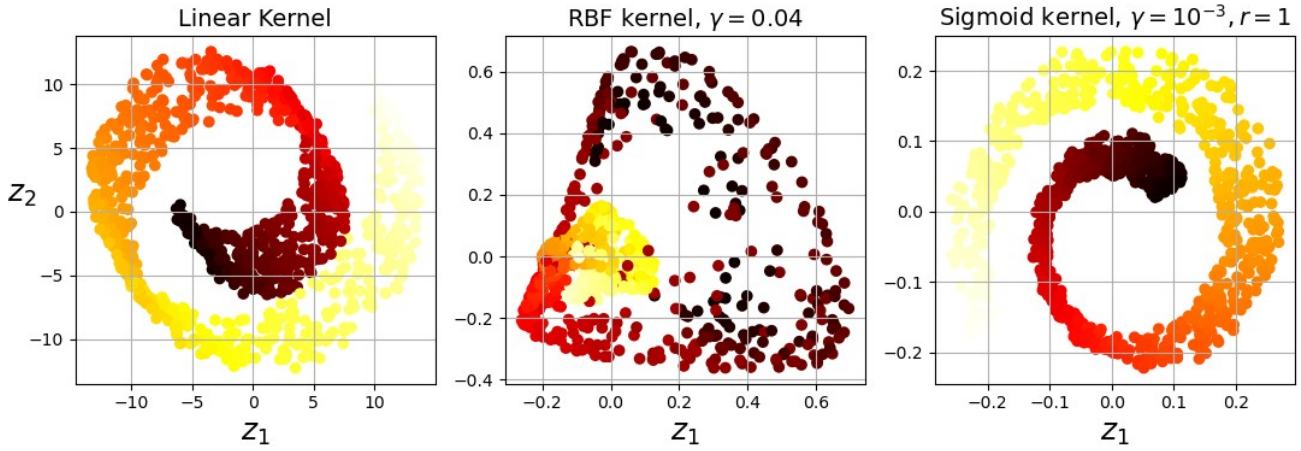
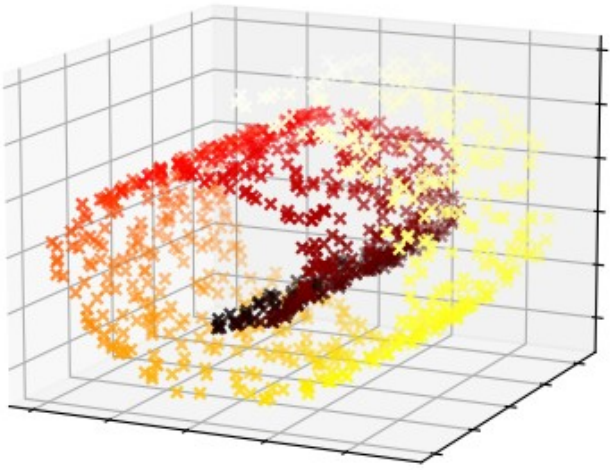
Q1-6. Kernel PCA

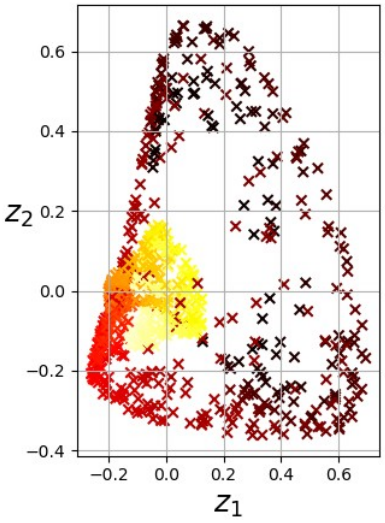
```
import numpy as np
import os

import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.decomposition import KernelPCA
from sklearn.datasets import make_swiss_roll

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

from sklearn.metrics import mean_squared_error

# 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 Dataset ##########################################################################################################
############################################################################################################################################
X, t = make_swiss_roll(n_samples=1000, noise=0.2, random_state=42)     # 1000 개 데이터로 구성된 3D Swiss Roll  데이터셋을 준비함

############################################################################################################################################
### Kernel PCA with different types of kernel ##############################################################################################
############################################################################################################################################
lin_pca = KernelPCA(n_components=2, kernel='linear', fit_inverse_transform=True)                        # Linear Kernel 을 이용한 Linear PCA
rbf_pca = KernelPCA(n_components=2, kernel="rbf", gamma=0.0433, fit_inverse_transform=True)             # Radial Function Kernel 을 이용한 RBF PCA
sig_pca = KernelPCA(n_components=2, kernel="sigmoid", gamma=0.001, coef0=1, fit_inverse_transform=True) # Sigmoid Kernel 을 이용한 Sigmoid PCA

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

# Kernel  종류를 변경하면서 PCA 를 수행함
for subplot, pca, title in ((131, lin_pca, 'Linear Kernel'), (132, rbf_pca, "RBF kernel, $\gamma=0.04$"), (133, sig_pca, "Sigmoid kernel, $\gamma=10^{-3}, r=1$")):

    X_reduced = pca.fit_transform(X)    # 현재 사용하는 PCA 를 통해 3D Swiss Roll  데이터셋을 재구성함

    if subplot == 132:
        X_reduced_rbf = X_reduced      # RBF PCA  결과 데이터셋을 별도로 추후 사용함

    plt.subplot(subplot)
    plt.title(title, fontsize=14)
    plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=t, cmap=plt.cm.hot)       #  재구성된 데이터셋과 Principle Component 로 그래프를 그림
    plt.xlabel("$z_1$", fontsize=18)
    if subplot == 131:
        plt.ylabel("$z_2$", fontsize=18, rotation=0)
    plt.grid(True)

save_fig("kernel_pca_plot")
plt.show()

plt.figure(figsize=(6, 5))

############################################################################################################################################
### Reprojection of RBF PCA-based dataset ##################################################################################################
############################################################################################################################################
X_inverse = rbf_pca.inverse_transform(X_reduced_rbf)   # RBF PCA  결과 데이터셋을 복구시킴

ax = plt.subplot(111, projection='3d')
ax.view_init(10, -70)
ax.scatter(X_inverse[:, 0], X_inverse[:, 1], X_inverse[:, 2], c=t, cmap=plt.cm.hot, marker="x") #  복구된 데이터셋을 3D  공간에 그래프로 그림
ax.set_xlabel("")
ax.set_ylabel("")
ax.set_zlabel("")
ax.set_xticklabels([])
ax.set_yticklabels([])
ax.set_zticklabels([])

save_fig("preimage_plot", tight_layout=False)
plt.show()

############################################################################################################################################
### Projection of dataset using RBF PCA ####################################################################################################
############################################################################################################################################
X_reduced = rbf_pca.fit_transform(X)   # RBF PCA 를 통해 3D Swiss Roll  데이터셋을 재구성함

plt.figure(figsize=(11, 4))
plt.subplot(132)
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=t, cmap=plt.cm.hot, marker="x") # Principle Component 를 기준으로 재구성된 데이터셋을 그래프로 그림
plt.xlabel("$z_1$", fontsize=18)
plt.ylabel("$z_2$", fontsize=18, rotation=0)
plt.grid(True)
plt.show()

############################################################################################################################################
### Finding the best Kernel PCA parameter for Logistic Regression ##########################################################################
############################################################################################################################################
# PCA 를 통해서 데이터셋을 재구성함
# PCA 로 재구성된 데이터셋을 Logistic Regression 에 사용하여 Classification 을 수행함
clf = Pipeline([
        ("kpca", KernelPCA(n_components=2)),
        ("log_reg", LogisticRegression(solver="lbfgs"))
    ])

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

y = t > 6.9     # Manifold  값을 기반으로 Output  데이터셋을 생성함
                # Logistic Regression 에 적합한 데이터셋을 구성하기 위해 조건문을 이용하여 True/False  형태로 Output  데이터셋을 준비함

grid_search = GridSearchCV(clf, param_grid, cv=3)   # Grid Search  준비
grid_search.fit(X, y)       #  데이터셋과 True/False 로 구성된 Output  데이터셋을 PCA 를 사용하여 재구성한 후 Logistic Regression 을 수행함
                          # 최상의 결과를 내는 PCA Parameter 를 구함

print(grid_search.best_params_)     #  최상 결과의 PCA Parameter 를 출력함

# 이전 단계의 RBF PCA Parameter 를 반영하여 PCA 를 수행함
rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.0433, fit_inverse_transform=True)
X_reduced = rbf_pca.fit_transform(X)                # RBF PCA 를 기반으로 데이터셋을 재구성함
X_preimage = rbf_pca.inverse_transform(X_reduced)   #  재구성된 데이터셋을 원본 Feature Space 로 복구시킴

print(mean_squared_error(X, X_preimage))    #  원본 데이터셋과 복구된 데이터셋 사이의 에러값을 출력함
```



Kernel 종류에 따른 PCA 로 재구성된 3D Swiss Roll 데이터셋 분포



RBF Kernel 으로 복구된 3D Swiss Roll 데이터셋



RBF Kernel 로 만들어진 Feature 로 구성된 3D Swiss Roll 데이터셋의 2D 분포