In [1]:
```
%matplotlib inline
import sys
import os
import copy
import numpy as np
import pandas as pd
from scipy.stats import multivariate_normal

if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

Download the IRIS data set.

In [2]:
```
def download(filename, source = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'):
    print("Downloading %s" % filename)
    urlretrieve(source + filename, filename)
```

In [3]:
```
pd_data = pd.read_csv('iris.data', sep =' ', delimiter = ',', names = ['s_length', 's_width', 'p_length', 'p
```

Split the data set into training/test data as follows: use the first 35 points in each class for training, and use the remaining 15 points for testing.

In [4]:
```
df_train_data = pd.DataFrame(pd_data.iloc[0:35, :])
df2 = pd.DataFrame(pd_data.iloc[50:85, :])
df3 = pd.DataFrame(pd_data.iloc[100:135, :])
df_train_data = df_train_data.append(df2)
df_train_data = df_train_data.append(df3)

df_test_data = pd.DataFrame(pd_data.iloc[35:50, :])
df4 = pd.DataFrame(pd_data.iloc[85:100, :])
df5 = pd.DataFrame(pd_data.iloc[135:150, :])
df_test_data = df_test_data.append(df4)
df_test_data = df_test_data.append(df5)
```

In [5]:
```
x_train_data = df_train_data.iloc[:,0:4]
y_train_label = df_train_data.iloc[:,4]

x_test_data = df_test_data.iloc[:,0:4]
y_test_label = df_test_data.iloc[:,4]
```

In [6]:
```
x_train_data.reset_index(drop = True, inplace = True)
y_train_label.reset_index(drop = True, inplace = True)

x_test_data.reset_index(drop = True, inplace = True)
y_test_label.reset_index(drop = True, inplace = True)
```

In [7]:
```
y_train_data = y_train_label.astype('category').cat.codes
y_test_data = y_test_label.astype('category').cat.codes
```

In [8]:
```
print('Shape of x_train_data:\n', x_train_data.shape)
print('\nShape of y_train_data:\n', y_train_data.shape)
```

```
Shape of x_train_data:
 (105, 4)

Shape of y_train_data:
 (105,)
```

In [9]:
```
print('Shape of x_test_data:\n', x_test_data.shape)
print('\nShape of y_test_data:\n', y_test_data.shape)
```

```
Shape of x_test_data:
 (45, 4)

Shape of y_test_data:
 (45,)
```

Build a classifier for this data set, based on multivariabte Gaussian model.

In [10]:
```
# create a multivariabte Gaussian model
def MultivariateGaussian(x, y):
    #labels 1,2,...,k
    k = 3

    #number of features
    d = x.shape[1]

    mu = np.zeros((k,d))
    sigma = np.zeros((k,d,d))
    pi = np.zeros(k)

    for label in range(k):
        indices = np.where(y == label)
        indices = indices[0]
        mu[label] = np.mean(x[indices,:], axis = 0)
        sigma[label] = np.cov(x[indices,:], rowvar = 0, bias = 1)
        pi[label] = float(len(indices))/float(len(y))
    return mu, sigma, pi
```

In [11]:
```
# get the mu, sigma, and pi
mu, sigma, pi = MultivariateGaussian(np.asarray(x_train_data), np.asarray(y_train_data))
```

In [12]:
```
print('Class probabilities:\n', pi)
```

```
Class probabilities:
 [0.33333333 0.33333333 0.33333333]
```

In [13]:
```
print('Mean of the corresponding data points:\n ', mu)
```

```
Mean of the corresponding data points:
  [[5.04571429 3.46857143 1.47714286 0.24      ]
 [6.00857143 2.76857143 4.31428571 1.34285714]
 [6.61714286 2.93714286 5.62571429 1.97714286]]
```

In [14]:
```
print('Covariance of the corresponding data points:\n ', sigma)
```

```
Covariance of the corresponding data points:
  [[[ 0.12762449  0.09972245  0.01333061  0.01445714]
  [ 0.09972245  0.13644082 -0.0012898   0.01411429]
  [ 0.01333061 -0.0012898   0.02919184  0.00462857]
  [ 0.01445714  0.01411429  0.00462857  0.0104    ]]

 [[ 0.27678367  0.09284082  0.17644898  0.0544898 ]
  [ 0.09284082  0.09929796  0.08330612  0.04391837]
  [ 0.17644898  0.08330612  0.21722449  0.07853061]
  [ 0.0544898   0.04391837  0.07853061  0.04530612]]

 [[ 0.47056327  0.12622041  0.37013061  0.04582041]
  [ 0.12622041  0.11890612  0.09247347  0.04627755]
  [ 0.37013061  0.09247347  0.35733878  0.05773061]
  [ 0.04582041  0.04627755  0.05773061  0.07262041]]]
```

In [15]:
```
# classify funtion using Bayes' rule
def decision(x, pi, mu, sigma):
    prob = np.zeros((3, x.shape[0]))

    for i in range (0,3):
        prob[i,:] = pi[i]* multivariate_normal.pdf(x, mean = mu[i,:], cov = sigma[i,:,:])

    pred = np.argmax(prob, axis = 0)

    return pred, prob
```

What error rate do you achieve?

In [16]:
```
# calculate the error rate on the test data set
accuracy = np.mean(decision(x_test_data, pi, mu, sigma)[0] == y_test_data)
error_rate = (1 - accuracy)*100

print('Error rate on the test data set:\n{}%'.format(error_rate))
```

```
Error rate on the test data set:
0.0%
```

Conclusion: For this data set, we get 100% accuracy with 0.0% error rate on this test data set.