模式识别课后作业五

2020E8013282019 李一帆

January 8, 2021

1 简述题

1.1 问题 1

问题描述：请简述 adaboost 算法的设计思想和主要计算步骤。

解答：

设计思想：Adaboost 从弱学习算法出发，得到一系列弱分类器；然后通过组合这些弱分类器构成一个强分类器。通过改变训练数据的概率分布，针对不同的训练数据的分布，调用弱学习算法来学习一些列分类器。在改变训练数据的权值分布上，Adaboost 通过提高那些被前一轮弱分类器分错的样本的权重，降低已经被正确分类的样本的权重。错分的样本将在下一轮弱分类器中得到更多的关注。于是分类问题被一系列弱分类器"分而治之"。在如何将一系列的弱分类器组合成一个强分类器方面，Adaboost 采用加权（多数）表决的方法。加大分类错误率较小的弱分类器的权重，使其在表决中起更大的作用。

计算步骤：输入为给定两类训练数据集 $T=\{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\}$。

Step1：初始化训练数据的权值分布 $D_1=\{w_{11}\},\{w_{12}\},...,\{w_{1n}\},w_{1i}=1/n,i=1,...,n$。

Step2：对 $m=1,2,...,M$

（a）使用具有权值分布 D_m 的训练数据，学习基本分类器 $G_m(x):X\to\{-1,+1\}$。

（b）计算 $G_m(x)$ 在训练数据集上的分类错误率（加权）：

$$e_m=P(G_m(x_i)\neq y_i)=\sum_{i=1}^{n}w_{mi}I(G_m(x_i)\neq y_i)$$

（c）计算 $G_m(x)$ 的贡献系数：

$$\alpha_m=\frac{1}{2}ln\frac{1-e_m}{e_m}$$

α_m 表示 $G_m(x)$ 在最终的分类器中的重要性。当 $e_m\leq 0.5$ 时，$\alpha_m\geq 0$。同时，α_m 将随着 e_m 的减小而增大。所以分类误差率越小的基本分类器在最终分类器的作用越大。

（d）更新训练数据集的权重分布：

$$D_{m+1}=\{w_{m+1,1},w_{m+1,2},...,w_{m+1,n}\}$$

具体计算如下：

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \times \begin{cases} \exp(-\alpha_m),\ if\ G_m(\mathrm{x}_i) = y_i \\ \exp(\alpha_m),\ if\ G_m(\mathrm{x}_i) \neq y_i \end{cases}$$
$$= \frac{w_{mi}}{Z_m} \times \exp(-\alpha_m y_i G_m(\mathrm{x}_i))$$

其中，Z_m 是规范化因子，它使 D_{m+1} 成为一个概率分布。

$$Z_m = \sum_{i=1}^{n} w_{mi} \exp(-\alpha_m y_i G_m(x_i))$$

Step3：构建基本分类器的线性组合。

$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$$

对于两类分类问题，得到最终的分类器。

$$G(x) = sign(f(x)) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$$

1.2 问题 2

问题描述：请从混合高斯密度函数估计的角度，简述 k-means 聚类算法的原理（主要用文字描述）；请给出 k-means 聚类算法的计算步骤；请说明哪些因素回影响 k-means 算法的聚类性能。

解答：k-means 假设各个聚类出现的概率相等，每个样本以概率 1 属于每个聚类，样本属于哪一类需要计算 $\|x_k - \hat{\mu}_i\|^2$ 来判断。因此需要通过迭代来得到 c 个高斯成分的均值。

k-means 聚类的计算步骤如下所示。

Algorithm 1: k-means Algorithm

```
Function k-means(W, k):
    begin initialization n, c, μ1, μ2, ..., μc;
    while no change in μi do
        classify n samples according to nearest μi;
        re-compute μi;
    return μ1, μ2, ..., μc
```

影响 k-means 算法的聚类性能的因素主要有聚类数目 k，初始中心 $\boldsymbol{\mu_i}$ 的选取以及聚类数据的分布。

1.3 问题 3

问题描述：请简述谱聚类算法的原理，给出一种谱聚类算法（经典算法、Shi 算法和 Ng 算法之一）的计算步骤；请指出哪些因素回影响聚类的性能。

解答：从图切割的角度，谱聚类就是要找到一种合理的分割图的方法，分割后能形成若干个子图。连接不同子图的边的权重尽可能小，子图内部边权重尽可能大。Ng 聚类算法的原理如 Algorithm 1 所示。

Algorithm 2: Spectral Clustering-Ng Algorithm

Input: *similarity matrix* W

Input: k number of clusters

Output: $A_1, ..., A_k$ clusters

Function `NgSpectralCluster(`W, k`):`

compute $\boldsymbol{L_{sym}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$;

compute the first k eigenvectors $\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_k$ of $\boldsymbol{L}_{sym}$;

Let $U \in R^{n\times k}$ be the matrix containing the vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_k$, namely, $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_k] \in R^{n\times k}$;

$max\_time = 0$;

form the matrix $\boldsymbol{T} \in R^{n\times k}$ from $\boldsymbol{U}$ by normalizing the rows to norm 1, namely, set $t_{ij} = u_{ij}/\sqrt{\sum_{m=1}^{n} u_{im}^2}$;

for $i = 1, 2, ..., n$, let $y_i \in R^k$be the vector corresponding to the i-th row of $\boldsymbol{T}$;

cluster the points $\{y_i\}_{i=1,2,...,n}$ in R^k with k-means algorithm into clusters $A_1, A_2, ..., A_k$;

output $A_1, A_2, ..., A_k$

影响谱聚类性能的因素主要有一下几点。局部连接 k 近邻的取值以及 ϵ 半径。如果样本点数目很多，对大型矩阵进行特征值分解的计算仍然不稳定。最后采用 k-means 进行聚类，聚类数目也会影响聚类结果。

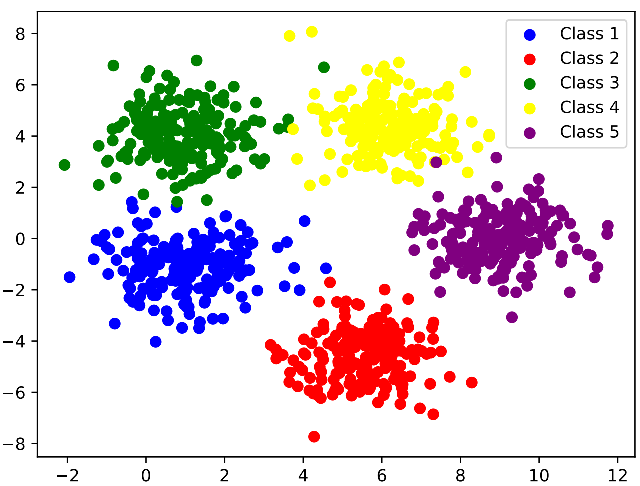

Figure 1: 聚类数据可视化.

2 计算机编程

2.1 问题 1

将数据点进行可视化后的结果如 Figure 1 所示。

问题描述：

（1）编写一个程序，实现经典的 k-means 聚类算法。

（2）令聚类个数等于 5，采用不同的初始值，报告聚类精度、以及最后获得的聚类中心。并计算所获得的聚类中心与对应的真实分布的均值之间的误差。

解答： k-means 聚类算法的代码如下所示。

```
def k_means(data, centers):
    '''
    K_means algorithm.
    :param data: The data that needs to be clustered.
    :param centers: The centers of cluster.
    :return: The cluster result and centers.
    '''
    N = data.shape[0]
    class_num = centers.shape[0]
    standard_ans = np.zeros(N)
    per_class = int(N / class_num)
    for i in range(class_num):
        standard_ans[i * per_class:(i + 1) * per_class] = i

    for step in range(10000):
        # caculate the distance between cluster centers and data
        dis = np.zeros([data.shape[0], class_num])
        for i in range(class_num):
            data_norm2 = np.linalg.norm(data - centers[i], ord=2,
               axis=1)
            dis[:, i] = data_norm2

        # find the min index of distance matrix
        result = np.argmin(dis, axis=1)
        centers_tmp = np.zeros(centers.shape, dtype='float')
        for i in range(class_num):
```

```
                class_inx = np.argwhere(result == i)
                if class_inx.all():
                    centers_tmp[i] = np.mean(data[class_inx], axis=0)
            error_matrix = (standard_ans - result)
            error_matrix[error_matrix < 0] = 1
            error_matrix[error_matrix > 0] = 1
            right_num = error_matrix.sum()
            print("Step %d , acc is : %f" % (step, (N - right_num) / N))
            if abs(centers - centers_tmp).sum() == 0:
                break
            else:
                centers = centers_tmp
        return result, centers
```

采用不同的初始值（随机选择每一聚类中数据点），最终的聚类精度为 98.4%，最终聚类中心与真实分布均值之间的误差如下所示。

1.00000000	1.00000000
0.01610427	0.00131740
0.09496927	0.09183495
0.16539437	0.08684510
0.01165332	0.01481903

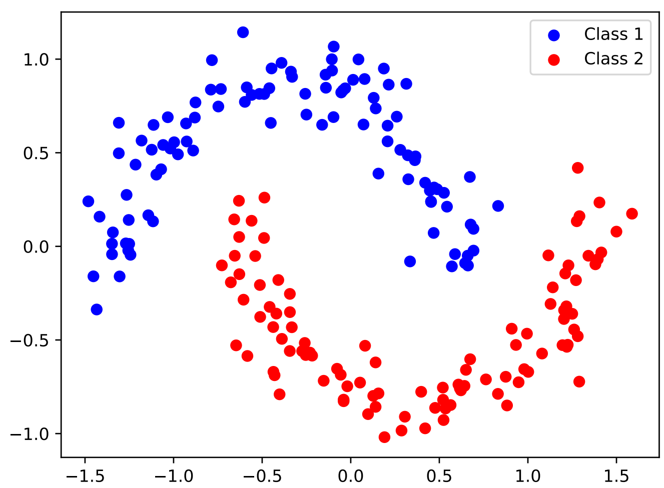

Figure 2: 谱聚类数据可视化.

2.2 问题 2

将数据点进行可视化后的结果如 Figure 2 所示。

问题描述：请编写一个程序，实现 Ng 谱聚类算法。

解答：Ng 算法代码如下所示。

```
def spectral_cluster(data, k=10, type='Ng', sim='classical', sigma=5, cluster_num=2):
N = data.shape[0]
W = np.zeros([N, N])
for i in range(N):
    dis = np.linalg.norm(data - data[i], ord=2, axis=1)
    dis = np.delete(dis, 0)
    k_idx = np.argsort(dis)[: k]
    if sim == 'classical':
        W[i, k_idx] = 1
    else:
        W[i, k_idx] = np.exp(-(dis[k_idx] ** 2) / (2 * (sigma ** 2)))
if sim == 'classical':
    D = k * np.eye(N)
    # W = (W.T + W) / 2
    L = D - W
else:
    D_diag = np.sum(W, axis=0)
    D = np.diag(D_diag)
    W = (W.T + W) / 2
    L = D - W
if type == 'Ng':
    L_sys = ((D * (k ** (-0.5))).dot(L)).dot(D * (k ** (-0.5)))
    vals, vecs = np.linalg.eig(L_sys)
    indx = np.argsort(vals)[:cluster_num]
    U = vecs[:, indx]
    T = U / np.sqrt(np.sum(U ** 2, axis=0))
    centers = np.array([T[60], T[150]])
    result, centers, acc = k_means(T, centers)
return result, centers, acc
```

通过 k-means 聚类和谱聚类的结果如下所示。

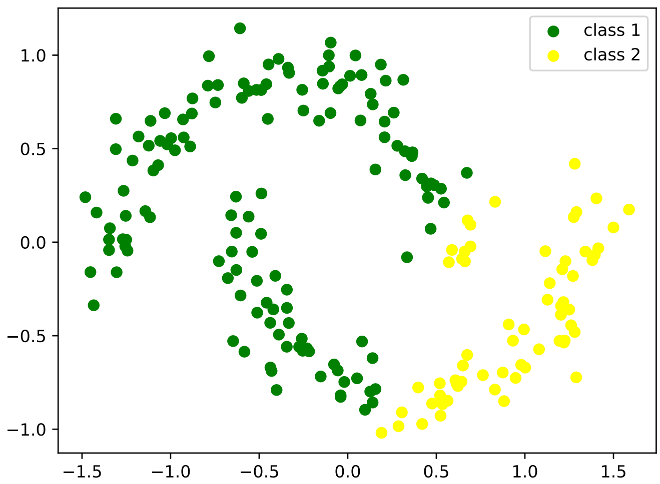

Figure 3: k-means 聚类结果可视化.

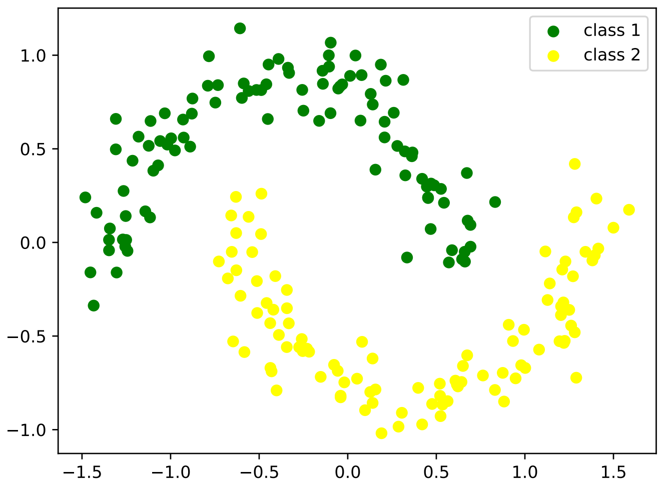

Figure 4: Ng 谱聚类结果可视化.

由上图可以看出，k-means 对于这种分布不服从高斯类型的聚类效果不是很好，而谱聚类则可以很好的解决这个问题。

问题描述：设点对亲和性（即边权值）采用如下计算公式。

$$w_{ij} = \exp\left(-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}\right)$$

数据图采用 k-近邻的方法来生成（即对于每个数据点 x_i，首先在所有样本中找出不包含 x_i 的 k 个最近邻的样本点，然后 x_i 与每个临近样本点均有一条边相连，从而完成图构造）。注

意，为了保证亲和度矩阵 W 是对称矩阵，可以令 $W = (W^T + W)/2$。假设已知前 100 个点为一个聚类，后 100 个点为一个聚类，请分析分别取不同的 σ 值和 k 值对聚类结果的影响。

解答：在固定 $sigma = 5$ 下，改变 k 可以得到的聚类精度变化曲线如 Figure 5 所示。在固定 $k = 10$ 下，改变 sigma 可以得到聚类精度变化曲线如 Figure 6 所示。

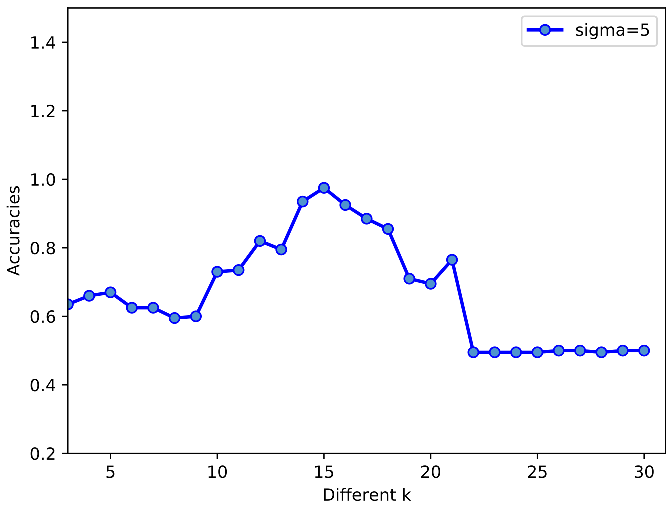

Figure 5: 聚类精度变化曲线（sigma=5，k 改变）.

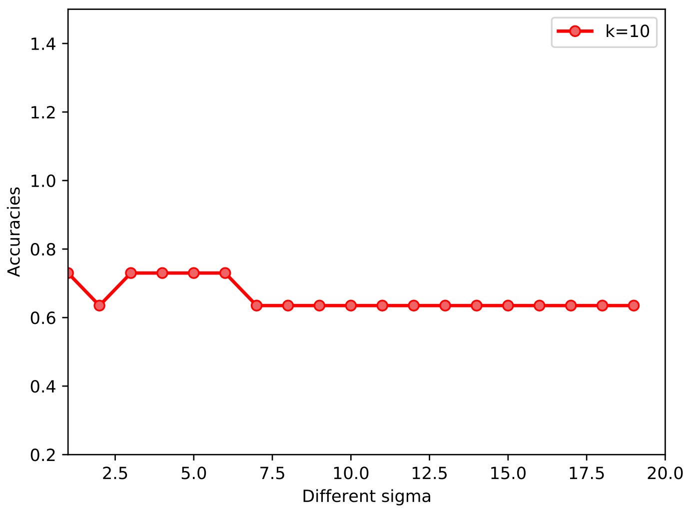

Figure 6: 聚类精度变化曲线（k=10，sigma 改变）.

由上图可以看出，在固定 sigma 的情况下，不同的 k 值会对聚类精度产生不同的影响，并且 k 值的选取往往并不是越大越好或是越小越好，最优取值是介于中间的某一个数。在固定 k 值的情况下，不同 sigma 的选取也会对聚类精度产生影响，但是影响不如 k 值那么明显。

A 附录——代码 (Python)

```python
import numpy as np
import matplotlib.pyplot as plt

data_name1 = 'data1.txt'
data_name2 = 'data2.txt'


def read_data(data_file):
    data = []
    split_string = ' '
    if data_file == 'data1.txt':
        split_string = '\t'
    elif data_file == 'data2.txt':
        split_string = ' '
    with open(data_file) as file:
        for line in file.readlines():
            data_line = np.array(line.split(split_string),
               dtype='float')
            data.append(data_line)
    data = np.array(data)
    return data


def k_means(data, centers, print_info=False):
    '''
    K_means algorithm.
    :param data: The data that needs to be clustered.
    :param centers: The centers of cluster.
    :return: The cluster result and centers.
    '''
    N = data.shape[0]
    class_num = centers.shape[0]
    standard_ans = np.zeros(N)
    per_class = int(N / class_num)
```

```
    for i in range(class_num):
        standard_ans[i * per_class:(i + 1) * per_class] = i

    for step in range(10000):
        # caculate the distance between cluster centers and data
        dis = np.zeros([data.shape[0], class_num])
        for i in range(class_num):
            data_norm2 = np.linalg.norm(data - centers[i], ord=2, axis=1)
            dis[:, i] = data_norm2

        # find the min index of distance matrix
        result = np.argmin(dis, axis=1)
        centers_tmp = np.zeros(centers.shape, dtype='float')
        for i in range(class_num):
            class_inx = np.argwhere(result == i)
            if class_inx.all():
                centers_tmp[i] = np.mean(data[class_inx], axis=0)
        error_matrix = (standard_ans - result)
        error_matrix[error_matrix < 0] = 1
        error_matrix[error_matrix > 0] = 1
        right_num = error_matrix.sum()
        acc = (N - right_num) / N
        if print_info:
            print("Step %d , acc is : %f" % (step, acc))
        if abs(centers - centers_tmp).sum() == 0:
            break
        else:
            centers = centers_tmp
    return result, centers, acc


def spectral_cluster(data, k=10, type='Ng', sim='classical', sigma=5, cluster_num=2):
```

```python
        N = data.shape[0]
        W = np.zeros([N, N])
        for i in range(N):
            dis = np.linalg.norm(data - data[i], ord=2, axis=1)
            dis = np.delete(dis, 0)
            k_idx = np.argsort(dis)[: k]
            if sim == 'classical':
                W[i, k_idx] = 1
            else:
                W[i, k_idx] = np.exp(-(dis[k_idx] ** 2) / (2 * (
                   sigma ** 2)))
        if sim == 'classical':
            D = k * np.eye(N)
            # W = (W.T + W) / 2
            L = D - W
        else:
            D_diag = np.sum(W, axis=0)
            D = np.diag(D_diag)
            W = (W.T + W) / 2
            L = D - W
        if type == 'Ng':
            L_sys = ((D * (k ** (-0.5))).dot(L)).dot(D * (k **
               (-0.5)))
            vals, vecs = np.linalg.eig(L_sys)
            indx = np.argsort(vals)[:cluster_num]
            U = vecs[:, indx]
            T = U / np.sqrt(np.sum(U ** 2, axis=0))
            centers = np.array([T[60], T[150]])
            result, centers, acc = k_means(T, centers)
        return result, centers, acc


    if __name__ == '__main__':
        # data1 = read_data(data_name1)
        #######True clusters#######
        # plt.figure()
```

```
        # plt.scatter(data1[:200, 0], data1[:200, 1], color='blue', label='Class 1')
        # plt.scatter(data1[200:400, 0], data1[200:400, 1], color='red', label='Class 2')
        # plt.scatter(data1[400:600, 0], data1[400:600, 1], color='green', label='Class 3')
        # plt.scatter(data1[600:800, 0], data1[600:800, 1], color='yellow', label='Class 4')
        # plt.scatter(data1[800:1000, 0], data1[800:1000, 1], color='purple', label='Class 5')
        # plt.legend()
        #######K-means#######
        # orig_clusters = np.array([data1[150], data1[350], data1[550], data1[750], data1[950]], dtype='float')
        # result, centers, acc = k_means(data1, centers=orig_clusters)
        # plt.figure()
        # colors = ['blue', 'red', 'green', 'yellow', 'purple']
        # classes = ['class 1', 'class 2', 'class 3', 'class 4', 'class 5']
        # for i in range(5):
        #     class_inx = np.argwhere(result == i)
        #     plt.scatter(data1[class_inx, 0], data1[class_inx, 1], color=colors[i], label=classes[i])
        # plt.legend()
        # centers_mean = np.array([[1, -1], [5.5, -4.5], [1, 4], [6, 4.5], [9, 0]])
        # error = abs(orig_clusters - centers_mean).sum()
        # print("Error between k-means centers and true centers is %f" % error)
        # print(abs(centers - centers_mean))
        #
        ##########True Clusters##########
        plt.figure()
        data2 = read_data(data_name2)
```

```python
    plt.scatter(data2[:100, 0], data2[:100, 1], color='blue', label='Class 1')
    plt.scatter(data2[100:200, 0], data2[100:200, 1], color='red', label='Class 2')
    plt.legend()
    # np.random.shuffle(data2)
    # orig_clusters2=np.array([data2[50], data2[150]])
    # result, centers = k_means(data2, centers=orig_clusters2)
    # result, centers, acc = spectral_cluster(data2, sim='others')
    # colors = ['green', 'yellow']
    # classes = ['class 1', 'class 2']
    # plt.figure()
    # for i in range(2):
    #     class_inx = np.argwhere(result == i)
    #     plt.scatter(data2[class_inx, 0], data2[class_inx, 1], color=colors[i], label=classes[i])
    # plt.legend()
    # plt.show()

    ########分析谱聚类参数的影响#######
    sigmas = range(1, 20)
    ks = range(3, 31)
    accuracies = np.zeros([len(sigmas), len(ks)])
    for sigma in sigmas:
        for k in ks:
            result, centers, acc = spectral_cluster(data2, sim='others' ,sigma=sigma, k=k)
            accuracies[sigma - 1, k - 3] = acc
    plt.figure()
    sigma = sigmas[5]
    plt.plot(ks, accuracies[sigma - 1], marker='o', mfc='#4F94CD', label='sigma=5', color='blue', linewidth=2)
    plt.xlim((3, 31))
    plt.ylim((0.2, 1.5))
    plt.xlabel('Different k')
```

```
        plt.ylabel('Accuracies')
        plt.legend()
        plt.figure()
        k = 10
        plt.plot(sigmas, accuracies[:, k - 3], marker='o', mfc='#EE6363', label='k=10', color='red', linewidth=2)
        plt.xlim((1, 20))
        plt.ylim((0.2, 1.5))
        plt.xlabel('Different sigma')
        plt.ylabel('Accuracies')
        plt.legend()
        plt.show()
```