◂ BIT ▸

文本聚类
大数据分析与应用

小组成员：何天阳 3220190944　袁金海 3220190994
刘百宇 3220190963　裴明哲 3220190971
齐若岩 3220190857

指导教师：张华平

时间：2019-11-28

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

德以明理　学以精工

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

目 录

CONTENTS

1 文本聚类概述

文本聚类

文本聚类主要是依据著名的聚类假设：同类的文档相似度较大，而不同类的文档相似度较小。作为一种无监督的机器学习方法，聚类由于不需要训练过程，以及不需要预先对文档手工标注类别，因此具有一定的灵活性和较高的自动化处理能力，已经成为对文本信息进行有效地组织、摘要和导航的重要手段，为越来越多的研究人员所关注。

应用领域

- 文档聚类可以作为多文档自动文摘等自然语言处理应用的预处理步骤
- 对搜索引擎返回的结果进行聚类，使用户迅速定位到所需要的信息
- 对用户感兴趣的文档（如用户浏览器cache中的网页）聚类，从而发现用户的兴趣模式并用于信息过滤和信息主动推荐等服务
- 聚类技术还可以用来改善文本分类的结果
- 数字图书馆服务
- 用于大数据中热点话题或事件的发现

文本聚类方法

作为NLP领域最经典的使用场景之一，文本聚类形成了它的一般流程，并积累了许多的实现方法。

- 文本预处理
- 构建词袋空间VSM
- TF-IDF构建词权重
- 使用聚类算法进行聚类

文本预处理

切词，去除停用词等操作

构建词袋空间VSM Vector Space Model

构建词袋空间的步骤如下：

- 将所有文档读入到程序中，再将每个文档切词。
- 去除每个文档中的停用词。
- 统计所有文档的词集合。
- 对每个文档，都将构建一个向量，向量的值是对应词语在本文档中出现的次数

TF-IDF构建词权重

只用词频进行数值化明显是不够的。因此，使用TF-IDF来**度量每个词的重要程度**。

使用聚类算法进行聚类

使用**聚类算法**进行文本聚类，如KMeans，层次聚类，DBSCAN等

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

2 文本预处理

文本预处理

英文文本挖掘预处理一般可以不做分词（特殊需求除外），而中文预处理分词是必不可少的一步，在这里我们使用jieba分词对文本进行中文分词

大部分英文文本都是uft-8的编码，这样在大多数时候处理的时候不用考虑编码转换的问题，而中文文本处理必须要处理unicode的编码问题

进行分词后，为节省存储空间和提高搜索效率，还需要参照中文停用词表，对文本进行去除停用词处理，比如“的”、“是”、“而且”、“但是”、”非常“等

获取方法一般有两种：使用别人做好的语料库、自己用爬虫去在网上去爬自己的语料数据

方法一

用的文本语料库在网上有很多，如果只是学习，则可以直接下载下来使用，但如果是某些特殊主题的语料库，比如“deep learning”相关的语料库，则这种方法行不通，需要用第二种方法去获取。

方法二

开源工具有很多，例如beautifulsoup。但如果需要某些特殊的语料数据，比如“机器学习”相关的语料库，则需要用主题爬虫来完成。ache允许我们用关键字或者一个分类算法模型来过滤出我们需要的主题语料，比较强大。

由于爬下来的内容中有很多html的一些标签，需要去掉。少量的非文本内容的可以直接用Python的正则表达式(re)删除，复杂的则可以用beautifulsoup来去除。

分词工具很多，包括盘古分词、Yaha分词、Jieba分词、清华THULAC等。这里我们使用**Jieba分词**对文本进行预处理。

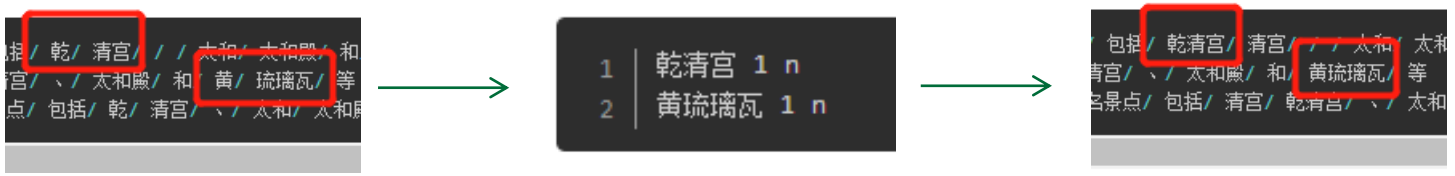
- 首先使用pip install jieba命令安装Jieba分词

```
pip install jieba
```

- Jieba分词简单易用，在这里我们对其使用的词图扫描，动态规划，Viterbi算法等不作深入介绍
- Jieba中文分词支持的三种分词模式包括：精确模式； 全模式；搜索引擎模式。

```
[全模式]:  我/ 来到/ 北京/ 清华/ 清华大学/ 华大/ 大学
[精确模式]:  我/ 来到/ 北京/ 清华大学
[默认模式]:  我/ 来到/ 北京/ 清华大学
[搜索引擎模式]:  我/ 来到/ 北京/ 清华/ 华大/ 大学/ 清华大学
```

- 对于一些专有名词，Jieba分词工具无法识别出，需要用户自定义词典



2 去除停用词

在信息检索中，为节省存储空间和提高搜索效率，在处理自然语言数据（或文本）之前或之后会自动过滤掉某些字或词，比如“的”、“是”、“而且”、“但是”、”非常“等。这些字或词即被称为Stop Words（**停用词**）。

这里我们参考了两个比较常见的中文停用词整理表

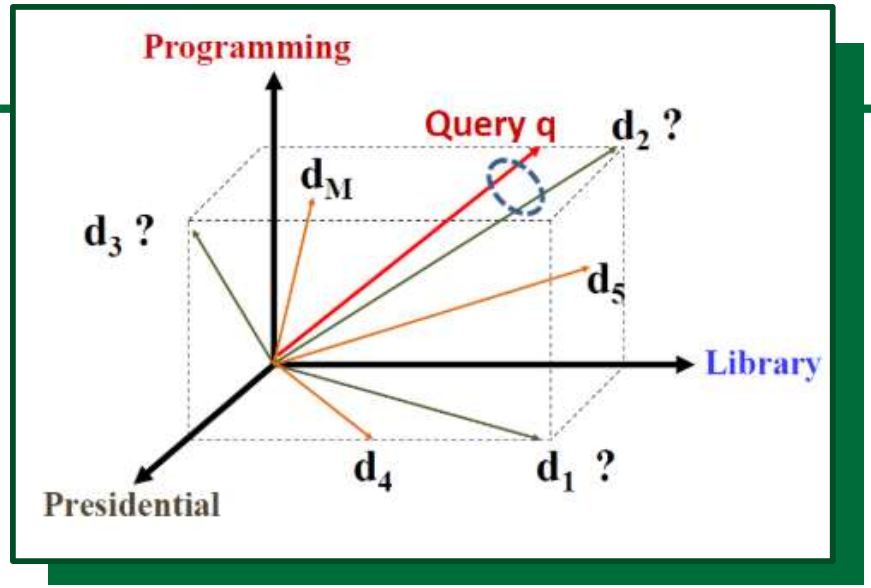
1. 中文停用词表(比较全面,有1208个停用词)
2. 最全中文停用词表整理（1893个）

经过去除停用词处理后的效果如下图所示

```
# 去除停用词
stopwords = {}.fromkeys(['的', '包括', '等', '是'])
text = "故宫的著名景点包括乾清宫、太和殿和午门等。其中乾清宫非常精美，午门是紫禁城的正门。"
```

```
2 故宫/ 著名景点/ 乾/ 清宫/ 、/ 太和殿/ 和/ 午门/ 。/ 其中/ 乾/ 清宫/ 非常/ 精美/ ，/ 午门/ 紫禁城/ 正
```

3 构建词袋空间VSM



什么是VSM

向量空间模型(VSM，Vector Space Model)将文本内容转换为易于数学处理的向量形式，并表示为多维空间中的一个点，把对文本内容的处理简化为向量空间中向量运算，使问题的复杂度大为降低使得各种相似计算和排序成为可能。

VSM模型

在向量空间模型中，文本空间被看作是由一组正交词条矢量所组成的矢量空间，每篇文本d表示为其中的一个范化矢量 V(d)=(t1,w1(d),…,tn,wn(d))，其中ti为词条项，wi(d) 表示词条ti 在文本d 中的权值，用于显示向量ti在文本d中的**重要程度**。

可以将文本d 中出现的所有词条作为t，也可以要求ti是d中出现的所有短语，从而提高内容特征表示的准确性。wi(d) 一般被定义为词条ti在文本d中的出现频率tfi(d)的函数，wi=f(tfi(d))，常用的f函数有布尔函数、平方根函数、对数函数、**TF-IDF函数**等。

文本经过分词程序后，首先去除停用词，合并数字和人名等词汇，然后统计词频，最终表示为一个向量。

4 TF-IDF构建词权重

词频(TF)

TF-IDF函数用来表示特征项的重要程度，与特征项的重要性相关的两个重要因素是词频 **TF(文本内频率)**和**逆文本频数IDF**(inverse documentfrequency)。

词频(TF)：即一个特征项在某一文档中出现的次数，反映了某一个特征项对该文本的重要性，其定义如下所示

$TF_{ij} = freq_{ij}$= 特征词 t_i 在文档d_j 中的频率

可利用对数降低词频对TF取值的影响，从而减少了少数高频词对特征权重计算的影响，如下所示：

$$TF_{ij} = \log(freq_{ij}) + 1$$

倒文档频度(IDF)

倒文档频度(IDF)：这一分量反映了某一特征项区别于其他文档的程度，是一个关键词在整个数据全局中重要性的全局性统计特征，称为倒文档频度。如果一个词在整个数据全集中出现的频度很小，则它应该是反映包含该类词的文档内容的重要词汇。因此，一个关键词的权重应该与该词所在的文档的总数成反比或近似反比的关系。

$$IDF_i = log(n/n_i) = log(\text{全集中文档的总数} / \text{含关键词}i\text{的文档总数})$$

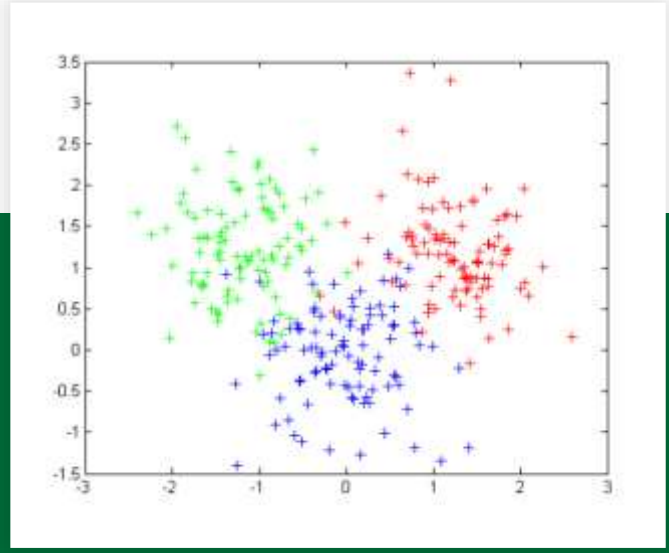
其中：n 为全部训练样本数，ni 为出现特征词 ti 的训练样本数。它反映了某一特征项在分类过程中对某一类的区分度。

TF-IDF

TF-IDF公式认为对区别文档最有意义的特征项应该是那些在一类文档中出现频率足够高，而在文档集合的其他文档中出现频率足够小的词语，所以引入了逆文本频度IDF的概念，并以TF和IDF的乘积作为特征空间坐标系的取值测度。设TFij为特征词ti在文档dj中的频度；n为全部训练样本数，ni为出现特征词ti的训练样本数，TF-IDF公式如下所示：

$$weight(T_{ij}) = TF_{ij} \times IDF_i = freq_{ij} \times log(n/n_i)$$

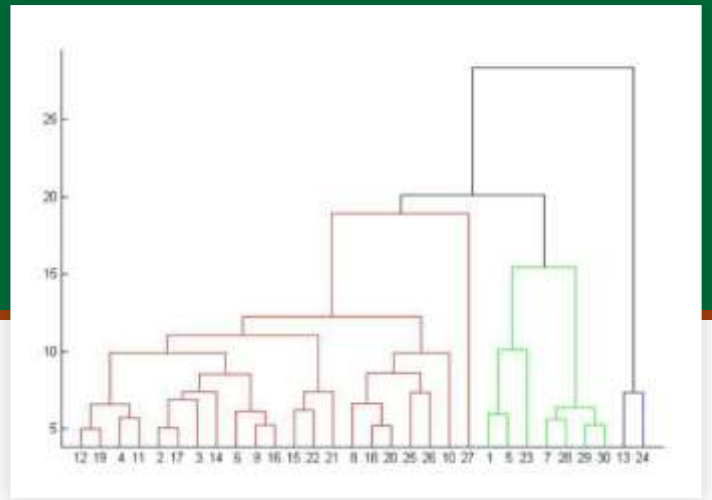
5 聚类算法



聚类算法是什么 ▶

给定n个训练样本（未标记），如：{X1,X2,...,Xn}，同时给定聚类的个数K

目标：把比较“接近”的样本放到一个簇类（cluster）里，总共得到K个簇类（cluster）。



常用的距离计算公式

1 **欧氏距离**

$$d(\mathbf{x},\mathbf{z}) = ||\mathbf{x}-\mathbf{z}|| = \sqrt{\sum_{d=1}^{D}(x_d - z_d)^2}$$

2 **曼哈顿距离**

$$d(\mathbf{x},\mathbf{z}) = \sum_{d=1}^{D}|x_d - z_d|$$

3 **核函数映射后距离**

$$d(\mathbf{x},\mathbf{z}) = ||\phi(\mathbf{x}) - \phi(\mathbf{z})||$$

如何评价聚类好坏

类间距高，类内距低，通俗地说就是“**抱团紧不紧，异族远不远**”。

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

概念解读

已知观测集(x_{1},x_{2},...,x_{n})，其中每个观测都是一个D-维实向量，k-平均聚类要把这n个观测划分到k个集合中(k≤n),使得组内平方和（WCSS within-cluster sum of squares）最小。换句话说，它的目标是

找到使得下式满足的聚类S_i

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}\in S_i} \|\mathbf{x}-\boldsymbol{\mu}_i\|^2$$

K-means算法原理

输入：

input1：N个样本$\{x_1, x_2, .., x_N\} x_n \in R^D$，即$x$是D维向量

input2：拟定的聚类个数K

初始化：

- 随机初始化K个D维的向量
- 选取K个不同的样本点作为初始聚类中心

迭代直至收敛：

- 对于每个样本x_n都将其指定为离其最近的聚类中心的cluster

$$C_k = \{n: k = argmin|x_n - \mu_k|^2\}$$

- 重新计算聚类中心

$$\mu_k = \frac{1}{|C_k|} \sum_{n \in C_k} x_n$$

迭代收敛定义

（1）聚类中心不在有变化

（2）每个样本到对应聚类中心的距离之和不再有很大变化

损失函数

假定为K个聚类中心，用表示是否属于聚类K，则损失函数定义如下：

$$J(\mu, r) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} |x_n - \mu_k|^2$$

K-means优缺点

优点

（1）计算复杂度低，为O(Nmq)，其中N是数据总量，m是类别（即k），q是迭代次数

（2）原理简单，实现容易，容易解释

（3）聚类效果不错

缺点

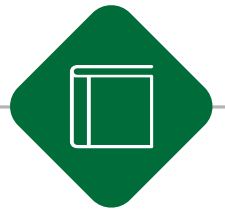
（1）对异常值（噪声）敏感，可以通过一些调整（如中心值不直接取均值，而是找均值最近的样本点代替）

（2）需要提前确定K值（提前确定多少类）

（3）分类结果依赖于分类中心的初始化（可以通过进行多次K-means取最优来解决）

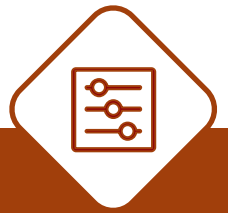
层次聚类

K-means一个最大的限制是，需要实现知道K值，即知道多少个分类。层次聚类是一种不需要确定K值就可以分类的聚类算法。
层次聚类分为**凝聚式层次聚类**和**分裂式层次聚类**。



凝聚式层次聚类

在初始阶段将每一个点都视为一个簇，之后每一次合并两个最接近的簇



分裂式层次聚类

在初始阶段将所有的点视为一个簇，之后每次分裂出一个簇，直到最后剩下单个点的簇为止

簇之间的距离定义

凝聚式聚类簇之间的距离定义有如下三种

1 **最小连接距离法（Single Linkage）**

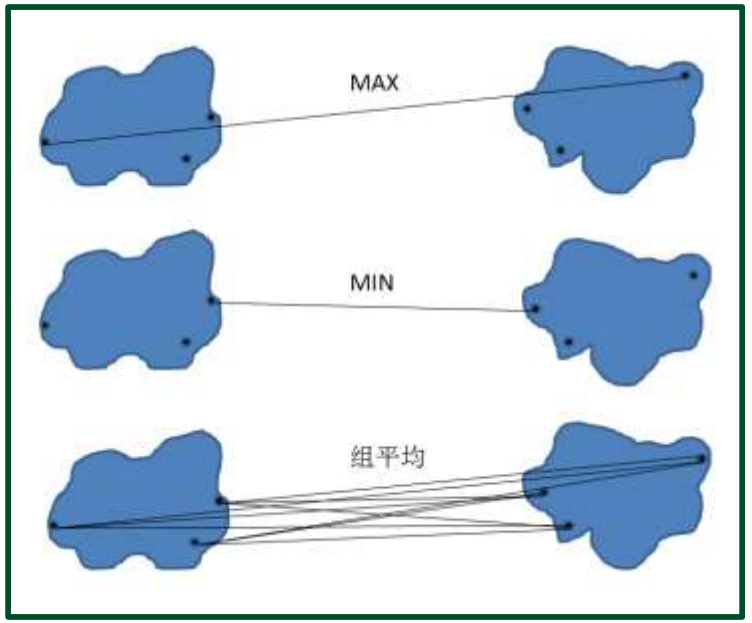
取两个簇中距离最近的两个样本的距离作为这两个簇的距离

$$d(R,S)=\min_{x_R\in R,x_S\in S} d(\mathbf{x}_R,\mathbf{x}_S)$$

2 **最大连接距离法（Complete Linkage）**

取两个簇中距离最远的两个点的距离作为这两个簇的距离

$$d(R,S)=\max_{x_R\in R,x_S\in S} d(\mathbf{x}_R,\mathbf{x}_S)$$



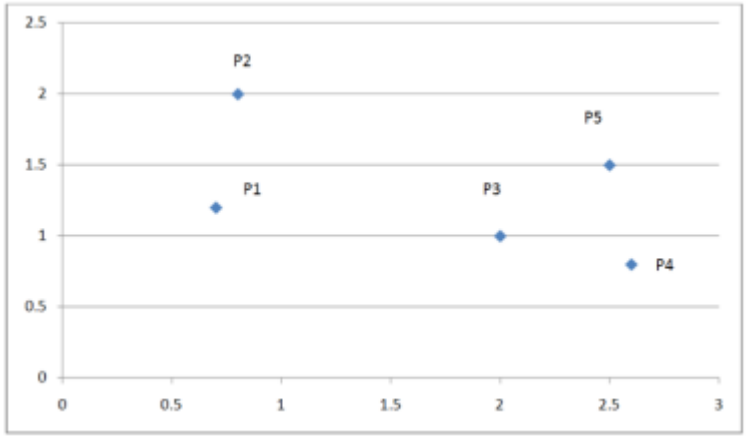
3 **平均连接距离法（Average Linkage）**

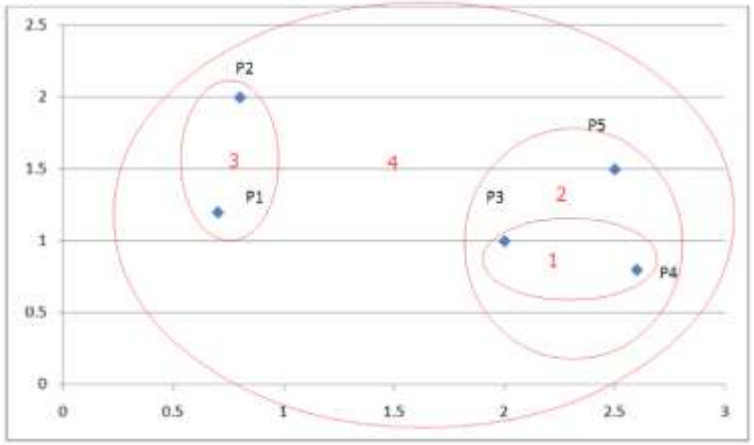
把两个簇中的点的两两的距离全部放在一起求均值，取得的结果也相对合适一点

$$d(R,S)=\frac{1}{|R||S|}\sum_{x_R\in R,x_S\in S} d(\mathbf{x}_R,\mathbf{x}_S)$$

层次聚类算法原理

1. 初始化，将每个样本都视为一个簇
2. 计算各个簇之间的距离
3. 寻找最近的两个簇，并将他们合并为一类
4. 重复步骤（2）和步骤（3），直到所有样本归为一类

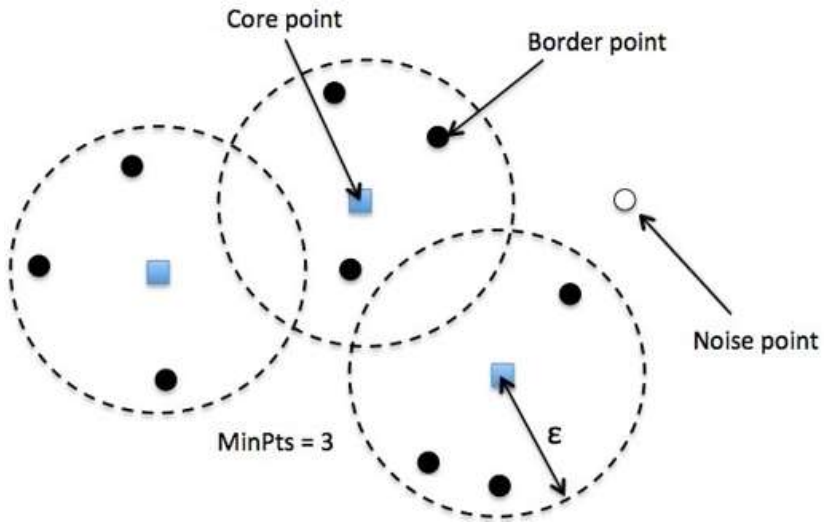




DBSCAN算法

DBSCAN(具有噪声的基于密度的聚类方法),是一种**基于密度**的空间聚类算法。该算法将具有足够密度的区域划分为簇，并在具有噪声的空间数据库中发现任意形状的簇，它将簇定义为密度相连的点的最大集合。在DBSCAN算法中将数据点分为三类：

- **核心点**（Core point）。若样本x_i的邻域 ε 内至少包含了MinPts个样本，即 $N_\varepsilon(X_i) \geq MinPts$，则称样本点$x_i$为核心点。
- **边界点**（Border point）。若样本x_i的 ε 邻域内包含的样本数目小于MinPts，但是它在其他核心点的邻域内，则称样本点 x_i 为边界点。
- **噪音点**（Noise）。既不是核心点也不是边界点的点

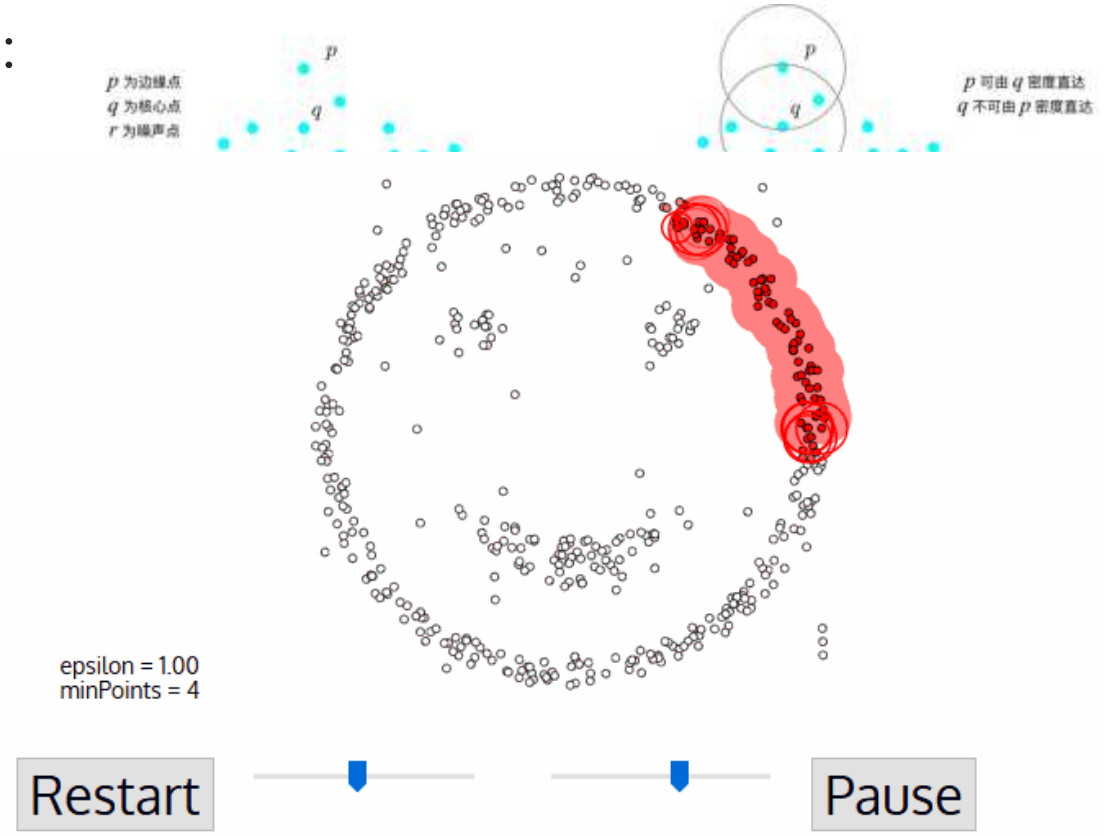


半径Eps ε，另一个是指定的数目MinPts

DBSCAN算法

在DBSCAN算法中，还定义了如下一些概念：

- **密度直达**(directly density-reachable)：我们称样本点 p 是由样本点 q 对于参数 {Eps,MinPts} 密度直达的，如果它们满足 p∈NEps(q) 且 |NEps(q)|≥MinPts （即样本点 q 是核心点）
- **密度可达**(density-reachable)：我们称样本点 p 是由样本点 q 对于参数{Eps,MinPts}密度可达的，如果存在一系列的样本点 p1,...,pn（其中 p1=q,pn=p）使得对于i=1,...,n−1，样本点 pi+1 可由样本点 pi 密度可达
- **密度相连**(density-connected)：我们称样本点 p 与样本点 q 对于参数 {Eps,MinPts} 是密度相连的，如果存在一个样本点 o，使得 p 和 q 均由样本点 o 密度可达。



在DBSCAN算法中，聚类“簇”定义为：**由密度可达关系导出的最大的密度连接样本的集合**

5 DBSCAN算法流程

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

1. 根据给定的邻域参数Eps和MinPts确定所有的核心对象
2. 对每一个核心对象
3. 选择一个未处理过的核心对象，找到由其密度可达的的样本生成聚类“簇”
4. 重复以上过程

伪代码

```
首先将数据集D中的所有对象标记为未处理状态
for（数据集D中每个对象p） do
    if （p已经归入某个簇或标记为噪声） then
        continue;
    else
        检查对象p的Eps邻域 NEps(p)；
        if（NEps(p)包含的对象数小于MinPts) then
                标记对象p为边界点或噪声点;
        else
                标记对象p为核心点，并建立新簇C，并将
                p邻域内所有点加入C
                for（NEps(p)中所有尚未被处理的对象q)
                    do
                    检查其Eps邻域NEps(q)，若
                    NEps(q)包含至少MinPts个对象，
                    则将NEps(q)中未归入任何一个簇
                    的对象加入C;
                end for
        end if
    end if
end for
```

DBSCAN优缺点

优点

相比K-Means，DBSCAN 不需要预先声明 聚类数量。

可以对任意形状的稠密数据集进行聚类，相对的，K-Means之类的聚类算法一般只适用于凸数据集。

可以在聚类的同时发现异常点，对数据集中的异常点不敏感。

聚类结果没有偏倚，相对的，K-Means之类的聚类算法初始值对聚类结果有很大影响。

缺点

当空间聚类的密度不均匀、聚类间距差相差很大时，聚类质量较差，因为这种情况下参数MinPts和Eps选取困难。

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树或者球树进行规模限制来改进。

在两个聚类交界边缘的点会视乎它在数据库的次序决定加入哪个聚类，幸运地，这种情况并不常见，而且对整体的聚类结果影响不大（DBSCAN*变种算法，把交界点视为噪音，达到完全决定性的结果。）

调参相对于传统的K-Means之类的聚类算法稍复杂，主要需要对距离阈值eps，邻域样本数阈值MinPts联合调参，不同的参数组合对最后的聚类效果有较大影响。

- 7号房的礼物 (豆瓣).txt
- 2001太空漫游 (豆瓣).txt
- V字仇杀队 (豆瓣).txt
- 阿凡达 (豆瓣).txt
- 阿飞正传 (豆瓣).txt
- 阿甘正传 (豆瓣).txt
- 爱在黎明破晓前 (豆瓣).txt
- 爱在日落黄昏时 (豆瓣).txt
- 傲慢与偏见 (豆瓣).txt
- 霸王别姬 (豆瓣).txt
- 爆裂鼓手 (豆瓣).txt
- 被解救的姜戈 (豆瓣).txt
- 被嫌弃的松子的一生 (豆瓣).txt
- 本杰明·巴顿奇事 (豆瓣).txt
- 蝙蝠侠：黑暗骑士 (豆瓣).txt
- 蝙蝠侠：黑暗骑士崛起 (豆瓣).txt
- 辩护人 (豆瓣).txt
- 冰川时代 (豆瓣).txt
- 布达佩斯大饭店 (豆瓣).txt
- 猜火车 (豆瓣).txt
- 侧耳倾听 (豆瓣).txt
- 超能陆战队 (豆瓣).txt
- 超脱 (豆瓣).txt
- 沉默的羔羊 (豆瓣).txt
- 初恋这件小事 (豆瓣).txt

战斗中负伤而下身瘫痪的前海军战士杰克·萨利（萨姆·沃辛顿SamWorthington饰）决定替死去的同胞哥哥来到潘多拉星操纵格蕾丝博士（西格妮·韦弗SigourneyWeaver饰）用人类基因与当地纳美部族基因结合创造出的“阿凡达”混血生物。杰克的目的是打入纳美部落，外交说服他们自愿离开世代居住的家园，从而SecFor公司可砍伐殆尽该地区的原始森林，开采地下昂贵的“不可得”矿。在探索潘多拉星的过程中，杰克遇到了纳美部落的公主娜蒂瑞（佐伊·索尔达娜ZoeSaldana饰），向她学习了纳美人的生存技能与对待自然的态度。与此同时，SecFor公司的经理和军方代表上校迈尔斯（史蒂芬·朗StephenLang饰）逐渐丧失耐心，决定诉诸武力驱赶纳美人……　　本片采用3D技术拍摄，共耗资5亿美元制作发行，是电影史上最为昂贵的作品。本片荣获...

```
for i in range(0,len(all_file)):
    filename=all_file[i]
    filelabel=filename.split('.')[0]
    labels.append(filelabel) #电影名称列表
    file_add='E:/db250/'+ filename
    doc=open(file_add,encoding='utf-8').read()
    data=jieba.cut(doc) #文本分词
    data_adj=''
    delete_word=[]
    for item in data:
        if item not in texts: #停用词过滤
            data_adj+=item+' '
        else:
            delete_word.append(item)
    corpus.append(data_adj) #语料库建立完成
```

```
vectorizer=CountVectorizer()
transformer=TfidfTransformer()
tfidf=transformer.fit_transform(vectorizer.fit_transform(corpus))
weight=tfidf.toarray()
```

```
num=10
mykms=KMeans(n_clusters=num)
```

```
y=AgglomerativeClustering(n_clusters=20).fit_predict(weight)
```

label_0:['做慢与偏见（豆瓣）', '冰川时代（豆瓣）', '狮子王（豆瓣）', '蝴蝶效应（豆瓣）', '贫民窟的百万富翁（豆瓣）', '魔女宅急便（豆瓣）']
label_1:['低俗小说（豆瓣）', '可可西里（豆瓣）', '大鱼（豆瓣）', '天空之城（豆瓣）', '头号玩家（豆瓣）', '射雕英雄传之东成西就（豆瓣）', '少年派的奇幻漂流（豆瓣）', '布达佩斯大饭店（豆瓣）', '心迷宫（豆瓣）', '忠犬八公的故事（豆瓣）', '恋恋笔记本（豆瓣）', '战争之王（豆瓣）', '海上钢琴师（豆瓣）', '疯狂动物城（豆瓣）', '看不见的客人（豆瓣）', '纵横四海（豆瓣）', '罗生门（豆瓣）', '虎口脱险（豆瓣）', '血战钢锯岭（豆瓣）', '记忆碎片（豆瓣）', '钢琴家（豆瓣）', '阳光姐妹淘（豆瓣）', '魂断蓝桥（豆瓣）']
label_2:['一个叫欧维的男人决定去死（豆瓣）', '七宗罪（豆瓣）', '三傻大闹宝莱坞（豆瓣）', '上帝之城（豆瓣）', '两杆大烟枪（豆瓣）', '乱世佳人（豆瓣）', '你看起来好像很好吃（豆瓣）', '侧耳倾听（豆瓣）', '加勒比海盗（豆瓣）', '十二怒汉（豆瓣）', '千与千寻（豆瓣）', '地球上的星星（豆瓣）', '奇迹男孩（豆瓣）', '小森林 冬春篇（豆瓣）', '当幸福来敲门（豆瓣）', '惊魂记（豆瓣）', '房间（豆瓣）', '拯救大兵瑞恩（豆瓣）', '断背山（豆瓣）', '无敌破坏王（豆瓣）', '无间道（豆瓣）', '时空恋旅人（豆瓣）', '末代皇帝（豆瓣）', '本杰明·巴顿奇事（豆瓣）', '楚门的世界（豆瓣）', '泰坦尼克号（豆瓣）', '活着（豆瓣）', '海洋（豆瓣）', '海边的曼彻斯特（豆瓣）', '消失的爱人（豆瓣）', '燃情岁月（豆瓣）', '玩具总动员3（豆瓣）', '疯狂原始人（豆瓣）', '真爱至上（豆瓣）', '素媛（豆瓣）', '美丽人生（豆瓣）', '致命ID（豆瓣）', '致命魔术（豆瓣）', '蝙蝠侠：黑暗骑士（豆瓣）', '西西里的美丽传说（豆瓣）', '让子弹飞（豆瓣）', '辩护人（豆瓣）', '闻香识女人（豆瓣）', '阿凡达（豆瓣）', '音乐之声（豆瓣）', '飞屋环游记（豆瓣）', '香水（豆瓣）', '黑客帝国（豆瓣）', '黑客帝国3：矩阵革命（豆瓣）']
label_3:['七武士（豆瓣）', '恐怖直播（豆瓣）', '情书（豆瓣）', '教父（豆瓣）', '教父2（豆瓣）', '星际穿越（豆瓣）', '狩猎（豆瓣）', '猫鼠游戏（豆瓣）', '盗梦空间（豆瓣）', '禁闭岛（豆瓣）', '红辣椒（豆瓣）', '美丽心灵（豆瓣）', '美国往事（豆瓣）', '蝙蝠侠：黑暗骑士崛起（豆瓣）', '被解救的姜戈（豆瓣）', '达拉斯买家俱乐部（豆瓣）']
label_4:['二十二（豆瓣）', '借东西的小人阿莉埃蒂（豆瓣）', '卢旺达饭店（豆瓣）', '告白（豆瓣）', '哪吒闹海（豆瓣）', '大闹天宫（豆瓣）', '头脑特工队（豆瓣）', '我是山姆（豆瓣）', '指环王1：魔戒再现（豆瓣）', '指环王2：双塔奇兵（豆瓣）', '指环王3：王者无敌（豆瓣）', '末麻的部屋（豆瓣）', '机器人总动员（豆瓣）', '模仿游戏（豆瓣）', '浪潮（豆瓣）', '海豚湾（豆瓣）', '玛丽和马克思（豆瓣）', '阳光灿烂的日子（豆瓣）', '风之谷（豆瓣）']
label_5:['三块广告牌（豆瓣）', '喜剧之王（豆瓣）', '幽灵公主（豆瓣）', '怪兽电力公司（豆瓣）', '教父3（豆瓣）', '死亡诗社（豆瓣）', '沉默的羔羊（豆瓣）', '爆裂鼓手（豆瓣）', '猜火车（豆瓣）', '神偷奶爸（豆瓣）', '第六感（豆瓣）', '终结者2：审判日（豆瓣）', '肖申克的救赎（豆瓣）', '触不可及（豆瓣）', '超脱（豆瓣）', '雨人（豆瓣）', '飞越疯人院（豆瓣）']
label_6:['2001太空漫游（豆瓣）', '7号房的礼物（豆瓣）', 'V字仇杀队（豆瓣）', '人工智能（豆瓣）', '哈利·波特与魔法石（豆瓣）', '唐伯虎点秋香（豆瓣）', '完美陌生人（豆瓣）', '寻梦环游记（豆瓣）', '心灵捕手（豆瓣）', '摩登时代（豆瓣）', '放牛班的春天（豆瓣）', '朗读者（豆瓣）', '窃听风暴（豆瓣）', '荒蛮故事（豆瓣）', '谍影重重（豆瓣）', '谍影重重2（豆瓣）', '谍影重重3（豆瓣）', '超能陆战队（豆瓣）', '辛德勒的名单（豆瓣）', '这个杀手不太冷（豆瓣）', '驯龙高手（豆瓣）', '鬼子来了（豆瓣）', '黑天鹅（豆瓣）']
label_7:['一一（豆瓣）', '东邪西毒（豆瓣）', '何以为家（豆瓣）', '你的名字。（豆瓣）', '初恋这件小事（豆瓣）', '哈利·波特与死亡圣器(下)（豆瓣）', '哈尔的移动城堡（豆瓣）', '天书奇谭（豆瓣）', '天使爱美丽（豆瓣）', '天堂电影院（豆瓣）', '完美的世界（豆瓣）', '小森林 夏秋篇（豆瓣）', '小鞋子（豆瓣）', '忠犬八公物语（豆瓣）', '怦然心动（豆瓣）', '控方证人（豆瓣）', '拜跋吧！爸爸（豆瓣）', '新世界（豆瓣）', '无人知晓（豆瓣）', '海蒂和爷爷（豆瓣）', '海街日记（豆瓣）', '穿条纹睡衣的男孩（豆瓣）', '萤火虫之墓（豆瓣）', '被雄弃的松子的一生（豆瓣）', '釜山行（豆瓣）', '饮食男女（豆瓣）', '龙猫（豆瓣）']
label_8:['大话西游之大圣娶亲（豆瓣）', '大话西游之月光宝盒（豆瓣）']
label_9:['倩女幽魂（豆瓣）', '入殓师（豆瓣）', '剪刀手爱德华（豆瓣）', '功夫（豆瓣）', '勇敢的心（豆瓣）', '喜宴（豆瓣）', '岁月神偷（豆瓣）', '幸福终点站（豆瓣）', '恐怖游轮（豆瓣）', '我不是药神（豆瓣）', '春光乍泄（豆瓣）', '爱在日落黄昏时（豆瓣）', '爱在黎明破晓前（豆瓣）', '牯岭街少年杀人事件（豆瓣）', '甜蜜蜜（豆瓣）', '疯狂的石头（豆瓣）', '穿越时空的少女（豆瓣）', '绿皮书（豆瓣）', '罗马假日（豆瓣）', '花样年华（豆瓣）', '英雄本色（豆瓣）', '菊次郎的夏天（豆瓣）', '萤火之森（豆瓣）', '请以你的名字呼唤我（豆瓣）', '重庆森林（豆瓣）', '阿甘正传（豆瓣）', '阿飞正传（豆瓣）', '雨中曲（豆瓣）', '霸王别姬（豆瓣）']

```
abel_4:['七武士 (豆瓣)', '东邪西毒 (豆瓣)', '射雕英雄传之东成西就 (豆瓣)', '罗生门 (豆瓣)']
abel_5:['摩登时代 (豆瓣)', '闻香识女人 (豆瓣)', '雨人 (豆瓣)']
abel_6:['教父 (豆瓣)', '教父2 (豆瓣)', '教父3 (豆瓣)']
abel_7:['三傻大闹宝莱坞 (豆瓣)', '地球上的星星 (豆瓣)', '放牛班的春天 (豆瓣)', '死亡诗社 (豆瓣)', '浪潮 (豆瓣)', '超脱 (豆瓣)']
abel_8:['哪吒闹海 (豆瓣)', '大闹天宫 (豆瓣)', '幽灵公主 (豆瓣)', '新世界 (豆瓣)', '盗梦空间 (豆瓣)', '红辣椒 (豆瓣)', '致命魔术 (豆瓣)', '萤火之森 (豆瓣)', '蝙蝠侠：黑暗骑士
(豆瓣)', '蝙蝠侠：黑暗骑士崛起 (豆瓣)', '风之谷 (豆瓣)']
abel_9:['加勒比海盗 (豆瓣)', '房间 (豆瓣)', '断背山 (豆瓣)', '泰坦尼克号 (豆瓣)', '阿凡达 (豆瓣)']
abel_10:['大话西游之大圣娶亲 (豆瓣)', '大话西游之月光宝盒 (豆瓣)']
abel_11:['爱在日落黄昏时 (豆瓣)', '爱在黎明破晓前 (豆瓣)']
abel_12:['剪刀手爱德华 (豆瓣)', '大鱼 (豆瓣)']
abel_13:['黑客帝国 (豆瓣)', '黑客帝国3：矩阵革命 (豆瓣)']
abel_14:['海街日记 (豆瓣)', '萤火虫之墓 (豆瓣)', '龙猫 (豆瓣)']
abel_15:['哈利·波特与死亡圣器(下) (豆瓣)', '哈利·波特与魔法石 (豆瓣)']
abel_16:['借东西的小人阿莉埃蒂 (豆瓣)', '头脑特工队 (豆瓣)', '无敌破坏王 (豆瓣)', '玛丽和马克思 (豆瓣)', '疯狂原始人 (豆瓣)', '神偷奶爸 (豆瓣)', '驯龙高手 (豆瓣)']
abel_17:['谍影重重 (豆瓣)', '谍影重重2 (豆瓣)', '谍影重重3 (豆瓣)']
abel_18:['告白 (豆瓣)', '指环王1：魔戒再现 (豆瓣)', '指环王2：双塔奇兵 (豆瓣)', '指环王3：王者无敌 (豆瓣)', '第六感 (豆瓣)', '飞越疯人院 (豆瓣)']
abel_19:['小森林 冬春篇 (豆瓣)', '小森林 夏秋篇 (豆瓣)']
```

6 未来研究目标

为什么要卷积

I do **not hate** this movie -good√

I **hate** this movie and will **not** choose it -bad×

无法捕获像not hate这样由连续两个词所构成的关键特征的词的含义

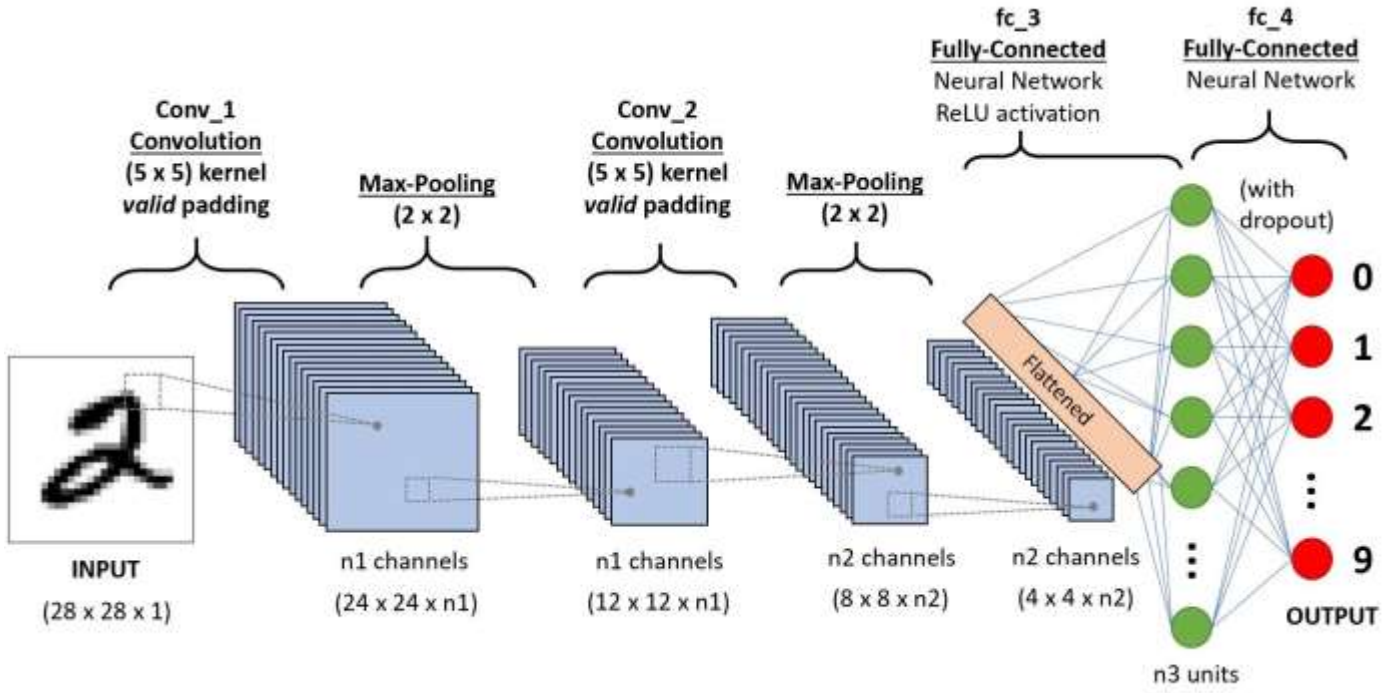
使用多元模型，如n-gram：

- 假设有20000个词，加入bi-gram实际上要有20000^2个词，这样参数训练显然是爆炸的。
- 不能共享例如参数权重等，会导致相似词无法获得交互信息。

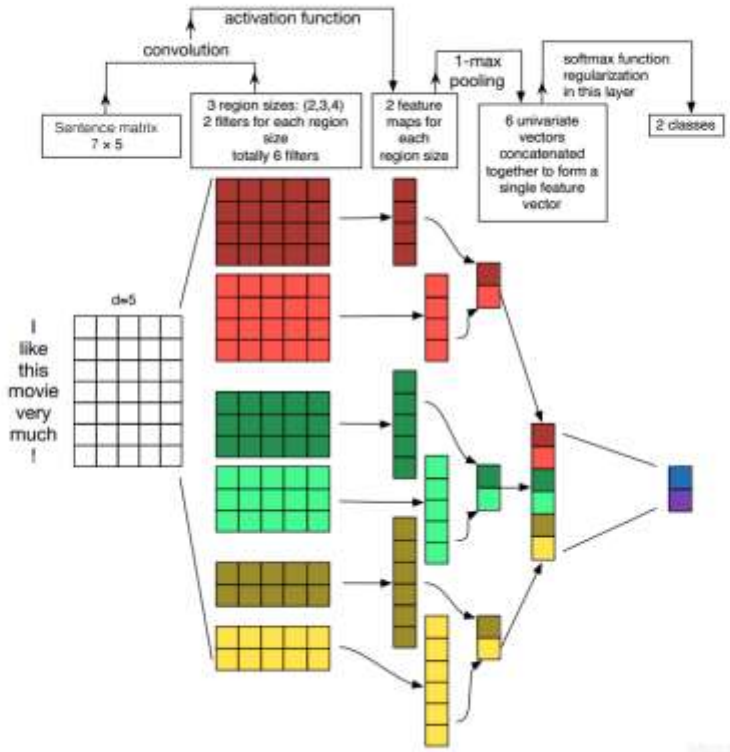
利用**卷积神经网络**是可以解决上述的问题

卷积神经网路

卷积神经网络其实就是多层卷积运算，然后对每层的卷积输出用非线性激活函数做转换。卷积过程中每块局部的输入区域与输出的一个神经元相连接。对每一层应用不同的**卷积核**，每一种卷积核其实可以理解为对图片的一种特征进行提取，然后将多种特征进行汇总，每一次做卷积操作是都是要对所有的特征通道进行卷积操作以便提取出更高级的特征。



卷积神经网络在NLP中的应用



在文本中，网络输入是句子（7*5），其中的每个单词已经映射为词向量，一个句子构成一个词数*词嵌入维度大小的矩阵，类似图片格式一样。然后经过不同大小的filter卷积，7*5，输出(句子长度- region_size+1)*1维的向量。经过最大池化得到一个数值，将各个filier的输出值经过最大池化后的值拼接起来进行最后的分类。所以在处理文本时，卷积核通常覆盖上下几行的词，所以此时卷积核的宽度与输入的宽度相同，通过这样的方式，我们就能够捕捉到多个连续词之间的特征，并且能够在同一类特征计算时中共享权重；然后，最后的softmax层接收此特征向量作为输入，并使用它对句子进行聚类。

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

谢谢观看
敬请老师批评指正