Compressing Data via Dimensionality Reduction

Dr. Saerom Park
Statistical Learning and Computational Finance Lab.
Department of Industrial Engineering
psr6275@snu.ac.kr
http://slcf.snu.ac.kr

Table of Contents

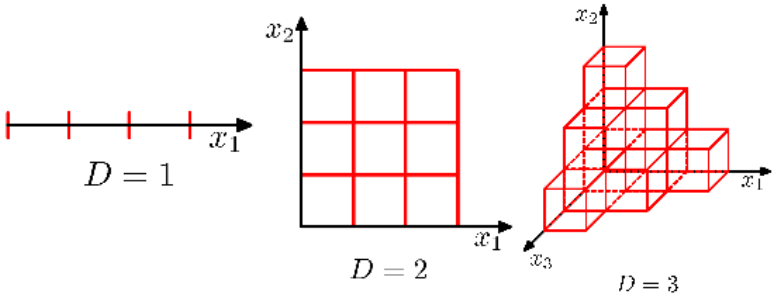
Reference

- **Reading:** [Raschka. (2017), chapter 5], [GÉRON. (2017), chapter 8].

Introduction

고차원 데이터

- 차원의 저주
  - 차원이 높아지면 데이터끼리의 거리가 멀어져 밀도가 낮아지는 (sparse) 문제 발생
    - 만약 unit square나 3D cube에서 두 점을 임의로 뽑는다면, 그 두 점 간의 평균 거리는 0.52와 0.66일 것이다. 하지만, 1,000,000 차원 hypercube에서의 거리는 408.25로 기하급수적으로 늘어난 것을 알 수 있다
  - 같은 맥락에서, training set의 차원이 높으면 overfitting의 문제가 발생
- 차원 감소 과정
  - 데이터 전처리
  - 모델 학습 (학습 데이터 이용)
  - 새로운 데이터를 위한 특성 변환
- 차원 감소 방법
  - 선형 방법:
    - Principal Component Analysis(PCA)
    - Linear Discriminant Analysis(LDA)
  - 비선형 방법:
    - Kernel Principal Component Analysis(KPCA)
    - Manifold learning

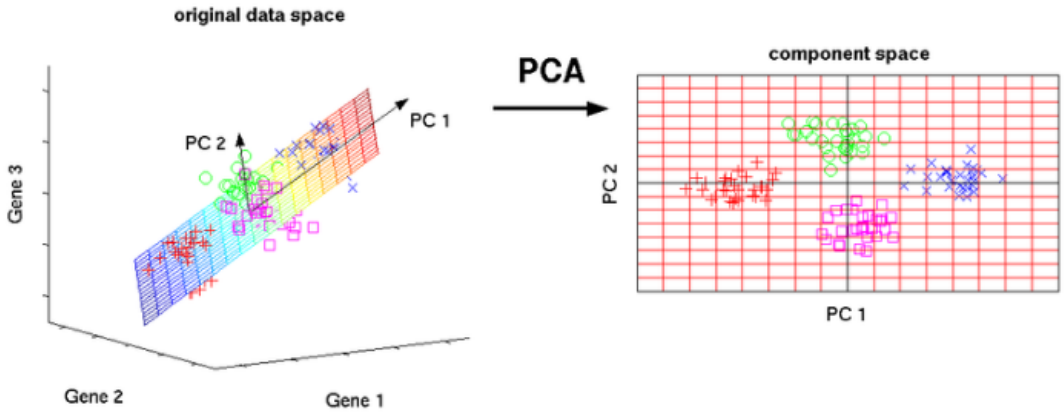


PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis(주성분 분석)

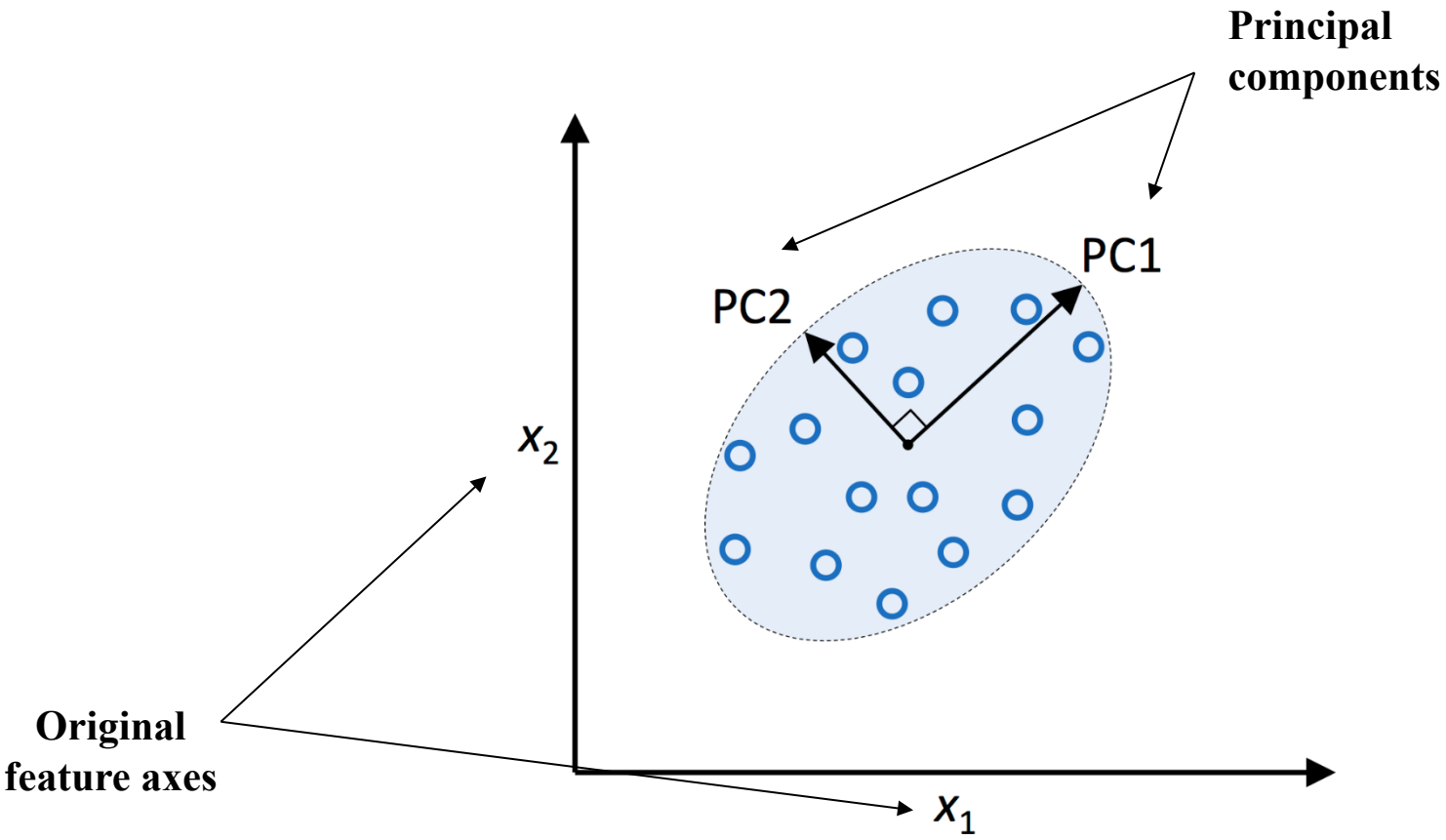
주성분 분석이란?

- PCA는 unsupervised linear transformation technique으로, 다양한 분야에서 feature extraction이나 dimensionality reduction을 위해 사용되고 있음
  - Ex) 주식 시장 트레이딩, 게놈 데이터 분석
- 비지도 학습 방법이며 선형 모델임
- PCA는 고차원 데이터의 분산을 최대화 하는 방향을 찾아내어, 이 방향으로 데이터를 projection 하여 차원을 줄이는 방법을 말한다.
  - 새로운 subspace의 두 수직 축(principal components)은 분산을 최대화하는 방향이 된다.



Principal Component Analysis

- 주성분 분석이란?



Principal Component Analysis

주성분 분석의 주요 과정

- PCA를 이용하여 차원 축소를 하기 위해서는, $d \times k$ 차원의 변환 행렬 $\boldsymbol{W}$를 만들어야 한다. 이 때 $\boldsymbol{W}$는 sample 벡터 $\mathbf{x}$를 새로운 k차원의 공간에 투영시켜주며, k는 original 차원인 d보다 작다.

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_d], \quad \boldsymbol{x} \in R^d$$

$$\downarrow \boldsymbol{x}\boldsymbol{W}, \quad \boldsymbol{W} \in R^{dxk}$$

$$\boldsymbol{z} = [z_1, z_2, \ldots, z_k], \quad z \in R^k$$

- PCA의 방향들은 data scaling에 매우 민감
  - feature들을 정규화가 중요함

Principal Component Analysis

주성분 분석의 주요 과정

- PCA 과정 요약:

① d 차원의 데이터셋을 정규화하여 X를 만든다.

② 공분산 행렬을 계산한다.

③ 공분산행렬의 eigenvalue와 eigenvector들을 구한다.

④ eigenvalue들을 내림차순으로 정렬하여, 이에 맞는 eigenvector들을 정렬한다.

⑤ k개의 가장 큰 eigenvalue들을 선택하고, 이에 맞는 eigenvector들을 선택한다. 이 때 k가 새로운 feature subspace의 차원이 된다. $(k \leq d)$.

⑥ 위에서 선택한 eigenvector들로 projection matrix **W**를 만든다

⑦ projection matrix **W**를 이용하여 데이터셋 X를 새로운 k-차원 feature subspace에 projection 한다.

Principal Component Analysis

데이터 전처리

- Wine Data set을 import 한 뒤, training set과 test set으로 분리한다(70:30).
- 정규화를 실시한다.

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Principal Component Analysis

공분산 행렬

- Symmetric한 $d\ x\ d$ 차원의 공분산행렬은, d차원 데이터 셋에 대하여 feature 간의 공분산 쌍을 모두 저장한다.
- 두 feature x_j and x_k 에 대해 공분산은 다음과 같이 계산:

$$\sigma_{jk} = \frac{1}{n}\sum_{i=1}^{n}(x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k)$$

(μ_j, μ_k are the sample means of features j and k)

- 3가지 feature를 가진 데이터의 공분산행렬은 다음과 같이 표현:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_2^2 & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_3^2 \end{bmatrix}$$

  - 이 공분산 행렬의 eigenvector들은 주성분(principal components)들을 나타내며, eigenvalue들은 주성분들의 크기를 나타내준다.

Principal Component Analysis

- 주성분 추출하기(step by step)
  - numpy.cov(m,..,bias, ….)
  - https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.cov.html

Parameters	
m	Input 데이터 셋의 array
bias	False일 경우 normalization을 n-1로 진행, 하지만 true일 경우 normalization을 n으로 진행

Returns	
out	ndarray 형태의 공분산 행렬

Principal Component Analysis

- 고유값과 고유벡터 구하기
  - 공분산 행렬의 고유값과 고유벡터들을 구함
    - 선형대수학에서, eigenvector v는 다음의 조건을 만족한다:

$$\Sigma v = \lambda v$$

λ: eigenvalue

Eigenvalues

[4.84274532 2.41602459 1.54845825 0.96120438 0.84166161 0.6620634 0.51828472 0.34650377 0.3131368 0.10754642 0.21357215 0.15362835 0.1808613]

Principal Component Analysis

- 고유값과 고유 벡터를 구하는 함수
  - numpy.linalg.eig(a)
  - https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linalg.eig.html

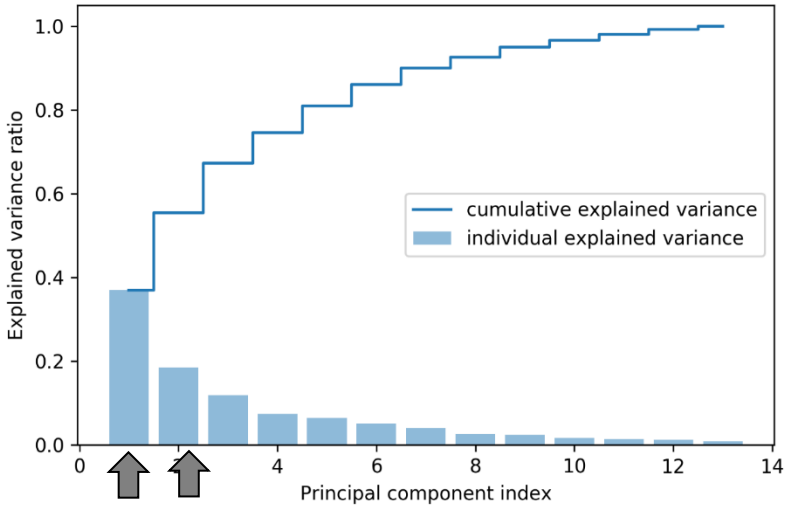
Parameters	
a	Eigenvalue와 eigenvector를 구하고자 하는 행렬

Returns	
w	Eigenvalue들을 array형태로 return. 이 때, multiplicity에 따라 중복되어 등장한다.
v	Normalized unit length eigenvector들을 반환한다.

Principal Component Analysis

Total and explained variance

- 차원 축소를 하기위하여, eigenvector를 모두 선택하는 것이 아니라 정보(여기서는 variance)를 많이 담고 있는 eigenvector의 부분 집합을 선택
  - k개의 top eigenvector들에 대해서만 관심이 있음
- The variance explained ratios of the eigenvalues: $\frac{\lambda_j}{\sum_{j=1}^{d} \lambda_j}$
  - numpy.cumsum을 이용하면 explained variances를 cumulative sum으로 구할 수 있다.



The first two principal components(almost 60% of the variance)

Principal Component Analysis

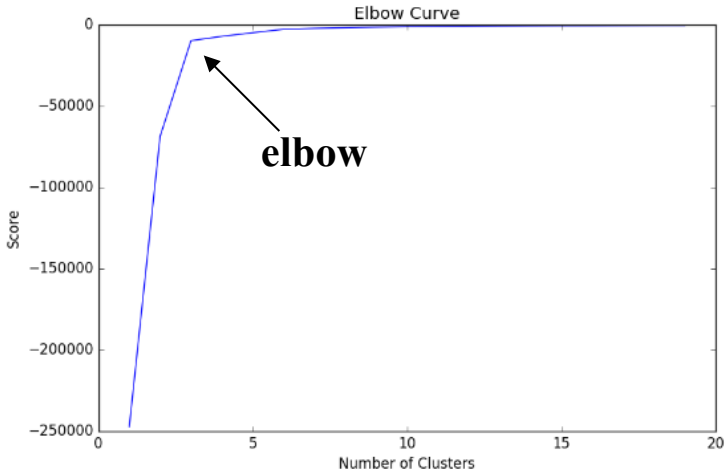
적절한 차원 정하기

- 유용한 옵션:

원하는 차원의 숫자 뿐만 아니라 **explained variance ratio**의 합을 설정해주어도 된다. **(0~1)**

```
pca = PCA(n_components = 0.95)
X_reduced = pca.fit-transform(X)
```

- Explained variance의 누적합 곡선에서, 증감폭이 급격히 감소하는 구간을 elbow라고 한다.
  - 아래의 그림은 clustering에서의 elbow



Principal Component Analysis

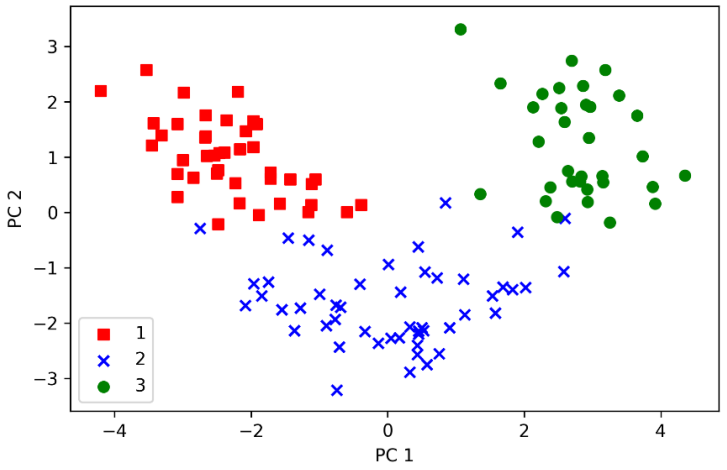
Feature transformation

- Note: Numpy 버전에 따라 W의 부호가 바뀔 수 있음
  - eigenvalue λ에 대하여 v와 $-v$가 모두 eigenvector이므로 상관없음

$$\Sigma \cdot (-v) = -\Sigma\, v = -\lambda\, v = \lambda \cdot (-\, v)$$

- projection matrix를 사용하면, 123 x 13의 training dataset을 123 x 2의 데이터 셋으로 변환 시킬 수 있음
  - 축은 두 principal components
  - 가장 큰 두 개의 eigenvalue 값에 해당하는 eigenvector들을 추출
    - 이 둘을 합치면, 13 x 2 차원의 projection matrix $\boldsymbol{W}$

$$X' = XW$$



Principal Component Analysis

- Principal component analysis in scikit-learn
  - PCA class implemented in scikit-learn
  - Sklearn.decomposition.PCA(n_components,…)
  - http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

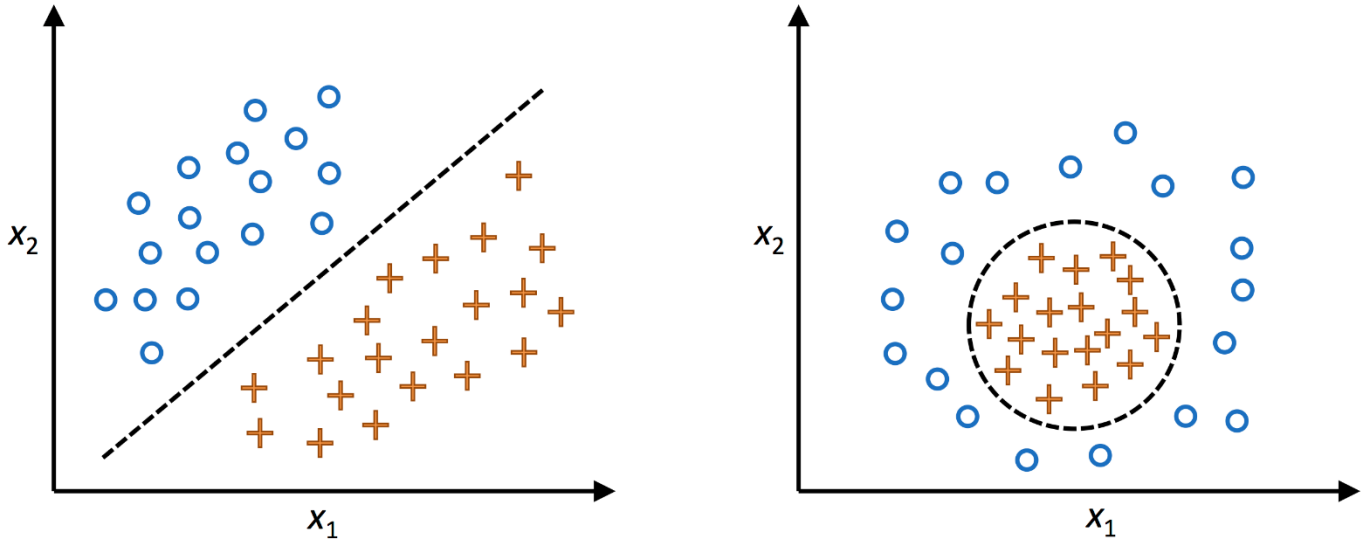
Parameters	
n_components	Keep 하고 싶은 component의 개수(지정해주지 않는 다면 모두 다 keep)

Attributes	
components	ndarray 형태의 공분산 행렬
explained_variance_	Explained variance(eigenvalue들)
Explained_variance_ratio	Percentage of variance explained(eigenvalue를 eigenvalue의 총합으로 나눈 것)

KERNEL PCA

Kernel Principal Component Analysis

- Kernel Component Principal Analysis
  - Real-world에서 등장하는 여러 비선형 문제들에 대하여, linear transformation technique인 PCA나 LDA는 좋은 선택이 아닐 수 있다.
  - Kernel PCA를 사용하면, 선형 분리가 가능하지 않은 데이터를 선형 분리가 가능하도록 transform시킬 수 있다.



Kernel Principal Component Analysis

- Kernel functions and the kernel trick
  - 샘플 $x \in R^d$ 를 더 고차원의 k-dimensional subspace로 transform 시키기 위해서는, nonlinear mapping function ∅을 정의해야 한다:

$$\emptyset: R^d \rightarrow R^k \ (\mathrm{k} \gg d)$$

  - ∅는 original feature들에 대한 비선형 조합으로, 이 함수를 통해 기존의 d-차원 데이터를 k차원(더욱 높은 차원)으로 mapping 시켜준다.
  - For example,

$$x = [x_1, x_2]^T$$

$$\downarrow \emptyset$$

$$z = [x_1^2, \sqrt{2x_1x_2}, x_2^2]^T$$

  - 그 뒤, 이 높은 차원의 데이터를 이용해 standard PCA를 실시하여 다시 lower-dimensional space로 project할 수 있다.

Kernel Principal Component Analysis

- Kernel functions and the kernel trick
  - 하지만, 이러한 ø를 구하고 계산하는 것이 쉽지 않기 때문에, kernel trick을 사용한다.
  - Kernel trick을 사용하면, 원래의 feature space에서 더욱 높은 차원에 존재하는 두 벡터의 similarity를 계산할 수 있다.
  - 공분산의 정의를 다시 한 번 기억해보자:

$$\sigma_{jk} = \frac{1}{n}\sum_{i=1}^{n}(x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k)$$

  - 데이터에 대해 정규화를 실시했다면, μ_j, μ_k = 0이다.

$$\sigma_{jk} = \frac{1}{n}\sum_{i=1}^{n}(x_j^{(i)})(x_k^{(i)})$$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} x^{(i)}x^{(i)^T}$$

Kernel Principal Component Analysis

- Kernel functions and the kernel trick
  - ∅를 사용하게 된다면, 우리가 원래 구해야 하는 공분산은 다음과 같다.

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} \emptyset(x^{(i)})\emptyset(x^{(i)})^T$$

  - 그리고 eigenvector를 구하기 위해, 다음의 등식을 풀어야 한다.

$$\Sigma v = \lambda v$$

$$\frac{1}{n}\sum_{i=1}^{n} \emptyset(x^{(i)})\emptyset(x^{(i)})^T v = \lambda v$$

$$v = \frac{1}{n\lambda}\sum_{i=1}^{n} \emptyset(x^{(i)})\emptyset(x^{(i)})^T v = \frac{1}{n}\sum_{i=1}^{n} a^{(i)}\emptyset(x^{(i)})$$

Kernel Principal Component Analysis

- Kernel functions and the kernel trick
  - Kernel matrix 유도:

$\emptyset(X)\ is\ an\ n\ x\ k\ dimensional\ matrix$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} \emptyset(x^{(i)})\emptyset(x^{(i)})^T = \frac{1}{n}\emptyset(X)^T\emptyset(X)$$

  - 이에 따라, eigenvector를 다음과 같이 표현할 수 있다.

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

  - $\Sigma v = \lambda v$이므로, 다음이 성립한다.

$$\frac{1}{n}\emptyset(X)^T\emptyset(X)\emptyset(X)^T\boldsymbol{a} = \lambda\emptyset(X)^T\boldsymbol{a}$$

Kernel Principal Component Analysis

- Kernel functions and the kernel trick
  - 양변에 $\emptyset(X)$를 곱하면,

$$\frac{1}{n}\emptyset(X)\emptyset(X)^T\emptyset(X)\emptyset(X)^T\boldsymbol{a} = \lambda\emptyset(X)\emptyset(X)^T\boldsymbol{a}$$

$$\frac{1}{n}\emptyset(X)\emptyset(X)^T\boldsymbol{a} = \lambda\,\boldsymbol{a}$$

$$\frac{1}{n}\boldsymbol{K}\boldsymbol{a} = \lambda\,\boldsymbol{a}$$

  - $\boldsymbol{K}$ is the similarity (kernel) matrix:

$$\boldsymbol{K} = \emptyset(\boldsymbol{X})\emptyset(\boldsymbol{X})^{\boldsymbol{T}}$$

Kernel Principal Component Analysis

- Kernel functions and the kernel trick
  - 커널 함수를 이용하면, sample x에 대한 ∅의 dot product를 구하지 않아도 되며, 직접 eigenvector를 구하지 않아도 된다.

$$k\left(x^{(i)}.x^{(j)}\right) = \emptyset(x^{(i)})^T\emptyset(x^{(j)})$$

  - 가장 널리 사용되는 커널은 **Radial Basis Function(RBF)** or Gaussian kernel 이다:

$$k\left(x^{(i)}, x^{(j)}\right) = \exp\left(-\frac{\|x^{(i)}-x^{(j)}\|^2}{2\sigma^2}\right)$$

$$\gamma = \frac{1}{2\sigma^2},\ then\ k\left(x^{(i)}, x^{(j)}\right) = \exp\left(-\gamma\|x^{(i)} - x^{(j)}\|^2\right)$$

Kernel Principal Component Analysis

Kernel functions and the kernel trick

- Kernel PCA 요약:

① Kernel matrix 를 $\boldsymbol{K}$계산한다.

$$k\left(x^{(i)}, x^{(j)}\right) = \exp\left(-\gamma \left\| x^{(i)} - x^{(j)} \right\|^2\right)$$

$$\boldsymbol{K} = \begin{bmatrix} k\left(x^{(1)}, x^{(1)}\right) & k\left(x^{(1)}, x^{(2)}\right) & \cdots & k\left(x^{(1)}, x^{(n)}\right) \\ k\left(x^{(2)}, x^{(1)}\right) & k\left(x^{(2)}, x^{(2)}\right) & \cdots & k\left(x^{(2)}, x^{(n)}\right) \\ \vdots & \vdots & \ddots & \vdots \\ k\left(x^{(n)}, x^{(1)}\right) & k\left(x^{(n)}, x^{(2)}\right) & \cdots & k\left(x^{(n)}, x^{(n)}\right) \end{bmatrix}$$

② 아래의 등식을 활용하여 $\boldsymbol{K}$를 centering 한다.

$$K' = K - 1_n K - K 1_n + 1_n K 1_n$$

1_n is an n x n-dimensional matrix where all values are equal to 1/n

③ 상위 k개의 eigenvector들을 구한다. 이 때, PCA와는 다르게 이 eigenvector 들은 principal component axes 들이 아니라 sample 그 자체가 project된 결과이다.

Kernel Principal Component Analysis

- Feature transformation
  - 만약 기존의 데이터가 아닌 새로운 데이터 샘플 x'를 projection하고 싶다면, $\emptyset(x')^T v$를 계산하여야 한다.

$$\emptyset(x')^T v = \sum_i a^{(i)} \emptyset(x')^T \emptyset(x^{(i)})$$

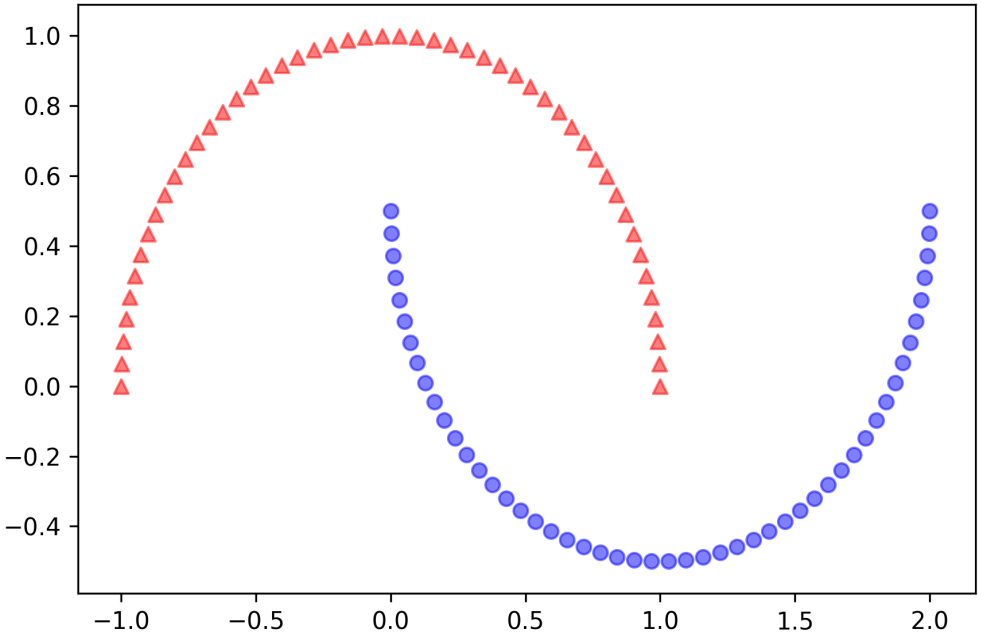
$$= \sum_i a^{(i)} k(x', x^{(i)})$$

  - 이 때, kernel matrix K의 eigenvalue λ와 이에 맞는 eigenvector a는 다음의 조건을 만족한다.

$$Ka = \lambda a$$

Kernel Principal Component Analysis

- Implementing a kernel principal component analysis in Python
  - Example 1 – Separating half-moon shapes

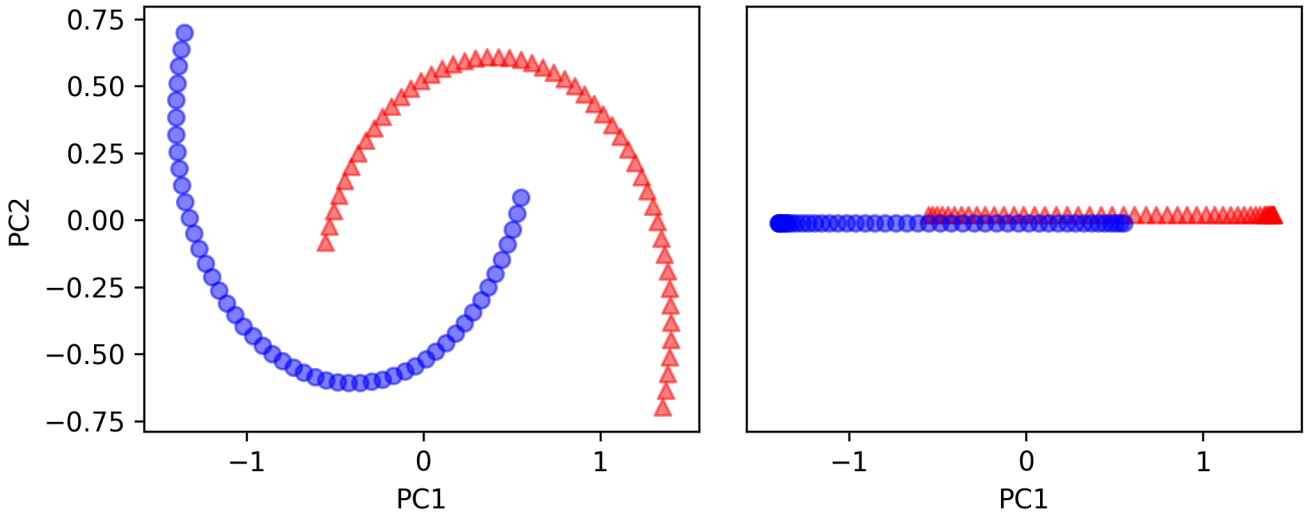


  - 이와 같은 data를 어떻게 선형 분리 가능하도록 만들 수 있을까?

Kernel Principal Component Analysis

- Implementing a kernel principal component analysis in Python
  - Example 1 – Separating half-moon shapes

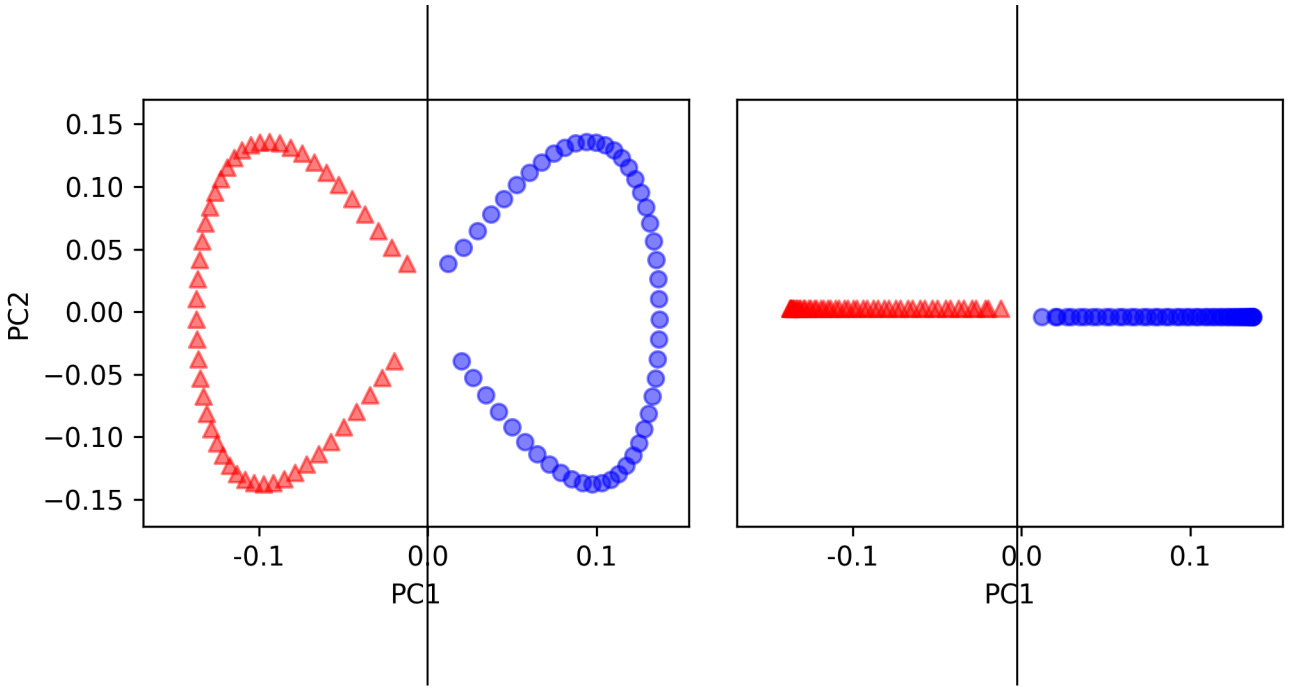
Standard PCA:



Kernel Principal Component Analysis

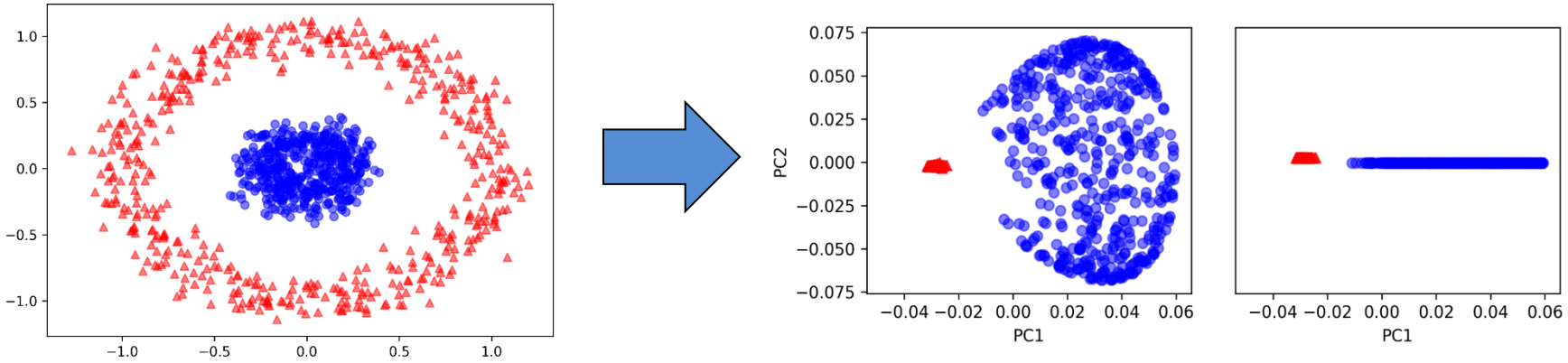
- Implementing a kernel principal component analysis in Python
  - Example 1 – Separating half-moon shapes

Kernel PCA:



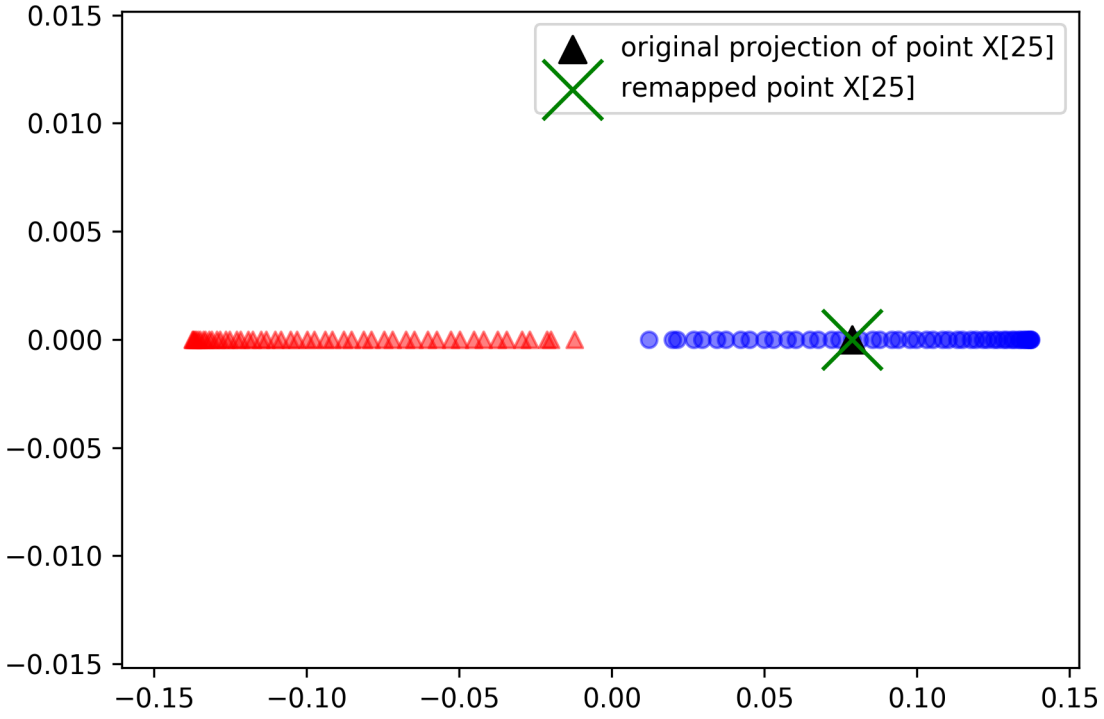
Kernel Principal Component Analysis

- Implementing a kernel principal component analysis in Python
  - Example 2 – Separating concentric circles



Kernel Principal Component Analysis

- Projecting new data points



Kernel Principal Component Analysis

- Kernel principal component analysis in scikit-learn
  - sklearn.decomposition.KernelPCA(n_components, kernel, ...)
  - http://scikit-learn.org/stable/auto_examples/decomposition/plot_kernel_pca.html

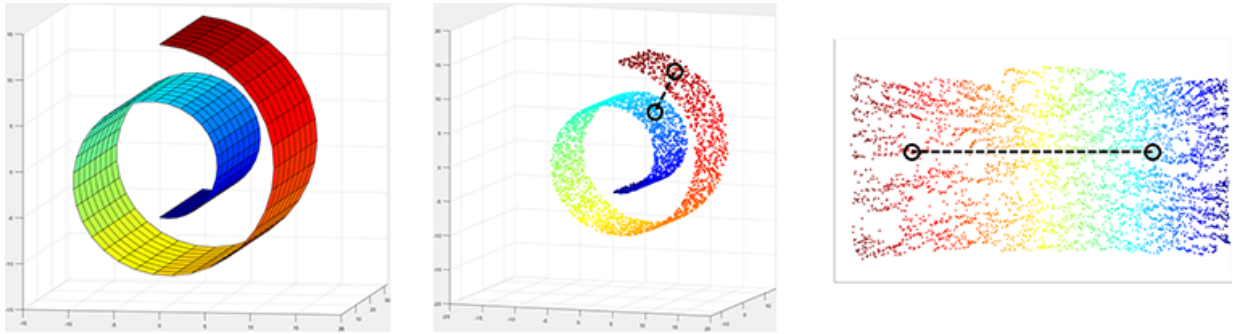
Parameters	
n_components	# of components
kernel	"linear", "poly", "rbf", "sigmoid", "cosine"..

Attributes	
lambdas_	Centered kernel matrix의 eigenvalue(내림차순)
alphas_	Centered kernel matrix의 eigenvector
X_transformed_fit	Fitted data의 KPCA projection

MANIFOLD LEARNING

Nonlinear Dimension Reduction Methods

- Manifold Learning(참고)
  - Manifold Learning은 비선형 차원 감소 기법임
  - Manifold란, 두 점 사이의 거리 혹은 유사도가 근거리에서는 유클리드 거리를 따르지만, 원거리에서는 그렇지 않은 공간을 말함
  - 이를 극복하기 위해 새로운 평면에 점을 찍고, 유클리드 거리와 실제 차이가 같게 되도록 하는 새로운 평면을 찾아내는 작업을 하는 것이 manifold learning이다.
  - 이를 학습하기 위한 알고리즘으로는 Isomap, LLE, T-SNE 등이 있다.
  - scikit-learn에서도 manifold learning에 관한 여러 클래스들을 제공하고 있다.
    - Isomap, LLE, MDS, SpectralEmbedding, t-SNE



(a) (b) (c)

Manifold Learning

- Isomap
  - sklearn.manifold.Isomap(n_neighbors, n_components, eigen_solver,..)
  - http://scikit-learn.org/stable/modules/generated/sklearn.manifold.Isomap.html

Parameters	
n_neighbors	각 point에 대해 고려할 neighbor의 수
n_components	manifold의 number of coordinates
Eigen_solver	Auto, arpack, dense의 옵션 존재. 각 옵션에 대한 자세한 설명은 사이트 참조

Methods	
fit(X[, y])	Data X에 대한 embedding vector들을 계산
fit_transform(X[, y])	Data X로 model fitting 뒤 X를 transform 시킴

Manifold Learning

- Locally linear embedding
  - sklearn.manifold.LocallyLinearEmbedding(n_neighbors, n_components, eigen_solver,..)
  - http://scikit-learn.org/stable/modules/generated/sklearn.manifold.LocallyLinearEmbedding

Parameters	
n_neighbors	각 point에 대해 고려할 neighbor의 수
n_components	manifold의 number of coordinates
eigen_solver	Auto, arpack, dense의 옵션 존재. 각 옵션에 대한 자세한 설명은 사이트 참조
method	Standard, hessian, modified, ltsa 옵션 존재. (옵션에 따라 다른 알고리즘으로 계산한다.)

Methods	
fit(X[, y])	Data X에 대한 embedding vector들을 계산
fit_transform(X[, y])	Data X로 model fitting 뒤 X를 transform 시킴

Manifold Learning

- Multi-dimensional scaling
  - sklearn.manifold.MDS(n_components, metric, n_init, max_iter, …)
  - http://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html

Parameters	
n_components	Number of dimensions in which to immerse the dissimilarities
metric	Boolean 옵션. True이면 metric MDS 실시, 아니면 nonmetric MDS
n_init	다른 초기화로 SMACOF 알고리즘이 실행될 횟수.
max_iter	Maximum number of iterations

Methods	
fit(X[, y])	Embedding space에서 점들의 position 계산
fit_transform(X[, y])	Data X로 model fitting 뒤 embedding coordinates return 해줌.

Manifold Learning

- Spectral embedding
  - sklearn.manifold.SpectralEmbedding(n_components, affinity, gamma, eigen_solver,..)
  - http://scikit-learn.org/stable/modules/generated/sklearn.manifold.SpectralEmbedding.html

Parameters	
n_components	Projected subspace의 차원
affinity	Nearest_neighbors, rbf, precomputed, callable(옵션에 대한 설명은 사이트 참조)
gamma	Kernel coefficient for rbf kernel
eigen_solver	Eigenvalue decomposition strategy (arpack, lobpcg, amg)

Methods	
fit(X[, y])	Data X로 model fitting
fit_transform(X[, y])	Data X로 model fitting 뒤 X를 transform 시킴

Manifold Learning

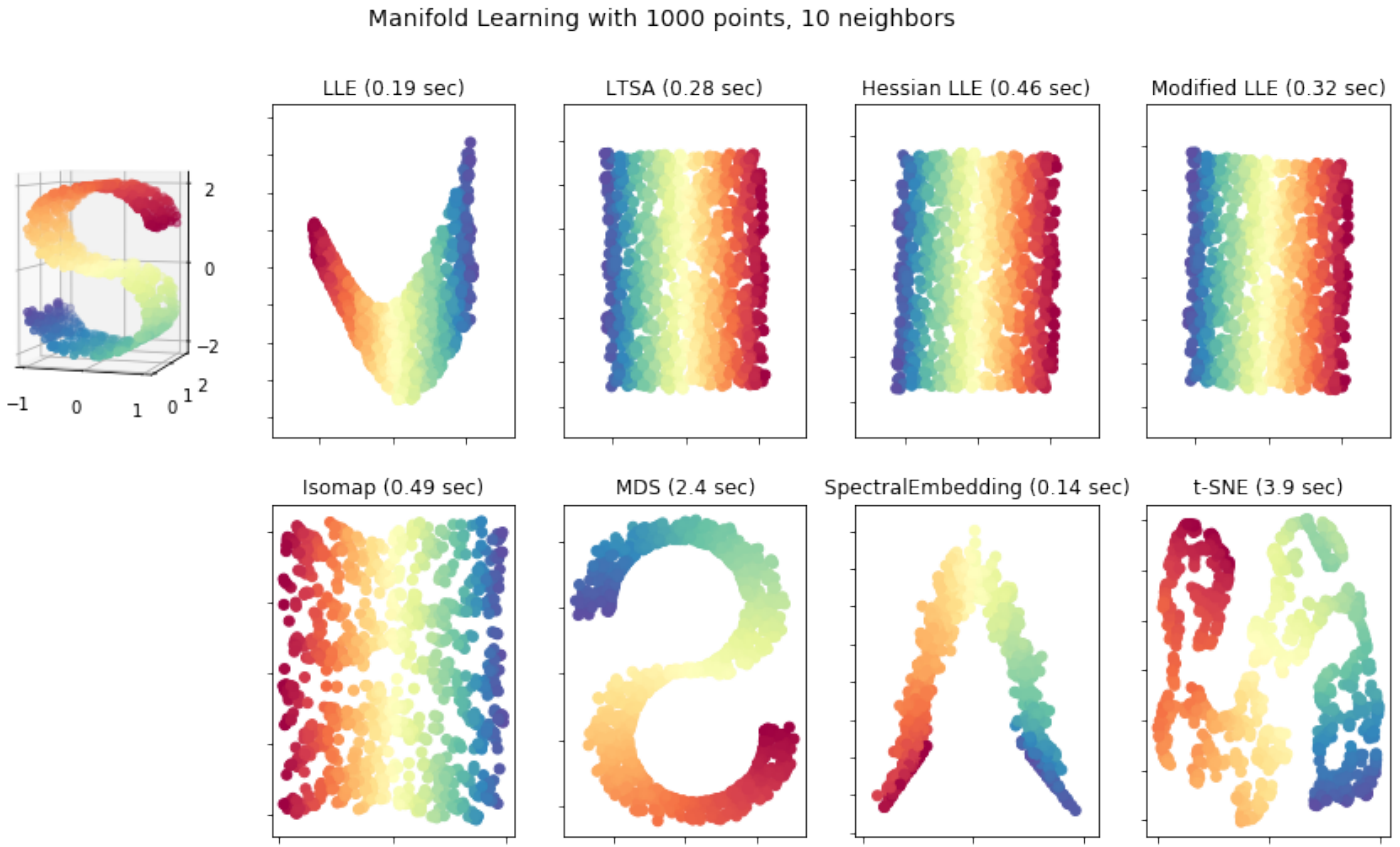
- T-SNE
  - sklearn.manifold.TSNE(n_components,..learning rate,n_iter, ..)
  - http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

Parameters	
n_components	Embedded space의 차원
learning rate	Learning rate의 값을 설정해 줌.(주로 [10~1000]의 값을 갖는다.)
n_iter	Maximum number of iterations for the optimization

Methods	
fit(X[, y])	Data X를 Embedded space fitting
fit_transform(X[, y])	Data X로 model fitting 뒤 X를 transform 시킴

Manifold Learning

- Manifold Learning 결과



Wine dataset

- 지금까지 주어진 Dimension Reduction Methods로 dimension을 변화시키면서 LR을 학습하여 비교
  - Test data에 대해서 transform이 가능한 경우만 사용!
  - 차원: 2/3/5 인 경우!

Swissroll data에 적용!

- **from sklearn import** manifold, datasets X, color = datasets.samples_generator.make_swiss_roll(n_samples=1500)

Reference

- [Raschka. (2017)] Raschka, Sebastian, and Vahid Mirjalili. *Python machine learning*. Packt Publishing Ltd, 2017.
- [Müller. (2016)] Müller, Andreas C., and Sarah Guido. *Introduction to machine learning with Python: a guide for data scientists*. 2016.
- [GÉRON. (2017)] GÉRON, Aurélien. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* Sebastopol, CA: O, 2017.