbb{R}^d,\ i = 1, 2, \ldots n$

* Bishop, C. M. *Pattern Recognition and Machine Learning.*

Computational Complexity

- The kernel PCA doesn't necessarily reduce or increase the computational complexity. Let's consider three cases.
- Note that: $x_i \in \mathbb{R}^m,\ \phi(x_i) \in \mathbb{R}^d,\ i = 1, 2, \ldots n$
- PCA: 1. Find covariance matrix: $O(nm^2)$; 2. SVD: $O(m^3)$

* Bishop, C. M. *Pattern Recognition and Machine Learning.*

Computational Complexity

- The kernel PCA doesn't necessarily reduce or increase the computational complexity. Let's consider three cases.
- Note that: $x_i \in \mathbb{R}^m,\ \phi(x_i) \in \mathbb{R}^d,\ i = 1, 2, \ldots n$
- PCA: 1. Find covariance matrix: $O(nm^2)$; 2. SVD: $O(m^3)$
- Mapping – PCA: 1. Mapping: $O(nmd)$; 2. Find covariance matrix: $O(nd^2)$; 3. SVD: $O(d^3)$;

* Bishop, C. M. *Pattern Recognition and Machine Learning.*

Computational Complexity

PCA	Mapping - PCA	Kernel PCA
$O(nm^2 + m^3)$	$O(nmd + nd^2 + d^3)$	$O(n^2m + n^3)$
Linear	Non-linear	Non-linear

1. For data without linear dependency, PCA is not enough.
2. Since d > m, mapping – PCA is always more complex than PCA. For very large d it will be impractical to carry out.
3. Kernel PCA can be more expensive, or not. It depends on the number of instances and the dimension of feature space. However, it can handle non-linear data.

Commonly Used Kernels*

- Polynomial kernel

$$K(x, y) = (\gamma x^T y + c_0)^d$$

- Radial basis function kernel (Gaussian or RBF kernel)

$$K(x, y) = \exp(-\gamma \|x - y\|^2)$$

- Sigmoid kernel

$$K(x, y) = \tanh(\gamma x^T y + c_0)$$

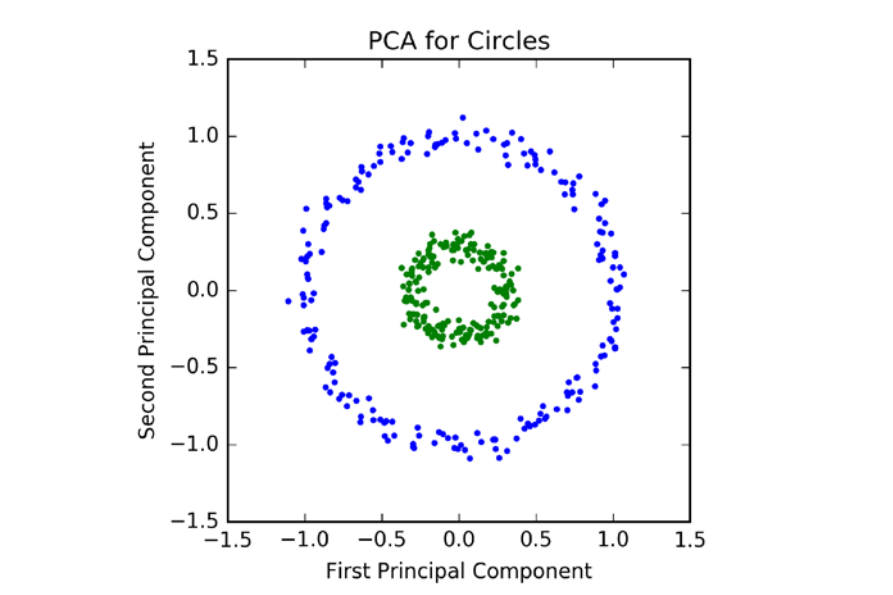
- Laplacian kernel

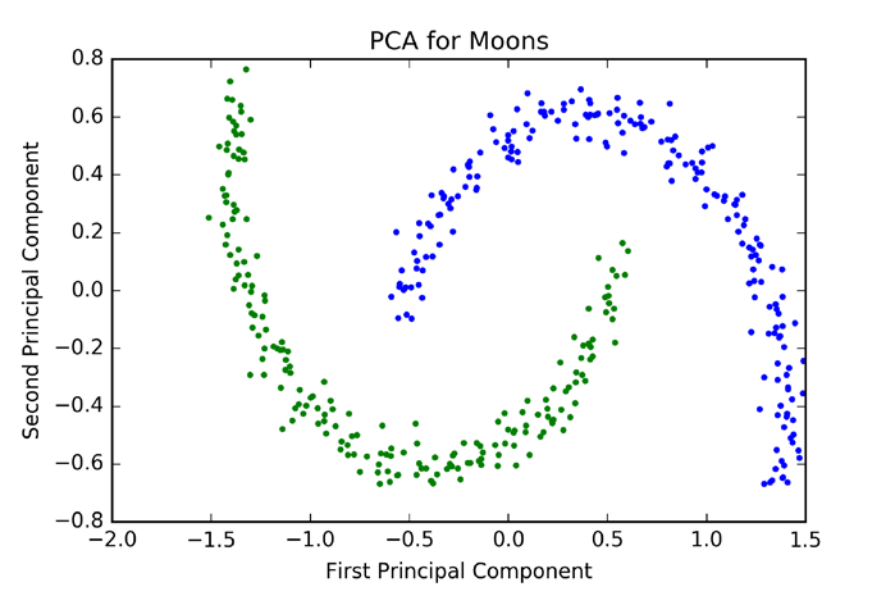
$$K(x, y) = \exp(-\gamma \|x - y\|_1)$$

- And so on.

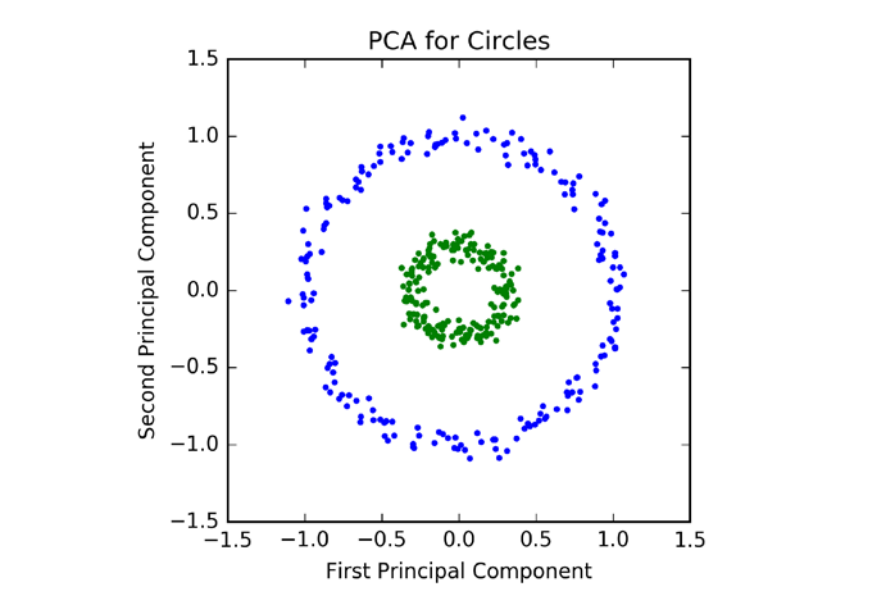
* http://scikit-learn.org/stable/modules/metrics.html#metrics

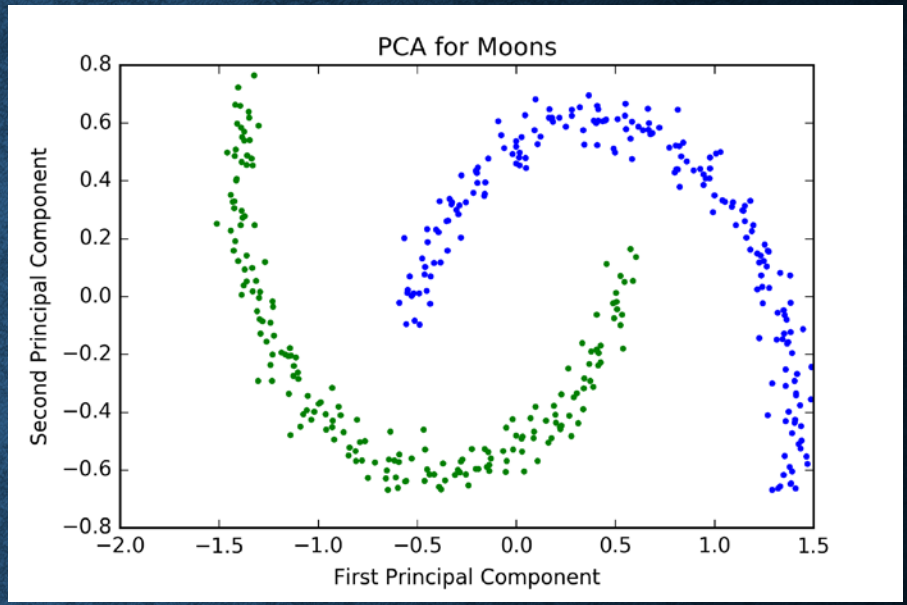
Toy Examples

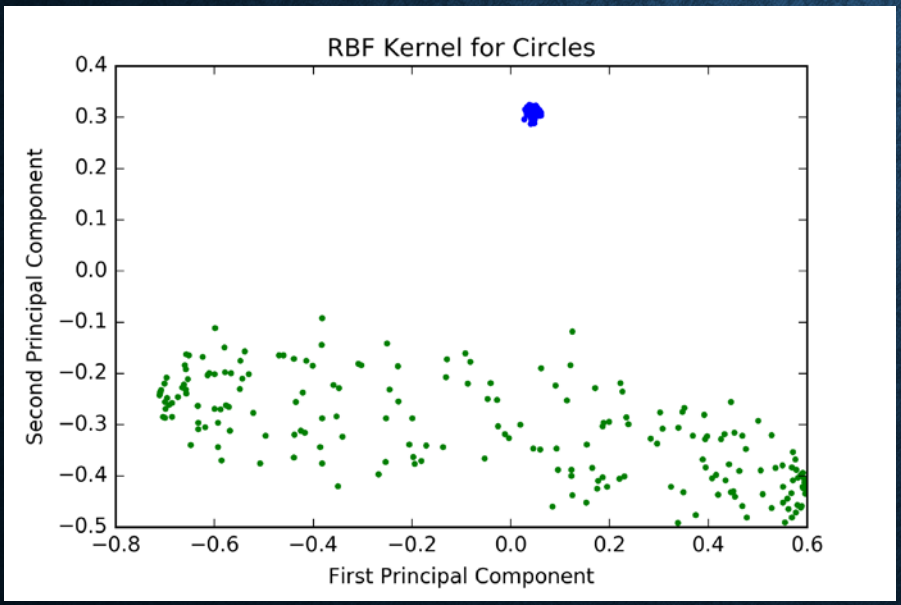


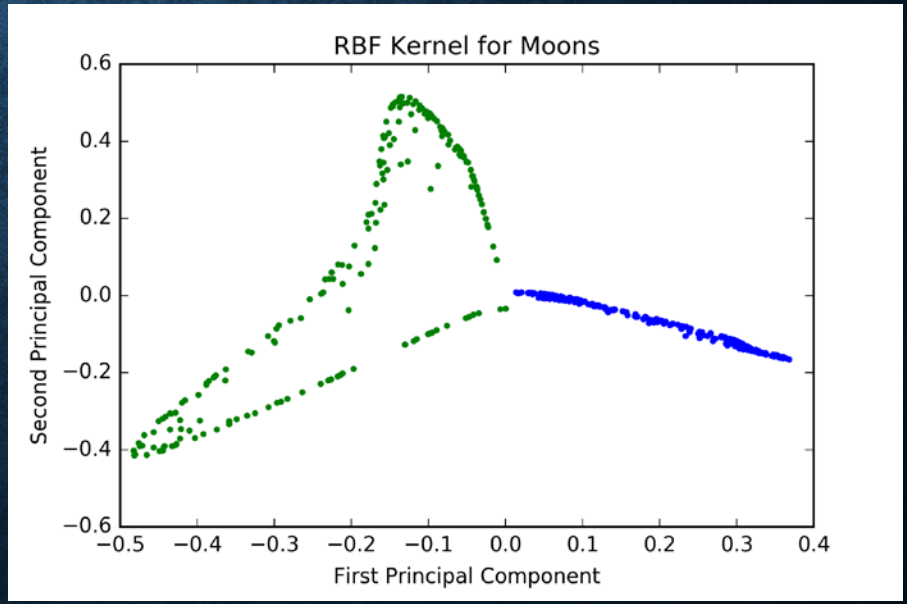


Toy Examples







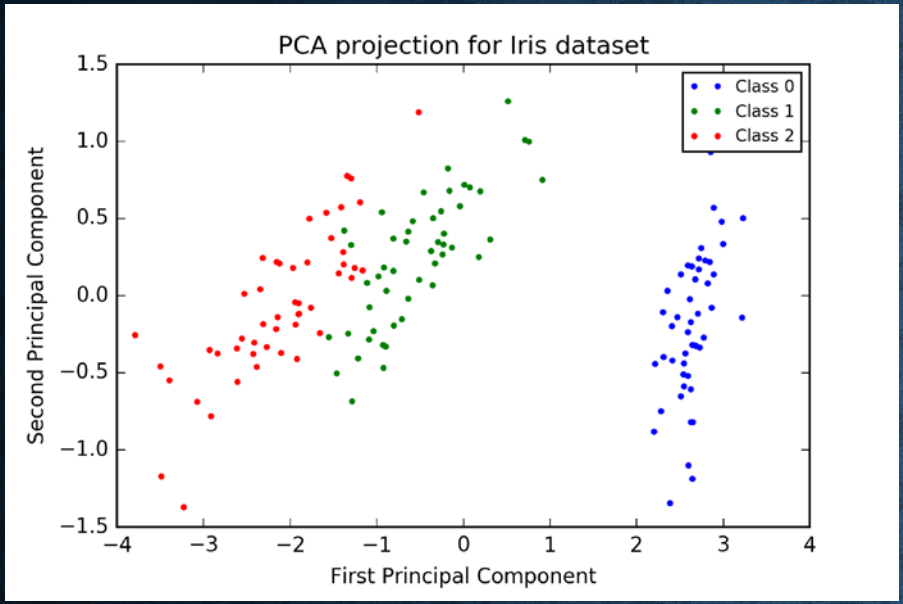


Iris Dataset – Introduction

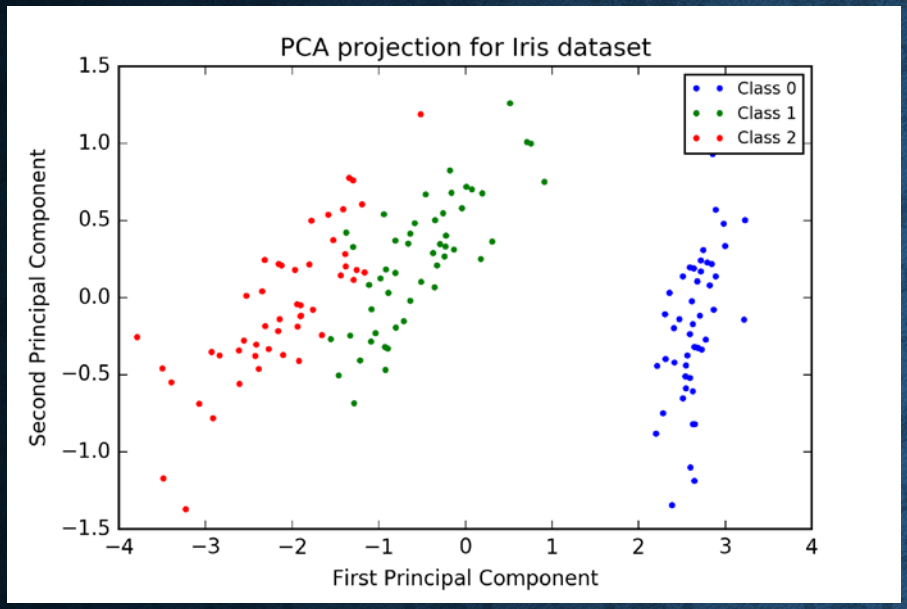
- Iris flower dataset is introduced by Ronald Fisher (1936)*
- It contains 3 different types of irises (Setosa, Versicolor, and Virginica)**.
- The dataset has 150 records and each has 4 features:
  - Sepal Length
  - Sepal Width
  - Petal Length
  - Petal Width.

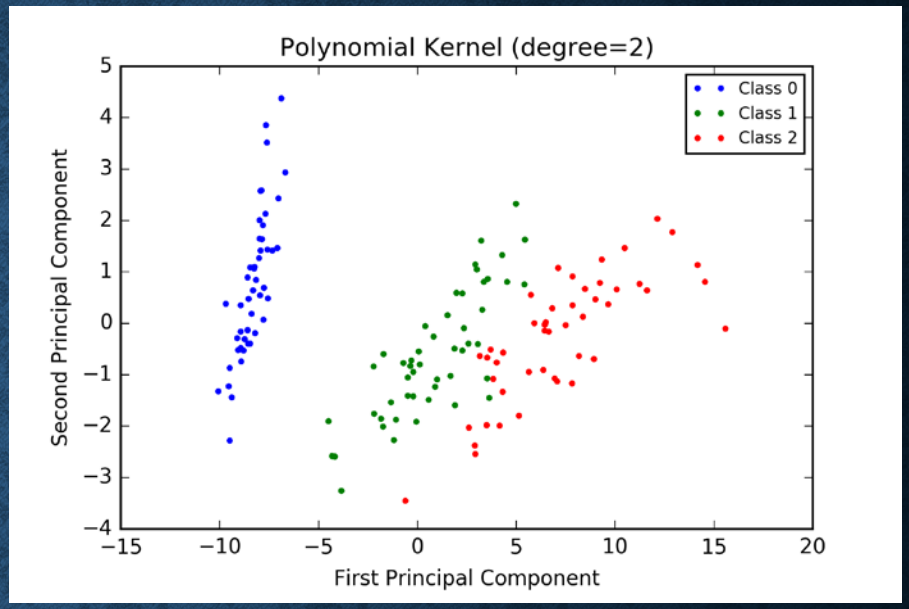
* https://en.wikipedia.org/wiki/Iris_flower_data_set
** http://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

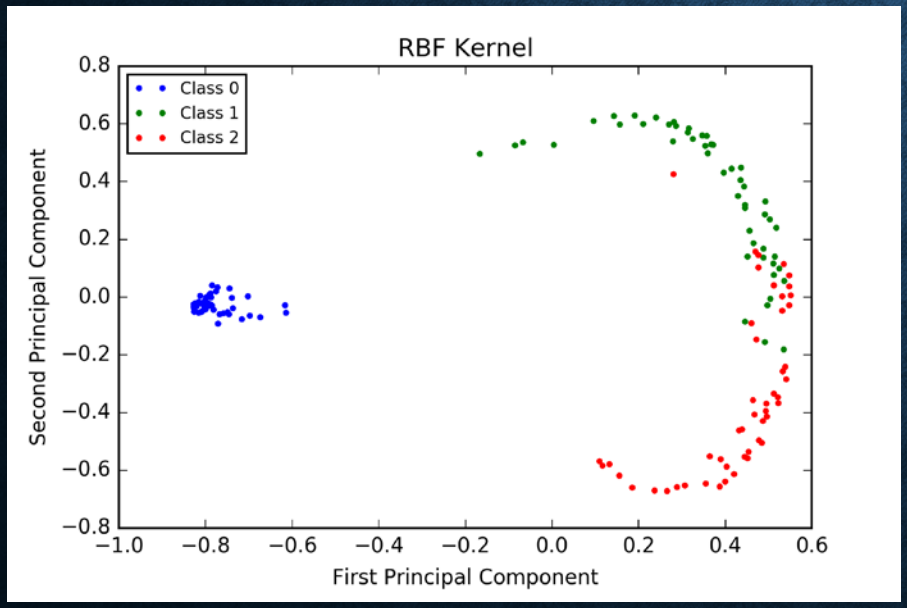
Iris Dataset – PCA vs. K-PCA

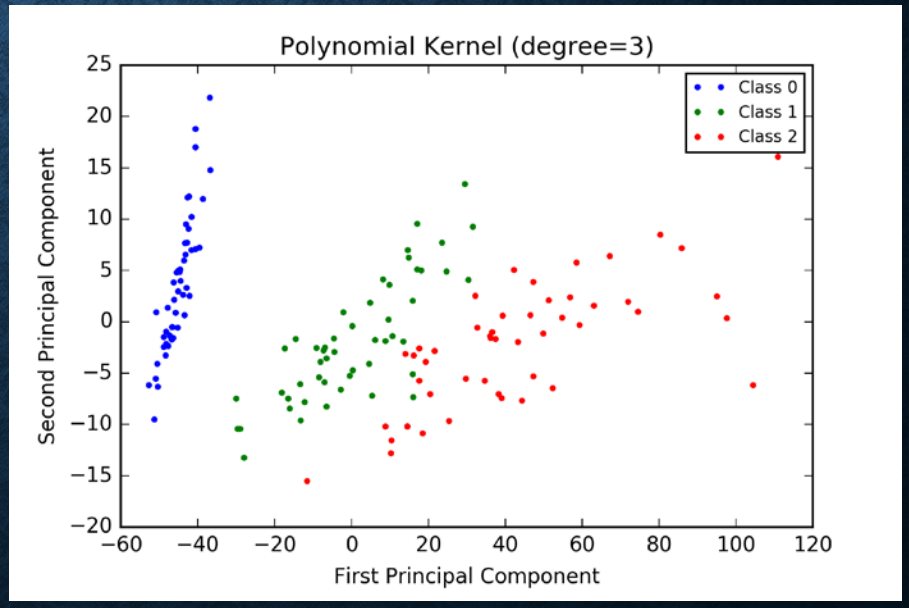


Iris Dataset – PCA vs. K-PCA









Digits Dataset – Introduction

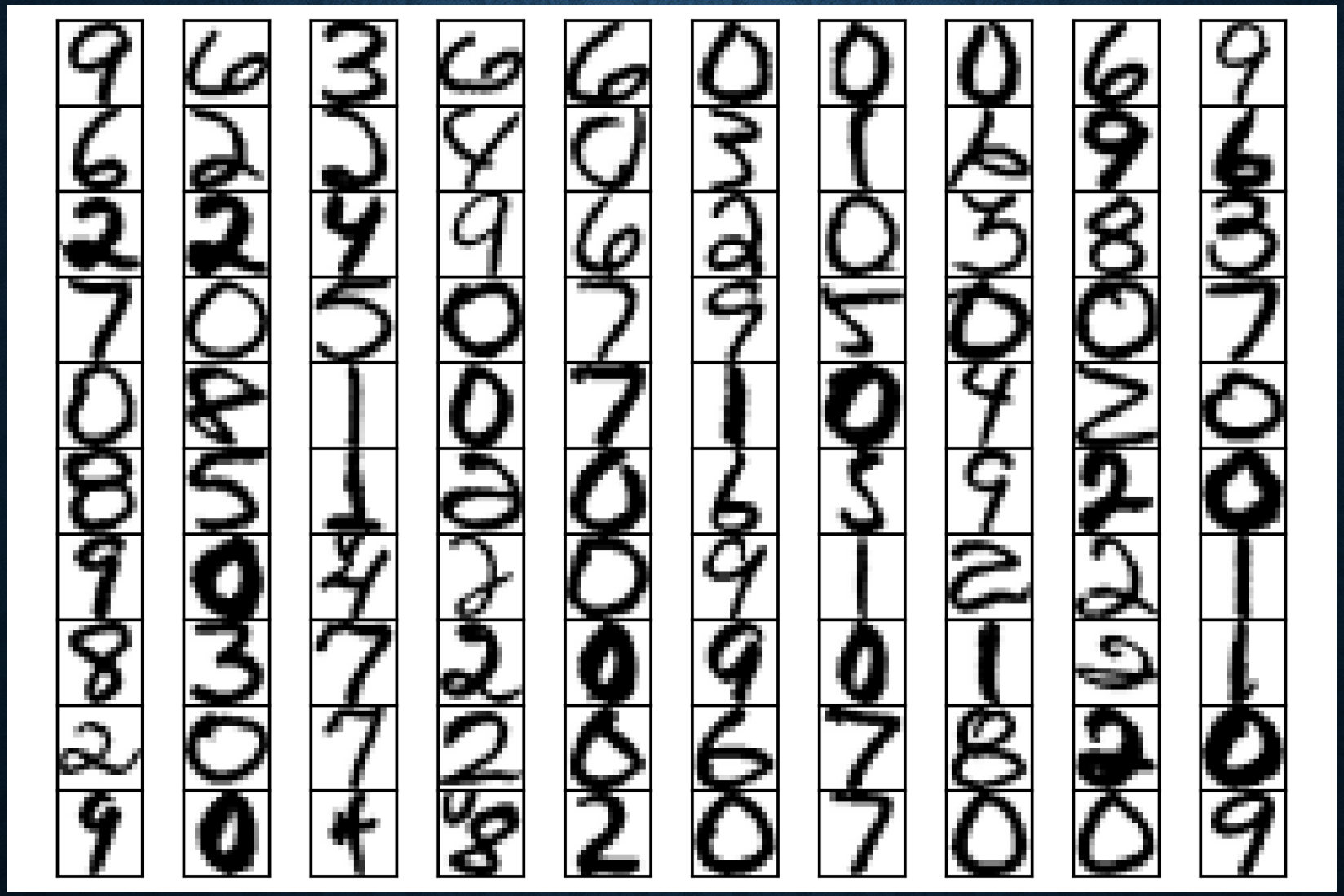
- Normalized handwritten digits, automatically scanned from envelops by the U.S. Postal Service*.
- All the images have been processed into 16×16 grayscale images (256 features).
- The dataset has 10 classes (0-9). There are 7291 training observations and 2007 test observations**.

Class	0	1	2	3	4	5	6	7	8	9	Total
Train	1194	1005	731	658	652	556	664	645	542	644	7291
Test	359	264	198	166	200	160	170	147	166	177	2007

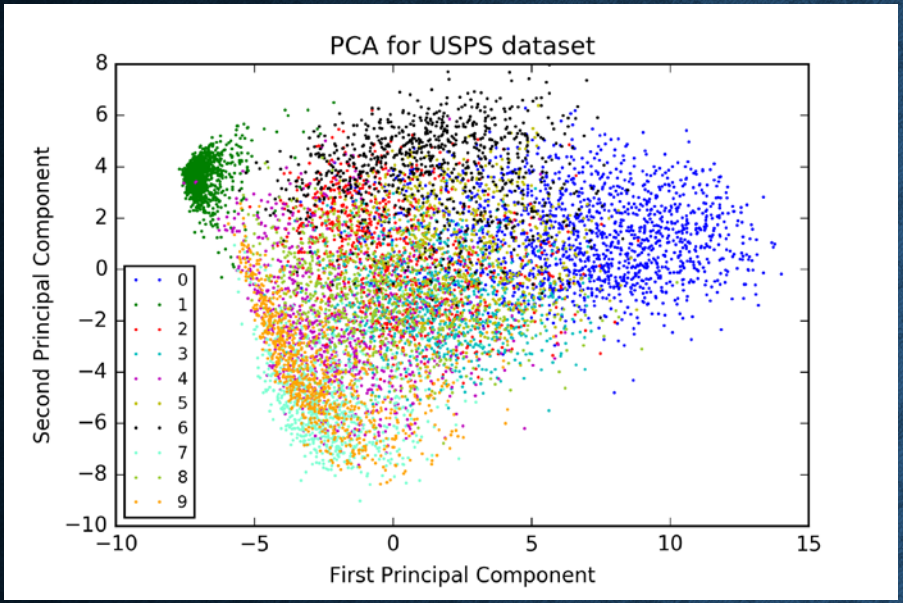
* http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/zip.info.txt

** https://www.otexts.org/1577

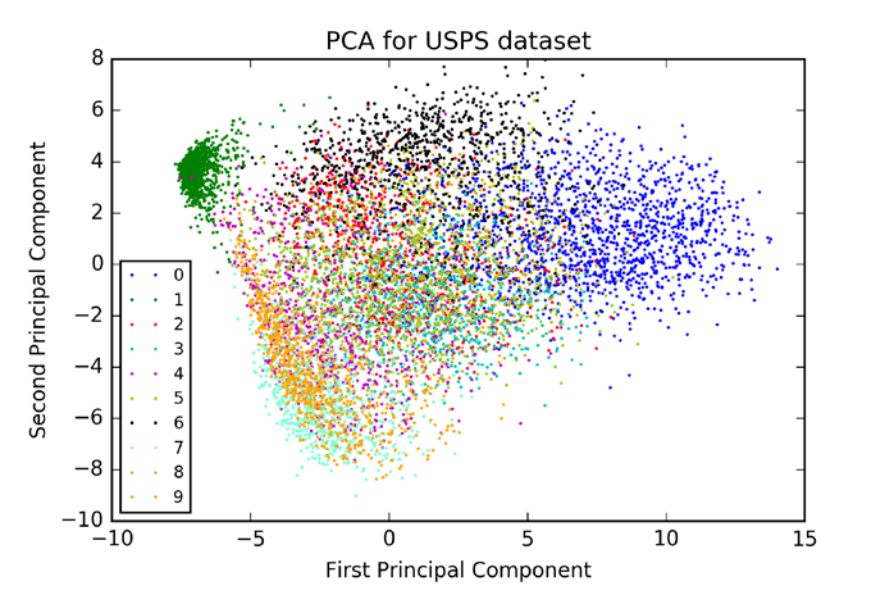
Digits Dataset – Introduction

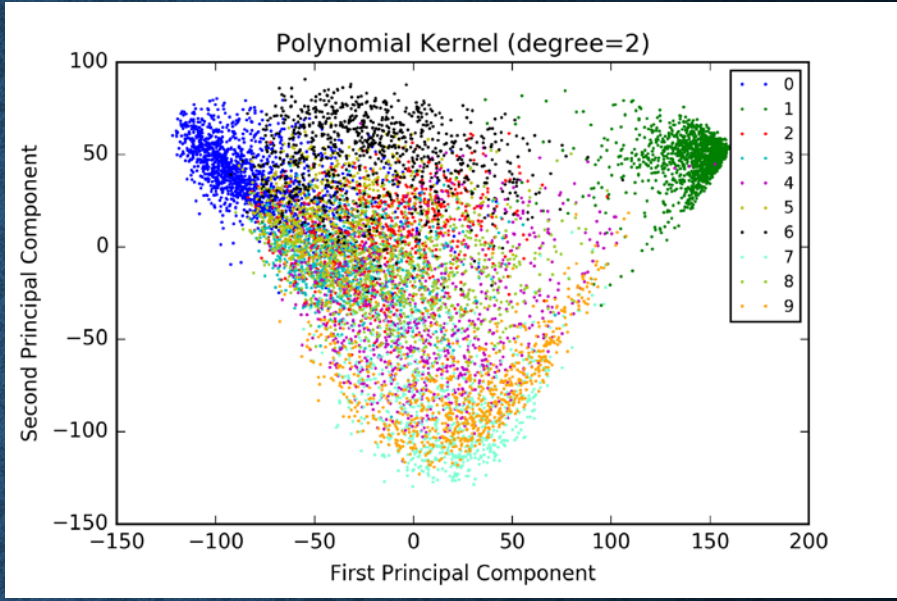


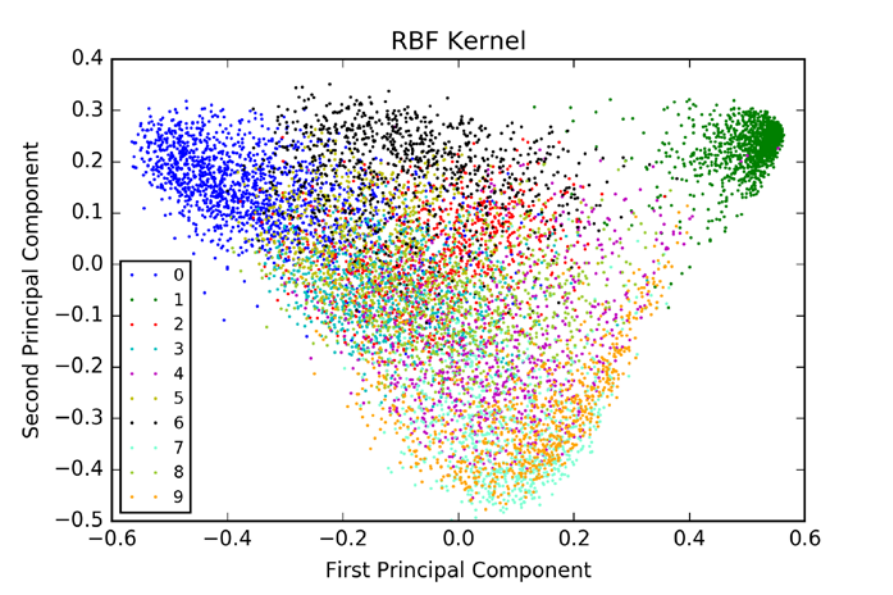
Digits Dataset – PCA vs. K-PCA

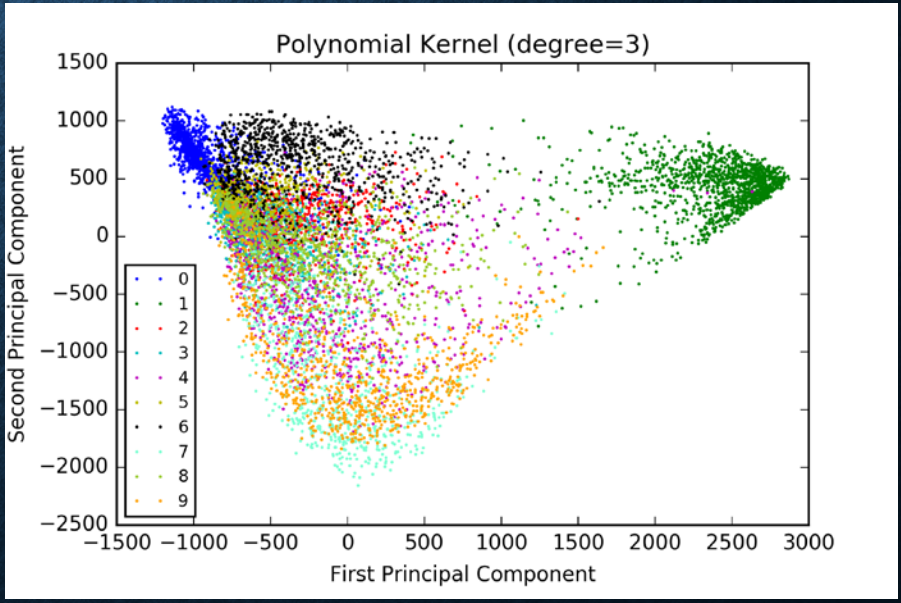


Digits Dataset – PCA vs. K-PCA



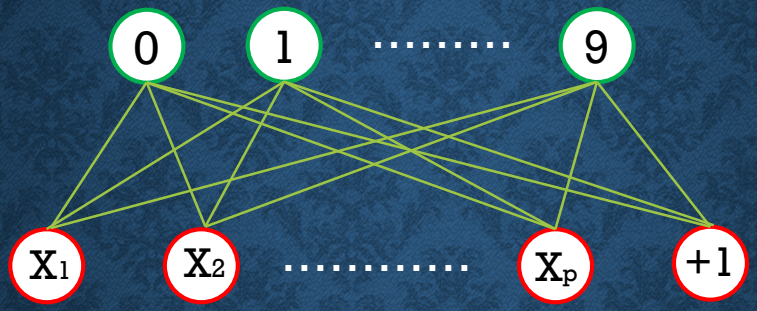






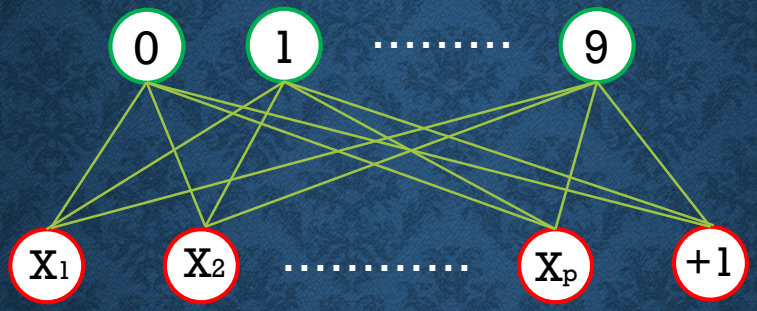
Digits Dataset – Classification

- Train a 10-way multi-class neural network for classification



Digits Dataset – Classification

- Train a 10-way multi-class neural network for classification



- Training accuracy and testing accuracy

Inputs	Raw image	PCA	Polynomial Kernel (degree=2)	Polynomial Kernel (degree=3)	RBF Kernel
Feature Number	256	128	512	1024	1024
Training Accuracy	95.52%	94.94%	99.99%	98.51%	93.58%
Test Accuracy	90.98%	90.23%	94.42%	93.92%	88.14%

Summary

- Computation complexity comparison
  - PCA: ~ $O(nm^2 + m^3)$
  - Kernel PCA: ~ $O(n^2 m + n^3)$
- Kernel PCA takes advantage of kernels to avoid expensive computations
- Kernels PCA can work better than PCA for some problems
- Need to note:
  - Kernel matrix K has dimension of $n \times n$. It will be expensive to find eigenvectors when n is large (~ $O(n^3)$)
  - Using kernels, the parameters d, γ, c_0 need to be determined by users

Extension

- Dimensionality reduction is more generally used
- There are methods for dimensionality reduction
    - Independent Component Analysis (ICA)
    - Non-negative Matrix Factorization (NMF)
    - Isometric Feature Mapping (ISOMAP)
    - Locality Sensitive Hashing (LSH)
    - Latent Semantic Analysis (LSA)
    - Restricted Boltzmann Machine (RBM)
    - Auto-encoders
    - And so on

Questions?

- Language:
  - Python + TensorFlow
- All source codes are available in GitHub
  - https://github.com/JifuZhao/UIUC_Courses/tree/master/UIUC_IE529/project/code
- Thank you.

References

- https://en.wikipedia.org/wiki/Principal_component_analysis
- http://scikit-learn.org/stable/modules/metrics.html#metrics
- https://en.wikipedia.org/wiki/Iris_flower_data_set
- http://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html
- https://www.otexts.org/1577
- http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/zip.info.txt
- Schölkopf, B., Smola, A. & Müller, K.R., 1998. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*.
- Friedman, J., Hastie, T., & Tibshirani, R. *The elements of statistical learning.*
- Bishop, C. M. *Pattern Recognition and Machine Learning.*