


ML_07_2. 차원 축소

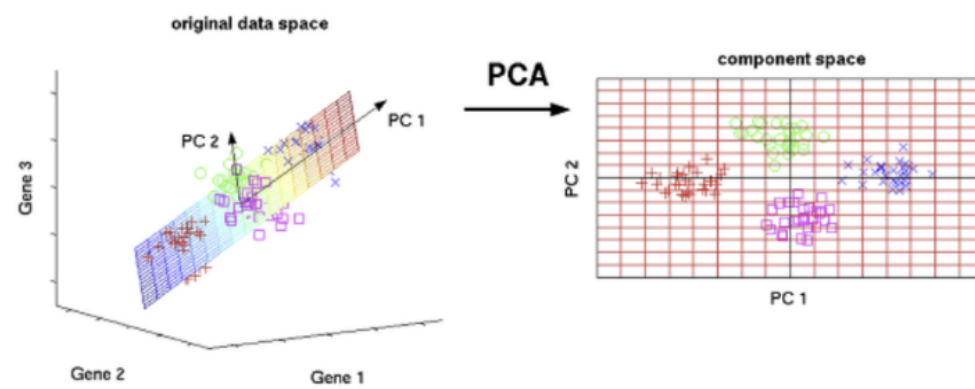
🕒 생성일	@2022년 6월 9일 오후 3:26
📁 유형	머신러닝/딥러닝
👤 작성자	 동훈 오

차원 축소는 pca 를 중점으로 코드 추가해가며 다룰 예정.

Business Analytics : PCA

고차원 공간에서는 노이즈가 발생할 가능성이 높고 계산 복잡성이 증가한다. 데이터를 설명하는 변수들의 상관관계가 여러 생기게 되고, 모델 퍼포먼스가 감소할 수 밖에 없다. 이러한 문제점을 해결하기 위해 데이터를 저차원으로 낮추어 저차원 공간 상에서 다루는 아이디어가 제시되었다. 다양한 차원 축소 알고리즘들이 있지만 대개 공통적인 목적은 저차원에서도 데이터를 잘 설명하는 변수들의 부분 집합을 찾는 것이다.

overview



PCA (Principal components analysis)

pca 는 원본 데이터의 분산을 보존하는, 서로 수직인 기저 집합을 찾는 것이다. 데이터가 투영된 이후에도 데이터의 분산을 최대한 보존하는 방향으로 투영시킨다. 이때 투영 방향은 차원 축소 이후 데이터의 분산이 큰 방향으로 진행한다.

covariance

pca 원리를 이해하기 위해서는 공분산(covariance)의 개념을 알아야 한다. 수식으로 나타내면 다음과 같다.

▪ \mathbf{X} : a data set (d by n, d: # of variables, n: # of records)

$$Cov(\mathbf{X}) = \frac{1}{n}(\mathbf{X} - \bar{\mathbf{X}})(\mathbf{X} - \bar{\mathbf{X}})^T$$

공분산은 서로 다른 변수들 간의 의존 관계를 수치적으로 표현한 것이다. 통계학에서는 x, y 간의 직선의 상관관계를 설명하는데 활용된다. 공분산의 직관적으로 어떤 변수가 증가 또는 감소 경향을 보일 때, 다른 변수가 그러한 경향을 어느 정도 따라가는가를 의미한다.

Eigenvalue & Eigenvector

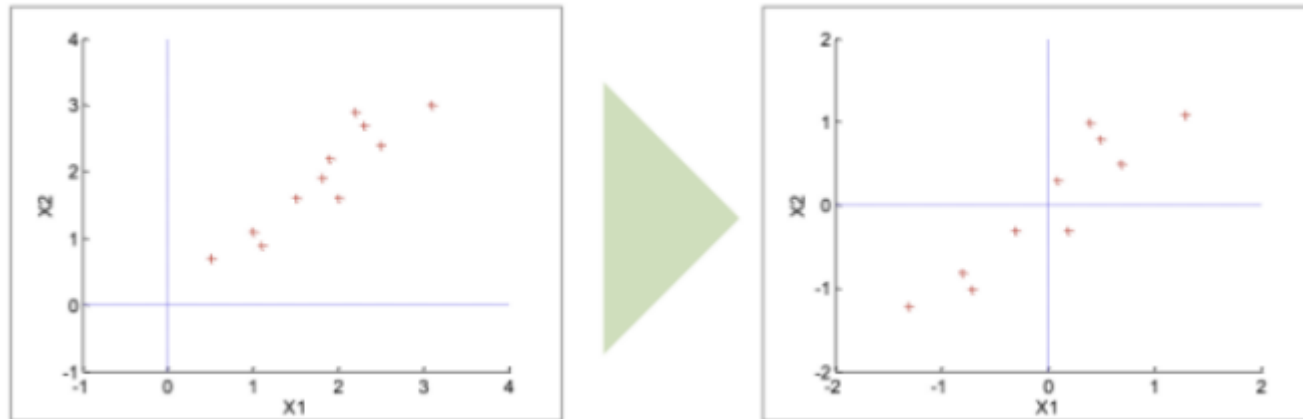
어떠한 벡터 A가 주어졌을 때, 아래 수식을 따르는 스칼라 값 λ 와 vector \mathbf{x} 가 존재한다. 각각을 고유값과 고유벡터라고 불린다.

$$Ax = \lambda x, (A - \lambda I)x = 0$$

고유값에 대응하는 벡터가 고유 벡터이다.

PCA Procedure

1. data centering : 데이터의 중심을 평균값으로 조정한다. 평균을 원점으로 맞추므로써 이후 공분산을 계산할 때 유용하다.

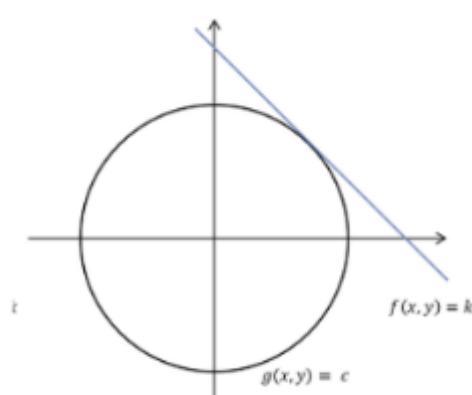


2. 최적화 문제 계산 : 벡터 X 가 기저 w 에 투영되었을 때 공분산을 계산한다. 아래 수식에서 S 는 벡터 X 가 정규화 되었을 때의 sample covariance matrix 라고 할 수 있다. PCA 목적은 아래 식의 V 를 최대화 하는 것이다. 즉, 저차원 투영 데이터의 분산을 크게 하는 것.

$$V = \frac{1}{n} (w^T X)(w^T X)^T = \frac{1}{n} w^T X X^T w = w^T S w$$

3. Lagrangian multiplier

pca 를 계산하는 데 라그랑지안 승수법의 테크닉이 필요하다. 라그랑지안 승수법을 설명하면 다음과 같다.



기본 가정

제약 조건 g 를 만족하는 f 의 최솟값 또는 최댓값은 f 와 g 가 접하는 지점에 존재할 수 있다.

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \quad (1) \quad \nabla f = \lambda \nabla g \quad (2)$$

어떠한 지점에서의 접선 벡터와 gradient 벡터의 내적은 0 이므로, f, g 의 gradient 가 서로 상수배인 관계.

Lagrangian multiplier 는 다음의 보조 함수가 정의된다.

$$L(x, y, \lambda) = f(x, y) - \lambda(g(x, y) - c) \quad (3)$$

위 가정에 따른 라그랑지안의 보조함수를 2번 과정에서 구한 식에 적용하면 다음과 같다.

$$\max \mathbf{w}^T \mathbf{S} \mathbf{w}$$

$$s.t. \mathbf{w}^T \mathbf{w} = 1$$

Lagrangian multiplier 를 적용

[결과]
 λ = 데이터 \mathbf{S} 의 고유값
 \mathbf{w} = 데이터 \mathbf{S} 의 고유 벡터

$$L = \mathbf{w}^T \mathbf{S} \mathbf{w} - \lambda(\mathbf{w}^T \mathbf{w} - 1)$$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{S} \mathbf{w} - \lambda \mathbf{w} = 0 \Rightarrow (\mathbf{S} - \lambda \mathbf{I}) \mathbf{w} = 0$$

4. 적용 & 설명된 분산

적용의 결과로 도출된 고유값, 고유 벡터는 1번 과정에서 그래프에 나타나는 데이터를 이용한 결과 값이다. 'explained variance' 라는 표현은 데이터를 고유 벡터 를 이용해 저차원 공간으로 이동시켰을 때, 원본 데이터에 대한 분산을 의미한다.

$$Eigenvectors = \begin{bmatrix} 0.6779 & -0.7352 \\ 0.7352 & 0.6779 \end{bmatrix} \quad Eigenvalues = (1.2840 \quad 0.0491)$$

$$\mathbf{v} = (\mathbf{w}_1^T \mathbf{X})(\mathbf{w}_1^T \mathbf{X})^T = \mathbf{w}_1^T \mathbf{X} \mathbf{X}^T \mathbf{w}_1 = \mathbf{w}_1^T \mathbf{S} \mathbf{w}_1$$

$$\mathbf{S} \mathbf{w}_1 = \lambda_1 \mathbf{w}_1, \quad \mathbf{w}_1^T \mathbf{S} \mathbf{w}_1 = \mathbf{w}_1^T \lambda_1 \mathbf{w}_1 = \lambda_1 \mathbf{w}_1^T \mathbf{w}_1 = \lambda_1$$

실제로 계산을 해보면, λ_1 값은 1.2840 이 나오고, λ_2 값은 0.0491이 나온다.

$$(\lambda_1) / (\lambda_1 + \lambda_2) = 0.96$$

2 차원 데이터를 첫 번째 주성분에 사영시키면 원데이터 분산의 96% 가 보존된다.

PCA - 데이터 재구성

압축된 데이터셋에 PCA 변환을 반대로 적용하면 원래의 차원으로 되돌릴 수 있다. 사이킷런에서는 `pca.inverse_transform()` 메서드를 제공하여 구현할 수 있다. 다만, 원본과 재구성 데이터 사이의 차이가 생길 수 밖에 없고, 차이를 재구성 오차라고 한다. 재구성 오차는 공간 상 데이터 분포를 살펴볼 때, 데이터 간 평균 제곱 거리를 의미한다.

아래 코드는 혼공머신에서 제시된 PCA 관련 예제 코드이며 사이킷런의 PCA document를 통해 PCA 클래스를 설명한다.

```
# data load
!wget https://bit.ly/fruits_300_data -O fruits_300.npy

import numpy as np
fruits = np.load('fruits_300.npy')

# [(전체 데이터 개수), (데이터 하나 사이즈)]
fruits_2d = fruits.reshape(-1, 100*100)

# PCA 객체 생성 및 적용
from sklearn.decomposition import PCA
pca = PCA(n_components=50)
pca.fit(fruits_2d)

# 데이터 복원(재구성)
fruits_pca = pca.transform(fruits_2d)
```

데이터 재구성이 가능한 이유는 사이킷런 pca document 를 살펴보면 알 수 있다. 우선 PCA 클래스를 살펴보자.

```
class PCA(_BasePCA):
    """
    Attributes
    -----
    components_ :
        Principal axes in feature space, representing the directions of
        maximum variance in the data.

    explained_variance_ :
        The amount of variance explained by each of the selected components.
    """
```

PCA 클래스는 주성분에 해당하는 components_ 와 설명된 분산 explained_variance_ 를 속성으로 저장하고 있다. 좀 더 깊게 들어가면, PCA 클래스가 상속하고 있는 _BasePCA 클래스를 살펴본다.

```
class _BasePCA(**kwargs):

    def inverse_transform(self, X):
        """
        Transform data back to its original space.
        In other words, return an input `X_original` whose transform would be X.
        Parameters
        -----
        X : array-like of shape (n_samples, n_components)
            New data, where `n_samples` is the number of samples
            and `n_components` is the number of components.
        Returns
        -----

        Notes
        -----
        If whitening is enabled, inverse_transform will compute the
        exact inverse operation, which includes reversing whitening.
        """
        if self.whiten:
            return (
                np.dot(
                    X,
                    np.sqrt(self.explained_variance_[:, np.newaxis]) * self.components_,
                )
                + self.mean_
            )
        else:
            return np.dot(X, self.components_) + self.mean_
```

'inverse_transform' 은 데이터를 원래의 차원으로 복원하는 메서드이다. 설명을 보면, 화이트닝 변환을 통해서 (사실상 pca 과정을 거꾸로 하는 계산.) 데이터 복원이 된다고 한다. 중요한 것은 연산에 사용되는 속성 값들이 이미 저장되어 있는 주성분과 설명된 분산이다. 따라서 PCA.inverse_transform() 코드 한줄 만으로도 데이터 복원이 가능하다.

(화이트닝 변환은 마할라노비스 거리 등의 개념을 추가적으로 알아야 하므로 생략.)

레퍼런스:

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/94e6235e-493e-403b-ac04-6823b34ae51d/01_4_Dimensionality_Reduction_PCA.pdf