1. LRT

**def** likelihood(x, n, cov, mu, cov\_det, cov\_inv):

"""

Likelihood of x given y = k

    """

    p = 1/(((2\*np.pi)\*\*(n/2))\*((cov\_det)\*\*(0.5)))

    p \*= np.exp(-0.5\*(np.matmul(np.matmul((x-mu).T,cov\_inv), x-mu)))

    return p[0][0]

if \_\_name\_\_ == "\_\_main\_\_":

*# data input*

data = pd.read\_excel("./data3.xlsx",header=None)

data = data.sample(frac=1).reset\_index(drop=True)

    X = data[[0,1,2,3]]

    y = data[4]-1

*# data preprocessing*

    mscaler = NormalScaler()

    for j in range(X.shape[1]):

        mscaler.fit(X[j])

        X[j] = mscaler.transform(X[j])

*# splitting data using holdout cross validation*

    train\_percent = 0.6

    X\_train = X[:int(train\_percent\*X.shape[0])].values

    y\_train = y[:int(train\_percent\*X.shape[0])].values

    X\_test = X[int(train\_percent\*X.shape[0]):].values

    y\_test = y[int(train\_percent\*X.shape[0]):].values

    X\_train\_y1 = X\_train[y\_train==0]

    N\_y1 = X\_train\_y1.shape[0]

    cov\_y1 = np.cov(X\_train\_y1.T)

    cov\_det\_y1 = np.linalg.det(cov\_y1)

    cov\_inv\_y1 = np.linalg.inv(cov\_y1)

    mu\_y1 = np.mean(X\_train\_y1 ,axis=0).reshape(-1,1)

    X\_train\_y2 = X\_train[y\_train==1]

    N\_y2 = X\_train\_y2.shape[0]

    cov\_y2 = np.cov(X\_train\_y2.T)

    cov\_det\_y2 = np.linalg.det(cov\_y2)

    cov\_inv\_y2 = np.linalg.inv(cov\_y2)

    mu\_y2 = np.mean(X\_train\_y2,axis=0).reshape(-1,1)

    y\_test\_pred = np.ndarray((y\_test.shape))

    p\_y1 = X\_train\_y1.shape[0]/X\_train.shape[0]

    p\_y2 = X\_train\_y2.shape[0]/X\_train.shape[0]

    for i in range(X\_test.shape[0]):

        px\_y1 = likelihood(X\_test[i].reshape(-1,1), N\_y1, cov\_y1, mu\_y1, cov\_det\_y1, cov\_inv\_y1)

        px\_y2 = likelihood(X\_test[i].reshape(-1,1), N\_y2, cov\_y2, mu\_y2, cov\_det\_y2, cov\_inv\_y2)

        y\_test\_pred[i] = ((px\_y1/px\_y2) < (p\_y2/p\_y1))

    print("Accuracy: ", sum(y\_test==y\_test\_pred)/y\_test.shape[0])

    print("Sensitivity: ", sum((y\_test==1) & (y\_test\_pred==1))/sum(y\_test==1))

    print("Specificity: ", sum((y\_test==0) & (y\_test\_pred==0))/sum(y\_test==0))

2. MAP

from preprocessing import NormalScaler

**def** likelihood(x, n, cov, mu, cov\_det, cov\_inv):

"""

 Likelihood of x given y = k

    """

    p = 1/(np.power(2\*np.pi,(n/2))\*np.power(cov\_det,0.5))

    p \*= np.exp(-0.5\*np.matmul(np.matmul((x-mu).T,cov\_inv), x-mu))

    return p[0][0]

if \_\_name\_\_ == "\_\_main\_\_":

    data = pd.read\_excel("./data4.xlsx",header=None)

    data = data.sample(frac=1).reset\_index(drop=True)

    X = data[[i for i in range(7)]]

    y = data[7]

    unique\_classes = np.unique(y)

    num\_classes = len(unique\_classes)

*# data preprocessing*

    mscaler = NormalScaler()

    for j in range(X.shape[1]):

        mscaler.fit(X[j])

        X[j] = mscaler.transform(X[j])

*# splitting data using holdout cross validation*

    train\_percent = 0.7

    X\_train = X[:int(train\_percent\*X.shape[0])].values

    y\_train = y[:int(train\_percent\*X.shape[0])].values

    X\_test = X[int(train\_percent\*X.shape[0]):].values

    y\_test = y[int(train\_percent\*X.shape[0]):].values

    y\_test\_pred = np.ndarray((y\_test.shape[0], num\_classes))

    y\_test\_t = np.ndarray((y\_test.shape[0]))

    for i in range(y\_test.shape[0]):

        x = X\_test[i].T

        for j in range(num\_classes):

            tmp = X\_train[y\_train==unique\_classes[j]]

            n = tmp.shape[0]

            cov = np.cov(tmp.T)

            cov\_inv = np.linalg.inv(cov)

            cov\_det = np.linalg.det(cov)

            mu = np.mean(tmp ,axis=0).reshape(-1,1)

            p\_yk = tmp.shape[0]/X\_train.shape[0]

            y\_test\_pred[i][j] = likelihood(x, n, cov, mu, cov\_det, cov\_inv)\*p\_yk

        y\_test\_t[i] = unique\_classes[np.argmax(y\_test\_pred[i])]

*# printing confusion matrix*

    conf\_mat = np.ndarray((num\_classes, num\_classes))

    for i in range(num\_classes):

        for j in range(num\_classes):

            conf\_mat[i][j] = sum((y\_test\_t==unique\_classes[i]) & (y\_test==unique\_classes[j]))

    print(conf\_mat)

3. ML

**def** likelihood(x, n, cov, mu, cov\_det, cov\_inv):

    """

        Likelihood of x given y = k

    """

    p = 1/(np.power(2\*np.pi,(n/2))\*np.power(cov\_det,0.5))

    p \*= np.exp(-0.5\*np.matmul(np.matmul((x-mu).T,cov\_inv), x-mu))

    return p[0][0]

if \_\_name\_\_ == "\_\_main\_\_":

    data = pd.read\_excel("./data4.xlsx",header=None)

    data = data.sample(frac=1).reset\_index(drop=True)

    X = data[[i for i in range(7)]]

    y = data[7]

    unique\_classes = np.unique(y)

    num\_classes = len(unique\_classes)

*# data preprocessing*

    mscaler = NormalScaler()

    for j in range(X.shape[1]):

        mscaler.fit(X[j])

        X[j] = mscaler.transform(X[j])

*# splitting data using holdout cross validation*

    train\_percent = 0.7

    X\_train = X[:int(train\_percent\*X.shape[0])].values

    y\_train = y[:int(train\_percent\*X.shape[0])].values

    X\_test = X[int(train\_percent\*X.shape[0]):].values

    y\_test = y[int(train\_percent\*X.shape[0]):].values

    y\_test\_pred = np.ndarray((y\_test.shape[0], num\_classes))

    y\_test\_t = np.ndarray((y\_test.shape[0]))

    for i in range(y\_test.shape[0]):

        x = X\_test[i].T

        for j in range(num\_classes):

            tmp = X\_train[y\_train==unique\_classes[j]]

            n = tmp.shape[0]

            cov = np.cov(tmp.T)

            cov\_inv = np.linalg.inv(cov)

            cov\_det = np.linalg.det(cov)

            mu = np.mean(tmp ,axis=0).reshape(-1,1)

            y\_test\_pred[i][j] = likelihood(x, n, cov, mu, cov\_det, cov\_inv)

        y\_test\_t[i] = unique\_classes[np.argmax(y\_test\_pred[i])]

*# printing confusion matrix*

    conf\_mat = np.ndarray((num\_classes, num\_classes))

    for i in range(num\_classes):

        for j in range(num\_classes):

            conf\_mat[i][j] = sum((y\_test\_t==unique\_classes[i]) & (y\_test==unique\_classes[j]))

    print(conf\_mat)