```
%copyright Charles Davi 2022
%BLACK TREE AUTOML
%HUMANITY'S FASTEST DEEP LEARNING SOFTWARE

%=====================================================================
%=====================================================================
%VECTORIZED COMPUTATIONAL GENOMICS - mtDNA Dataset Command Line
%=====================================================================
%=====================================================================

%this code approximates the number of bases that would appear using a
local alignment
%however it still uses a global allignment, and is statistcal

%=====================================================================
%RUNS NEAREST NEIGHBOR ON SINGLE CLASS ROW
%=====================================================================

%class in question
C = 31;

%finds the rows for the class in question
x = find(dataset(:,N+1) == C);
num_items = num_rows_vector(C);

%housekeeping initialization
class_distribution = zeros(1, num_classes);

%optional percentage threshold instead of avg + std. dev
M = .5*N;

%======================================================================
==========
%initialization for inner j-loop values
%======================================================================
==========

%the number of segments the genome is broken into
num_segments = 100;

%stores indexes of matching genomes
matching_genome_indexes = [];

%the size of each segment
segment_size = floor(N/num_segments);

%a matrix to store the results for each genome in the class
match_distribution_matrix = zeros(num_items, num_segments);

%beginning of outer loop
```

```
for i = 1 : num_items

index = x(i);

%runs nearest neighbor on it
[nearest_neighbor, match_count, match_vector, match_matrix] =
Genetic_Nearest_Neighbor_Single_Row(index, dataset, N);

%finds all rows that have a match above the threshold
y = find(match_vector >= M);

%stores the average match count for the genome
avg_match_vector(i) = mean(match_vector(y));

matching_genome_indexes = [matching_genome_indexes y'];

%counts the number of matches
num_matches = size(y,1)

  %builds a distribution of matching bases within each segment, for
each match
  for j = 1 : num_matches

    %the last index of a segment initialized to 0
    end_index = 0;

    %the full match vector
    genome_match_vector = match_matrix(y(j),:);

    %iterates over each segment
    for k = 1 : num_segments

      %the start and end index of the segment
      initial_index = end_index + 1;
      end_index = min(initial_index + segment_size - 1, N);
      match_distribution_matrix(i,k) = match_distribution_matrix(i,k)
+ sum(genome_match_vector(initial_index : end_index));

    endfor

  endfor

  %potentially two sizes for the segments due to truncation
  %the first is the full segment size
  match_distribution_matrix(i,1 : k-1) =
match_distribution_matrix(i,1 : k-1)./(segment_size*num_matches);

  %the second is the last segment size
  end_segment_size = 1 + N - ((num_segments-1)*segment_size) + 1;
  %normalizes the last column
  match_distribution_matrix(i,k) = match_distribution_matrix(i,k)/
(end_segment_size*num_matches);

endfor
%end of outer i-loop

figure
hold

for i = 1 : num_items

  plot(match_distribution_matrix(i,:))

endfor

num_matching_genomes = size(unique(matching_genome_indexes),2);
```