

共词聚类

基于关键词共现的文献主题聚类方法及局限

汇报人：杜朋东 日期：2020.05.19

聚类算法及局限分析

目录
CONTENTS

01

关键词抽取及共现矩阵

02

03

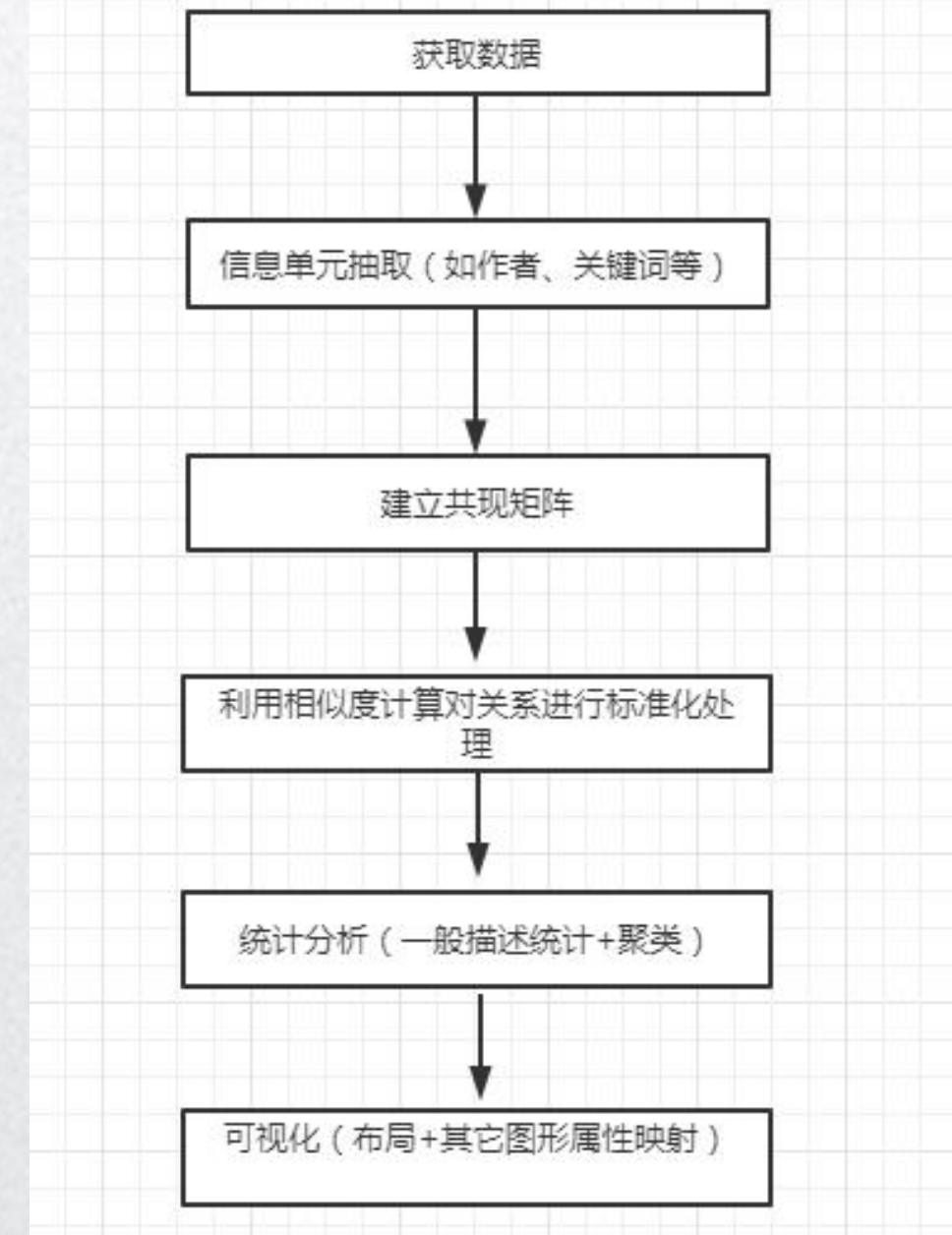
聚类效果演示并比较

PART
01

关键词抽取及共现矩阵

共词聚类的一般流程：

获取数据——信息单元抽取
(如作者、关键词等)——建立
共现矩阵——利用相似度计算对
关系进行标准化处理——统计分
析(一般描述统计+聚类)——可
视化展现(布局+其它图形属性映
射)



科技论文主要共现类型

x可以进行共现分析的项目

+作者：合著

+单位：合作

+主题词、关键词：共词分析

+引文：同被引分析、耦合分析

序号	特征项一	特征项二	特征项一、二之间关系	共现类型
1	论文	关键词	论文中使用了关键词	异共现
2	论文	期刊	论文发表的期刊	异共现
3	论文	作者	作者撰写了论文	异共现
4	论文	引文	论文引用了引文	异共现
5	作者	作者机构	作者与作者所隶属的机构	异共现
6	引文	引文作者	引文作者撰写了引文	异共现
7	引文	引文期刊	引文发表的期刊	异共现
8	引文	引文	引文之间的关联（论文同被引）	同共现
9	论文	论文	论文之间的关联（文献耦合）	同共现
10	作者	作者	作者之间的关联（作者合作）	同共现
11	引文作者	引文作者	引文作者之间的关联（作者同被引）	同共现
12	关键词	关键词	内容相关的关键词	同共现
13	关键词	作者机构	机构论文涉及的研究词汇	异共现
14	关键词	期刊	期刊论文涉及的研究词汇	异共现

共词分析一不足

共词分析对词的选择非常敏感，作者的取词习惯、未经规范的关键词、关键词在表征论文内容的完整性及其它原因都会造成结论的模糊、晦涩。些研究认为共词分析存在随意性较大、不确定性的缺陷。

> 共现矩阵

共现矩阵

通过对高频主题词的词频统计分析，我们可以了解到目前某一专题领域里研究的热点。但是，仅仅对这些主题词按照出现频次由高到低的排列还不能表现出这些高频主题词之间的联系，因此我们可以采用共现分析的技术来进一步挖掘这些主题词之间的联系。主题词的共现分析是根据主题词在同一篇论文中共同出现的次数来表示主题词之间的联系。一般认为，如果两个主题词频繁在同一篇论文中同时出现，往往表明这两个主题词之间具有比较密切的联系。这就是共现分析的理论基础。

主题 同行评议

同行评议的指数分析结果

分组浏览：主题 发表年度

auto (14113) 同行评议 (166)
educator compet... (913) journal
drug trials (375) >>

排序：相关度 发表时间↓ 已选文献： 309 清除

数据来源：中国知网数据库

步骤：利用“同行评议”作为检索词进行主题检索，按照学科领域“图书情报”精炼选择其中309篇文章，以“refworks”导出文本 (不足：为作演示方便，选取的数据量很小)

Cnki 中国知网 cnki.net | 文献管理中心-文献输出

文献导出格式

- GB/T 7714-2015 格式引文
- CAJ-CD格式引文
- 查新（引文格式）
- 查新（自定义引文格式）
- 知网研学（原E-Study）
- Refworks**
- EndNote
- NoteExpress
- NoteFirst
- 自定义

Refworks ?

以下是在您将按照当前格式导出的文献，如需重选文献 [请点击这里](#)

发表时间↓ 被引频次

导出 复制到剪贴板 打印 XLS DOC 生成检索报告

RT Journal Article
SR 1
A1 王凌峰;唐碧群;
AD 桂林电子科技大学商学院;桂林电子科技大学图书馆;
T1 论文发表系统:完备分类与中长期格局演化
JF 情报探索
YR 2020
IS 04
OP 1-8
K1 论文发表系统;分类;F1000 Research;预印本;在线评级平台;演化 paper publishing system;classification;F1000 Research;preprint;online rating platform;evolution
AB [目的/意义]互联网时代出现看多种新论文发表系统,对多种论文发表系统之间逻辑分类关系以及彼此之间的中长期格局演化进行讨论。[方法/过程]从论文发表系统对一篇论文是否进行评议、评议是否分级、是否拒稿3个维度提出论文交流系统分类框架,将期刊、预印本、F1000 Research h、在线评级平台4种论文发表系统全部统一在一个分类框架中。另外,基于研究者与研究管理部门对论文交流系统的不同需求,定性分析了论文交流系统的中长期演化格局。[结果/结论]中长期看,在线评级平台将对期刊系统形成明显冲击,期刊内部两极分化。

```

# CalHamletV1.py
def getText():
    txt = open("hamlet.txt", "r", encoding='utf-8').read()
    txt = txt.lower()
    for ch in '!"#$%&()*+,-./;:<=>?@[\\]^_`{|}~':
        txt = txt.replace(ch, " ")
    # 将文本中特殊字符替换为空格
    return txt

```

```

hamletTxt = getText()
words = hamletTxt.split()
counts = {}
for word in words:
    counts[word] = counts.get(word, 0) + 1
items = list(counts.items())
items.sort(key=lambda x: x[1], reverse=True)
for i in range(10):
    word, count = items[i]
    print("{0:<10}{1:>5}".format(word, count))

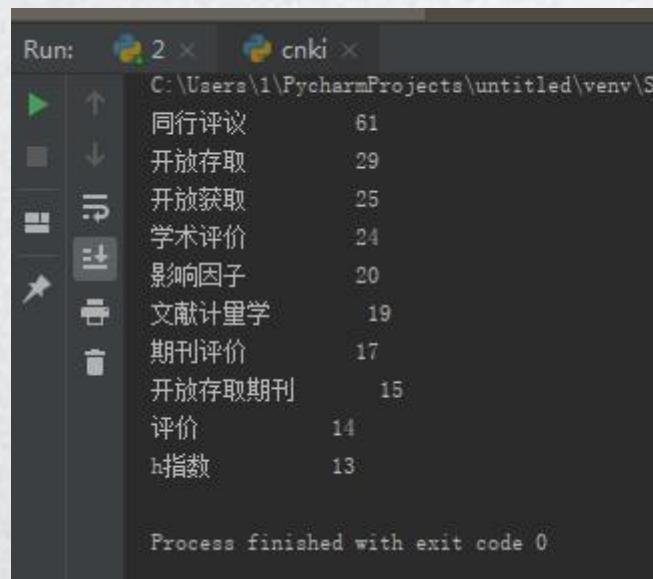
```



能用TF-IDF、TextRank
(统计)、Word2Vec词聚类(机器学习)提取文本的关键词

高频关键词如下：

论文发表系统；分类；F1000 Research；预印本；在线评级平台；演化
SCI论文；科技评价问题；四唯；文献计量
预印本系统；学术评价；同行评议；评分模型；激励机制
学术论文评价；智能评价；同行评议；引文指标；选题新颖度；创新性评价
ESI；ESI高被引；Altmetrics；学术影响力；社会影响力；同行评议
贝尔纳；期刊取消论；预印本；同行评议；自组织同行评议
中文字术图书；Altmetrics；学术评价；学术影响力；社会影响力
学术影响力；评价；科学计量学；同行评议
标签词；学术博客；图书情报学；非正式交流渠道；信息计量；研究热点
《2017环境扫描报告》；学术图书馆；高等教育模式转型；管理研究数据；开放获取
人文社会科学；科学评价；大数据
开放存取(OA)；同行评审；系统理论
学术创新力；评价；机器学习；指标体系；特征重要性
预印本；同行评议；中国科技论文在线；高校认可
计量溯；负责任计量；知情同行评议；科学评价
学术期刊；学术地位；地位流动；同行评议
环境教育；网络图谱；热点主题；研究前沿；CiteSpace
学术评价；核心期刊；CSSCI；定性评价；定量评价；代表作评价制度
生存优势；耶鲁大学；阅读者；
科技期刊；论文；撤稿
档案学术；理性评价；评价制度



不足：技术允许的话，从标题、摘要、文章内容中分词，并提取高频关键词会更好

输出若干篇文章的K1关键词频次

```
import pandas as pd
pd.set_option('display.max_columns', None)

#将每一份文本的标签隔开，转换成列表
def readlabel(path,label_name, id_name):
    read_data = pd.read_csv(path)
    read_data[label_name] = read_data[label_name].apply(lambda x: '/' + x)
    data_concat = read_data.groupby(by=id_name).sum()
    data_concat[label_name] = data_concat[label_name].apply(lambda x: x[1:])
    data_concat.index = range(len(data_concat[label_name]))
    row_num = len(data_concat[label_name])
    data = []
    for i in range(0, row_num):
        data.append(data_concat.loc[i,label_name])
    return (data)
```

```
#将标签列表转换为二维数组
def format_data(data):
    formated_data = []
    for ech in data:
        ech_line = ech.split('/')
        formated_data.append(ech_line)
    return formated_data
```

#建立关于标签的字典

```
def dic(readpath, label_name):
    label_data = pd.read_csv(readpath)
    label_list = list(set(label_data[label_name].values.tolist()))
    labeldic = {}
    pos = 0
    for i in label_list:
        pos = pos+1
        labeldic[pos] = str(i)
    # print(labeldic)
    return labeldic
```

#构建空矩阵用于存放标签的共现矩阵

```
def buildmatrix(x, y):
    return [[0 for j in range(y)] for i in range(x)]
```

#将标签分行列构建初始共现矩阵用于存放计算结果

```
def inimatrix(matrix, dic, length):
```

```
    matrix[0][0] = 'label'
    for i in range(1, length):
        matrix[0][i] = dic[i]
    for i in range(1, length):
        matrix[i][0] = dic[i]
```

```
return matrix
```

```
#计算标签与标签之间的共现次数
def countmatirx(matrix, formated_data):
    keywordlist=matrix[0][1:] #列出所有标签
    appeardict={} #各个标签为键，出现在哪些文本（将文本按顺序排列，出现的序号添加到列表中）为值，构造字典
    for w in keywordlist:
        appearlist=[]
        i=0
        for each_line in formated_data:
            print(each_line)
            if w in each_line:
                appearlist.append(i)
            i +=1
        appeardict[w]=appearlist
    print(appeardict)
    for row in range(1, len(matrix)):
        # 遍历矩阵行标签
        for col in range(1, len(matrix)):
            # 遍历矩阵列标签
            # 实际上就是为了跳过matrix中下标为[0][0]的元素，因为[0][0]为空
            if col >= row:
```

#仅计算上半个矩阵

```
if matrix[0][row] == matrix[col][0]:  
    # 如果取出的行标签和列标签相同，则其对应的共现次数为0，即矩阵对角线为0  
    matrix[col][row] = int(0)  
else:  
    counter = len(set(appeardict[matrix[0][row]])&set(appeardict[matrix[col][0]]))  
  
    matrix[col][row] = int(counter)  
else:  
    matrix[col][row]=matrix[row][col]  
return matrix
```

#取出标签之间的共现值用于PMI计算

```
def label_concurrence_value(data, a, b):  
    con_value = data.loc[a,b]  
    return con_value
```

```

def main():
    readpath = r'C:\git-test-master\concorrence_matrix\test.csv' #原始输入数据
    save_path = r'C:\git-test-master\concorrence_matrix\concurrence_data.xlsx' #存放共现矩阵
    co_calcul_path = r'C:\git-test-master\concorrence_matrix\test1.xlsx' #需要计算PMI值的文档
    co_result_path = r'C:\git-test-master\concorrence_matrix\test_result.xlsx' #存放计算结果
    label_list = readlabel(readpath, label_name='label', id_name='id') #将每一份文本的标签隔开，转换成列表
    formated_data = format_data(label_list)
    label_dict = dic(readpath, label_name='label') #建立关于标签的字典
    length = len(label_dict)+1
    matrix = buildmatrix(length, length) #构建空矩阵用于存放标签的共现矩阵
    matrix = inimatrix(matrix, label_dict, length) #将标签分行列构建初始共现矩阵用于存放计算结果
    matrix = countmatirx(matrix, formated_data) #计算标签与标签之间的共现次数
    concurrence_data = pd.DataFrame(matrix, columns=matrix[0], index=matrix[0])
    concurrence_data = concurrence_data.drop(['label'], axis=0)
    concurrence_data = concurrence_data.drop(['label'], axis=1)
    # print(concurrence_data)
    concurrence_data.to_excel(save_path)
    co_data = pd.read_excel(co_calcul_path)
    row_num = co_data.shape[0]

```

选定高频词

同行评议	61
开放存取	29
开放获取	25
学术评价	24
影响因子	20
文献计量学	19
期刊评价	17
开放存取期刊	15
评价	14
h指数	13

61	peer review
54	open aces (合并后)
24	academic evaluation
20	impact factor
19	bibliometrics

```

for j in range(0, row_num):
    data1 = co_data.loc[j, 'A标签']
    data2 = co_data.loc[j, 'B标签']
    con_value = label_concurrence_value(concurrence_data, data1, data2)
    co_data.loc[j, '共现频次'] = con_value
print(co_data)
co_data = co_data[['A标签', 'B标签', '共现频次']]
co_data.to_excel(co_result_path)

if __name__ == '__main__':
    main()

```

	A标签	B标签	共现频次
0	peer review	open acess	54
1	peer review	academic evaluation	24
2	peer review	impact factor	20
3	peer review	bibliometrics	19
4	open acess	bibliometrics	19
5	open acess	impact factor	20
6	open acess	academic evaluation	24
7	bibliometrics	impact factor	19
8	bibliometrics	academic evaluation	19
9	impact factor	academic evaluation	20

假设id为“doi号”，“peer review”在61篇文章中出现

	A标签	B标签	共现频次
0	peer review	open acess	54.0
1	peer review	academic evaluation	24.0
2	peer review	impact factor	20.0
3	peer review	bibliometrics	19.0
4	open acess	bibliometrics	19.0
5	open acess	impact factor	20.0
6	open acess	academic evaluation	24.0
7	bibliometrics	impact factor	19.0
8	bibliometrics	academic evaluation	19.0
9	impact factor	academic evaluation	20.0

生成的高频关键词共词矩阵

	open acess	bibliometrics	impact factor	peer review	academic evaluation
open acess	0	19	20	54	24
bibliometrics	19	0	19	19	19
impact factor	20	19	0	20	20
peer review	54	19	20	0	24
academic evaluation	24	19	20	24	0

id	label
1	peer review
2	peer review
3	peer review
4	peer review
5	peer review
6	peer review
7	peer review
8	peer review
9	peer review
10	peer review
11	peer review
12	peer review
58	peer review
59	peer review
60	peer review
61	peer review
1	open acess
2	open acess
3	open acess
4	open acess
5	open acess
6	open acess
7	open acess
8	open acess
47	open acess
48	open acess
49	open acess
50	open acess
51	open acess
52	open acess
53	open acess
54	open acess
1	academic evaluation
2	academic evaluation
3	academic evaluation
4	academic evaluation
5	academic evaluation
6	academic evaluation

改进：利用Python实现中文文本关键词抽取的三种方法

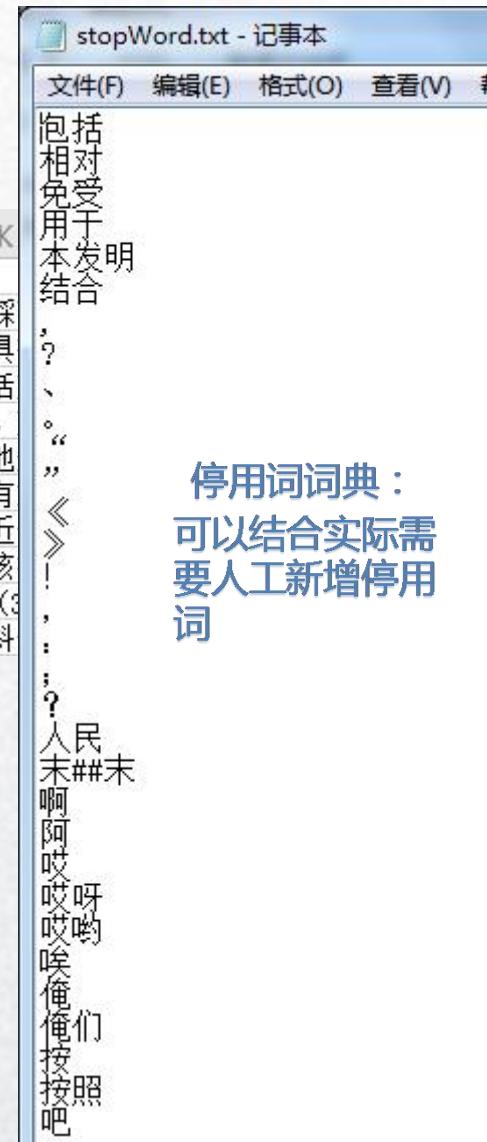
分别采用TF-IDF方法、TextRank方法和Word2Vec词聚类方法实现对专利文本（同样适用于其它类型文本）的关键词抽取

将汽车行业的10篇专利作为样本数据集，见文件

“data/sample_data.csv”。文件中依顺序包含编号（id）、标题（title）和摘要（abstract）三个字段，其中标题和摘要都要参与到关键词的抽取。

A	B	C	D	E	F	G	H	I	J	K
1	id	title	abstract							
2	1	永磁电机驱动本发明公开了一种永磁电机驱动的纯电动大巴车坡道起步防溜策略，即本策略当制动踏板已踩								
3	2	机动车车辆一种溃缩结构是作为内部支撑件而被提供在机动车辆的车门衬板上的肘靠中，所述溃缩结构具								
4	3	仪表板支撑本发明公开了一种支撑结构，其配置用于在车辆的乘客车厢内定位仪表板。所述支撑结构包括								
5	4	铰接的头枕一种车辆座椅总成，包括座椅靠背、头枕和支承结构，支承结构在头枕和座椅靠背之间延伸。								
6	5	用于评估和本发明涉及用于评估和控制电池系统的系统和方法。介绍了用于估计电池系统中的各独立电池								
7	6	侧气囊装置本发明公开了一种侧气囊装置，其中，横向分隔件由一对结构织物部形成，各结构织物部具有								
8	7	制造气囊的本发明公开了以下方式制造气囊。在第一结合步骤中，使各结构织物部中安装时被引到更靠近								
9	8	上部椅背托一种车辆座椅总成，其包括第一和第二侧支承部，该第一和第二侧支承部限定了椅背结构，该								
10	9	用于在机动车本发明描述了一种用于在具有手动变速器(33)的机动车(32)的行驶过程中关闭和开启内燃机(3								
11	10	半倾斜货箱本发明涉及半倾斜货箱卸载系统。一变型可包括一种产品，包括：运输工具，其包括具有倾斜								

```
PS C:\Users\1> pip install jieba
Requirement already satisfied: jieba in c:\users\1\anaconda3\lib\site-packages
PS C:\Users\1> pip install --user jieba
Requirement already satisfied: jieba in c:\users\1\anaconda3\lib\site-packages
PS C:\Users\1> pip install numpy
Requirement already satisfied: numpy in c:\users\1\anaconda3\lib\site-packages
PS C:\Users\1> pip install --user numpy
Requirement already satisfied: numpy in c:\users\1\anaconda3\lib\site-packages
PS C:\Users\1> pip install scipy
Requirement already satisfied: scipy in c:\users\1\anaconda3\lib\site-packages
Requirement already satisfied: numpy>=1.13.3 in c:\users\1\anaconda3\lib\site-packages
PS C:\Users\1> pip install --user scipy
Requirement already satisfied: scipy in c:\users\1\anaconda3\lib\site-packages
Requirement already satisfied: numpy>=1.13.3 in c:\users\1\anaconda3\lib\site-packages
PS C:\Users\1> pip install gensim
Collecting gensim
  Downloading gensim-3.8.3-cp37-cp37m-win_amd64.whl (24.2MB)
    686 kB 27 kB/s
```



停用词词典：
可以结合实际需要人工新增停用词

Select	Cluster ID	Size	Silhouette	mean(Year)	Top Terms (tf*idf weighting)
<input type="checkbox"/>	13	3	1	2010	(7.48) "隐私" 保护"; (7.48)...
<input type="checkbox"/>	12	7	0.903	2010	(10.5) "伦理责任"; (9.3) "技术..."
<input type="checkbox"/>	11	7	0.775	2009	(8.6) "伦理问题"; (7.48) "环境..."
<input type="checkbox"/>	10	3	0.823	2011	(5.86) "社会"; (5.86) "科学知..."
<input type="checkbox"/>	9	5	0.642	2009/2011	"隐私"; (7.48) "保护"; (7.48)...

1 基于TF-IDF的文本关键词抽取方法

TF-IDF算法思想

词频 (Term Frequency, TF) 指某一给定词语在当前文件中出现的频率。由于同一个词语在长文件中可能比短文件有更高的词频，因此根据文件的长度，需要对给定词语进行归一化，即用给定词语的次数除以当前文件的总词数。

逆向文件频率 (Inverse Document Frequency, IDF) 是一个词语普遍重要性的度量。即如果一个词语只在很少的文件中出现，表示更能代表文件的主旨，它的权重也就越大；如果一个词在大量文件中都出现，表示不清楚代表什么内容，它的权重就应该小。

TF-IDF的主要思想是，如果某个词语在一篇文章中出现的频率高，并且在其他文章中较少出现，则认为该词语能较好的代表当前文章的含义。即一个词语的重要性与它在文档中出现的次数成正比，与它在语料库中文档出现的频率成反比。

计算公式如下：

$$\text{词频 (TF)} = \frac{\text{词 w 在文档中出现的次数}}{\text{文档的总词数}}$$

$$\text{逆文档频率 (IDF)} = \log \left(\frac{\text{语料库的文档总数}}{\text{包含词 w 的文档数} + 1} \right)$$

$$\text{TF-IDF} = \text{TF} * \text{IDF}$$

代码实现

基于TF-IDF方法实现文本关键词抽取的代码执行步骤如下：

(1) 读取样本源文件sample_data.csv;

(2) 获取每行记录的标题和摘要字段，并拼接这两个字段；

(3) 加载自定义停用词表stopWord.txt，并对拼接的文本进行数据预处理操作，包括分词、筛选出符合词性的词语、去停用词，用空格分隔拼接成文本；

(4) 遍历文本记录，将预处理完成的文本放入文档集corpus中；

(5) 使用CountVectorizer()函数得到词频矩阵， $a[j][i]$ 表示第j个词在第i篇文档中的词频；

(6) 使用TfidfTransformer()函数计算每个词的tf-idf权值；

(7) 得到词袋模型中的关键词以及对应的tf-idf矩阵；

(8) 遍历tf-idf矩阵，打印每篇文档的词汇以及对应的权重；

(9) 对每篇文档，按照词语权重值降序排列，选取排名前topN个词最为文本关键词，并写入数据框中；

(10) 将最终结果写入文件keys_TFIDF.csv中。

keyextract_tfidf
防溜 0.0
降低 0.09193682992190591
限制 0.0
限定 0.0
随后 0.0
靠背 0.0
靠近 0.0
面板 0.0
驱动 0.0
驱动器 0.0
气囊 0.0

A	B	C	D	E	F	G	H
1	id	title	key				
2		1 永磁电机	永磁电机	防溜	永磁	控制	策略
3		2 机动车	车	车辆	结构	安排	特别
4		3 仪表板	支撑	支架	横向	端部	偏压
5		4 铰接的	头枕	座椅	靠背	联接	支承
6		5 用于评估	电池	系统	容量	分部	总和
7		6 侧气囊	装置	织物	结合部	结构	内管
8		7 制造气囊	的织物	主体	步骤	结构	安装
9		8 上部椅背	椎椅背	枢轴	悬挂	组件	位置
10		9 用于在机	动内燃机	机动车	方法	行驶	启动
11		10 半倾斜货	新倾斜	货箱	纵向	部分	最近
12							TFIDF

2 基于TextRank的文本关键词抽取方法

PageRank算法思想

TextRank算法是基于PageRank算法的，因此，在介绍TextRank前不得不了解一下PageRank算法。

PageRank算法是Google的创始人拉里·佩奇和谢尔盖·布林于1998年在斯坦福大学读研究生期间发明的，是用于根据网页间相互的超链接来计算网页重要性的技术。该算法借鉴了学术界评判学术论文重要性的方法，即查看论文的被引用次数。基于以上想法，PageRank算法的核心思想是，认为网页重要性由两部分组成：

- ① 如果一个网页被大量其他网页链接到说明这个网页比较重要，即被链接网页的数量；
- ② 如果一个网页被排名很高的网页链接说明这个网页比较重要，即被链接网页的权重。

一般情况下，一个网页的PageRank值（PR）计算公式如下所示：

PageRank计算公式

$$PR(p_i) = \frac{1 - \alpha}{N} + \alpha \sum_{p_j \in M_{p_i}} \frac{PR(p_j)}{L(p_j)}$$

其中， $PR(P_i)$ 是第*i*个网页的重要性排名即PR值； α 是阻尼系数，一般设置为0.85； N 是网页总数； M_{p_i} 是所有对第*i*个网页有出链的网页集合； $L(p_j)$ 是第*j*个网页的出链数目。

初始时，假设所有网页的排名都是 $1/N$ ，根据上述公式计算出每个网页的PR值，在不断迭代趋于平稳的时候，停止迭代运算，得到最终结果。一般来讲，只要10次左右的迭代基本上就收敛了。

代码实现

基于TextRank方法实现文本关键词抽取的代码执行步骤：

- (1) 读取样本源文件sample_data.csv;
- (2) 获取每行记录的标题和摘要字段，并拼接这两个字段；
- (3) 加载自定义停用词表stopWord.txt;
- (4) 遍历文本记录，采用jieba.analyse.textrank函数筛选出指定词性，以及topN个文本关键词，并将结果存入数据框中；
- (5) 将最终结果写入文件keys_TextRank.csv中。

最终运行结果如右图所示。

id	title	key
1	永磁电机驱动控制 防溜 电机 永磁 单元 踏板 策略 车辆 整车 转速	
2	机动车车辆结构 溃缩 车辆 车门 机动车辆 吸收 负荷 增加 交叉 元件	
3	仪表板支撑支架 支撑 横向 仪表板 偏压 端部 导引 面板 车身 车辆	
4	铰接的头枕头枕 构件 座椅 靠背 联接 支承 结构 枢转地 总成 驱动器	
5	用于评估和电池 系统 总和 容量 控制 分部 计算 方法 利用 独立	
6	侧气囊装置织物 结构 结合部 膨胀 横向 分隔 装置 越过 延伸 内管	
7	制造气囊的织物 结构 主体 气囊 制造 部分 部上 安装 重叠 方法	
8	上部椅背枢轴背 构架 枢轴 位置 悬挂 组件 总成 耦接 斜倚 坚直	
9	用于在机动车方法 内燃机 关闭 启动 行驶 状态 机动车 变速器 过程 手动	
10	半倾斜货箱倾斜 部分 货箱 具有 纵向 卸载 运输工具 最靠近 系统 变型	
11		TextRank
12		
13		

```
keyextract_textrank
电池 系统 总和 容量 控制 分部 计算 方法 利用 独立
"侧气囊装置" 10 Keywords - TextRank:
织物 结构 结合部 膨胀 横向 分隔 装置 越过 延伸 内管
"制造气囊的方法" 10 Keywords - TextRank:
织物 结构 主体 气囊 制造 部分 部上 安装 重叠 方法
"上部椅背枢轴系统" 10 Keywords - TextRank:
椅背 结构 枢轴 位置 悬挂 组件 总成 耦接 斜倚 坚直
"用于在机动车的行驶过程中关闭和启动内燃机的方法" 10 Keywords - TextRank:
方法 内燃机 关闭 启动 行驶 状态 机动车 变速器 过程 手动
"半倾斜货箱卸载系统" 10 Keywords - TextRank:
倾斜 部分 货箱 具有 纵向 卸载 运输工具 最靠近 系统 变型

Process finished with exit code 0
```

3 基于Word2Vec词聚类的文本关键词抽取方法

Word2Vec词向量表示

众所周知，机器学习模型的输入必须是数值型数据，文本无法直接作为模型的输入，需要首先将其转化成数学形式。基于Word2Vec词聚类方法正是一种机器学习方法，需要将候选关键词进行向量化表示，因此要先构建Word2Vec词向量模型，从而抽取出候选关键词的词向量。

K-means聚类算法

聚类算法旨在数据中发现数据对象之间的关系，将数据进行分组，使得组内的相似性尽可能的大，组件的相似性尽可能的小。

K-Means是一种常见的基于原型的聚类技术，本文选择该算法作为词聚类的方法。其算法思想是：首先随机选择K个点作为初始质心，K为用户指定的所期望的簇的个数，通过计算每个点到各个质心的距离，将每个点指派到最近的质心形成K个簇，然后根据指派到簇的点重新计算每个簇的质心，重复指派和更新质心的操作，直到簇不发生变化或达到最大的迭代次数则停止。

基于Word2Vec词聚类文本关键词抽取方法流程

Word2Vec词聚类文本关键词抽取方法的主要思路是对于用词向量表示的文本词语，通过K-Means算法对文章中的词进行聚类，选择聚类中心作为文章的一个主要关键词，计算其他词与聚类中心的距离即相似度，选择topN个距离聚类中心最近的词作为文本关键词，而这个词间相似度可用Word2Vec生成的向量计算得到。

假设Dn为测试语料的大小，使用该方法进行文本关键词抽取的步骤如下所示：

- (1) 对Wiki中文语料进行Word2vec模型训练，参考我的文章“利用Python实现wiki中文语料的word2vec模型构建”(<http://www.jianshu.com/p/ec27062bd453>)，得到词向量文件“wiki.zh.text.vector”；
- (2) 对于给定的文本D进行分词、词性标注、去重和去除停用词等数据预处理操作。本分采用结巴分词，保留'n','nz','v','vd','vn','l','a','d'这几个词性的词语，最终得到n个候选关键词，即 $D=[t_1, t_2, \dots, t_n]$ ；
- (3) 遍历候选关键词，从词向量文件中抽取候选关键词的词向量表示，即 $WV=[v_1, v_2, \dots, v_m]$ ；
- (4) 对候选关键词进行K-Means聚类，得到各个类别的聚类中心；
- (5) 计算各类别下，组内词语与聚类中心的距离（欧几里得距离），按聚类大小进行升序排序；
- (6) 对候选关键词计算结果得到排名前TopN个词汇作为文本关键词。

步骤(4)中需要人为给定聚类的个数，本文测试语料是汽车行业的专利文本，因此只需聚为1类，各位可根据自己的数据情况进行调整；步骤(5)中计算各词语与聚类中心的距离，常见的方法有欧式距离和曼哈顿距离，本文采用的是欧式距离，计算公式如下：

欧式距离计算公式

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

代码实现

Python第三方工具包Scikit-learn提供了K-Means聚类算法的相关函数，本次用到了sklearn.cluster.KMeans()函数执行K-Means算法、sklearn.decomposition.PCA()函数用于数据降维以便绘制图形。

基于Word2Vec词聚类方法实现文本关键词抽取的代码执行步骤：

(1) 读取样本源文件sample_data.csv；

(2) 获取每行记录的标题和摘要字段，并拼接这两个字段；

(3) 加载自定义停用词表stopWord.txt，并对拼接的文本进行数据预处理操作，包括分词、筛选出符合词性的词语、去重、去停用词，形成列表存储； 使用python对中文wiki语料的词向量建模就全部结束了，`wiki.zh.text.vector`中是每个词对应的词向量，可以在此基础上作文本特征的提取以及分类

(4) 读取词向量模型文件'wiki.zh.text.vector'，从中抽出所有候选关键词的词向量表示，存入文件中；

(5) 读取文本的词向量表示文件，使用KMeans()函数得到聚类结果以及聚类中心的向量表示；

(6) 采用欧式距离计算方法，计算得到每个词语与聚类中心的距离；

(7) 按照得到的距离升序排列，选取排名前topN个词作为文本关键词，并写入数据框中

(8) 将最终结果写入文件keys_word2vec.csv中。

	A	B	C	D	E	F	G
1	id	title	key				
2	1	永磁电机	驻油门	设定值	踩下	使能	永磁
3	2	机动车辆	机动车辆	衬板	凹陷	峰值	负荷
4	3	仪表板支撑	仪表板	端部	偏压	支架	面板
5	4	铰接的头枕支承	联接	头枕	靠背	总成	铰接
6	5	用于评估和电池组	储能	相关联	电能	导放电	总和
7	6	侧气囊装置内管	结合部	气囊	使该	横向	织物
8	7	制造气囊的摊开	部上	引到	气囊	主体	部分
9	8	上部椅背枢支承	椅背	枢轴	竖直	总成	斜倚
10	9	用于在机动车发生器	空挡	内燃机	停转	机动车	变速器
11	10	半倾斜货箱运输工具	卸载	货箱	最靠近	变型	纵向
12							
13							

word2vec

```
keyextract_textrank
电池 系统 总和 容量 控制 分部 计算 方法 利用 独立
“侧气囊装置” 10 Keywords - TextRank :
织物 结构 结合部 胫膨 横向 分隔 装置 越过 延伸 内管
“制造气囊的方法” 10 Keywords - TextRank :
织物 结构 主体 气囊 制造 部分 部上 安装 重叠 方法
“上部椅背枢轴系统” 10 Keywords - TextRank :
椅背 结构 枢轴 位置 悬挂 组件 总成 鞍接 斜倚 竖直
“用于在机动车的行驶过程中关闭和启动内燃机的方法” 10 Keywords - TextRank :
方法 内燃机 关闭 启动 行驶 状态 机动车 变速器 过程 手动
“半倾斜货箱卸载系统” 10 Keywords - TextRank :
倾斜 部分 货箱 具有 纵向 卸载 运输工具 最靠近 系统 变型
Process finished with exit code 0
```

PART
02

聚类算法及局限分析

聚类

聚类是机器学习解决的四大问题之一，是典型的非监督学习，将没有标签的数据划分
为不同的簇。

常见的聚类算法包括k-means, DBSCAN, 谱聚类, 层次聚类等。聚类广泛用于图像
压缩、用户画像、数据可视化、数据挖掘等。

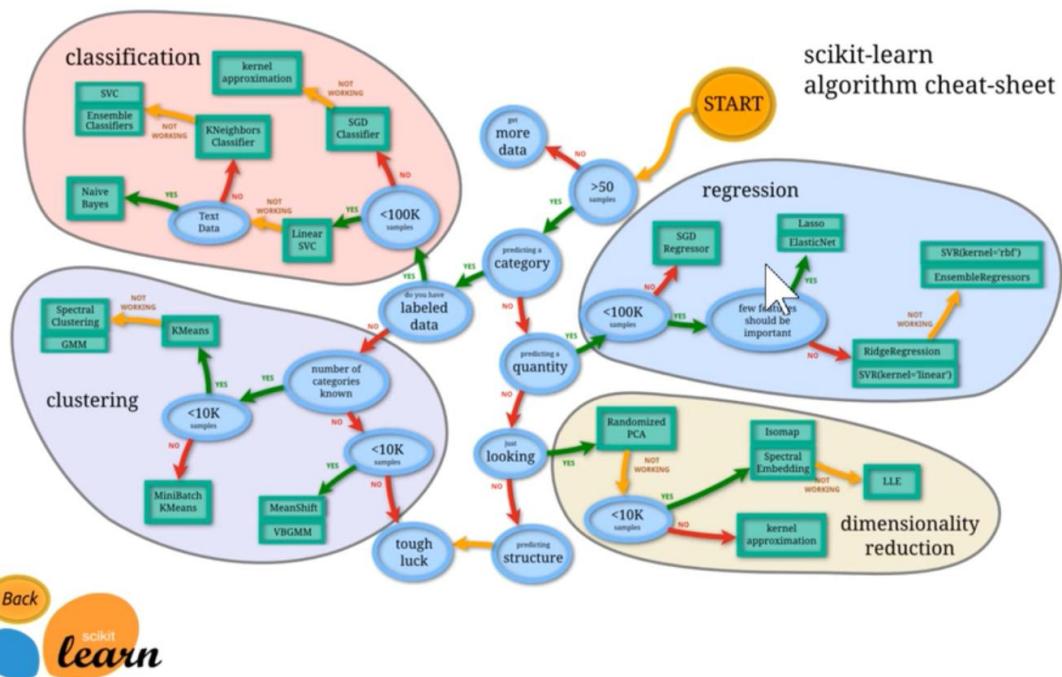


表 6-6 聚类算法类别

算法类别	包括的主要算法
划分(分裂)方法	K-Means 算法(K-平均)、K-MEDOIDS 算法(K-中心点)和 CLARANS 算法(基于选择的算法)
层次分析方法	BIRCH 算法(平衡迭代规约和聚类)、CURE 算法(代表点聚类)和 CHAMELEON 算法(动态模型)
基于密度的方法	DBSCAN 算法(基于高密度连接区域)、DENCLUE 算法(密度分布函数)和 OPTICS 算法(对象排序识别)
基于网格的方法	STING 算法(统计信息网络)、CLIQUE 算法(聚类高维空间)和 WAVE-CLUSTER 算法(小波变换)

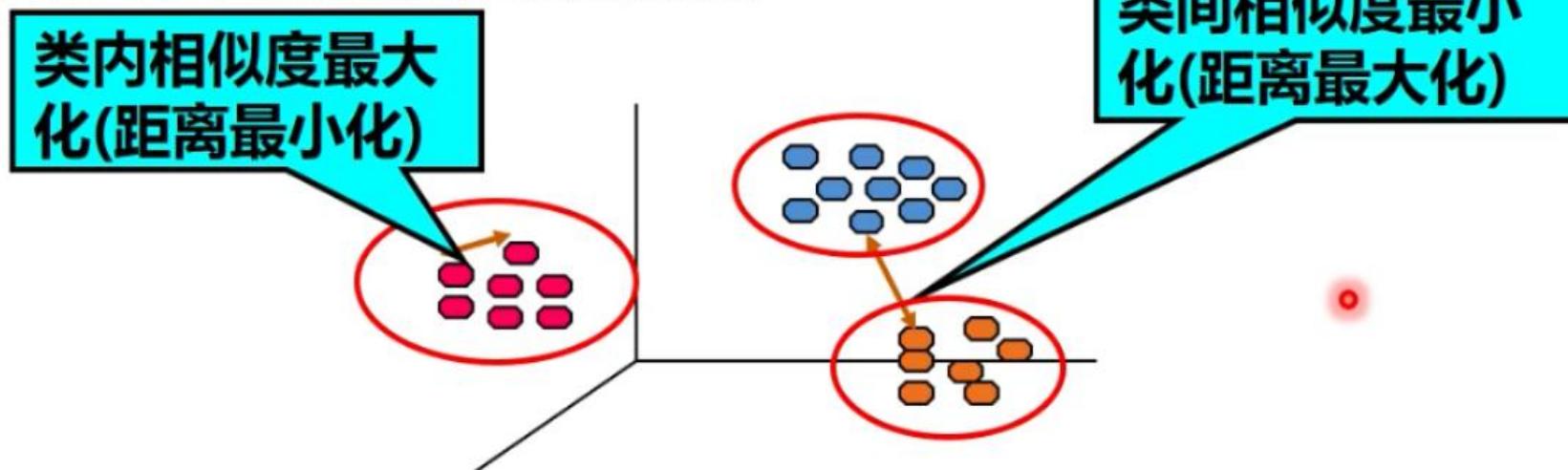
https://blog.csdn.net/qj_43691842

DBSCAN算法聚类效果最好

概述：聚类定义

“物以类聚，人以群分”。

简单地描述，聚类(Clustering)是将数据集划分为若干相似对象组成的多个组(group)或簇(cluster)的过程，使得同一组中对象间的相似度最大化，不同组中对象间的相似度最小化。或者说一个簇(cluster)就是由彼此相似的一组对象所构成的集合，不同簇中的对象通常不相似或相似度很低。



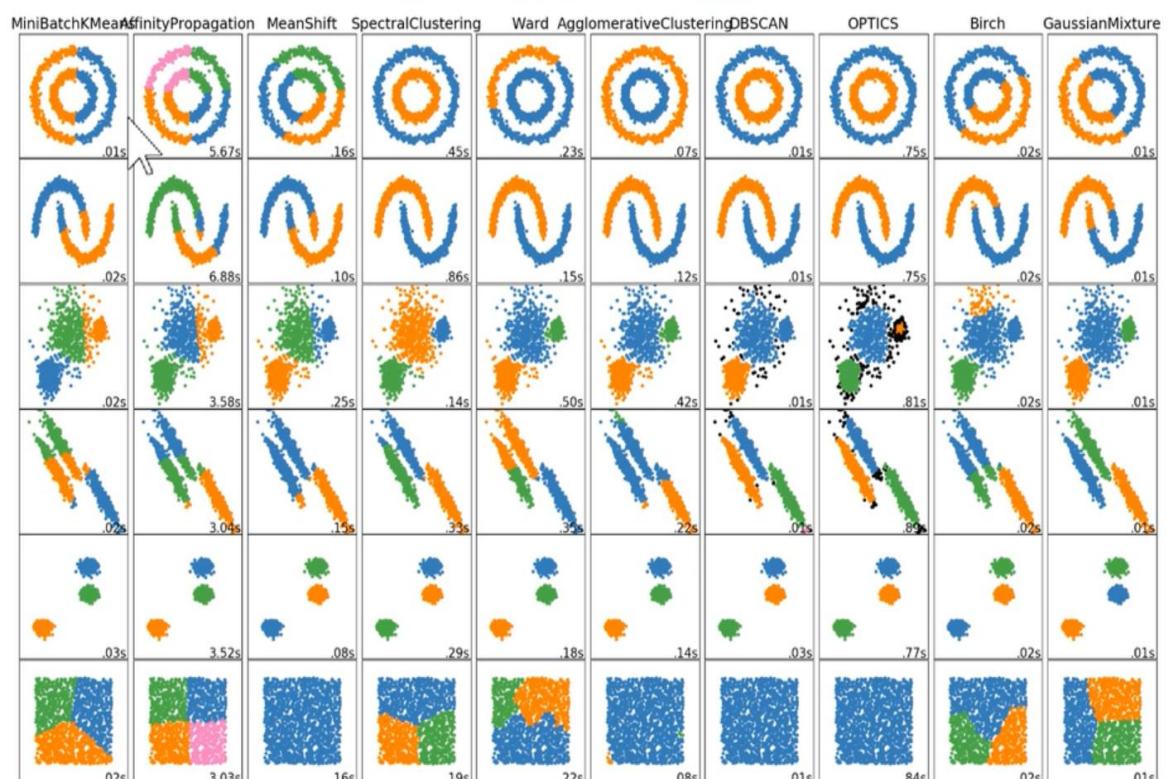
概述：聚类的理论解释

- 聚类是一种无监督的机器学习方法

- 即事先对数据集的分布没有任何的了解，它是将物理或抽象对象的集合组成为由类似的对象组成的多个类的过程。聚类方法的目的是寻找数据中：潜在的自然分组结构和感兴趣的关系。

- 聚类分析中“簇”的特征：

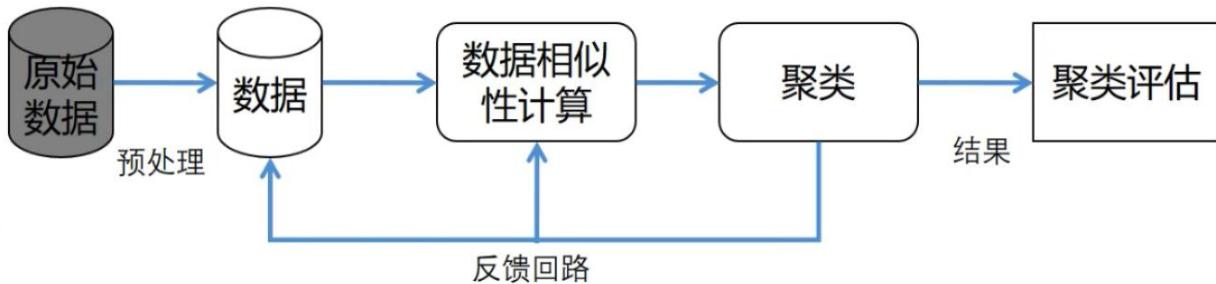
- 聚类所说的簇不是事先给定的，而是根据数据的相似性和距离来划分
- 聚的数目和结构都没有事先假定



聚类的基本流程

- 典型的数据聚类基本步骤如下：

- 对数据进行表示和预处理，包括数据清洗、特征选择或特征抽取；
- 给定数据之间的相似度或相异度及其定义方法；
- 根据相似度，对数据进行划分，即聚类；
- 对聚类结果进行评估。



- 聚类结果的表示形式

	样本1	样本2	样本3	样本4	样本5	样本6
聚类1	1	0	0	1	1	0
聚类2	0	1	0	0	1	1
聚类3	0	0	1	0	0	0

- 每个聚类至少有一个样本
- 每个样本至少属于一个聚类

聚类分析 (cluster analysis)

- + 是一个将数据集划分为若干组或类的过程，
- + 同一个组内的数据对象具有较高的相似度；而不同组中的数据对象是不相似的。
- + 相似或不相似的描述
 - ✖ 基于数据描述属性的取值。
 - ✖ 用各对象间的距离来表示。

表 6-7 cluster 提供的聚类算法及其适用范围

函数名称	参数	适用范围	距离度量
K-Means	簇数	可用于样本数目很大、聚类数目中等的场景	点之间的距离
Spectral clustering	簇数	可用于样本数目中等、聚类数目较小的场景	图距离
Ward hierarchical clustering	簇数	可用于样本数目较大、聚类数目较大的场景	点之间的距离
Agglomerative clustering	簇数、链接类型、距离	可用于样本数目较大、聚类数目较大的场景	任意成对点线图间的距离
DBSCAN	半径大小、最低成员数目	可用于样本数目很大、聚类数目中等的场景	最近的点之间的距离
Birch	分支因子、阈值、可选全局集群	可用于样本数目很大、聚类数目较大的场景	点之间的欧式距离

【概述】

DBSCAN是Density-Based Spatial Clustering of Applications with Noise的缩写，是一种简单有效的基于密度的聚类算法。

基于密度的聚类旨在检测高密度的区域，这些区域由低密度的区域相互分开

【算法原理】

DBSCAN聚类算法是基于密度的聚类。这种算法假定类别可以通过样本分布的紧密程度来决定。同一类别的样本，它们是比较紧密的，也就是说，对于属于一个类别的样本，在这个样本的不远处很大可能有同一类别的样本。

应用DBSCAN算法时，我们需要估计数据集中特定点的密度，特定点的密度是通过计算该点在指定半径下数据点个数（包括特定点），这种计算得到的某个点的密度也被称为局部密度。

计算数据集中每个点的密度时，我们需要把每个点归为以下三类：

1. 如果点的局部密度大于某个阈值，称这个点为核心点。
2. 如果点的局部密度小于某个阈值，但是它落在核心点的邻域内，称这个点为边界点。
3. 如果点不属于核心点且不属于噪声点，称点为噪声点。

除了标记数据集中每个点的类别，我们要做的是根据类别将每个样本进行聚类。对于同一个还未分配的核心点，我们将它邻域内的所有点归为一个新的类 C_{new} 。如果邻域内有其他核心点的话，我们将重复上面相同情况的动作。

举个例子，下图核心点1邻域内的点归为类 C_{new} ，因为邻域包括核心点2，那么核心点2邻域内的未分配类别的也归类为 C_{new} ，核心点2邻域包含核心点3，核心点3也进行与核心点2相同的操作，依次类推（图中绿箭头标识的过程）直到核心点邻域内不包含新的核心点。

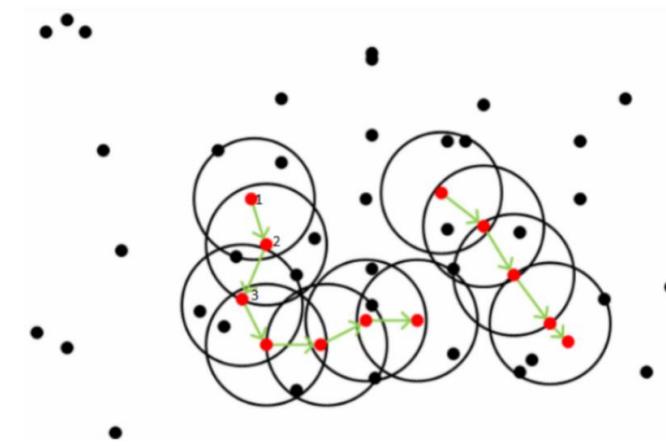
和K-Means，
BIRCH这些一般只
适用于凸样本集的
聚类相比，

DBSCAN既可
以适用于凸样本集，
也可以适用于非凸
样本集。

DBSCAN算法
的显著优点能够有
效处理噪声点和发
现任意形状的空间
聚类。

【算法流程】

- 如果所有点已经处理，停止
- 对于以前没有处理的特定点，检查它是否是核心点
- 如果不是核心点
 - 将其标记为噪声点
- 如果是核心点，将其标记并
 - 使用这一点形成一个新的聚类 C_{new} ，并包括集群内的邻域内或边界上的所有点。
 - 将所有这些在邻域内的点插入队列中。
 - 当队列不为空
 - 从队列中删除一个点
 - 如果这个点不是核心点，则将其标记为边界点
 - 如果这个点是核心点，则标记它并检查其邻居中以前没有分配给类的每个点。对于每一未分配的相邻点
 - 将该点分配给当前类 C_{new}
 - 将该点插入队列中



【代码实现】

- 根据上面的算法流程，我们需求求出一个点的邻域内所有的点，目的是判断这点是否为核心点以及处理核心点邻域内的点。

```
def neighbor_points(data, pointId, radius):
    """
    得到邻域内所有样本点的Id
    :param data: 样本点
    :param pointId: 核心点
    :param radius: 半径
    :return: 邻域内所用样本Id
    """
    points = []
    for i in range(len(data)):
        if dist(data[i, 0: 2], data[pointId, 0: 2]) < radius:
            points.append(i)
    return np.asarray(points)
```

- 对于一个核心点，我们需要将它和它邻域内所有未分配的样本点分配给一个新类。若邻域内有其他核心点，重复上一个步骤，但只处理邻域内未分配的点，

```

def to_cluster(data, clusterRes, pointId, clusterId, radius, minPts):
    """
    判断一个点是否是核心点，若是则将它和它邻域内的所用未分配的样本点分配给一个新类
    若邻域内有其他核心点，重复上一个步骤，但只处理邻域内未分配的点，并且仍然是上一个步骤的类。
    :param data: 样本集合
    :param clusterRes: 聚类结果
    :param pointId: 样本Id
    :param clusterId: 类Id
    :param radius: 半径
    :param minPts: 最小局部密度
    :return: 返回是否能将点PointId分配给一个类
    """

    points = neighbor_points(data, pointId, radius)
    points = points.tolist()

    q = queue.Queue()

    if len(points) < minPts:
        clusterRes[pointId] = NOISE
        return False
    else:
        clusterRes[pointId] = clusterId
        for point in points:
            q.put(point)
            if clusterRes[point] == UNASSIGNED:
                clusterRes[point] = clusterId

    while not q.empty():
        neighborRes = neighbor_points(data, q.get(), radius)
        if len(neighborRes) > minPts: # 核心点
            for i in range(len(neighborRes)):
                resultPoint = neighborRes[i]
                if clusterRes[resultPoint] == UNASSIGNED:
                    q.put(resultPoint)
                    clusterRes[resultPoint] = clusterId
                elif clusterRes[resultPoint] == NOISE:
                    clusterRes[resultPoint] = clusterId

    return True

```

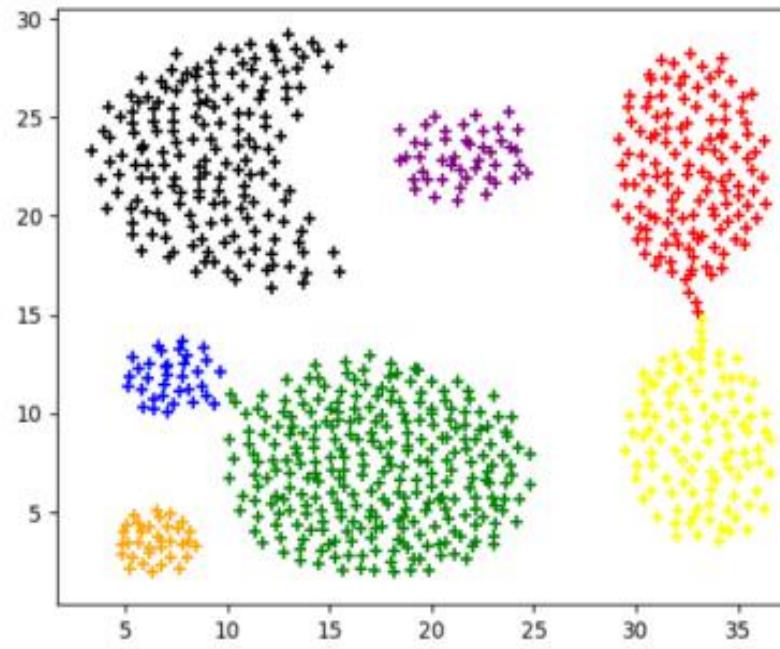
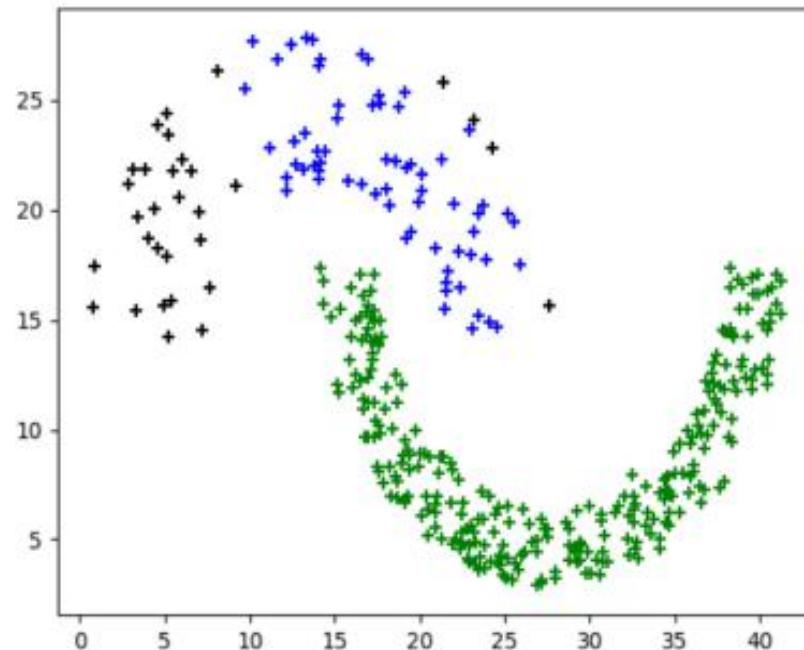
3. 扫描整个数据集，为每个数据集打上核心点、边界点和噪声点标签的同时为样本聚类。

```

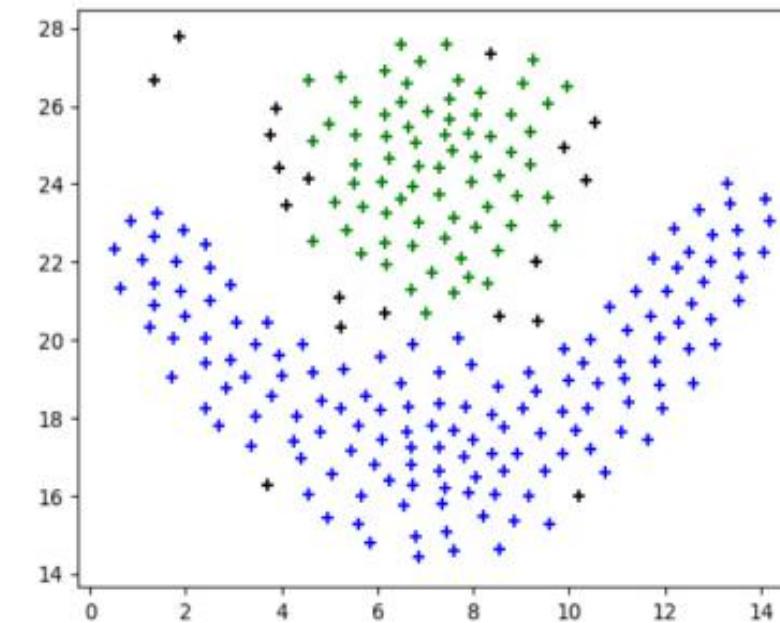
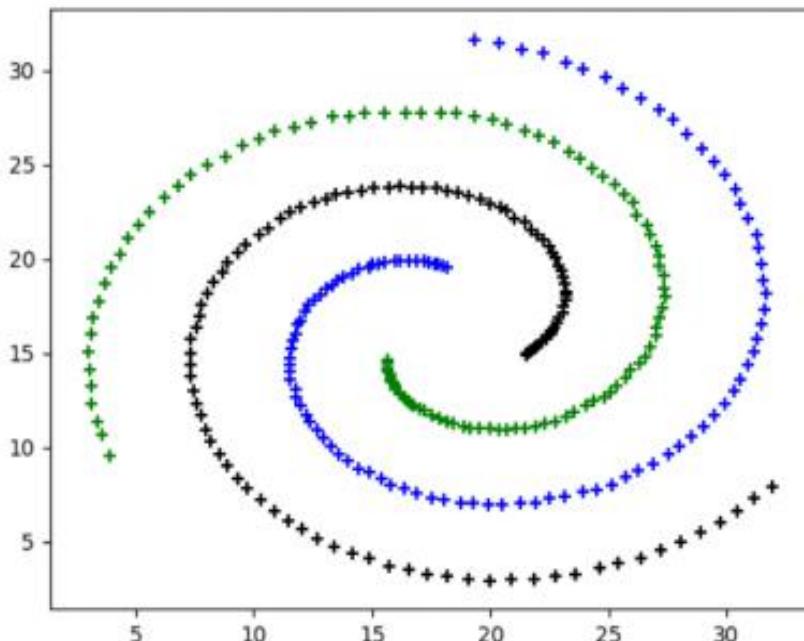
def dbscan(data, radius, minPts):
    """
    扫描整个数据集，为每个数据集打上核心点，边界点和噪声点标签的同时为
    样本集聚类
    :param data: 样本集
    :param radius: 半径
    :param minPts: 最小局部密度
    :return: 返回聚类结果，类id集合
    """

    clusterId = 0
    nPoints = len(data)
    clusterRes = [UNASSIGNED] * nPoints
    for pointId in range(nPoints):
        if clusterRes[pointId] == UNASSIGNED:
            if to_cluster(data, clusterRes, pointId, clusterId, radius, minPts):
                clusterId = clusterId + 1
    return np.asarray(clusterRes), clusterId

```



从左图聚类结果我们可以看出，DBSCAN对空间中任意形状的聚类簇都有比较好的聚类效果。除了有比较好的聚类效果之外，在实验过程中，相比于K-means，DBSCAN不需要确定聚类的个数。



图一展示了DBSCAN聚类算法的不足之处：各个簇之间密度分布不均匀，而且簇之间相距不大时，由于参数半径和局部密度阈值选取困难，导致聚类效果比较差。

Kemans 算法

【算法原理】

Kemans算法是一种无监督算法，用于将相似的样本归为一个类中。对于给定包含N个样本的数据集 $\{x_1, x_2, \dots, x_N\}$, K-means算法的目标是将样本分配到k个类中。K-means算法需要假定有K个中心点作为每个类的代表，目标函数如下：

$$\min_{\mu_k, c_{ik}} \sum_{i=1}^N \sum_{k=1}^K c_{ik} \|x_i - \mu_k\|_2^2$$

$$\text{s.t. } c_{ik} \in \{0, 1\}, \sum_{k=1}^K c_{ik} = 1,$$

其中， μ_k 是k个类的中心， c_{ik} 代表第i个样本是否属于第k个类。

k-means的算法流程是寻找合适中心点并将样本分配到k个中心点的过程。K-means算法需要初始化k个中心点并将样本分配到离它最近的中心点，然后在新的聚类中重新计算各个聚类的中心点，这样不断更新k个中心点直到收敛。

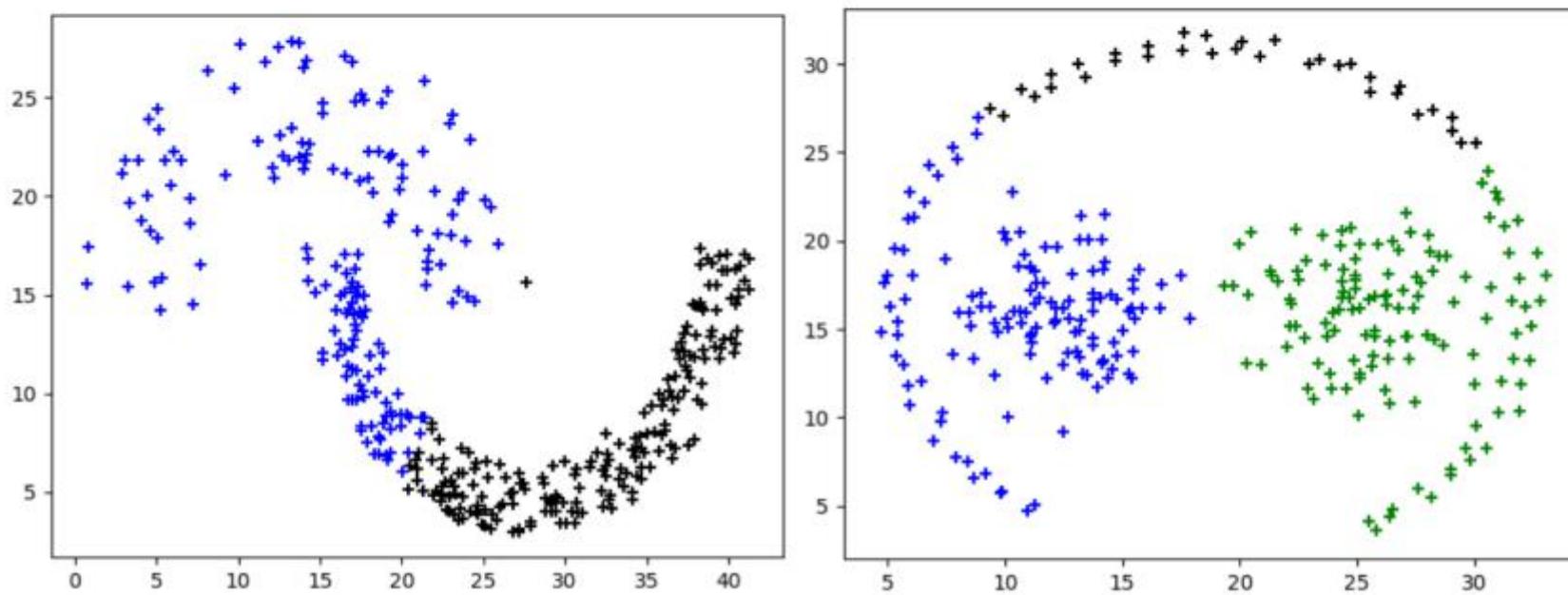
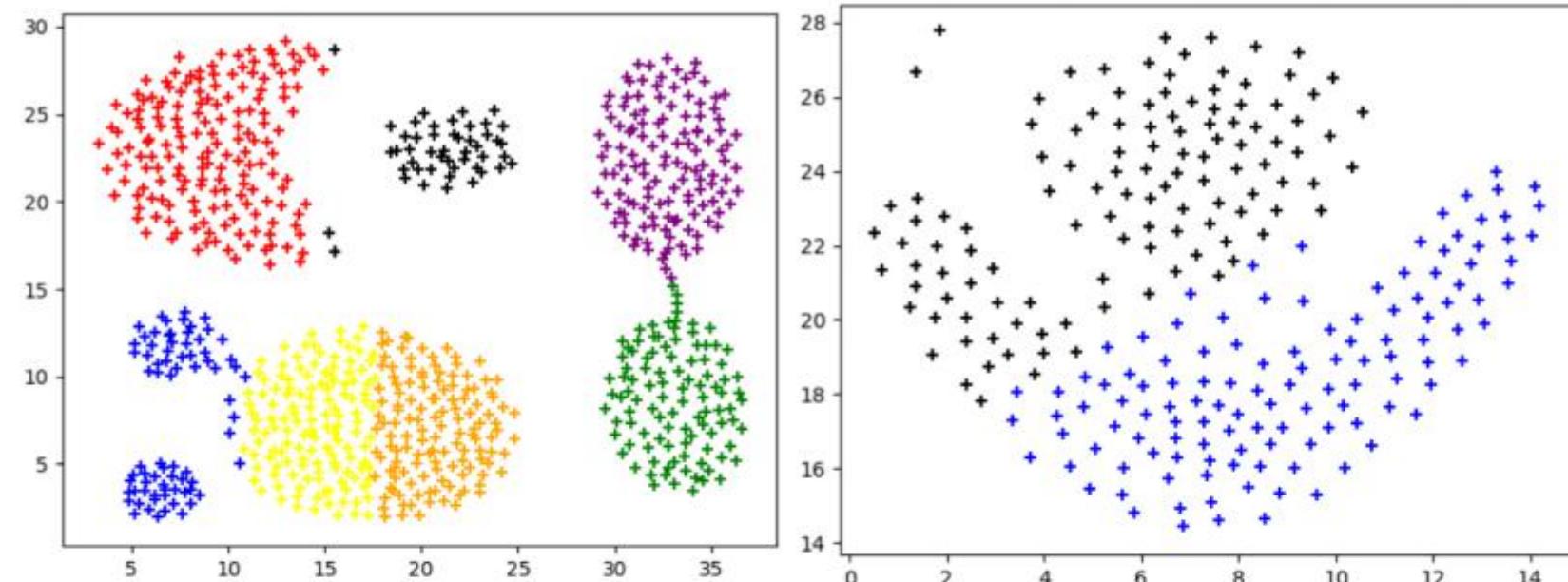
K-means算法缺陷

通过右侧结果展示,我们可以看到K-means算法有很多明显的缺陷:

1. K-means算法在多种不同情况下的聚类表现得不太好,我们可以看到K-means得到的簇更偏向于球形,这意味着K-means算法不能处理非球形簇的聚类问题,而现实中数据的分布情况是十分复杂的,所以K-means 算法不太适用于现实大多数情况.

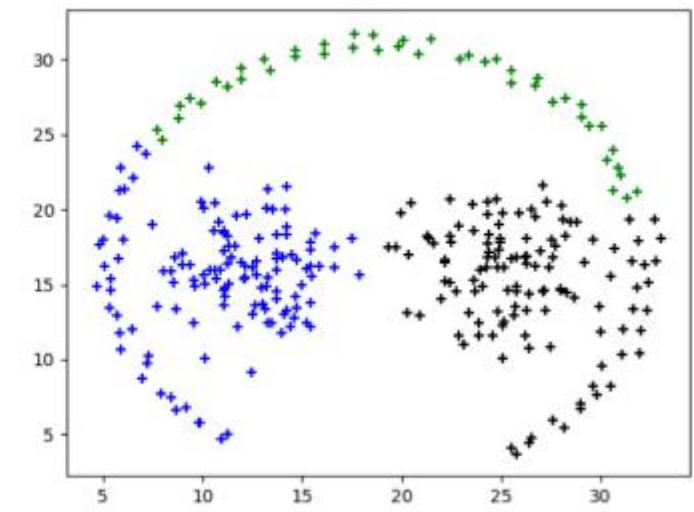
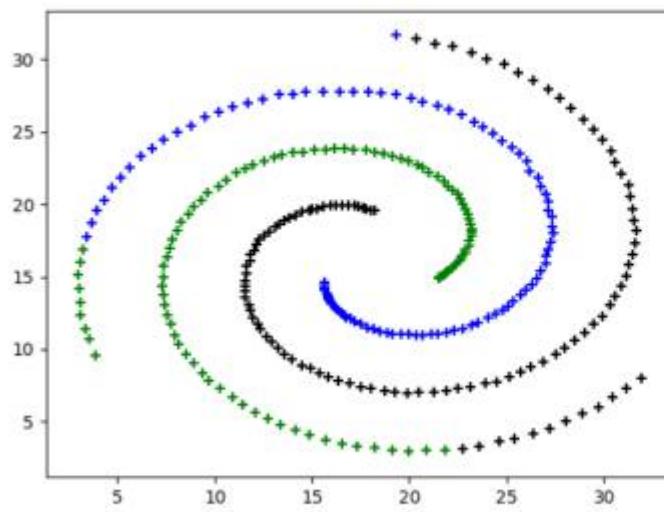
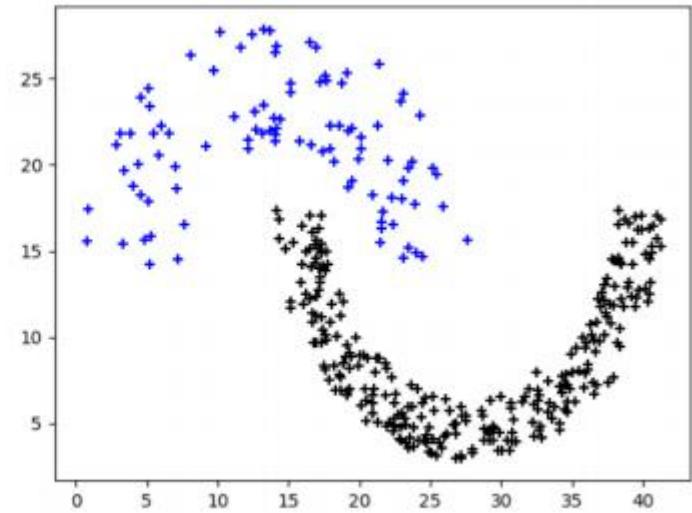
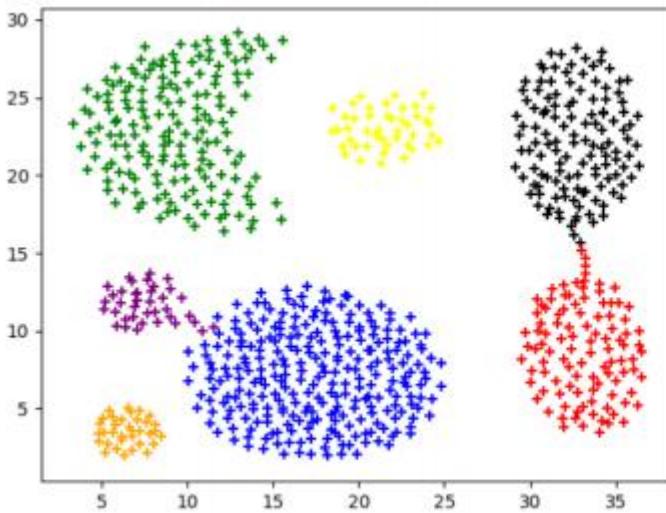
2. K-means算法需要预先确定聚类的个数.

3. K-means算法对初始选取的聚类中心点敏感.我们可以看到在 Aggression数据聚类中,K-means 算法会把不同簇聚合在一起形成足球形状簇的聚类,而这种情况下得到的中心点在两个簇中间.这说明此时K-means陷入了一个局部最优解,而陷入局部最优的一个原因是初始化中心点的位置不太好.



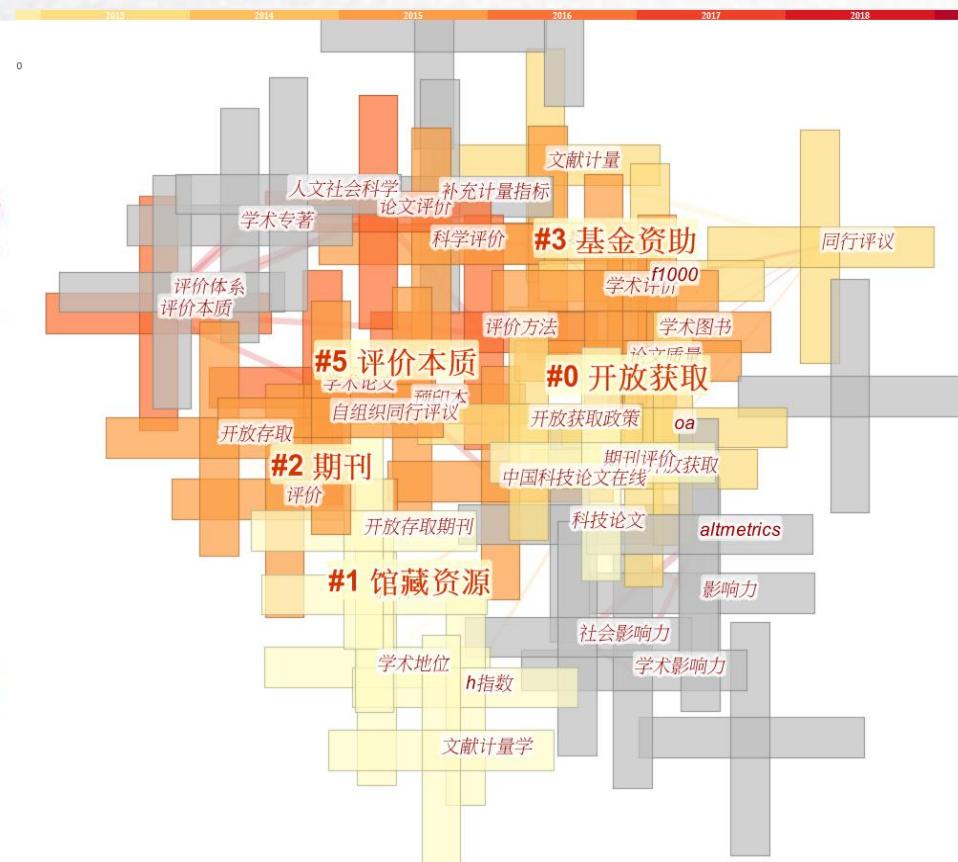
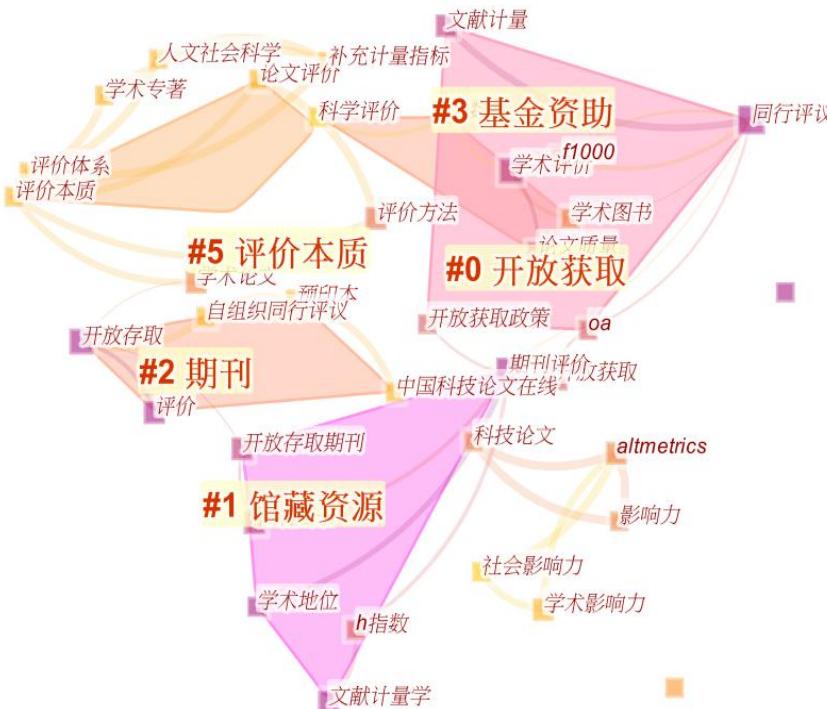
谱聚类

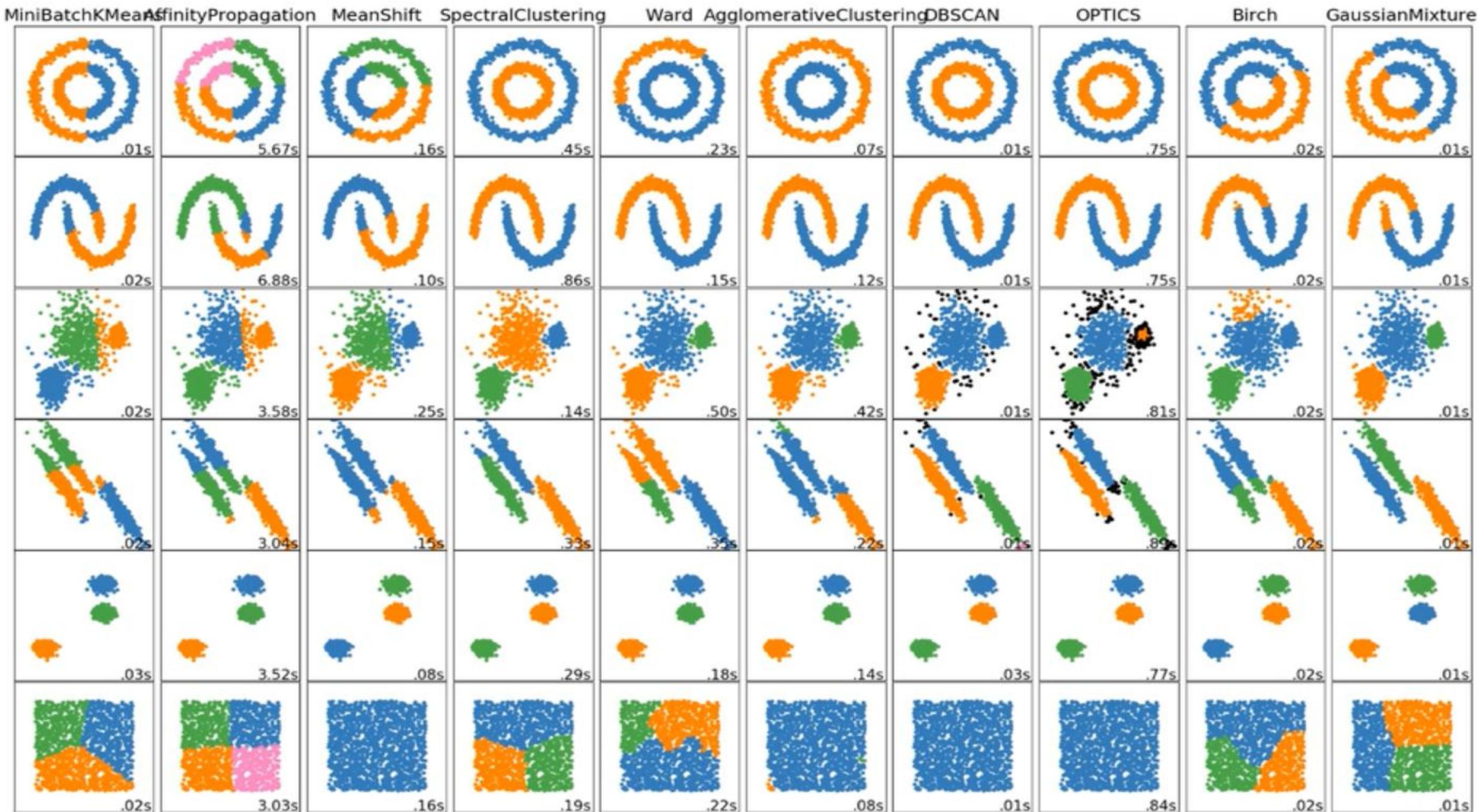
谱聚类是一种基于图论的聚类方法，它将样本看作无向图中的结点，根据样本之间的相似度构建边，然后根据图内点相似度将图分为多个子图，使子图内部的点相似度最高，子图之间点的相似度最低。



谱聚类算法

citespace自动聚类的实现是依据谱聚类算法，谱聚类本身就是基于图论的一种算法，因此它对共引网络这种基于链接关系而不是节点属性的聚类具有天然的优势。传统的聚类算法，如K均值算法，EM算法等都是建立在凸球形的样本空间上，但样本空间不为凸时，算法会陷入局部最优。谱聚类算法正是为了弥补上述算法的这一缺陷而产生的。





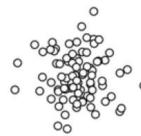
> 对于高维数据效果差

传统的聚类算法已经比较成功的解决了低维数据的聚类问题。但是由于实际应用中数据的复杂性，在处理许多问题时，现有的算法经常失效，特别是对于高维数据和大型数据的情况。因为传统聚类方法在高维数据集中进行聚类时，主要遇到两个问题。

- ①高维数据集中存在大量无关的属性使得在所有维中存在簇的可能性几乎为零；
- ②高维空间中数据较低维空间中数据分布要稀疏，其中数据间距离几乎相等是普遍现象，而传统聚类方法是基于距离进行聚类的，因此在高维空间中无法基于距离来构建簇。

PART
03

演示聚类效果



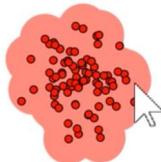
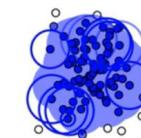
用网页简单讲解

Visualizing K-Means Clustering

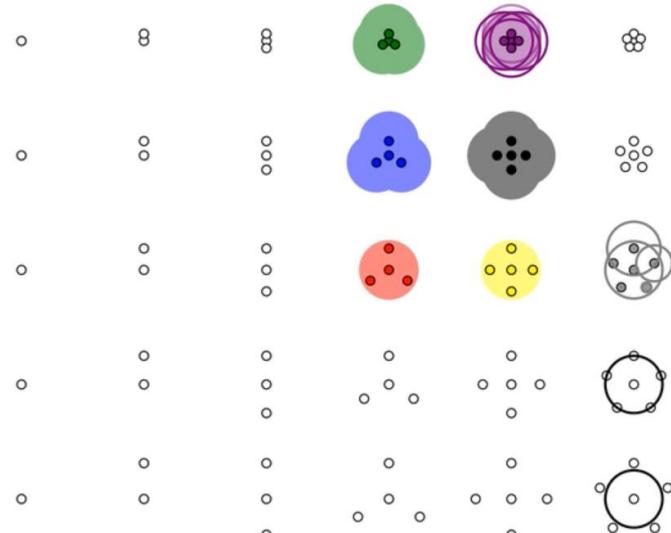
January 19, 2014

Suppose you plotted the screen width and height of all the devices accessing this website. You'd probably find that the points form three clumps: one clump with small dimensions, (smartphones), one with moderate dimensions, (tablets), and one with large dimensions, (laptops and desktops). Getting an algorithm to recognize these clumps of points without help is called *clustering*. To gain insight into how common clustering techniques work (and don't work), I've been making some visualizations that illustrate three fundamentally different approaches. This post, the first in this series of three, covers the k-means algorithm. To begin, click an initialization strategy below:

epsilon = 1.00
minPoints = 4



How to pick the initial centroids?



I'll Choose

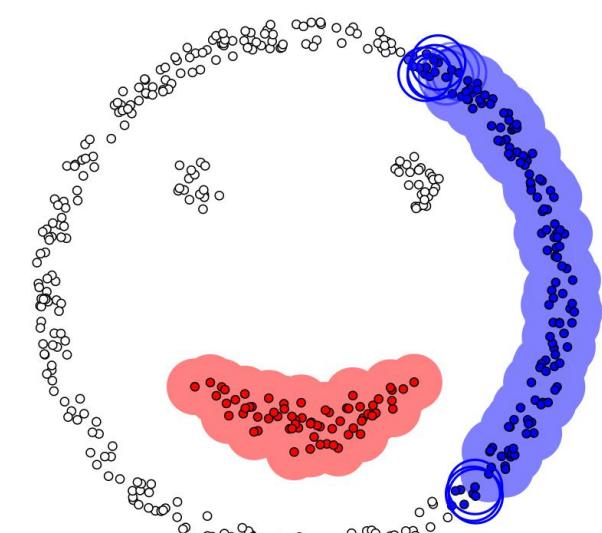
Randomly

K-means聚类和
DBSCAN聚类的可视化
动态过程及原理

epsilon = 1.00
minPoints = 4

Restart

Pause





THANKS

谢谢观赏 感谢聆听