プログラム

実行環境と用いた言語・ライブラリを以下の表 1 に示す。

表 1: プログラムの実行環境

OS	:	Microsoft Windows 10 Pro (64bit)
CPU	:	Intel(R) Core(TM) i5-4300U
RAM	:	4.00 GB
使用言語	:	Python3.6
可視化	:	matplotlib ライブラリ

Listings 1: assignment2.py

```
# -*- coding: utf-8 -*-


import numpy as np
import matplotlib.pyplot as plt


def true_model(x):
    pix = np.pi * x
    target = np.sin(pix) / pix + 0.1 * x
    return target


def gauss_kernel(x, c, h):
    return np.exp(-(x - c)**2 / (2*h**2))


def generate_sample(xmin, xmax, sample_size):
    x = np.linspace(start=xmin, stop=xmax, num=sample_size)
    target = true_model(x)
    noise = 0.05 * np.random.normal(loc=0., scale=1., size=sample_size)
    return x, target + noise


def split(x, y, n_split=5):
    n_data = len(y)
    n_data_in_one_split = int(n_data / n_split)
    idx = np.arange(n_data)
    np.random.shuffle(idx)

    x_split = []
    y_split = []
    for i in range(n_split):
```

```
        idx_start = i * n_data_in_one_split
        idx_end = (i+1) * n_data_in_one_split
        if idx_end == n_data:
            idx_end = None
        x_split.append(x[idx_start:idx_end])
        y_split.append(y[idx_start:idx_end])
    return x_split, y_split


def split_train_test(x_split, y_split, k):
    n_split = len(y_split)
    x_test, y_test = x_split[k], y_split[k]
    x_train, y_train = [], []
    for _k in range(n_split):
        if _k != k:
            x_train.extend(x_split[_k])
            y_train.extend(y_split[_k])
    x_train = np.array(x_train)
    y_train = np.array(y_train)
    return x_train, y_train, x_test, y_test


def calc_design_matrix(x, c, h, kernel):
    return kernel(x[None], c[:, None], h)


def solve_gauss_kernel_model(x, y, h, lamb):
    k = calc_design_matrix(x, x, h, gauss_kernel)
    theta = np.linalg.solve(
        k.T.dot(k) + lamb*np.identity(len(k)),
        k.T.dot(y[:, None]),
        )
    return theta


def solve_gauss_kernel_sparse_model(x, y, h, lamb, eps=1e-2, iter_max=100,
    n_look=5):
    def update(theta, z, u, lamb, K, y):
        theta = np.linalg.inv(K.T.dot(K)+ np.eye(len(K))).dot(K.dot(y[:,
    None]) + z - u)

        z_plus = theta + u - lamb
        z_plus[z_plus < 0] = 0
        z_minus = theta + u + lamb
        z_minus[z_minus > 0] = 0
        z = z_plus + z_minus

```

```
        u = u + theta - z
        return theta, z, u

    def compute_lagrange_func(theta, z, u, lamb, K, y):
        loss = (
            (1/2) * np.linalg.norm(K.dot(theta) - y)
            + lamb * np.linalg.norm(z, ord=1)
            + u.T.dot(theta - z)
            + (1/2) * np.linalg.norm(theta - z)
            )
        return loss

    K = calc_design_matrix(x, x, h, gauss_kernel)
    theta = np.linalg.solve(K.T.dot(K), K.T.dot(y[:, None]))
    z = np.random.rand(*theta.shape) * 0.1
    u = np.random.rand(*theta.shape) * 0.1
    lagrange_list = []
    for i in range(iter_max):
        theta, z, u = update(theta, z, u, lamb, K, y)
        lagrange = compute_lagrange_func(theta, z, u, lamb, K, y)
        if i < n_look:
            lagrange_list.append(lagrange)
        else:
            lagrange_list = lagrange_list[1:] + [lagrange]
            if max(lagrange_list) - min(lagrange_list) < eps:
                break
    return theta, z, u, i+1


def compute_loss(x_train, x_test, y, h, theta, lamb):
    k = calc_design_matrix(x_train, x_test, h, gauss_kernel)
    loss = (1/2)*np.linalg.norm(k.dot(theta) - y)
    # loss += (lamb/2)*np.linalg.norm(theta)
    return loss


def main():
    np.random.seed(0)  # set the random seed for reproducibility

    # create sample
    xmin, xmax = -3, 3
    sample_size = 50
    n_split = 5
    x, y = generate_sample(xmin=xmin, xmax=xmax, sample_size=sample_size)
    # print(x.shape, y.shape)

    x_split, y_split = split(x, y, n_split=n_split)
```

```
    # print(x_split[0].shape, y_split[0].shape)

    # global search
    h_cands = [1e-2, 1e-1, 1, 1e1,]
    lamb_cands = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1,]

    # local search
    #searched_range_base = np.arange(0.5, 1.5, 0.1)
    #h_cands = 1.0 * searched_range_base
    #lamb_cands = 1e-6 * searched_range_base

    loss_min = 1e8
    h_best = None
    lamb_best = None
    theta_best = None
    n_iter_best = None

    n_row = len(lamb_cands)
    n_col = len(h_cands)
    fig = plt.figure(figsize=(n_col*4, n_row*4))
    fig_idx = 0

    for lamb in lamb_cands:
        for h in h_cands:
            losses = []
            for k in range(n_split):
                x_train, y_train, x_test, y_test = split_train_test(x_split, y_split, k)
                # print(x_train.shape, y_train.shape)

                theta, z, u, n_iter = solve_gauss_kernel_sparse_model(
                    x_train, y_train, h, lamb,
                    eps=1e-3, iter_max=200, n_look=10,
                    )
                loss_k = compute_loss(x_train, x_test, y_test, h, theta, lamb)
                losses.append(loss_k)
            loss = np.mean(losses)

            if loss < loss_min:
                loss_min = loss
                h_best = h
                lamb_best = lamb
                theta_best = theta
                n_iter_best = n_iter

            # for visualization
```

```
171             X = np.linspace(start=xmin, stop=xmax, num=5000)
172             true = true_model(X)
173             K = calc_design_matrix(x_train, X, h, gauss_kernel)
174             prediction = K.dot(theta)
175 
176             # visualization
177             fig_idx += 1
178             ax = fig.add_subplot(n_row, n_col, fig_idx)
179             ax.set_title('$h = {},\ \lambda = {}$, L = {:.2f}'.format(h,
        lamb, loss))
180             ax.scatter(x, y, c='green', marker='o', label='data')
181             ax.plot(X, true, linestyle='dashed', label='true')
182             ax.plot(X, prediction, linestyle='solid', label='predicted')
183             ax.legend()
184 
185     print('Best Model')
186     print('\th = {}'.format(h_best))
187     print('\tlambda = {}'.format(lamb_best))
188     print('\tloss = {}'.format(loss_min))
189     print('\tn_iter: {}'.format(n_iter_best))
190     print('\ttheta: \n', theta_best)
191 
192     plt.savefig('../figures/assignment2_result.png')
193     plt.show()
194 
195 
196 if __name__ == '__main__':
197     main()
```