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 generate_data(n=3000):
    x = np.zeros(n)
    u = np.random.rand(n)
    index1 = np.where((0 <= u) & (u < 1 / 8))
    x[index1] = np.sqrt(8 * u[index1])
    index2 = np.where((1 / 8 <= u) & (u < 1 / 4))
    x[index2] = 2 - np.sqrt(2 - 8 * u[index2])
    index3 = np.where((1 / 4 <= u) & (u < 1 / 2))
    x[index3] = 1 + 4 * u[index3]
    index4 = np.where((1 / 2 <= u) & (u < 3 / 4))
    x[index4] = 3 + np.sqrt(4 * u[index4] - 2)
    index5 = np.where((3 / 4 <= u) & (u <= 1))
    x[index5] = 5 - np.sqrt(4 - 4 * u[index5])
    return x


def split(x, n, shuffle=True):
    n_data = len(x)
    n_data_split = n_data // n
    n_abandoned = n_data % n
    if n_abandoned != 0:
        print(f'Warning: {n_abandoned} samples are abandoned')
    if shuffle:
        x_split = np.random.permutation(x)
    else:
        x_split = x.copy()
```

```
    x_split = [x_split[i:i+n_data_split] for i in range(0, n_data,
    n_data_split)]
    return x_split


def make_train_data(x, n_split, i):
    x_valid = x[i]
    x_train = []
    for j in range(n_split):
        if j != i:
            x_train.extend(x[j])
    x_train = np.array(x_train)
    return x_train, x_valid


def gauss_kernel(x, d):
    k = (1/(2*np.pi)**(d/2)) * np.exp(-(1/2)*x**2)
    #print((1/(2*np.pi)**(d/2)), k, -(1/2)*x.T.dot(x))
    return k


def kernel_density(x, h, x_axis, kernel):
    n = x.shape[0]
    if len(x.shape) == 1:
        d = 1
    else:
        d = x.shape[1]
    prob = np.zeros(len(x_axis))
    for x_i in x:
        kernel_input = (x_axis - x_i) / h
        #print(kernel_input)
        prob += kernel(kernel_input, d)
    prob = prob / (n * h**d)
    #print(x)
    #print(prob)
    return prob


def estimate_kernel_density(
        x, n_split, bandwidth_list, kernel,
        offset=1.0, num=100,
        path=None,
        ):
    x_axis = np.linspace(x.min(), x.max(), num)
    x_split = split(x, n=n_split, shuffle=True)

    n_bandwidth = len(bandwidth_list)
```

```
    n_row = 1
    n_col = n_bandwidth
    fig = plt.figure(figsize=(n_col*4, n_row*6))

    lcv_list = []
    for i, bandwidth in enumerate(bandwidth_list):
        # calc LCV by likelihood cross validation
        lcv_list_tmp = []
        for j in range(n_split):
            x_train, x_valid = make_train_data(x_split, n_split, j)
            p = kernel_density(x_valid, bandwidth, x_train, kernel)
            #lcv = p.sum()
            lcv = np.log(p).sum()
            lcv_list_tmp.append(lcv)
        lcv = np.mean(lcv_list_tmp)
        lcv_list.append(lcv)

        # plot
        p = kernel_density(x_train, bandwidth, x_axis, kernel)
        ax = fig.add_subplot(n_row, n_col, (i+1))
        ax.set_title(f'$h = {bandwidth}$')
        ax.hist(x, bins=50, normed=True)
        ax.plot(x_axis, p)
        ax.set_ylim([0, 1.0])

    if path:
        plt.savefig(str(path))
    plt.show()
    return lcv_list


def main():
    # settings
    n_sample = 3000
    n_split = 10
    h_list = [0.01, 0.05, 0.1, 0.5,]  # global
    #h_list = [0.05, 0.075, 0.1, 0.15]  # local
    offset = 1.0
    num = 100
    fig_path = '../figures/assignment1_result_global.png'
    np.random.seed(0)


    # load data
    x = generate_data(n_sample)
    #print('x shape: {}'.format(x.shape))
    #plt.hist(x, bins=50)
```

```
127     #plt.show()
128
129
130     # train
131     lcv_list = estimate_kernel_density(
132         x, n_split, h_list, gauss_kernel,
133         offset=offset, num=num,
134         path=fig_path,
135         )
136
137     # result
138     form = '{:5.2f}'
139     tab = '  '
140     string_h = '{:3}'.format('h')
141     string_lcv = '{:3}'.format('LCV')
142     for h, lcv in zip(h_list, lcv_list):
143         string_h += tab + form.format(h)
144         string_lcv += tab + form.format(lcv)
145     result = string_h + '\n' + string_lcv
146     print(result)
147
148
149 if __name__ == '__main__':
150     main()
```

Listings 2: assignment2.py

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


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


def load_data(n_label=None, n_train=None, n_test=None):
    data_dir = '../data/'
    data_dir = pathlib.Path(data_dir)
    categories = list(range(10))
    train_X = []
    train_y = []
    test_X = []
    test_y = []
    for category in categories[:n_label]:
        # train data
        data_path = data_dir / 'digit_train{}.csv'.format(category)
        data = np.loadtxt(str(data_path), delimiter=',')[:n_train]
        n_data = len(data)
        train_X.extend(data)
        train_y.extend(np.ones(n_data) * category)

        # test data
        data_path = data_dir / 'digit_test{}.csv'.format(category)
        data = np.loadtxt(str(data_path), delimiter=',')[:n_test]
        n_data = len(data)
        test_X.extend(data)
        test_y.extend(np.ones(n_data) * category)
    train_X = np.array(train_X)
    train_y = np.array(train_y)
    test_X = np.array(test_X)
    test_y = np.array(test_y)
    labels = categories[:n_label]
    return train_X, train_y, test_X, test_y, labels


def shuffle(data_X, data_y):
    n_data = len(data_y)
    indices = np.arange(n_data)
    np.random.shuffle(indices)
    data_X_shuffled = data_X[indices]
    data_y_shuffled = data_y[indices]
    return data_X_shuffled, data_y_shuffled

```

```
47
48 def split(data_X, data_y, n):
49     n_data = len(data_y)
50     n_data_split = n_data // n
51     n_abandoned = n_data % n
52     if n_abandoned != 0:
53         print(f'Warning: {n_abandoned} samples are abandoned')
54     data_X_split = [data_X[i:i+n_data_split] for i in range(0, n_data,
       n_data_split)]
55     data_y_split = [data_y[i:i+n_data_split] for i in range(0, n_data,
       n_data_split)]
56     return data_X_split, data_y_split
57
58
59 def make_train_data(train_X, train_y, n_split, i):
60     train_X_valid = train_X[i]
61     train_y_valid = train_y[i]
62     train_X_train = []
63     train_y_train = []
64     for j in range(n_split):
65         if j != i:
66             train_X_train.extend(train_X[j])
67             train_y_train.extend(train_y[j])
68     train_X_train = np.array(train_X_train)
69     train_y_train = np.array(train_y_train)
70     return train_X_train, train_y_train, train_X_valid, train_y_valid
71
72
73 def knn(train_X, train_y, test_X, k_list, save_memory=False):
74     if save_memory:
75         n_train = train_X.shape[0]
76         n_test = test_X.shape[0]
77         dist_matrix = np.zeros((n_test, n_train))
78         for i in range(n_test):
79             test_X_i = test_X[i]
80             dist_matrix[i, :] = np.sum((train_X - test_X_i[np.newaxis,
       :])**2, axis=1)
81     else:
82         dist_matrix = np.sqrt(
83             np.sum((train_X[None] - test_X[:, None])**2, axis=2)
84             )
85
86     sorted_index_matrix = np.argsort(dist_matrix, axis=1)
87     ret_matrix = None
88     for k in k_list:
89         knn_label = train_y[sorted_index_matrix[:, :k]]
90         label_sum_matrix = None
```

```
91          for i in range(10):
92              predict = np.sum(np.where(knn_label == i, 1, 0), axis=1)[:, None]
93              if label_sum_matrix is None:
94                  label_sum_matrix = predict
95              else:
96                  label_sum_matrix = np.concatenate(
97                      [label_sum_matrix, predict],
98                      axis=1)
99          if ret_matrix is None:
100             ret_matrix = np.argmax(label_sum_matrix, axis=1)[:, None]
101         else:
102             ret_matrix = np.concatenate([
103                 ret_matrix,
104                 np.argmax(label_sum_matrix, axis=1)[:, None]
105                 ], axis=1)
106     #asert ret_matrix.shape == (len(test_x), len(k_list))
107     return ret_matrix
108
109
110 def train(train_X, train_y, k_list, save_memory=False):
111     n_split = len(train_y)
112     n_corrects_list = []
113     for i in range(n_split):
114         train_X_train, train_y_train, train_X_valid, train_y_valid =
        make_train_data(
115             train_X, train_y, n_split, i
116             )
117         y_preds = knn(
118             train_X_train, train_y_train, train_X_valid,
119             k_list, save_memory=save_memory
120             )
121         result = (y_preds == train_y_valid[:, np.newaxis])
122         n_corrects = result.astype(int).sum(axis=0)
123         n_corrects_list.append(n_corrects)
124     n_corrects_list = np.array(n_corrects_list)
125     n_corrects = n_corrects_list.mean(axis=0)
126     return n_corrects
127
128
129 def test(train_X, train_y, test_X, test_y, k, labels):
130     n_label = len(labels)
131     confusion_matrix = np.zeros((n_label, n_label), dtype=int)
132     n_data_all = len(test_y)
133     result = {}
134     print('Test')
135
136     preds_all = knn(train_X, train_y, test_X, [k]).reshape(n_data_all)
```

```
    #result = (preds_all == test_y)
    #n_corrects = result.sum(axis=0)

    for label in labels:
        print(f'Label: {label}\t', end='')

        indices = np.where(test_y == label)[-1]
        n_data = len(indices)
        preds = preds_all[indices]

        # make confusion matrix
        for i in labels:
            n = (preds == i).sum()
            confusion_matrix[label, i] = n

        # calc accuracy
        n_correct = confusion_matrix[label, label]
        acc = n_correct / n_data
        print(f'#Data: {n_data}\t#Correct: {n_correct}\tAcc: {acc:.3f}')

        result[label] = {
            'data': n_data,
            'correct': n_correct,
            'accuracy': acc,
            }
    result['confusion_matrix'] = confusion_matrix

    # overall score
    n_crr_all = np.diag(confusion_matrix).sum()
    acc_all = n_crr_all / n_data_all
    result['all'] = {
        'data': n_data_all,
        'correct': n_crr_all,
        'accuracy': acc_all,
        }
    print(f'All\t#Data: {n_data_all}\t#Correct: {n_crr_all}\tAcc:
    {acc_all:.3f}')
    print()
    print('Confusion Matrix:\n', confusion_matrix)
    print()
    return result


def print_result_in_TeX_tabular_format(result):
    labels = list(range(10))
    print('Scores')
    for label in labels:
```

```
183         print('{} & {} & {} & {:.3f} \\\\'.format(
184             label,
185             int(result[label]['data']),
186             int(result[label]['correct']),
187             result[label]['accuracy']
188             ))
189     print()
190     print('Confusion Matrix')
191     for i in labels:
192         print('{}    '.format(i), end='')
193         for j in labels:
194             print(' & {}'.format(int(result['confusion_matrix'][i, j])),
    end='')
195         print(' \\\\')
196     return
197 
198 
199 def main():
200     # settings
201     k_list = list(range(1, 11, 1))
202     np.random.seed(0)
203     print('Settings')
204     print(f'k Candidates: {k_list}\n')
205 
206     # load data
207     train_X, train_y, test_X, test_y, labels = load_data(
208         n_label=None, n_train=None, n_test=None,
209         )
210     _train_X, _train_y = shuffle(train_X, train_y)
211     _train_X, _train_y = split(_train_X, _train_y, n=10)
212 
213     # train
214     print('Train')
215     n_corrects = train(_train_X, _train_y, k_list, save_memory=True)
216     print(f'#Correct: {n_corrects}')
217     k_best = np.argmax(n_corrects) + 1
218     print(f'Best k: {k_best}\n')
219 
220     # test
221     result = test(train_X, train_y, test_X, test_y, k_best, labels)
222     print_result_in_TeX_tabular_format(result)
223 
224 
225 if __name__ == '__main__':
226     main()
```