

**聚类算法的具体应用**

**学 院 电子与信息工程学院**

**学 号 1950095**

**姓 名 何正潇**

**任课教师 陈启军、王祝萍**

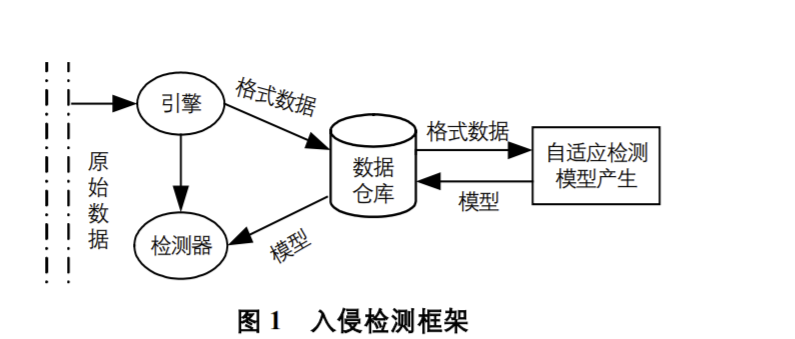
1. 背景

现如今随着大数据时代的到来，对于信息的处理显得尤为重要。作为大数据的学生，而且本次选修了这门人工智能与机器人，我也希望能写出一些围绕本专业的ai应用。但是缘于学识的有限以及笔者在写本篇报告的时候存在一些身体状况的不适，如果论文不令人满意，请老师见谅。而我在这里也没有引用一些机器学习理论方面的论文，更多引用的是机器学习应用方面的文章。其实聚类算法的优化和聚类算法的数学证明同样是极其具有魅力的。我主要举出的应用是在网络入侵中的应用。当然由于我们专业本身并不具备物理目标，因此可能选择的范围比较宽泛。在这里也选择的是我比较感兴趣的应用。

1. 问题描述

实际上来说，这类问题并不新鲜，而且聚类算法也是一个比较成熟的算法，因此我们在这里讲述的应用可能并不是最新的聚类算法应用案例。我采用的论文更多也是来自2010年左右的期刊。问题的主要背景如下，随着网络的迅速发展和广泛使用，人们得益于网络的同时，网上的数据也频繁地受到黑客的攻击和篡改，网络安全 变得越来越重要。目前常用的安全技术如信息加密、防火墙 等可以作为保护网络的第 1 道防线，但仅有上述技术是不够 的，比如目前广泛使用的防火墙技术不能阻止内部攻击，不能提供实时检测等，人们由此提出了网络安全的第 2 道防线 ——入侵检测技术。入侵检测用于识别非授权使用计算机系统的个体(如黑客)和虽有合法授权但滥用其权限的用户(如内部攻击)。现有的入侵检测系统大都采用专家系统或基于统计的方法，这需要较多的经验，而数据挖掘(data mining)方法的 优势在于它能从大量数据中提取人们感兴趣的、事先未知的知识和规律，而不依赖经验[1]。本文运用数据挖掘中的聚类分析方法，建立入侵检测模型数据库。它的优点是能高度自动化地分析原有数据，作出归纳性推理，从中挖掘出潜在的 模式，预测出客户的行为，更重要的是它能够优化或完全抛 弃既有的模型，对入侵行为重新划分并用显示或隐式的方法进行描述。仿真实验表明该方法具有较强的实用性和自适应功能。本文的聚类分析方法是基于距离的 K-平均值(K-means) 方法，利用此技术对于入侵黑客的身份进行分类以及防范。当然这个话题可能我们专业隔壁信息安全的同学很感兴趣，但是我感觉这确实是聚类一个比较实在的应用，因此加以应用。

整体的决策框架



1. 研究进展

入侵检测系统：

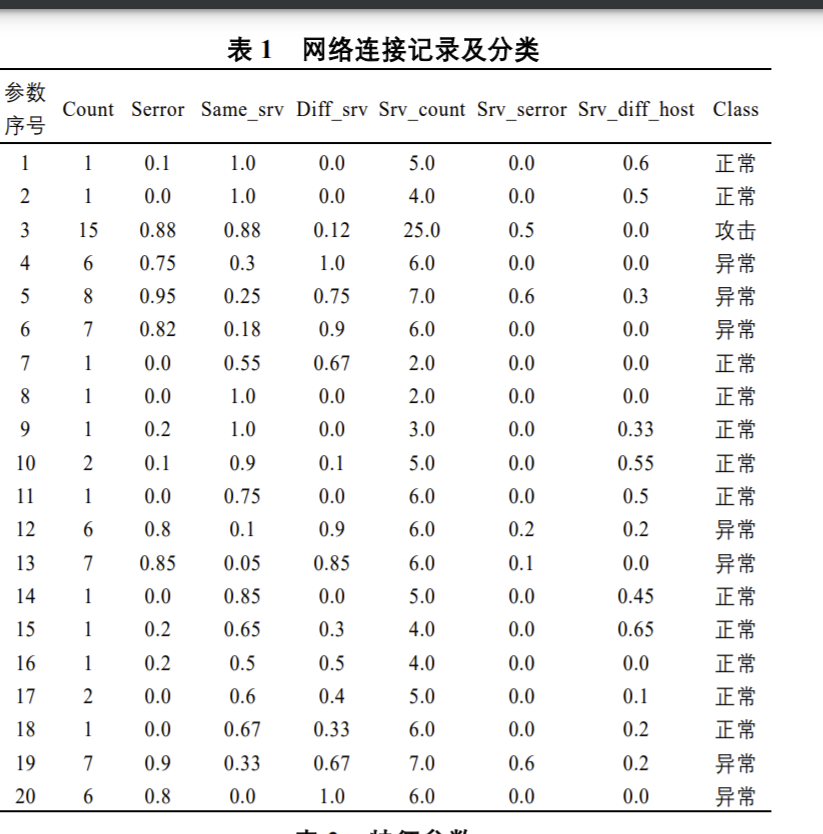
入侵检测是对入侵行为的发觉。入侵检测系统将收集到的信息加以分析，判断网络中是否有违反安全策略的行为和遭到攻击的迹象，若找到入侵痕迹，认为与正常行为相符合的行为是正常行为，与攻击行为相符合的是入侵行为，二者都不符合的，则认为是异常数据，将其加入到数据仓库中作进一步分析。当然系统中存在的数据越多，该系统的判别结果越准确，该模型也是一个自适应模型，会随着输入数据的增多，而产生更加准确的结果。聚类按照“最大化类内相似性，最小化类间相似性”的原则，将数据对象分组为多个类或簇(cluster)，同一个簇中的 对象具有较高相似度，而不同簇间的对象差别较大，对象间的相异度根据对象的属性值进行修改，而整体对象随着元素的增加和减少实时进行更新，以产生可靠的结果。当然聚类的方法以及准测有很多，在这里我们采用二分聚类的方法，因为普通的k-means聚类很依赖初始基准点的选择，同时会陷入局部最优解的问题，导致结果不准确。而二分k-means则能保证比较准确的运算结果，同时不用对初始点的选择过于苛刻。当然底下的k-means算法介绍更多还是基于普通k-means，二分k-means是我自己做的一点小修正。

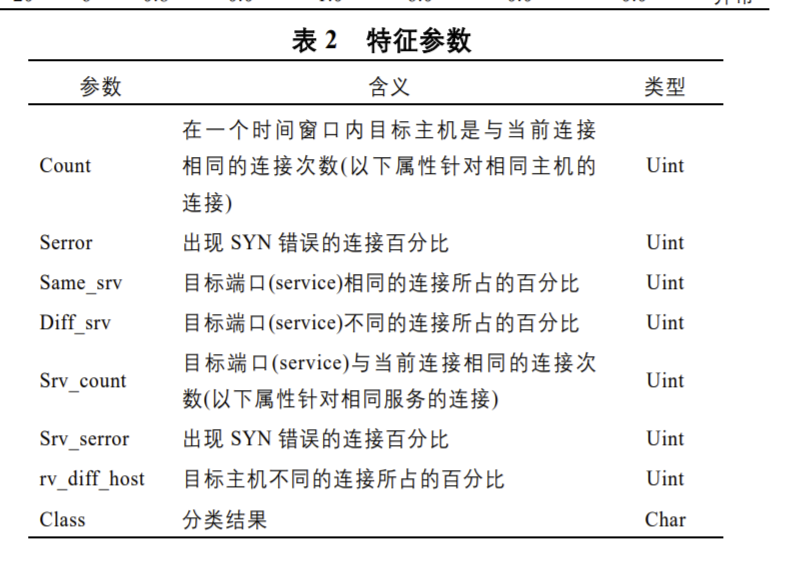
1. means 算法以 K 为参数，把 N 个对象分为 K 个簇，以 使簇内具有较高的相似度，而簇间的相似度较低。相似度的计算根据一个簇中的平均值(视为簇重心)进行。K-means 算法 的处理过程为：(1)随机选择K个对象，每个对象初始代表一 个簇的平均值或中心。对剩余的每个对象，根据其与各个簇中心的距离，将它赋给最近的簇。(2)重新计算每个簇的平均值。这个过程不断重复，直至准则函数收敛到期望值。由于实际应用中对象数据选用的度量单位将直接影响聚类分析结果，不同度量单位可能产生迥异的聚类结构，因此为避免对度量单位选择的依赖，实际中应先对数据进行标准化处理。 标准化的步骤如下：

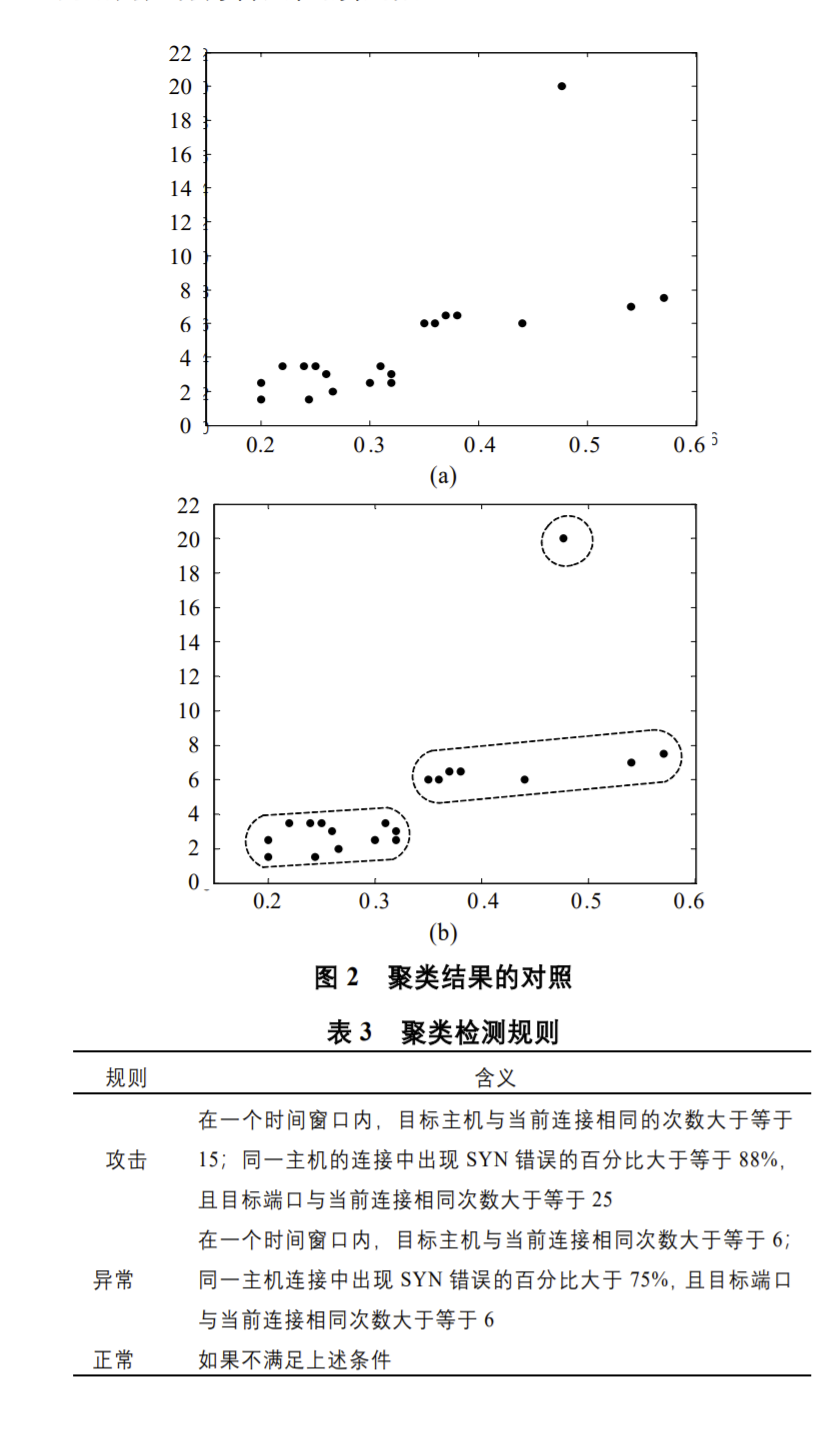
K-means 算法流程如下。算法 K-means 基于簇中对象平均值输入簇的数目 K 和 N 个对象的数据库输出K个簇，满足均方误差函数值最小步骤：(1)任意选择 K 个对象作为初始簇中心(也是初始平均值)； (2)对 K 个对象数据度量值进行标准化处理； (3)Repeat； (4)计算分配后每个簇中对象的平均值(第一次按初始平均值)； (5)根据簇中对象平均值，重新将每个对象赋给最类似的簇； (6)Until 对象的重新分配不再变化。

当然现如今k-means的聚类算法更多的是优化，已经进入了比较成熟的阶段，因此我们更多研究的是对于参数的选择以及学习时间的考量，因此我们关注的更多是数据的选择和收集还有清洗，而不在于聚类算法的研究。

图例中为参数选择的例子和聚类结果







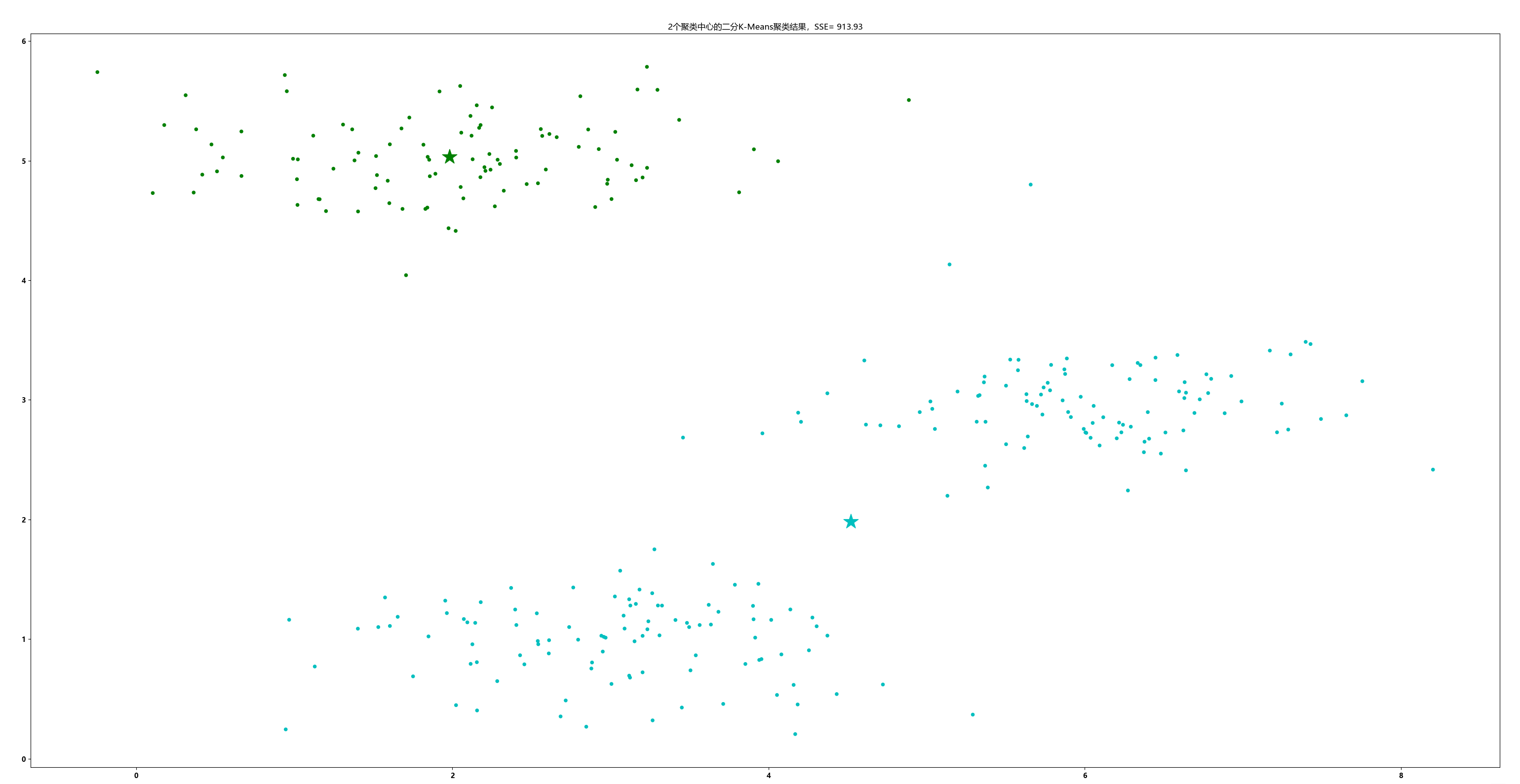
聚类效果如上图

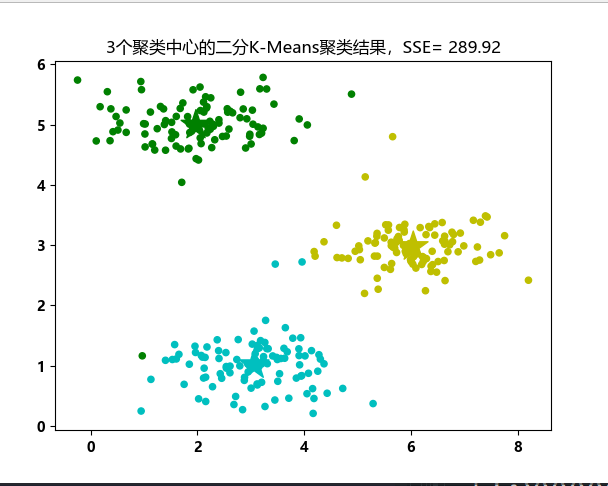
还是能很明确的界定不同种类访问记录的性质的。

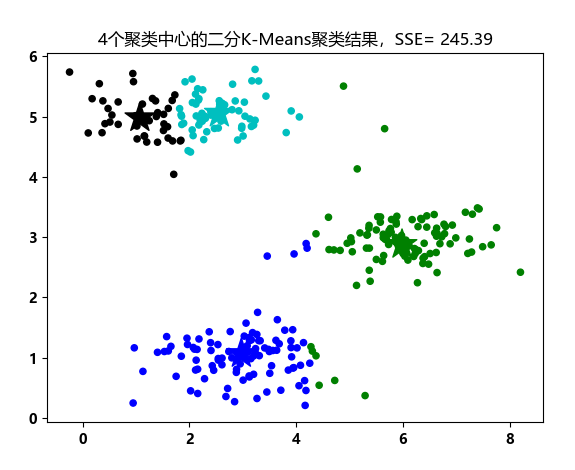
1. 解决方案

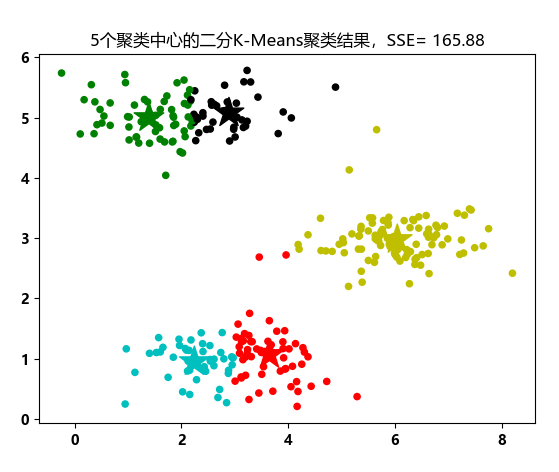
基本在业界这个问题的解决方案已经不是用简单的k-means算法解决了，但是基于我现在所学知识，我写了一个二分k-means的程序进行聚类测试，来完成k-means对于该类问题的算法实现。

效果图如下:









基本的k-means算法如上面所描述，当然由于没有可靠数据集，只能采用随机数据集进行测试，可以看出实现效果基本还比较准确。基本实现的思想就是尽量缩小SSE进行cluster分裂，如果有合适的数据集的话，这个程序能进行简单的网络入侵数据聚类，得到比较可靠的结论。

代码如下:

# -\*- coding: utf-8 -\*-

"""

Created on Sun Dec 5 22:15:16 2021

@author: zheng

"""

#!/usr/bin/env python

# coding: utf-8

# In[2]:

import numpy as np

import matplotlib.pyplot as plt

import matplotlib as mpl

mpl.rc('font',family='MicroSoft YaHei',weight='bold')

import scipy.io

import math

# In[3]:

def loaddata():

data = np.loadtxt('data/cluster\_data.csv',delimiter=',')

return data

# In[4]:

X = loaddata()

plt.scatter(X[:, 0], X[:, 1], s=20)

# In[5]:

def kMeansInitCentroids(X, k):

#从X的数据中随机取k个作为质心

index = np.random.randint(0,len(X),k)

return X[index]

# ### 计算数据点到质心的距离，并判断该数据点属于哪个质心

# In[6]:

def findClosestCentroids(X, centroids):

#idx中数据表明对应X的数据是属于哪一个质心的

idx = np.zeros(len(X)\*2).reshape( X.shape[0],2)

for i in range(len(X)):

minDistance = float('inf');#初始无限大

index = 0

for k in range(len(centroids)):

#距离计算，使用欧几里得距离

distance = np.sum(np.power(X[i]-centroids[k],2))

if(distance<minDistance):

minDistance = distance

index = k

idx[i][0]=index

idx[i][1]=distance

return idx

def computeCentroids(X, idx):

k = set(np.ravel( idx[:, 0]).tolist()) #找到所有聚类中心索引

k = list(k)

centroids = np.ndarray((len(k),X.shape[1]))

for i in range(len(k)):

#选择数据X中类别为k[i]的数据

# print([np.where(idx==k[i])])

data = X[np.where( idx[:, 0]==k[i])[0],]

#重新计算聚类中心

centroids[i] = (np.sum(data,axis=0))/len(data)

return centroids

def k\_means(X, k, max\_iters):

#初始化聚类中心

initial\_centroids = kMeansInitCentroids(X,k)

#迭代

for i in range(max\_iters):

if(i==0):

centroids = initial\_centroids

idx = findClosestCentroids(X,centroids)

cm\_dark = mpl.colors.ListedColormap(['g', 'r', 'b','y','k','c'])

plt.scatter(X[:, 0], X[:, 1],c=np.ravel(idx[: ,0]), cmap=cm\_dark,s=20)

#计算样本到质心的距离，并返回每个样本所属的质心

idx = findClosestCentroids(X,centroids)

#重新计算聚类中心

centroids2=centroids

centroids = computeCentroids(X,idx)

distance2 = sum(idx[:,1])

plt.clf()

plt.title("{0:d}个聚类中心的常规K-Means聚类结果，SSE= {1:.2f}".format(k, distance2))

plt.scatter(X[:, 0], X[:, 1],c=np.ravel(idx[: ,0]), cmap=cm\_dark,s=20)

plt.scatter(centroids[:, 0], centroids[:, 1], c=np.arange(len(centroids)), cmap=cm\_dark, marker='\*',s=500)

outname = "Kmeans\_result" +(str) (i) + ".png"

plt.savefig(outname)

for j in range(len(centroids)):

if(math.fabs(np.sum(np.power(centroids2[j]-centroids[j], 2))) >= 0.05):

break;

if(j==len(centroids)-1):

break

plt.show()

return idx,centroids

def kmeans(X, k, max\_iters):

initial\_centroids = kMeansInitCentroids(X,k)

#迭代

for i in range(max\_iters):

if(i==0):

centroids = initial\_centroids

idx = findClosestCentroids(X,centroids)

#计算样本到质心的距离，并返回每个样本所属的质心

idx = findClosestCentroids(X,centroids)

#重新计算聚类中心

centroids2=centroids

centroids = computeCentroids(X,idx)

for j in range(len(centroids)):

if(math.fabs(np.sum(np.power(centroids2[j]-centroids[j], 2))) >= 0.05):

break;

if(j==len(centroids)-1):

break

plt.show()

return idx,centroids

def biKmeans(X, k):

m = np.shape(X)[0]#确定数据集中数据点的总数

#创建一个矩阵来存放每个点的簇分配结果，包含两列：一列是记录簇索引值，第二列是存储误差。

#误差是指当前点到簇质心的距离，后面将使用该误差来评价聚类的效果。

clusterAssment = np.zeros((m,2))

centroids0 = np.mean(X, axis=0).reshape(-1,2)#计算整个数据集的质心，即初始时的质心的坐标为所有数据点的均值

centList =centroids0 #创建一个初始化只要一个初始质心的列表

# #计算所有数据点到初始质心的距离平方误差

for j in range(m):

clusterAssment[j,1] = np.sum(np.power(centroids0-X[j],2))

times=0

# #该while循环不断地对簇进行划分，直到得到设定的簇数目为止

while (len(centList) < k):

lowestSSE = math.inf

for i in range(len(centList)):#对每一个质心

ptsInCurrCluster = X[np.where(clusterAssment[:,0]== i)[0],:]

#重新计算聚类中心

splitClustAss, centroidMat = kmeans(ptsInCurrCluster, 2, 5)#通过KMeans()函数，得到生成两个质心的簇，即二分，获取到质心及其每个簇的误差值

#将二分kMeans结果中的平方和的距离进行求和

sseSplit = sum(splitClustAss[:,1])#compare the SSE to the currrent minimum

#将未参与二分kMeans分配结果中的平方和的距离进行求和

print(clusterAssment[:,0]==i)

sseNotSplit = sum(clusterAssment[np.where(clusterAssment[:,0]!=i)[0],1])

print("sseSplit, and notSplit: ",sseSplit,sseNotSplit)

#？？？总的（未拆分和已拆分）误差和越小，越相似，效果越优化，划分的结果越好

if (sseSplit + sseNotSplit) < lowestSSE:

bestCentToSplit = i

bestNewCents = centroidMat

bestClustAss = splitClustAss.copy()

lowestSSE = sseSplit + sseNotSplit

# bestClustAss[np.where(bestClustAss[:,0]== 1)[0],0] = len(centList) #调用二分kMeans的结果，默认簇是0,1

bestClustAss[np.where(bestClustAss[:,0]== 1)[0],0] = len(centList)

bestClustAss[np.where(bestClustAss[:,0] == 0)[0],0] = bestCentToSplit #更新为最佳质心

print('最好的质心列表是: ',bestCentToSplit)

print('最好的簇分配结果的长度是the len of bestClustAss is: ', len(bestClustAss))

print('lowestSSE',lowestSSE)

#更新质心列表

centList=np.insert(centList,bestCentToSplit,bestNewCents[0],axis = 0)

centList=np.delete(centList,bestCentToSplit+1,axis=0)

print(bestNewCents[0])

print(bestNewCents[1])

centList=np.append(centList,[bestNewCents[1]],axis=0)

clusterAssment[np.where(clusterAssment[:,0]== bestCentToSplit)[0],:]= bestClustAss

cm\_dark = mpl.colors.ListedColormap(['g', 'r', 'b','y','k','c'])

plt.clf()

plt.scatter(X[:, 0], X[:, 1], c=np.ravel(clusterAssment[: ,0]), cmap=cm\_dark, s=20)

plt.scatter(centList[:, 0], centList[:, 1], c=np.arange(len(centList)), cmap=cm\_dark, marker='\*',s=500)

plt.title("{0:d}个聚类中心的二分K-Means聚类结果，SSE= {1:.2f}".format(len(centList), lowestSSE))

plt.show()

outname = "Kmeans\_result" +(str) (times) + ".png"

plt.savefig(outname)

times=times+1

return centList, clusterAssment

# idx,centroids = k\_means(X, 6, 5)

# print(idx)

# print(centroids)

# cm\_dark = mpl.colors.ListedColormap(['g', 'r', 'b','y','k','c'])

# plt.scatter(X[:, 0], X[:, 1], c=np.ravel(idx[: ,0]), cmap=cm\_dark, s=20)

# plt.scatter(centroids[:, 0], centroids[:, 1], c=np.arange(len(centroids)), cmap=cm\_dark, marker='\*', s=500)

# plt.show()

centroids,idx =biKmeans(X,5)

print(idx)

print(centroids)

cm\_dark = mpl.colors.ListedColormap(['g', 'r', 'b','y','k','c'])

5.参考文献

1.[1]李洋.K-means聚类算法在入侵检测中的应用[J].计算机工程,2007(14):154-156.

2. 胡建斌. 入侵检测技术[DB/OL]. 北京大学网络与信息安全研究 室, 2004.

3 Han J, Kamber M. 数据挖掘概念与技术[M]. 范 明, 孟小峰, 译. 北京: 机械工业出版社, 2001.

4 宁玉杰, 郭晓淳. 基于数据挖掘技术的网络入侵检测系统[J]. 计 算机测量与控制, 2002, 10(3).

5 戴英侠, 连一峰, 王 航, 等. 系统安全与入侵检测[M]. 北京: 清 华大学出版社, 2002.

6 罗守山. 入侵检测[M]. 北京: 邮电大学出版社, 2004.

1. [1]高继森,赵杰,张忠辅,张晶.增强型K-means聚类算法在入侵检测中的应用[J].微计算机信息,2008(09):72-73.
2. [1]孙吉贵,刘杰,赵连宇.聚类算法研究[J].软件学报,2008(01):48-61.
3. [1]邹洪,杨逸岳,张佳发.基于K-means聚类算法的网络入侵监测系统设计[J].自动仪表,2020(09):123-126.DOI:10.14016/j.cnki.1001-9227.2020.09.123.

10.机器学习，周志华