In [ ]:
```
import numpy as np
import csv
import matplotlib.pyplot as plt
from utils.plotDecBoundaries import plotDecBoundaries
```

In [ ]:
```
def getData(fname):
    with open(fname, mode ='r')as file:
        # reading the CSV file
        csvFile = csv.reader(file)

        # create a new array to store the data
        dataA = np.empty([0,2])
        dataB = np.empty([0,2])
        labels = np.empty([0,1])

        # displaying the contents of the CSV file
        for lines in csvFile:
            if(float(lines[2]) == 1.):
                dataA = np.row_stack((dataA,[float(lines[0]), float(lines[1])]))
            else:
                dataB = np.row_stack((dataB,[float(lines[0]), float(lines[1])]))

    N1 = len(dataA)
    N2 = len(dataB)
    labels = np.ones( N1 + N2 )
    labels[N1:] += 1

    return (dataA, dataB, labels, N1, N2)
```

In [ ]:
```
def nearest_means_classifier(fname):
    (dataA,dataB,labels,N1,N2) = getData(fname)

    N = N1 + N2
    data = np.row_stack((dataA,dataB))

    sample_means = np.zeros((2,2))
    sample_means[0] = np.mean(data[:N1], axis=0)
    sample_means[1] = np.mean(data[N1:], axis=0)


    print(f'{N} total data points.  {N1} in class 1 and {N2} in class 2\n')

    print(f'The sample mean for class 1 data is: {sample_means[0]}\n')

    print(f'The sample mean for class 2 data is: {sample_means[1]}')

    plotDecBoundaries(data, labels, sample_means, fsize=(10,10))
```

In [ ]:
```
def classify_test(fname, means):
    (dataA,dataB,labels,N1,N2) = getData(fname)

    N = N1 + N2
    data = np.row_stack((dataA,dataB))

    N1_errors = 0
    N2_errors = 0
    i = 0
    for ele in data:
        dis_mean1 = np.square((ele[0] - means[0][0])**2 + (ele[1] - means[0][1]) ** 2)
        dis_mean2 = np.square((ele[0] - means[1][0])**2 + (ele[1] - means[1][1]) ** 2)
        if(dis_mean1 < dis_mean2 and labels[i] == 2.):
            N1_errors += 1
        elif(dis_mean1 > dis_mean2 and labels[i] == 1.):
            N2_errors += 1
        i += 1

    error_rate =( N1_errors + N2_errors ) / 100 * 100
    print(f'Error rate = {error_rate : 0.3f}%')

    sample_means = np.zeros((2,2))
    sample_means[0] = means[0]
    sample_means[1] = means[1]

    plotDecBoundaries(data, labels, sample_means, fsize=(10,10))
```

In [ ]:
```
nearest_means_classifier('dataset1_train.csv')
```

In [ ]:
```
classify_test('dataset1_train.csv',[[0.08893115, 1.08956606],[1.04128622 ,0.01994688]])
```

In [ ]:
```
classify_test('dataset1_test.csv',[[0.08893115,1.08956606],[1.04128622,0.01994688]])
```

In [ ]:
```
nearest_means_classifier('dataset2_train.csv')
```

In [ ]:
```
classify_test('dataset2_train.csv',[[-0.3536011,0.99036239],[1.03652797,-0.03223129]])
```

In [ ]:
```
classify_test('dataset2_test.csv',[[-0.3536011,0.99036239],[1.03652797,-0.03223129]])
```

In [ ]:
```
nearest_means_classifier('dataset3_train.csv')
```

In [ ]:
```
classify_test('dataset3_train.csv',[[-0.06738853 , 0.21324203],[ 0.52230078 , 0.93267215]])
```

In [ ]:
```
classify_test('dataset3_test.csv',[[-0.06738853 , 0.21324203],[ 0.52230078 , 0.93267215]])
```

In [ ]:
```
def normalize_data(data):
    # caculate the standard deviation and the mean
    total_means = np.mean(data[:], axis=0)
    print(f'The total mean for class 1&2 data is: {total_means}\n')

    total_stds = np.std(data[:], axis=0)
    print(f'The total std for class 1&2 data is: {total_stds}\n')

    # normalize the data
    data = (data[:] - total_means) / total_stds

    return data
```

In [ ]:
```
def normalize_data_mean(data, total_means, total_stds):
    # normalize the data
    data = (data[:] - total_means) / total_stds

    total_means_after = np.mean(data[:], axis=0)
    print(f'The total mean for class 1&2 data is: {total_means_after}\n')
    total_stds_after = np.std(data[:], axis=0)
    print(f'The total std for class 1&2 data is: {total_stds_after}\n')

    return data
```

In [ ]:
```
def nearest_mean_classifier_standardize(fname):
    (dataA,dataB,labels,N1,N2) = getData(fname)

    N = N1 + N2
    data = np.row_stack((dataA,dataB))

    data = normalize_data(data)


    sample_means = np.zeros((2,2))
    sample_means[0] = np.mean(data[:N1], axis=0)
    sample_means[1] = np.mean(data[N1:], axis=0)
    print(f'The sample mean for class 1 data is: {sample_means[0]}\n')
    print(f'The sample mean for class 2 data is: {sample_means[1]}\n')

    total_means = np.mean(data[:], axis=0)
    print(f'The total mean for data is: {total_means}\n')
    total_stds = np.std(data[:], axis=0)
    print(f'The total std for class 1&2 data is: {total_stds}\n')


    plotDecBoundaries(data, labels, sample_means, fsize=(10,10))
```

In [ ]:
```
def classify_standard_test(fname, total_means, means, total_stds):

    (dataA,dataB,labels,N1,N2) = getData(fname)

    N = N1 + N2
    data = np.row_stack((dataA,dataB))

    # normalize the data
    data = normalize_data_mean(data, total_means, total_stds)

    N1_errors = 0
    N2_errors = 0
    i = 0

    for ele in data:
        dis_mean1 = np.square((float(ele[0]) - means[0][0])**2 + (float(ele[1]) - means[0][1]) ** 2)
        dis_mean2 = np.square((float(ele[0]) - means[1][0])**2 + (float(ele[1]) - means[1][1]) ** 2)
        if(dis_mean1 < dis_mean2 and labels[i] == 2.):
            N1_errors += 1
        elif(dis_mean1 > dis_mean2 and labels[i] == 1.):
            N2_errors += 1
        i += 1
    error_rate =( N1_errors + N2_errors ) / 100 * 100

    print(f'Error rate = {error_rate : 0.3f}%')

    sample_means = np.zeros((2,2))
    sample_means[0] = means[0]
    sample_means[1] = means[1]
    plotDecBoundaries(data, labels, sample_means, fsize=(10,10))
```

In [ ]:
```
nearest_mean_classifier_standardize('dataset1_train.csv')
```

In [ ]:
```
classify_standard_test('dataset1_train.csv', [0.56510868, 0.55475647],
                       [[-0.40664534,0.45213344],[ 0.40664534, -0.45213344]] , [1.17098977,1.18285787])
```

In [ ]:
```
classify_standard_test('dataset1_test.csv', [0.56510868, 0.55475647],
                       [[-0.40664534,0.45213344],[ 0.40664534, -0.45213344]] , [1.17098977,1.18285787])
```

In [ ]:
```
nearest_mean_classifier_standardize('dataset2_train.csv')
```

In [ ]:
```
classify_standard_test('dataset2_train.csv', [0.34146344,0.47906555],
                       [[-0.53490724,0.95882365],[0.53490724,-0.95882365]] , [1.29941135,0.53325431])
```

In [ ]:
```
classify_standard_test('dataset2_test.csv', [0.34146344,0.47906555],
                       [[-0.53490724,0.95882365],[0.53490724,-0.95882365]] , [1.29941135,0.53325431])
```

In [ ]:
```
nearest_mean_classifier_standardize('dataset3_train.csv')
```

In [ ]:
```
classify_standard_test('dataset3_train.csv', [0.22745613,0.57295709],
                       [[-0.2061046,-0.30508577],[0.2061046,0.30508577]] , [1.43055836,1.17906206])
```

In [ ]:
```
classify_standard_test('dataset3_test.csv', [0.22745613,0.57295709],
                       [[-0.2061046,-0.30508577],[0.2061046,0.30508577]] , [1.43055836,1.17906206])
```

In [ ]:
```
def caculate_error_rate(data, N1, N2, labels, sample_means):

    N1_errors = 0
    N2_errors = 0
    i = 0

    for ele in data:
        dis_mean1 = abs(ele - sample_means[0])
        dis_mean2 = abs(ele - sample_means[1])
        if(dis_mean1 < dis_mean2 and labels[i] == 2.):
            N1_errors += 1
        elif(dis_mean1 > dis_mean2 and labels[i] == 1.):
            N2_errors += 1
        i += 1
    error_rate =( N1_errors + N2_errors ) / 100 * 100

#     print(f'Error rate = {error_rate : 0.3f}%')
    return error_rate
```

In [ ]:
```
def projector(fname):
    (dataA,dataB,labels,N1,N2) = getData(fname)

    N = N1 + N2
    data = np.row_stack((dataA,dataB))

    # caculate the standard deviation and the mean
    total_means = np.mean(data[:], axis=0)
    total_stds = np.std(data[:], axis=0)

    # normalize the data
    data = (data[:] - total_means) / total_stds

    # find the min m
    rm_star = []
    m_star = 100
```

```
    errorRate = 100.
    train_mean = []

    x = []
    e = []
#     x = np.empty(40)
#     e = np.empty(40)

    for m in range(10):
        rm = [10, m ]
        rmUnit = rm / np.linalg.norm(rm)
        dotdata = np.dot(data,rm)

        sample_means = np.zeros(2)
        sample_means[0] = np.mean(dotdata[:N1])
        sample_means[1] = np.mean(dotdata[N1:])
        error_rate_this = float(caculate_error_rate(dotdata, N1, N2, labels, sample_means))
#         np.row_stack((e,error_rate_this))
        e.append(error_rate_this)
        x.append(m)
#         e = np.row_stack((e,error_rate_this))
#         x = np.row_stack((x,m))
#         e = np.append(e, [error_rate_this], axis=0)
#         x =np.append(x,[m],axis = 0)

        if error_rate_this < errorRate:
            errorRate = error_rate_this
            m_star = m
            rm_star = rm
            train_mean = sample_means
        print('m = ', m, ',rm = ', rm, f',errorRate = { error_rate_this : 0.3f}%' )

    for m in range(10,30):
        rm = [20 - m, 10]
        rmUnit = rm / np.linalg.norm(rm)
        dotdata = np.dot(data,rm)

        sample_means = np.zeros(2)
        sample_means[0] = np.mean(dotdata[:N1])
        sample_means[1] = np.mean(dotdata[N1:])
        error_rate_this = float(caculate_error_rate(dotdata, N1, N2, labels, sample_means))
#         np.row_stack((e,error_rate_this))
        e.append(error_rate_this)
        x.append(m)
#         e = np.append(e, [error_rate_this], axis=0)
#         x = np.append(x,[m],axis = 0)



        if error_rate_this < errorRate:
            errorRate = error_rate_this
            m_star = m
            rm_star = rm
            train_mean = sample_means
```

```
        print('m = ', m, ',rm = ', rm, f',errorRate = { error_rate_this : 0.3f}%')

    for m in range(30,40):
        rm = [-10, 40-m]
        rmUnit = rm / np.linalg.norm(rm)
        dotdata = np.dot(data,rm)

        sample_means = np.zeros(2)
        sample_means[0] = np.mean(dotdata[:N1])
        sample_means[1] = np.mean(dotdata[N1:])

        error_rate_this = float(caculate_error_rate(dotdata, N1, N2, labels, sample_means))
#         np.row_stack((e,error_rate_this))
        e.append(error_rate_this)
        x.append(m)
#         e = np.append(e, [error_rate_this], axis=0)
#         x = np.append(x,[m],axis = 0)

        if error_rate_this < errorRate:
            errorRate = error_rate_this
            m_star = m
            rm_star = rm
            train_mean = sample_means
        print('m = ', m, ',rm = ', rm, f',errorRate = { error_rate_this : 0.3f}%')

    plt.plot(x,e)
    plt.show()

    rmUnit = rm_star / np.linalg.norm(rm_star)
    dotdata = np.dot(data,rm_star)

    dataUnit = np.empty([0,2])
    for e in dotdata:
        dataUnit = np.row_stack((dataUnit, e * rmUnit))
#     print(dataUnit)

#     print(np.mean(dotdata))
    sample_means = np.zeros((2,2))
    sample_means[0] = np.mean(dataUnit[:N1], axis=0)
    sample_means[1] = np.mean(dataUnit[N1:], axis=0)

    print('m* = ',m_star,f', errorRate = { errorRate: 0.3f}%, rm* = ',rm_star,', means = ',sample_means, 'train means = ',train_mean)
    plotDecBoundaries(dataUnit, labels, sample_means, fsize=(10,10))


#     print(sample_means)
```

In [ ]:
```
def classify_project_test(fname, total_means, means, total_stds, rm, sample_means):
    (dataA,dataB,labels,N1,N2) = getData(fname)

    N = N1 + N2
    data = np.row_stack((dataA,dataB))

    # normalize the data
    data = (data[:] - total_means) / total_stds

    # dataUnit = np.dot(data,rm) / np.dot(rm,rm) * rm
    rmUnit = rm / np.linalg.norm(rm)
    dotdata = np.dot(data,rm)
    dataUnit = np.empty([0,2])
    for e in dotdata:
        dataUnit = np.row_stack((dataUnit, e * rmUnit))

    N1_errors = 0
    N2_errors = 0
    i = 0

    error_rate = float(caculate_error_rate(dotdata, N1, N2, labels, sample_means))
    print(f'Error rate = {error_rate : 0.3f}%')

    sm = np.zeros((2,2))
    sm[0] = means[0]
    sm[1] = means[1]

    plotDecBoundaries(dataUnit, labels, sm, fsize=(10,10))
```

In [ ]:
```
projector('dataset1_train.csv')
```

In [ ]:
```
classify_project_test('dataset1_test.csv', [0.56510868, 0.55475647], [[-4.22518553,6.03597933],
[ 4.22518553,-6.03597933]] , [1.17098977,1.18285787], [-7,10], [ 7.36785174 , -7.36785174])
```

In [ ]:
```
projector('dataset2_train.csv')
```

In [ ]:
```
classify_project_test('dataset2_test.csv', [0.34146344,0.47906555], [[ 1.67060001,8.35300006],
 [-1.67060001, -8.35300006]] , [1.29941135,0.53325431],[2,10],[ 8.51842206, -8.51842206])
```

In [ ]:
```
projector('dataset3_train.csv')
```

In [ ]:
```
classify_project_test('dataset3_test.csv', [0.22745613,0.57295709], [[-3.51526224,-2.81220979],
 [ 3.51526224,2.81220979]] , [1.43055836,1.17906206],[10,8],[-4.50173217 , 4.50173217])
```