統計的機械学習
第六回　レポート

37-196360　森田涼介

2019 年 5 月 27 日

宿題 1

ガウスカーネルに対するカーネル密度推定法を行う。バンド幅は尤度交差確認法によって決定する。

結果を以下の表 4 と図 1 に示した。表 4 より，最も尤度の平均の大きいバンド幅は 0.1 となった。なお，プログラムは 4 ページの Listing 1 に示した。

表 1: バンド幅とそれに対応する LCV の値

h	0.01	0.05	0.10	0.50
LCV	-6344	-4009	-3938	-4219

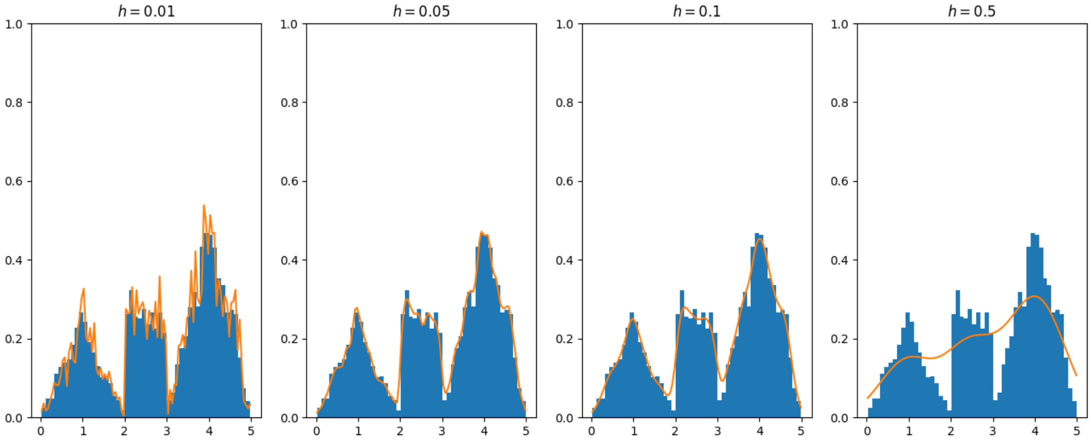

図 1: 各バンド幅に対するヒストグラムと確率密度

宿題 2

最近傍識別器によって 0–9 までの 10 クラスの手書き文字認識を行う。近傍数 k は識別誤差に関する交差確認により決定する。

以下に結果を示す。近傍数 k の候補として，1–10 を考えた。各 k についての，交差確認法で求めた正解数の平均値を表 2 に示す。これより，最も高い正解数を与える k を選ぶと，$k = 3$ となる。$k = 3$ に対する混同行列は表 3 のようになり，また，各カテゴリごとの正解率等は表 4 のようになった。

プログラムは 8 ページの Listing 2 に示した。

表 2: 各 k についての，交差確認法で求めた正解数の平均値

1	2	3	4	5	6	7	8	9	10
484.7	483.2	485.1	484.8	483.4	482.7	481.6	481.2	480.0	479.7

表 3: 混同行列

	0	1	2	3	4	5	6	7	8	9
0	198	0	1	0	0	0	1	0	0	0
1	0	199	1	0	0	0	0	0	0	0
2	0	0	195	0	0	0	0	2	3	0
3	0	0	0	190	0	4	0	1	4	1
4	0	1	0	0	189	0	3	0	0	7
5	2	0	3	3	1	186	0	0	1	4
6	1	0	2	0	0	0	197	0	0	0
7	0	1	0	0	4	0	0	191	0	4
8	2	0	2	3	0	3	0	0	187	3
9	0	0	0	0	2	0	0	0	0	198

表 4: 各カテゴリごとの結果

Category	#Data	#Correct	Accuracy
0	200	198	0.990
1	200	199	0.995
2	200	195	0.975
3	200	190	0.950
4	200	189	0.945
5	200	186	0.930
6	200	197	0.985
7	200	191	0.955
8	200	187	0.935
9	200	198	0.990
All	2,000	1,930	0.965

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 5 に示す。

表 5: 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)
    #plt.show()


    # train
    lcv_list = estimate_kernel_density(
        x, n_split, h_list, gauss_kernel,
        offset=offset, num=num,
        path=fig_path,
        )

    # result
    form = ’{:5.2f}’
    tab = ’  ’
    string_h = ’{:3}’.format(’h’)
    string_lcv = ’{:3}’.format(’LCV’)
    for h, lcv in zip(h_list, lcv_list):
        string_h += tab + form.format(h)
        string_lcv += tab + form.format(lcv)
    result = string_h + ’\n’ + string_lcv
    print(result)


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

Listings 2: assignment2.py

```python
# -*- 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

```

```

def split(data_X, data_y, n):
    n_data = len(data_y)
    n_data_split = n_data // n
    n_abandoned = n_data % n
    if n_abandoned != 0:
        print(f’Warning: {n_abandoned} samples are abandoned’)
    data_X_split = [data_X[i:i+n_data_split] for i in range(0, n_data, n_data_split)]
    data_y_split = [data_y[i:i+n_data_split] for i in range(0, n_data, n_data_split)]
    return data_X_split, data_y_split


def make_train_data(train_X, train_y, n_split, i):
    train_X_valid = train_X[i]
    train_y_valid = train_y[i]
    train_X_train = []
    train_y_train = []
    for j in range(n_split):
        if j != i:
            train_X_train.extend(train_X[j])
            train_y_train.extend(train_y[j])
    train_X_train = np.array(train_X_train)
    train_y_train = np.array(train_y_train)
    return train_X_train, train_y_train, train_X_valid, train_y_valid


def knn(train_X, train_y, test_X, k_list, save_memory=False):
    if save_memory:
        n_train = train_X.shape[0]
        n_test = test_X.shape[0]
        dist_matrix = np.zeros((n_test, n_train))
        for i in range(n_test):
            test_X_i = test_X[i]
            dist_matrix[i, :] = np.sum((train_X - test_X_i[np.newaxis, :])**2, axis=1)
    else:
        dist_matrix = np.sqrt(
            np.sum((train_X[None] - test_X[:, None])**2, axis=2)
            )

    sorted_index_matrix = np.argsort(dist_matrix, axis=1)
    ret_matrix = None
    for k in k_list:
        knn_label = train_y[sorted_index_matrix[:, :k]]
        label_sum_matrix = None
        for i in range(10):
            predict = np.sum(np.where(knn_label == i, 1, 0), axis=1)[:, None]
            if label_sum_matrix is None:
                label_sum_matrix = predict
            else:
                label_sum_matrix = np.concatenate(
                    [label_sum_matrix, predict],
                    axis=1)
        if ret_matrix is None:
            ret_matrix = np.argmax(label_sum_matrix, axis=1)[:, None]
        else:
            ret_matrix = np.concatenate([
                ret_matrix,
                np.argmax(label_sum_matrix, axis=1)[:, None]
                ], axis=1)
    #asert ret_matrix.shape == (len(test_x), len(k_list))
    return ret_matrix


def train(train_X, train_y, k_list, save_memory=False):
    n_split = len(train_y)
    n_corrects_list = []
    for i in range(n_split):
        train_X_train, train_y_train, train_X_valid, train_y_valid = make_train_data(
            train_X, train_y, n_split, i
            )
        y_preds = knn(
            train_X_train, train_y_train, train_X_valid,
            k_list, save_memory=save_memory
            )
        result = (y_preds == train_y_valid[:, np.newaxis])
        n_corrects = result.astype(int).sum(axis=0)
        n_corrects_list.append(n_corrects)
    n_corrects_list = np.array(n_corrects_list)
    n_corrects = n_corrects_list.mean(axis=0)
    return n_corrects


def test(train_X, train_y, test_X, test_y, k, labels):
    n_label = len(labels)
    confusion_matrix = np.zeros((n_label, n_label), dtype=int)
    n_data_all = len(test_y)
    result = {}
    print(’Test’)

    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:
        print(’{} & {} & {} & {:.3f} \\\\’.format(
            label,
            int(result[label][’data’]),
            int(result[label][’correct’]),
            result[label][’accuracy’]
            ))
    print()
    print(’Confusion Matrix’)
    for i in labels:
        print(’{}    ’.format(i), end=’’)
        for j in labels:
            print(’ & {}’.format(int(result[’confusion_matrix’][i, j])), end=’’)
        print(’ \\\\’)
    return


def main():
    # settings
    k_list = list(range(1, 11, 1))
    np.random.seed(0)
    print(’Settings’)
    print(f’k Candidates: {k_list}\n’)

    # load data
    train_X, train_y, test_X, test_y, labels = load_data(
        n_label=None, n_train=None, n_test=None,
        )
    _train_X, _train_y = shuffle(train_X, train_y)
    _train_X, _train_y = split(_train_X, _train_y, n=10)

    # train
    print(’Train’)
    n_corrects = train(_train_X, _train_y, k_list, save_memory=True)
    print(f’#Correct: {n_corrects}’)
    k_best = np.argmax(n_corrects) + 1
    print(f’Best k: {k_best}\n’)

    # test
    result = test(train_X, train_y, test_X, test_y, k_best, labels)
    print_result_in_TeX_tabular_format(result)


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