```
function [predicted_class_vector delta_vector prediction_vector_array modal_probability_vector cluster_size] =
MASS_Sup_BlackTree(sorted_dataset,N)

%Copyright Charles Davi, all rights reserved

%loads dataset

num_rows = size(sorted_dataset,1);

%finds the testing rows in the sorted dataset
sorted_testing_rows = find(sorted_dataset(:,N+1) == -1);
num_testing_rows = size(sorted_testing_rows,1);

RH_cluster_size = zeros(1,num_testing_rows);
LH_cluster_size = zeros(1,num_testing_rows);

%applies prediction
for i = 1 : num_testing_rows

  %loads the testing row
  testing_row = sorted_testing_rows(i);
  testing_vector = sorted_dataset(i,:);

  %initial values
  RH_boundary_vector = testing_vector;
  LH_boundary_vector = testing_vector;

  %right hand side--------------------------------------------------

  %if true, it's not the end of the list
  if(testing_row + 1 <= num_rows)

    j = testing_row + 1;
    RH_class(i) = sorted_dataset(j,N+1);

    %if true, then the adjecent entry is a training row
    if(RH_class(i) != -1)

      RH_cluster_size(i) = 1;
      break_loop = 0;

      %finds the cluster until first error
      while(break_loop == 0 && j <= num_rows)

        test_class = sorted_dataset(j,N+1);

        %if true, then we increment the cluster size
        if(test_class == RH_class(i))

          RH_cluster_size(i) = RH_cluster_size(i) + 1;

        %otherwise, we break the loop
        else

          break_loop = 1;

        endif

        j = j + 1;
      endwhile

      RH_boundary_vector = sorted_dataset(j - 1,:);
      RH_delta = norm(RH_boundary_vector .- testing_vector);

    %otherwise, it's a testing row, we flag a rejection
    else

      RH_class(i) = -1;
      RH_delta = 0;

    endif

  %otherwise, it's the end of the list, we flag a rejection
  else

    RH_class(i) = -1;
    RH_delta = 0;

  endif

  %left hand side---------------------------------------------------

  %if true, it's not the first entry in the list
  if(testing_row - 1 > 0)

    j = testing_row - 1;
    LH_class(i) = sorted_dataset(j,N+1);

    %if true, then the adjecent entry is a training row
    if(LH_class(i) != -1)

      LH_cluster_size(i) = 1;
      break_loop = 0;

      %finds the cluster until first error
      while(break_loop == 0 && j > 0)

        test_class = sorted_dataset(j,N+1);

        %if true, then we increment the cluster size
        if(test_class == LH_class(i))

          LH_cluster_size(i) = LH_cluster_size(i) + 1;

        %otherwise, we break the loop
        else

          break_loop = 1;

        endif

        j = j - 1;

      endwhile

      LH_boundary_vector = sorted_dataset(j + 1,:);
      LH_delta = norm(RH_boundary_vector .- testing_vector);

      delta_vector(i) = min(RH_delta, LH_delta);
    %otherwise, it's a testing row, we flag a rejection
    else

      LH_class(i) = -1;
      delta_vector(i) = 0;

    endif

    %otherwise, it's the beginning of the list, we flag a rejection
    else

      LH_class(i) = -1;
      delta_vector(i) = 0;

    endif


endfor

%stores clusters and predictions------------------------------------
for i = 1 : num_testing_rows

  testing_row = sorted_testing_rows(i);

  %finds the cluster for each testing row---------------------------
  if(RH_class(i) != -1 && LH_class(i) != -1)

    %righthand prediction
    RH_temp_cluster_size = RH_cluster_size(i);
    RH_prediction_vector = RH_class(i)*ones(1,RH_temp_cluster_size); %creates a vector with class labels

    %lefthand prediction
    LH_temp_cluster_size = LH_cluster_size(i);
    LH_prediction_vector = LH_class(i)*ones(1,LH_temp_cluster_size); %creates a vector with class labels

    prediction_vector_array{i} = [RH_prediction_vector LH_prediction_vector];

    cluster_size(i) = RH_cluster_size(i) + LH_cluster_size(i);

  %otherwise the testing row was either the front or end of the list, rejection
  else

    prediction_vector_array{i} = [];
    cluster_size(i) = 0;

  endif

  %generates predictions--------------------------------------------

  %if true, then the cluster is empty, or the classes are not equal, a rejection
  if((size(prediction_vector_array{i},2)) == 0 || LH_class(i) != RH_class(i))

    predicted_class_vector(i) = -1;
    delta_vector(i) = 0;
    modal_probability_vector(i) = 0;

  %otherwise, we find the modal class
  else
```

```
    prediction_vector = prediction_vector_array{i};
    predicted_class = mode(prediction_vector);
    predicted_class_vector(i) = predicted_class;

    x = find(prediction_vector == predicted_class);
    mode_density = size(x,2);
    num_items = size(prediction_vector,2);
    modal_probability_vector(i) = mode_density/num_items;

  endif

endfor

endfunction
```