Кластеризация категориальных данных

Категориальные данные — это данные с фиксированным набором значений или категорий (пол, цвет, день недели, …) в качестве характеристик. Категориальные данные могут быть упорядоченными (1-й курс, 2-й курс, 3-й курс, бакалавриат, магистратура), в этом случае можно ввести степень схожести или расстояние (ближе или дальше по оси признака).

Рассмотрим неупорядоченный набор категориальных характеристик. В этом случае представление о кластерах как о более плотном распределении точек в пространстве характеристик не имеет смысла.

Для примера возьмём датасет о голосовании представителей двух партий по ряду вопросов:

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

В первой колонке датасет содержит принадлежность к партии – одна из двух категориальных характеристик (демократ или республиканец). В остальных колонках представлены результаты голосования по 16 вопросам - категориальные характеристики из набора (y, n, ?).

Сравним данные о голосовании в 1-й, 2-й и 3-й строках:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | y | n | y | y | Y | n | n | n | y | ? | y | y | y | n | y |
| n | y | n | y | y | Y | n | n | n | n | n | y | y | y | n | ? |
| ? | y | y | ? | y | Y | n | n | n | n | y | n | y | y | n | n |

Данные 1-й и 2-й строк совпадают в 13 случаях, 1-й и 3-й строк – в 9 случаях. Можно сказать, что 1-я и 2-я строки более «похожи» чем 1-я и 3-я строки. Подобную «похожесть» можно использовать в качестве меры близости категориальных данных.

Введём расстояние между объектами с категориальными признаками, равное числу несовпадающих признаков :

здесь s – номер признака,

– символ Кронекера, который принимает значение:

Такое определение расстояния называется метрикой Хэмминга.

***Алгоритм кластеризации K-modes*** является обобщением алгоритма K-means (К-средних): вводятся центры кластеров и объект относится к ближайшему (по расстоянию до центра) кластеру. Для определения расстояния используется метрика Хэмминга, категориальные характеристики центра кластера совпадают с наиболее часто встречающимися характеристиками объектов, входящих в этот кластер.

Центр кластера определим следующим образом: пусть – объект с набором категориальных характеристик. Каждой категориальной характеристике соответствует множество пар возможных значений и частот появления данного значения в кластере - , здесь – индекс характеристики в кластере. Набор таких пар задаёт центр кластера . Расстояние до центра кластера определяется по формуле:

Мы учли тот факт, что сумма частот по кластеру равна 1:

Алгоритм кластеризации представим в виде следующей последовательности шагов (аналогично K-means):

1. Задаём число кластеров и положение центров кластеров
2. Каждую точку входного множества включаем в тот кластер, центр которого оказался ближайшим
3. Вычисляем новые значения центров кластеров, в соответствии с распределением точек, полученным на предыдущем шаге
4. Повторяем шаги 2 и 3 до тех пор, пока распределение точек по кластерам не стабилизируется

Реализация

import warnings

warnings.filterwarnings("ignore")

import numpy as np

import pandas as pd

from sklearn import preprocessing

import random

# Read the data

vote = pd.read\_csv('../../house\_votes\_84\_data.csv')

POINT\_N = vote.shape[0]

FEATURE\_N = vote.shape[1]-1

CLUST\_N = 2

vote = np.array(vote)

targ = vote[:,0]

data = vote[:,1:17]

label\_encoder = preprocessing.LabelEncoder()

targ = label\_encoder.fit\_transform(targ)

onehot\_encoder = preprocessing.OneHotEncoder(sparse = False)

data = onehot\_encoder.fit\_transform(data)

FEATURE\_N = 3\*FEATURE\_N

class POINT:

def \_\_init\_\_(self, x):

self.clust = CLUST\_N

self.D = x

class CLUSTER:

def \_\_init\_\_(self, cl):

self.clust = cl

self.F = np.zeros(FEATURE\_N)

def Dist(self, p):

s = 0.0

for i in range(FEATURE\_N):

s += 1-self.F[i]\*p.D[i]

return s

def eval\_Center(self, P):

# self.X = np.zeros(len(self.X))

self.F = [0 for i in range(FEATURE\_N)]

s = 0

for i in range(POINT\_N):

for j in range(FEATURE\_N):

if P[i].clust == self.clust:

self.F[j] += P[i].D[j]

s += P[i].D[j]

for j in range(FEATURE\_N):

self.F[j] = self.F[j]/s

return

PP = [POINT(data[i,:]) for i in range(POINT\_N)]

for i in range(POINT\_N):

# PP[i].clust = targ[i]

PP[i].clust = random.randint(0,1)

Clust = [CLUSTER(n) for n in range(CLUST\_N)]

Clust[0].eval\_Center(PP)

Clust[1].eval\_Center(PP)

#a = 0.0

#b = 0.0

#for k in range(FEATURE\_N):

# a += Clust[0].F[k]

# b += Clust[1].F[k]

#print(Clust[0].F)

#print(a)

#print(Clust[1].F)

#print(b)

for n in range(10):

res = 0.0

for i in range(len(PP)):

res += (PP[i].clust==targ[i])

print('\n', res/len(PP), 1.0-res/len(PP))

for p in PP:

if Clust[0].Dist(p) < Clust[1].Dist(p):

p.clust = 0

else:

p.clust = 1

Clust[0].eval\_Center(PP)

Clust[1].eval\_Center(PP)