Metoda celor mai apropiaţi k vecini - implementare Python

Algoritmul kNN aparține familiei algoritmilor de învățare bazaţi pe instanţe. Exemple de algoritmi bazați pe instanțe sunt acei algoritmi care modelează problema folosind instanțele (rânduri din matricea reprezentând datele) pentru a face mai departe predicţii. Algoritmul kNN este o formă extremă a metodelor bazate pe instanțe deoarece toate observațiile de antrenament sunt reținute ca parte a modelului.

kNN un algoritm competitiv de învățare, deoarece utilizează în mod intern concurența între elementele modelului (instanțe de date) pentru a face o predicţie. Măsura obiectivă a similitudinii dintre instanțele de date determină fiecare instanță de date să concureze pentru a "câștiga" sau să fie cea mai asemănătoare cu o instanță de date dată și astfel pentru a contribui la predicție.

Algoritmul kNN este un algoritm leneș de învăţarea. Învățarea lenește se referă la faptul că algoritmul nu construiește un model până la momentul în care este necesară o predicție. Este leneș deoarece funcționează numai în ultima secundă. Acest lucru are avantajul de a include doar date relevante pentru datele noi. Un dezavantaj este că poate fi costisitor din punct de vedere computational, şi să se repete aceleași căutări sau căutări similare pe seturi de date mai mari de antrenament.

kNN este puternic deoarece nu face presupuneri despre date și poate fi calculat între oricare două instanțe. Ca atare, se numește non-parametric sau neliniar, deoarece nu presupune o formă funcțională.

Modelul pentru kNN este întregul set de date de antrenament. Atunci când este necesară o predicție pentru o instanță nouă, algoritmul kNN va căuta în setul de date de antrenament cele k instanțe cele mai similar cu instanţa nouă. Clasa-eticheta a celor mai asemănătoare k instanțe este returnat ca predicție pentru instanța nouă. În cazul problemelor de regresie, poate fi returnată media atributului prezis. În cazul clasificării, clasa cea majoritară va fi returnată.

Măsura de similaritate depinde de tipul de date. Pentru datele cu valoare reală, se poate utiliza distanța euclidiană. Se pot utiliza și alte tipuri de date, cum ar fi date categorice sau binare, sau distanță Hamming.

Clasificarea florilor de Iris folosind kNN în Python

Setul de date este compus din 150 de observații ale florilor de iris de la trei specii diferite. Există 4 măsurători ale florilor date: lungimea sepalei, lățimea sepalei, lungimea petalei și lățimea petalei, toate în aceeași unitate de centimetri. Atributul prezis este specia care este una dintre următoarele categorii: setosa, versicolor sau virginica.

Este un set de date standard în care specia este cunoscută pentru toate cazurile. Ca atare, putem împărți datele în seturi de date de antrenare și testare și vom folosi rezultatele pentru a evalua implementarea algoritmului nostru.

Rezolvarea acestei probleme constă în următorii pași:

1. Manipularea datelor: deschideți setul de date din CSV și împărțiți în seturi de date pentru antrenare şi test.
2. Similaritate: Calculați distanța dintre două instanțe.
3. Vecinii: localizați cele mai multe instanțe similare.
4. Răspuns: generați un răspuns dintr-un set de instanțe.
5. Acuratețea: Rezumați precizia predicțiilor.
6. Programul principal: Conectaţi funcţiile.

1. **Manipularea datelor: deschideți setul de date din CSV și împărțiți în seturi de date pentru antrenare şi test.**

Primul lucru pe care trebuie să-l facem este să încărcăm fișierul de date. Datele sunt în format CSV fără o linie antet sau alte citate. Putem deschide fișierul cu funcția deschisă și citim liniile de date utilizând funcția de citire csv.

Apoi trebuie să divizăm datele într-un set de date de antrenament care poate fi folosit pentru a învăţa modelul și un set de date de testare pe care îl folosim pentru a evalua exactitatea modelului.

Mai întâi trebuie să transformăm datele încărcate în șiruri în numere cu care putem lucra. Apoi trebuie să divizăm datele aleatoriu în set de antrenare și set de testare. Un raport de 67/33 pentru antrenare / test este un raport standard folosit.

Definim o funcție numită loadDataset care încarcă un fișier CSV cu numele fișierului furnizat ca şi parametru și împarte aleatoriu datele într-un set de antrenare şi un set de testare folosind raportul de divizare furnizat.

```
import csv
import random
def loadDataset(filename, split, trainingSet=[] , testSet=[]):
      with open(filename, 'r') as csvfile:
          lines = csv.reader(csvfile)
          dataset = list(lines)
          for x in range(len(dataset)-1):
              for y in range(4):
                  dataset[x][y] = float(dataset[x][y])
              if random.random() < split:
                  trainingSet.append(dataset[x])
              else:
                  testSet.append(dataset[x])
```

```
trainingSet=[]
testSet=[]
loadDataset('Calea… /iris.data.txt', 0.66, trainingSet, testSet)
print('Train: ' + repr(len(trainingSet)))
print('Test: ' + repr(len(testSet)))
```

2. Similaritate: Calculați distanța dintre două instanțe.

Pentru a face predicții trebuie să calculam asemănarea dintre două instanțe de date. Acest lucru este necesar pentru a putea găsi cele mai asemănătoare instanțe din setul de date de antrenament pentru o anumită instanţă din setul de date de testare și a face o predicție.

Având în vedere că toate cele patru măsurători de flori sunt numerice și au aceleași unități, putem folosi direct măsura de distanță euclidiană. Aceasta este definită ca rădăcina pătrată a sumei diferențelor pătrate dintre cele coloane. În plus, dorim să controlam câmpurile care trebuie incluse în calcularea distanței. Mai precis, dorim doar să includem primele 4 atribute. O abordare este de a limita distanța euclidiană la o lungime fixă, ignorând dimensiunea finală.

Defini, funcția EuclideanDistance după cum urmează:

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

Testam aceasta funcţie:

```
data1 = [2, 2, 2, 'a']
data2 = [4, 4, 4, 'b']
distance = euclideanDistance(data1, data2, 3)
print('Distance: ' + repr(distance))
```

3. Vecinii: localizați cele mai multe instanțe similare.

Având o măsură a similitudinii, o putem folosi pentru a identifica cele mai asemănătoare instanțe cu o anumită instanță nouă.

Acesta este un proces direct de calcul al distanței pentru toate instanțele și selectarea unui subset cu cele mai mici distanțe.

Mai jos este funcția getNeighbors care returnează cei mai mulți vecini asemănători din setul de antrenament pentru o anumită instanță de testare (folosind funcția euclidiană deja definită)

```
import operator
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
```

şi testarea acesteia:

```
trainSet = [[2, 2, 2, 'a'], [4, 4, 4, 'b']]
testInstance = [5, 5, 5]
k = 1
neighbors = getNeighbors(trainSet, testInstance, 1)
print(neighbors)
```

4. **Răspuns: generați un răspuns dintr-un set de instanțe.**

O dată ce am localizat vecinii cei mai apropiați de instanța de test, următorul pas este să facem o predicţie bazată pe acei vecini. Putem face acest lucru astfel încât fiecare vecin să voteze pentru atributul lui de clasă și se ia votul majoritar ca şi predicţie.
Mai jos este o funcție pentru obținerea majorității votate de la un număr de vecini. Clasa este ultimul atribut pentru fiecare vecin.

```
import operator
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]
```

şi testarea:

```
neighbors = [[1,1,1,'a'], [2,2,2,'a'], [3,3,3,'b']]
response = getResponse(neighbors)
print(response)
```

Această abordare returnează un răspuns în cazul unui vot egal, însă aceste cazuri are putea fi tratate într-un mod specific, cum ar fi returnarea nici unui mesaj corespunzător sau selectarea unui răspuns aleatoriu imparțial.

5. **Acuratețea: Rezumați precizia predicțiilor.**

O modalitate de a evalua exactitatea modelului este de a calcula un raport dintre predicțiile corecte dintre toate predicțiile făcute, numit şi acurateţe.
Mai jos este funcția getAccuracy care însumează predicţiile corecte și returnează precizia ca procent din clasificările corecte.

```
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
```

şi testarea acesteia:

```
testSet = [[1,1,1,'a'], [2,2,2,'a'], [3,3,3,'b']]
predictions = ['a', 'a', 'a']
accuracy = getAccuracy(testSet, predictions)
print(accuracy)
```

6. **Programul principal: Conectaţi funcţiile.**

```
def main():
      # prepare data
      trainingSet=[]
      testSet=[]
      split = 0.67
      loadDataset('Calea....;,/iris.data.txt', split, trainingSet, testSet)
      print('Train set: ' + repr(len(trainingSet)))
      print('Test set: ' + repr(len(testSet)))
      # generate predictions
      predictions=[]
      k = 3
      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]))
      accuracy = getAccuracy(testSet, predictions)
      print('Accuracy: ' + repr(accuracy) + '%')
```