|  |
| --- |
| *# 编写时间：2020-11-22 # 作者：李渊明 # 2020数模校内赛代码* import pandas as pd import numpy as np import matplotlib.pyplot as plt import random  *# plt初始化* def plt\_init():  plt.style.use(**'seaborn-whitegrid'**)  fig = plt.figure()  ax = plt.axes()  *# 从文件中读出数据* def get\_df\_form\_file(path):  *# df = pd.read\_csv('./data\_v3.CSV', error\_bad\_lines=False)* df = pd.read\_csv(path, error\_bad\_lines=False)  return df  *# 预处理1：将变量正向化* def preprocessing1(df):  max\_df = df.max()  min\_df = df.min()  df[**'hour'**] = df[**'hour'**].apply(lambda x: max\_df[3] - x)  df[**'distance'**] = df[**'distance'**].apply(lambda x: max\_df[4] - x)  return df  *# 预处理2：使用Z-Score标准化去除量纲的影响* def preprocessing2(df):  df=df.drop([**'name'**], axis=1)  data = (df-df.mean())/(df.std())  return data  *# 预处理3：抽取出需要聚类的变量，并进行放大处理* def preprocessing3(data):  li=np.array(data.loc[:,(**'realm'**,**'proportion'**)])  for r in range(len(li)):  li[r, 0] \*= 10  li[r, 1] \*= 10  return li  *# 查看未聚类前簇* def see\_original\_cluster(li):  plt\_init()  x\_original = li[:, 0]  y\_original = li[:, 1]  plt.plot(x\_original, y\_original, **'o'**, color=**'red'**)  *# 两点距离* def distance(p1, p2):  return np.sqrt((p1[0] - p2[0]) \*\* 2 + (p1[1] - p2[1]) \*\* 2)  *# 随机获取初始簇心* def get\_randCenter(dataset, k):  K\_center = np.array([])  centerList = random.sample(range(0, 20), k)  *# print('centerList',centerList)* for i in range(0, len(centerList)):  dataset[centerList[i], 2] = i  if i == 0:  K\_center = np.array([dataset[centerList[i], :2]])  else:  K\_center = np.append(K\_center, np.array([dataset[centerList[i], :2]]), axis=0)  return K\_center  *# 人为获取初始簇心* def get\_original\_center(dataset):  K\_center = np.array([])  centerList = [40, 45, 99, 3, 75, 65]  for i in range(0, len(centerList)):  dataset[centerList[i], 2] = i  if i == 0:  K\_center = np.array([dataset[centerList[i], :2]])  else:  K\_center = np.append(K\_center, np.array([dataset[centerList[i], :2]]), axis=0)  return K\_center  *# K-means算法实现* def kMeans(dataset=None, k=0):  centerList = get\_original\_center(dataset)  center\_change = True  distance\_List = np.full((1, k), -1)  count = 0  while center\_change:  count += 1  *# print('第', count, '次训练')* center\_change = False  *# 更新簇* for point in range(0, len(dataset)):  for i in range(0, len(centerList)):  distance\_List[0, i] = distance(dataset[point], centerList[i])  minIndex = np.argmin(distance\_List)  dataset[point][2] = minIndex  *# 重新计算簇心* for i in range(0, k):  xSum = 0  ySum = 0  psum = 0  for point in dataset:  if point[2] == i:  xSum += point[0]  ySum += point[1]  psum += 1  xAve = int(xSum / psum)  yAve = int(ySum / psum)  newCenter = np.array([xAve, yAve])  if (newCenter != centerList[i]).all():  center\_change = True  centerList[i] = newCenter  return dataset  *# 预处理4，给待聚类点表上初始簇号* def preprocessing4(li):  original\_cluster = np.full((len(li), 1), -1)  dataset = np.append(li, original\_cluster, axis=1)  return dataset  *# 查看聚类好后的数据点可视化信息* def view\_clusters(res):  plt\_init()  data0 = []  data1 = []  data2 = []  data3 = []  data4 = []  data5 = []  for i in res:  i = i.tolist()  if i[2] == 0:  data0.append(i)  elif i[2] == 1:  data1.append(i)  elif i[2] == 2:  data2.append(i)  elif i[2] == 3:  data3.append(i)  elif i[2] == 4:  data4.append(i)  else:  data5.append(i)  data0 = np.array(data0)  data2 = np.array(data2)  data1 = np.array(data1)  data3 = np.array(data3)  data4 = np.array(data4)  data5 = np.array(data5)  x1 = data1[:, 0]  y1 = data1[:, 1]  plt.plot(x1, y1, **'o'**, color=**'red'**)  x0 = data0[:, 0]  y0 = data0[:, 1]  plt.plot(x0, y0, **'o'**, color=**'blue'**)  x2 = data2[:, 0]  y2 = data2[:, 1]  plt.plot(x2, y2, **'o'**, color=**'black'**)  x3 = data3[:, 0]  y3 = data3[:, 1]  plt.plot(x3, y3, **'o'**, color=**'green'**)  x4 = data4[:, 0]  y4 = data4[:, 1]  plt.plot(x4, y4, **'o'**, color=**'yellow'**)  x5 = data5[:, 0]  y5 = data5[:, 1]  plt.plot(x5, y5, **'o'**, color=**'purple'**)  *# KNN算法实现* def knn(p0, dataset, k, df):  dis = [[0, 0]] \* k  dis = np.array(dis)  top = 0  p0[0] = ((p0[0] - df[**'realm'**].mean()) / df[**'realm'**].std()) \* 10  p0[1] = ((p0[1] - df[**'proportion'**].mean()) / df[**'proportion'**].std()) \* 10  for p\_idx in range(len(dataset)):   d = distance(dataset[p\_idx], p0)  if top != k:  dis[top][0] = d  dis[top][1] = p\_idx  top += 1  else:  dis = sort(dis)  dis[k - 1][0] = d  dis[k - 1][1] = p\_idx  p0\_clusters = []  for i in dis:  p0\_clusters.append(dataset[i[1], 2])  return p0\_clusters  *# 冒泡排序算法* def sort(dis):  for i in range(len(dis)-1):  for j in range(len(dis)-1-i):  if(dis[j][0]>dis[j+1][0]):  temp = dis[j]  dis[j]=dis[j+1]  dis[j+1]=temp  return dis  *# 获取dis列表中出现次数最多的元素* def get\_most(p0\_clusters):  tmp = {i: p0\_clusters.count(i) for i in set(p0\_clusters)}  *# 找出次数最大的那个* most = max(zip(tmp.values(), tmp.keys()))[1]  return most  *# 利用topsis模型对工作进行评分* def topsis(p0\_clusters,df,w):  df = df.drop([**'name'**], axis=1)  cluster\_idx=get\_most(p0\_clusters)  index = np.argwhere(res[:, 2] == cluster\_idx).tolist()  idx = []  for i in index:  idx.append(i[0])  topsis\_df = df.iloc[idx,:]  *# 重新排列索引* topsis\_df = topsis\_df.reset\_index()  topsis\_idx = [**'salary'**, **'hour'**, **'distance'**]  for c in range(len(topsis\_idx)):  s = 0  ti=topsis\_idx[c]  for r in range(df.shape[0]):  s = s + np.array(df.loc[r:r, (ti)] \*\* 2).tolist()[0]  *# print(s)* topsis\_df[ti] = topsis\_df[ti].apply(lambda x: x / np.sqrt(s))  *# 加权* for i in range(len(topsis\_idx)):  topsis\_df[topsis\_idx[i]] = topsis\_df[topsis\_idx[i]].apply(lambda x: x \* w[i])  *# 获取正负理想解* max\_df = topsis\_df.max()  min\_df = topsis\_df.min()  topsis\_df[**'score'**] = 0  *# 计算分数* for r in range(topsis\_df.shape[0]):  f = 0  d\_p = 0  d\_n = 0  for c in range(len(topsis\_idx)):  d\_p += (topsis\_df.iloc[r, :][topsis\_idx[c]] - max\_df[topsis\_idx[c]]) \*\* 2  d\_n += (topsis\_df.iloc[r, :][topsis\_idx[c]] - min\_df[topsis\_idx[c]]) \*\* 2  d\_n = np.sqrt(d\_n)  d\_p = np.sqrt(d\_p)  *# print(d\_n / (d\_n + d\_p))* topsis\_df.loc[r:r, (**'score'**)] = [d\_n / (d\_n + d\_p)]  i=topsis\_df[**'score'**].idxmax(axis=0)  *# print(topsis\_df)* best\_idx=topsis\_df.loc[i:i,(**'index'**)].tolist()[0]  return best\_idx  def get\_best\_choice(best\_idx,path):  df0=get\_df\_form\_file(path)  best\_choice=np.array(df0.loc[best\_idx:best\_idx,(**'name'**,**'distance'**,**'salary'**,**'hour'**)]).tolist()  return best\_choice  if \_\_name\_\_ == **"\_\_main\_\_"**:  *# 用户输入* usr\_realm=100  usr\_proportion=0.3  usr\_weight=[1/3,1/3,1/3]  *# 构建模型* k=6  df0=get\_df\_form\_file(**'./data\_v3.CSV'**)  df=preprocessing1(df0)  data=preprocessing2(df)  li=preprocessing3(data)  dataset=preprocessing4(li)  res=kMeans(dataset=dataset,k=6)  p0\_clusters=knn([usr\_realm,usr\_proportion], res, k=k+1,df=df)  *# topsis计算分数，获取最佳索引* best\_idx=topsis(p0\_clusters, df, usr\_weight)  best\_choice=get\_best\_choice(best\_idx=best\_idx,path=**'./data\_v3.CSV'**)  print(best\_choice) |