비타민 8기 3조

곽아람, 김보현, 김진호, 홍성현

차원축소(Dimension Reduction)

CONTENTS

차원의 저주

(Curse of dimensionality)

문제점

- 수학적 공간 차원(Feature Space)이 늘어나면서, 문제 계산법이 지수적으로 커지는 상황
- 차원이 높아질수록 데이터 사이의 거리가 멀어지고, 빈공간(밀도 감소)이 증가하는 공간의 성김 현상(Sparsity)을 보임 그에 따라서 과적합 확률이 올라간다.

1차원 2차원 3차원

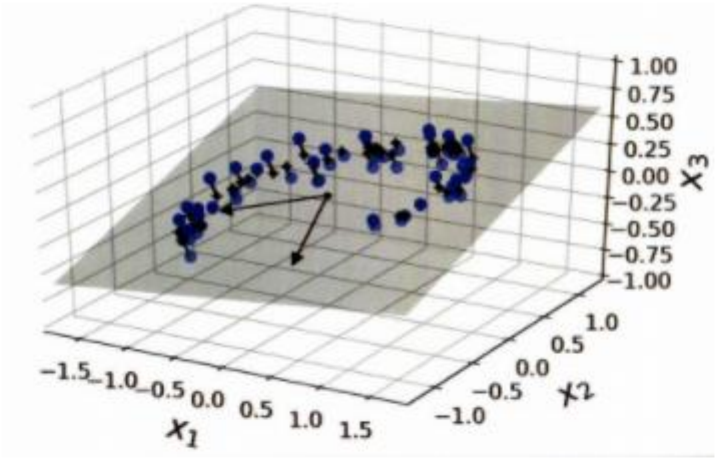
차원의 저주(Curse of dimensionality)

해결방법

방법 1
데이터 추가

빈 공간을 채우기 위해서 밀도가 높아질 때까지 데이터를 추가한다.
하지만, 데이터셋 자체가 너무 고차원 이면 필요 데이터 수를 만족할 수 없다.

방법 2
차원 축소



필요 데이터 수를 줄이기 위해서 Feature의 수를 줄여준다.

차원축소

우리는 일반적으로 기계학습 모델을 만들 때 매니폴드 가정을 하고 만든다.
즉, 우리의 데이터셋이 n차원 공간이 아니라 실제로는 n차원보다 적은 부분 공간에 놓여있을 것이다.

장점

훈련 속도 증가

메모리 절약

데이터 시각화

단점

일부 정보 손실

해석 불가능

PCA

(Principal component **analysis)**

차원의 저주
PCA
LDA
QDA
실습

정의

PCA 차원축소 방법은 고차원 데이터셋을 저차원으로 투영(Projection)하는 방식으로 투영시 조건은 분산이 최대로 보존되는 축을 선택하는 것
->정보가 가장 적게 손실되므로 합리적(원본 데이터셋과 투영된 것 사이의 평균 제곱 거리를 최소화)

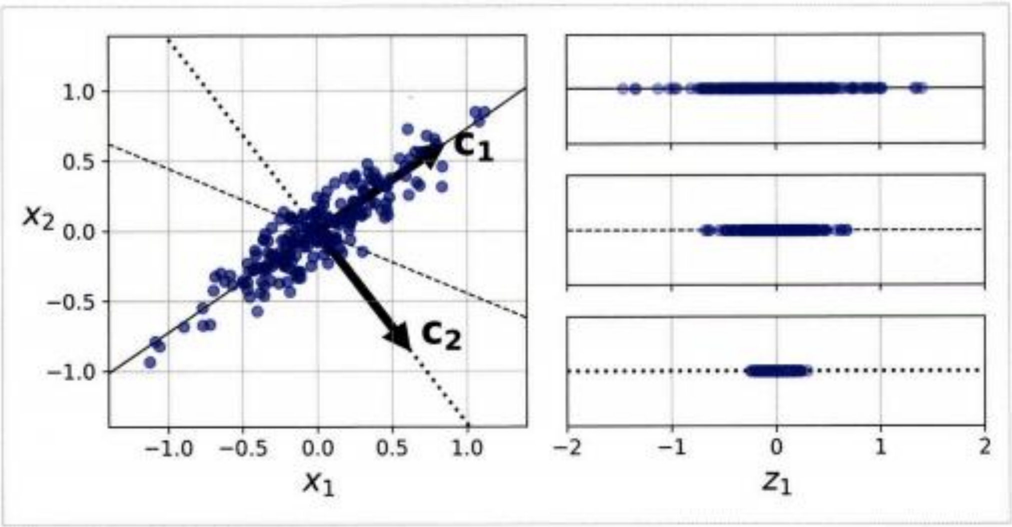


그림 8-7 투영할 부분 공간 선택하기

출처 : 핸즈온 머신러닝

방법

만약 데이터셋을 d차원 공간으로 축소하고 싶을 경우 SVD를 통하여 얻은 주성분 행렬 V에서 d열로 구성된 Wd를 사용하면 된다.

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X}\mathbf{W}_d$$

SVD는 표준행렬 분해 기술로 데이터셋 X를 X = UDV_T로 표현할 수 있다.
X(mxm)일 때
U = mxm , left-singular vectors,
D = mxn, 열벡터는 singular values
V = nxn, 열벡터는 right-singular vectors

출처 : 핸즈온 머신러닝

PCA (Principal component analysis)

방법

넘파이의 SVD 함수를 이용하여 데이터셋을 분해 후 차원수 만큼 V_t열을 가져와서 데이터셋과 계산하여 축소된 데이터셋을 얻는다.

```
X_centered = X - X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_centered)
c1 = Vt.T[:, 0]
c2 = Vt.T[:, 1]

W3 = Vt.T[:, :3]
X3D = X_centered.dot(W3)
```

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X}\mathbf{W}_d$$

3개의 차원으로 축소

	1_?	2_?	3_?
0	-2.684126	0.319397	-0.027915
1	-2.714142	-0.177001	-0.210464
2	-2.888991	-0.144949	0.017900
3	-2.745343	-0.318299	0.031559
4	-2.728717	0.326755	0.090079

출처 : 핸즈온 머신러닝

차원의 저주
PCA
LDA
QDA
실습

실습

Iris 데이터셋은 총 4개의 feature로 이루어져 있어서 시각화할 수 없다.
하지만, PCA 통하여 3개로 줄인다면 시각화가 가능하다.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
import pandas as pd
iris = datasets.load_iris()
X = iris.data
y = iris.target

pd.DataFrame(X, columns=iris.feature_names)
```

4개의 feature

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

차원의 저주
PCA
LDA
QDA
실습

실습

Sklearn에서 PCA를 제공하기 때문에 따로 SVD를 구하여 PCA를 사용할 필요는 없다.
N_components는 우리가 줄일 차원의 수 이다.

```
from sklearn.decomposition import PCA

X_reduced = PCA(n_components=3).fit_transform(X)
pd.DataFrame(X_reduced, columns=["1_?", "2_?", "3_?"])
```

특성 수가 아니라 누적 분산 비율을 직접 지정할 수 있다.

```
X_95 = PCA(n_components=0.95).fit_transform(X)
```

	1_?	2_?	3_?
0	-2.684126	0.319397	-0.027915
1	-2.714142	-0.177001	-0.210464
2	-2.888991	-0.144949	0.017900
3	-2.745343	-0.318299	0.031559
4	-2.728717	0.326755	0.090079

차원 축소로 줄어든 특성은 인간이 해석 불가능 하다.

실습

PCA는 차원축소 후 다시 원래 차원으로 돌아올 수 있다. 하지만, 축소 과정에서 잃은 분산은 복구할 수 없다.

```
pca = PCA(n_components=3)
X_reduced = pca.fit_transform(X)
X4D = pca.inverse_transform(X_reduced)
pd.DataFrame(X4D, columns=iris.feature_names)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.099286	3.500723	1.401086	0.198295
1	4.868758	3.031661	1.447517	0.125368
2	4.693700	3.206384	1.309582	0.184951
3	4.623843	3.075837	1.463736	0.256958
4	5.019326	3.580414	1.370606	0.246168

처음 특성과 비교하여 어느 정도 오차가 있는 것을 확인할 수 있다.

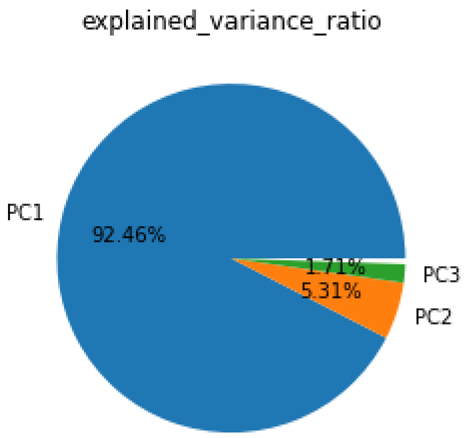
실습

PCA 후 주성분과 각 주성분이 표현하는 분산을 확인할 수 있다.

```
# 주성분
print("주성분 : \n", pca.components_)

# 표현 분산
print("표현 분산 : \n", pca.explained_variance_ratio_)
```

```
주성분 :
 [[ 0.36138659 -0.08452251  0.85667061  0.3582892 ]
 [ 0.65658877  0.73016143 -0.17337266 -0.07548102]
 [-0.58202985  0.59791083  0.07623608  0.5458314 3]]
표현 분산 :
 [0.92461872 0.05306648 0.01710261]
```

```
ratio = pca.explained_variance_ratio_

df_v = pd.DataFrame(ratio, index=['PC1','PC2','PC3'], columns=['V_ration'])
plt.pie(df_v['V_ration'], labels=df_v.index, autopct="%.2f%%")
plt.title("explained_variance_ratio")
plt.show()
```

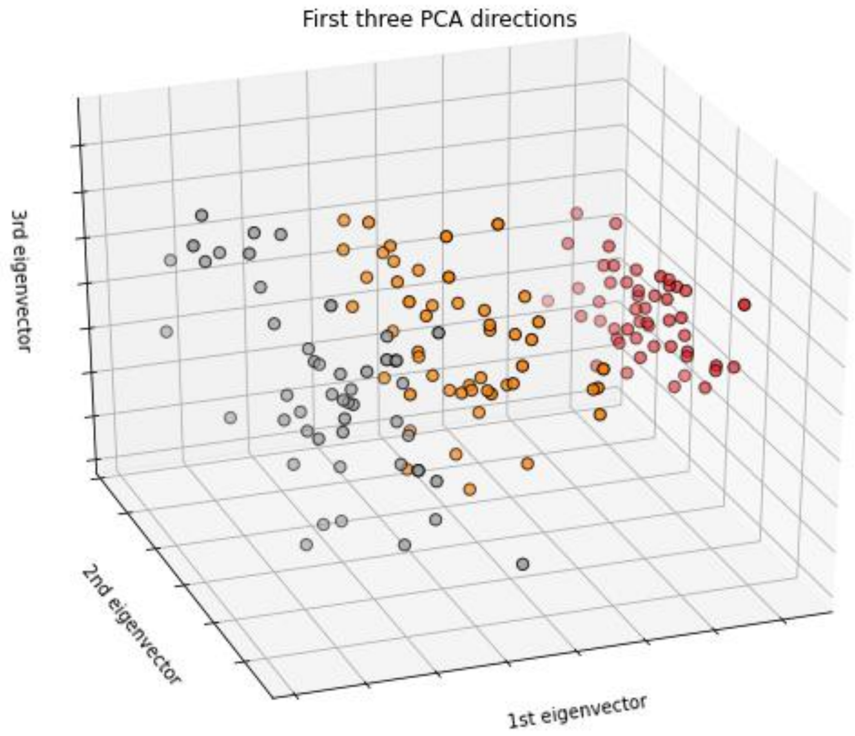


차원의 저주
PCA
LDA
QDA
실습

실습

줄어든 차원으로 iris데이터 시각화 하기

```
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(
    X_reduced[:, 0],
    X_reduced[:, 1],
    X_reduced[:, 2],
    c=y,
    cmap=plt.cm.Set1,
    edgecolor="k",
    s=40,
)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])
plt.show()
```



종류

랜덤 PCA $O(m \times d^2) + O(d^3)$

- d개의 주성분에 대한 근삿값을 빠르게 찾음
- 차원을 많이 줄일 경우 SVD보다 훨씬 빠름

```
rnd_pca = PCA(n_components=3, svd_solver="randomized", random_state=42)
X_reduced_rnd = rnd_pca.fit_transform(X)
```

점진적 PCA

- 훈련 데이터가 큰 경우 효과적
- 미니배치를 이용하여 한 번에 하나씩 학습
- 온라인 학습 시 사용
- 일반 PCA보다 느리다.

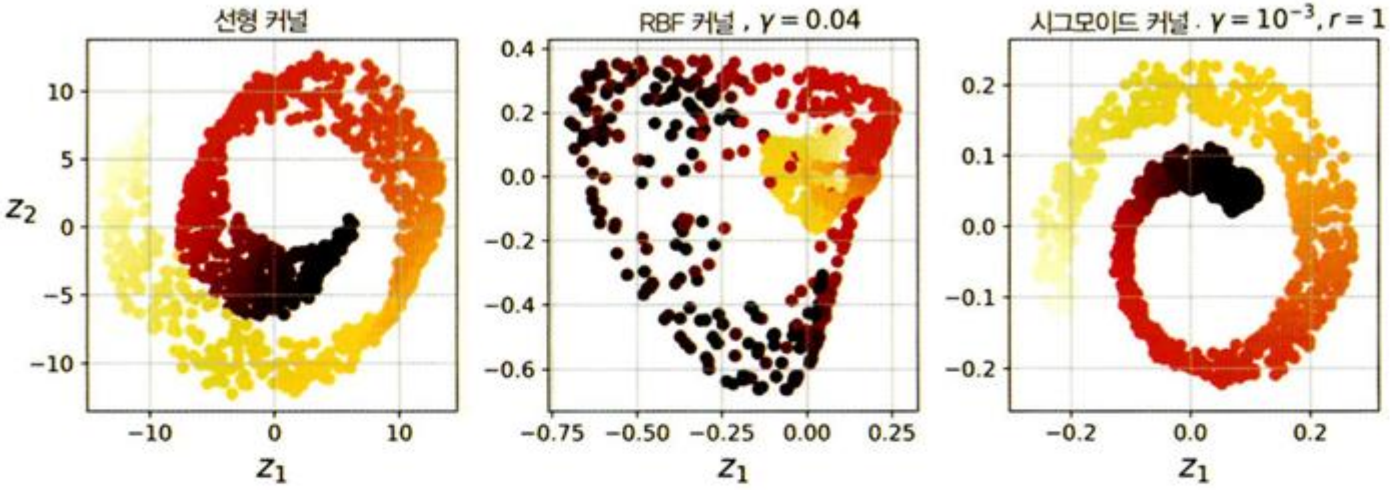
```
from sklearn.decomposition import IncrementalPCA
import numpy as np

n_batches = 10
inc_pca = IncrementalPCA(n_components=3)
for X_batch in np.array_split(X, n_batches):
    inc_pca.partial_fit(X_batch)

X_reduced_inc = inc_pca.transform(X)
```

커널 PCA(kPCA)

일반 PCA로 구분되면 아주 좋지만, 일반적으로 선형 PCA로 구분되지 않는 경우가 많다. 이럴 경우 모델 자체의 복잡도를 올려서 비선형적인 데이터를 구분할 수 있지만, 이럴 경우 cost 및 과적합 확률이 올라간다. 따라서, 특정 mapping function을 이용하여 고차원으로 데이터셋을 보낸 뒤 선형적으로 구분시키는 것이 kPCA이다.

대표적인 함수

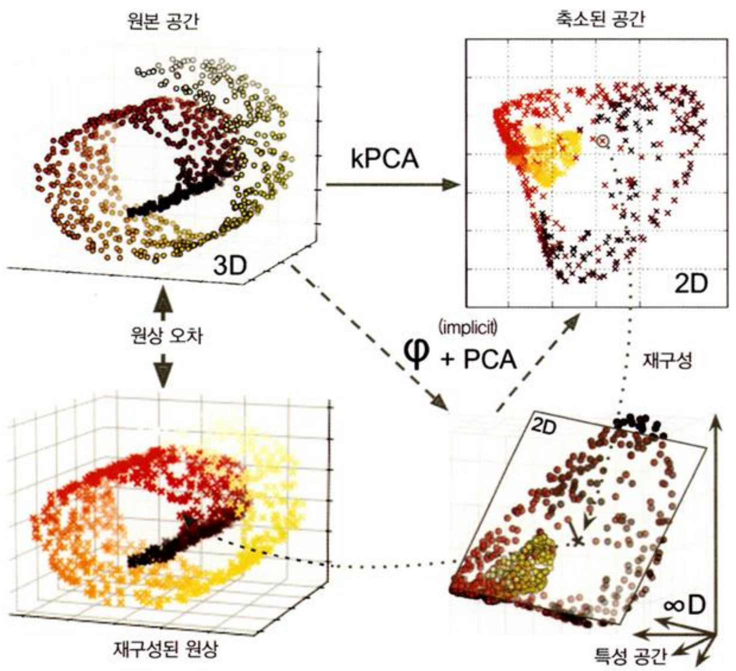
- rbf
- poly
- sigmoid



출처 : 핸즈온 머신러닝

커널 PCA(kPCA)

일반 PCA와 다르게 kPCA는 재구성하는 것이 쉽지 않다.
따라서, 특성 맵을 사용하여 고차원으로 보낸 뒤 일반 PCA를 사용하여 줄인 공간이 kPCA와 수학적으로 같다는 사실을 이용해야 한다.
N차원 공간에서 다시 PCA를 사용하여 재구성된 원상을 만들 수 있다. 그리고 이 원상과 원본 공간을 비교하여 재구성 오차를 비교할 수 있다.

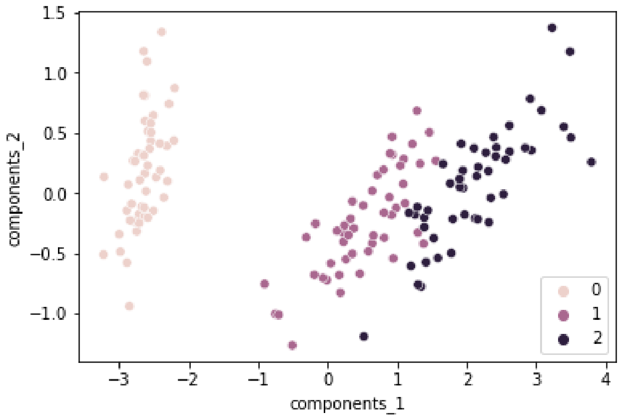


출처 : 핸즈온 머신러닝

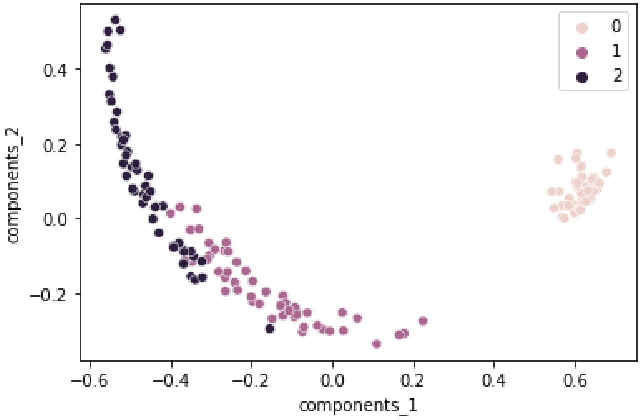
실습

```
from sklearn.decomposition import KernelPCA

lin_pca = KernelPCA(n_components=2, kernel="linear", fit_inverse_transform=True)
rbf_pca = KernelPCA(n_components=2, kernel="rbf", gamma=0.0433, fit_inverse_transform=True)
sig_pca = KernelPCA(n_components=2, kernel="sigmoid", gamma=0.001, coef0=1, fit_inverse_transform=True)

X_reduced_lin = lin_pca.fit_transform(X)
X_reduced_rbf = rbf_pca.fit_transform(X)
X_reduced_sig = sig_pca.fit_transform(X)
```

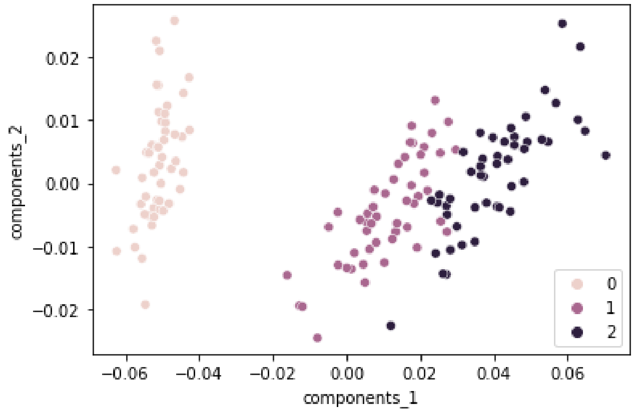
```
import seaborn as sns

sns.scatterplot(x=X_reduced_lin[:, 0], y=X_reduced_lin[:, 1], hue=y)
plt.xlabel("components_1")
plt.ylabel("components_2")
plt.show()
```




```
sns.scatterplot(x=X_reduced_rbf[:, 0], y=X_reduced_rbf[:, 1], hue=y)
plt.xlabel("components_1")
plt.ylabel("components_2")
plt.show()
```




```
sns.scatterplot(x=X_reduced_sig[:, 0], y=X_reduced_sig[:, 1], hue=y)
plt.xlabel("components_1")
plt.ylabel("components_2")
plt.show()
```



차원의 저주
PCA
LDA
QDA
실습

PCA 평가 방법

지도학습 모델과 함께 사용

PCA는 비지도학습 이기 때문에 따로 성능평가를 할 수 없다.
하지만, 지도학습 모델의 전처리로 사용할 경우 파이프라인을 통하여 모델을 학습 후 평가할 수 있다.

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

clf = Pipeline([
        ("kpca", KernelPCA(n_components=2)),
        ("log_reg", LogisticRegression(solver="lbfgs"))
    ])

param_grid = [{
        "kpca__gamma": np.linspace(0.03, 0.05, 10),
        "kpca__kernel": ["rbf", "sigmoid"]
    }]

grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('kpca', KernelPCA(n_components=2)),
                                       ('log_reg', LogisticRegression())]),
             param_grid=[{'kpca__gamma': array([0.03      , 0.03222222, 0.03444444, 0.03666667, 0.03888889,
       0.04111111, 0.04333333, 0.04555556, 0.04777778, 0.05      ]),
                          'kpca__kernel': ['rbf', 'sigmoid']}])
```

재구성 오차

지도학습 모델의 전처리로 사용하지 않고 단독적으로 사용할 경우 축소된 차원에서 재구성시킨 뒤 원본 데이터와 재구성 데이터 사이의 평균제곱거리를 지표로 사용할 수 있다.

```
rbf_pca = KernelPCA(n_components=2, kernel="rbf", gamma=0.0433,
                    fit_inverse_transform=True)
X_reduced = rbf_pca.fit_transform(X)
X_preimage = rbf_pca.inverse_transform(X_reduced)

from sklearn.metrics import mean_squared_error

mean_squared_error(X, X_preimage)
```

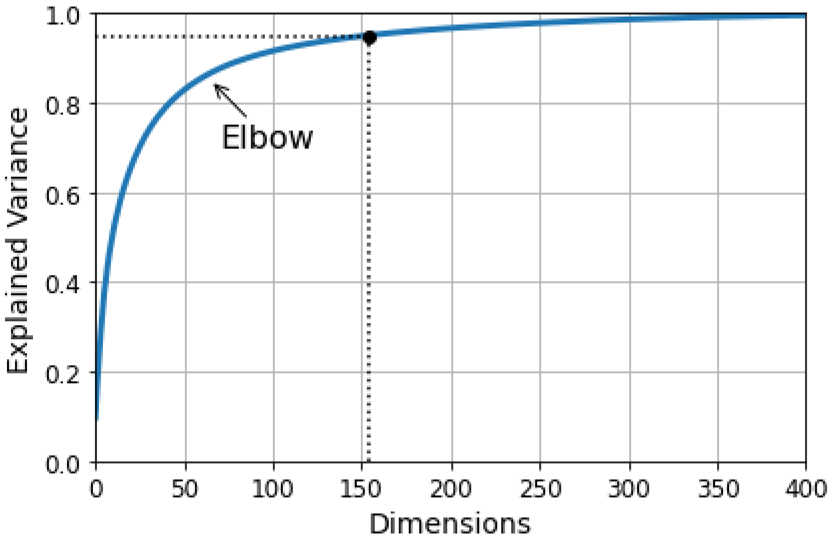
```
0.1588300630170411
```

특징

PCA에서 줄일 차원수는 직접 정해줘야 하는 하이퍼파라미터에 속한다.
적절한 차원수는 보존하고자 하는 누적 분산 비율을 보고 결정해야 한다.

```
pca = PCA()
pca.fit(X)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
d
```

2



그래프를 그려서 급격하게 누적분산비율 증가가 감소하는 구간을 Elbow하고 한다.

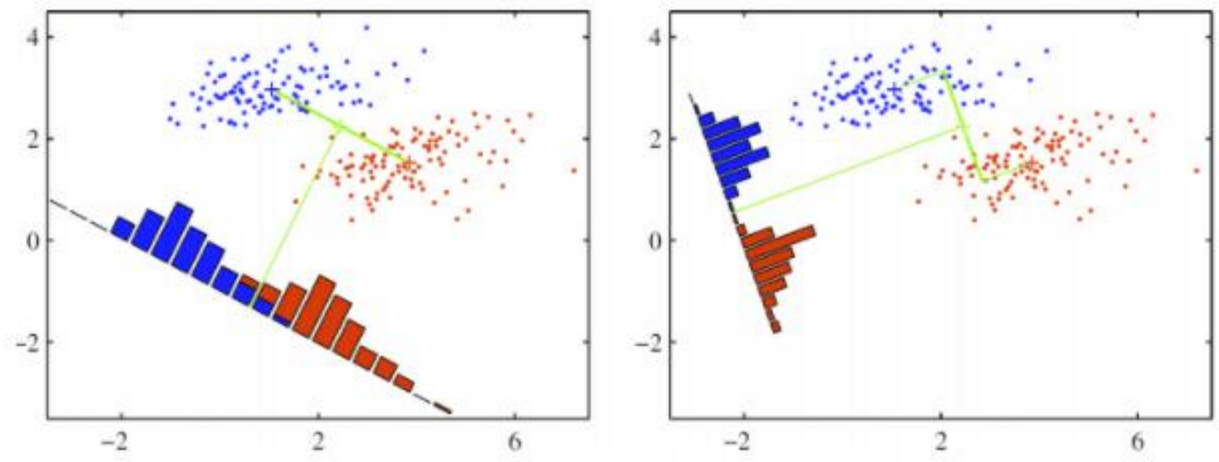
LDA

(Linear Discriminant **analysis)**

▤ 차원의 저주
▤ PCA
▤ LDA
▤ QDA
▤ 실습

정의

LDA는 원래 선형판별분석기 이지만, 분류 축을 찾는 방법이 PCA와 같은 투영 방식을 사용합니다.
이 둘의 차이점은 PCA는 목적이 분산을 가장 잘 보존하는 공간으로 투영시키는 것이지만, LDA는 label간의 거리를 최대한 크게 분리하도록 투영 시킨다.
장점으로는 투영 후 데이터셋 분리가 잘되지만, 정보 손실이 크고, label이 필요하다는 단점이 존재한다.



출처 : ratsgo's

방법

이진 분류 문제일 경우 LDA과정을 보면

$$\sigma^2_{within} = W^T\Sigma_1 W + W^T\Sigma_2 W$$

$$\Sigma_i = \sum_{x \in g_i} W^T(x-\mu_i)(x-\mu_i)^T W$$

Label 내의 분산이고, W는 우리가 찾을 축이다.

$$\sigma^2_{between} = W^T(\mu_1-\mu_2)(\mu_1-\mu_2)^T W$$

Label간의 분산이다.

$$J(W) = \frac{\sigma^2_{between}}{\sigma^2_{within}}$$

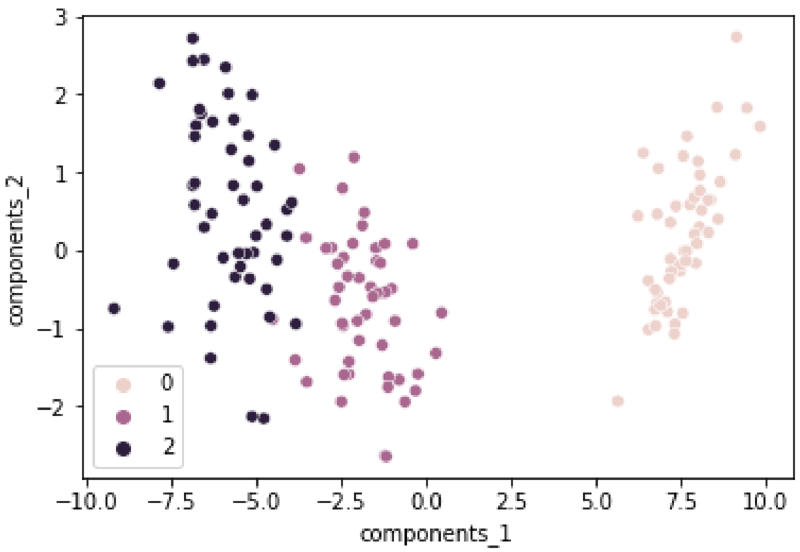
위 두 분산을 엮어서 목적함수를 만들어 준다. 분모는 label 내의 분산으로 커져야 하며, 분자는 label 간의 분산으로 작아져야한다.

출처 : 방향 분선가

실습

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis(n_components=2)
lda.fit(X, y)
X_reduced_lda = lda.transform(X)
```

4개의 차원을 2개로 축소



두개의 components로 시각화

출처 : 방향 분석가

3. 이차판별분석법 QDA
(Quadratic Discriminant Analysis)

QDA 정의 및 이해

- 비모수적 방법인 KNN과 선형의 LDA 및 로지스틱 회귀 사이에서 절충한 방법
- $class\ k$와 관계없는 공통 공분산 구조 시그마에 대한 가정을 버린 것이 QDA

➡ $class$ 범주별로 서로 다른 공분산 구조를 가진 경우에 활용 가능

QDA에 필요한 가정

: 독립변수 X가 실수이고 확률분포가 다변량 정규분포이다.

$$X|Y = k \sim N(\mu_k, \Sigma_k), \qquad k = 1,\cdots,K$$

비교) LDA의 경우 : $X|Y = k \sim N(\mu_k, \Sigma), \qquad k = 1,\cdots,K$

차원의 저주
PCA
LDA
QDA
실습

Quadratic discriminant functions

*class k*에 대한 이차판별함수

π_k : *class k*에 속할 사전 확률
Σ_k : *class k*에 대한 공분산 행렬
μ_k : *class k*에 대한 평균 벡터

X에 대한 2차 형태(Quadratic form)

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x-\mu_k)^T\mathbf{\Sigma}_k^{-1}(x-\mu_k) - \frac{1}{2}\log|\mathbf{\Sigma}_k| + \log\pi_k \\ &= -\frac{1}{2}x^T\mathbf{\Sigma}_k^{-1}x + x^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\log|\mathbf{\Sigma}_k| + \log\pi_k\end{aligned}$$

비교) LDA의 경우 : Linear discriminant functions 선형판별함수

$$\delta_k(x) = x^T\mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}^{-1}\mu_k + \log\pi_k$$

➡ 가장 큰 값을 가지는 class로 분류

Ex) 2개의 class가 있다고 하자 (k=1 ,2)

$\delta_1(x) > \delta_2(x)$이면? Class 1로 할당

LDA와 QDA의 비교(공통점)

LDA	QDA

베이즈 정리를 이용하여 사후확률을 다음과 같이 정의하여 분류를 수행한다.

관측치 x가 주어졌을 때
그것이 k $class$에 속할 확률

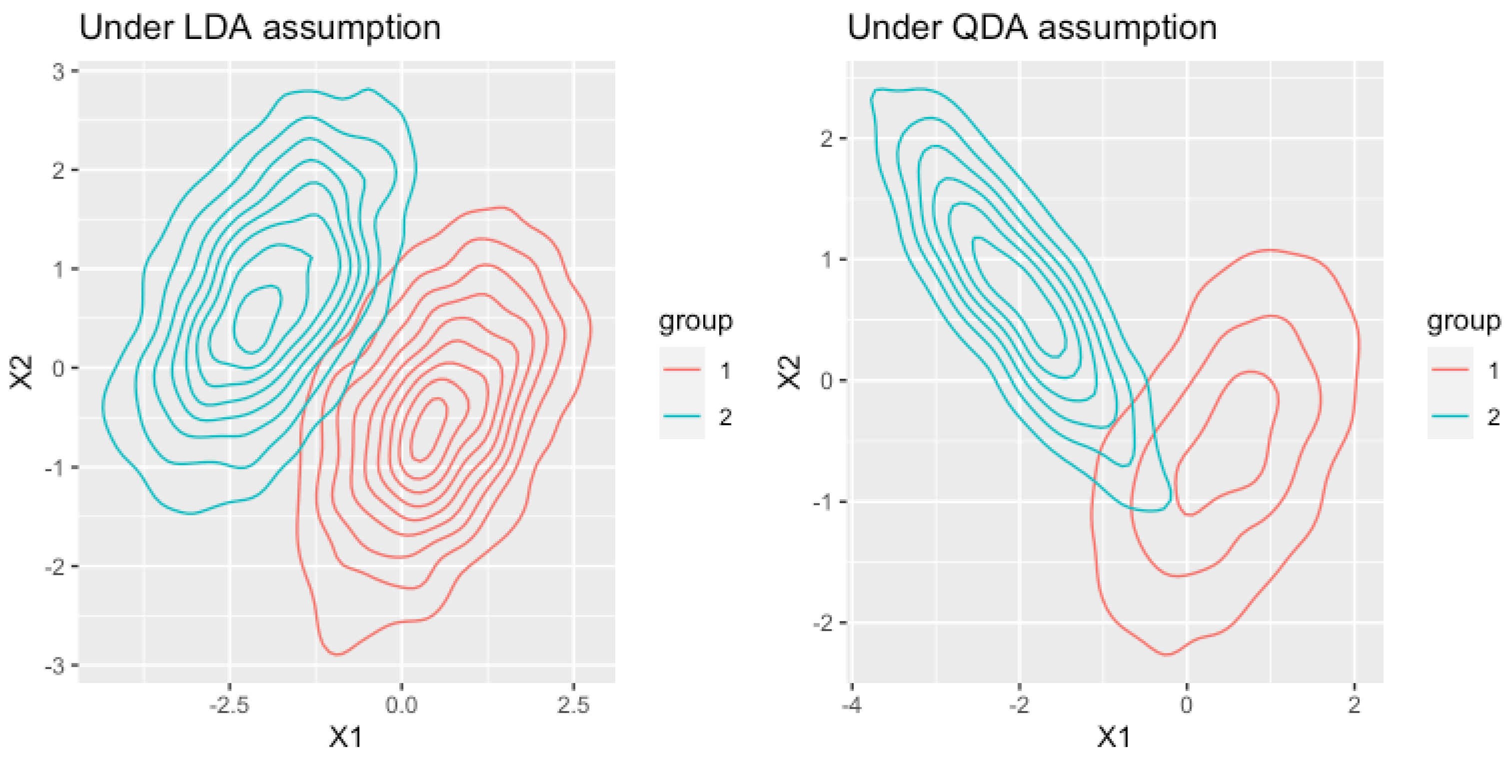
- 베이즈 정리 : $P(B|A) = \frac{P(A|B)P(B)}{P(A)}$
- $P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$

사후확률을 계산하기 위해서는 $f_k(x)$을 알아야 하는데, $f_k(x) = P(X = x|Y = k)$를 추정하기 위해 $class$ k에 속한 관측치 X의 분포가 **다변량 정규분포**임을 가정한다.

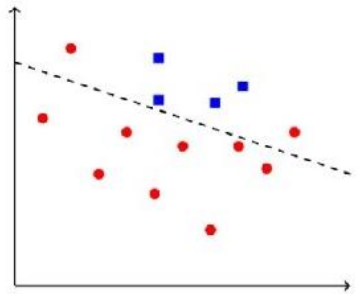
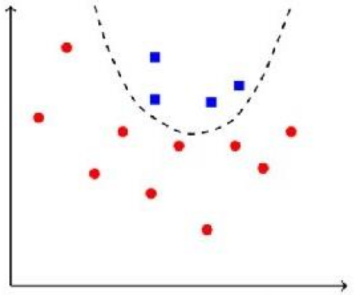
$$f_k(x) = \frac{1}{(2\pi)^{p/2}|\Sigma_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)}$$

LDA와 QDA의 비교(차이점)

LDA	QDA
Class 별로 공분산 구조가 같음을 가정한다. $\Sigma_1 = \Sigma_2 = \cdots = \Sigma_K = \Sigma$	Class 별로 다른 공분산 구조를 갖는다. $\Sigma_1 \neq \Sigma_2 \neq \cdots \neq \Sigma_K$

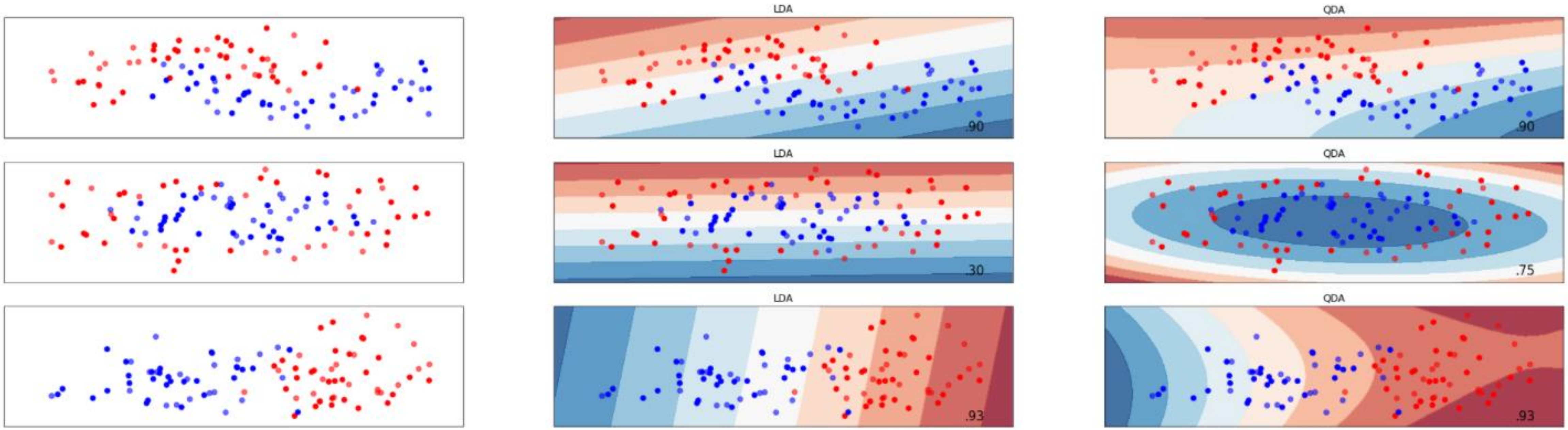


LDA와 QDA의 비교(차이점)

LDA	QDA
추정해야 할 모수의 개수가 상대적으로 적다. ▶ $\frac{p(p+1)}{2} + Kp + K$ 개	추정해야 할 모수의 개수가 상대적으로 많다. ▶ $\frac{Kp(p+1)}{2} + Kp + K$ 개
QDA보다 덜 유연한(less flexible) 분류기이므로 상당히 작은 분산을 가진다.	안정적인 모수를 갖기 위해서는 많은 관측치를 필요로 한다.
 (a) Separating with LDA	 (b) Separating with QDA

- 훈련 관측치의 수가 비교적 작아 분산을 줄이는 것이 중요하다면 LDA가 QDA보다 나을 수 있다.
- 반대로, 훈련 관측치의 수가 아주 커서 분류기의 분산이 주요 우려사항이 아니거나 K개의 클래스들이 공통의 공분산행렬을 갖는다는 가정이 명백히 맞지 않으면 QDA를 사용하는 것이 권장된다.

LDA vs QDA



데이터셋 산점도 | LDA의 Decision Boundary | QDA의 Decision Boundary

2번의 QDA가 LDA보다 현저하게 높은 분류결과를 가져온다는 것을 제외하고는 LDA와 QDA중 어떤 방법이 더 낫다고 단정짓지 못한다.

➡ **데이터의 분포 상태에 따라 가장 적합한 분류기가 달라질 수 있다.**
따라서 다른 분석과 마찬가지로 데이터의 사전 검토작업이 꼭 필요하다!

QDA 간단 실습

QDA 간단 실습

1. 필요 라이브러리 import

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
```

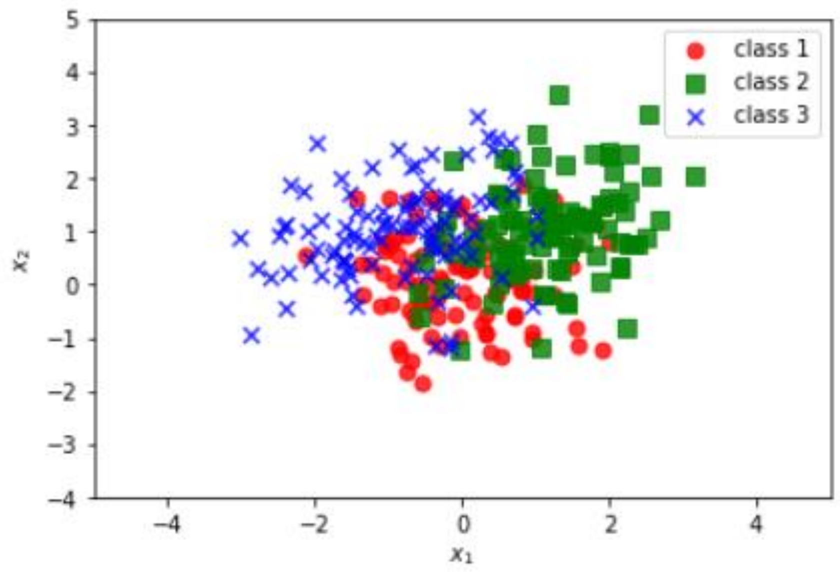
SciPy의 stats 서브패키지에 있는 다변량 정규 분포를 위한 multivariate_normal 클래스 이용

2. 데이터 생성 및 시각화

```
# 데이터 생성
N = 100
rv1 = stats.multivariate_normal([ 0, 0], [[0.7, 0.0], [0.0, 0.7]])
rv2 = stats.multivariate_normal([ 1, 1], [[0.8, 0.2], [0.2, 0.8]])
rv3 = stats.multivariate_normal([-1, 1], [[0.8, 0.2], [0.2, 0.8]])
np.random.seed(0)
X1 = rv1.rvs(N)
X2 = rv2.rvs(N)
X3 = rv3.rvs(N)
y1 = np.zeros(N)
y2 = np.ones(N)
y3 = 2 * np.ones(N)
X = np.vstack([X1, X2, X3])
y = np.hstack([y1, y2, y3])

# 산점도 그리기
plt.scatter(X1[:, 0], X1[:, 1], alpha=0.8, s=50, marker="o", color='r', label="class 1")
plt.scatter(X2[:, 0], X2[:, 1], alpha=0.8, s=50, marker="s", color='g', label="class 2")
plt.scatter(X3[:, 0], X3[:, 1], alpha=0.8, s=50, marker="x", color='b', label="class 3")
plt.xlim(-5, 5)
plt.ylim(-4, 5)
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.legend()
plt.show()
```

평균벡터 μ_k

공분산행렬 Σ_k



차원의 저주
PCA
LDA
QDA
실습

3. QDA 적합

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

qda = QuadraticDiscriminantAnalysis(store_covariance=True).fit(X, y)
```

4. 적합 결과

```
# 각 클래스 k의 사전확률
qda.priors_
```

```
array([0.33333333, 0.33333333, 0.33333333])
```

```
# 각 클래스 k에서 x의 기댓값 벡터 μk의 추정치 벡터
qda.means_
```

```
array([[-8.01254084e-04,  1.19457204e-01],
       [ 1.16303727e+00,  1.03930605e+00],
       [-8.64060404e-01,  1.02295794e+00]])
```

```
# 각 클래스 k에서 x의 공분산 행렬 Σk의 추정치 행렬. (생성자 인수 store_covariance 값이 True인 경우에만 제공)
qda.covariance_[0]
```

```
array([[ 0.73846319, -0.01762041],
       [-0.01762041,  0.72961278]])
```

```
qda.covariance_[1]
```

```
array([[0.66534246, 0.21132313],
       [0.21132313, 0.78806006]])
```

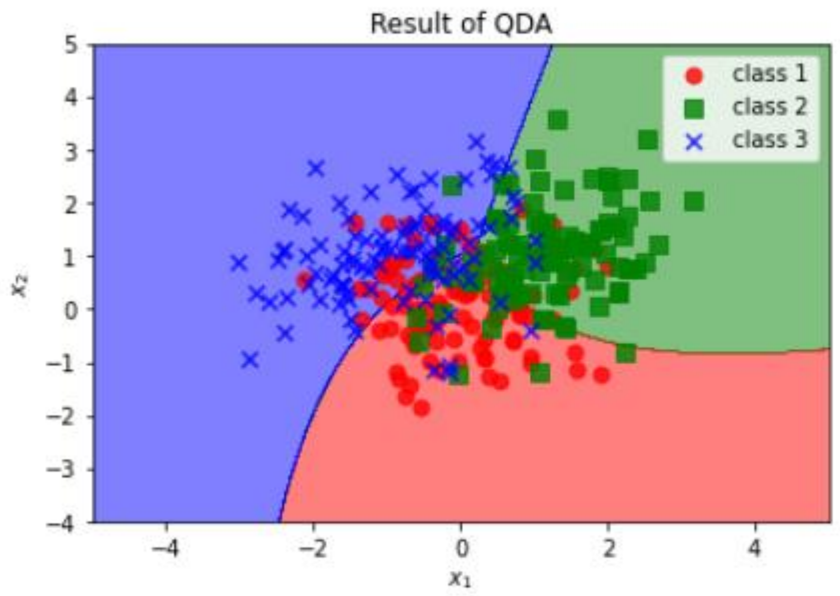
```
qda.covariance_[2]
```

```
array([[0.9351386 , 0.22880955],
       [0.22880955, 0.79142383]])
```

차원의 저주
PCA
LDA
QDA
실습

5. QDA Decision Boundary 시각화

```
import matplotlib as mpl
import seaborn as sns

x1min, x1max = -5, 5
x2min, x2max = -4, 5
XX1, XX2 = np.meshgrid(np.arange(x1min, x1max, (x1max-x1min)/1000),
                       np.arange(x2min, x2max, (x2max-x2min)/1000))
YY = np.reshape(qda.predict(np.array([XX1.ravel(), XX2.ravel()]).T), XX1.shape)
cmap = mpl.colors.ListedColormap(sns.color_palette(["r", "g", "b"]).as_hex())
plt.contourf(XX1, XX2, YY, cmap=cmap, alpha=0.5)
plt.scatter(X1[:, 0], X1[:, 1], alpha=0.8, s=50, marker="o", color='r', label="class 1")
plt.scatter(X2[:, 0], X2[:, 1], alpha=0.8, s=50, marker="s", color='g', label="class 2")
plt.scatter(X3[:, 0], X3[:, 1], alpha=0.8, s=50, marker="x", color='b', label="class 3")
plt.xlim(x1min, x1max)
plt.ylim(x2min, x2max)
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.title("Result of QDA")
plt.legend()
plt.show()
```



실습
(LDA & QDA)

1. 필요한 라이브러리 및 데이터셋 로드

```
!pip install pydataset
```

```
import pandas as pd
from pydataset import data
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, accuracy_score, roc_curve, auc
import seaborn as sns
```

"pydataset" 라이브러리에서 가져온 "Wages1" 데이터 사용

변수 설명
- Exper : 경력기간
- Sex : 여성/남성
- School : 교육 받은 기간
- Wage : 시간당 급여

```
df = data('Wages1')
df.head()
```

	exper	sex	school	wage
1	9	female	13	6.315296
2	12	female	12	5.479770
3	11	female	11	3.642170
4	9	female	14	4.593337
5	8	female	14	2.418157

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3294 entries, 1 to 3294
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   exper   3294 non-null   int64
 1   sex     3294 non-null   object
 2   school  3294 non-null   int64
 3   wage    3294 non-null   float64
dtypes: float64(1), int64(2), object(1)
memory usage: 128.7+ KB
```

```
df.describe()
```

	exper	school	wage
count	3294.000000	3294.000000	3294.000000
mean	8.043412	11.630540	5.757585
std	2.290661	1.657545	3.269186
min	1.000000	3.000000	0.076556
25%	7.000000	11.000000	3.621570
50%	8.000000	12.000000	5.205781
75%	9.000000	12.000000	7.304506
max	18.000000	16.000000	39.808917

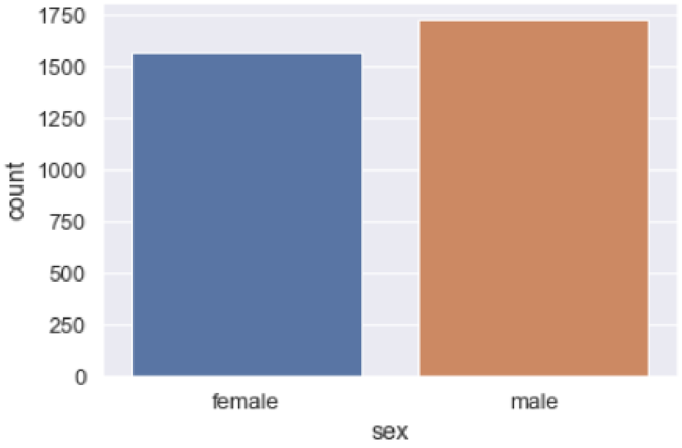
데이터 실습 : 성별 예측하기

2. EDA

범주형 변수

```
sns.countplot(df['sex'])
```

<AxesSubplot:xlabel='sex', ylabel='count'>



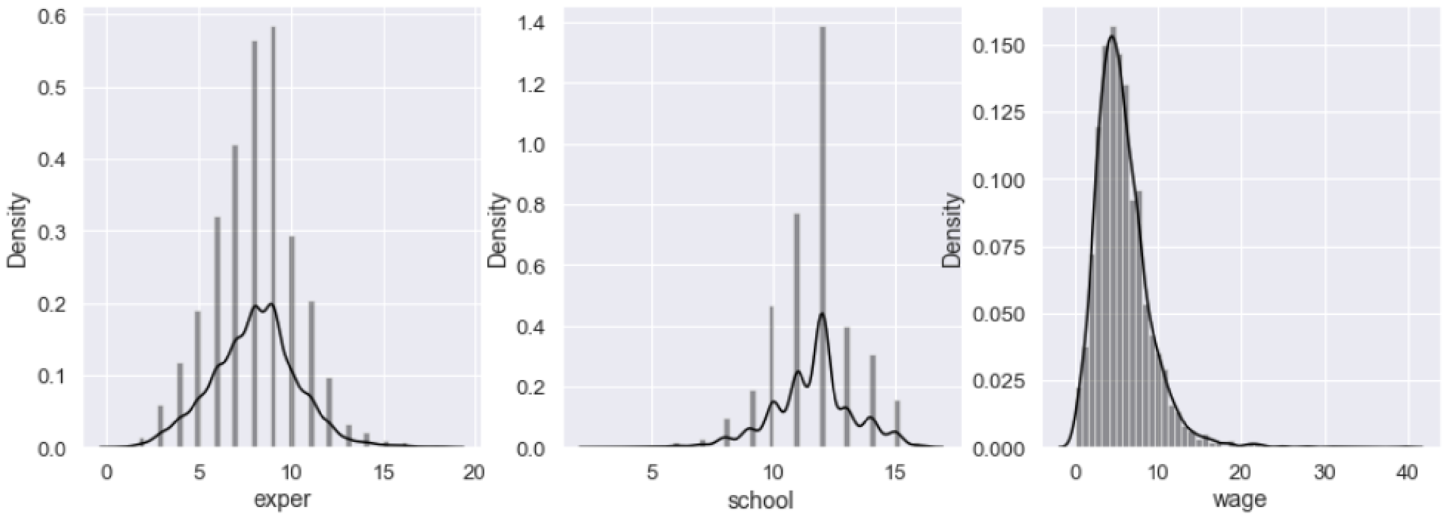
➡ 남녀의 성비는 거의 균형을 이룸

수치형 변수

```
fig = plt.figure()
fig, axs = plt.subplots(figsize=(15, 5),ncols=3)
sns.set(font_scale=1.4)
sns.distplot(df['exper'],color='black',ax=axs[0])
sns.distplot(df['school'],color='black',ax=axs[1])
sns.distplot(df['wage'],color='black',ax=axs[2])
```

<AxesSubplot:xlabel='wage', ylabel='Density'>

<Figure size 432x288 with 0 Axes>



➡ 나머지 변수들도 분포가 종 모양을 띄므로 정규분포에 가깝다고 할 수 있다

데이터 실습 : 성별 예측하기

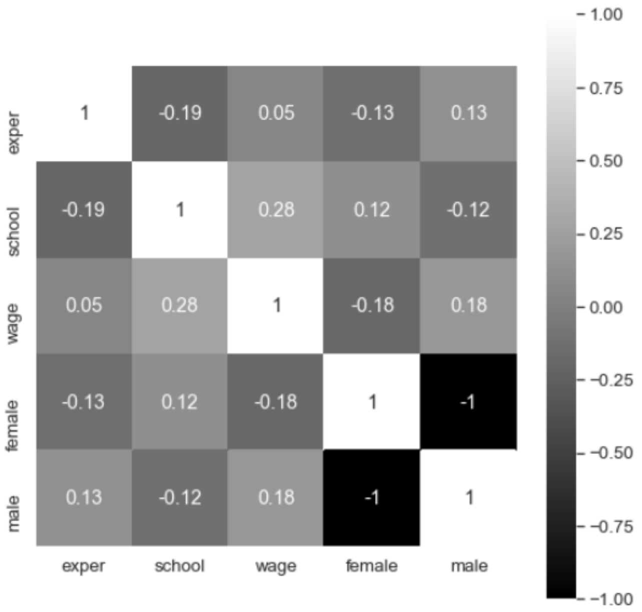
3. 범주형 변수를 더미 변수로 변환하여 결합하기

```
sex = pd.get_dummies(df['sex'])
df.drop(['sex'],axis=1, inplace=True)
df = pd.concat([df, sex],axis=1)
df.head()
```

	exper	school	wage	female	male
1	9	13	6.315296	1	0
2	12	12	5.479770	1	0
3	11	11	3.642170	1	0
4	9	14	4.593337	1	0
5	8	14	2.418157	1	0

```
cormat=df.corr(method='pearson')
f,ax=plt.subplots(figsize=(8,8))
sns.set(font_scale=1.2)
sns.heatmap(round(cormat,2),
vmax=1.,square=True,
cmap="gist_gray",annot=True)
```

<AxesSubplot:>



변수들 간 상관계수 또한 높지 않다.

4. Feature / Target 지정한 후, Train / Test 데이터 나누기

```
X=df[['exper','school','wage']]
y=df['male']

X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=.2, random_state=50)
```

차원의 저주
PCA
LDA
QDA
실습

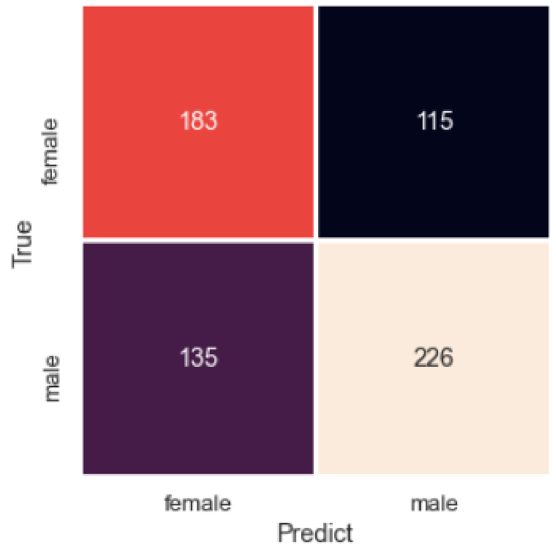
5. LDA 적합

```
lda = LinearDiscriminantAnalysis()
model_lda = lda.fit(X_train,y_train)
y_pred_lda =lda.predict(X_test)
```

6. LDA 평가

```
cm = confusion_matrix(y_test, y_pred_lda)
ax= plt.subplots(figsize=(5,5))
with sns.axes_style('white'):
    sns.heatmap(cm, cbar=False, square=True,annot=True,fmt='g',linewidths=2.5,
                xticklabels={'female','male'}, yticklabels={'female','male'})
plt.ylabel('True'); plt.xlabel('Predict')
```

```
Text(0.5, 19.5, 'Predict')
```




```
round(accuracy_score(y_test, y_pred_lda),4)
```

```
0.6206
```

▶ LDA의 정확도

```
print(classification_report(y_test, y_pred_lda))
```

```
              precision    recall  f1-score   support

           0       0.58      0.61      0.59       298
           1       0.66      0.63      0.64       361

    accuracy                           0.62       659
   macro avg       0.62      0.62      0.62       659
weighted avg       0.62      0.62      0.62       659
```

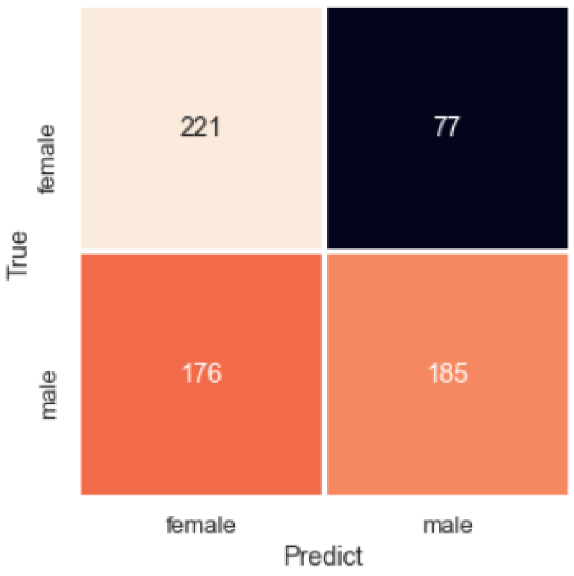
7. QDA 적합

```
qda = QuadraticDiscriminantAnalysis()
model_qda = qda.fit(X_train,y_train)
y_pred_qda = qda.predict(X_test)
```

8. QDA 평가

```
cm = confusion_matrix(y_test, y_pred_qda)
ax= plt.subplots(figsize=(5,5))
with sns.axes_style('white'):
    sns.heatmap(cm, cbar=False, square=True,annot=True,fmt='g',linewidths=2.5,
                xticklabels={'female','male'}, yticklabels={'female','male'})
plt.ylabel('True'); plt.xlabel('Predict')
```

```
Text(0.5, 19.5, 'Predict')
```




```
round(accuracy_score(y_test, y_pred_qda),4)
```

0.6161 ▶ QDA의 정확도

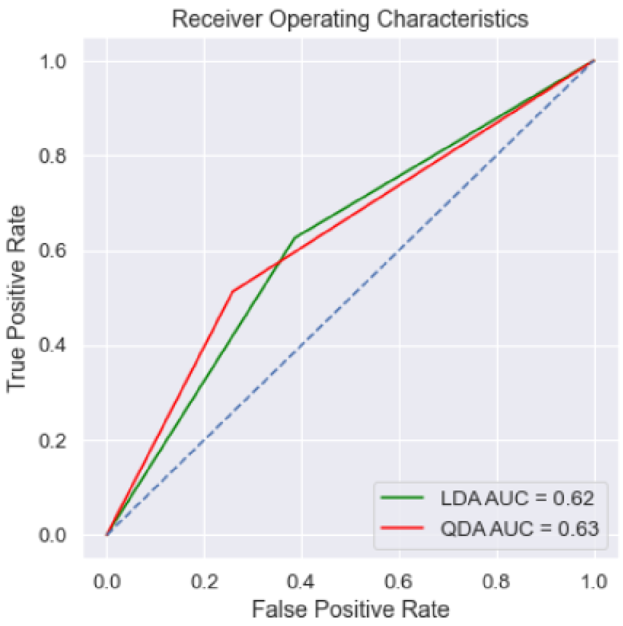
```
print(classification_report(y_test, y_pred_qda))
```

	precision	recall	f1-score	support
0	0.56	0.74	0.64	298
1	0.71	0.51	0.59	361
accuracy			0.62	659
macro avg	0.63	0.63	0.61	659
weighted avg	0.64	0.62	0.61	659

9. ROC curve & AUC

```
false_positive_rate_qda, true_positive_rate_qda, thresholds = roc_curve(y_test, y_pred_qda)
roc_auc_qda = auc(false_positive_rate_qda, true_positive_rate_qda)
false_positive_rate_lda, true_positive_rate_lda, thresholds = roc_curve(y_test, y_pred_lda)
roc_auc_lda = auc(false_positive_rate_lda, true_positive_rate_lda)

plt.figure(figsize=(6, 6))
plt.title('Receiver Operating Characteristics')
plt.plot(false_positive_rate_lda, true_positive_rate_lda,
         color='green', label='LDA AUC = {:.2f}'.format(roc_auc_lda))
plt.plot(false_positive_rate_qda, true_positive_rate_qda,
         color='red', label='QDA AUC = {:.2f}'.format(roc_auc_qda))
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
Text(0.5, 0, 'False Positive Rate')
```



LDA와 QDA의 성능의 차이가 거의 나지 않았지만
QDA가 약간 더 좋게 나왔다

감사합니다 ☺