```swift
//
//  AICode.swift
//  Hello World
//
//  Created by Charles Davi on 9/27/21.
//  Copyright © 2021 Charles Davi. All rights reserved.
//

import Foundation

//runs nearest neighbor on dataset
func runNNDataset(dataset:[Double], numRows: Int, numCols: Int)
-> [Int]{

    var nearestNeighbors = [1]
    nearestNeighbors.remove(at:0)

    for i in 0 ... numRows - 1{

        var NNRow = findNN(dataset: dataset, inputRow: i,
numRows: numRows, numCols: numCols)

        nearestNeighbors.insert(NNRow, at: i)

        var index: Int = getMatrixEntry(row: NNRow, col:numCols
- 1, numCols:numCols)
        var predictedClass = dataset[index]

        index = getMatrixEntry(row: i, col:numCols - 1,
numCols:numCols)
        var actualClass = dataset[index]

        if(predictedClass != actualClass){

            numErrors = numErrors + 1
        }

    }

    return(nearestNeighbors)
}

//finds the nearest neighbor of a vector in a dataset
func findNN(dataset: [Double], inputRow: Int, numRows: Int,
numCols: Int) -> Int{

    var NNRow = -1
```

```
    var minNorm = Double.greatestFiniteMagnitude
    var inputVectorIndexes = getMatrixRow(row: inputRow,
numCols: numCols)
    var inputVector = loadVector(dataset: dataset,
vectorIndexes: inputVectorIndexes)

    inputVector.remove(at: numCols - 1)

    for i in 0 ... numRows - 1{

        if(i != inputRow){

            var testVectorIndexes = getMatrixRow(row: i,
numCols: numCols)
            var testVector = loadVector(dataset:dataset,
vectorIndexes: testVectorIndexes)

            testVector.remove(at: numCols - 1)

            var norm = getNorm(vector1: inputVector, vector2:
testVector)

            if(norm < minNorm){

                minNorm = norm
                NNRow = i

            }

        }

    }

    return(NNRow)

}

//generates clusters for each row in a dataset given delta
func generateCluster(dataset: [Double], inputRow: Int, numRows:
Int, numCols: Int, clusterMatrix:[Int], delta: Double) -> [Int]{

    var newClusterMatrix = clusterMatrix
    var minNorm = Double.greatestFiniteMagnitude
    var inputVectorIndexes = getMatrixRow(row: inputRow,
numCols: numCols)
    var inputVector = loadVector(dataset: dataset,
vectorIndexes: inputVectorIndexes)
```

```
    inputVector.remove(at: numCols - 1)

    for i in 0 ... numRows - 1{

        var testVectorIndexes = getMatrixRow(row: i, numCols:
numCols)
        var testVector = loadVector(dataset:dataset,
vectorIndexes: testVectorIndexes)

        testVector.remove(at: numCols - 1)

        var norm = getNorm(vector1: inputVector, vector2:
testVector)

        if(norm <= delta){

            var index = getMatrixEntry(row: inputRow, col: i,
numCols: numRows)
            newClusterMatrix[index] = 1

        }

    }

    return(newClusterMatrix)
}

//returns the rows that are in a given cluster
func getClusterRows(clusterMatrix: [Int], inputRow: Int,
numCols: Int) -> [Int]{

    var clusterRows = [0]
    clusterRows.remove(at: 0)

    for i in 0 ... numCols - 1{

        var index = getMatrixEntry(row: inputRow, col: i,
numCols: numCols)
        var currentEntry = clusterMatrix[index]

        //if true, then that row is in the cluster for inputRow
        if(currentEntry == 1){

            clusterRows.insert(i, at: clusterRows.endIndex)
        }
    }
```

```
    return(clusterRows)

}

//returns the classes of the rows in a cluster
func getClassVector(clusterRows: [Int], dataset: [Double],
numCols: Int) -> [Int]{

    var numEntries = clusterRows.count
    var classVector:[Int] = []

    for i in 0 ... numEntries - 1 {

        var currentClassIndex = getMatrixEntry(row:
clusterRows[i], col: numCols - 1, numCols: numCols)
        var currentClass = Int(dataset[currentClassIndex])
        classVector.insert(currentClass, at:
classVector.endIndex)

    }

    return(classVector)
}

//returns the distribution of classes within a cluster, as
integer frequences
func getDistributionVector(classVector: [Int],
uniqueClassVector: [Int]) -> [Int]{

    var numClasses = uniqueClassVector.count
    var numEntries = classVector.count
    var distributionVector = getZeros(numRows: 1, numCols:
numClasses)

    for i in 0 ... numClasses - 1{

        var currentClass = uniqueClassVector[i]

        for j in 0 ... numEntries - 1{

            var currentEntry = classVector[j]

            if(currentEntry == currentClass){

                distributionVector[i] = distributionVector[i] +
1
```

```
            }
        }
    }

    return(distributionVector)
}

//returns the index of the inputClass in the uniqueClassVector
func getClassIndex(uniqueClassVector:[Int], inputClass: Int) -
>Int{

    var numClasses = uniqueClassVector.count
    var classIndex = -1

    for i in 0 ... numClasses - 1{

        var currentClass = uniqueClassVector[i]

        if(inputClass == currentClass){

            classIndex = i
        }
    }

    return(classIndex)
}

//returns a measure of confidence rooted in information theory
func calcConfidence(numClasses: Double, classVector: [Int],
entropy: Double) -> Double{

    var numEntries = Double(classVector.count)
    var confidence = numEntries*(log(numClasses)/log(2.0)) -
numEntries*entropy

    return(confidence)
}
```