Subject: slow processing of my k-nearest neighour code
Posted by humphreymurray on Mon, 14 Aug 2006 00:21:18 GMT
View Forum Message <> Reply to Message

Hi,

I am trying to implement a k-nearest neighbout classifier in IDL. The problem is that it's running really, really slow. After reading through much of the IDL documentation, I have managed to increase it's processing speed significantly, by reordering my arrays to make better use of contiguous memory. However it still runs quite slow. Can anybody help me make this more efficient?

Cheers, Humphrey Murray

```
; knn_classifer
; This code preforms a k-nearest neighbour classification.
; - training_data :: A 2d array containing the training data [Image
data, different bands]
; - training_classes :: A 1d array containing the classes that
represent the data [class value (integer)]
; - testing_data: A 2d array with the same dimensions as training_data,
which contains the data to be classified
; - k: The number of nearest neighbours to look at
; - result: The result of the classifier, a 1d array.

pro knn_classifier, training_data, training_classes, testing_data, k,
result

    ; Find out the sizes of the input arrays
    testing_data_sizes = size(testing_data)
    training_data_sizes = size(training_data)

    ; Check to make sure that the input arrays are of the correct
dimensions, and contain the same number of attributes
    IF training_data_sizes[0] NE 2 THEN Message, 'The training data
must be an array of 2 dimensions.'
    IF testing_data_sizes[0] NE 2 THEN Message, 'The testing data must
be an array of 2 dimensions.'
    IF testing_data_sizes[2] NE training_data_sizes[2] THEN Message,
'The training and testing data must have the same number of attributes
(i.e., the arrays need to be the same size in their first dimension)'

    ; Find out how many elements there are to test
    num_testing_elements = testing_data_sizes[1]
    num_training_elements = training_data_sizes[1]
```

```
    ; Find out the number of attributes
    num_attributes = training_data_sizes[2]

    ; A temporary storage spot
    squared = make_array(num_training_elements, num_attributes)
    euclidean = make_array(num_training_elements)

    ; Create an array for storing the results
    result = make_array(num_testing_elements, /INTEGER)
    temp_testing_data = make_array(num_training_elements,
num_attributes)

    ; calculate the distances for each training item
    for i = long(0), num_testing_elements - 1 do begin

       ; Calculate the squared distance for each attribute.
       squared = make_array(num_training_elements, num_attributes)
       for attrib = 0, num_attributes-1 do begin
         squared[*,attrib] = (testing_data[i, attrib] -
training_data[*,attrib])^2
       endfor

       ; Calculate the sums of the squared differences accross the
attributes
       euclidean = sqrt(total(squared, 2))

       ; Calculate the distances and sort the indexs of these
       sorted_indexs = sort(euclidean)

       ; Create an array that contains the classes of the items with
the k
       k_closest_classes = training_classes[sorted_indexs[0:k-1]]

       ; Store the mode (classes with the highest frequency)
       result[i] = mode(k_closest_classes)

    endfor

end
```