Lab2：聚类与分类

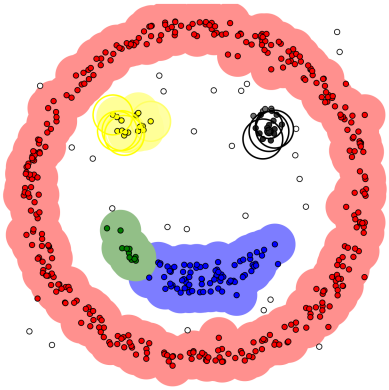
1 Clustering 聚类：发现数据点的相似性

1.1 理论

共同点：通过距离来判定相似性

可视化理解各种算法：网站

1. K-means：中心点为重心
   (a) 流程
      i. 先取k个中心点（需要设定k和位置）
      ii. 然后去遍历所有点，每个点都属于离他最近的中心点，这样就形成了很多簇
      iii. 接下来去计算这些簇的重心，以找到新的中心点
      iv. 接着重复b, c
   (b) 缺陷：可能陷入局部最优解；计算量比较大；需要指定k的数量（实操下来这个影响最大）；对初始点比较敏感；不利于非凸模型（就比如一个长条）
2. K-medoids：中心点为实体点，到类内距离之和最小的点
3. DBScan
   (a) 流程
      i. 先设定epsilon（每个圈的大小）和minPoints（每个圈内要有minPoints个点，否则就无法构成簇）
      ii. 以每个点为中心，以epsilon为半径画圆，如果满足minPoints要求，则为一个簇
      iii. 遍历每个点即可
      iv.



   (b) 优点：可以减少噪声影响，分类效果好
   (c) 缺陷：暴力求解复杂度高；对epsilon和minPoints有要求
4. 谱聚类：对于一个图，我们去找最小割边，使其变为两个子图。使子图内部相似度高，子图间相似度低

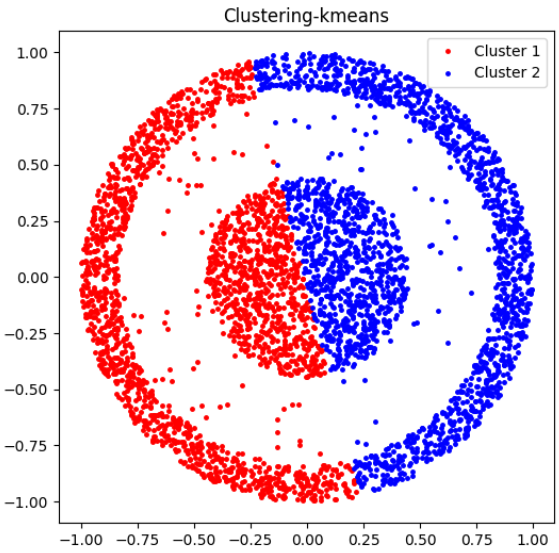
   （具体数学推导较复杂）

1.2 分类效果：

1.2.1 KMeans

如图所示，采用kmeans算法，最终的分类效果一般

此处采用的是2个中心点，由于kmeans的算法设计，使其注定不善于这种图形




```python
def kmeans(X, k):
    '''
    K-Means clustering algorithm

    Input:  x: data point features, N-by-P maxtirx
            k: the number of clusters

    Output:  idx: cluster label, N-by-1 vector
    '''

    N, P = X.shape # N 表示数据点的数量（行数）；P 表示每个数据点的特征数量（列数）
    idx = np.zeros(N) # 初始化 idx 为全零向量，表示每个数据点的初始簇标签

    # 选择中心
    centers = X[np.random.choice(N, k, replace=False)]  # 随机选择 k 个数据点作为初始中
心
    # 迭代：把每个样本分配到最近的中心，再重新找中心
    for _ in range(100):
        # 计算每个数据点到每个中心的距离，并将每个数据点分配到最近的中心
        dist = cdist(X, centers, 'euclidean')  # 计算每个数据点到每个中心的欧几里得距离
        idx = np.argmin(dist, axis=1)  # 将每个数据点分配到最近的中心
        print(idx)

        # 重新计算中心
        new_centers = np.array([X[idx == i].mean(axis=0) for i in range(k)])
        # 如果中心没有变化，则退出迭代
        if np.all(centers == new_centers):
            break
        centers = new_centers

    return idx
```

1.3 谱聚类

1.3.1 构造相似度图

- 将原始数据点构造成图结构，每个数据点是一个节点。
- 边的权重表示数据点之间的相似度，常用方法包括：**KNN 图**：只连接每个点与其最近的 k 个邻居。

1.3.2 构造拉普拉斯矩阵

```
def spectral(W, k):
    '''
    Spectral clustering algorithm

    Input:  W: Adjacency matrix, N-by-N matrix
            k: number of clusters

    Output:  idx: data point cluster labels, N-by-1 vector
    '''
    N = W.shape[0]

    # 计算度矩阵D
    D = np.diag(np.sum(W, axis=1))

    # 构造归一化拉普拉斯矩阵 D^{-1}L
    D_inv = inv(D)
    L_rw = D_inv @ (D - W)
```

1.3.3 特征值分解

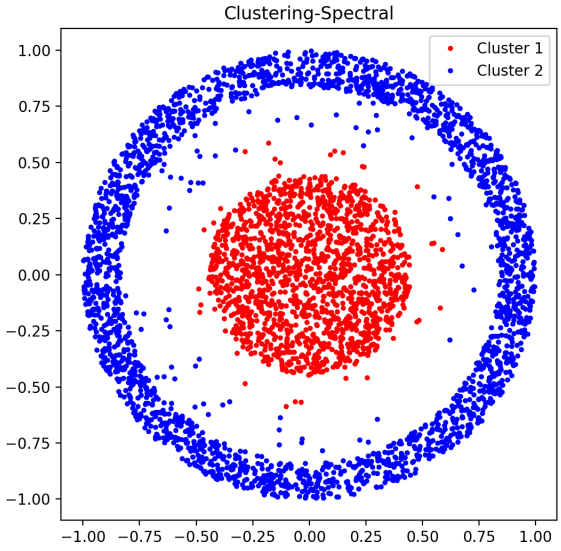
```
    # 计算特征值和特征向量
    eigenvalues, eigenvectors = eig(L_rw)

    # 取最小的k个特征值对应的特征向量
    idx = np.argsort(np.real(eigenvalues))[:k]

    # 构建K维特征表示
    X = np.real(eigenvectors[:, idx])
    X = X / np.linalg.norm(X, axis=1, keepdims=True)
    X = X.astype(float)
```

1.3.4 使用 KMeans 聚类

```
    # 进行K-means聚类
    idx = kmeans(X, k)
    return idx
```

1.4 实验结果

如图所示，由于谱聚类适用于非凸的图形，且对于初始点要求小，所以相比kmeans的聚类效果有明显提升



2 分类

2.1 理论

本实验使用的是感知机SVM

SVM 的核心思想是不仅要找到一个能正确分类样本的超平面，而且要找到**使两类之间间隔（margin）最大的那个超平面**。

- 间隔定义为两个类别中离超平面最近样本之间的距离；
- 这些最近的样本称为**支持向量（Support Vectors）**；
- 最大化间隔可以提高模型的泛化能力，减少过拟合风险。

2.2 实验过程

1. 首先划分训练集和测试集的比例，此处设为7:3

```
no_train = 70  # 70% 作为训练数据
no_test = 30   # 30% 作为测试数据
```

2. 编写计算训练的代码

```
def func(X, y):
    # 获取特征维度 P 和样本数量 N
    P, N = X.shape

    # 初始化权重参数 w，包括一个偏置项（bias）
    # 权重维度为 (P+1, 1)，其中 P+1 表示特征维度 + 偏置项
    w = np.zeros((P + 1, 1))

    # 将偏置项加入数据集 X，构造增广特征矩阵 X_bias
    # X_bias 的形状为 (P+1, N)，第一行为全1表示偏置项
    X_bias = np.vstack((np.ones((1, N)), X))

    # 感知机训练过程
```

```
    for i in range(100):  # 最大迭代次数为100
        mis_classified = False  # 标记本轮是否有误分类样本

        # 遍历所有训练样本
        for j in range(N):
            # 计算当前样本的预测值与真实标签的乘积
            # 若 <= 0，说明该样本被误分类，需要更新权重
            if y[0, j] * np.dot(w.T, X_bias[:, j:j+1]) <= 0:
                # 更新权重：w = w + y_j * x_j
                w += y[0, j] * X_bias[:, j:j+1]
                mis_classified = True  # 标记本轮有误分类样本

        # 如果本轮没有发生任何权重更新，说明已找到完美分类的超平面，提前结束训练
        if not mis_classified:
            break

    # 返回学习到的参数 w
    return w
```

3.测试错误率

```
# 开始多次实验，统计平均误差
for i in range(no_iter):
    # 生成数据：X 是特征矩阵，y 是标签，w_gt 是真实权重（用于绘图和比较）
    X, y, w_gt = gen_data(no_data)

    # 划分训练集与测试集
    X_train, X_test = X[:, :no_train], X[:, no_train:]  # 按列划分特征
    y_train, y_test = y[:, :no_train], y[:, no_train:]  # 按列划分标签

    # 使用训练数据调用分类算法学习参数 w_l（learned weights）
    w_l = func(X_train, y_train)

    # 在训练集和测试集上加入偏置项（即构造增广特征向量）
    X_train_bias = np.vstack((np.ones((1, no_train)), X_train))
    X_test_bias = np.vstack((np.ones((1, no_test)), X_test))

    # 计算预测结果（使用符号函数判断正负类）
    train_preds = np.sign(np.dot(w_l.T, X_train_bias))  # 训练集预测结果
    test_preds = np.sign(np.dot(w_l.T, X_test_bias))    # 测试集预测结果

    # 计算当前迭代的训练误差和测试误差
    train_err = np.mean(train_preds[0] != y_train[0])  # 判断是否与真实标签一致
    test_err = np.mean(test_preds[0] != y_test[0])

    # 累加误差以便后续求平均
    cumulative_train_err += train_err
    cumulative_test_err += test_err

# 计算平均训练误差和测试误差
train_err = cumulative_train_err / no_iter
```

```
    test_err = cumulative_test_err / no_iter

    # 可视化最终结果：原始数据、真实边界、学习到的边界
    plot(X, y, w_gt, w_l, "Classification")
```

2.3 实验结果

Training error: 0.001

Testing error: 0.019

分类效果如下图所示，分类效果很好