```python
import numpy as np
from sklearn import metrics

#function [CM, acc, arrR, arrP]=func_confusion_matrix(teY, hatY)
def func_confusion_matrix(y_test, y_pred):
    """ this function is used to calculate the confusion matrix and a set of metrics.
    INPUT:
        y_test, ground-truth lables;
        y_pred, predicted labels;
    OUTPUT:
        CM, confuction matrix
        acc, accuracy
        arrR[], per-class recall rate,
        arrP[], per-class prediction rate.
    """

    y_test = np.array(y_test)
    y_pred = np.array(y_pred)

    unique_values = set(y_pred)
    sorted(unique_values)  # updates unique values to be in ascending order
    num_classes = len(unique_values)
    unique_values = np.array(list(unique_values))  # change to array so can use indexes
    possible_string_dict = {}
    # make sure all values are 0 based, so can use built-in "zip" function
    if(issubclass(type(y_test[ ]), np.integer)): # if values are integers
        y_test_min = y_test.min()
        if(y_test_min !=  ):# if does not contain 0, reduce both test and pred by min value to get 0 based for both
            y_test = y_test - y_test_min;
            y_pred = y_pred - y_test_min;
    else:
        # assume values are strings, change to integers
        # TODO, change to convert list from string to int
        y_test_int = np.empty(len(y_test), dtype=int)
        y_pred_int = np.empty(len(y_pred), dtype=int)
        for index in range( , num_classes):  # for selecting a class to work with
            current_value = unique_values[index]
            possible_string_dict[index] = current_value
            y_test_int[y_test == current_value] = index
            y_pred_int[y_pred == current_value] = index
        y_test = y_test_int
        y_pred = y_pred_int

    ## your code for creating confusion matrix;
    conf_matrix = np.zeros((num_classes, num_classes), dtype=np.int)
    for a, p in zip(y_test, y_pred):
        conf_matrix[a][p] +=


    ## your code for calcuating acc;
    accuracy = conf_matrix.diagonal().sum() / conf_matrix.sum()

    ## your code for calcualting arrR and arrP;
    recall_array = np.empty(num_classes, dtype=float)
    precision_array = np.empty(num_classes, dtype=float)
    for index in range( , num_classes):
        value = conf_matrix[index,index]
        recall_sum = conf_matrix[index,:].sum()
        precision_sum = conf_matrix[:, index].sum()
        recall_array[index] = value / recall_sum
        precision_array[index] = value / precision_sum

    return conf_matrix, accuracy, recall_array, precision_array


def get_confusion_matrix_and_test(y_test, y_pred):
    """ get confusion matrix, accuracy, array of recall and precision
        test confusion matrix and accuracy
    """
    cm, acc, arrR, arrP = func_confusion_matrix(y_test, y_pred)
    expected_matrix = metrics.confusion_matrix(y_test, y_pred)
    assert(np.array_equal(expected_matrix, cm))
    expected_acc = metrics.accuracy_score(y_test, y_pred)
    assert(round(expected_acc,  ) == round(acc,  ))
    return cm, acc, arrR, arrP

```

```
def _test_confusion_matrix():
    y_test = [ ,  ,  ,  ,  ,
               ,  ,  ,  ,  ,  ,  ,  ,
               ,  ,  ,  ,  ,  ,  ]
    y_pred = [ ,  ,  ,  ,  ,
               ,  ,  ,  ,  ,  ,  ,  ,
               ,  ,  ,  ,  ,  ,  ]
    cm, acc, arrR, arrP = get_confusion_matrix_and_test(y_test, y_pred)

def _perform1point1(confidence_threshold):
    y_test = ['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'N', 'N', 'N']
    y_pred_conf = [    ,     ,     ,     ,    ,    ,    ,    ,    ,    ,    ,    ]
    num_elements = len(y_pred_conf)
    y_pred = np.empty(num_elements, dtype=object)
    for index in range( , num_elements):
        if y_pred_conf[index] > confidence_threshold:
            y_pred[index] = 'Y'
        else:
            y_pred[index] = 'N'

    cm, acc, arrR, arrP = get_confusion_matrix_and_test(y_test, y_pred)


### Main function.  Not called if imported elsewhere as a module.
if __name__ == "__main__":
    # test with example from previous Machine Learning class homework
    # _test_confusion_matrix()
    _perform1point1(   )
```