Mohammad Saleh
GRA at PHY WSU
m.saleh@cern.ch
fa9520@wayne.edu
March 8, 2017

CSC 5825
Homework 3

Solutions:

Question 1:

```
##### I implemented the Uniform Kernel distribution
#####estimator down below, which is in problem one
### The values are 0.05, 0.0125, and 0.075 for x=3, 10, 15 respectively

import numpy as np
def Prob(x,value,h):
    sum=0
    for t in range (0,np.size(x)):
        if(abs(value-x[t])<h):
            sum=sum+0.5
    return ((1/(np.size(x)*h))*sum)

x=np.array([4, 5, 5, 6, 12, 14, 15, 15, 16, 17])
print (Prob(x,15,4))
```

Question 2:

For two set of normal distribution which has a joint normal distribution defined as:
$p(x, y) = \frac{1}{2\pi * \sqrt{1-\rho^2}} exp(-\frac{x^2-2\rho xy+y^2}{2(1-\rho^2)})$, which is in the book page 97, where x here $=(x_a - \mu_a)/\sigma_a$ and similarily for y so we want to look at the conditional probability:
$p(x|y)$ is Gaussain too?????
$p(x|y) = \frac{p(x,y)}{p(y)}$
$=\frac{1}{2\pi * \sqrt{1-\rho^2}} exp(-\frac{x^2-2\rho xy+y^2}{2(1-\rho^2)}) * \sqrt{2\pi} exp(y^2/2) = \frac{1}{2\pi * \sqrt{1-\rho^2}} exp(-\frac{x^2-2\rho xy+\rho^2 y^2}{2(1-\rho^2)})$
$=\frac{1}{2\pi * \sqrt{1-\rho^2}} exp(-\frac{(x-\rho y)^2}{2(1-\rho^2)})$,
which is in the form of normal (gaussain) distribution $N(\rho y, 1-\rho^2)$

Question 3

First, lets start with the advantages of Mahalanobis distance over Euclidean distance. The Mahalanobis distance takes into account the covariance among the variables when calculating the distances where the Euclidean distance is blind to correlated variables. As Mahalanobis distance takes into account the covariance, the problem of scale and correlation in the Euclidean distance are no longer there. The advantages of Euclidean distance over Mahalanobis distance is that the the Euclidean is easy and simple to calculate also easier to code.

Programming Questions

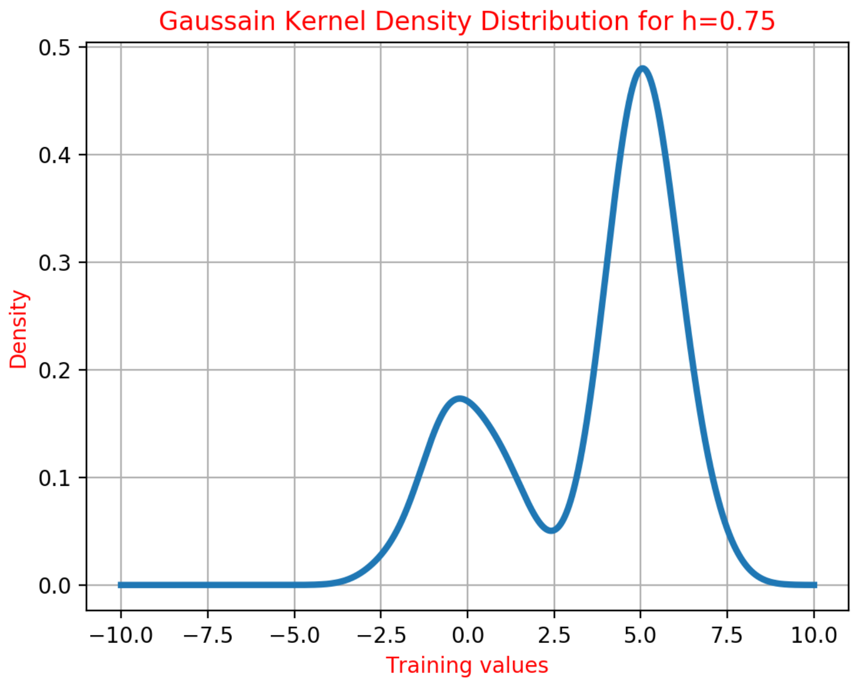

Figure 1: snapshot of my code after compiling. It shows the kernel Gaussain distribution.

Question 1:

Below is the syntax for my code and figure.1 shows a snapshot of the compiled code

```
import numpy as np
import matplotlib.pyplot as plt

def GaussainKernel(x,h,data):
    sum=0
    for t in range (0,np.size(data)):
        sum=sum+(1/((2*3.14*h*h)**(1/2))*np.exp(-((x-data[t])**2)/(2*h*h)))
    return sum/np.size(data)
# Read Data

##Data= np.loadtxt('housing.data')
Data= np.loadtxt('data_Kernel.txt',float)
x = np.arange(-10, 10,0.0001)
plt.plot(x,GaussainKernel(x,0.75,Data),linewidth='3')
#plt.plot(x,x)
plt.xlabel('Training values',color='red')
plt.ylabel('Density', color='red')
plt.title('Gaussain Kernel Density Distribution for h=0.75',color='red')
plt.grid(True)
plt.show()
```

Question 2:

Below is the syntax for my code and figure.2 shows a snapshot of the compiled code before applying PCA. Where figure.3 shows a snapshot of the compiled code after applying PCA

```
import numpy as np
import matplotlib.pyplot as plt

Data=np.loadtxt('WisconBreastCancer.txt')
Class1=Data[Data[:,0]==1]
Class2=Data[Data[:,0]==2]
Class3=Data[Data[:,0]==3]
Class4=Data[Data[:,0]==4]
Class5=Data[Data[:,0]==5]

feature1=2
feature2=32
alphavalue=0.6
sizevalue=170
markerstyle=['x','o','+','s','*']
markercolor=['black','red','blue','green','magenta']
plt.figure(figsize=(12,6))
plt.scatter(Class1[:,feature1],Class1[:,feature2],marker=markerstyle[0],color=markercolor[0],label='class1=1',alpha
plt.scatter(Class2[:,feature1],Class2[:,feature2],marker=markerstyle[1],color=markercolor[1],label='class2=2',alpha
plt.scatter(Class3[:,feature1],Class3[:,feature2],marker=markerstyle[2],color=markercolor[2],label='class3=3',alpha
plt.scatter(Class4[:,feature1],Class4[:,feature2],marker=markerstyle[3],color=markercolor[3],label='class4=4',alpha
plt.scatter(Class5[:,feature1],Class5[:,feature2],marker=markerstyle[4],color=markercolor[4],label='class5=5',alpha
plt.legend(loc='upper right')
plt.ylabel('Tumor Size',size=20)
plt.xlabel('Radius Mean',size=20)
plt.title('Scatter Plot for different features',size=30)
plt.show()






plt.figure(figsize=(12,6))

def Norm(PCAData):
    for features in range (1,34):
        means=PCAData.mean(axis=0)[features]                    #recentering
        PCAData[:,features]=PCAData[:,features]-means           #recentering
        stds=PCAData.std(axis=0)[features]
        if(stds!=0):
            PCAData[:,features]=PCAData[:,features]/stds
    return PCAData
### PCA implementation
DataN=Norm(Data)
###### SVD for inout feature matrix X ###########
U, s, V=np.linalg.svd(DataN[:,1:34])
###### classes after the SVD is done ################
pcaClass1=DataN[DataN[:,0]==1]
pcaClass2=DataN[DataN[:,0]==2]
pcaClass3=DataN[DataN[:,0]==3]
pcaClass4=DataN[DataN[:,0]==4]
pcaClass5=DataN[DataN[:,0]==5]
######## Removin the class information from the features #############
pca1=pcaClass1[:,1:34]
pca2=pcaClass2[:,1:34]
pca3=pcaClass3[:,1:34]
pca4=pcaClass4[:,1:34]
pca5=pcaClass5[:,1:34]
```

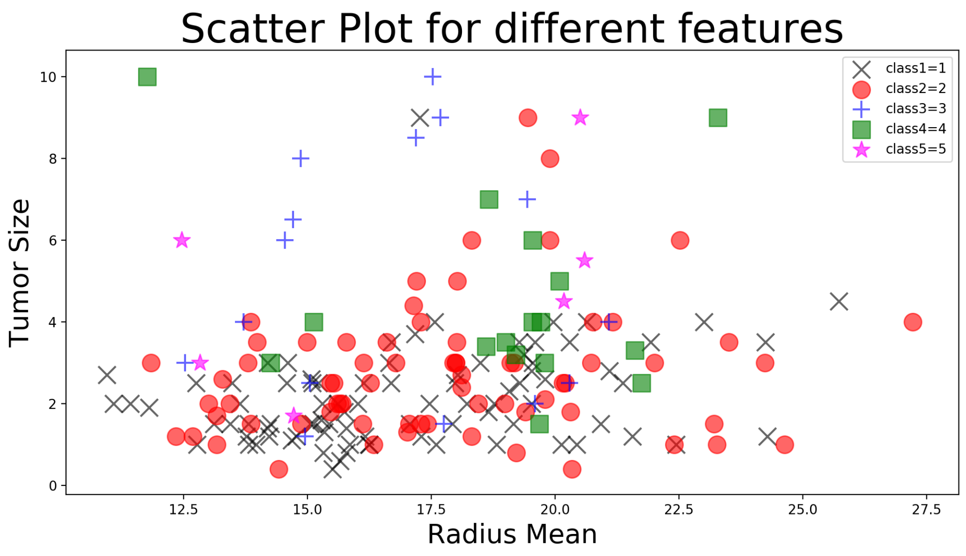

Figure 2: snapshot of my code after compiling. It shows the scatter plot before PCA. Two random features were used

```
########################################## projection x to get z
z1=np.dot(pca1,V)
z2=np.dot(pca2,V)
z3=np.dot(pca3,V)
z4=np.dot(pca4,V)
z5=np.dot(pca5,V)

plt.scatter(z1[:,0],z1[:,1],marker=markerstyle[0],color=markercolor[0],alpha=alphavalue,s=sizevalue,label='class1=1
plt.scatter(z2[:,0],z2[:,1],marker=markerstyle[1],color=markercolor[1],alpha=alphavalue,s=sizevalue,label='class2=2
plt.scatter(z3[:,0],z3[:,1],marker=markerstyle[2],color=markercolor[2],alpha=alphavalue,s=sizevalue,label='class3=3
plt.scatter(z4[:,0],z4[:,1],marker=markerstyle[3],color=markercolor[3],alpha=alphavalue,s=sizevalue,label='class4=4
plt.scatter(z5[:,0],z5[:,1],marker=markerstyle[4],color=markercolor[4],alpha=alphavalue,s=sizevalue,label='class5=5
plt.legend(loc='upper right')
plt.ylabel('z1',size=20)
plt.xlabel('z2',size=20)
plt.show()
```

Bonus question Below is the syntax for creating initial arrays and matrices for SVD and defining the SVD function to return u, s, and v, . It can be more optimized in case of memory leaks.

```
#include <math.h>

double *vector(int len)
{
   double *array={0};

   array = (double *) malloc((len*sizeof(double)));
   return(array);
}

double **Matrix(int Nrows, int Ncol)
{
   int i;
```

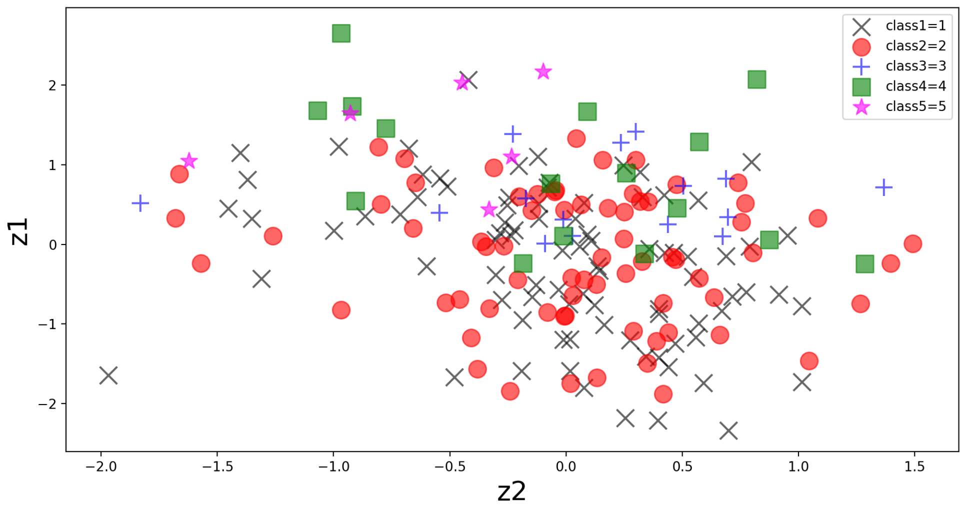

Figure 3: snapshot of my code after compiling. It shows the scatter plot after PCA using z_1 and z_2.

```
   double **mat={0};
   mat = (double **) malloc((Nrows*sizeof(double*)));
   mat[0]=(double*) malloc((Nrows*Ncol)*sizeof(double));
   for(i=1; i<Nrows; i++)
      mat[i]=mat[i-1] + Ncol;

   return mat;
}

// to create initial arrays for vectors and matrix of variable sizes
void SVD(double **a, int Nrows, int Ncol, double **u, double **s, double **v)
{
   int check, lwork;
   double *tempoa, *tempou, *tempov, *tempos;
   double wkopt;
   double* work;
   int i, j, turn, yes;

   tempoa = vector(Nrows*Ncol);
   tempou = vector(Nrows*Nrows);
   tempov = vector(Ncol*Ncol);
   tempos = vector(Ncol);

   turn = 0;
   for(i=0; i<Ncol; i++)
   {
      for(j=0; j<Nrows; j++)
      {
         tempoa[turn] = a[j][i];
         turn = turn+1;
      }
   }

   lwork = -1;
   dgesvd_( "All", "All", &Nrows, &Ncol, tempoa, &Nrows, tempos, tempou, &Nrows, tempov, &Ncol,
        &wkopt,
       &lwork, &check );

   lwork = (int)wkopt;
   work = (double*)malloc( lwork*sizeof(double) );

   dgesvd_( "All", "All", &Nrows, &Ncol, tempoa, &Nrows, tempos, tempou, &Nrows, tempov, &Ncol,
        work,
       &lwork, &check );

   if( check > 0 ) {
      printf( "Cannot do the transformation" );
      exit( 1 );
   }

   turn = 0;
   for(i=0; i<Nrows; i++)
   {
      for(j=0; j<Nrows; j++)
      {
         u[j][i] = tempou[turn];
         turn = turn+1;
      }
   }

   turn = 0;
   for(i=0; i<Ncol; i++)
   {
      for(j=0; j<Ncol; j++)
      {
         v[j][i] = tempov[turn];
         turn = turn+1;
      }
   }

   if(Nrows<Ncol) yes = Nrows;
   else yes=Ncol;

   for(i=0; i<yes; i++)
     s[i][i] = tempos[i];

}

int main(int argc, char **argv)
{
   int Nrows, Ncol;
   double **m, **u, **s, **v;

   Nrows = 4;    // changable
   Ncol = 5;

   m = Matrix(Nrows, Ncol);
   u = Matrix(Nrows, Nrows);
   s = Matrix(Nrows, Ncol);
   v = Matrix(Ncol, Ncol);

   double h= 1.0;
   int i, j;
```

```
  for(i=0; i<Nrows; i++)
  {
    for(j=0; j<Ncol; j++)
    {
      m[i][j] = h;
      h = h+1.0;
    }
  }
  SVD(m, Nrows, Ncol, u, s, v);
  return(2);
}
```