1. Likelihood Ratio Test

```python
from preprocessing import NormalScaler

def likelihood(x, n, cov, mu, cov_det, cov_inv):
    """

     Likelihood of x given y = k
```

$$p\left(\frac{x}{y=k}\right) = \frac{1}{(2\Pi)^{\frac{1}{n}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-u)^T\Sigma^{-1}(x-u)}$$

```python
    """

    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]
```

```python
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
```


```python
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
```

```python
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]
```

```python
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))
```

Results:

Accuracy: 1.0

Sensitivity: 1.0

Specificity: 1.0

2. Maximum a Posteriori

```python
from preprocessing import NormalScaler

def likelihood(x, n, cov, mu, cov_det, cov_inv):
    """

     Likelihood of x given y = k
```

$$p\left(\frac{x}{y=k}\right) = \frac{1}{(2\Pi)^{\frac{1}{n}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-u)^T\Sigma^{-1}(x-u)}$$

```python
    """

    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)
```

Results:

Confusion Matrix: [[7. 0. 0.]
[10. 13. 5.]
[0. 3. 7.]]

3. Maximum Likelihood test

```
def likelihood(x, n, cov, mu, cov_det, cov_inv):
    """
        Likelihood of x given y = k
```

$$p\left(\frac{x}{y=k}\right) = \frac{1}{(2\Pi)^{\frac{1}{n}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-u)^T\Sigma^{-1}(x-u)}$$

```
    """
    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)
```

Results:

Confusion Matrix

[[4. 1. 0.]

 [9. 12. 10.]

 [0. 5. 4.]]

Name: S Rohith

Id: 2017A7PS0034H