1　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

ソースコードは Listing 1 に示した。以下に簡単に各関数の説明を記す。

- load_data
  .mat ファイルからデータを取り出す。
- plot
  点群及び境界をプロットする。2 クラスを o と x で表し，分類結果が正しいものを青，誤っているものを赤で示す。
- perceptron
  パーセプトロンを用いて重みを求める関数。
- mse
  MSE 法を用いて重みを求める関数。LMS 法を用いる場合と解析的に求める場合とを使い分けられる。

Listings 1: assignment1.py

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

import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat


def load_data(path):
    data = loadmat(path)
    #print(data.keys())
    x = data['x']
    l = data['l']
    n = data['n'][0, 0]
    d = data['d'][0, 0]
    return x, l, n, d


def plot(x, l, aw, neg):
    plt.clf()
    plt.xlim([-1, 1])
```

```
    plt.ylim([-1, 1])
    plt.plot(
        x[0, np.where(np.logical_and(l==1, ~neg))],
        x[1, np.where(np.logical_and(l==1, ~neg))],
        'bo',
        )
    plt.plot(
        x[0, np.where(np.logical_and(l==-1, ~neg))],
        x[1, np.where(np.logical_and(l==-1, ~neg))],
        'bx',
        )
    plt.plot(
        x[0, np.where(np.logical_and(l==1, neg))],
        x[1, np.where(np.logical_and(l==1, neg))],
        'ro',
        )
    plt.plot(
        x[0, np.where(np.logical_and(l==-1, neg))],
        x[1, np.where(np.logical_and(l==-1, neg))],
        'rx',
        )

    if abs(aw[1]) > abs(aw[2]):
        plt.plot(
            [-1, 1],
            [-(aw[0]-aw[1])/aw[2], -(aw[0]+aw[1])/aw[2]]
            )
    else:
        plt.plot(
            [-(aw[0]-aw[2])/aw[1], -(aw[0]+aw[2])/aw[1]],
            [-1, 1]
            )
    plt.waitforbuttonpress()


def perceptron(x, l, n, d, max_iter=100, fig_path=None,):
    # hyperparameter
    rho = 0.1

    # augmented vectors
    ax = np.concatenate((np.ones((1, n)), x))
    aw = (2*np.random.rand(d+1) - np.array([1, 1, 1]))[:, np.newaxis]

    # normalize
    ax[:, np.where(l == -1)] = -ax[:, np.where(l == -1)]

    # solve
```

```
    neg = ((ax.T.dot(aw)).T < 0)[-1]
    plt.figure()
    plt.ion()
    for n_iter in range(1, 1+int(max_iter)):
        # update
        aw += rho * ax[:, neg].sum(axis=1)[:, np.newaxis]

        # result
        neg = ((ax.T.dot(aw)).T < 0)[-1]
        n_left_neg = len(np.where(neg)[-1])

        print(f'#Iter: {n_iter}\t#Left Neg: {n_left_neg}')
        print('aw: ', aw.reshape(d+1))
        print()

        #plot(x, l, aw, neg)  # use when want to look move of boundary while learning

        # convergence condition
        if n_left_neg == 0:
            break

    # plot
    plot(x, l, aw, neg)
    if fig_path:
        plt.savefig(fig_path)
    plt.show()

    return aw


def mse(x, l, n, d, use_lms=False, max_iter=100, eps=1e-4, fig_path=None):
    # hyperparameter
    #rho = 0.015
    rho = 0.001

    # augmented vectors
    ax = np.concatenate((np.ones((1, n)), x))
    aw = (2*np.random.rand(d+1) - np.array([1, 1, 1]))[:, np.newaxis]

    plt.figure()
    plt.ion()

    # solve
    if not use_lms:
        pseudo_inverse_matrix = np.linalg.inv(ax.dot(ax.T)).dot(ax)
        aw = pseudo_inverse_matrix.dot(l.T)
```

```
    else:
        aw_last = aw.copy()
        for n_iter in range(1, 1+int(max_iter)):
            # predict
            g = (ax.T.dot(aw)).T

            # update
            aw -= rho * (g - l).dot(ax.T).T

            # result
            g[g > 0] = 1
            g[g < 0] = -1
            neg = (g != l)
            n_wrong = len(np.where(neg)[-1])

            print(f'#Iter: {n_iter}\t#Wrong: {n_wrong}')
            print('aw: ', aw.reshape(d+1))
            print()

            plot(x, l, aw, neg)  # use when want to look move of boundary while learning

            # convergence condition
            if np.linalg.norm(aw - aw_last) < eps:
                break
            aw_last = aw.copy()

    # plot
    g = (ax.T.dot(aw)).T
    g[g > 0] = 1
    g[g < 0] = -1
    neg = (g != l)
    plot(x, l, aw, neg)
    if fig_path:
        plt.savefig(fig_path)
    plt.show()

    return aw


def main():
    # settings
    data_type = 'linear'
    #data_type = 'nonlinear'
    #data_type = 'slinear'
    data_path = f'../../code/{data_type}-data.mat'
    np.random.seed(0)
```

```

    # load data
    x, l, n, d = load_data(data_path)
    print(f'Data Type: {data_type}  #Sample: {n}  #Dim: {d}\n')

    # perceptron
    #fig_path = f'../figures/assignment1_1_{data_type}_result.png'
    #aw = perceptron(x, l, n, d, max_iter=100, fig_path=fig_path)

    # MSE
    fig_path = f'../figures/assignment1_2_{data_type}_result.png'
    aw = mse(
        x, l, n, d,
        use_lms=False, max_iter=100, eps=1e-4,
        fig_path=fig_path
        )


if __name__ == '__main__':
    main()
```