# 13장 주성분 분석(PCA) : 고객 데이터셋 및 익명 데이터셋

#### 학습 목표

비지도 학습에 속하는 PCAPrincipal Component Analysis, 주성분 분석을 구현하고, 어떻게 활용/해석하는지 학습하고, PCA의 원리를 이해합니다. 두 가지 데이터셋을 사용해서 각각 ‘차원 축소 후 그래프 그리기’, ‘

#### 학습 순서



#### PCA 소개

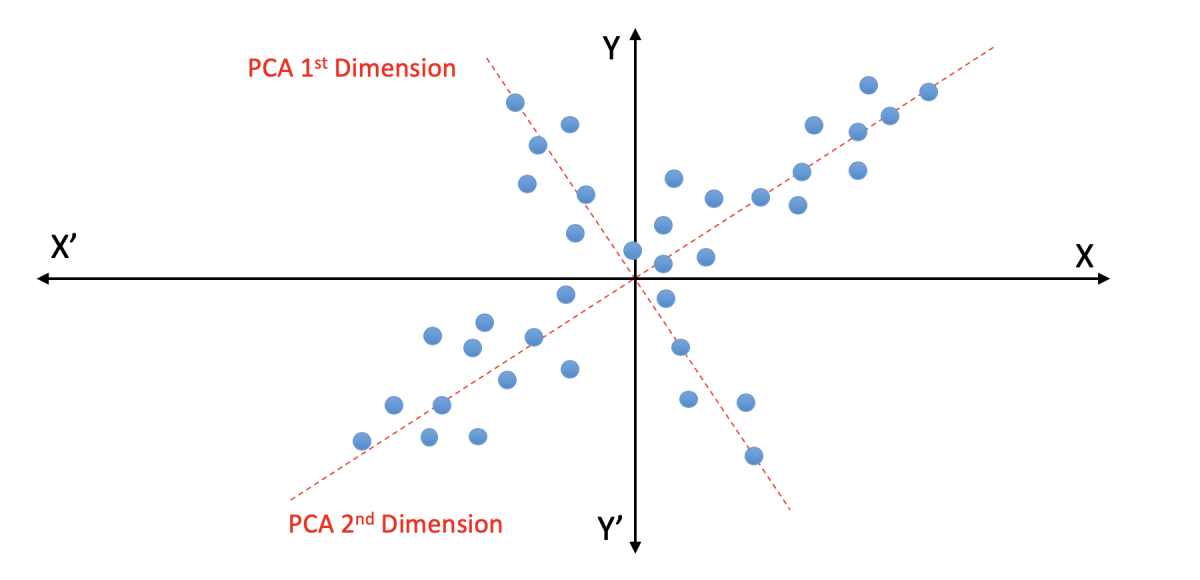
PCA는 Principal Component Analysis의 약자로, 주성분 분석이라고도 부릅니다. PCA는 지금까지 배운 알고리즘과는 전혀 다른 방식의 알고리즘입니다. 비지도 학습에 속하기 때문에 당연히 종속 변수는 존재하지 않고, 어떤 것을 예측하지도 분류하지도 않습니다. PCA의 목적은 데이터의 차원을 축소하는 데 있습니다. 차원 축소를 간단히 말하면 변수의 개수를 줄이되, 가능한 그 특성을 보존해내는 기법입니다. PCA는 기존의 변수 중 일부를 그대로 선택하는 방식이 아니라, 기존 변수들의 정보를 모두 반영하는 새로운 변수들을 만드는 방식으로 차원 축소를 합니다.

<용어/>

**차원 축소**

변수 두 개면 2차원 그래프로, 세 개면 3차원 그래프로 나타낼 수 있습니다. 즉, 데이터의 차원은 변수의 개수와 직결되는 겁니다. 차원 축소는 변수의 수를 줄여 데이터의 차원을 축소합니다.

</>



#### 장단점

| **장점** | **단점** |
| --- | --- |
| 다차원을 2차원에 적합하도록 차원 축소하여 시각화에 유용합니다. | 기존 변수가 아닌 새로운 변수를 사용하여 해석하는 데 어려움이 있습니다. |
| 변수 간의 높은 상관관계 문제를 해결해줍니다. | 차원이 축소됨에 따라 정보 손실이 불가피합니다. |

#### 유용한 곳

* 다차원 변수들을 2차원 그래프로 표현하는 데 사용할 수 있습니다.
* 변수가 너무 많아 모델 학습에 시간이 너무 오래 걸릴 때 (차원 축소를 진행하면 학습에 드는 시간을 줄일 수 있어) 유용합니다.
* 오버피팅을 방지하는 용도로 사용할 수도 있습니다.

#### TOP 10 선정 이유

* PCA는 차원 축소 방법 중 가장 인기있으며 구현하기 또한 쉬운 편입니다. 프로젝트 특성에 따라서 차원 축소가 필요하지 않은 경우도 많지만, 차원 축소를 시도해봄으로써 시각화 내지 모델링 효율성을 개선할 여지는 항상 있습니다. 따라서 알아두면 언젠가 유용하게 쓰게 될 알고리즘입니다.

## 13.1 차원을 축소해서 그래프 그리기 : 고객 데이터셋

그래프를 그려서 고객 데이터셋을 분석합니다. 이 과정에서 차원을 축소하는 방법을 알아보겠습니다.

### 13.1.1 문제 정의 : 한눈에 보는 분석 목표

<금토끼의 문제 정의> 금토끼는 지난번 클러스터링 모델의 예측 결과에 대한 프레젠테이션을 준비하고 있었습니다. 클러스터링 알고리즘을 통하여 고객들을 여러 그룹으로 나누었으나, 여기에 사용한 독립변수들이 너무 많아서 이것들을 한눈에 보이는 그림으로 표현하기 불가능했습니다. 고객 그룹들이 각각 얼마나 잘 분리되어있는지를 시각적으로 보여주고 싶어서 방법을 궁리하던 금토끼는, PCA 알고리즘을 이용하여 여러 변수를 압축하는 차원 축소 방법을 사용하기로 했습니다.

| **난이도** | ⭐⭐☆ | | |
| --- | --- | --- | --- |
| **알고리즘** | 주성분 분석(Principal Component Analysis, PCA) | | |
| **데이터셋 파일명** | customer\_pca.csv | | |
| **데이터셋 소개** | 12장 K-평균 군집화에서 사용한 예측 결과가 포함된 최종 고객 분석 데이터셋을 사용합니다. PCA를 사용해 시각화를 용이하게 하는 방법을 학습합니다. | | |
| **미션** | 데이터의 차원을 축소하여 이해하기 쉽게 시각화하시오. | | |
| **문제 유형** | 비지도 학습 | **평가지표** | 설명된 분산 |
| **사용한 모델** | PCA | | |
| **사용 라이브러리** | * numpy (numpy==1.19.5) * pandas (pandas==1.3.5) * seaborn (seaborn==0.11.2) * matplotlib (matplotlib==3.2.2) * sklearn (scikit-learn==1.0.2) | | |
| **예제 코드 노트북** | 위치 : <https://github.com/musthave-ML10/notebooks/>  파일 : 13\_PCA.ipynb | | |

### 13.1.2 라이브러리 및 데이터 불러오기

필수 라이브러리와 예측 결과가 포함된 고객 분석 데이터인 customer\_pca.csv를 불러옵니다.

| import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns  file\_url = 'https://raw.githubusercontent.com/musthave-ML10/data\_source/main/customer\_pca.csv' customer = pd.read\_csv(file\_url) # 데이터셋 읽기 |
| --- |

지난 장에서 이미 살펴본 데이터이지만 head() 함수를 호출해 다시 한번 데이터를 간단히 살펴보겠습니다.

| customer.head() # 상위 5행 출력 |
| --- |

|  | amt | category\_entertainment | category\_food\_dining | category\_gas\_transport | category\_grocery | category\_health\_fitness | category\_home | category\_kids\_pets | category\_misc | category\_personal\_care | category\_shopping | category\_travel | label |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | -1.40 | -1.14 | -0.97 | -1.00 | -1.12 | -1.56 | -1.15 | -1.28 | -1.12 | -1.07 | -1.14 | -0.62 | 0 |
| 1 | 1.08 | 0.41 | 0.17 | 2.09 | 0.77 | 1.04 | 0.84 | 1.13 | 1.21 | 1.81 | 0.66 | -0.22 | 3 |
| 2 | 1.20 | 0.75 | 1.62 | -0.61 | 0.44 | 1.38 | 2.13 | 0.38 | -0.20 | 0.13 | 1.30 | 2.77 | 1 |
| 3 | -1.47 | -1.13 | -1.54 | -0.93 | -1.08 | -1.22 | -1.57 | -1.52 | -1.27 | -1.19 | -1.17 | -0.60 | 0 |
| 4 | 0.90 | 0.26 | -0.32 | 2.08 | 0.54 | 0.71 | 1.10 | 0.96 | 1.20 | 1.64 | 0.58 | -0.48 | 3 |

고객별 총 지출금액 및 카테고리별 지출금액이 스케일링된 상태로 정리되어 있고, 마지막 컬럼에는 각 고객이 속한 클러스터 라벨(label)이 들어있습니다.

9장에서 첫 번째 연습용 데이터로 클러스터링했을 때 클러스터가 잘 나뉘었는지 확인하는 데 산점도 그래프를 이용했습니다. 당시 사용한 데이터는 변수가 오직 2개 뿐이었기에 x축과 y축에 각각 지정하여 쉽게 그래프를 그릴 수 있지만, 지금은 변수가 너무 많아서 2차원의 그림으로 표현하기 어렵습니다. PCA를 사용하여 이 변수들을 2 변수로 축소한 뒤 산점도 그래프로 출력해 클러스터가 어떻게 나뉘었는지를 확인하겠습니다.

우선 독립변수와 종속변수를 분리합니다. 종속변수(label)는 그대로 유지되어야 하기 때문에 PCA를 적용할 대상에서 제외해야 합니다.

| customer\_X = customer.drop('label', axis = 1) # 독립변수 지정 customer\_y = customer['label'] # 종속변수 지정 |
| --- |

### 13.1.3 그래프 표현을 위한 차원 축소

PCA 알고리즘은 사이킷런의 decomposition에서 불러올 수 있습니다.

| from sklearn.decomposition import PCA # 임포트 |
| --- |

다음은 모델링이나 스케일링처럼 특정 이름으로 속성을 부여해야 하는데, 여기에서 몇 개의 주성분(PC)으로 분석할지를 하이퍼파라미터로 정해야 합니다. 이전 장에서 배운 K-평균처럼 이를 사용자가 직접 설정해야 합니다. 2차원의 평면에 그래프를 그리는 것이 목적이기 때문에 2개 주성분만을 가지도록 설정합니다.

| pca = PCA(n\_components=2) # 주성분 개수 지정 |
| --- |

pca에 속성이 할당되었으니, fit()과 transform()을 사용하여 차원을 축소하겠습니다. 코드 형태는 스케일링 코드와 유사합니다.

| pca.fit(customer\_X) # 학습 customer\_pca = pca.transform(customer\_X) # 변환 |
| --- |

customer\_pca를 출력하면 다음과 같은 넘파이 형태로 출력됩니다.

| customer\_pca # 결과 확인 |
| --- |

array([[-3.92906072e+00, 1.02604491e-01],

[ 3.10758276e+00, -1.74887930e+00],

[ 3.02379272e+00, 3.21221215e+00],

[-4.28241767e+00, 1.13781030e-02],

[ 2.59065802e+00, -1.90612064e+00],

[ 8.70343520e-01, -1.79154348e-02],

[-1.16508938e+00, 2.00387900e+00],

[ 1.41163657e+00, -1.21220170e+00],

... 생략 ...

PCA도 스케일링과 마찬가지로 넘파이 형태의 결과물을 만들어내기 때문에, 이를 판다스 데이터프레임 형태로 변환해주겠습니다. 각 변수 이름은 PC1, PC2로 지정하겠습니다..

| customer\_pca = pd.DataFrame(customer\_pca, columns = ['PC1','PC2']) # 데이터프레임으로 변환 |
| --- |

변환된 데이터프레임에 기존 데이터의 목푯값인 label을 붙이겠습니다.

| customer\_pca = customer\_pca.join(customer\_y) # 데이터 합치기 |
| --- |

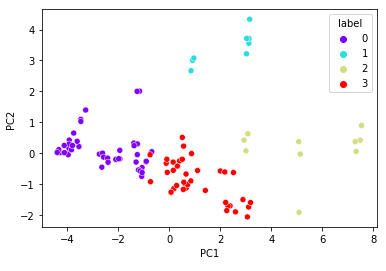
이제 최종 데이터를 head로 확인해봅시다.

| customer\_pca.head() # 상위 5행 출력 |
| --- |

|  | PC1 | PC2 | label |
| --- | --- | --- | --- |
| 0 | -3.929061 | 0.102604 | 0 |
| 1 | 3.107583 | -1.748879 | 3 |
| 2 | 3.023793 | 3.212212 | 1 |
| 3 | -4.282418 | 0.011378 | 0 |
| 4 | 2.590658 | -1.906121 | 3 |

이제 명확하게 단 2개 변수와 label이 있으니, 9장의 초반에 만든 산점도 그래프를 그릴 수 있습니다.

| sns.scatterplot(x='PC1',y='PC2', data = customer\_pca, hue = 'label', palette='rainbow') # 산점도 그리기 |
| --- |



보라색과 빨간색 클러스터는 가깝게 붙어 있어서 경계가 모호하기는 하지만 언뜻 보기에 그럴싸하게 잘 나뉜 것 같습니다. 안타깝게도 이 그림에서 알 수 있는 것은 클러스터들이 얼마나 잘 나뉘었는지를 대략 확인하는 것뿐입니다. PCA를 통해 얻어낸 변수 PC1과 PC2는 기존의 모든 변수를 복합적으로 반영하여 만들어졌기 때문에 명료하게 이 새로운 변수들을 해석하기가 쉽지 않습니다.

그나마 추가적으로 각 주성분과 기존 변수와의 상관관계를 알 수 있습니다. PCA 학습에 사용한 pca 뒤에 components\_를 붙여 이를 확인할 수 있습니다.

| pca.components\_ # 주성분과 변수의 관계 확인 |
| --- |

array([[ 0.3484681 , 0.32447242, 0.30303652, 0.14186907, 0.30618347,

0.31297263, 0.29718852, 0.3045823 , 0.29341337, 0.30287672,

0.32053447, 0.08927503],

[ 0.05827591, 0.06034266, 0.15264674, -0.54435586, 0.03109502,

0.03790586, 0.23809571, -0.2315275 , -0.2471928 , -0.20898284,

0.14479001, 0.65946781]])

넘파이 형태로 출력되기 때문에 읽기 쉽도록 데이터프레임으로 변환하겠습니다. 각 columns에 기존 데이터의 독립변수 이름들을 입력합니다.

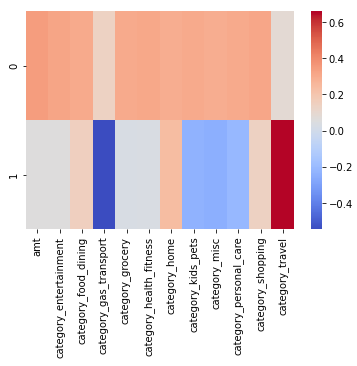
| df\_comp = pd.DataFrame(pca.components\_,columns=customer\_X.columns) # 데이터프레임으로 변환 |
| --- |

이제 df\_comp를 확인하면 다음과 같은 데이터를 확인할 수 있습니다.

|  | amt | category\_entertainment | category\_food\_dining | category\_gas\_transport | category\_grocery | category\_health\_fitness | category\_home | category\_kids\_pets | category\_misc | category\_personal\_care | category\_shopping | category\_travel |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0.35 | 0.32 | 0.30 | 0.14 | 0.31 | 0.31 | 0.30 | 0.30 | 0.29 | 0.30 | 0.32 | 0.09 |
| 1 | 0.06 | 0.06 | 0.15 | -0.54 | 0.03 | 0.04 | 0.24 | -0.23 | -0.25 | -0.21 | 0.14 | 0.66 |

왼쪽의 인덱스 0과 1은 각각 주성분 PC1과 PC2를 의미합니다. 그리고 위쪽 컬럼명에는 기존 데이터의 독립변수들이 나열되어 있습니다. 이 숫자들이 의미하는 것은 특정 주성분과 특정 변수와의 상관관계입니다. 예를 들어 인덱스 0(PC1)과 amt와의 상관관계는 0.35이므로, PC1이 높다는 것은 그만큼의 크기만큼 amt가 높다는 의미입니다. 다른 예로 인덱스1(PC2)와 category\_gas\_transport를 보면 -0.54입니다. 이는 PC2가 클수록 category\_gas\_transport에 해당하는 값은 낮다는 의미입니다. 각 주성분은 이렇게 수많은 기존 변수와의 관계성을 내포하여 이루어졌기 때문에 명료하게 해석하는 데 어려움이 있습니다. 마지막으로 앞의 테이블을 heatmap으로 만들어 시각적으로 표현해보겠습니다.

| sns.heatmap(df\_comp,cmap='coolwarm') # 히트맵 그리기 |
| --- |



양수이면 빨간색, 음수이면 파란색 계열로 표현되어 있어, 각 주성분이 변수들과 어떠한 상관관계를 보이는지 쉽게 읽어낼 수 있습니다.

## 13.2 속도와 예측력을 향상시키기 : 익명 데이터셋

차원 축소를 진행해 학습 시간을 줄이고 성능을 향상시키는 방법을 알아보겠습니다.

### 13.2.1 문제 정의 : 한눈에 보는 분석 목표

금토끼는 익명의 고객에게 최대한 빠르고 정확한 예측 모델을 만들어달라는 요청을 받았습니다. 비밀리에 진행되는 프로젝트이기 때문에 이 데이터의 독립변수 이름들은 모두 익명처리가 되어 있지만, 금토끼는 이런 데이터를 가지고도 충분히 좋은 예측 모델을 만들어낼 수 있었습니다. 하지만 금토끼는 만족하지 못했습니다. "지금 모델보다 좀 더 빠르고 정확한 모델을 만들 여지는 없을까?"하는 생각이 들어서, 주성분 분석을 활용해 독립변수를 줄이는 차원 축소를 시도해보기로 합니다.

| **난이도** | ⭐⭐☆ | | |
| --- | --- | --- | --- |
| **알고리즘** | 주성분 분석(Principal Component Analysis, PCA) | | |
| **데이터셋 파일명** | anonymous.csv | | |
| **데이터셋 소개** | 변수 개수가 천 개가 넘는 데이터셋입니다. 변수 이름이 익명 처리되어 있습니다. 차원 축소 전후의 모델 학습 속도 및 예측 결과를 비교해보겠습니다. | | |
| **미션** | 데이터의 차원을 축소해 학습 시간을 줄이고 성능 상향을 향상시키세요. | | |
| **문제 유형** | 비지도 학습 | **평가지표** | AUC (두 번째 데이터셋 한정) |
| **사용한 모델** | PCA | | |
| **사용 라이브러리** | * numpy (numpy==1.19.5) * pandas (pandas==1.3.5) * seaborn (seaborn==0.11.2) * matplotlib (matplotlib==3.2.2) * sklearn (scikit-learn==1.0.2) | | |
| **예제 코드 노트북** | 위치 : <https://github.com/musthave-ML10/notebooks/>  파일 :13\_PCA.ipynb | | |

### 13.2.2 다차원 데이터 불러오기

이번에는 anonymous.csv 데이터셋을 사용합니다. 데이터가 크기때문에 불러오는 데 시간이 좀 걸립니다.

| file\_url = 'https://media.githubusercontent.com/media/musthave-ML10/data\_source/main/anonymous.csv' anonymous = pd.read\_csv(file\_url) |
| --- |

데이터 크기가 매우 크고, 어차피 변수 이름에 아무 정보가 없기 때문에 간단하게만 데이터를 살펴보고 넘어가겠습니다.

| anonymous.head() # 상위 5행 출력 |
| --- |

우선 head() 함수를 호출해 어떻게 생겼는지 보겠습니다.

|  | class | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V4287 | V4288 | V4289 | V4290 | V4291 | V4292 | V4293 | V4294 | V4295 | V4296 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1.00 | 0.00 | 0.00 | 0.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.83 | 0.84 | ... | -0.26 | 0.12220 | 0.348620 | 0.12957 | 0.43846 | -0.108020 | 0.128330 | 0.318820 | -0.041559 | 0.22589 |
| 1 | 1.00 | 0.00 | 0.00 | 0.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.83 | 0.84 | ... | -0.15 | 0.14062 | -0.030201 | 0.10134 | -0.14546 | -0.166650 | 0.401300 | 0.035392 | 0.019906 | 0.31952 |
| 2 | 1.00 | 0.00 | 0.00 | 0.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.83 | 0.84 | ... | -0.40 | 0.13790 | 0.138350 | 0.15746 | 0.51216 | -0.330690 | 0.070346 | 0.179250 | -0.188740 | 0.16386 |
| 3 | 1.00 | 0.00 | 0.00 | 0.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.83 | 0.84 | ... | -0.16 | -0.26686 | 0.158930 | -0.13204 | 0.32221 | -0.042086 | 0.588970 | 0.526990 | -0.574320 | 0.20891 |
| 4 | 1.00 | 0.00 | 0.00 | 0.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.83 | 0.84 | ... | 0.24 | 0.44605 | -0.163560 | 0.45311 | 0.59073 | 0.356800 | 0.395780 | 0.035871 | -0.293430 | 0.38124 |

대부분 변수 이름이 V1, V2, V3처럼 되어 있어서 의미를 알수가 없습니다. 독립변수는 총 4296개입니다. 종속 변수(class)는 0과 1로 구성되어 있습니다.

그럼 이제 종속 변수인 class에서 1의 비율을 확인해봅시다. 평소 같으면 describe() 함수를 사용하여 전반적인 통계 정보를 출력해 눈으로 확인하겠지만, 지금은 변수가 너무 많아 mean() 함수를 사용해서 비율만 확인하겠습니다.

| anonymous['class'].mean() # 종속변수의 평균 확인 |
| --- |

0.25

다음은 결측치가 없는지 확인하겠습니다. 변수 종류가 많지 않을 때는 mean()이나 sum() 함수를 한 번 사용해서 눈으로 확인했지만, 지금은 변수 개수가 4천 개가 넘어 일일이 확인할 수가 없습니다. 따라서 변수별 결측치 합을 구하고, 거기에 또 한 번 더 합을 구해서, 데이터프레임 전체에 결측치가 얼마인지만 확인하겠습니다.

| anonymous.isna().sum().sum() # 결측치 확인 |
| --- |

0

결측치가 하나도 없기 때문에 결측치 처리 이슈는 없습니다. PCA에 따른 모델 성능차를 비교해야 하므로 피처 엔지니어링 없이 곧바로 모델링으로 넘어가겠습니다.

### 13.2.3 PCA에 따른 모델링 성능/결과 비교하기

모델링에 사용할 데이터를 훈련셋, 시험셋으로 분할하겠습니다. 데이터가 충분히 크니 test\_size를 0.2로 설정합니다.

| from sklearn.model\_selection import train\_test\_split X\_train, X\_test, y\_train, y\_test = train\_test\_split(anonymous.drop('class', axis=1), anonymous['class'], test\_size=0.2, random\_state=100) # 학습셋, 시험셋 분리 |
| --- |

PCA에서도 변수 간의 스케일을 일정하게 맞춰주는 것이 중요합니다. 따라서 모델링 전에 StandardScaler()를 적용하겠습니다.

| from sklearn.preprocessing import StandardScaler # 임포트  scaler = StandardScaler() # 스케일러 객체 생성 scaler.fit(X\_train) # 학습  X\_train\_scaled = scaler.transform(X\_train) # 변환 X\_test\_scaled = scaler.transform(X\_test) # 변환 |
| --- |

모델 학습에 사용할 데이터 준비를 마쳤습니다. 모델링에 사용할 알고리즘 정할 차례입니다. PCA를 사용하면 학습 시간이 얼마나 단축되는지를 확인하려는 목적도 있으므로 시간이 다소 걸리는 랜덤 포레스트를 사용하겠습니다.

| from sklearn.ensemble import RandomForestClassifier # 임포트 model\_1 = RandomForestClassifier(random\_state=100) # 모델 객체 생성 |
| --- |

하이퍼파라미터로 random\_state를 설정했습니다. model\_1으로 학습시키며 소요 시간을 계산하는 코드를 작성합니다.

| import time # 임포트 start\_time = time.time() # 시작시간 설정 model\_1.fit(X\_train, y\_train) # 학습 print(time.time() - start\_time) # 소요 시간 출력 |
| --- |

109.44180679321289

제 PC에서는 109초가 걸렸습니다. 이제 학습된 모델을 가지고 정확도를 확인하겠습니다.

| from sklearn.metrics import accuracy\_score, roc\_auc\_score # 임포트  pred\_1 = model\_1.predict(X\_test\_scaled) # 예측 accuracy\_score(y\_test, pred\_1) # 정확도 확인 |
| --- |

0.755

종속 변수에서 1의 비율이 약 25%였던 점을 고려하면 정확도 75.5%를 좋다고 말하기는 어렵습니다.

다음은 AUC입니다. AUC는 소수점 형태로 된 예측값을 사용하기 때문에 predict가 아닌 predict\_proba를 사용해야 한다는 점 다시 한번 강조하겠습니다.

| proba\_1 = model\_1.predict\_proba(X\_test\_scaled) # 예측 roc\_auc\_score(y\_test, proba\_1[:,1]) # AUC 확인 |
| --- |

0.8441136991070199

0.84로 기대했던 것보다는 괜찮은 수준입니다. 그럼 이제 PCA를 사용하면 어떻게 달라지는지 확인하겠습니다.

우선 몇 개의 주성분으로 만들 것인지를 정해야 합니다. 앞서 2차원 그래프를 그릴 때는 주성분을 2개로 설정했습니다. 4천 개가 넘는 변수들을 종전과 같이 단 2개로 차원 축소하면 데이터 손실이 너무 클 것으로 염려됩니다. 일단 시험 삼아 주성분을 2개로 지정해 학습해보겠습니다. 그 후 얼마만큼의 데이터 손실이 있는지 확인하겠습니다.

| pca = PCA(n\_components=2) # 주성분 개수 지정 pca.fit(X\_train\_scaled) # 학습 |
| --- |

pca에 explained\_variance\_ratio\_를 붙여 각 주성분이 기존 변수의 분산을 얼마만큼 대변해주고 있는지를 확인해보겠습니다.

<note/>

explained\_variance\_ratio\_ : 기존 변수들의 분산을 주성분이 어느 비율로 설명하는지 보여주는 지표. 우리말로 ‘설명된 분산’ 정도로 직역할 수 있습니다. 1은 기존 변수들의 분산을 100% 대변한다는 의미입니다. 여기서 ‘변수들의 분산’은 각 변수들이 가지는 데이터 분포입니다. 쉽게 말하면 기존 변수들의 특성을 얼마만큼 내포하고 있는가를 보여줍니다.

</>

| pca.explained\_variance\_ratio\_ # 데이터 반영 비율 확인 |
| --- |

array([0.04992445, 0.03331409])

각각 0.0499, 0.0333 정도로, 이 둘을 합쳐봐야 기존 데이터의 0.08 정도의 정보만 반영한다는 의미입니다. 즉 그만큼 정보의 손실이 크다는 의미입니다. 최적의 주성분 개수는 주관적인 판단에 의한 겁니다만, 우리가 K-평균에서 사용했던 엘보우 기법을 이용하면 조금 도움을 받을 수도 있습니다. 엘보우 기법에서처럼 다양한 숫자의 주성분을 만들어보고, 각 반복문에서 explained\_variance\_ratio\_에 대한 합이 어떤지를 확인해봅시다.

우선 explained\_variance\_ratio\_의 결과를 받아줄 빈 리스트를 만듭니다.

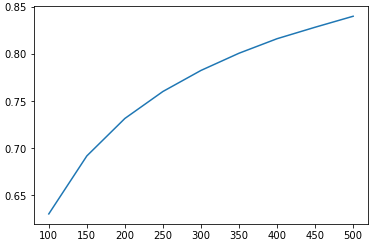
| var\_ratio = [] # 빈 리스트 생성 |
| --- |

이제 반복문을 만들 차례입니다. 변수가 총 4천 개 이상이기 때문에 확인할 영역이 넓습니다. 모든 변수를 확인하려면 엄청난 시간이 들 겁니다. 임의로 주성분 개수를 50개 단위로 100부터 500까지 확인하겠습니다.

| for i in range(100,550, 50): # 순회  pca = PCA(n\_components=i) # 주성분 개수 지정  pca.fit\_transform(X\_train\_scaled) # 학습 및 변환  ratio = pca.explained\_variance\_ratio\_.sum() # ❶ 데이터 반영비율 합  var\_ratio.append(ratio) # 반영비율 합을 리스트에 추가 |
| --- |

❶ explained\_variance\_ratio\_ 뒤에 sum()을 붙여서 주성분 전체의 데이터 반영 비율을 얻습니다. 그리고 이 정보를 var\_ratio 리스트에 추가시킵니다. 이를 엘보우 기법처럼 선형 그래프로 그려보겠습니다.

| sns.lineplot(x=range(100,550,50), y=var\_ratio) # 선형 그래프 그리기 |
| --- |



주성분 수를 100에서 500까지 확인한 결과 이 범위에서 얻을 수 있는 데이터의 반영 비율은 약 62%~82% 정도입니다. K-평균의 엘보우 기법에서처럼 드라마틱하게 꺾이는 부분이 없기 때문에 한눈에 적절한 값을 찾을 수는 없지만, 본인의 기준에 맞는 적정값을 찾는 가이드 라인으로 삼을 수 있습니다. 데이터 손실을 어디까지 감수할 것인지를 정하고 해당 수준에 적합한 주성분 수를 결정하면 되겠습니다. 주성분 개수를 정하는 데 정답은 없습니다. 본인이 허용할 수 있는 데이터 손실율을 고려하여 시도를 해보고, 모델링 이후에 점차 다른 수의 주성분을 시도해보는 식으로 발전시켜 나가야 합니다.

여기서는 약 80%를 기준으로 잡고, 이에 가장 근사치인 400을 채택하여, 주성분이 400개인 데이터를 만들겠습니다.

| pca = PCA(n\_components=400) # 주성분 개수 지정 pca.fit(X\_train\_scaled) # ❶ 학습 X\_train\_scaled\_pca = pca.transform(X\_train\_scaled) # ❷ 변환 X\_test\_scaled\_pca = pca.transform(X\_test\_scaled) # ❸ 변환 |
| --- |

스케일링 때와 마찬가지로 ❶ 주성분 분석을 할 때 훈련셋에만 fit()을 적용하고, 학습된 모델로 ❷ 훈련셋과 ❸ 시험셋을 transform()을 적용했습니다.

이제 PCA를 수행해 만든 데이터를 가지고 랜덤 포레스트로 예측 모델을 만들겠습니다.

| model\_2 = RandomForestClassifier(random\_state=100) # 모델 객체 생성  start\_time = time.time() # 시작 시간 지정 model\_2.fit(X\_train\_scaled\_pca, y\_train) # 학습 print(time.time() - start\_time) # 소요 시간 출력 |
| --- |

67.78307747840881

소요 시간은 약 61초로 기존 109.4초 대비 대략 절반 정도 수준으로 속도가 향상되었습니다. 그럼 이번에는 정확도와 AUC를 살펴보겠습니다.

| pred\_2 = model\_2.predict(X\_test\_scaled\_pca) # 예측 accuracy\_score(y\_test, pred\_2) # 정확도 확인 |
| --- |

0.99

기존 0.755보다 훨씬 높은 정확도를 보여줍니다.

| proba\_2 = model\_2.predict\_proba(X\_test\_scaled\_pca) # 예측 roc\_auc\_score(y\_test, proba\_2[:,1]) # AUC 확인 |
| --- |

0.9982789692274063

AUC 역시 0.998로 기존의 0.844에 비해 훨씬 좋은 결과입니다.

이러한 평가 지표와 소요 시간 등을 표로 정리하여 비교하겠습니다.

|  | 소요 시간 | Accuracy Score | AUC |
| --- | --- | --- | --- |
| PCA 이전 | 약 109초 | 0.755 | 0.8441 |
| PCA 이후 | 약 67초 | 0.99 | 0.998 |

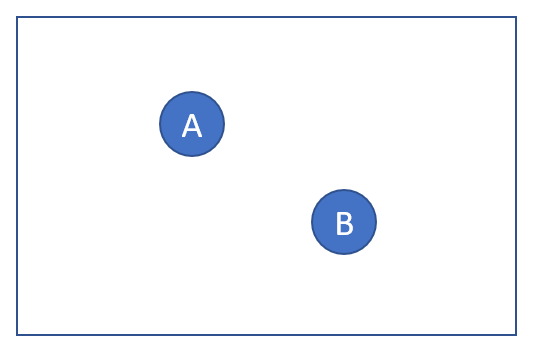
PCA를 사용해서 학습 시간을 줄이고 동시에 예측 정확도 또한 상승시켰습니다. 하지만 PCA가 언제나 이런 성과를 내는 것은 아닙니다. 사용하는 알고리즘에 따라, 다루는 데이터에 따라, 때로는 PCA가 그다지 적절한 역할을 해내지 못할 수도 있습니다. 따라서 PCA를 사용했는데도 속도와 예측 성능이 개선되지 않는다고 해서 무언가가 잘못된 것은 아닙니다. 그저 PCA를 사용하기에 적합하지 않은 상황일 뿐이니, 이러한 특징을 알고 사용해야 합니다.

## 13.3 이해하기 : 주성분 분석

PCA는 특성을 최대한 유지하는 방향으로 차원 축소를 한다고 설명한 바 있습니다. 이를 더 구체적으로 살펴보겠습니다.

### 13.3.1 3차원을 2차원으로 차원 축소하는 예

3차원 공간인 작은 방 안에 눈높이쯤에 오는 모빌 2개가 천장에 매달려 있다고 상상해봅시다. 방을 위쪽에서 내려다본 모습은 아래 그림과 같습니다.



이 방 안에서 여러분이 사진을 찍어서 이 2개의 모빌의 위치를 최대한 잘 담아내려면 어떻게 해야 할까요?

왼쪽 그림처럼 화살표의 위치에서 사진을 찍으면 모빌 A와 모빌 B가 완전히 겹쳐 보여서, 우측 그림처럼 A와 B 위치를 전혀 분간할 수 없게 됩니다.



그럼 다음 위치는 어떤가요? 모빌 A와 B가 겹치지는 않지만 다소 가까운 위치에 놓인 사진을 얻게 될 겁니다.



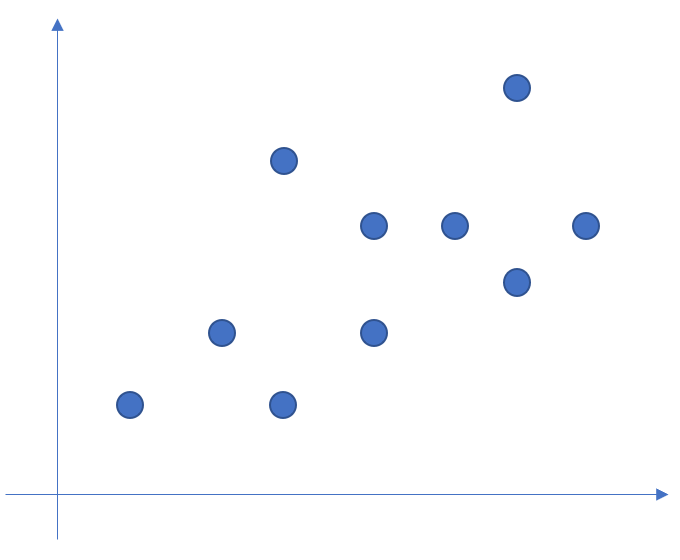
그렇다면 다음 위치는 어떤가요? 모빌 A와 B가 가장 동떨어진 위치로 포착되었습니다.



이 모빌들을 데이터라고 생각해보면 3차원 공간인 방에 위치한 데이터를, 2차원 평면인 사진으로 옮기는 작업을 수행한 겁니다. 사진 속 모빌 간의 거리가 곧 데이터의 기존 분산을 얼마나 유지하는지를 의미합니다. 첫 번째 사진에서는 A와 B가 전혀 구분되지 않기 때문에 데이터 분산이 0이 되도록 차원이 축소된 것이고, 마지막 사진은 A와 B가 가장 먼 거리(분산)를 보이며 구분되었기 때문에 기존 데이터의 분산을 거의 그대로 담아내면서 차원 축소된 것이라 볼 수 있습니다. PCA는 이처럼 데이터들의 분산을 최대한 담아내면서 차원 축소를 합니다.

### 13.3.2 2차원을 1차원으로 차원 축소하는 예

이번에는 그래프에 그려진 예시를 들어보겠습니다. 앞의 예가 3차원에서 2차원으로 차원 축소를 하는 예시라면, 이번에는 2차원 평면 그래프를 1차원인 선으로 차원 축소하는 예를 다룰 겁니다. 예시 데이터 분포는 다음과 같습니다.



2차원 평면 위의 데이터이기 때문에 차원 축소를 하면 1차원인 선이 됩니다. X 축과 Y축에 해당하는 선으로 차원 축소를 하면 어떤 모양이 되는지 확인해봅시다.

우선 X 축에 해당하는 선 위에 투영시키면 다음과 같은 모양이 될 겁니다.



이번에는 Y축에 해당하는 선으로 투영시켜보겠습니다.



마지막으로 X 축도 Y축도 아닌, 선형 회귀에 가까운 대각선을 기준으로 하여 투영시켜보겠습니다.



이제 각기 다른 방식으로 차원 축소가 된 결과물을 나란히 놓고 비교하겠습니다. 모든 결과물은 수평으로 재정렬하여 비교할 겁니다.

<그림/>



X 축 기준

Y 축 기준

대각선 기준

</>

각각 선 위에 투영된 데이터 분포가 어떻게 다른지 보이나요? ❶ x축에 투영된 결과물은 일부 데이터가 겹쳐 7개 데이터만 구분이 가능합니다. ❷ y축에 투영한 결과물은 10개 데이터 중 절반 가까이가 겹쳐 5개밖에 보이지 않네요. 반면 ❸ 대각선 기준으로 투영한 데이터는 일부 데이터가 굉장히 근접하기는 하지만 10개 데이터 모두가 중복 없이 보입니다. 기존의 2차원 데이터에서는 x에 대한 분포와 y에 대한 분포가 있는데, 이 두 가지 차원의 분포를 최대한 손실 없게끔 1차원 선 위에 녹여내는 겁니다. 바로 이런 대각선에 투영된 결과 같은 차원 축소를 PCA가 수행합니다.

그럼 위와 같이 다양한 각도에서 투영해 최적의 대각선을 찾아냈다고 가정합시다. PCA는 그다음으로 해당 대각선에 대하여 직교하는 또다른 선을 긋고 다시 한번 데이터를 투영시킵니다.



위 그림에서 ❶번 선이 데이터의 분산을 가장 잘 보여주는 선입니다. 첫 번째 주성분인 PC1이 됩니다. 초록색 선은 주황색 선에 직교하는 선입니다. 이렇게 직교하는 선을 그은 후 ❷번 선에 대해서도 데이터를 투영시킵니다. 이 선이 두 번째 주성분, 즉 PC2가 됩니다. 이렇게 직교시킨 선을 이용하는 이유는, 첫 번째 주성분이 담아내지 못한 특성을 최대한 담아낼 수 있는 방향이기 때문입니다(❷번 선에서 10도만 각도를 비틀어 데이터를 투영한다면 ❶번 선에 투영된 데이터의 분산과 크게 다르지 않은 결과가 될 겁니다). 여기서 ❶번 선과 ❷ 선은 벡터의 개념이며, 이 벡터의 방향과 크기를 통하여 데이터의 분포가 어떤 형태인지 효과적으로 파악할 수 있는 겁니다.

여기에 아주 중요한 포인트가 있습니다. 위 예시에서는 변수가 애초에 2가지(x와 y)밖에 없는 데이터였습니다. 그런데 방금 주성분 두 가지를 언급했습니다. 차원을 축소한다고 했는데 2개 변수에서 2개 주성분을 뽑아낸다고 하니 아이러니할 겁니다. 사실 PCA는 변수 개수만큼 새로운 주성분을 만들어냅니다. PCA는 변수 개수만큼 새로운 주성분을 만들되, 첫 번째 주성분에서 데이터 분산을 가급적 크게 만들고, 두 번째 주성분에서는 그다음으로 큰 분산을, 결국 마지막 주성분에서는 분산의 크기가 가장 미미한 식으로 만듭니다. 우리는 이렇게 만든 주성분 중 일부를 취해서 기존의 분산을 어느 정도 유지하면서 변수 개수는 줄이는 효과를 누릴 수 있습니다.

모델링 결과를 비교하면서 4천 개가 넘는 변수가 있는 데이터에서, 우리는 PCA를 통해 400개의 주성분을 취했습니다. 이 과정에서 PCA는 기존 변수 개수(4천 개 이상)만큼을 새 변수를 만들고, 우리는 400개를 취해서 기존 데이터 정보의 80% 정도를 유지하게 되었습니다.

## 학습 마무리

#### 되짚어보기

13.1

1. 시각화를 위하여 차원 축소를 하는 문제입니다. 고객 데이터를 사용합니다.
2. label을 제외하면 변수가 총 12개 있는데, 이를 2차원 형태의 그래프로 나타낼 목적으로 변수 2개로 차원 축소를 진행했습니다.
3. 산점도를 그려 데이터가 얼마나 잘 분류되었는지 확인했습니다.

13.2

1. PCA를 통해 속도와 예측력을 향상시켜보는 문제입니다. 익명의 데이터를 사용합니다.
2. 변수가 4000개가 넘고 변수 이름은 익명 처리되어 있습니다.
3. 랜덤 포레스트로 예측 모델을 만들었더니 109초가 걸리고, AUC는 약 0.84입니다. 반면 PCA로 변수를 400개로 축소하니 소요 시간은 67초로 줄고, AUC는 0.99 수준으로 높아졌습니다.



#### 과제

5장에서 다중공선성을 제거하는 데 SibSp와 Parch 변수에 대한 피처 엔지니어링을 진행했습니다. PCA를 사용해서도 다중공선성을 제거할 수 있으니 5장 타이타닉 데이터셋에 PCA를 적용하여 예측 모델을 만들어봅시다.

#### 유의할 점 관련 모델

1. **재귀적 특성 제거**Recursive Feature Elimination, RFE  
   패키지: from sklearn.feature\_selection import RFE  
   PCA가 새로운 피쳐들을 생성feature extraction하면서 차원을 축소한 반면, 이 알고리즘은 중요한 피쳐들을 남기는 피쳐 셀렉션feature selection 방식으로 차원을 축소합니다.

#### 핵심 용어

1. **PCA** : 변수의 특성을 최대한 유지하면서 그 수를 줄이는 차원 축소 방법입니다. 시각화를 위한 용도로 사용하기도 하고, 모델링 시간 단축이나 오버피팅 방지의 목적으로 사용할 수 있습니다. 차원 축소에는 변수 선택과 변수 추출이 있습니다. 변수 선택은 특정 변수만을 선택함으로써 데이터의 차원을 줄이는 것이고, 변수 추출은 기존의 변수들을 이용하여 새로운 변수를 생성하는 겁니다. PCA는 새로운 변수를 만드는 것이므로 변수 추출에 해당합니다.
2. **차원 축소** : 변수 2개면 2차원 그래프로, 세 개면 3차원 그래프로 나타낼 수 있습니다. 즉, 데이터의 차원은 변수의 개수와 직결되는 겁니다. 차원 축소는 변수의 수를 줄여 데이터의 차원을 축소합니다.

#### 새로운 함수와 라이브러리

* **pca모델.explained\_variance\_ratio\_()** : 학습된 pca 모델에서, 각 주성분이 기존 변수의 분산을 얼마나 대변하는지 확인합니다.

#### 전체 코드

## 연습문제

1. 다음 PCA의 장단점 중 옳지 않은 것은?

① 시각화에 유리하다.

② 변수 간의 상관관계가 있는 경우, 이를 해결할 수 있다.

③ 해석에 어려움이 있다.

④ 일부의 주성분만 선택해도 정보 손실은 전혀 없다.

2. 다음 중 PCA가 필요한 경우가 아닌 것은?

① 변수가 너무 많아서 시각화에 어려움이 있는 경우

② 변수가 너무 많아서 오버피팅이 우려되는 경우

③ 변수가 너무 많아서 모델링 시간이 오래 걸리는 경우

④ 변수가 너무 많아서 해석에 어려움이 있는 경우

3. 변수가 30개 있는 데이터셋이 있다. 여기에 PCA를 사용하여 얻을 수 있는 최대 주성분 수는?

① 3개

② 10개

③ 15개

④ 30개

#### 정답 및 해설

1. 4

④ 일부의 주성분만 선택해도 정보 손실은 전혀 없다. ← 일부 주성분을 선택하면 그만큼 정보손실이 따릅니다.

2. 4

④ 변수가 너무 많아서 해석에 어려움이 있는 경우 ← PCA는 오히려 해석을 어렵게 하지, 해석을 도울 수는 없습니다.

3. 4

④ 30개 ← 실질적으로는 소수 일부의 주성분만을 사용하지만, 뽑아낼 수 있는 최대 주성분 수는 기존 변수의 수와 같습니다.