Trabalho 1.1. Classificadores simples usando k-Nearest Neighbour, Hamming, etc

Implementar um programa que satisfaça as seguintes propriedades:

- O programa deve ser capaz de trabalhar com 4 tipos de conjuntos de dados:
  - Um conjunto de dados 2D (duas variáveis apenas)
  - Pelo menos dois conjuntos de dados ND (com um grande número de variáveis)
  - Conjuntos de dados de espiral simples (imagem).
  - Conjuntos de dados de espiral dupla (imagem).
- Implemente o cálculo da Distância de Hamming+ e da Distância Euclidiana entre padrões de dimensões arbitrárias, de maneira que possam ser usados para todos os conjuntos de dados gerados.
- Implemente o algoritmo de NN (Nearest Neighbour) e kNN (k-Nearest Neighbours) e teste em todos os 4 conjuntos de dados.

:information_source: Observações:

- Os dados 2D e nD podem ser utilizados a partir de conjuntos de dados disponíveis nos sites sugeridos ou na Página de Dados da Disciplina.
- Os dados em espiral devem ser gerados a partir de algoritmos que você mesmo vai implementar. Inclua no algoritmo a possibilidade de introduzir ruído na geração dos dados (através, por + exemplo, de uma variável aleatória que modifica levemente o comprimento do raio gerado para o próximo ponto).
- Observe que na espiral dupla cada braço da espiral em seu total representa uma classe.
- O programa deve ser capaz de desenhar os padrões na tela com cores distintas para cada classe.
- Deve ser possível entrar com padrões arbitrários (2D e ND) para que o programa os classifique.

Como foi feito?

Esse trabalho foi desenvolvido utilizando a linguagem de programação R e utiliza as bibliotecas `readxl` para leitura de arquivos xlsx, `hashmap` para utilizar hashmap e `deldir` para triangulação Delaunay.

```
install.packages(c("readxl", "hashmap", "deldir"), dependencies=TRUE)
```

Datasets

Para facilitar a criação dos pontos dos datasets foram criados arquivos que devem obedecer ao método `point`. Um exemplo de dataset encontrado na pasta `/dataset` é a espiral (**spiral.R**) que pode ser criada configurando o número de pontos, a taxa de crescimento, a direção do crescimento e um identificador.

```
Spiral = setRefClass("Spiral",
  contains = "Dataset",
  fields = list(length = 'numeric', growthSpeed = 'numeric', inverseGrowth =
'logical', identifier = 'character'),
  methods = list(
     initialize = function(length, growthSpeed, inverseGrowth = FALSE, identifier =
paste(c('id', sample(1:100, 1)), collapse = " ")) {
      .self$length = length
```

```
      .self$growthSpeed = growthSpeed
      .self$inverseGrowth = inverseGrowth
      .self$identifier = identifier
     },

    points = function() {
      spiralPoints = matrix(0, length, 2)

      for (index in 1:length) {
          angle = growthSpeed * index;
          x = (1 + angle) * cos(angle);
          y = (1 + angle) * sin(angle);
          spiralPoints[index, ] = c(x, y)
      }
      if (inverseGrowth) {
          spiralPoints = spiralPoints * -1
      }

      cbind(spiralPoints, matrix(identifier, length, 1))
    }
  )
)
```

Os pontos da espiral podem ser obtidos como o exemplo abaixo:

```
spiral  = Spiral$new(700, 0.1, FALSE, 'spiral')
spiralPoints = spiral$points();
```

Para dataset de duas dimensões escolhi utilizar o bank.xlsx extraindo somente as variáveis *Duration of Credit Month* e *Credit Amount dimensions*.

Calculadores de distância (euclidean e hamming+)

Foi criado uma abstração para os algoritmos que calculam distância. Novos calculadores devem herdar de `distance_calculator/distance_calculator.R` e implementarem o método `calculate`. Veja os calculadores implementados:

Abstração - **distance_calculator.R**

```
distanceCalculator = setRefClass("distanceCalculator",
    methods = list(
      calculate = function() {
        print('Must implement!')
      }
    )
)
```

euclidean_distance_calculator.R

```
euclideanDistanceCalculator <- setRefClass("euclideanDistanceCalculator",
    contains = "distanceCalculator",
    methods = list(
        calculate = function(dimensionsOne, dimensionsTwo) {
            sqrt(sum((as.numeric(dimensionsOne) - as.numeric(dimensionsTwo)) ^ 2))
        }
    )
)
```

hamming_distance_calculator.R

```
hammingDistanceCalculator <- setRefClass("hammingDistanceCalculator",
    contains = "distanceCalculator",
    methods = list(
        calculate = function(vectorOne, vectorTwo) {
            sum(abs(as.numeric(vectorOne) - as.numeric(vectorTwo)))
        }
    )
)
```

Classificadores NN (Nearest Neighbour) e kNN (k-Nearest Neighbours)

Assim como as calculadores de distância os classificadores também foram abstraidos. Eles devem herdar de `classifiers/classifier.R` e implementar o método `classify`.

Os classificadores recebem uma injeção de depedência de `distanceCalculator` e um dataset no formato de matrix durante sua construção.

Abstração - **classifier.R**

```
classifier = setRefClass("classifier",
    fields = list(distanceCalculator = 'distanceCalculator', dataset = 'matrix'),
    methods = list(
       initialize = function(distanceCalculator, dataset) {
        .self$distanceCalculator = distanceCalculator
        .self$dataset = dataset
       },

      classify = function(point) {
        print('Must implement!')
      }
    )
)
```

nearest_neighbour_classifier.R

```
nearestNeighbourClassifier = setRefClass("nearestNeighbourClassifier",
    contains = "classifier",
    methods = list(

      classify = function(point) {
        datasetDimensions = dim(dataset)
        clazzColumnIndex = datasetDimensions[2]
        clazz = dataset[1,clazzColumnIndex]
        shortDistance = 10e100

        rowSize = datasetDimensions[1]
        for(rowNumber in 1:rowSize) {
            row = dataset[rowNumber, 1:clazzColumnIndex-1]
            class(row)<- "numeric"
            distance = distanceCalculator$calculate(point, row)
            if(distance < shortDistance) {
                shortDistance = distance
                clazz = dataset[rowNumber, clazzColumnIndex]
            }
        }
```

```
        clazz
      }
    )
)
```

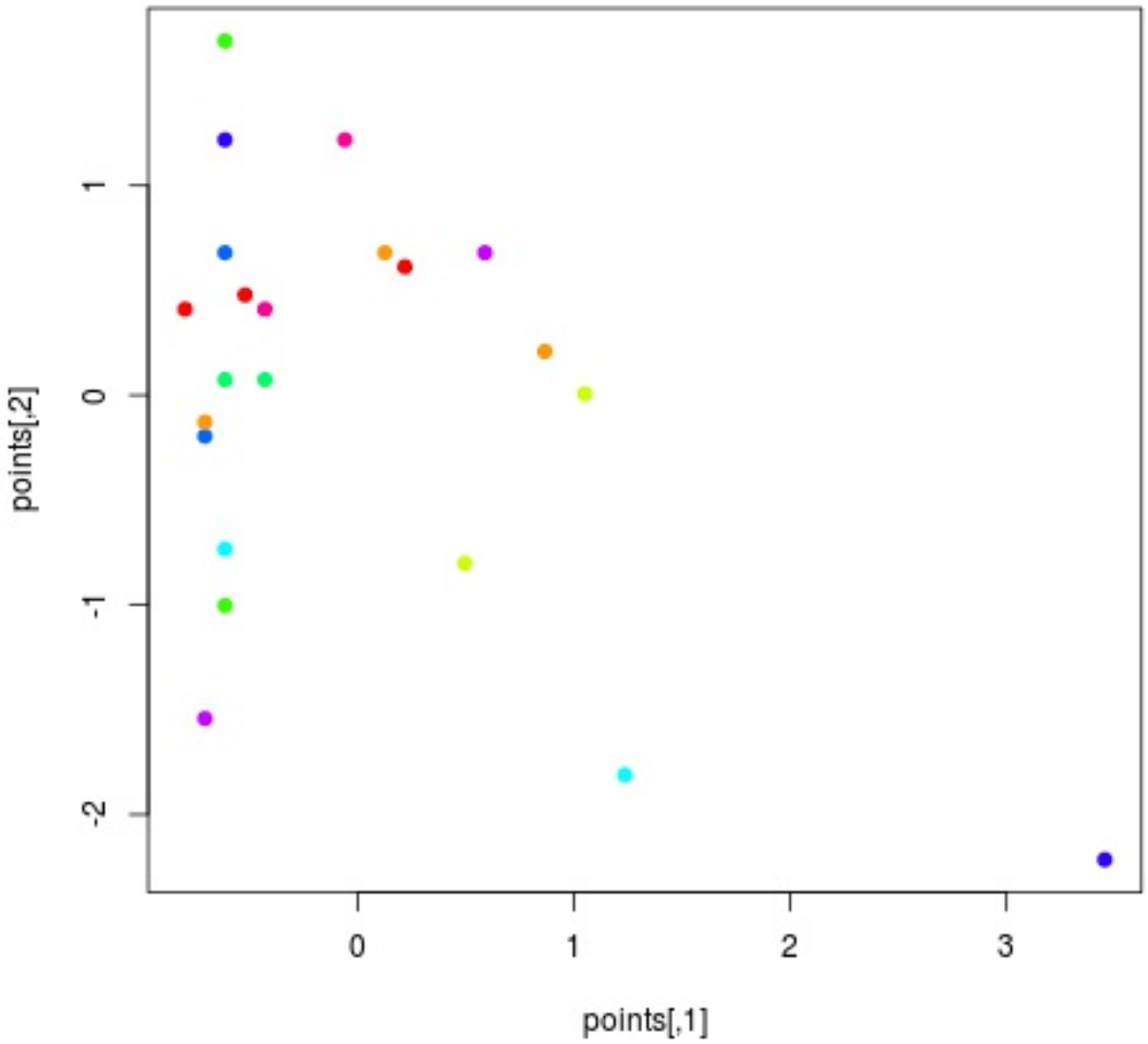
k_nearest_neighbour_classifier.R

```
kNearestNeighbourClassifier = setRefClass("kNearestNeighbourClassifier",
    contains = "classifier",
    methods = list(

      classify = function(point, k = 5) {
        datasetDimensions = dim(dataset)
        rowSize = datasetDimensions[1]
        clazzColumnIndex = datasetDimensions[2]

        shortsDistances = hashmap(1e100, dataset[1, clazzColumnIndex])

        for(rowNumber in 1:rowSize) {
            clazz = dataset[rowNumber, clazzColumnIndex]
            row = as.numeric(dataset[rowNumber, 1:clazzColumnIndex-1])

            calculatedDistance = distanceCalculator$calculate(point, row)
            distances = sortDecreasing(shortsDistances$keys())
            if(length(distances) < k) {
              shortsDistances$insert(calculatedDistance, clazz)
            } else {
              for(distanceIndex in 1:length(distances)) {
                shortDistance = distances[distanceIndex]
                if(calculatedDistance < shortDistance) {
                    shortsDistances$insert(calculatedDistance, clazz)
                    shortsDistances$erase(shortDistance)
                    break;
                 }
              }
            }
        }

        findMostFrequent(shortsDistances$values())
      },

      findMostFrequent = function (values) {
        names(sort(summary(as.factor(values)), decreasing = T)[1:1])
      },

      sortDecreasing = function (values) {
        values[order(unlist(values), decreasing = TRUE)]
      }
    )
)
```

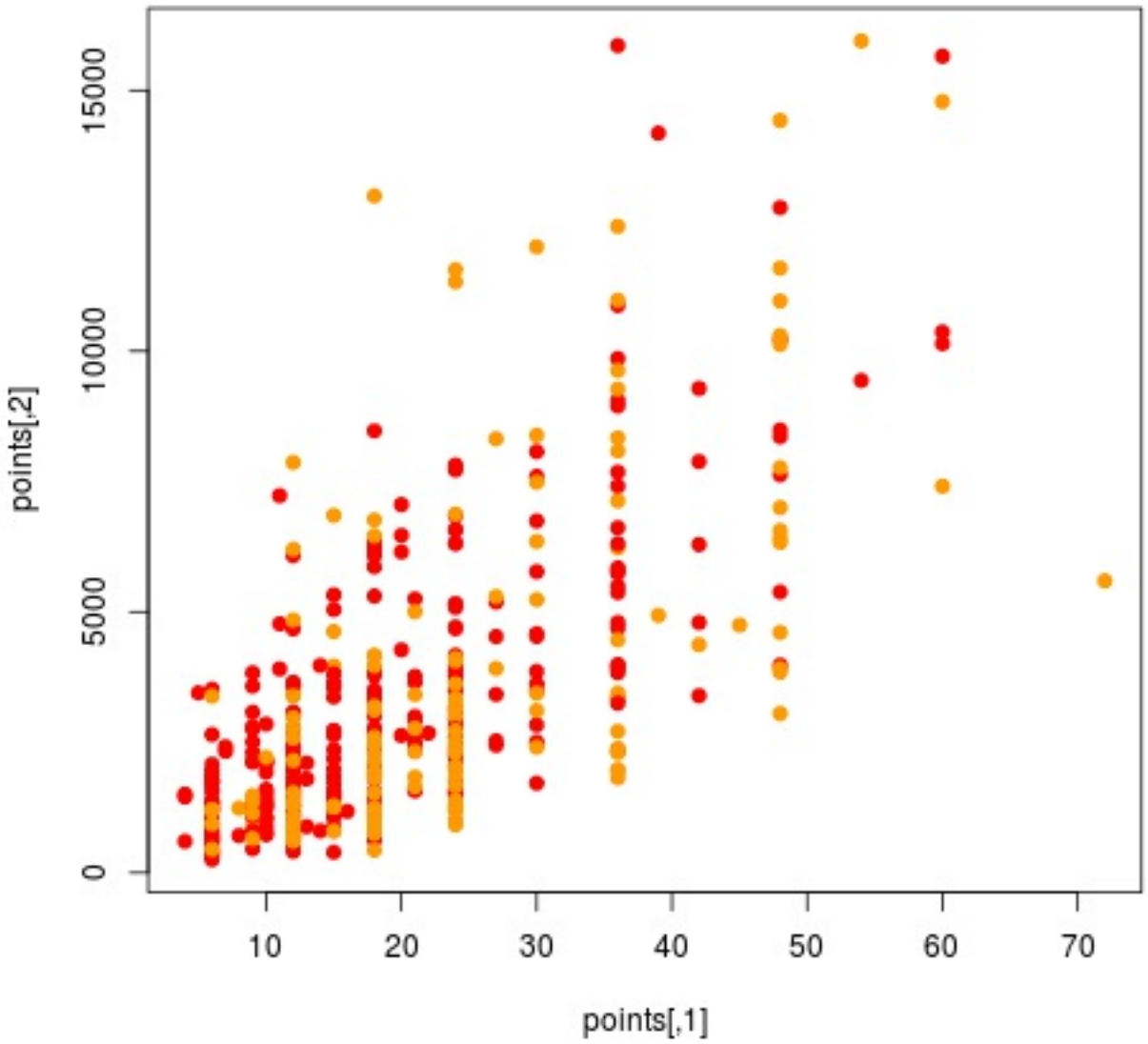
Auxiliares

- Na pasta /**samples** existem alguns exemplos de uso dos datasets
- Em /**plots** há um arquivo que plota os datasets. O resultado pode ser verificado abaixo:

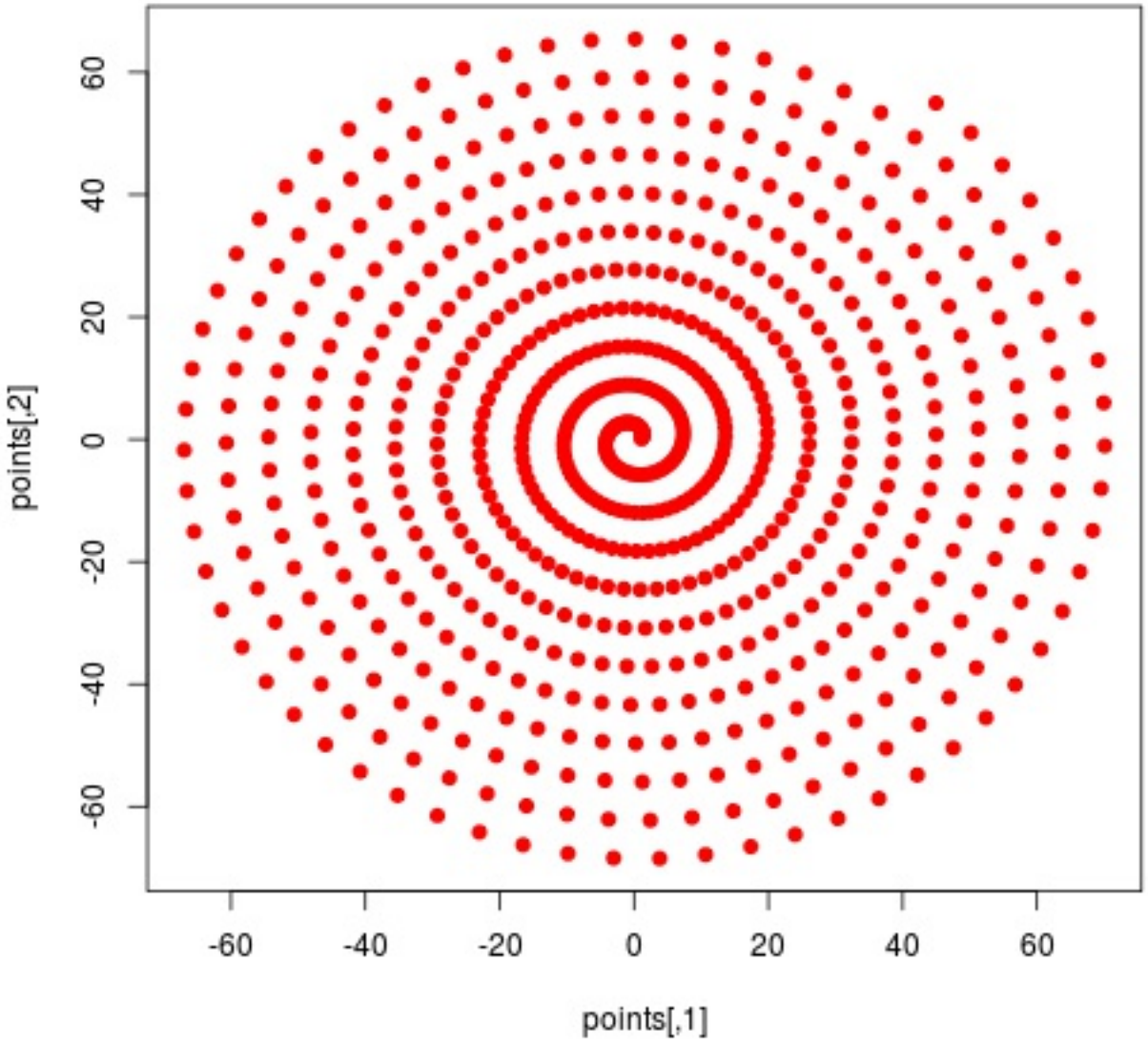
Carros



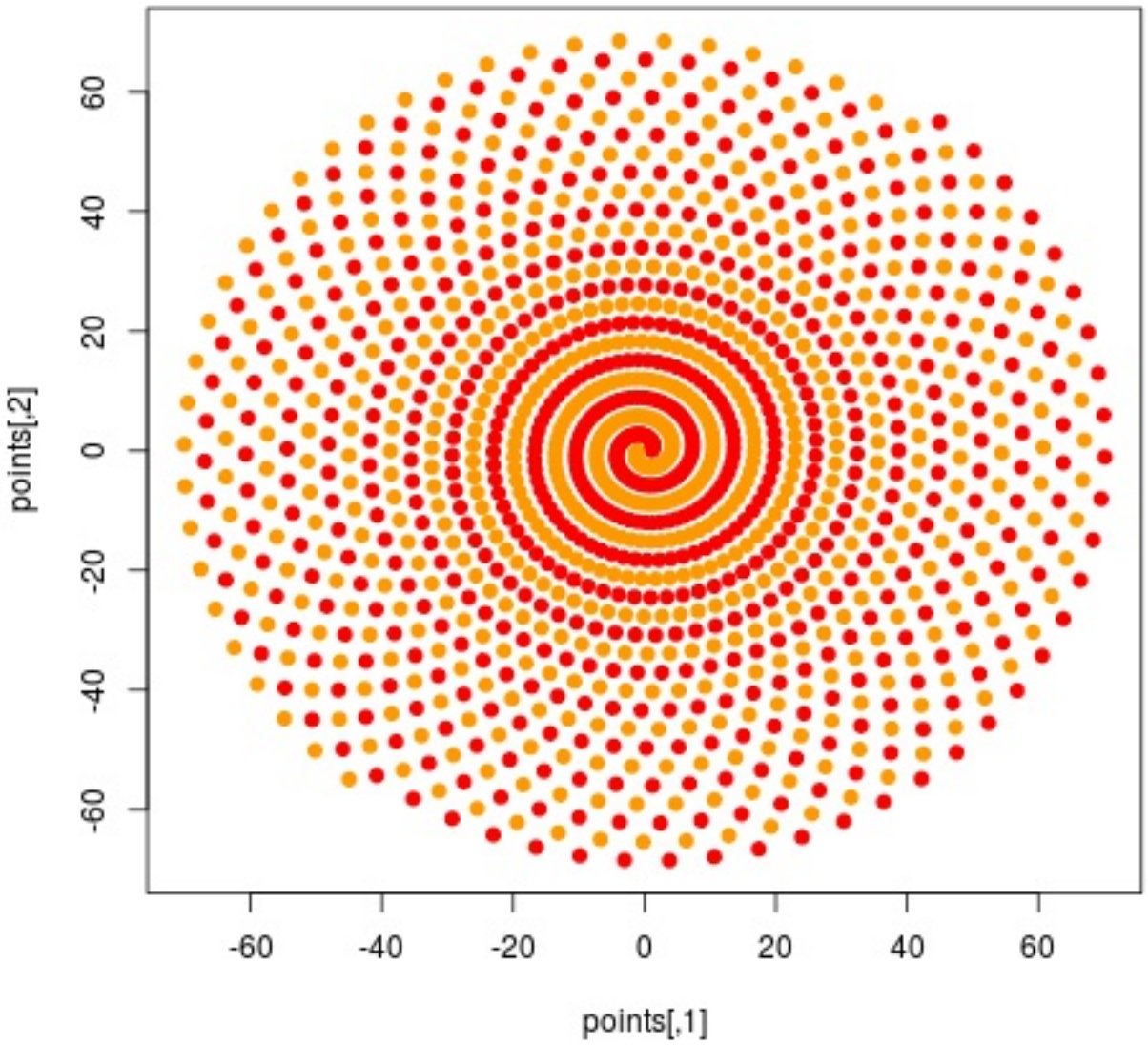
Banco



Espiral Simples



Espiral dupla



Análises

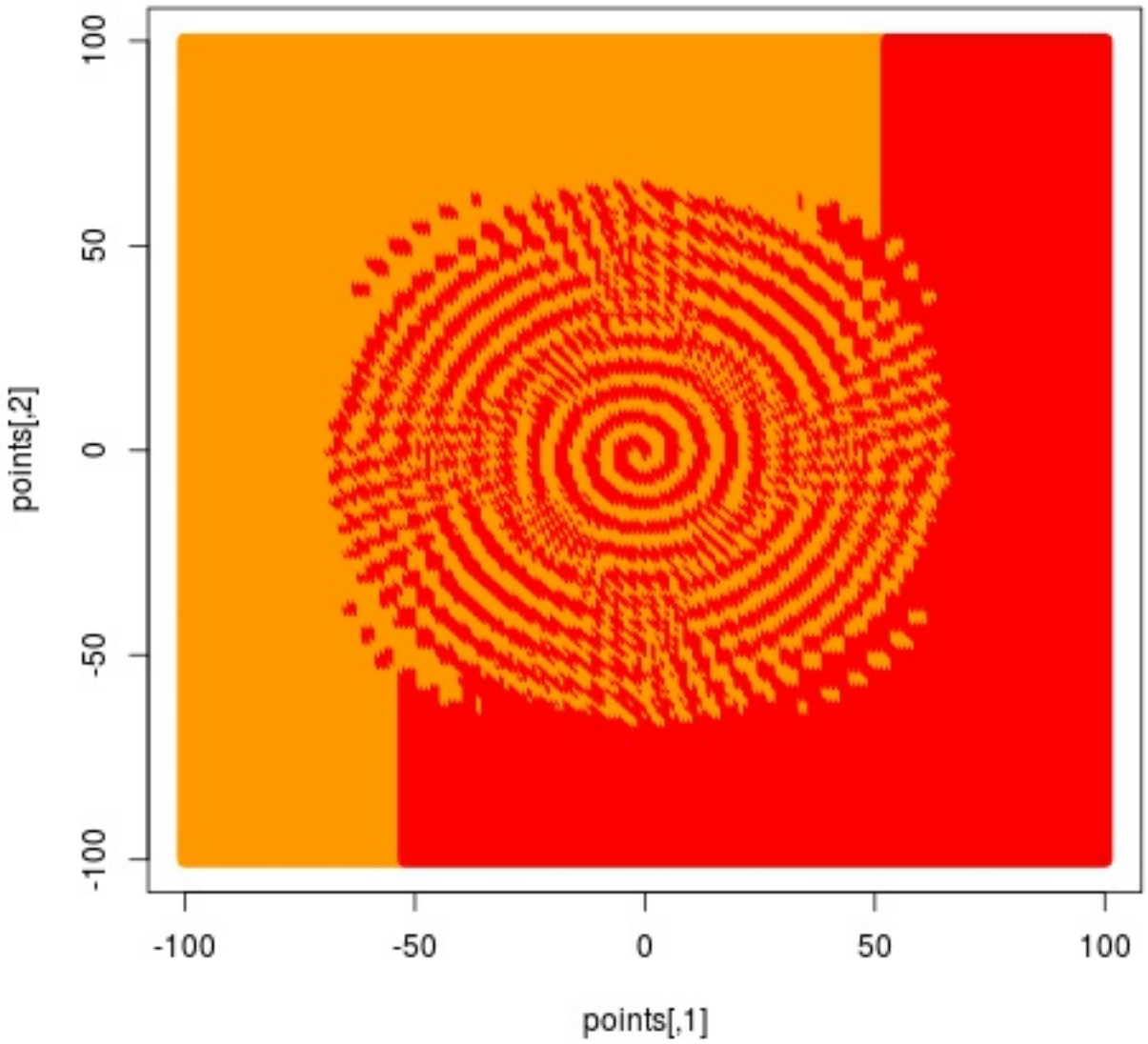
Com a base do trabalho concluído foi possível analisar a distribuição das classes. Para analisar as espirais duplas temos o seguinte código:

dataset_analysis/double_spiral_analysis.R

```
# Load all dependencies
source('dependencies.R')

doubleSpiral = doubleSpiral$new()
doubleSpiralPoints = doubleSpiral$points();

distanceCalculator = hammingDistanceCalculator$new()
kNearestNeighbourClassifier = kNearestNeighbourClassifier$new(distanceCalculator,
doubleSpiralPoints)
```

```
doubleSpiralPointsDimensions = dim(doubleSpiralPoints)
bankRowsSize = doubleSpiralPointsDimensions[1]
classifiedPoints = matrix(0, bankRowsSize, doubleSpiralPointsDimensions[2])

xPoints = seq(-100, 100)
yPoints = seq(-100, 100)

pointsToClassify = matrix(0, length(xPoints) * length(yPoints), 2)
count = 1
for(xPointIndex in 1:length(xPoints)) {
    for(yPointIndex in 1:length(yPoints)) {
      pointsToClassify[count, ] = c(xPoints[xPointIndex], yPoints[yPointIndex])
      count = count + 1
      print(count)
    }
}

pointsToClassifySize = dim(pointsToClassify)[1]
classifiedPoints = matrix(0, pointsToClassifySize, 3)

for (pointsToClassifyIndex in 1:pointsToClassifySize) {
  point = pointsToClassify[pointsToClassifyIndex, ]
  clazz = kNearestNeighbourClassifier$classify(point)
  classifiedPoints[pointsToClassifyIndex, ] = c(point[1], point[2], clazz)
}

plotter = plotter$new()
allPoints = rbind(doubleSpiralPoints, classifiedPoints)
plotter$plotGraph(allPoints, 'plots/double_spiral_analysis.png')
```

O gráfico gerado foi:



Outra análise feita foi utilizando o dataset do banco. Segue código e resultado:

dataset_analysis/bank_analysis.R

```
# Load all dependencies
source('dependencies.R')

bank = bankTwoDimensionsDataset$new('dataset/bank.xlsx')
bankPoints = bank$points();

distanceCalculator = hammingDistanceCalculator$new()
kNearestNeighbourClassifier = kNearestNeighbourClassifier$new(distanceCalculator,
bankPoints)

bankPointsDimensions = dim(bankPoints)
bankRowsSize = bankPointsDimensions[1]
classifiedPoints = matrix(0, bankRowsSize, bankPointsDimensions[2])
```

```
pointsToClassifySize = 2000
randomDurationOfCreditMonth = as.integer(runif(n = pointsToClassifySize, min = 6, max
= 60))
randomCreditAmount = as.integer(runif(n = pointsToClassifySize, min = 300, max =
12000))
pointsToClassify = matrix(c(randomDurationOfCreditMonth, randomCreditAmount),
nrow=pointsToClassifySize)
classifiedPoints = matrix(0, pointsToClassifySize, 3)

for (pointsToClassifyIndex in 1:pointsToClassifySize) {
  point = pointsToClassify[pointsToClassifyIndex, ]
  clazz = kNearestNeighbourClassifier$classify(point)
  classifiedPoints[pointsToClassifyIndex, ] = c(point[1], point[2], clazz)
}

plotter = plotter$new()
allPoints = rbind(bankPoints, classifiedPoints)
plotter$plotGraph(allPoints, 'plots/bank_analysis.png')
```