1º Desafio

Desenvolvedor: Thayse Cristina Araújo Rodrigues

Versão: 2

Informações: código desenvolvido em Python com algoritmo KNN sobre a base de dados waveform. São usados 67% dos dados para treino e 33% para teste. Ao fim da execução ele retorna a acurácia do algoritmo, ou seja, seu percentual de acertos.

O classificador utilizado no teste foi o KNN devido a ser indicado nas informações do respoistório do UCI e por já ter conhecimento do algoritmo para poder desenvolvê-lo em Python, linguagem que eu tive pouco contato até hoje mas que apresenta bom rendimento no trabalho devido a ser uma linguagem de alto nível.

No repositório de dados existem três tipos de aondas sendo classe: 0, 1 e 2.

Aplicado à base de dados waveform.data temos uma média de 80% de acertos, e aplicado à base de dados waveform-+noise.data que é uma base de dados com presença de ruídos foi obtida uma média de 78% de acertos.

Programa retorna a distribuição das classes dentro do grupo de treino, matriz de confusão e acurácia. Desta forma tem-se mais informações sobre os dados que estão sendo trabalhados pelo algoritmo e qual foi o resultado do processo de aprendizagem.

Código Fonte

```
import csv
import random
import math
import operator

def loadDataset(filename = 'waveform.data', split = 0.67, trainingSet=[] , testSet=[], attr = 21):
	with open(filename) as csvfile:
	   lines = csv.reader(csvfile)
	   dataset = list(lines)
	   for x in range(len(dataset)):
	      for y in range(attr):
	         dataset[x][y] = float(dataset[x][y])
	      if random.random() < split:
	         trainingSet.append(dataset[x])
	      else:
	         testSet.append(dataset[x])


def euclideanDistance(instance1, instance2, length):
	distance = 0
	for x in range(length):
		distance += pow((instance1[x] - instance2[x]), 2)
	return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
	distances = []
	length = len(testInstance)-1
	for x in range(len(trainingSet)):
		dist = euclideanDistance(testInstance, trainingSet[x], length)
		distances.append((trainingSet[x], dist))
	distances.sort(key=operator.itemgetter(1))
	neighbors = []
	for x in range(k):
		neighbors.append(distances[x][0])
	return neighbors

def getResponse(neighbors):
	classVotes = {}
	for x in range(len(neighbors)):
		response = neighbors[x][-1]
		if response in classVotes:
			classVotes[response] += 1
		else:
			classVotes[response] = 1
	sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
	return sortedVotes[0][0]

def getAccuracy(testSet, predictions):
	correct = 0
	for x in range(len(testSet)):
```

```
            if testSet[x][-1] == predictions[x]:
                correct += 1
        return (correct/float(len(testSet))) * 100.0

def execKNN(dados = 'waveform.data', split = 0.67, attr = 21):
        trainingSet=[]
        testSet=[]
        loadDataset(dados, split, trainingSet, testSet, attr)
        print('\n\nUsing KNN in the ', dados, ' database')
        print ('Train set: ' + repr(len(trainingSet)))
        print ('Test set: ' + repr(len(testSet)))
        # generate predictions
        predictions=[]
        k = 3
        tr0 = 0
        tr1 = 1
        tr2 = 2
        for x in range(len(trainingSet)):
            if trainingSet[x][-1] == '0':
                tr0 = tr0 + 1
            elif trainingSet[x][-1] == '1':
                tr1 = tr1 + 1
            else:
                tr2 = tr2 + 1
        total = tr0 + tr1 + tr2
        print('\nTraining Dataset Distribution:\n >Class0 ' + repr(round((tr0/total)*100,2))+'%\n >Class1 ' +
repr(round((tr1/total)*100,2))+'%\n >Class2 ' + repr(round((tr2/total)*100,2))+'%\n')

        classe00 = 0 #class 0 and classified as 0
        classe01 = 0
        classe02 = 0
        classe10 = 0
        classe11 = 0
        classe12 = 0
        classe20 = 0
        classe21 = 0
        classe22 = 0
        for x in range(len(testSet)):
            neighbors = getNeighbors(trainingSet, testSet[x], k)
            result = getResponse(neighbors)
            predictions.append(result)
            #print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))

            if result == '0':
                if testSet[x][-1] == '0':
                    classe00 = classe00 + 1
                elif testSet[x][-1] == '1':
                    classe01 = classe01 + 1
                else:
                    classe02 = classe02 + 1
            if result == '1':
                if testSet[x][-1] == '0':
                    classe10 = classe10 + 1
                elif testSet[x][-1] == '1':
                    classe11 = classe11 + 1
```

```
                else:
                    classe12 = classe12 + 1
            if result == '2' :
                if testSet[x][-1] == '0':
                    classe20 = classe20 + 1
                elif testSet[x][-1] == '1':
                    classe21 = classe21 + 1
                else:
                    classe22 = classe22 + 1
        print('Confusion Matrix \n\t  0\t  1\t  2\t\t <= Classified as')
        print('----------------------------------------------------------')
        print('\t', classe00, '\t', classe01, '\t', classe02, '\t|\t0\n')
        print('\t', classe10, '\t', classe11, '\t', classe12, '\t|\t1\n')
        print('\t', classe20, '\t', classe21, '\t', classe22, '\t|\t2\n')
        accuracy = getAccuracy(testSet, predictions)
        accuracy = round(accuracy,2)
        print('Accuracy: ' + repr(accuracy) + '%')

def main():
        # prepare data and use KNN to classify the instances
        split = 0.67
        attr = 21
        dados = 'waveform.data'
        execKNN(dados, split, attr)
        split = 0.67
        attr = 40
        dados = 'waveform-+noise.data'
        execKNN(dados, split, attr)

main()
```