Subject: Re: slow processing of my k-nearest neighbour code
Posted by humphreymurray on Mon, 14 Aug 2006 13:50:56 GMT
View Forum Message <> Reply to Message

Oh yes, I should have explained it in better detail.

I am trying to classify pixels within an image into various classes(or regions) using a k-nearest neighbour classifier (http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm). It is this classifier that I am trying to implement.

The imagery that I am dealing with consists of any number of bands. These bands may be the standard RGB bands, or they may be the result of some other calculation. Two different sets of pixels need to be passed into this function. The first set of these is the training data. This is a collection of pixels of which the regions that they come from is known. This information originates from a 2d image, however, I have converted it into a linear vector. I have then used the 2nd dimension of the array to specify what band the pixels are from.

There is also a 1d vector that is passed in that contains the regions that the known pixels belong to. I have just used integers to identify these.

The final input data is what I've called the testing data. This is a collection of pixels of which we don't know what region they belong to. The procedure I have written is there to calculate what region they belong to. This is done by plotting these pixels in feature space. For example, if there are 2 bands in the image, then one of these bands could be plotted along the x axis, and the other along the y. Then the distance between the pixel in question, and every training pixel is calculated. The k closest pixels are then looked at the see which region is the most common among those neighbouring pixels. This region is the result for that pixel.

So in my code which I posted previously. I am looping through all of the pixels to be classified, and then for each of these pixels, I am computing the distances, etc. The code that I have written works. I have tested the results with values that I calculated by hand. However it runs extremely slow. One reason for this is that if I am trying to classify every pixel within a 256x256 pixel image, then the other loop of my code has to run about 65,500 times. When I add a nested loop, this slows down my code even more. Any ideas as to make it more efficient would be great, thanks.

Humphrey

On 8/14/06, Bakim wrote:

Hello!!

Can you explain more on your program...what is the input data...
whether you input an array or read from a image first..I would
appreciate if you explian more on your algorithms used for
classification.

Regard's

humphreymurray@gmail.com wrote:

```
>  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
```