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

Listings 1: assignment1.py

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


import numpy as np
import matplotlib.pyplot as plt


def dataset1():
    n = 1000
    alpha = 0.3

    # prior probability and number of samples
    n1 = sum(np.random.rand(n) < alpha)
    n2 = n - n1

    mean1, mean2 = np.array([2, 0]), np.array([-2, 0])
    cov1 = np.array([[1, 0], [0, 9]])
    cov2 = np.array([[1, 0], [0, 2]])

    # generate data
    x1 = sampling_normal_dist(mean1, cov1, n1)
    x2 = sampling_normal_dist(mean2, cov2, n2)

    return x1, x2, mean1, mean2, cov1, cov2


def dataset2():
    n = 1000
    alpha = 0.4

    # prior probability and number of samples
    n1 = sum(np.random.rand(n) < alpha)
    n2 = n - n1
```

```
34
35      mean1, mean2 = np.array([2, 2]), np.array([-2, -2])
36      cov1 = np.array([[5, 0], [0, 6]])
37      cov2 = np.array([[6, 0], [0, 4]])
38
39      # generate data
40      x1 = sampling_normal_dist(mean1, cov1, n1)
41      x2 = sampling_normal_dist(mean2, cov2, n2)
42
43      return x1, x2, mean1, mean2, cov1, cov2
44
45
46  def dataset3():
47      n = 1000
48      alpha = 0.5
49
50      # prior probability and number of samples
51      n1 = sum(np.random.rand(n) < alpha)
52      n2 = n - n1
53
54      mean1, mean2 = np.array([1, 1]), np.array([1, 1])
55      cov1 = np.array([[1, 0], [0, 2]])
56      cov2 = np.array([[8, 0], [0, 5]])
57
58      # generate data
59      x1 = sampling_normal_dist(mean1, cov1, n1)
60      x2 = sampling_normal_dist(mean2, cov2, n2)
61
62      return x1, x2, mean1, mean2, cov1, cov2
63
64
65  def sampling_normal_dist(mean, cov, n):
66      return np.random.multivariate_normal(mean, cov, n)
67
68
69  def log_probability_normal_dist(x, mean, cov, prior_prob,):
70      d = len(mean)
71      cov_inv = np.linalg.inv(cov)
72      logp = (
73          -(1/2) * ((x - mean).dot(cov_inv) * (x - mean)).sum(axis=1)
74          -(1/2) * np.log(np.linalg.det(cov))
75          -(d/2) * np.log(2 * np.pi)
76          )
77      return logp
78
79
80  def classifier_binary(x, a, b, c):
```

```
    result = (x.dot(a) * x).sum(axis=1)
    result += b.dot(x.T)
    result += c
    return result


def classify_binary_1(x, mean1, mean2, sigma, p1, p2):
    d = len(mean1)
    a = np.zeros((d, d))
    b = (1/sigma**2) * (mean1 - mean2)
    c = -(1/(2*(sigma)**2)) * (np.linalg.norm(mean1)**2 -
    np.linalg.norm(mean2)**2) + np.log(p1/p2)
    result = classifier_binary(x, a, b, c)
    return result, a, b, c


def classify_binary_2(x, mean1, mean2, cov, p1, p2):
    d = len(mean1)
    cov_inv = np.linalg.inv(cov)
    a = np.zeros((d, d))
    b = cov_inv.dot(mean1 - mean2)
    c = -(1/2) * (mean1.T.dot(cov_inv).dot(mean1) -
    mean2.T.dot(cov_inv).dot(mean2)) + np.log(p1/p2)
    result = classifier_binary(x, a, b, c)
    return result, a, b, c


def classify_binary_3(x, mean1, mean2, cov1, cov2, p1, p2):
    cov1_inv = np.linalg.inv(cov1)
    cov2_inv = np.linalg.inv(cov2)
    a = -(1/2) * (cov1_inv - cov2_inv)
    b = cov1_inv.dot(mean1)  - cov2_inv.dot(mean2)
    c = (
        -(1/2)*(mean1.T.dot(cov1_inv).dot(mean1) -
    mean2.T.dot(cov2_inv).dot(mean2))
        - (1/2)*np.log(np.linalg.det(cov1)/np.linalg.det(cov2))
        + np.log(p1/p2)
        )
    result = classifier_binary(x, a, b, c)
    return result, a, b, c


def measure_accuracy(result, label, n_data):
    if label == 1:
        is_correct = (result >= 0)
    elif label == 2:
        is_correct = (result < 0)
```

```
    else:
        raise ValueError("'label' must be 1 or 2.")

    n_correct = is_correct.sum()
    acc = n_correct / n_data
    print('#Data: {}\t#Correct: {}\tAcc: {:.3f}'.format(n_data, n_correct,
    acc))
    return is_correct


def sampling_normal_dist_for_contour(x, y, mean, cov):
    icov = np.linalg.inv(cov)
    xt = x - mean[0]
    yt = y - mean[1]
    p = (
        1./(2. * np.pi * np.sqrt(np.linalg.det(cov)))
        * np.exp(
            -(1./2.)*(
                icov[0, 0]*xt*xt
                + (icov[0, 1] + icov[1, 0])*xt*yt
                + icov[1, 1]*yt*yt
                )
            )
        )
    return p


def plot_data(
        x1_o, x1_x, x2_o, x2_x,
        a, b, c,
        mean1, mean2, cov1, cov2,
        x_linespace=None, y_linespace=None,
        show=True, save=False, path=None,
        size=10,
        ):
    if x_linespace is None:
        x_linespace = np.linspace(-10, 10, 100)
    if y_linespace is None:
        y_linespace = np.linspace(-10, 10, 100)
    _x, _y = np.meshgrid(x_linespace, y_linespace)

    plt.figure(figsize=(15, 15))
    plt.axis('equal')

    p1_contour = sampling_normal_dist_for_contour(_x, _y, mean1, cov1)
    plt.scatter(x1_o[:, 0], x1_o[:, 1], s=size, marker='o',
    color='darkorange')
```

```
170     plt.scatter(x1_x[:, 0], x1_x[:, 1], s=size, marker='o', color='blue')
171     cs1 = plt.contour(_x, _y, p1_contour, cmap='hsv')
172     plt.clabel(cs1)
173 
174     p2_contour = sampling_normal_dist_for_contour(_x, _y, mean2, cov2)
175     plt.scatter(x2_o[:, 0], x2_o[:, 1], s=size, marker='x',
        color='royalblue')
176     plt.scatter(x2_x[:, 0], x2_x[:, 1], s=size, marker='x', color='red')
177     cs2 = plt.contour(_x, _y, p2_contour, cmap='hsv')
178     plt.clabel(cs2)
179 
180     # decision boundary
181     _xy = np.c_[np.reshape(_x, -1), np.reshape(_y, -1)]
182     pp = classifier_binary(_xy, a, b, c)
183     pp = np.reshape(pp, _x.shape)
184     cs = plt.contour(_x, _y, pp, levels=[0.0], colors=['darkcyan'])
185     # plt.clabel(cs)
186 
187     if save:
188         plt.savefig(path)
189     if show:
190         plt.show()
191     return
192 
193 
194 def main():
195     # settings
196     case = 3
197     dataset_id = 3
198     fig_path =
        '../figures/result_assignment1_dataset{}_case{}.png'.format(dataset_id,
        case)
199     offset = 0.5
200     n_linespace = 100
201     np.random.seed(0)
202 
203     print('Case {}'.format(case))
204     print('Dataset: {}'.format(dataset_id))
205     print()
206 
207 
208     # load data
209     x1, x2, mean1, mean2, cov1, cov2 = dataset3()
210     n1 = len(x1)
211     n2 = len(x2)
212     n = n1 + n2
213     p1 = n1 / n
```

```
    p2 = n2 / n


    # decide which model to use
    if case == 1:
        sigma = (np.diag(cov1).sum() + np.diag(cov2).sum())/4
        result_1, a, b, c = classify_binary_1(
            x1, mean1, mean2, sigma, p1, p2
            )
        result_2, a, b, c = classify_binary_1(
            x2, mean1, mean2, sigma, p1, p2
            )
    elif case == 2:
        cov = (cov1 + cov2)/2
        result_1, a, b, c = classify_binary_2(
            x1, mean1, mean2, cov, p1, p2
            )
        result_2, a, b, c = classify_binary_2(
            x2, mean1, mean2, cov, p1, p2
            )
    elif case == 3:
        result_1, a, b, c = classify_binary_3(
            x1, mean1, mean2, cov1, cov2, p1, p2
            )
        result_2, a, b, c = classify_binary_3(
            x2, mean1, mean2, cov1, cov2, p1, p2
            )
    else:
        raise ValueError("'case' must be 1, 2, or 3.")


    # classify x1
    print('x1')
    is_correct_1 = measure_accuracy(result_1, 1, len(x1))
    x1_o = x1[is_correct_1]
    x1_x = x1[~is_correct_1]
    print()

    # classify x2
    print('x2')
    is_correct_2 = measure_accuracy(result_2, 2, len(x2))
    x2_o = x2[is_correct_2]
    x2_x = x2[~is_correct_2]
    print()

    acc = (is_correct_1.sum() + is_correct_2.sum()) / (len(is_correct_1) +
    len(is_correct_2))
```

```
260     print('Accuracy: {:.3f}'.format(acc))
261     print()
262
263
264     # plot
265     _x = np.concatenate([x1, x2], axis=0)
266     x_linespace = np.linspace(
267         _x[:, 0].min()-offset, _x[:, 0].max()+offset, n_linespace
268         )
269     y_linespace = np.linspace(
270         _x[:, 1].min()-offset, _x[:, 1].max()+offset, n_linespace
271         )
272     plot_data(
273         x1_o, x1_x, x2_o, x2_x,
274         a, b, c,
275         mean1, mean2, cov1, cov2,
276         x_linespace, y_linespace,
277         save=True, path=fig_path,
278         size=20,
279         )
280
281
282 if __name__ == '__main__':
283     main()
```

Listings 2: assignment2.py

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


import pathlib
import numpy as np
import matplotlib.pyplot as plt

from pprint import pprint

import mnread


def load_data():
    x_train = mnread.readim(mnread.trdatafz)
    y_train = mnread.readlabel(mnread.trlabelfz)
    x_test = mnread.readim(mnread.tstdatafz)
    y_test = mnread.readlabel(mnread.tstlabelfz)
    return x_train, y_train, x_test, y_test


def get_statistics(data, labels):
    n_data = len(labels)
    all_labels = sorted(list(set(labels)))
    # n_label = len(all_labels)
    statistics = {}
    for label in all_labels:
        _data = data[np.where(labels == label), :]
        n = _data.shape[1]
        _data = np.reshape(_data, [n, -1])
        statistics[label] = {
            'n': n,
            'p': n/n_data,
            'mean': np.mean(_data, axis=0),
            'cov': np.cov(_data.T),
            }
    return statistics


def train(case, statistics):
    all_labels = list(statistics.keys())
    n_label = len(all_labels)
    if case == 1:
        sigma = 0.0
        for label in all_labels:
            cov = statistics[label]['cov']
            sigma += np.diag(cov).mean()
```

```
        sigma /= n_label
        for label in all_labels:
            statistics[label]['cov_train'] = np.sqrt(sigma)
        log_prob = log_prob_1
    elif case == 2:
        Sigma = np.zeros_like(statistics[all_labels[0]]['cov'])
        for label in all_labels:
            cov = statistics[label]['cov']
            Sigma += cov
        Sigma /= n_label
        for label in all_labels:
            statistics[label]['cov_train'] = Sigma
        log_prob = log_prob_2
    elif case == 3:
        for label in all_labels:
            statistics[label]['cov_train'] = statistics[label]['cov']
        log_prob = log_prob_3
    else:
        raise ValueError("'case' must be 1, 2, or 3.")
    return statistics, log_prob


def log_probability_normal_dist(x, mean, cov, prior_prob,):
    d = len(mean)
    cov_inv = np.linalg.pinv(cov)
    logp = (
        -(1/2) * ((x - mean).dot(cov_inv) * (x - mean)).sum(axis=1)
        -(1/2) * np.log(np.linalg.det(cov))
        -(d/2) * np.log(2 * np.pi)
        + np.log(prior_prob)
        )
    return logp


def log_prob_1(x, mean, sigma, prior_prob, eps=1e-4):
    logp = mean.T.dot(x.T) - (1/2)*mean.T.dot(mean) +
    (sigma**2)*np.log(prior_prob)
    return logp


def log_prob_2(x, mean, cov, prior_prob, eps=1e-4):
    #cov_new = cov + eps*np.eye(len(cov))
    #cov_inv = np.linalg.inv(cov_new)
    cov_inv = np.linalg.pinv(cov)
    logp = mean.T.dot(cov_inv).dot(x.T) -
    (1/2)*mean.T.dot(cov_inv).dot(mean) + np.log(prior_prob)
    return logp
```

```
92
93
94  def log_prob_3(x, mean, cov, prior_prob, eps=1e-4):
95      #cov_new = cov + eps*np.eye(len(cov))
96      #cov_inv = np.linalg.inv(cov_new)
97      #det = np.linalg.det(cov_new)
98      #print(det, np.linalg.det(cov))
99      cov_inv = np.linalg.pinv(cov)
100     logp = -(1/2) * ((x - mean).dot(cov_inv) * (x - mean)).sum(axis=1)
101     #logp += - (1/2)*np.log(det)
102     logp += np.log(prior_prob)
103     return logp
104
105
106 def classify(x, statistics, log_prob, eps=1e-4):
107     n_data = len(x)
108     x = np.reshape(x, [n_data, -1])
109     all_labels = list(statistics.keys())
110     y_pred = []
111     for label in all_labels:
112         mean = statistics[label]['mean']
113         cov = statistics[label]['cov_train']
114         prior_prob = statistics[label]['p']
115         logp = log_prob(x, mean, cov, prior_prob, eps)
116         y_pred.append(logp)
117     y_pred = np.array(y_pred).T
118     y_pred = np.argmax(y_pred, axis=1)
119     return y_pred
120
121
122 def evaluate(y_true, y_pred, labels):
123     n_data = len(y_true)
124     n_label = len(labels)
125
126     # accuracy
127     n_correct = (y_pred == y_true).sum()
128     acc = n_correct / n_data
129     print('All\t#Data: {}\t#Correct: {}\tAcc: {:.3f}'.format(n_data,
        n_correct, acc))
130
131     # acc per label
132     confusion_matrix = np.zeros((n_label, n_label), dtype=int)
133     for i, label_true in enumerate(labels):
134         idx_y_true = (y_true == label_true)
135         _y_pred = y_pred[idx_y_true]
136         _n_data = len(_y_pred)
137         for j, label_pred in enumerate(labels):
```

```
138             n = (_y_pred == label_pred).sum()
139             confusion_matrix[i, j] = n
140 
141         n_correct = (_y_pred == label_true).sum()
142         acc = n_correct / _n_data
143         print('Label: {}\t#Data: {}\t#Correct: {}\tAcc:
        {:.3f}'.format(label_true, _n_data, n_correct, acc))
144     print()
145     print('Confusion Matrix\n', confusion_matrix)
146     print()
147     return confusion_matrix
148 
149 
150 def visualize(case, x, y_true, y_pred, image_dir, n=50):
151     image_dir = pathlib.Path(image_dir)
152 
153     plt.figure()
154     plt.suptitle('correct')
155     indices_correct =
        np.random.permutation(np.where(y_pred==y_true)[-1])[range(n)]
156     for i, idx_correct in enumerate(indices_correct):
157         plt.subplot(5, 10, i+1)
158         plt.axis('off')
159         plt.imshow(x[idx_correct, :, :], cmap='gray')
160         plt.title(y_pred[idx_correct])
161     plt.savefig(str(image_dir /
        'result_assignment2_case{}_correct.png'.format(case)))
162 
163     plt.figure()
164     plt.suptitle('wrong')
165     indices_wrong =
        np.random.permutation(np.where(~(y_pred==y_true))[-1])[range(n)]
166     for i, idx_wrong in enumerate(indices_wrong):
167         plt.subplot(5,10,i+1)
168         plt.axis('off')
169         plt.imshow(x[idx_wrong, :, :], cmap='gray')
170         plt.title('{} ({})'.format(y_pred[idx_wrong], y_true[idx_wrong]))
171     plt.savefig(str(image_dir /
        'result_assignment2_case{}_wrong.png'.format(case)))
172     plt.show()
173 
174 
175 def main():
176     # settings
177     case = 3
178     image_dir = pathlib.Path().cwd().parent / 'figures'
179     np.random.seed(0)
```

```
180     print('Case: {}'.format(case))
181
182     # load data
183     x_train, y_train, x_test, y_test = load_data()
184     #print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
185
186     all_labels = sorted(list(set(y_train)))
187     #n_data = len(y_train)
188     #n_label = len(all_labels)
189     #print(n_data, n_label, all_labels)
190
191     # train (get statistics and calc sigmas)
192     train_statistics = get_statistics(x_train, y_train)
193     train_statistics, log_prob = train(case, train_statistics)
194
195     # test
196     y_pred = classify(x_test, train_statistics, log_prob, eps=1e-2)
197     evaluate(y_test, y_pred, all_labels)
198     visualize(case, x_test, y_test, y_pred, image_dir, n=50)
199
200
201 if __name__ == '__main__':
202     main()
```