```python
import numpy as np

# preparing input
x_coords = [-1, -1, 0, 0, 1, 1, 2, 2, 3, 3]
y_coords = [2, 1, 3, 2, 3, -1, 0, -1, 1, 0]
points = [list(x) for x in zip(x_coords, y_coords)]
classification_input = ['+', 'o', '+', 'o', 'o', '+', '+', 'o', '+', 'o']

def euclidean_dist(x1, y1, x2, y2):
    # returns euclidean distance between (x1, y1) and (x2, y2)
    return np.sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));

def nearest_neighbors(x_coords, y_coords, classification, x, y):
    # returns a list of points sorted in ascending order by distance from (x,y)
    distances = []
    for i in range(0, len(x_coords)):
            distances.append((euclidean_dist(x, y, x_coords[i], y_coords[i]),
x_coords[i], y_coords[i], classification[i]))
            # append a tuple to the list of form (euclidean_dist(x, y, xi, yi), xi,
yi, class of (xi, yi))
    distances.sort()
    # sort the list by euclidean_dist i.e. first element of the tuple (default)
    return distances

def get_knn(x_coords, y_coords, classification, x, y, k):
    # returns 'k' nearest neighbors of (x,y)
    return nearest_neighbors(x_coords, y_coords, classification, x, y)[0: k];

def knn_prediction(x_coords, y_coords, classification, x, y, k):
    # checks class of k nearest neighbours and returns the majority class as
prediction for that point
    knn = get_knn(x_coords, y_coords, classification, x, y, k)
    # print(x, y, knn)
    PLUSes = 0
    Os = 0
    for j in range(0, k):
        if(knn[j][3]=='o'):
            Os += 1;
        else:
            PLUSes += 1
    if(Os>PLUSes):
        return 'o'
    elif(PLUSes>Os):
        return '+'
    else:
        return '='

def knn(x_coords, y_coords, classification, k):
    # returns array of predicted values for all the points in the input set
    prediction = []
    for i in range(0, len(x_coords)):
        prediction.append(knn_prediction(x_coords, y_coords, classification,
x_coords[i], y_coords[i], k))
    return prediction

def getLOOCVError(x_coords, y_coords, classification, k):
    wrong_prediction = 0
    for i in range(0, len(x_coords)):
        new_x_coords = x_coords[:i]+x_coords[i+1:]
        new_y_coords = y_coords[:i]+y_coords[i+1:]
        new_classification = classification[:i]+classification[i+1:]
        element_prediction = knn_prediction(new_x_coords, new_y_coords,
new_classification, x_coords[i], y_coords[i], k)
        # print(x_coords[i], y_coords[i], element_prediction, classification[i])
        if(element_prediction != classification[i]):
            wrong_prediction += 1
    return wrong_prediction/len(x_coords)


# Running classification on the training set for an example output]
print('Actual Classes:', classification_input)
predictionk1 = knn(x_coords, y_coords, classification_input, 1)
print('Predicted Classes:')
print('k=1 ', predictionk1)
predictionk3 = knn(x_coords, y_coords, classification_input, 3)
print('k=3 ', predictionk3)
predictionk5 = knn(x_coords, y_coords, classification_input, 5)
print('k=5 ', predictionk5)
predictionk7 = knn(x_coords, y_coords, classification_input, 7)
print('k=7 ', predictionk7)
predictionk9 = knn(x_coords, y_coords, classification_input, 3)
print('k=9 ', predictionk9)

print('\nLOOCV Errors: ')
print('k=1 ', getLOOCVError(x_coords, y_coords, classification_input, 1))
print('k=3 ', getLOOCVError(x_coords, y_coords, classification_input, 3))
print('k=5 ', getLOOCVError(x_coords, y_coords, classification_input, 5))
print('k=7 ', getLOOCVError(x_coords, y_coords, classification_input, 7))
print('k=9 ', getLOOCVError(x_coords, y_coords, classification_input, 9))
```