Q1-2. Manifold Learning

```
import sys

import sklearn

import numpy as np
import os

import matplotlib.pyplot as plt
from matplotlib import gridspec

from sklearn.datasets import make_swiss_roll

# 그래프 결과를 저장한 경로 정의
PROJECT_ROOT_DIR = '.'
CHAPTER_ID = 'dim_reduction'
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, 'images', CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

# 그래프 결과를 저장하는 함수
def save_fig(fig_id, tight_layout=True, fig_extension='png', resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + '.' + fig_extension)
    print('Save Image ', fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

############################################################################################################################################
### Prepare 3D Swiss Roll ##################################################################################################################
############################################################################################################################################

X, t = make_swiss_roll(n_samples=1000, noise=0.2, random_state=42)      # 1000 개의 샘플 데이터로 구성된 3D Swiss Roll 데이터셋 준비

axes = [-11.5, 14, -2, 23, -12, 15]

fig = plt.figure(figsize=(6, 5))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=t, cmap=plt.cm.hot)     # Swiss Roll 을 3D Scatter 그래프로 그림

# 그래프 영역 준비
ax.view_init(10, -70)
ax.set_xlabel("$x_1$", fontsize=18)
ax.set_ylabel("$x_2$", fontsize=18)
ax.set_zlabel("$x_3$", fontsize=18)
ax.set_xlim(axes[0:2])
ax.set_ylim(axes[2:4])
ax.set_zlim(axes[4:6])

save_fig("swiss_roll_plot")
plt.show()

plt.figure(figsize=(11, 4))

plt.subplot(121)
plt.scatter(X[:, 0], X[:, 1], c=t, cmap=plt.cm.hot)         # 3D Swiss Roll 데이터셋을 원본 X1, X2 영역에 투영하여 그림
plt.axis(axes[:4])
plt.xlabel('$x_1$', fontsize=18)
plt.ylabel('$x_2$', fontsize=18, rotation=0)
plt.grid(True)

plt.subplot(122)
plt.scatter(t, X[:, 1], c=t, cmap=plt.cm.hot)               # 3D Swiss Roll 데이터셋을 Manifold 에 맞춰서 투영하여 그림
plt.axis([4, 15, axes[2], axes[3]])
plt.xlabel("$z_1$", fontsize=18)
plt.grid(True)

save_fig("squished_swiss_roll_plot")
plt.show()

############################################################################################################################################
### 01 More Complicated Unrolled Manifold with Simple Decision Boundary #####################################################################
############################################################################################################################################

axes = [-11.5, 14, -2, 23, -12, 15]

x2s = np.linspace(axes[2], axes[3], 10)
x3s = np.linspace(axes[4], axes[5], 10)
x2, x3 = np.meshgrid(x2s, x3s)

fig = plt.figure(figsize=(6, 5))
ax = plt.subplot(111, projection='3d')

positive_class = X[:, 0] > 5        # 데이터셋의 원본 Feature 를 기반으로 간단한 Decision Boundary 를 사용하여 데이터셋을 구성함
                    # 원본 Feature 0 이 5 보다 큰 데이터를 Positive Class 로 설정함

X_pos = X[positive_class]           # Decision Boundary 를 기반으로 Positive Class 로 분류된 데이터셋
X_neg = X[~positive_class]          # Decision Boundary 를 기반으로 Negative Class 로 분류된 데이터셋

# 그래프 영역 준비
ax.view_init(10, -70)
ax.set_xlabel("$x_1$", fontsize=18)
ax.set_ylabel("$x_2$", fontsize=18)
ax.set_zlabel("$x_3$", fontsize=18)
ax.set_xlim(axes[0:2])
ax.set_ylim(axes[2:4])
ax.set_zlim(axes[4:6])

ax.plot(X_neg[:, 0], X_neg[:, 1], X_neg[:, 2], 'y^')    # 원본 Feature 를 기반으로 Negative Class 데이터셋을 그래프로 그림
ax.plot(X_pos[:, 0], X_pos[:, 1], X_pos[:, 2], 'gs')    # 원본 Feature 를 기반으로 Positive Class 데이터셋을 그래프로 그림

ax.plot_wireframe(5, x2, x3, alpha=0.5)         # 3D 영역에서 현재 Decision Boundary 에 의해 생성된 Decision Plane 을 그림

save_fig("manifold_decision_boundary_plot1")
plt.show()

fig = plt.figure(figsize=(5, 4))
ax = plt.subplot(111)

plt.plot(t[positive_class], X[positive_class, 1], 'gs')         # Manifold 값을 기반으로 Positive Class 데이터셋을 그래프로 그림
plt.plot(t[~positive_class], X[~positive_class, 1], 'y^')       # Manifold 값을 기반으로 Negative Class 데이터셋을 그래프로 그림

plt.axis([4, 15, axes[2], axes[3]])
plt.xlabel("$z_1$", fontsize=18)
plt.ylabel("$x_2$", fontsize=18, rotation=0)
plt.grid(True)

save_fig("manifold_decision_boundary_plot2")
plt.show()

############################################################################################################################################
### 02 More Simplified Manifold with Complex Decision Boundary ##############################################################################
############################################################################################################################################

fig = plt.figure(figsize=(6, 5))
ax = plt.subplot(111, projection='3d')

positive_class = 2 * (t[:] - 4) > X[:, 1]       # Manifold 기반으로 좀 더 복잡한 Decision Boundary 를 사용하여 데이터셋을 구성함
                                                # Manifold 에 의해 설정된 직선 위에 배치된 값을 Positive Class 로 분류됨

X_pos = X[positive_class]       # Decision Boundary 를 기반으로 Positive Class 로 분류된 데이터셋
X_neg = X[~positive_class]      # Decision Boundary 를 기반으로 Negative Class 로 분류된 데이터셋

# 그래프 영역 준비
```

```
ax.view_init(10, -70)
ax.set_xlabel("$x_1$", fontsize=18)
ax.set_ylabel("$x_2$", fontsize=18)
ax.set_zlabel("$x_3$", fontsize=18)
ax.set_xlim(axes[0:2])
ax.set_ylim(axes[2:4])
ax.set_zlim(axes[4:6])

ax.plot(X_neg[:, 0], X_neg[:, 1], X_neg[:, 2], 'y^')         # 원본 Feature 를 기반으로 Negative Class 데이터셋을 그래프로 그림
ax.plot(X_pos[:, 0], X_pos[:, 1], X_pos[:, 2], 'gs')         # 원본 Feature 를 기반으로 Positive Class 데이터셋을 그래프로 그림

save_fig("manifold_decision_boundary_plot3")
plt.show()

fig = plt.figure(figsize=(5, 4))
ax = plt.subplot(111)

plt.plot(t[positive_class], X[positive_class, 1], 'gs')      # Manifold 값을 기반으로 Positive Class 데이터셋을 그래프로 그림
plt.plot(t[~positive_class], X[~positive_class, 1], 'y^')    # Manifold 값을 기반으로 Negative Class 데이터셋을 그래프로 그림

plt.plot([4, 15], [0, 22], "b-", linewidth=2)       # Manifold 영역 (2D) 상에 Decision Boundary 를 그림

plt.axis([4, 15, axes[2], axes[3]])
plt.xlabel("$z_1$", fontsize=18)
plt.ylabel("$x_2$", fontsize=18, rotation=0)
plt.grid(True)

save_fig("manifold_decision_boundary_plot4")
plt.show()
```