Main code

```
clear

clc

branchPath = {};

table = readtable('id3_data.csv');
iteration = 0;
informationGainSum = 0;


tableSize = size(table);
nRows = tableSize(1);
nColumns = tableSize(2);
targetColumn = table{:,end};
partitionMapCellArray = {};

% Finding the names of the attributes. Storing in attributesCollection.
attributesCollection = IdentifyAttributes(table,nRows,nColumns);
targetAttributes = attributesCollection{:,end};

% Computing H(S).
originalEntropy = ComputeEntropy(table{:,end}, attributesCollection{:,end})
targetEntropy = originalEntropy;

% Producing a branch that divides the table according to the ID3 algorithm.
while targetEntropy > 0
    iteration = iteration + 1;

    % Finding the partition with highest information gain.
    informationGainVector = zeros(1,nColumns-1);
    for j = 1:nColumns-1
        nAttributes = length(attributesCollection{:,j});
        attributeColumn = table{:,j};
        attributes = attributesCollection{:,j};
        if nAttributes == 2

            [informationGain, partitionMapCellArray{1,j},...
                conditionalEntropy] = ComputeIG(attributeColumn,...
                attributes, targetColumn, targetAttributes, targetEntropy);

            informationGainVector(j) = informationGain;


        elseif nAttributes == 3

            % 4 possible partitions for 3 attributes.
            % Finding and choosing the one with most IG.
            [informationGain, ~, partitionMapCellArray{1,j},...
                conditionalEntropy] = ComputeMaxIG3(attributeColumn,...
                attributes, targetColumn, targetAttributes, targetEntropy);
```

```
            informationGainVector(j) = informationGain;

        end
    end

    % Determining the best partitioning.
    [maxInformationGain, index] = max(informationGainVector);
    minConditionalEntropy = targetEntropy - maxInformationGain


    % Picking a subset from the optimal
    % partitioning for the next iteration.
    chosenAttribute = attributesCollection{index};
    branchPath{1,iteration} = chosenAttribute{1};
    chosenPartitionMap = partitionMapCellArray{index};
    table = table(chosenPartitionMap{1},:)
    informationGainSum = informationGainSum + maxInformationGain;

    % Computing H(S|A).
    targetEntropy = ComputeEntropy...
        (table{:,end}, attributesCollection{:,end})




    % From this point on, the same steps are taken, but for the new
    % subset table.
    tableSize = size(table);
    nRows = tableSize(1);
    nColumns = tableSize(2);
    targetColumn = table{:,end};
    partitionMapCellArray = {};

    % Finding the names of the attributes again.
    attributesCollection = IdentifyAttributes(table,nRows,nColumns);
    targetAttributes = attributesCollection{:,end};



end
```

IdentifyAttributes.m

```
function attributesCollection = IdentifyAttributes(table,nRows,nColumns)

    attributesCollection = {};
    for j = 1:nColumns
        attributes = table{1,j};
        for i = 2:nRows
```

```
            s = table{i,j};
            isNewString = abs(sum(strcmp(s,attributes))-1); %Check later.
            if isNewString
                attributes{end+1} = s{:};
            end
        end
        attributesCollection{1,j} = attributes;
    end

end
```

ComputeEntropy.m

```
function entropy = ComputeEntropy(inputCellArr, attributes)

    nAttributes = length(attributes);
    countVector = zeros(1,nAttributes);
    nRows = length(inputCellArr);


    if nAttributes == 1
        entropy = 0;
        return
    end


    for i = 1:nRows
        s = inputCellArr{i};
        countVector = countVector + strcmp(s,attributes);
    end
    probabilityVector = countVector/nRows;
    entropy = 0;
    for i = 1:nAttributes
        if probabilityVector(i) ~= 0
            entropy = entropy - ...
                probabilityVector(i)*log2(probabilityVector(i));
        else
            entropy = 0;
        end
    end

end
```

ComputeIG.m

```
function [informationGain, attributeLocationCollection,...
    conditionalEntropy] = ComputeIG(attributeColumn, attributes,...
```

```
    targetColumn, targetAttributes, targetEntropy)

nRows = length(targetColumn);
conditionalEntropy = 0;
nAttributes = length(attributes);
attributeLocationCollection = {};
for k = 1:nAttributes
    attribute = attributes{1,k};
    attributeLocations = ismember(attributeColumn, attribute);
    targetPartition = targetColumn(attributeLocations);
    attributeLocationCollection{1,k} = attributeLocations;
    targetPartitionEntropy = ComputeEntropy(targetPartition,...
        targetAttributes);
    proportion = length(targetPartition)/nRows;
    conditionalEntropy =...
        conditionalEntropy + proportion * targetPartitionEntropy;
end
informationGain = targetEntropy - conditionalEntropy;
```

ComputeMaxIG3.m

```
function [informationGain, bestPartition, partitionMap,...
    conditionalEntropy] = ComputeMaxIG3(attributeColumn, attributes,...
    targetColumn, targetAttributes, originalEntropy)

    informationGain = 0;
    for mPartition = 1:4
        pair = {};
        if mPartition == 1
            pair{1,1} = attributes{1,1};
            pair{1,2} = attributes{1,2};
            attributeSplit{1,1} = pair;
            attributeSplit{1,2} = attributes{1,3};
        elseif mPartition == 2
            pair{1,1} = attributes{1,2};
            pair{1,2} = attributes{1,3};
            attributeSplit{1,1} = pair;
            attributeSplit{1,2} = attributes{1,1};
        elseif mPartition == 3
            pair{1,1} = attributes{1,1};
            pair{1,2} = attributes{1,3};
            attributeSplit{1,1} = pair;
            attributeSplit{1,2} = attributes{1,2};
        elseif mPartition == 4
            attributeSplit = attributes;
        end

        [newIG, tempLocations, conditionalEntropy] = ...
            ComputeIG(attributeColumn, attributeSplit,...
            targetColumn, targetAttributes, originalEntropy);
```

```
        % This part makes sure it is the max IG split that is saved.
        if informationGain < newIG
            informationGain = newIG;
            bestPartition = attributeSplit;
            partitionMap = tempLocations;
        end
    end
```