# 統計的機械学習レポート

工学部電子情報工学科 3 年　03190449 堀 紡希

5 月 21 日

宿題 1 Python で与えられた数の分布に対するガウスカーネルによるカーネル密度推定法を実装した。

```
import numpy as np

np.random.seed(0)

def data_generate(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
import math
def K(x):
    return math.exp(-0.5*x**2)/(2*math.pi)**(1/2)
def gauss(X, h):
    m = min(X)-2
    M = max(X)+2
    n = int((M - m)/h)
    p = []
    r = []
    for i in range(0, n):
        xi = float(m + i*h + h/2)
        sum = 0
        for x in X:
            sum += K((x-xi)/h)
        sum /= n*h
        p.append(sum)
        r.append(xi)
```

```
    P = 0
    for a in p:
        P += a*h
    q = []
    for b in p:
        q.append(b/P)
    return q, r
def cross(gauss, t, X, h):
    n = len(X)
    LCV = 0
    T = []
    for i in range(0, t):#t 個に分ける
        T1 = X[int(i*n/t): int(n*(i+1)/t)]
        T.append(T1)
    for i in range(0, t):
        train = []
        for j in range(0, t):#T[i] 以外のを推定用に
            if i != j:
                for r1 in T[j]:
                    train.append(r1)
        p, r = gauss(train, h)
        sum1 = 0
        for x in T[i]:
            if x < min(r):
                sum1 += np.log(p[0])
            elif x>max(r):
                sum1 += np.log(p[len(p)-1])
            else:
                for k in range(0, len(r)):
                    if r[k] > x:
                        l = k
                        break
                sum1 += np.log(p[l])
        sum1 /= len(T[i])
        LCV += sum1
    return LCV/t

X = data_generate()
plt.hist(X, range = (-2, 7), bins = 100)
plt.show()
c = []
H = [0.01, 0.03, 0.05, 0.07, 0.09, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
for h in H:
    p, x = gauss(X, h)
    plt.plot(x, p)
    c.append(cross(gauss, 5, X, h))
    print('h=' + str(h))
    plt.savefig("gaussbest.png")
    plt.show()
plt.plot(H, c)
```

```
plt.show()
```

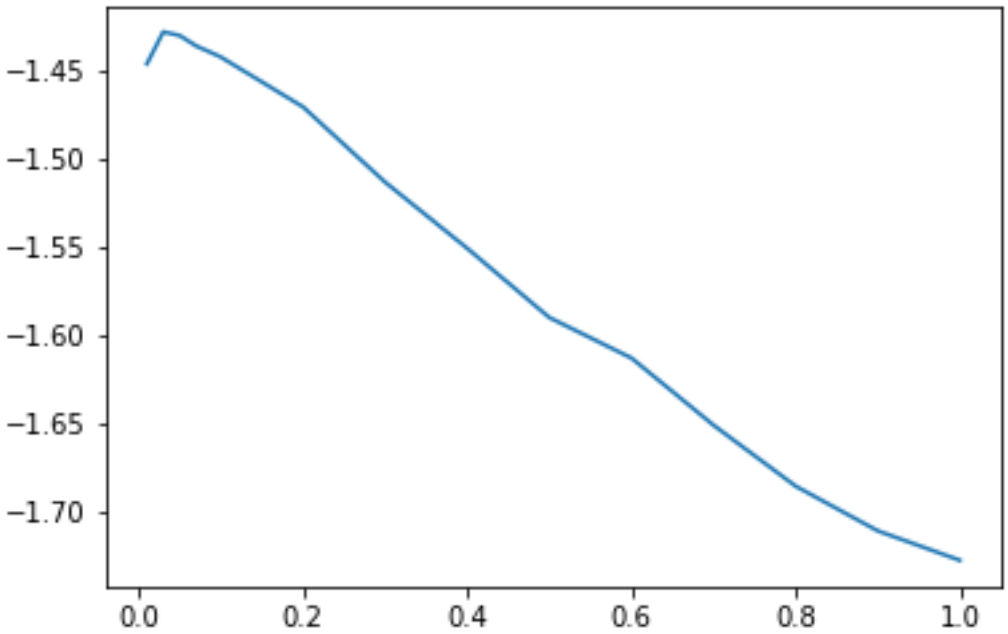


Figure 1: 幅 h を変えた時の尤度の変化

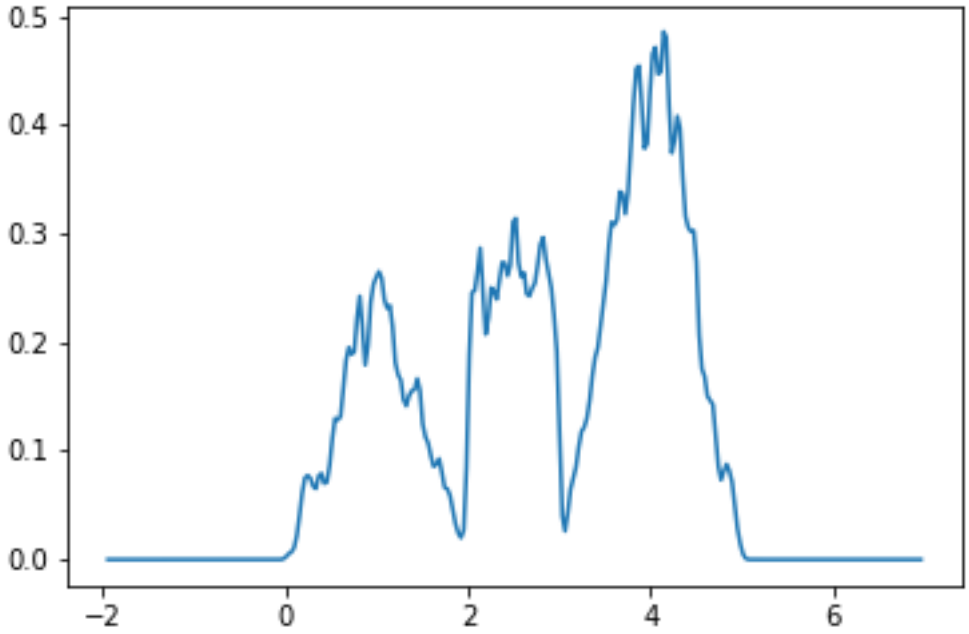


Figure 2: 最も尤度が高い (h=0.03) 確率分布

結果 実装するとこのようになった。h=0.03 の時細かすぎず大雑把すぎない良い分布の予測ができていた。

宿題 2 Python で最近傍識別器による手書き文字認識を実装した。なおサンプルコードの使い方がわからなかったので knn は自前で実装した。k を 1 から 11 の奇数として k に対するそれぞれの正解率を計算した。

```
\%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat

data = loadmat('digit.mat')
train = data['X']
test = data['T']

X = []
tag = []
for i in range(0, 10):
    for j in range(0, 500):
        X.append(train[:,j,i])
        if i == 9:
            tag.append(0)
```

```
        else:
            tag.append(i+1)

Test = []
ans = []
for i in range(0, 10):
    for j in range(0, 200):
        Test.append(test[:,j, i])
        if i ==9:
            ans.append(0)
        else:
            ans.append(i+1)

def myknn(train_x, train_y, test, k):
    X = np.array(train_x)
    y = np.array(train_y)
    test = np.array(test)
    dist = np.array([])
    l = np.array([])
    for x in X:
        s = np.sqrt(np.sum((x-test)**2))
        dist = np.append(dist, s)
    K = np.max(dist)
    for i in range(1, k+1):
        a = np.argmin(dist)
        l = np.append(l, y[a])
        dist[a]=K
    L = [0]*10
    for a in l:
        a = int(a)
        L[a] += 1
    return np.argmax(L)

k = 5
k_list = [2*i + 1 for i in range(0, k+1)]
print(k_list)
score = []
for x in k_list:
    correct = 0
    miss = 0
    for i in range(0, len(Test)):
        if myknn(X, tag, Test[i], x) == ans[i]:
            correct += 1
        else:
            miss += 1
    print(correct/(correct+miss))
    score.append(correct/(correct+miss))
plt.plot(k_list, score)
plt.show()
```

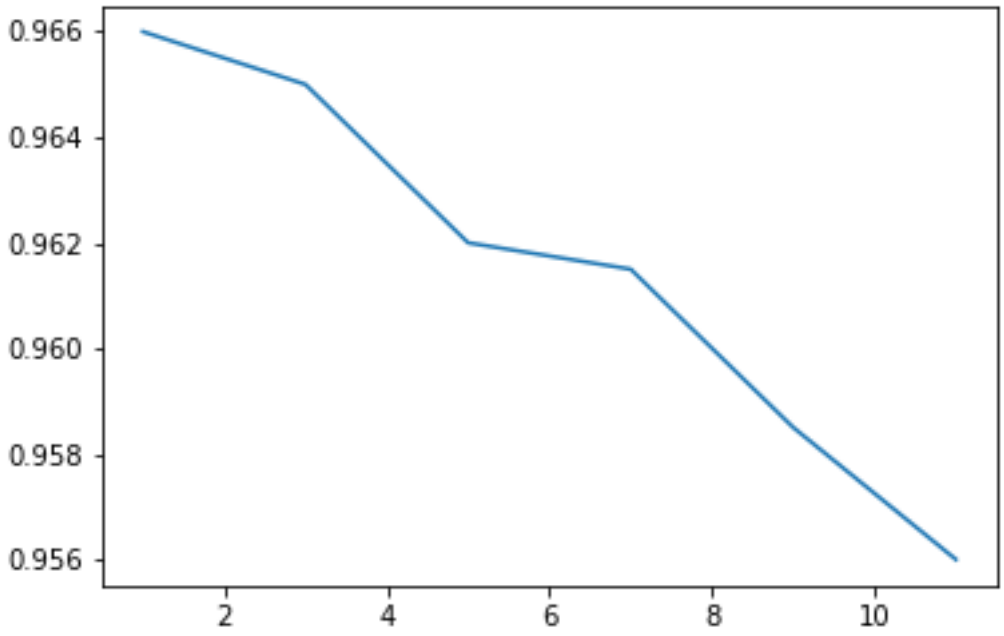


Figure 3: k に対する k-近傍識別器の正解率

結果 実装するとこのようになった。k=1 の時正解率が最大となり、k が大きくなるほど正解率は下がっていった。

また実行に非常に時間がかかったが、これは myknn の実装が同じような計算を何度も繰り返すようになっていて効率が悪かったためであると考えられる。