Assignment 3 - Tanmoy Das (B00844483)

Question 1

1. [20 marks] MLP for 3-class problem: This assignment is once more an opportunity to implementing an MLP f assignment can be loaded with X = np.loadtxt("FeaturesX.csv") Y = np.loadtxt("LabelsY.csv") Convert the labe and implement a neural network by adapting the MLPxor program from the manuscript. Estimate and state th classification problem.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

```
import numpy as np

# X = np.loadtxt("/content/drive/My Drive/Dal - Academic/CSCI 6505/Assignments csci-6505/Assi
# Y_raw = np.loadtxt("/content/drive/My Drive/Dal - Academic/CSCI 6505/Assignments csci-6505/
X = np.loadtxt("FeaturesX.csv")
Y_raw = np.loadtxt("LabelsY.csv")

Y = Y_raw.reshape(1, len(Y_raw))
print(X.shape)
print(Y.shape)

# X = np.loadtxt("FeaturesX.csv")
# Y = np.loadtxt("LabelsY.csv")
```

```
(7, 800)
(1, 800)
```

```
# Convert the labels of these data set to a one hot representation
import matplotlib.pyplot as plt
from keras import models, layers, optimizers, utils
y_1hot_raw = utils.to_categorical(Y)
y_hot = y_1hot_raw.reshape((y_1hot_raw.shape[0]*y_1hot_raw.shape[1]),y_1hot_raw.shape[2])
X_800_7 = X.transpose()


# Split the dataset
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X_800_7, y_hot, test_size=0.30, random_st
```

```
# Implement a neural network by adapting the MLPxor program from the manuscript.
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(560, 7) (240, 7) (560, 4) (240, 4)
```

```
#### #    Training stage start ----------------------
import numpy as np
import matplotlib.pyplot as plt

# model specifications
Ni=7; Nh=5; No=4;
learning_rate = 0.1

#parameter and array initialization
Ntrials=1000
wh=np.random.randn(Nh,Ni); dwh=np.zeros(wh.shape)
wo=np.random.randn(No,Nh); dwo=np.zeros(wo.shape)
error=np.array([])

for trial in range(Ntrials):
    h=1/(1+np.exp(-wh@x_train.T)) #hidden activation for all pattern
    y=1/(1+np.exp(-wo@h)) #output for all pattern

    do=y*(1-y)*(y_train.T-y)  # delta output
    dh=h*(1-h)*(wo.T@do)  # delta backpropagated

  # update weights with momentum
    dwo=0.9*dwo+do@h.T
    wo=wo+learning_rate*dwo
    dwh=0.9*dwh+dh@x_train
    wh=wh+learning_rate*dwh

    #error=np.append(error,np.sum(abs(y_train.T-y)))
    error=np.append(error,np.mean((y_train.T-y)**2))
```

Best classification performance:

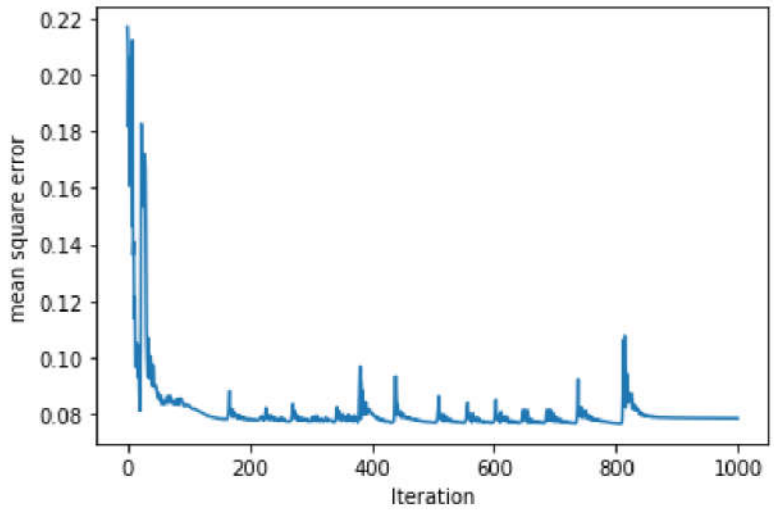
Accuracy on testing: 0.8760416666666667

```
######## testing  start ##########################################################


# test
h=1/(1+np.exp(-wh@x_test.T)) #hidden activation for all pattern
y=1/(1+np.exp(-wo@h))        #output for all pattern
y = np.around(y).T
```

```
accuracy   = (y_test == y).astype(np.int)
mean_arry  = np.asarray(accuracy).mean()
print('Accuracy on testing: ' + str(mean_arry))

######## testing  end ##########################################################################
```

```
Accuracy on testing: 0.9177083333333333
```

```
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf', 'png')

plt.xlabel("Iteration")
plt.ylabel("mean square error")
plt.plot(error)
plt.show()
```



Question 2 CSCI 4155 only

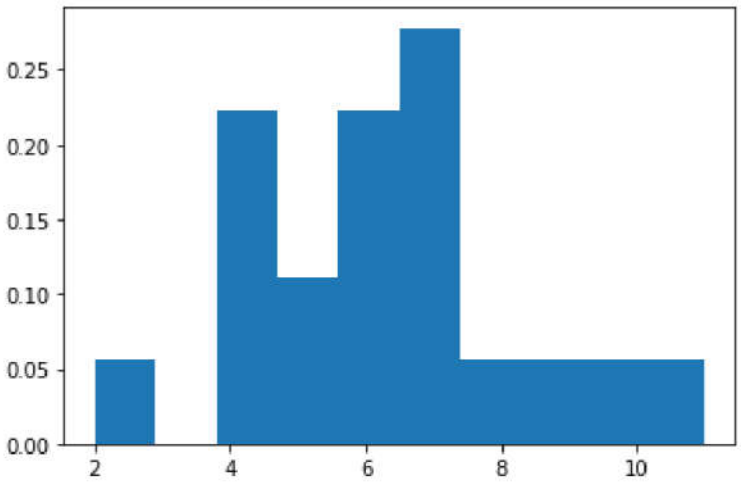
Question 3

3.1. Write a program that rolls two dice (randomly selects a number between 1 and 6 inclusive for ea each trial, you should add the two numbers that appear on each die and save it in a vector. Plot a hist the vector. Based on this histogram, what are the estimates for the probabilities of each number?

```
#
import numpy as np
import pylab
import matplotlib.pyplot as plt
from collections import Counter

np.random.seed(3)
```

```
vec = []
def Probability3_1():
        roll = 1
        while roll<=20:
            rand1 = np.random.randint(1,high = 7)
            rand2 = np.random.randint(1,high = 7)
            vec.append(rand1+rand2)
            roll+=1
Probability3_1()

N = len(vec)
output = dict(Counter(vec))
print('The Probabilities after rolling the dice 20 times are')
for i,j in zip(output.keys(),output.values()):
    print(str(i)+' : '+ str(j/N))

plt.hist(vec, align = 'mid', normed = True)
plt.show()
plt.close()
```

```
The Probabilities after rolling the dice 20 times are
4 : 0.2
6 : 0.2
2 : 0.05
7 : 0.25
10 : 0.05
9 : 0.05
11 : 0.05
5 : 0.1
8 : 0.05
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:23: MatplotlibDeprecationWa
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'der
```

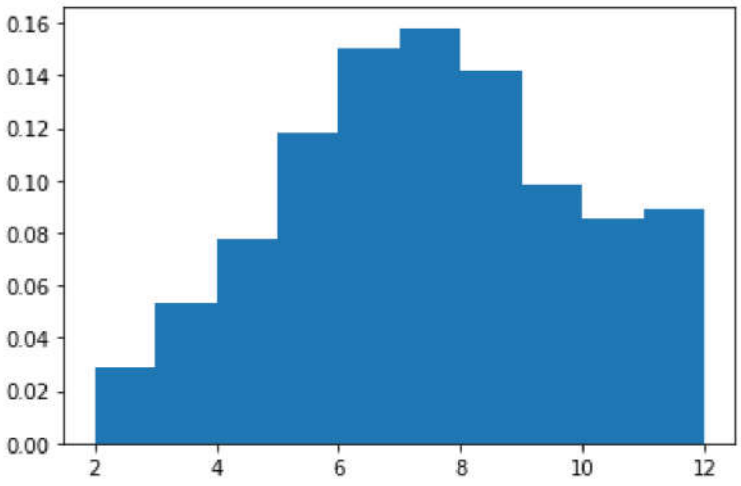


3.2. Run your code for 1000 times. What is your estimation now?

```
vec2 = []
for i in range(1000):
            rand1 = np.random.randint(1,high = 7)
            rand2 = np.random.randint(1,high = 7)
```

```
            rand2 = np.random.randint(1,high = 7)
            vec2.append(rand1+rand2)
output2 = dict(Counter(vec2))

N = len(vec2)
print('The Probabilities after rolling the dices 1000 times are given below:  ')
for i,j in zip(output2.keys(),output2.values()):
    print(str(i)+' := '+ str(j/N))

plt.hist(vec2, align = 'mid', normed = True)
plt.show()
plt.close()
```

```
The Probabilities after rolling the dices 1000 times are given below:
5 := 0.118
11 := 0.062
8 := 0.142
10 := 0.085
2 := 0.029
12 := 0.027
4 := 0.078
9 := 0.098
7 := 0.158
6 := 0.15
3 := 0.053
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:13: MatplotlibDeprecationWa
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'den
  del sys.path[0]
```



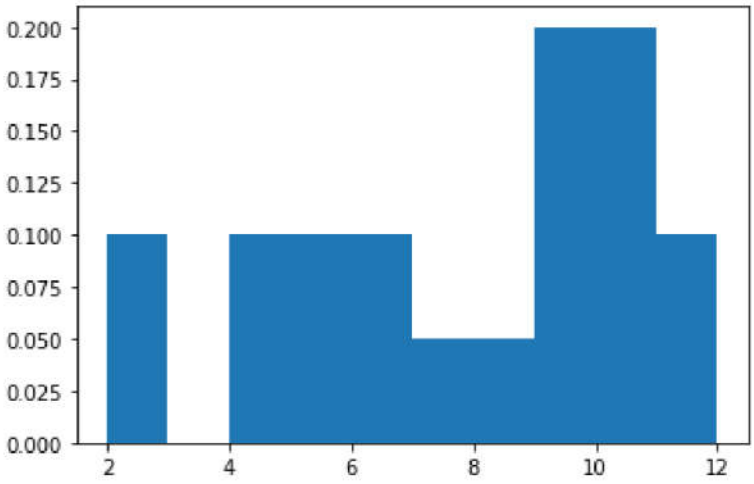
Comment of Estimation

3.3. Change your program and assume that you have one altered die and one correct die. The altered number 2 on one face. This means that when you roll this defect die, a random number should be ch histogram of the values you have gathered in the vector.

```
np.random.randint
vec3_3 = []
for i in range(20):
```

```
            rand1 = np.random.choice([1,0,3,4,5,6])
            rand2 = np.random.randint(1,high = 7)
            vec3_3.append(rand1+rand2)
output3_3 = dict(Counter(vec3_3))
N = len(vec3_3)

plt.hist(vec3_3, normed = True)
plt.show()
plt.close()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:10: MatplotlibDeprecationWa
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'den
  # Remove the CWD from sys.path while we load stuff.
```



Calculate analytically the probability of getting the number 7 as the sum of the numbers of the two di

```
print('The Probability of number 7 is '+ str(output3_3[7]/N))
```

```
The Probability of number 7 is 0.05
```

```
try:
    print('The Probability of number 3 is '+ str(output3_3[3]/N))
except KeyError:
    print('keyError might be thrown if the the random number generator of numpy module did no
```

```
keyError might be thrown if the the random number generator of numpy module did not gene
```

Note:

keyError might be thrown if the the random number generator of numpy module did not generate the from the two given dice.

```
output3_3
```

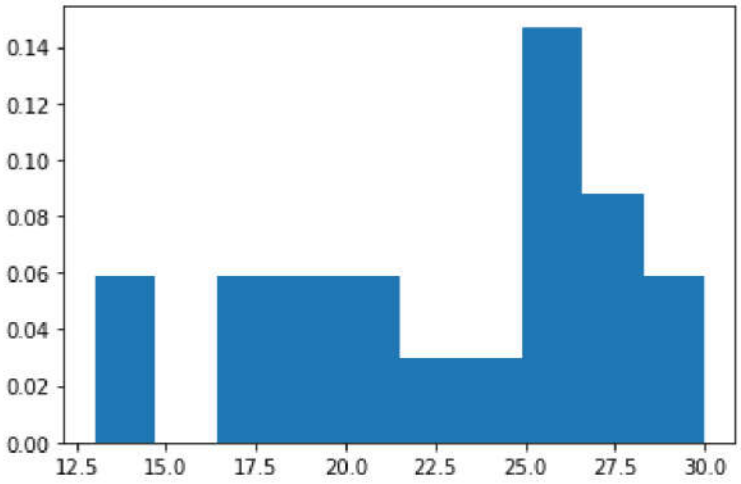
3.4. [CSCI 6505 students only] Run experiments 3.1 and 3.2 with 7 dice and plot the distribution toge and variance as calculated directly from the data as well as the fitted parameters.

```
np.random.seed(3)
vec341 = []
def Probability3_4_1():
        roll = 1
        while roll<=20:
            rand1 = np.random.randint(1,high = 7)
            rand2 = np.random.randint(1,high = 7)
            rand3 = np.random.randint(1,high = 7)
            rand4 = np.random.randint(1,high = 7)
            rand5 = np.random.randint(1,high = 7)
            rand6 = np.random.randint(1,high = 7)
            rand7 = np.random.randint(1,high = 7)
            vec341.append(rand1+rand2+rand3+rand4+rand5+rand6+rand7)
            roll+=1
Probability3_4_1()

N = len(vec341)
output = dict(Counter(vec341))
print('The Probabilities for question 3.4 (as of 3.1) are given below: ')
for i,j in zip(output.keys(),output.values()):
    print(str(i)+' : '+ str(j/N))
```

```
The Probabilities for question 3.4 (as of 3.1) are given below:
13 : 0.05
27 : 0.1
21 : 0.05
25 : 0.15
17 : 0.05
30 : 0.1
19 : 0.1
26 : 0.1
14 : 0.05
28 : 0.05
23 : 0.05
20 : 0.05
18 : 0.05
24 : 0.05
```

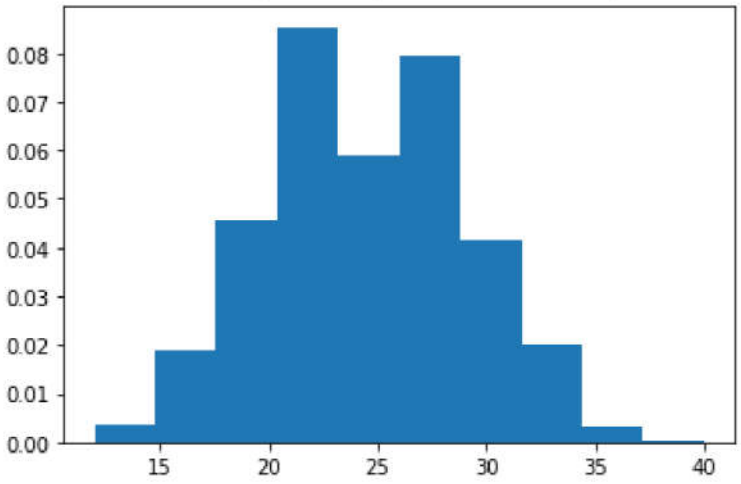
```
plt.hist(vec341, align = 'mid', normed = True)
plt.show()
plt.close()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: MatplotlibDeprecationWar
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'der
  """Entry point for launching an IPython kernel.
```

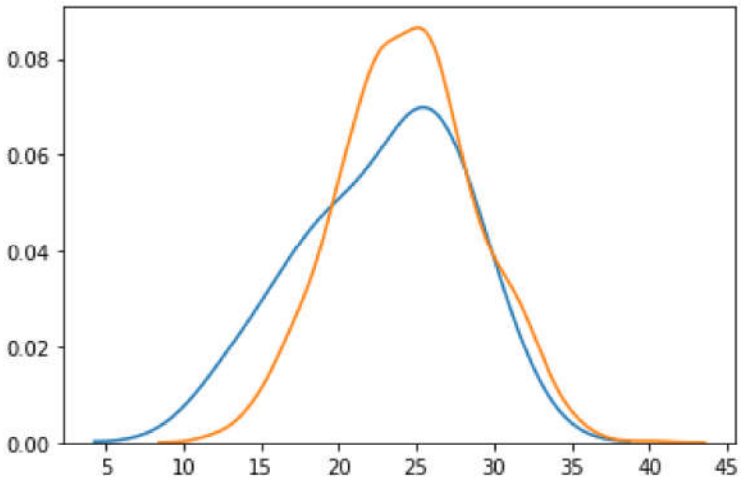



```
vec342 = []
for i in range(1000):
            rand1 = np.random.randint(1,high = 7)
            rand2 = np.random.randint(1,high = 7)
            rand3 = np.random.randint(1,high = 7)
            rand4 = np.random.randint(1,high = 7)
            rand5 = np.random.randint(1,high = 7)
            rand6 = np.random.randint(1,high = 7)
            rand7 = np.random.randint(1,high = 7)
            vec342.append(rand1+rand2+rand3+rand4+rand5+rand6+rand7)
output2 = dict(Counter(vec342))

N = len(vec342)
print('Probabilities are given below:  ')
for i,j in zip(output2.keys(),output2.values()):
    print(str(i)+' : '+ str(j/N))

plt.hist(vec342, align = 'mid', normed = True)
plt.show()
plt.close()
```

```
Probabilities are given below:
25 : 0.095
23 : 0.097
28 : 0.055
21 : 0.061
30 : 0.032
22 : 0.081
26 : 0.095
32 : 0.03
15 : 0.011
29 : 0.047
27 : 0.073
20 : 0.063
31 : 0.037
34 : 0.009
24 : 0.07
33 : 0.017
18 : 0.029
17 : 0.029
16 : 0.013
19 : 0.036
14 : 0.006
35 : 0.006
12 : 0.003
40 : 0.001
37 : 0.001
36 : 0.002
13 : 0.001
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:18: MatplotlibDeprecationWa
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'der
```




```
import seaborn as sns, numpy as np
ax = sns.distplot(vec341, hist=False)
ax1 = sns.distplot(vec342, hist=False)
```




```
# Fit Gaussian distribution
from scipy.stats import norm
# Fit Gaussian - Rolling 20 times
mean_fitted_dist20, std_fitted_dist20 = norm.fit(vec341)

# Fit Gaussian - Rolling 1000 times
mean_fitted_dist1000, std_fitted_dist1000 = norm.fit(vec342)
```

Report the mean and variance as calculated directly from the data as well as the

```
# Report the mean and variance as calculated directly from the data
mean_7dice_20times = np.mean(vec341)
print('Mean of the data for rolling 7 dices 20 times is:  ', mean_7dice_20times)
variance_7dice_20times = np.var(vec341)
print('Variance of the data for rolling 7 dices 20 times is:  ', variance_7dice_20times)

mean_7dice_1000times = np.mean(vec342)
print('\nMean of the data for rolling 7 dices 1000 times is:  ', mean_7dice_1000times)
variance_7dice_1000times = np.var(vec342)
print('Variance of the data for rolling 7 dices 20 times is:  ', variance_7dice_1000times)

# Report the mean and variance as calculated from the fitted parameters.
print('\nMean of the fitted data for rolling 20 times is:  ', mean_fitted_dist20)
print('Variance of the fitted data for rolling 20 times is:  ', std_fitted_dist20*std_fitted_
print('\nMean of the fitted data for rolling 1000 times is:  ', mean_fitted_dist1000)
print('Variance of the fitted data for rolling 1000 times is:  ', std_fitted_dist1000*std_fit
```

```
Mean of the data for rolling 7 dices 20 times is:   22.85
Variance of the data for rolling 7 dices 20 times is:   23.6275

Mean of the data for rolling 7 dices 1000 times is:   24.451
Variance of the data for rolling 7 dices 20 times is:   19.803599000000002

Mean of the fitted data for rolling 20 times is:   22.85
Variance of the fitted data for rolling 20 times is:   23.6275

Mean of the fitted data for rolling 1000 times is:   24.451
Variance of the fitted data for rolling 1000 times is:   19.803599000000002
```

Question 4

This Assignment requires you to write a Python script to calculate some inference of a simplified ver
manuscript.

Given are the following probabilities: The marginal probability that the alternator is broken is 1/1000
belt is broken is 2/100. The probability that the battery is charging when either the alternator or the fa
both are working there is a 5/1000 probability that the battery is not charging. When the battery is no
the battery is flat, though even if the battery is charging then there is a 10% chance that the battery is
the battery is flat, or there is no gas, or the starter is broken. However, even if these three conditions
won't start.

a. Draw the causal model of this system (submit picture).

Hint: You might use lea.Lea methods from the Lea 2 distribution as in the manuscript or implement it

Please see the PDF file attached.

```
#
!pip install lea
```

```
Requirement already satisfied: lea in /usr/local/lib/python3.6/dist-packages (3.2.0)
```

```
import lea
alternator_broken_a1 = lea.event(0.001)
fanbelt_broken_a2 = lea.event(0.02)

# Creating conditional probability table (CPT) based on the Directed Acyclic Graphs (DAGs)
# This part helps to reduce the number of nodes and play a vital part in the causal models
# from lea import Lea
battery_not_charging_a3 = lea.joint(alternator_broken_a1, fanbelt_broken_a2).switch({ (True
                                                    (True ,False) : lea.event(1),
                                                    (False,True ) : lea.event(1),
                                                    (False,False) : lea.event(0.005) })
```

```
battery_flat_a4 = battery_not_charging_a3.switch({ True  : lea.event(0.90), False : lea.event

car_wont_start_a7 = battery_flat_a4.switch({ True  : lea.event(1), False : lea.event(0.05) }
```

b. What is the probability that the alternator is broken given that the car won't start?

```
print("The probability of the alternator being broken given that the car wont start is: ")
lea.P(alternator_broken_a1.given(car_wont_start_a7))
```

```
The probability of the alternator being broken given that the car wont start is:
0.0054960045079625736
```

c. What is the probability that the fan belt is broken given that the car won't start?

```
print("The probability that the fan belt is broken given that the car wont start is: ")
lea.P(fanbelt_broken_a2.given(car_wont_start_a7))
```

```
The probability that the fan belt is broken given that the car wont start is:
0.1099200901592515
```

d. What is the probability that the fan belt is broken given that the car won't start and the alternator is

```
print("The probability that the fan belt is broken given that the car wont start and the alte
lea.P(fanbelt_broken_a2.given(car_wont_start_a7 & alternator_broken_a1))
```

```
The probability that the fan belt is broken given that the car wont start and the alterr
0.02
```

e. What is the probability that the alternator and the fan belt is broken given that the car won't start?

```
print("The probability that the alternator and the fan belt is broken given that the car wont
lea.P((alternator_broken_a1 & fanbelt_broken_a2).given(car_wont_start_a7))
```

```
The probability that the alternator and the fan belt is broken given that the car wont s
0.00010992009015925147
```

Question 5

Please find the PDF file attached

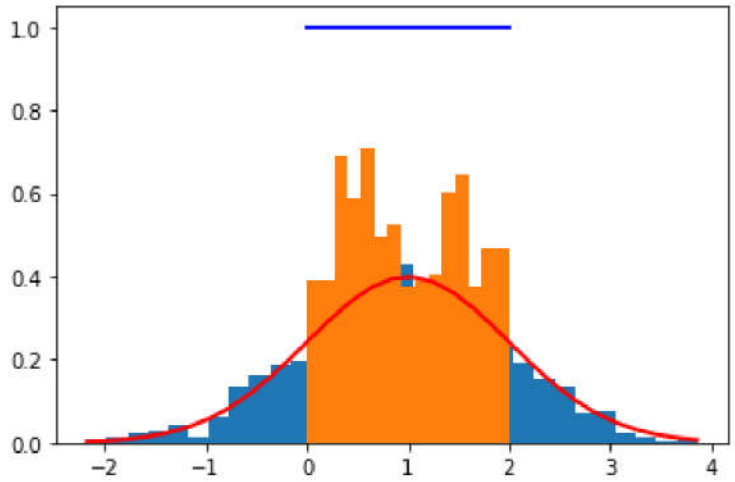
```
# Data preparation
import numpy as np
# preparing data which is normally distributed
mu, sigma = 1, 1 # mean and standard deviation
```

```
data_normal = np.random.normal(mu, sigma, 1000)
mean_normdata = np.mean(data_normal)
#print(data_normal)

# preparing data which is uniformly distributed
data_uniform = np.random.uniform(0,2,500)
mean_unidata = np.mean(data_uniform)
# print(data_uniform)


# Fit the proper distribution model to determine the Maximum Likelihood Estimation
# Data drawn from uniform distribution
from scipy.stats import uniform, norm
mean_uniform, std_uniform = uniform.fit(data_uniform)
print('Estimated parameters for the uniformly distributed class: ')
print(mean_uniform, std_uniform)
```

```
Estimated parameters for the uniformly distributed class:
0.002101282655827097 1.9950089808281257
```

```
import matplotlib.pyplot as plt
count, bins, ignored = plt.hist(data_normal , 30, density=True)
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
               np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
         linewidth=2, color='r')
#plt.show()
count, bins, ignored = plt.hist(data_uniform, 15, normed=True)
plt.plot(bins, np.ones_like(bins), linewidth=2, color='b')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7: MatplotlibDeprecationWar
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use 'der
  import sys
```




```
# Error function
# Gaussian Error function
from scipy import special
error_function = special.erf(data_normal)
```

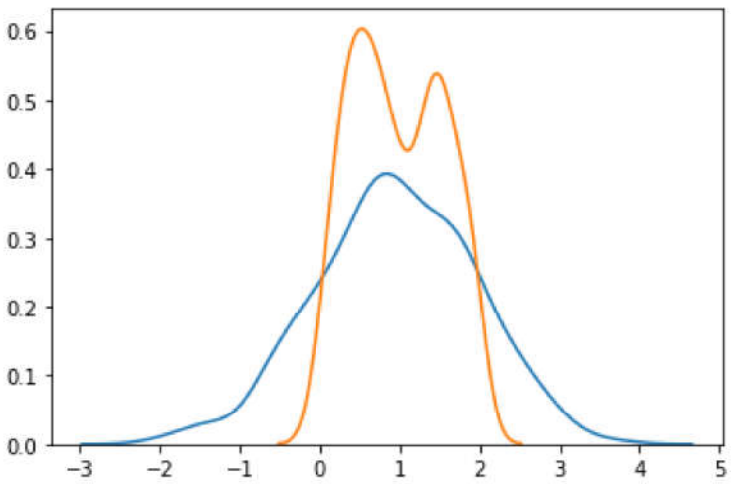
```
# The best available accuracy:
accuracy_bestavailable = 1 - special.erf(1/sqrt(2))/2
accuracy_bestavailable
```

```
0.6586552539314571
```

```
import scipy.stats
# Class of normal dist in between 0 to 2
mean_normal = 1
standard_deviation_normal = 1
probability_norm_lt = scipy.stats.norm.cdf(0, mean_normal,standard_deviation_normal)
probability_norm_gt = scipy.stats.norm.cdf(2, mean_normal,standard_deviation_normal) # greate
probability_in_between =  probability_norm_gt - probability_norm_lt
print(probability_in_between)
```

```
0.6826894921370859
```

```
import seaborn as sns, numpy as np
ax = sns.distplot(data_normal, hist=False)
ax1 = sns.distplot(data_uniform, hist=False)
```



The classification accuracy will be less than (1 - 0.32).