**基于PyTorch实现文本聚类算法DBSCAN附带案例**

1. **前言**

最近在更新Pytorch相关文章的时候，公众号后台小伙伴问起来风控中的聚类算法的应用问题。那么我们今天就先从文本的聚类这场景开始，并使用Pytorch来实现其中的聚类算法。各位，走起！

1. **什么是聚类**

百度百科中对聚类（Clustering）的释义为：将物理或抽象对象的集合分成由类似的对象组成的多个类的过程被称为聚类。

在机器学习领域中，我们对聚类（Clustering）的理解可以是这样：按照某个特定标准（比如说可以基于距离）把一个[数据集](https://so.csdn.net/so/search?q=%E6%95%B0%E6%8D%AE%E9%9B%86&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_44177594/article/details/_blank)分割成不同的类或簇，使得同一个簇内的数据对象的相似性尽可能大，不在同一个簇中的数据对象的相似性尽可能地小，总而言之就是“同簇相近，异簇相离”

**聚类和分类我这里也稍微说明一下他们的不同：**

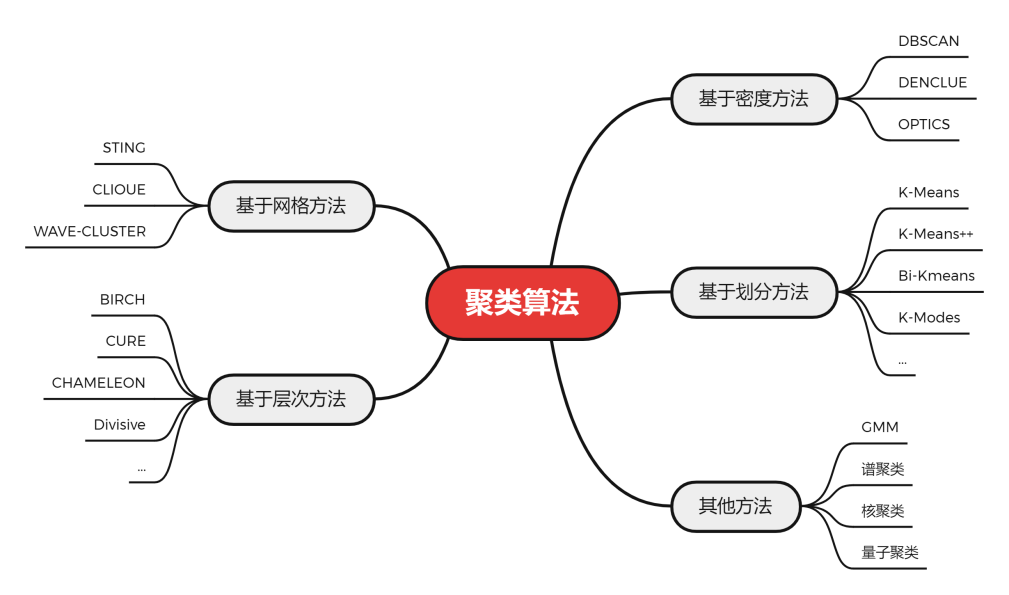
聚类：将相似的数据划分到一起，划分数据的时候不关心数据的标签，属于无监督学习范畴。

分类：将不同的数据分开，在分开数据的时候通过学习一些带有标签的数据得到一个分类器，使用分类器来划分数据，属于监督学习范畴。

那么文本聚类也就比较好理解了，就是划分的数据集是文本数据，比如说两个文本样本数据：今天天气真好，天气真nice。在一定程度上它们是相似的

1. **常见的聚类算法**

在机器学习等领域，聚类算一个老生常谈的话题了。在前人的工作上，后来者不断地研究和创新，衍生出很多聚类算法，接下来我们简单的罗列下：



常见的聚类算法还是很多的，不会一个个去介绍，我们今天的主角为：DBSCAN算法。

1. **DBSCAN算法**

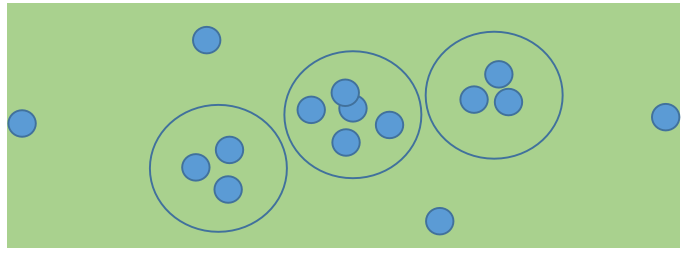
DBSCAN即Density-Based Spatial Clustering of Applications with Noise 。翻译过来的意思大概就是：一种基于密度的适用于噪声数据的空间聚类算法。

这里给出这个算法的论文，大家可以下载下来研究一下：

Ester, M., H. P. Kriegel, J. Sander, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, pp. 226-231. 1996

Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (2017). DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. ACM Transactions on Database Systems (TODS), 42(3), 19.

DBSCAN属于基于密度的聚类算法，即将足够密度的区域划分为一个簇，将高密度区域从低密度区域中分开。那么这个密度怎么理解呢？举一个不是很恰当的例子来理解一下，假设在足球比赛进行中，两方运动员在某一个时间点的部分人员分布如下所示：



我们使用一个圈将运动员密度高的区域给圈起来，也即将紧密相连的数据划为一类，可以看出密度大于3的区域聚集为一簇，这就是一种聚类密度的理解。

至此也引出几个需要解决的问题：

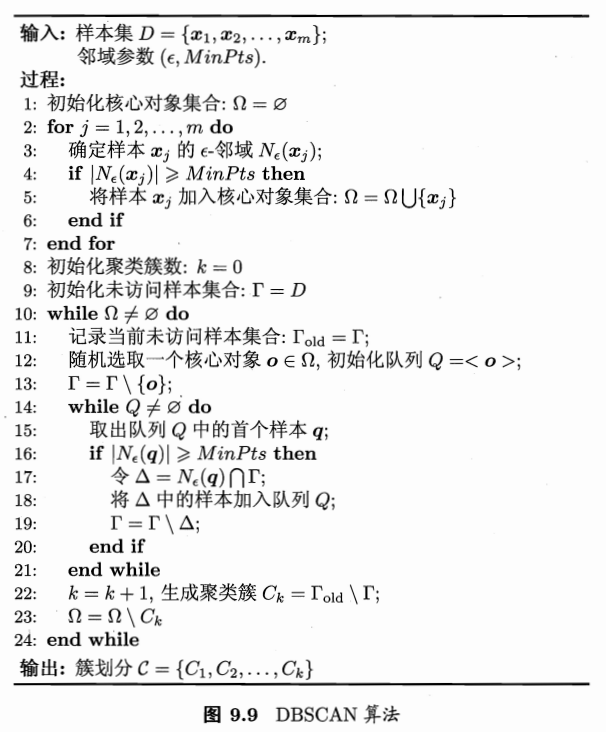
（1）在哪里开始画圈

（2）圈画多大才合适

（3）画几个圈

以上是我们关心的问题，大家看完文章之后应该就有答案了。

接下来我们就要具体的看这个算法的真实面目和工作原理了，这里先给出周志华老师《机器学习》一书中对于算法的描述：

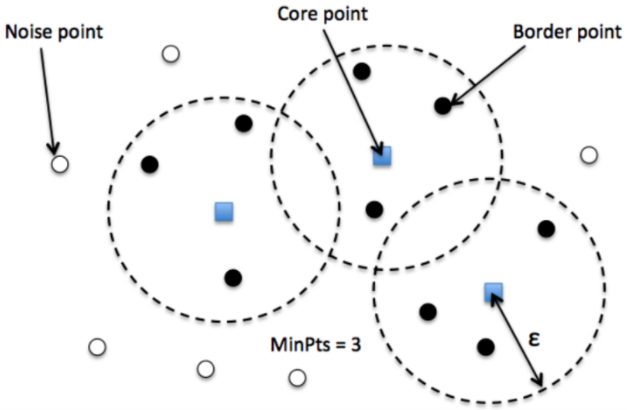


可以看出这个算法的输入是一些不包含label的数据和2个领域参数，这2个参数是整个算法的核心参数，也是在拟合数据的时候需要去寻优的参数，比较难确定一个合适的值。其中：

ϵ: 邻域参数，表示邻域的半径大小。

MinPts: 密度参数，表示领域内最小点的个数。

基于这两个参数，我们可以将点划分为一下几种类型：



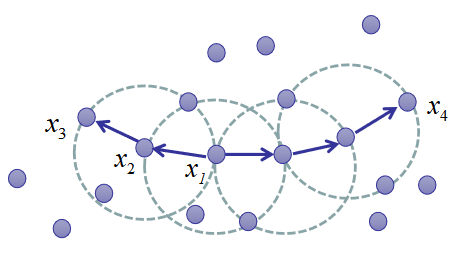
（1）核心点（Core Point）：指在邻域ε内其周围区域中至少有minPts个点的点。

（2）边界点（Border Point）：指其周围区域内的点数小于minPts，且该对象落在核心点的领域内。

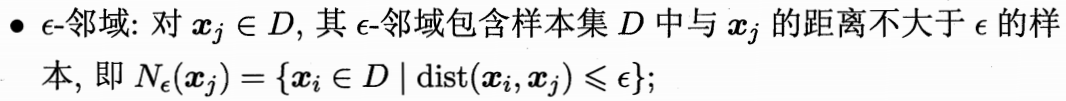
（3）离群点（Noise Point）：指不是核心点和边界点之外的点。

上图中核心点为蓝色点，且各个核心点邻域样本的个数均为4（包含自身）

通过这两个参数可以很好的刻画样本分布的密度情况，先给出包含4个样本的概念图，我们再来解释一些其他的术语：



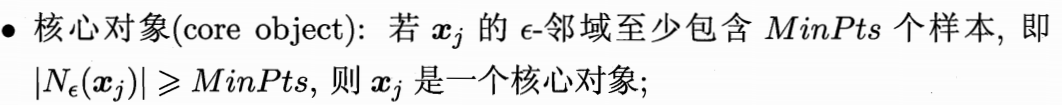
假设上图圆的半径为1，我们取ϵ =1,MinPts=3则有下述定义：



对样本来说，其ϵ邻域有5个样本，包含样本；

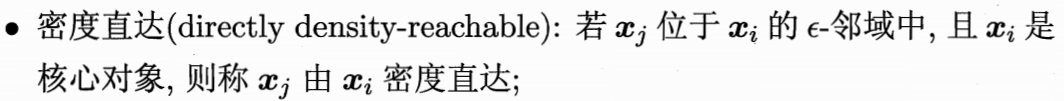
对样本来说，其ϵ邻域有4个样本，包含样本；

虚线所围成的区域均属于ϵ邻域只是属于不同的样本。

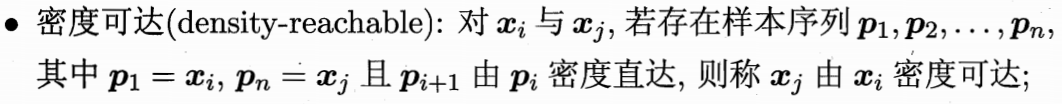


**这里所说的核心对象和上述的核心点事一致的，在判断核心对象计算包含MinPts点的时候要包含点 本身，这一点得非常注意。**

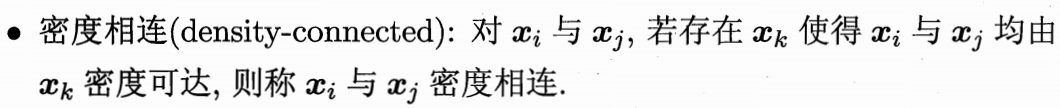
根据描述，在中，和属于核心对象的范畴。



这句话很好理解，翻译过来就是我是核心对象，且我在邻域中包含你，那么你由我密度直达。根据描述，在中，由密度直达。



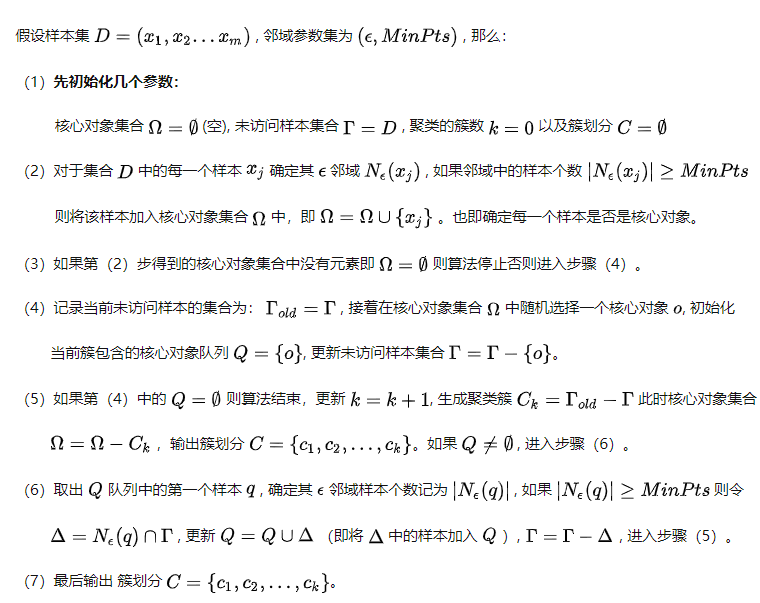
细品有一种传递性的感觉，就不解释了。根据描述，在中，由密度可达，由密度可达。



根据描述，在中，与密度可达，此时。

那么我们可以推导出一个结论：选定一个核心点从核心点出发，不断向密度可达的ϵ-邻域进行扩增，将会得到一个包含核心点的边界最大化区域，区域中任意两点密度相连。

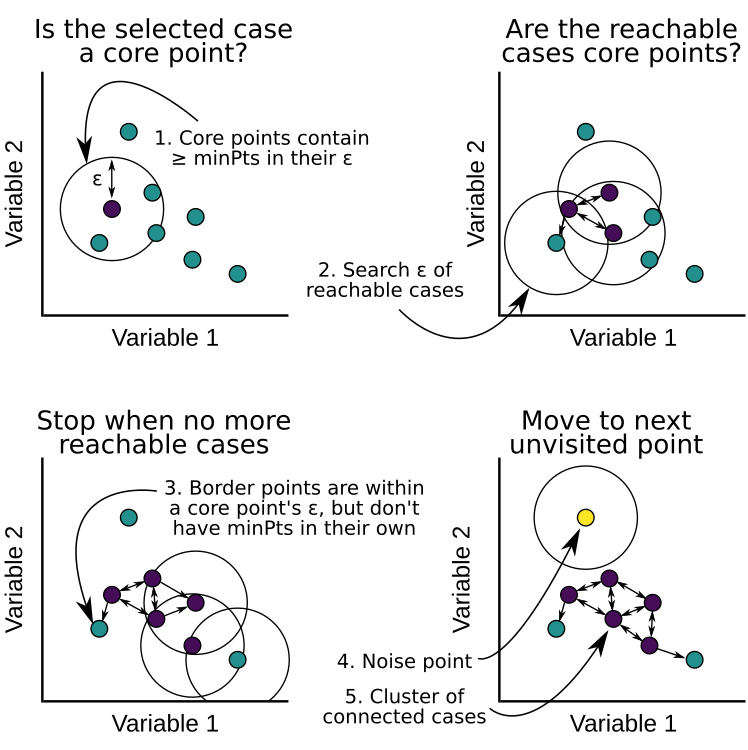
上述概念理解起来还是很简单的，在实际情况中样本与样本之间还是容易让人混乱的。那么这个算法是怎么工作的呢，我们来整理一下过程：



看起来很复杂，其实我们只要记住两大步骤就可以：

**第一步，寻找核心点形成临时的聚类簇。**

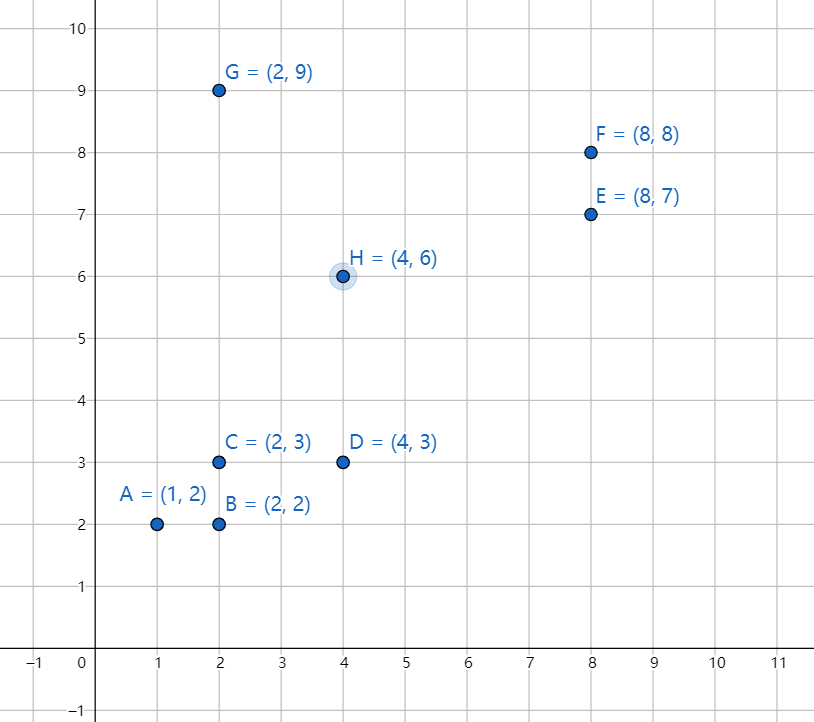
**第二步，合并临时聚类簇得到聚类簇。**



我们来看一个具体的案例，先创建一些数据点：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Point | A | B | C | D | E | F | G | H |
| Var1 | 1 | 2 | 2 | 4 | 8 | 8 | 2 | 4 |
| Var2 | 2 | 2 | 3 | 3 | 7 | 8 | 9 | 6 |

数据集中的样本数据在二维空间的标注如下所示：



### 假设距离公式使用欧式距离进行计算[(Euclidean Distance)](https://www.baidu.com/link?url=cceIK2ox9DGNkln9OgE5LzYhzWeIpWTTWmALzVcYDghy5k12LaLnHrCaDSblHSQpzp_AAfUmWWrVTCsH4Anz9Gl4p6lkIwg_julz0pqsVzC&wd=&eqid=fefcf0cf0007e4b2000000066292ce19" \t "https://www.baidu.com/_blank)，ϵ=2,MinPts=2。

我们较为简单的说一下实际的过程：

**第一步，寻找核心点形成临时的聚类簇。**

1. 扫描全部样本点，寻找核心点，放入核心点集合，显然核心点集合CP = {A,B,C,D,E,F}。
2. 对于核心点集合中的点，并将其**密度直达**的点形成对应的临时聚类簇。

A核心点的临时聚类簇为{A,B,C};

B核心点的临时聚类簇为{A,B,C};

C核心点的临时聚类簇为{A,B,C,D};

D核心点的临时聚类簇为{C,D};

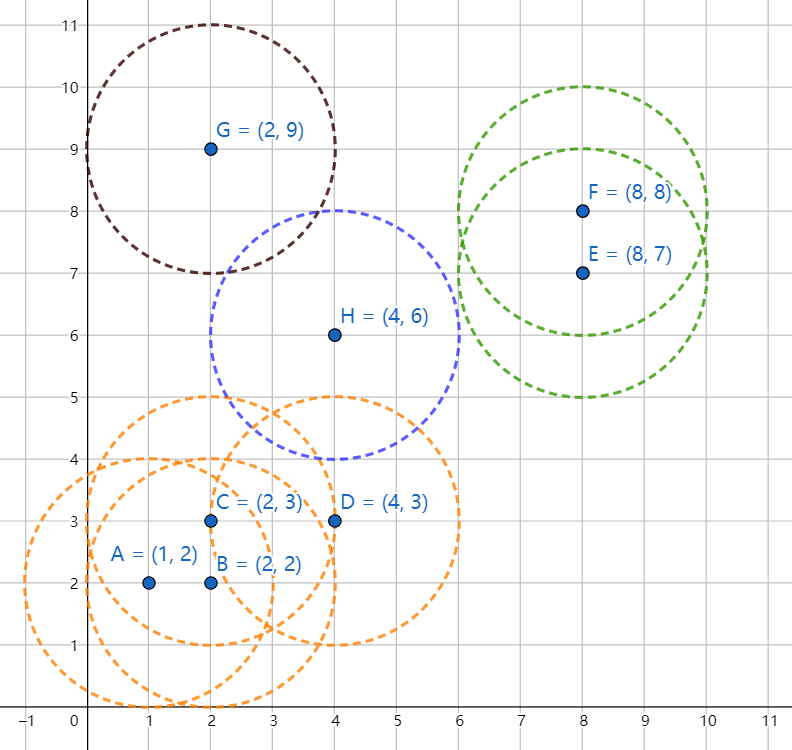
E核心点的临时聚类簇为{E,F};

F核心点的临时聚类簇为{E,F};

**第二步，合并临时聚类簇得到聚类簇。**

1. 以任意一个核心点出发，找出其**密度可达**的样本生成聚类簇，直到所有核心对象均被访问过为止。
2. 从核心点集合中的一个核心点A出发，密度直达的样本点为B和C,密度可达的样本点为D,生成一个聚类簇，簇内点为C1= {A,B,C,D}。
3. 访问核心点B,C,D由于它们已经在簇C1中，选择下一个核心点。
4. 访问核心点E,密度直达的样本点为F，生成一个聚类簇，簇内点C2 ={E,F}。
5. 核心对象F已经在簇C2中已经访问，核心点集合中不存在没有访问的核心点，算法结束。
6. 最终生成两个簇C1={A,B,C,D}和C2={E,F}

其实我们在循环判断的时候，如果簇间内的对象密度可达，那么即可合并生成聚类簇。



1. **Sklearn中的DBSCAN算法**

在Python的sklearn模块中已经封装好了DBSCAN算法，我们借助sklearn来实现上述案例，代码如下：

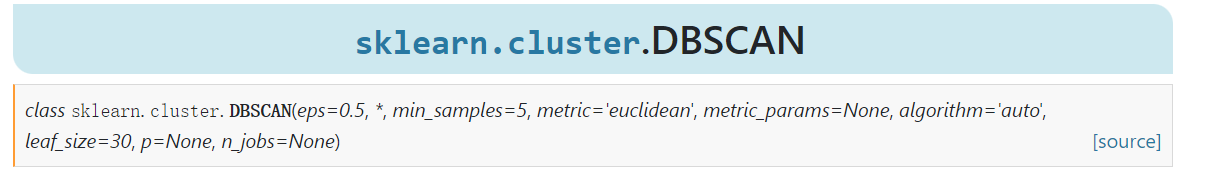
import numpy as np  
from sklearn.cluster import DBSCAN  
X = np.array([  
 [1, 2],  
 [2, 2],  
 [2, 3],  
 [4, 3],  
 [8, 7],  
 [8, 8],  
 [2, 9],  
 [4, 6]])  
clustering = DBSCAN(eps=2, min\_samples=2).fit(X)  
print(clustering.labels\_)

# 输出

[ 0 0 0 0 1 1 -1 -1]

可知前四个点聚为一簇，后两个类聚为一簇，最后两个-1的label表示噪点。

Skearn的DBSCAN类定义如下：



当然非常重要的参数还是那两个：

eps: ϵ-邻域，float，默认值为0.5

min\_samples ：点成为核心点需在ϵ-邻域内拥有的点的最少个数，int，默认值5

其他的参数和属性大家可以从Sklearn的DBSCAN的介绍传送门中查看：

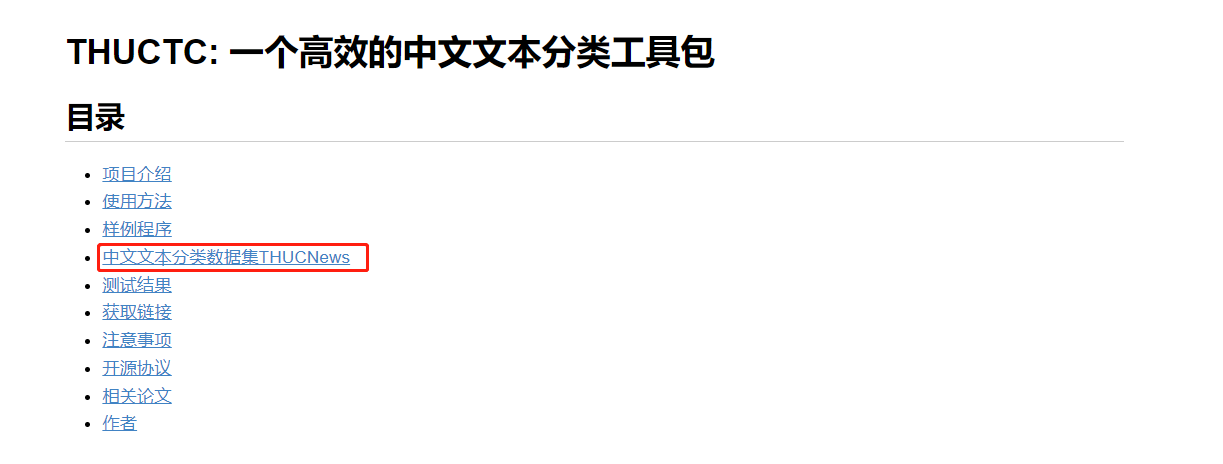
<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html>

1. **使用DBSCAN实现文本聚类**

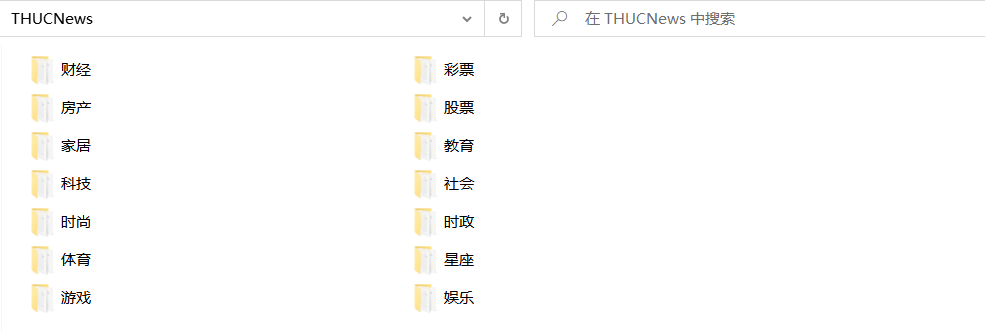
上述例子中我们聚类的样本是二维空间的向量，我们也可以使用DBSCAN进行文本聚类。唯一不一样的是我们需要将文本装换为向量即可。接下来我们就开始吧：

**6.1 数据集来源**

本次实验数据原来清华大学THUCNews数据集创送门：<http://thuctc.thunlp.org/>



没错这个数据集是真的很大，它包含了14类中文新闻，共n个txt文件，每个txt文件中1篇新闻。数据集目录如下所示：



**6.2 构造实验数据集**

本实验我们不使用全部的数据集，挑选其中财经、彩票、房产、股票、房居和教育数据集来进行实验，通过代码我们从以上6个分类中各选取300个样本：

import os  
import shutil  
import random  
  
def get\_random\_text(root\_path,save\_path):  
 if not os.path.exists(save\_path):  
 os.makedirs(save\_path)  
 category = ['财经','彩票','房产','股票','家居','教育']  
 get\_file\_num = 300  
 for i in range(len(category)):  
 file\_path = os.path.join(root\_path, category[i])  
 realfile = os.listdir(file\_path)  
 random\_index = random.sample(range(0, len(realfile)), get\_file\_num) *# 随机选取* for j in range(get\_file\_num):  
 raw\_file\_path = os.path.join(root\_path, category[i], realfile[random\_index[j]])  
 save\_file\_path = os.path.join(save\_path, category[i] + str(j) + ".txt")  
 shutil.copyfile(raw\_file\_path, save\_file\_path)  
get\_random\_text(r'E:\DataSet\THUCNews',r'F:\MyArticle\Pytorch\data')

这样我们就完成了数据的抽样。

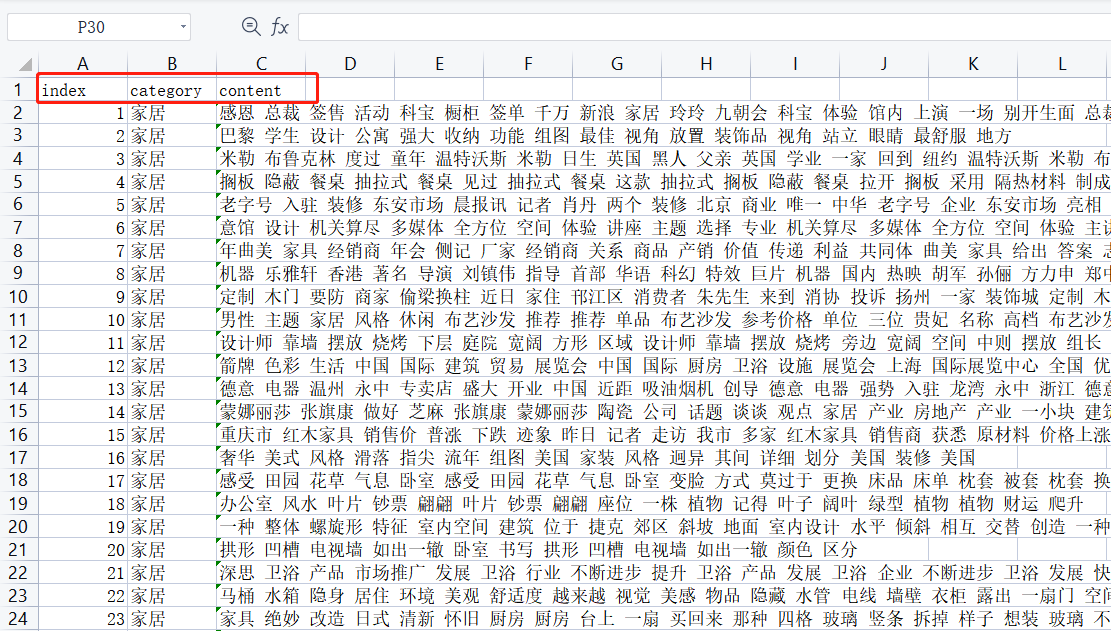
**6.3 分词和去停**

接下来我们将上述的数据装换为一个dadaframe的格式（也可以单独进行保存）：

我们对数据集进行清洗、分词并去除停用词，代码如下：

import os  
import re  
import jieba  
import pandas as pd  
  
def getM2Chinese(s):  
 *# 匹配两个字词以上的中文词语* pattern=r'^[\u4e00-\u9fa5]{2,}$'  
 regex = re.compile(pattern)  
 results = regex.findall(s)  
 return "". join(results)  
  
*# 创建停用词列表*def load\_stopwords(stopwords\_file):  
 *# 特殊字符列表* stopwords = ['\u3000', '\n', ' ']  
 if stopwords\_file:  
 with open(stopwords\_file, 'r', encoding='utf-8') as f:  
 for line in f:  
 stopwords.append(line.strip())  
 return stopwords  
  
def create\_corpus(stopwords,rawfilepath,cut\_sw\_res\_path):  
 *"""  
 Args:  
 stopwords: 停用词列表  
 rawfilepath: 原始数据文件夹路径  
 cut\_sw\_res\_path:各个文件分词保存结果根路径  
 Returns:  
 """* realfile = os.listdir(rawfilepath)  
 print('文本数量:',len(realfile))  
 """创建实验数据语料库"""  
 filename\_list = []  
 corpus = []  
 for i in range(0, len(realfile)):  
 filename = realfile[i]  
 *# 获取各个文本分类名称* filename\_list.append(filename[:2])  
 file\_full\_path = os.path.join(rawfilepath, filename)with open(file\_full\_path,encoding='utf-8') as f:  
 data = f.read()  
 *# 文本分词处理* cut\_data = jieba.cut(data,cut\_all=False)  
 *# 过滤停用词和* res\_cut\_data = ''  
 for each in cut\_data:  
 if each not in stopwords:if getM2Chinese(each):  
 res\_cut\_data = res\_cut\_data + each + ' '  
 corpus.append(res\_cut\_data)  
 *# 可以将各个不同分类文本分词、去除停用词后单独保存  
 # cut\_sw\_res\_file = os.path.join(cut\_sw\_res\_path, filename)  
 # with open(cut\_sw\_res\_file,'w', encoding='utf-8') as resf:  
 # 写入文件  
 # resf.write(res\_cut\_data)  
 # print(data\_adj, file=resf)  
  
 # 构造DataFrame数据文件* df = pd.DataFrame(columns=['index', 'category', 'content'])  
 df['index'] = [i+1 for i in range(len(corpus))]  
 df['category'] = filename\_list  
 df['content']= corpus  
 *# 保存*  
 df.to\_csv('myDataF.csv',index = False)  
   
stopwords\_file = r'F:\MyArticle\Pytorch\stopwords.txt'  
rawfilepath = r'F:\MyArticle\Pytorch\data'  
cut\_sw\_res\_path = r'F:\MyArticle\Pytorch\categoryfile'  
  
stopwords = load\_stopwords(stopwords\_file)  
create\_corpus(stopwords,rawfilepath,cut\_sw\_res\_path)

这样我们就可以输出我们的进行转换后的数据集了，截图示意如下：



**6.4 提取特征**

在此之前，我们来看一下Sklearn有关文本体征提取的一些方法：

先导入一些要使用的模块和使用的语料库：

from sklearn.feature\_extraction.text import TfidfTransformer  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.pipeline import Pipeline  
  
*# 语料库*corpus = ['this is the first document',  
 'this document is the second document',  
 'is this the first document'  
 ]

我们先看这一段代码：

*# 初始化对象*vectorizer = CountVectorizer()  
*# 将文本中的词语装换为词频矩阵*X = vectorizer.fit\_transform(corpus)  
*# 获取每一个词语的位置*loc = vectorizer.get\_feature\_names\_out()  
word = vectorizer.vocabulary\_  
print(X)  
print(X.toarray())  
print(X.shape)  
print(loc)  
print(word)

*# 输出为*

(0, 5) 1

(0, 2) 1

(0, 4) 1

(0, 1) 1

(0, 0) 1

(1, 5) 1

(1, 2) 1

(1, 4) 1

(1, 0) 2

(1, 3) 1

(2, 5) 1

(2, 2) 1

(2, 4) 1

(2, 1) 1

(2, 0) 1

[[1 1 1 0 1 1]

[2 0 1 1 1 1]

[1 1 1 0 1 1]]

(3, 6)

['document' 'first' 'is' 'second' 'the' 'this']

{'this': 5, 'is': 2, 'the': 4, 'first': 1, 'document': 0, 'second': 3}

可以看出CountVectorizer()的作用是将语料库中的文本文档装换为计数的稀疏矩阵。上述输出中X是词频矩阵，一个稀疏矩阵。(0,5) 1表示语料库中的第一行数据'this is the first document'第5个单词“document”出现了2次。第二行数据'this document is the second document'中的单词“document”出现了2次，所以(1,0) 2。通过使用toarray()方法可以将稀疏矩阵转换为计数矩阵。通过使用get\_feature\_names\_out()方法可以获取单词字典，vectorizer.vocabulary\_可获取相应的词和词在词袋中的索引值，再来看一些代码：

trainsform = TfidfTransformer()  
Y = trainsform.fit\_transform(X)  
print(Y)  
print(Y.toarray())

*# 输出*

(0, 5) 0.42040098658605557

(0, 4) 0.42040098658605557

(0, 2) 0.42040098658605557

(0, 1) 0.5413428136679054

(0, 0) 0.42040098658605557

(1, 5) 0.3183559679257789

(1, 4) 0.3183559679257789

(1, 3) 0.539023509507965

(1, 2) 0.3183559679257789

(1, 0) 0.6367119358515578

(2, 5) 0.42040098658605557

(2, 4) 0.42040098658605557

(2, 2) 0.42040098658605557

(2, 1) 0.5413428136679054

(2, 0) 0.42040098658605557

[[0.42040099 0.54134281 0.42040099 0. 0.42040099 0.42040099]

[0.63671194 0. 0.31835597 0.53902351 0.31835597 0.31835597]

[0.42040099 0.54134281 0.42040099 0. 0.42040099 0.42040099]]

TfidfTransforme很简单就是用来计算计算tf-idf，至于什么是tf-idf这里就不在赘述了。很幸运的是现在我们计算tf-idf的话可以直接使用sklearn中现有的方法

TfidfVectorizer来直接进行计算。

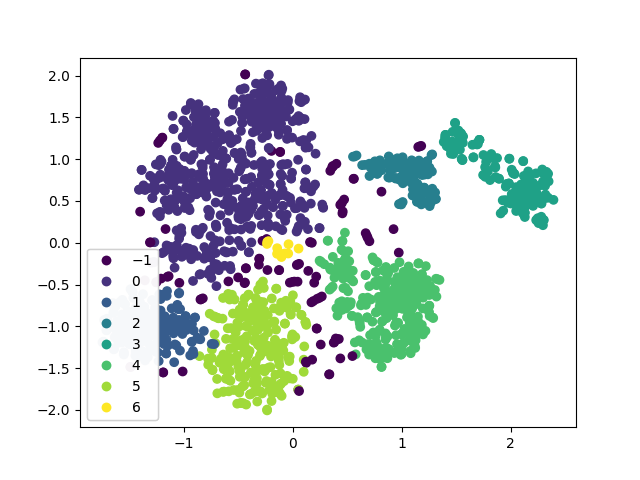
**6.5 聚类实验**

回到我们的案例,我们先将获取的数据进行向量化后抽取特征，之后进行稍微的降维，之后使用DBSCAN进行聚类：

import random  
import pandas as pd  
from matplotlib import pyplot as plt  
from sklearn import manifold, metrics  
from sklearn.cluster import DBSCAN  
from sklearn.feature\_extraction.text import TfidfTransformer, CountVectorizer  
from sklearn.preprocessing import StandardScaler  
  
def calculate\_tfidf(data):  
 *# 分词向量化* vectorizer = CountVectorizer()  
 *# 将文本转为词频矩阵* word\_v = vectorizer.fit\_transform(data)  
 *# 提取 TF-IDF 词向量* transformer = TfidfTransformer()  
 *# tf-idf* tfidf = transformer.fit\_transform(word\_v)  
 *# 转矩阵形式* tfidf\_matrix = tfidf.toarray()  
 *# 对tf-idf矩阵降维为了后续的可视化* tsne = manifold.TSNE(n\_components=2,random\_state=0)  
 tsne\_tfidf\_w = tsne.fit\_transform(tfidf\_matrix)  
 return tsne\_tfidf\_w  
  
  
def get\_text\_feature(filepath):  
 *# 读取数据* df = pd.read\_csv(filepath)  
 text = df['content'].values.tolist()  
 *# 打乱数据* random.shuffle(text)  
 tsne\_weights = calculate\_tfidf(text)  
 print('tsne\_weights shape:',tsne\_weights.shape)  
 *# StandardScaler一下* tsne\_weights = StandardScaler().fit\_transform(tsne\_weights)  
 *# 聚类* clf = DBSCAN(eps=0.14, min\_samples=8)  
 y = clf.fit\_predict(tsne\_weights)if True:  
 fig,ax = plt.subplots()  
 scatter = ax.scatter(tsne\_weights[:, 0], tsne\_weights[:, 1], c=y)  
 legend1 = ax.legend(\*scatter.legend\_elements(),loc="lower left")  
 ax.add\_artist(legend1)  
 plt.show()  
 *# labels\_属性为具体的标签*  
 labels = clf.labels\_  
 df['labels'] = pd.Series(clf.labels\_)  
 *# # labels=-1的个数除以总数，计算噪声点个数占总数的比例* raito = df.loc[df['labels'] == -1]['content'].count() / df['content'].count()  
 *# 获取分簇的数目* n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0) *# 每一个分簇的样本数量* every\_clust\_num = pd.Series(clf.labels\_).value\_counts()  
 *#* print('分簇的数目: %d' % n\_clusters\_)  
 print('噪声比:', format(raito, '.2%'))print("每一个种类下的样本数:", every\_clust\_num)  
  
filepath = r"F:\MyArticle\Pytorch\myDataF.csv"  
get\_text\_feature(filepath)

*#输出如下所示：*

tsne\_weights shape: (1800, 2)  
分簇的数目: 7  
噪声比: 5.61%  
每一个种类下的样本数: 0 631  
 4 296  
 5 264  
 1 202  
 3 175  
 2 119  
-1 101  
 6 12  
dtype: int64



可以看出我们的此次聚类分了7个簇，其中噪音点有101个，各个簇中的样本分布还是比较均衡的。代码就不在过多的进行解释了，方案中使用了TSNE进行降维，然后再进行DBSCAN的训练，大家也可以使用PCA进行数据的降维也可以尝试使用其他的降维方法。降维原因是分词后的词转向量之后的维度过高，我们在进行可视化的时候不好进行展示。接下来我们尝试使用PyTorch来实现我们的DBSCAN算法。

1. **PyTorch实现DBSCAN**

上述案例中我们借助Sklearn来实现文本的聚类的，接下来我们使用PyTorch来实现DBSCAN算法并我们的数据进行聚类，主要的代码如下：

import copy  
import pandas as pd  
import torch  
import numpy as np  
from sklearn import manifold  
from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer  
from sklearn.preprocessing import StandardScaler  
import matplotlib.pyplot as plt  
  
  
class TORCHDBSCAN():  
 def \_\_init\_\_(self, eps, minpts):  
 self.eps=eps  
 self.minpts=minpts  
  
 *# 计算tf-idf值* def calculate\_tfidf(self,data):  
 *# 分词向量化* vectorizer = CountVectorizer()  
 *# 将文本转为词频矩阵* word\_v = vectorizer.fit\_transform(data)  
 *# 提取 TF-IDF 词向量* transformer = TfidfTransformer()  
 *# tf-idf* tfidf = transformer.fit\_transform(word\_v)  
 *# 转矩阵形式* tfidf\_matrix = tfidf.toarray()  
 *# 对tf-idf矩阵降维为了后续的可视化* tsne = manifold.TSNE(n\_components=2, random\_state=0)  
 tsne\_tfidf\_w = tsne.fit\_transform(tfidf\_matrix)  
 return tsne\_tfidf\_w  
  
  
 *# 此函数计算点与其他点之间的距离，然后在数据集中返回阈值距离小于eps的点的列表。* def regionQuery(self,nowPoint,dataM):  
 *# 距离计算* dist=torch.sum(torch.sqrt((nowPoint['value']-dataM)\*\*2), dim=1)  
 *# 对于计算的距离小于eps的时候都加入到附近点* neighbor\_p\_l = list(torch.where(dist<=self.eps)[0])  
 return neighbor\_p\_l  
  
 def fitPoint(self, X\_matrix):  
 *"""  
 X\_matrix: tensor  
 """* belong\_leabel=1  
 *# 对每一个点进行访问* dataM = copy.deepcopy(X\_matrix)  
 *# 转换为字典形式如：{'value': tensor([ 0.6939, -0.1682]), 'label': 0}* dataDict = [{'value': eachval, 'label': 0} for eachval in (list(X\_matrix))]  
 *# 访问每一个点* for index, each\_point in enumerate(dataDict):  
 *# 如果这个点还没有被访问，则进行访问* if each\_point['label'] == 0:  
 neighbor\_p\_l=self.regionQuery(each\_point,dataM)  
 *# 如果是非噪声点(为核心点)* if len(neighbor\_p\_l)>=self.minpts:  
 *# 标记已经访问* dataDict[index]['label']=belong\_leabel  
 *# 开始聚类点增长* self.growCluster(neighbor\_p\_l, belong\_leabel, dataM, dataDict)  
 else:  
 *# 噪声点标记为-1* dataDict[index]['label']=-1  
 belong\_leabel+=1  
 *# 返回各个点标记的数据* return dataDict  
  
 def growCluster(self, neighbor\_p\_l, belong\_leabel,dataM, dataDict):  
 *# 停止条件* loop = 0  
 *# 访问和核心点集合中的每一个点* while (loop < len(neighbor\_p\_l)):  
 index=neighbor\_p\_l[loop]  
 *# 非噪声数据* if dataDict[index]['label']==0:  
 dataDict[index]['label']=belong\_leabel  
 temp\_neighbor\_l=self.regionQuery(dataDict[index],dataM)  
 if len(temp\_neighbor\_l)>=self.minpts:  
 *# 合并* neighbor\_p\_l=neighbor\_p\_l+temp\_neighbor\_l  
 *# 噪声数据* if dataDict[index]['label']==-1:  
 dataDict[index]['label']=belong\_leabel  
 loop+=1

在调用的时候，需要注意的是需要将输入数据tensor化：

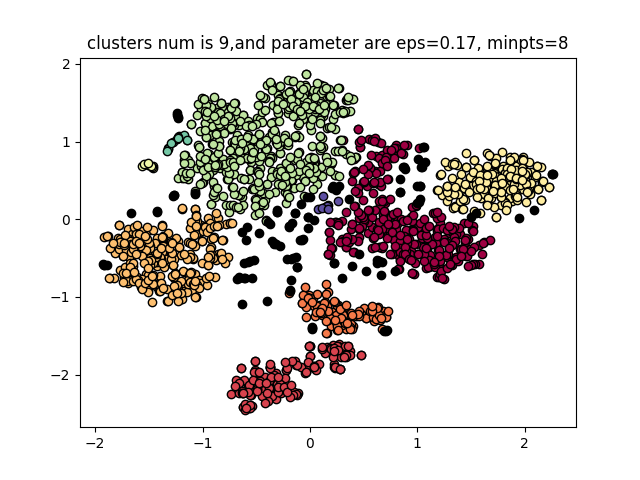
X = StandardScaler().fit\_transform(tsne\_weights)X\_matrix = torch.tensor(X)

在参数eps=0.17,minpts=8的条件下，我们获得得到结果：

分簇的数目: 9

噪声比: 6.06%

同时可视化结果如下：



可以看出实际的聚类的效果还是可以的。

通过调用results = cls.fitPoint(X\_matrix)对获得的变量results的数据进行解析就可以获取各个数据属于的簇。这个工作交给小伙伴自己研究了。

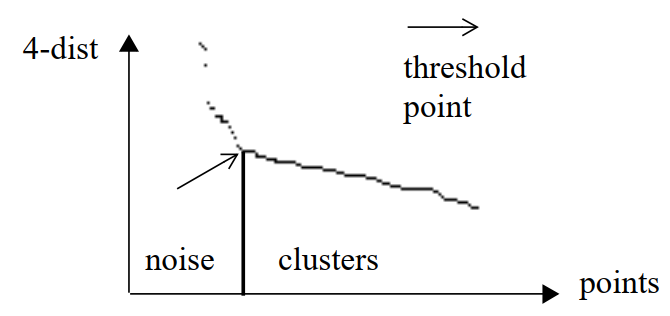
1. **DBSCAN参数和优缺点**

DBSCAN中两个比较注意的参数是ϵ-邻域和MinPts，那么怎么确定一个比较好的参数呢？在原论文中，使用以下步骤获取（大家可以研究一下原文的方法）：

（1）选取k值，一般为2\*维度-1。（其中维度为数据维度）

（2）计算并绘制k-distance图。

（3）找到拐点位置的距离，即为Eps的值。



当然我们也可使用网格搜索（暴力搜索）的思想进行参数寻优，也就是说不断使用一组参数去训练DBSCAN,然后统计参数值和聚类的情况，大致代码如下：

result = []  
for eps in np.arange(0.001,1,0.05):  
 for min\_samples in range(2,10):  
 dbscan = DBSCAN(eps = eps, min\_samples = min\_samples)  
 dbscan.fit(X)  
 *# 统计各参数组合下的聚类个数（-1表示异常点）* n\_clusters = len([i for i in set(dbscan.labels\_) if i != -1])  
 *# 异常点的个数*  noise = np.sum(np.where(dbscan.labels\_ == -1, 1, 0))  
 *# 统计每个簇的样本个数*  custing= str(pd.Series([i for i in dbscan.labels\_ if i != -1]).value\_counts().values)  
result.append({'eps':eps,'MinPts':min\_samples,

'n\_clusters':n\_clusters,'noise ':noise ,'custing':custing

})df = pd.DataFrame(res)

**DBSCAN优点：**

（1）可以能处理任意形状和大小的簇，而k-means之类的聚类算法一般只适用于凸数据集。

（2）对数据集中的异常点不敏感，且可以在聚类的时候进行异常点的检测。

**DBSCAN缺点：**

（1）如果样本集的密度不均匀、聚类间距差相差很大时，聚类质量较差，这时且对于高维问题，密度定义比较麻烦。

（2）如果样本集较大时，聚类收敛时间较长，此时可以对搜索最近邻时建立的KD树进行规模限制来改进。

（3）不同的参数组合对最后的聚类效果有较大影响。

1. **总结**

本次文章的内容是真的很多，涉及到数据的获取，数据的分词和停用词处理，聚类的实现和参数的寻优等等。大家也可以基于清华新闻数据集使用K-means进行分类,看看实际的聚类效果会有什么不同，也可以验证一下在数据集很大的时候PyTorch的聚类会不会在速度上有一定的优势。

最后参考链接中的几个GitHub资料大家可以好好研究一下，尤其是一个基于“HR\_data.csv”的数据集进行的实验。

1. **参考资料**

《机器学习》-周志华

《Machine Learning with R, the tidyverse, and mlr 》

<https://en.wikipedia.org/wiki/DBSCAN>

https://blog.csdn.net/Cyrus\_May/article/details/111499606

<https://blog.csdn.net/Cyrus_May/article/details/113504879>

<https://github.com/siddharth-agrawal/DBSCAN>

<https://github.com/LindsayMoir/dbscan>

https://github.com/Kmysiak/hr\_clustering