```
load('partOneData.mat');
fprintf('Assignment 2 - Part 1\n\n');

% Function for k = 5, 3, 7, 15
% k = 5;
% myKNN(classOne, classTwo, k)
k = 3;
myKNN(classOne, classTwo, k)
% k = 7;
% myKNN(classOne, classTwo, k)
% k = 15;
% myKNN(classOne, classTwo, k)

% Function that runs the code
function kNN = myKNN(classOne, classTwo, k)

% Partition the data into 5 random parts
% A, B, C, D, E with each part of 20%
% Combines 60% for training and 40% for testing
[class1IndicesA, class1IndicesB, class1IndicesC, class1IndicesD,
 class1IndicesE] = Partition();
[class2IndicesA, class2IndicesB, class2IndicesC, class2IndicesD,
 class2IndicesE] = Partition();

class1A = classOne(class1IndicesA);
class1B = classOne(class1IndicesB);
class1C = classOne(class1IndicesC);
class1D = classOne(class1IndicesD);
class1E = classOne(class1IndicesE);

class2A = classTwo(class2IndicesA);
class2B = classTwo(class2IndicesB);
class2C = classTwo(class2IndicesC);
class2D = classTwo(class2IndicesD);
class2E = classTwo(class2IndicesE);

% Assigns classes to the testing data based on the training data.
 Either class 1 or class 2.
[class1TestResults1] = assignClasses([class1A, class1B, class1C],
 [class2A, class2B, class2C], [class1D, class1E], k);
[class2TestResults1] = assignClasses([class1A, class1B, class1C],
 [class2A, class2B, class2C], [class2D, class2E], k);

[class1TestResults2] = assignClasses([class1B, class1C, class1D],
 [class2B, class2C, class2D], [class1E, class1A], k);
[class2TestResults2] = assignClasses([class1B, class1C, class1D],
 [class2B, class2C, class2D], [class2E, class2A], k);

[class1TestResults3] = assignClasses([class1C, class1D, class1E],
 [class2C, class2D, class2E], [class1A, class1B], k);
[class2TestResults3] = assignClasses([class1C, class1D, class1E],
 [class2C, class2D, class2E], [class2A, class2B], k);
```

```
[class1TestResults4] = assignClasses([class1D, class1E, class1A],
 [class2D, class2E, class2A], [class1B, class1C], k);
[class2TestResults4] = assignClasses([class1D, class1E, class1A],
 [class2D, class2E, class2A], [class2B, class2C], k);

[class1TestResults5] = assignClasses([class1E, class1A, class1B],
 [class2E, class2A, class2B], [class1C, class1D], k);
[class2TestResults5] = assignClasses([class1E, class1A, class1B],
 [class2E, class2A, class2B], [class2C, class2D], k);

% Computes the accuracy for the data being classified as classOne
class1Accuracy = [];
class1Accuracy(1) = accuracy(class1TestResults1, 1);
class1Accuracy(2) = accuracy(class1TestResults2, 1);
class1Accuracy(3) = accuracy(class1TestResults3, 1);
class1Accuracy(4) = accuracy(class1TestResults4, 1);
class1Accuracy(5) = accuracy(class1TestResults5, 1);

% Computes the accuracy for the data being classified as classTwo
class2Accuracy = [];
class2Accuracy(1) = accuracy(class2TestResults1, 2);
class2Accuracy(2) = accuracy(class2TestResults2, 2);
class2Accuracy(3) = accuracy(class2TestResults3, 2);
class2Accuracy(4) = accuracy(class2TestResults4, 2);
class2Accuracy(5) = accuracy(class2TestResults5, 2);

% Combines the 10 prediction accuracies in an array
Accuracy = [class1Accuracy, class2Accuracy];

% Finds the mean of the prediction accuracy and displays it.
mu = mean(Accuracy);
fprintf('Accuracy mean is %d\n', mu);

% Finds the standard deviation of the prediction accuracy and displays
 it.
sigma = std(Accuracy);
fprintf('Accuracy standard deviation is %d\n', sigma);
end

% Function that computes the accuracy by dividing the number of
% correct classifications by the total number of classifications
function report = accuracy(results, class)
report = 0.0;
sizeResults = size(results);
    for i = 1:size(results,2);
        if (results(i) == class)
            report = report+1;
        end
    end
report = report/size(results,2);
end

% Function that determines the class a test data belongs to
```

```
function [class] = assignClasses(trainingClass1, trainingClass2,
 testData, k)
class = [];
    for i = 1:size(testData,2)
        class(i) = determineClass(trainingClass1, trainingClass2,
 testData(i), k);
    end
end

% Function that partitions the data into 5 random parts of 20% each
function [dataA, dataB, dataC, dataD, dataE] = Partition()
indices = [1:10000];
dataA = randsample(indices,2000);
indices = setdiff(indices, dataA, 'stable');
dataB = randsample(indices, 2000);
indices = setdiff(indices, dataB, 'stable');
dataC = randsample(indices, 2000);
indices = setdiff(indices, dataC, 'stable');
dataD = randsample(indices, 2000);
dataE = setdiff(indices, dataD, 'stable');
 %stable helps it stay random
end

% Function that computes the Euclidean distance
function euclidDistance = computeEuclideanDistance(testValue,
 trainValue)
euclidDistance = sqrt((testValue-trainValue)^2);
end

% Function that uses the Euclidean distance to find all the distances
% between a test value and the training data
function [distances] = computeAllDistances(trainData, testValue)
distances = [];
    for i = 1:size(trainData, 2)
        distances(i) = computeEuclideanDistance(testValue,
 trainData(1, i));
    end
end

% Function that finds the neighbors by sorting the distances for each
 class
% into ascending order. We get back a sorted order of the list and the
% indices of the original matrix that represent the order of the
 ascended
% matrix.
function [neighbors] = findKNearestNeighbors(distancesClass1,
 distancesClass2, k)
neighbors = [];
[sortedDistanceClass1, sortedIndicesClass1] = sort(distancesClass1);
[sortedDistanceClass2, sortedIndicesClass2] = sort(distancesClass2);

% Use the first k indices to get the values for each class
sortedDistanceClass1 = distancesClass1(sortedIndicesClass1(1:k));
sortedDistanceClass2 = distancesClass2(sortedIndicesClass2(1:k));
```

```
% combine the 2 ks
TwoKNearestNeighbors = [sortedDistanceClass1, sortedDistanceClass2];

% Sort the 2k nearest neighbors
[NearestNeighbors, nearestIndices] = sort(TwoKNearestNeighbors);

%Categorize as class one as class two by using the indices
neighbors = NearestNeighbors(1:k);

% COULD NOT FIGURE OUT HOW TO SORT A MATRIX BY SORTING ONLY ONE ARRAY,
% WHICH IS WHAT I NEEDED.

%if indices above k, categorize as class 2, otherwise categorize as
%class 1
    for i = 1 : k
       if (nearestIndices(i) > k)

           neighbors(2,i) = 2;
       else
           neighbors(2,i) = 1;
       end
    end
end

% Function that determines the class by computing the distance between
 the
% testing values and the training values.
function class = determineClass(trainingDataClass1,
 trainingDataClass2, testValue, k)
classOneDistances = computeAllDistances(trainingDataClass1,
 testValue);
classTwoDistances = computeAllDistances(trainingDataClass2,
 testValue);
neighbors = findKNearestNeighbors(classOneDistances,
 classTwoDistances, k);
classOneHits = 0;
classTwoHits = 0;
    for i = 1:k
        if (neighbors(2,k) == 1)
            classOneHits = classOneHits + 1;
        else
            classTwoHits = classTwoHits + 1;
        end
    end
        if (classOneHits > classTwoHits)
            class = 1;
        else
            class = 2;
        end
end
```

Assignment 2 - Part 1

```
Accuracy mean is 7.791250e-01
Accuracy standard deviation is 7.672326e-03
```

Published with MATLAB® R2017b