2. Nearest neighbors using a simple kernel. Recall that the nearest neighbors (NN) classifier takes a "plurality vote" among the K training points nearest to our test point. In this problem, you will modify the implementation of NN such that the nearest neighbor receives weight (K)/[K(K+1)/2], the second nearest weight (K-1)/[K(K+1)/2], down to the K'th nearest neighbor whose weight is (1)/[K(K+1)/2]. (Note that the weights sum to 1, because the values in the numerators of the weights, i.e., 1+2+…+K, sum to K(K+1)/2.) Compare the original NN to the modified version with different weights. You may want to compare them on the class pdf's defined in Problem 1.

Solution: We added a column containing the weights to the top k predicted labels and repredicted the class labels.

```
W = (num_neighbors:-1:1)/(num_neighbors*(num_neighbors+1)/2);
....

A = cat(2,train(3, neighbors)', W');
%A = cat(2,class_samples(neighbors), W');
sum0=0;sum1=0;
for i = 1:num_neighbors
    if A(i,1)==0
        sum0 = sum0+A(i,2);
    else
        sum1 = sum1+A(i,2);
    end
end
class_predicted=(sum1/sum0 > 1); % NN classifier
test_NNK(n1)=class_predicted; % store classification
```

we have modularized the classification.m code into four matlab codes.

```
Main.m : driver code
Knn: regular knn with majority voting
knnWithKernel.m: Knn with kernel
dataSetCreator.m: this creates data set with multivariate Gausisan distribution, given, number of
clusters, N ( size of complete dataset, and cluster_variations
```

```
% main.m
% This is used to compare kNN and the kerneled one
%% creating dataset and spliting it into training and testing sets
clear % often useful to clean up thework space from old variables
close all
num_clusters=5; % number of components (clusters) in mixture model
N=6*800; % total number of samples of training data
cvs = .2:.2:1; % different cluster varations
nns = 3:2:9;
avgErrors = zeros(length(cvs), length(nns),2);
repeat = 20;
for i=1:length(cvs)
    for j=1:length(nns)
        err0tot =0;
        err1tot=0;
        [train, test] = dataSetCreator(num_clusters, N,cvs(i));
        for it =1:repeat
            err0tot= err0tot +           knn(nns(j), train, test);
            err1tot= err1tot+  knnWithKernel(nns(j), train, test)
        end
        avgErrors(i,j,1) = err0tot/repeat;
        avgErrors(i,j,2) = err1tot/repeat;
    end
end
```

The code main.m returns average error for each method. Running above (for 20 times repetition) code we have:

kNN with simple kernel errors

```
avgErrors(:,:,2) =
   0.1425   0.1931   0.1812   0.3056
   0.3863   0.2619   0.4150   0.2956
   0.2919   0.4219   0.2969   0.3606
   0.4600   0.4306   0.3656   0.4494
   0.4262   0.3894   0.4462   0.3963

avgErrors(:,:,1) =

   0.1344   0.1881   0.1781   0.2900
   0.3694   0.2581   0.4025   0.2856
   0.2669   0.4313   0.2875   0.3513
   0.4437   0.4206   0.3619   0.4288
   0.4300   0.3844   0.4375   0.3863

sum(sum(avgErrors))

ans(:,:,1) =

   6.7363


ans(:,:,2) =

   6.9162
```

We see that kNN with kernel has larger average error. Also the difference, exept two instances, kNN outperformed kNN with kernel.

```
avgErrors(:,:,2)-avgErrors(:,:,1)

ans =

   0.0081   0.0050   0.0031   0.0156
   0.0169   0.0037   0.0125   0.0100
   0.0250  -0.0094   0.0094   0.0094
   0.0163   0.0100   0.0037   0.0206
```

-0.0038   0.0050   0.0087   0.0100

As we can see, kNN performs better than kNN with the simple kernel the class pdf’s defined in Problem 1.