第六次作业

姓名：樊宇

学号：2021E8018782022

Part 1

Q1

设计思想：

从弱学习算法出发，反复学习，得到一系列弱分类器，然后组合这些弱分类器，构成一个强分类器。

在每轮训练中，提高被前一轮分类器分错的样本的权重，使其在下一轮分类中的重要性更高；

组合弱分类器时 ，提高分类错误率小的弱分类器，使其表达性更强。

计算步骤：

- 初始化权值分布

$$D_1 = \{w_{11}, w_{12}, \dots, w_{1n}\}$$
$$w_{1i} = 1/n, i = 1, 2, \dots, n$$

- $m = 1, 2, \dots, M$，对于M个弱分类器
  - 使用具有权值分布的训练样本学习分类器$G_m(x)$
  - 计算$G_m(x)$在训练数据上的加权分类错误率：

$$e_m = P(G_m(x_i) \neq y_i) = \sum_i w_{mi} I(G_m(x_i) \neq y_i)$$

  - 计算$G_m(x)$的贡献系数：

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

  - 更新训练数据集的权值分布：

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

- 构建基本分类器的线性组合：

$$f(x) = \sum_m a_m G_m(x)$$

- 分类器：

$$G(x) = sgn(f(x))$$

Q2

基本原理:

在GMM的基础之上

- 假设每个类别出现的概率都相等
- 不再以概率分布的形式给出当前样本属于每个类别的概率，而是直接将样本归为与其相似度最高的类别
- 直接用欧氏距离来计算样本与类别之间的相似度，欧式距离越小，相似度越大

计算步骤:

- 随机初始化k个聚类中心：$\mu_1, \mu_2, ..., \mu_k$
- 遍历n个训练样本，将每个训练样本归为与类中心相似度最大的类别
- 更新聚类中心，聚类中心为当前类别所属样本的均值
- 若收敛（聚类中心不再大幅度变化或者损失函数不再大幅度变化）则停止计算，返回结果，否则回到step2

影响因素:

- 类别数目k的大小
- 计算相似度的方式
- 初始化聚类中心的方式

Q3

基本原理:

建立在图论的谱图理论基础之上，本质上是将聚类问题转化为一个图上关于顶点划分的最优问题， 建立在点对亲和性基础之上，理论上能对任意分布形状的样本空间上进行聚类。

经典算法:

- 计算顶点的相似度矩阵W和基于相似度矩阵的拉普拉斯矩阵L
- 计算L的k个最小特征值的特征向量 $\mu_1, \mu_2, ..., \mu_k$
- 令$U = [\mu_1, ..., \mu_k] \in \mathbb{R}^{n \times k}$
- 将U的每一行作为一个向量 y_i，对其使用k-means

影响因素:

- 相似度矩阵W中的权重系数的计算方法
- 基于k近邻的权重矩阵的k的大小
- 基于ϵ权重矩阵的ϵ的大小
- 聚类数目
- 聚类方法
- 归一化方法

Part 2

```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
np.random.seed(123)
```

```
mu1 = np.array([1, -1]).reshape(1,-1)
mu2 = np.array([5.5, -4.5]).reshape(1,-1)
mu3 = np.array([1, 4]).reshape(1,-1)
mu4 = np.array([6, 4.5]).reshape(1,-1)
mu5 = np.array([9, .0]).reshape(1,-1)
mu = np.r_[mu1, mu2, mu3, mu4, mu5]
mu
```

```
array([[ 1. , -1. ],
       [ 5.5, -4.5],
       [ 1. ,  4. ],
       [ 6. ,  4.5],
       [ 9. ,  0. ]])
```

```
import scipy.io as scio
data = scio.loadmat('./X.mat')
```

```
X = data['X']
```

```
X.shape
```

```
(1000, 2)
```

```
X
```

```
array([[ 1.53766714, -0.81677274],
       [ 2.83388501, -2.02976754],
       [-1.25884686, -0.05077817],
       ...,
       [ 8.44433855, -0.54890192],
       [ 7.64844366, -0.12601136],
       [ 9.36421132,  0.29958041]])
```

```python
labels  = np.ones(shape = (1000, 1))
for i in range(5):
    labels[200*i:200*(i+1), :] *= i + 1
labels = labels.reshape(-1)
```

```python
class KMeans():
    def __init__(self, n_clusters=6):
        self.k = n_clusters

    def fit(self, data):
        """
        Fits the k-means model to the given dataset
        """
        n_samples, _ = data.shape
        # initialize cluster centers
        self.centers = np.array(random.sample(list(data), self.k))
        self.initial_centers = np.copy(self.centers)

        # We will keep track of whether the assignment of data points
        # to the clusters has changed. If it stops changing, we are
        # done fitting the model
        old_assigns = None
        n_iters = 0

        while True:
            new_assigns = [self.classify(datapoint) for datapoint in data]

            if new_assigns == old_assigns:
                print(f"Training finished after {n_iters} iterations!")
                return

            old_assigns = new_assigns
            n_iters += 1

            # recalculate centers
            for id_ in range(self.k):
                points_idx = np.where(np.array(new_assigns) == id_)
                datapoints = data[points_idx]
                self.centers[id_] = datapoints.mean(axis=0)

    def l2_distance(self, datapoint):
        dists = np.sqrt(np.sum((self.centers - datapoint)**2, axis=1))
        return dists

    def classify(self, datapoint):
        """
        Given a datapoint, compute the cluster closest to the
        datapoint. Return the cluster ID of that cluster.
        """
        dists = self.l2_distance(datapoint)
        return np.argmin(dists)

    def plot_clusters(self, data):
        plt.figure(figsize=(12,10))
```

```
        plt.title("Initial centers in black, final centers in red")
        plt.scatter(data[:, 0], data[:, 1], marker='.', c='y')
        plt.scatter(self.centers[:, 0], self.centers[:,1], c='r')
        plt.scatter(self.initial_centers[:, 0], self.initial_centers[:,1],
c='k')
        plt.show()
```

```
for i in range(10):
    #随机初始化
    model = KMeans(5)
    model.fit(X)
    acc = 0
    err = []
    centers = model.centers
    for i in range(centers.shape[0]):
        x = centers[i, :]
        x = x.reshape(1, -1)
        dist = np.sum(np.power(x - mu, 2), axis = 1)
        index = np.argmin(dist)
        y = mu[index, :]
        err.append(np.sqrt(np.sum(np.power(x - y, 2))))
        for j in range(labels.shape[0]):
            a = model.classify(X[j, :])
            if a == i and labels[j] == index + 1:
                acc += 1
    acc /= labels.shape[0]
    err = np.array(err)
    err = err.sum()
    print(acc, '\n', err, '\n', centers, '\n=================\n')
```

```
Training finished after 5 iterations!
0.991
 0.4343965817583033
 [[ 0.99169186 -1.0700545 ]
 [ 5.51671392 -4.53724851]
 [ 6.16539437  4.4131549 ]
 [ 0.89181905  4.04551055]
 [ 9.01165332  0.01481903]]
=================

Training finished after 20 iterations!
0.99
 0.42993806112651645
 [[ 5.51671392 -4.53724851]
 [ 0.88845587  4.05856206]
 [ 6.16539437  4.4131549 ]
 [ 9.01165332  0.01481903]
 [ 0.9944716  -1.05720189]]
=================

Training finished after 7 iterations!
0.99
 0.42993806112651645
 [[ 9.01165332  0.01481903]
 [ 5.51671392 -4.53724851]
 [ 0.88845587  4.05856206]
```

```
 [ 6.16539437  4.4131549 ]
 [ 0.9944716  -1.05720189]]
=================

Training finished after 5 iterations!
0.99
 0.42993806112651645
 [[ 5.51671392 -4.53724851]
 [ 9.01165332  0.01481903]
 [ 0.88845587  4.05856206]
 [ 0.9944716  -1.05720189]
 [ 6.16539437  4.4131549 ]]
=================

Training finished after 24 iterations!
0.991
 0.4343965817583033
 [[ 6.16539437  4.4131549 ]
 [ 5.51671392 -4.53724851]
 [ 0.89181905  4.04551055]
 [ 9.01165332  0.01481903]
 [ 0.99169186 -1.0700545 ]]
=================

Training finished after 4 iterations!
0.991
 0.4343965817583033
 [[ 9.01165332  0.01481903]
 [ 6.16539437  4.4131549 ]
 [ 0.99169186 -1.0700545 ]
 [ 0.89181905  4.04551055]
 [ 5.51671392 -4.53724851]]
=================

Training finished after 3 iterations!
0.99
 0.42993806112651645
 [[ 9.01165332  0.01481903]
 [ 0.88845587  4.05856206]
 [ 0.9944716  -1.05720189]
 [ 5.51671392 -4.53724851]
 [ 6.16539437  4.4131549 ]]
=================

Training finished after 9 iterations!
0.991
 0.4343965817583033
 [[ 0.99169186 -1.0700545 ]
 [ 0.89181905  4.04551055]
 [ 6.16539437  4.4131549 ]
 [ 5.51671392 -4.53724851]
 [ 9.01165332  0.01481903]]
=================

Training finished after 14 iterations!
0.99
 0.42993806112651645
 [[ 0.9944716  -1.05720189]
```

```
 [ 0.88845587  4.05856206]
 [ 9.01165332  0.01481903]
 [ 5.51671392 -4.53724851]
 [ 6.16539437  4.4131549 ]]
=================

Training finished after 8 iterations!
0.99
 0.42993806112651645
 [[ 5.51671392 -4.53724851]
 [ 9.01165332  0.01481903]
 [ 6.16539437  4.4131549 ]
 [ 0.88845587  4.05856206]
 [ 0.9944716  -1.05720189]]
=================
```