import csv

import random

import math

import operator

import MySQLdb

from project.generate import Feature

from project import data

mydb = MySQLdb.connect(

host="localhost",

user="root",

passwd="root",

db="learningdb"

)

def loadDataset(split, trainingSet=[], testSet=[]):

query=""

if testSet==None:

query="select \* from dataset where problem='switch'"

else:

query="select \* from dataset"

mydb.query(query)

r = mydb.store\_result()

dataset = r.fetch\_row(maxrows=0, how=1)

for x in dataset:

f = Feature()

f.problem = x['problem']

f.student\_id = x['student\_id']

f.knowledgeLevel = x['knowledge\_level']

f.learningObject = x['learning\_object']

f.learningStyle = x['learning\_style']

f.path = x['path']

f.testPerformance = x['test\_performance']

if(split != None):

if random.random() < split:

trainingSet.append(f)

else:

testSet.append(f)

else:

trainingSet.append(f)

def euclideanDistance(instance1, instance2, length):

distance = 0

for x in range(length):

distance += pow((instance1[x] - instance2[x]), 2)

return math.sqrt(distance)

def getScore(x, testInstance, attr):

learningStyleScore = calculate(

x.learningStyle, testInstance.learningStyle, data.learningStyles)

kScore = calculate(

x.knowledgeLevel, testInstance.knowledgeLevel, data.knowledgeLevels)

oScore = calculate(x.learningObject, testInstance.learningObject,

data.learningObjects1)

pathScore = calculate(x.path, testInstance.path, data.path)

testScore = calculate(int(x.testPerformance or 0), int(testInstance.testPerformance or 0),

data.testPerformance)

total = learningStyleScore+kScore+testScore

if(attr == "path"):

total = total+int(oScore or 0)

if(attr == "learningObject"):

total = total+int(pathScore or 0)

return math.sqrt(total)

def calculate(xValue, testValue, dictVals):

if(xValue==None or xValue=='' or testValue==None or testValue==''):

return 0

l = None

if type(dictVals) is dict:

l = list(dictVals.values())

xValue = dictVals[xValue]

testValue = dictVals[testValue]

else:

l = dictVals

v1 = normalize(l, xValue)

v2 = normalize(l, testValue)

return pow((v1 - v2), 2)

def getNeighbors(trainingSet, testInstance, k, attr):

distances = {}

for x in trainingSet:

dist = getScore(x, testInstance, attr)

distances[dist] = x

neighbors = []

for x in sorted(distances):

if(k == 0):

break

neighbors.append(distances[x])

k = k-1

return neighbors

def getResponse(neighbors, attr):

classVotes = {}

for x in range(len(neighbors)):

response = getattr(neighbors[x], attr)

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, attr):

correct = 0

i = 0

for x in testSet:

if getattr(x, attr) == predictions[i]:

correct += 1

i = i+1

return (correct/float(len(testSet))) \* 100.0

def normalize(data, val):

return ((val - min(data)) / (max(data) - min(data)))

def main(attr):

# prepare data

trainingSet = []

testSet = []

split = 0.8

displayResult = {}

loadDataset(split, trainingSet, testSet)

print('Train set: ' + repr(len(trainingSet)))

print('Test set: ' + repr(len(testSet)))

# generate predictions

predictions = []

k = 4

displayList = []

# attr="path"

# attr="learningObject"

for x in range(len(testSet)):

neighbors = getNeighbors(trainingSet, testSet[x], k, attr)

result = getResponse(neighbors, attr)

predictions.append(result)

print('> predicted=' + repr(result) +

', actual=' + repr(getattr(testSet[x], attr)))

displayList.append('<b>predicted</b>=' + repr(result) +

', <b>actual</b>=' + repr(getattr(testSet[x], attr)))

accuracy = getAccuracy(testSet, predictions, attr)

displayResult['TrainSet'] = repr(len(trainingSet))

displayResult['TestSet'] = repr(len(testSet))

displayResult['results'] = displayList

displayResult['accuracy'] = repr(accuracy)

displayResult['TrainData'] = trainingSet

displayResult['TestData'] = testSet

print('Accuracy: ' + repr(accuracy) + '%')

return displayResult

def predict(x, attr):

trainingSet = []

k = 4

displayResult = ''

loadDataset(split=None, trainingSet=trainingSet, testSet=None)

neighbors = getNeighbors(trainingSet, x, k, attr)

result = getResponse(neighbors, attr)

print('> predicted=' + repr(result) +

', actual=' + repr(getattr(x, attr)))

displayResult =result

return displayResult

# main()