# Python Libraries

```
import numpy as np
from pylab import unique
import matplotlib.pyplot as plt
from scipy import linalg
from numpy import linalg as LA
from numpy import *
import scipy.io as sio
import numpy as np
import pandas as pd
from scipy import stats, integrate
import matplotlib.pyplot as plt
from scipy import stats
#import seaborn as sns
#sns.set(color_codes=True)
from scipy.linalg import sqrtm
from sklearn.lda import LDA
from sklearn.qda import QDA
from sklearn.neighbors.nearest_centroid import NearestCentroid
from __future__ import division
from sklearn.metrics import accuracy_score
%pylab inline --no-import-all
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.cross_validation import StratifiedShuffleSplit
np.set_printoptions(precision=4, suppress=True)
from sklearn.metrics import accuracy_score
from sklearn import datasets
```

# Load Datasets

```
App = np.loadtxt('appendicitis.txt')
XApp, YApp = App[:,0:7], App[:,7]
Balance = np.loadtxt('balance.txt')
XBal, YBal = Balance[:,1:5], Balance[:,0]
Banana = np.loadtxt('banana.txt')
XBanana, YBanana = Banana[:,0:2], Banana[:,2]
Bands = sio.loadmat('bands.mat')
TargBands = Bands['targets']
```

```
InpBands = Bands['inputs']
YBan = TargBands.transpose()
YBands = YBan.ravel()
XBands = InpBands.transpose()
data_cancer = np.loadtxt('cancer.txt')
XCancer, YCancer = data_cancer[:,0:9], data_cancer[:,9]
BreastW = np.loadtxt('breastw.txt')
XBreastW, YBreastW = BreastW[:,0:10], BreastW[:,10]
Bupa = np.loadtxt('bupa.txt')
XBupa, YBupa = Bupa[:,0:6], Bupa[:,6]
Chess = np.loadtxt('chess.txt')
XChess, YChess = Chess[:,0:36], Chess[:,36]
data_gaussian_10 = np.loadtxt('gaussian10.txt')
XGaussian10, YGaussian10 =    data_gaussian_10[:,0:10], data_gaussian_10[:,10]
Hay = np.loadtxt('hayesroth.txt')
XHay, YHay = Hay[:,0:4], Hay[:,4]
Ilpd = sio.loadmat('ilpd.mat')
TargIlpd = Ilpd['targets']
InpIlpd = Ilpd['inputs']
YIl = TargIlpd.transpose()
YIlpd = YIl.ravel()
XIlpd = InpIlpd.transpose()
data_ionosphere = np.loadtxt('ionosphere.txt')
XIonosph, YIonosph = data_ionosphere[:,0:34], data_ionosphere[:,34]
Iris = np.loadtxt('iris.txt')
XIris, YIris = Iris[:,0:4], Iris[:,4]
Iris0 = np.loadtxt('iris0.txt')
XIris0, YIris0 = Iris0[:,0:4], Iris0[:,4]
Liver2 = np.loadtxt('liver2.txt')
XLiv2, YLiv2 = Liver2[:,0:10], Liver2[:,10]
Monk = np.loadtxt('monk.txt')
XMonk, YMonk = Monk[:,0:6], Monk[:,6]
Moon = np.loadtxt('moon.txt')
XMoon, YMoon = Moon[:,0:2], Moon[:,2]
MutBon = np.loadtxt('mutagenesisbonds.txt')
XMutBon, YMutBon = MutBon[:,0:17], MutBon[:,17]
Page = np.loadtxt('pageblocks0.txt')
XPage, YPage = Page[:,0:10], Page[:,10]
data_pima = np.loadtxt('pima.txt')
XPima, YPima = data_pima[:,0:8], data_pima[:,8]
Ring = np.loadtxt('ring.txt')
XRing, YRing = Ring[:,0:20], Ring[:,20]
Segment0 = np.loadtxt('segment0.txt')
XSeg0, YSeg0 = Segment0[:,0:19], Segment0[:,19]
Thyroid1 = np.loadtxt('newthyroid1.txt')
XThy1, YThy1 = Thyroid1[:,0:5], Thyroid1[:,5]
Thyroid2 = np.loadtxt('newthyroid2.txt')
XThy2, YThy2 = Thyroid2[:,0:5], Thyroid2[:,5]
TicTac = np.loadtxt('tictac.txt')
XTicTac, YTicTac = TicTac[:,0:9], TicTac[:,9]
```

# Density Pattern and Trace Distance

**%%%%%NORM%%%%%%%%%**

```
def norm_quadr(x):
	s = []
	i=0
	while i < (len(x)):
		s.append(x[i]**2)
		i = i+1
	somquadrati = sum(s)
	return somquadrati
```

**%%%%SQRT_NORM%%%%%%%%**

```
def sqrt_norm_quadr(x):
	norm = sqrt(norm_quadr(x))
	return norm
```

**%%%%DIFFERENCE_BETWEEN_MATRICES%%%%%%%%**

```
def diffMatrix(M1,M2):
	M1 = np.array(M1)
	M2 = np.array(M2)
	l = len(M1[0,:])
	D = np.zeros([l,l])
	for i in range(l):
		for j in range(l):
			D[i,j] = M1[i,j] - M2[i,j]
	return D
```

**%%%%DENSITY_PATTERN%%%%%%%%**

```
def rhoplusnew(x):
	stprmenouno = []
	i = 0
	while i < (len(x)):
			stprmenouno.append(x[i]/sqrt_norm_quadr(x))
			i = i + 1
	stprmenouno.append(sqrt_norm_quadr(x))
	stprnew = stprmenouno/sqrt_norm_quadr(stprmenouno)
	Pr = np.matrix(stprnew)
```

```
    PrTrasp = np.transpose(Pr)
    ProjPlus = np.dot(PrTrasp,Pr)
    #ProjPlus
    return ProjPlus
```

**%%%%TRACE_DISTANCE%%%%%%%**

```
def TraceDistance(rho1,rho2):
    MEigAbs = []
    Mdiff = diffMatrix(rho1,rho2)
    MEigvalvec = linalg.eig(Mdiff)
    for i in range(len(MEigvalvec[0])):
        MEigAbs.append(abs(MEigvalvec[0][i]))
    Td = (1/2)*sum(MEigAbs)
    return Td
```

**%%%%STATISTICAL_PARAMETERS%%%%%%%**

- FUNCTION 1: TWO-CLASS PROBLEM

```
def perf_measure(y_actual, y_hat):
    #y_hat = YClassification
    C = len(y_hat)
    y_actual1 = [y_actual[i] for i in range(len(y_actual)) if y_actual[i] == 1]
    y_actual2 = [y_actual[i] for i in range(len(y_actual)) if y_actual[i] == 2]
    #y_actual2 = [y_actual[i] for i in range(len(y_actual)) if y_actual[i] == 2]
    Len1 = len(y_actual1)
    Len2 = len(y_actual2)
    L1 = Len1/C
    L2 = Len2/C
    TP = 0
    FP = 0
    TN = 0
    FN = 0

    for i in range(len(y_hat)):
        if y_actual[i]==y_hat[i]==1:
            TP += 1
    for i in range(len(y_hat)):
        if y_hat[i]==1 and y_actual[i]!=y_hat[i]:
            FP += 1
    for i in range(len(y_hat)):
        if y_actual[i]==y_hat[i]==2:
            #if y_actual[i]==y_hat[i]==-1:
            TN += 1
    for i in range(len(y_hat)):
        if y_hat[i]==2 and y_actual[i]!=y_hat[i]:
        #if y_hat[i]==-1 and y_actual[i]!=y_hat[i]:
```

```
                FN += 1
        #TP <---> TN, FP <---> FN
        TPR1 = TP/(TP + FN)
        TPR2 = TN/(TN + FP)
        TPRMean = (Len1*TPR1+Len2*TPR2)/(Len1+Len2)
        TNR1 = TN/(TN + FP)
        TNR2 = TP/(TP + FN)
        TNRMean = (Len1*TNR1+Len2*TNR2)/(Len1+Len2)
        FPR1 = FP/(FP + TN)
        FPR2 = FN/(FN + TP)
        FPRMean = (FPR1*Len1 + FPR2*Len2)/(Len1 + Len2)
        FNR1 = FN/(FN + TP)
        FNR2 = FP/(FP + TN)
        FNRMean = (FNR1*Len1 + FNR2*Len2)/(Len1 + Len2)
        A1 = (TP + TN)/C
        B1 = ((TP + FP)*(TP + FN) + (FP + TN)*(TN + FN))/(C**2)
        A2 = (TN + TP)/C
        B2 = ((TN + FN)*(TN + FP) + (FN + TP)*(TP + FP))/(C**2)
        K1 = (A1 - B1)/(1 - B1)
        K2 = (A2 - B2)/(1 - B2)
        KMean = (K1*Len1 + K2*Len2)/(Len1 + Len2)
        Pr1 = TP/(TP + FP)
        Pr2 = TN/(TN + FN)
        PrMean = (Pr1*Len1 + Pr2*Len2)/(Len1 + Len2)
        return(1-A1, TPRMean, TNRMean, FPRMean, FNRMean, PrMean, KMean)
```

- FUNCTION 2: THREE-CLASS PROBLEM

```
def perf_measure3class(y_actual, y_hat):
        #y_hat = YClassification
        C = len(y_hat)
        y_actual1 = [y_actual[i] for i in range(len(y_actual)) if y_actual[i] == 1]
        y_actual2 = [y_actual[i] for i in range(len(y_actual)) if y_actual[i] == 2]
        y_actual3 = [y_actual[i] for i in range(len(y_actual)) if y_actual[i] == 3]
        Len1 = len(y_actual1)
        Len2 = len(y_actual2)
        Len3 = len(y_actual3)
        L1 = Len1/C
        L2 = Len2/C
        L3 = Len3/C
        TP1 = 0
        FP1 = 0
        TN1 = 0
        FN1 = 0
        TP2 = 0
        FP2 = 0
        TN2 = 0
        FN2 = 0
        TP3 = 0
```

```
FP3 = 0
TN3 = 0
FN3 = 0

for i in range(len(y_hat)):
      #if y_actual[i]==y_hat[i]==1:
      if y_actual[i]==y_hat[i]==1:
            TP1 += 1
for i in range(len(y_hat)):
      if y_hat[i]==1 and y_actual[i]!=y_hat[i]:
      #if y_hat[i]==1 and y_actual!=y_hat[i]:
          FP1 += 1
for i in range(len(y_hat)):
      if y_actual[i]!=1 and y_hat[i]!=1:
            TN1 += 1
for i in range(len(y_hat)):
      if y_actual[i]==1 and y_actual[i]!=y_hat[i]:
            FN1 += 1

for i in range(len(y_hat)):
      #if y_actual[i]==y_hat[i]==1:
      if y_actual[i]==y_hat[i]==2:
            TP2 += 1
for i in range(len(y_hat)):
      if y_hat[i]==2 and y_actual[i]!=y_hat[i]:
      #if y_hat[i]==1 and y_actual!=y_hat[i]:
          FP2 += 1
for i in range(len(y_hat)):
      if y_actual[i]!=2 and y_hat[i]!=2:
            TN2 += 1
for i in range(len(y_hat)):
      if y_actual[i]==2 and y_actual[i]!=y_hat[i]:
            FN2 += 1

for i in range(len(y_hat)):
      #if y_actual[i]==y_hat[i]==1:
      if y_actual[i]==y_hat[i]==3:
            TP3 += 1
for i in range(len(y_hat)):
      if y_hat[i]==3 and y_actual[i]!=y_hat[i]:
      #if y_hat[i]==1 and y_actual!=y_hat[i]:
          FP3 += 1
for i in range(len(y_hat)):
      if y_actual[i]!=3 and y_hat[i]!=3:
            TN3 += 1
for i in range(len(y_hat)):
      if y_actual[i]==3 and y_actual[i]!=y_hat[i]:
            FN3 += 1
#TP <---> TN, FP <---> FN
TPR1 = TP1/(TP1 + FN1)
```

```
    TPR2 = TP2/(TP2 + FN2)
    TPR3 = TP3/(TP3 + FN3)
    TPRMean = (Len1*TPR1+Len2*TPR2+Len3*TPR3)/(Len1+Len2+Len3)
    TNR1 = TN1/(TN1 + FP1)
    TNR2 = TN2/(TN2 + FP2)
    TNR3 = TN3/(TN3 + FP3)
    TNRMean = (Len1*TNR1+Len2*TNR2+Len3*TNR3)/(Len1+Len2+Len3)
    FPR1 = FP1/(FP1 + TN1)
    FPR2 = FP2/(FP2 + TN2)
    FPR3 = FP3/(FP3 + TN3)
    FPRMean = (FPR1*Len1 + FPR2*Len2 + FPR3*Len3)/(Len1 + Len2 + Len3)
    FNR1 = FN1/(FN1 + TP1)
    FNR2 = FN2/(FN2 + TP2)
    FNR3 = FN3/(FN3 + TP3)
    FNRMean = (FNR1*Len1 + FNR2*Len2 + FNR3*Len3)/(Len1 + Len2 + Len3)
    A = accuracy_score(y_actual, y_hat)
    B1 = ((TP1 + FP1)*(TP1 + FN1) + (FP1 + TN1)*(TN1 + FN1))/(C**2)
    B2 = ((TP2 + FP2)*(TP2 + FN2) + (FP2 + TN2)*(TN2 + FN2))/(C**2)
    B3 = ((TP3 + FP3)*(TP3 + FN3) + (FP3 + TN3)*(TN3 + FN3))/(C**2)
    A1 = (TP1 + TN1)/C
    A2 = (TP2 + TN2)/C
    A3 = (TP3 + TN3)/C
    K1 = (A1 - B1)/(1 - B1)
    K2 = (A2 - B2)/(1 - B2)
    K3 = (A3 - B3)/(1 - B3)
    KMean = (K1*Len1 + K2*Len2 + K3*Len3)/(Len1 + Len2 + Len3)
    Pr1 = TP1/(TP1 + FP1)
    Pr2 = TP2/(TP2 + FP2)
    Pr3 = TP3/(TP3 + FP3)
    PrMean = (Pr1*Len1 + Pr2*Len2 + Pr3*Len3)/(Len1 + Len2 + Len3)
    return(1-A, TPRMean, TNRMean, FPRMean, FNRMean, PrMean, KMean)
```

**%%%%NMC%%%%%%%%**

- FUNCTION 1: NMC OUTPUT FOR ONE RUN (TWO-CLASS PROBLEM)

```
def NMCDivisionRates(X_train,Y_train, X_test,Y_test):
    nmc = NearestCentroid()
    nmc.fit(X_train,Y_train)
    YClassification = nmc.predict(X_test)
    return perf_measure(Y_test, YClassification)
```

- FUNCTION 2: NMC OUTPUT FOR ONE RUN (THREE-CLASS PROBLEM)

```
def NMCDivisionRates3(X_train,Y_train, X_test,Y_test):
    nmc = NearestCentroid()
    nmc.fit(X_train,Y_train)
    YClassification = nmc.predict(X_test)
    return perf_measure3class(Y_test, YClassification)
```

- FUNCTION 3: NMC OUTPUT FOR 10 RUNS (TWO-CLASS PROBLEM)

```
def NMCRunsRate(X,Y):
    runs = 10
    num_param = 7
    runsQNMCDivisionRates = np.zeros([runs,num_param])
    for i in range(runs):
        X_train, X_test, Y_train, Y_test = \
        train_test_split(X,Y,test_size=0.20)
        runsQNMCDivisionRates[i,:] = NMCDivisionRates(X_train,Y_train, X_test,
Y_test)
    for j in range(num_param):
        print (np.mean(runsQNMCDivisionRates[:,j]),
np.std(runsQNMCDivisionRates[:,j]))
```

- FUNCTION 4: NMC OUTPUT FOR 10 RUNS (THREE-CLASS PROBLEM)

```
def NMCRunsRate3(X,Y):
    runs = 10
    num_param = 7
    runsQNMCDivisionRates = np.zeros([runs,num_param])
    for i in range(runs):
        X_train, X_test, Y_train, Y_test = \
        train_test_split(X,Y,test_size=0.20)
        runsQNMCDivisionRates[i,:] = NMCDivisionRates3(X_train,Y_train, X_test,
Y_test)
    for j in range(num_param):
        print (np.mean(runsQNMCDivisionRates[:,j]),
np.std(runsQNMCDivisionRates[:,j]))
```

**%%%%QNMC%%%%%%%%**

FUNCTION 1: QNMC OUTPUT FOR ONE RUN (TWO-CLASS PROBLEM)

```
def ErrorClassificationDivisionNewRates(X_train,Y_train,X_test,Y_test):
    num_patt_train = len(Y_train)
    num_patt_test = len(Y_test)
    YClassification = np.zeros(num_patt_test)
    Rho1 = []
    Rho2 = []
    Class1 = [X_train[i,:] for i in range(num_patt_train) if Y_train[i] == 1]
    Class2 = [X_train[i,:] for i in range(num_patt_train) if Y_train[i] == 2]
    for i in range(len(Class1)):
        Rho1.append(rhoplusnew(Class1[i]))
    for i in range(len(Class2)):
        Rho2.append(rhoplusnew(Class2[i]))
    QCentr1 = np.matrix(np.mean(Rho1,axis = 0))
    QCentr2 = np.matrix(np.mean(Rho2,axis = 0))
```

```
    for i in range(num_patt_test):
        if TraceDistance(rhoplusnew(X_test[i]),QCentr1) -
TraceDistance(rhoplusnew(X_test[i]),QCentr2) < 0:
            YClassification[i] = 1
        else:
            YClassification[i] = 2
            #YClassification[i] = -1
    return perf_measure(Y_test, YClassification)
```

- FUNCTION 2: QNMC OUTPUT FOR ONE RUN (THREE-CLASS PROBLEM)

```
def ErrorClassificationMultipleClassNew3Division(X_train,Y_train,X_test,Y_test):
    num_patt_train = len(Y_train)
    num_patt_test = len(Y_test)
    YClassification = np.zeros(num_patt_test)
    Rho1 = []
    Rho2 = []
    Rho3 = []
    Class1 = [X_train[i,:] for i in range(len(Y_train)) if Y_train[i] == 1]
    Class2 = [X_train[i,:] for i in range(len(Y_train)) if Y_train[i] == 2]
    Class3 = [X_train[i,:] for i in range(len(Y_train)) if Y_train[i] == 3]
    for i in range(len(Class1)):
        Rho1.append(rhoplus(Class1[i]))
    for i in range(len(Class2)):
        Rho2.append(rhoplus(Class2[i]))
    for i in range(len(Class3)):
        Rho3.append(rhoplus(Class3[i]))
    QCentr1 = np.matrix(np.mean(Rho1,axis = 0))
    QCentr2 = np.matrix(np.mean(Rho2,axis = 0))
    QCentr3 = np.matrix(np.mean(Rho3,axis = 0))
    for i in range(num_patt_test):
        if TraceDistance(rhoplus(X_test[i]),QCentr1) -
TraceDistance(rhoplus(X_test[i]),QCentr2) < 0 and
TraceDistance(rhoplus(X_test[i]),QCentr1) -
TraceDistance(rhoplus(X_test[i]),QCentr3):
            YClassification[i] = 1
        if TraceDistance(rhoplus(X_test[i]),QCentr2) -
TraceDistance(rhoplus(X_test[i]),QCentr1) < 0 and
TraceDistance(rhoplus(X_test[i]),QCentr2) -
TraceDistance(rhoplus(X_test[i]),QCentr3):
            YClassification[i] = 2
        if TraceDistance(rhoplus(X_test[i]),QCentr3) -
TraceDistance(rhoplus(X_test[i]),QCentr2) < 0 and
TraceDistance(rhoplus(X_test[i]),QCentr3) -
TraceDistance(rhoplus(X_test[i]),QCentr1):
            YClassification[i] = 3
    #print (len(Y_test))
    #print (len(Y_train))
    return perf_measure3class(Y_test,YClassification)
```

- FUNCTION 3: QNMC OUTPUT FOR 10 RUNS (TWO-CLASS PROBLEM)

```
def ErrorClassificationRunsNewRate(X,Y):
	runs = 10
	num_param = 7
	runsQNMCDivisionRates = np.zeros([runs,num_param])
	for i in range(runs):
		X_train, X_test, Y_train, Y_test = \
		train_test_split(X,Y,test_size=0.20)
		runsQNMCDivisionRates[i,:] =
ErrorClassificationDivisionNewRates(X_train,Y_train, X_test, Y_test)
	for j in range(num_param):
		print (np.mean(runsQNMCDivisionRates[:,j]),
np.std(runsQNMCDivisionRates[:,j]))
```

- FUNCTION 4: QNMC OUTPUT FOR 10 RUNS (THREE-CLASS PROBLEM)

```
def ErrorQNMCRunsMultipleNew3(X,Y):
	runs = 10
	num_param = 7
	runsQNMCDivisionNew = np.zeros([runs,num_param])
	for i in range(runs):
		X_train, X_test, Y_train, Y_test = \
		train_test_split(X,Y,test_size=0.20)
		runsQNMCDivisionNew[i,:] =
ErrorClassificationMultipleClassNew3Division(X_train,Y_train, X_test, Y_test)
	for j in range(num_param):
		print (np.mean(runsQNMCDivisionNew[:,j]),
np.std(runsQNMCDivisionNew[:,j]))
```

**%%%%CREATE_GAUSSIAN_DATASET%%%%%%%%%%**

- CODE FOR THREE-CLASS GAUSSIAN DATASET

```
def make_gaussian_dataset_3Classi(n_samples_1, n_samples_2, n_samples_3,
centro_1, centro_2, centro_3, Cov_1, Cov_2, Cov_3):

	X_1 = np.random.multivariate_normal(centro_1, Cov_1, n_samples_1)
	X_2 = np.random.multivariate_normal(centro_2, Cov_2, n_samples_2)
	X_3 = np.random.multivariate_normal(centro_3, Cov_3, n_samples_3)
	#label
	Y_1= (1)*np.ones([n_samples_1,],dtype=np.int8)
	Y_2= (2)*np.ones([n_samples_2,],dtype=np.int8)
	Y_3= (3)*np.ones([n_samples_3,],dtype=np.int8)

	#concatenate
```

```
    X=np.concatenate((X_1, X_2, X_3), axis=0)
    Y=np.concatenate((Y_1, Y_2, Y_3), axis=0)
    #Z = np.concatenate((X_3, Y_3), axis=0)
    return (X,Y)

%%%%RESCALING_PROBLEM%%%%%%%%%%%%%

def Array(X,Y):
    runsQNMC = 1
    rescQNMC = 20
    result = []
    runsQNMCDivision = np.zeros([runsQNMC,rescQNMC])
    X_train, X_test, Y_train, Y_test = \
    train_test_split(X,Y,test_size=0.20)
    for i in range(1,rescQNMC):
        for j in range(runsQNMC):
            runsQNMCDivision[j,i] =
ErrorClassificationDivisionNew(i*(1/10)*X_train,Y_train,i*(1/10)*X_test,Y_test)
            result.append(np.mean(runsQNMCDivision[:,i]))
    return   (result)
```