Homework 5

Jingwei Zhang 201528013229095

2015-12-26

1 Problem 1

1. 对一个问题，找一个弱的分类器，这个通常比较容易
2. 由由这个弱学习算法产生一组分类器，依次针对前面学习中比较难学的部分数据
3. 由这一组分类器投票来分类新数据.

2 Problem 2

① 将数据集分成k份.

② 将其中一份作为测试集，余下k-1份作训练样本训练σ，测试得到错误率

③ 对数据集的k份中每一份进行②得到k个σ，k个错误率，

④ 将所得σ结合起来（比如选最小错误率对应的那个）.

3 Problem 3

3. (4) $\mu_1=(2.5, 4.5)$ $\mu_2=(2.5, 0.5)$

则对w_1样本，$x_i-\mu_1$分别为 $(1.5, 0.5)$，$(-1.5, -0.5)$

对w_2样本，$x_i-\mu_2$为 $(-2.5, 0.5)$，$(2.5, -0.5)$

$$\therefore J_{e_1}=\|(1.5,0.5)\|^2+\|(-1.5,-0.5)\|^2+\|(2.5,0.5)\|^2+\|(2.5,-0.5)\|^2=18$$

$$S_{w_1}=2\begin{bmatrix}1.5^2 & 0.5\times1.5\\ 0.5\times1.5 & 0.5^2\end{bmatrix}+2\begin{bmatrix}2.5^2 & -2.5\times0.5\\ -2.5\times0.5 & 0.5^2\end{bmatrix}=\begin{bmatrix}17 & -1\\ -1 & 1\end{bmatrix}$$

$$|S_{w_1}|=16$$

同理，对第2种划分，$\mu_1=(4.5, 2.5)$ $\mu_2=(0.5, 2.5)$

$J_{e_2}=18$，$S_{w_2}=\begin{bmatrix}1 & -1\\ -1 & 17\end{bmatrix}$ $|S_{w_2}|=16$

对第3种划分 $\mu_1=(\frac{5}{3}, \frac{10}{3})$，$\mu_2=(5, 0)$

$J_{e_3}=\frac{52}{3}\approx17.3$，$S_{w_3}=\frac{1}{3}\begin{bmatrix}26 & 22\\ 22 & 26\end{bmatrix}$ $|S_{w_3}|=\frac{64}{3}\approx21.3$

$\therefore$ 对平方误差 $J_{e_3}<J_{e_2}=J_{e_1}$，故第3种划分最好

对类内散度矩阵行列式最小，$|S_{w_1}|=|S_{w_2}|<|S_{w_3}|$，故1，2种划分较好.

4 Programming Problem 1

4.1 Result

ClasturNo.	*center*	*Numberofsamples*
1	200	(5.93835690, 4.47548968)
2	201	(5.47541305, −4.52601645)
3	200	(1.02155866, −0.93527395)
4	199	(1.07159471, 4.01702333)
5	200	(9.00870491, −0.04025460)

4.2 Code

```
#!/usr/bin/python
# coding=utf-8
import numpy as np
```

```
import matplotlib.pyplot as plt

Sigma = [[1, 0], [0, 1]]
gen_means = [[1, -1], [5.5, -4.5], [1, 4], [6, 4.5], [9, 0]]
EPS = 1E-4

def gen_points():
    x, y = [],[]
    for mean in gen_means:
        x[len(x):len(x)], y[len(y):len(y)] =
            np.random.multivariate_normal(mean, Sigma, 200).T
    return x,y

def get_samples(x, y):
    samples = []
    for i in range(len(x)):
        samples.append([x[i], y[i]])
    return samples

def get_rand_mean(x, y, size):
    x_min = np.min(x)
    x_max = np.max(x)
    y_min = np.min(y)
    y_max = np.max(y)
    means = np.random.random((size, 2))
    # print(means)
    for xy in means:
        xy[0] = (x_max - x_min) * xy[0] + x_min
        xy[1] = (y_max - y_min) * xy[1] + y_min
    return means

def vector_add(a, b):
    c = []
    c[0:0] = a
    for i in range(len(a)):
        c[i] += b[i]
    return c

def vector_add_inplace(a, b):
    for i in range(len(a)):
        a[i] += b[i]
    return a

def vector_sub(a, b):
    c = []
    c[0:0] = a
    for i in range(len(a)):
        c[i] -= b[i]
    return c

def k_means(samples, means):
    # print(sums, cnt)
    changed = True
    # str = ""
    while changed:
        sums = []
```

```
        for i in range(len(means)):
            sums.append([0, 0])
        cnt = [0] * len(means)
        for sample in samples:
            mi = 0
            min_l = 1E10
            for i in range(len(means)):
                dif = np.array(vector_sub(sample, means[i]))
                l = np.sqrt(dif.dot(dif))
                if l < min_l:
                    mi = i
                    min_l = l
            vector_add_inplace(sums[mi], sample)
            cnt[mi] += 1
        changed = False
        print("cnt:", cnt)
        # print("sum:", np.array(sums))
        for i in range(len(means)):
            if cnt[i] > 0:
                for j in range(len(sums[i])):
                    sums[i][j] /= cnt[i]
            if np.max(np.abs(vector_sub(means[i], sums[i]))) > EPS:
                changed = True
            means[i] = sums[i]
        # print("means:", np.array(means))
    return means

def rand_k_mins(x, y, samples, times):
    for cnt in range(times):
        means = get_rand_mean(x, y, 5)
        print(means)
        new_means = k_means(samples, means)
        print(new_means)

if __name__ == '__main__':
    x, y = gen_points()
    # plt.scatter(x, y)
    # plt.show()
    samples = get_samples(x, y)
    rand_k_mins(x, y, samples, 1)
    # means = gen_means
    # new_means = k_means(samples, means)
    # print(new_means)
    plt.scatter(x, y)
    plt.show()
```

5 Programming Problem 2

5.1 Result

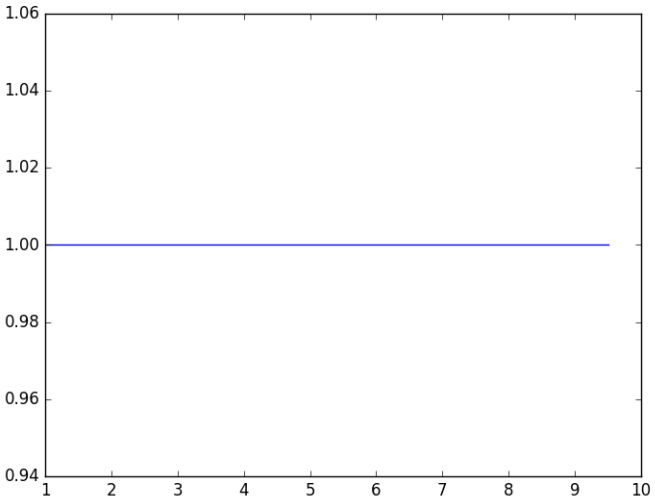

Figure 1: Clustering Accuracy when σ changes

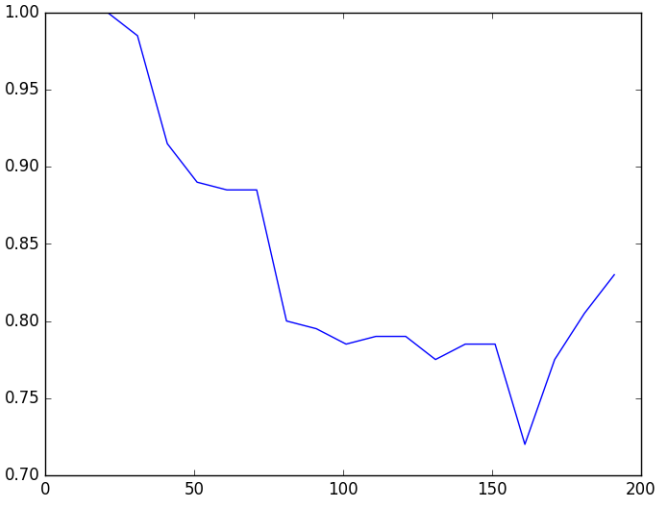

Figure 2: Clustering Accuracy when k changes

5.2 Code

```
#!/usr/bin/python
# coding=utf-8
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA

data_file_path = './PP2.data'
```

```
sigma2 = 1**2
k_nearest = 5
k_eigen_vectors = 2
EPS = 1E-9
num_of_clusters = 2


def get_samples():
    samples = []
    with open(data_file_path, mode="r") as in_file:
        for line in in_file:
            row = line.split()
            xs = [float(a) for a in row]
            samples.append(xs)
    return samples

def get_wij(pa, pb):
    d2 = np.sum([(pa[i] - pb[i])**2 for i in range(len(pa))])
    return np.exp(-d2 / 2 / sigma2)

def get_wij(d2):
    return np.exp(-d2 / 2 / sigma2)

def build_graph(samples):
    S = np.asarray(samples)
    N = len(samples)
    G = np.zeros([N, N])
    dis = np.zeros(N)
    for r in range(N):
        for c in range(N):
            if r == c:
                dis[c] = 0
            else:
                v = np.subtract(S[r], S[c])
                dis[c] = np.sum([x**2 for x in v])
        indexes = np.argsort(dis)
        k = k_nearest
        for i in indexes[1:k+1]:
            G[r][i] = get_wij(dis[i])
    return G

def compute_L_sym(W):
    W_mat = np.asmatrix(W)
    W_mat = (W_mat.T + W) / 2
    D_half_rev = np.asmatrix(np.zeros(W.shape))
    # print(D_half_rev)
    for r in range(len(D_half_rev)):
        s = np.sum(W_mat[r])
        # print(D_half_rev[r, r])
        D_half_rev[r, r] = 1 / np.sqrt(s)
    # print(D_half_rev)
    I = np.identity(len(W_mat))
    L_sym = I - D_half_rev * W_mat * D_half_rev
    return L_sym

def normalize(v):
```

```
    s = v * v.T
    v /= np.sqrt(s)
    return v

def compute_Accu(omega):
    dic = {}
    for i in range(num_of_clusters):
        dic[i] = 0
    for i in range(0, 100):
        dic[omega[i]] += 1
    n1 = max(dic.items(), key=lambda x: x[1])
    for i in range(num_of_clusters):
        dic[i] = 0
    for i in range(100, 200):
        dic[omega[i]] += 1
    n2 = max(dic.items(), key=lambda x: x[1])
    Accu = (n1[1] + n2[1]) / 200
    print(n1, n2, Accu)
    return Accu

def spectral_clustering(W):
    L_s = compute_L_sym(W)
    w, v = LA.eigh(L_s)
    f = 0  # first none-zero eigen column
    while np.abs(w[f]) < EPS:
        f += 1
    U = v[:, f:f + k_eigen_vectors]
    # Normalizing
    for row in U:
        normalize(row)
    # print("U: ", U)
    Accu = 0
    for cnt in range(5):
        omega = k_means(U, num_of_clusters)
        t = compute_Accu(omega)
        Accu = max(Accu, t)
    return Accu

def dis2(a, b):
    d = np.subtract(a, b)
    return d * d.T

def k_means(samples, clusters):
    N = clusters
    S = np.asmatrix(samples)
    means = 2 * np.random.random(S.shape[1] * N) - 1
    means = means.reshape([clusters, S.shape[1]])
    # print("initial means: ", means)
    changed = True
    omega = np.zeros(S.shape[0])
    while changed:
        changed = False
        sums = np.asmatrix(np.zeros(means.shape))
        cnt = np.zeros(means.shape[0])
        for s_i in range(len(S)):
            x = S[s_i]
```

```
            mean_i = np.argmin([dis2(x, mean) for mean in means])
            np.add(sums[mean_i], x, sums[mean_i])
            omega[s_i] = mean_i
            cnt[mean_i] += 1
        # print("cnt: ", cnt)
        for i in range(len(sums)):
            sums[i] /= cnt[i]
            diff = np.subtract(sums[i], means[i])
            if np.max(np.abs(diff)) > EPS:
                changed = True
            means[i] = sums[i]
        # print(means)
        # input()
    return omega

def frange(x, y, jump):
  while x < y:
    yield x
    x += jump

if __name__ == '__main__':
    samples = get_samples()
    # When sigma changes
    Accu = []
    x = []
    for sig in frange(1, 10, 0.5):
        x.append(sig)
        sigma2 = sig**2
        W = build_graph(samples)
        Accu.append(spectral_clustering(W))
    plt.plot(x, Accu)
    plt.show()

    # When k changes
    Accu = []
    x = []
    sigma2 = 1
    for k in range(1, 200, 10):
        x.append(k)
        k_nearest = k
        W = build_graph(samples)
        Accu.append(spectral_clustering(W))
    plt.plot(x, Accu)
    plt.show()
```