

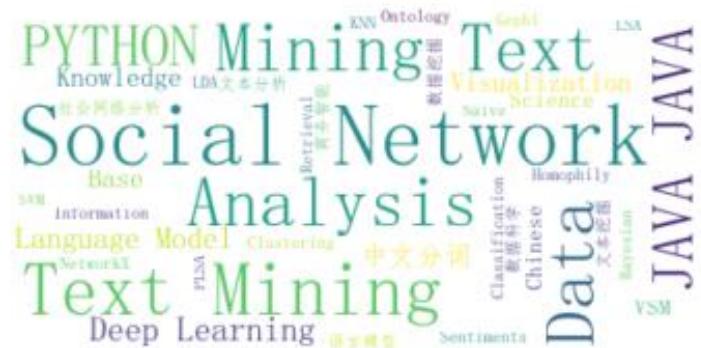
文本挖掘和社会网络分析

Text Mining and Social Network Analysis

西南财经大学经济信息工程学院

课程讲义

2018 版



邱江涛 编著

目录

目录	1
第一章：介绍	4
第二章：文本处理.....	5
第一节：网页爬虫	5
第二节：网页中的主内容抽取.....	10
第三节：使用正则表达式进行信息抽取.....	13
第四节：文本处理基础	26
第五节：中文分词	29
第三章：文本分析.....	34
第一节：从文档抽取关键词	34
第二节：句子的语义分析	40
第三节：词的语义分析	40
第四节：文本可视化:词云图.....	43
第四章：信息检索系统	46
第一节: 布尔检索	46
第二节：倒排索引	49
第三节：向量空间模型	54
第四节：概率检索模型和 BM25	62
第五节：统计语言模型	68

第六节：专题：使用 Lucene 构建文本检索系统.....	74
第五章：文本分类.....	86
第一节：朴素贝叶斯分类	86
第二节：特征选择	90
第三节: KNN.....	97
第四节：支持向量机.....	102
第五节：文本分类器总结	112
第六节：使用 mallet 进行文本分类实验.....	113
第六章：文本聚类.....	118
第一节：聚类算法介绍	118
第二节: K-Means.....	121
第三节：层次聚类	124
第四节：高斯混合模型和有约束的高斯混合模型	126
第五节：给文本聚类的簇贴标签	130
第六节：使用聚类算法的误区.....	135
第七章：矩阵分解与话题模型	136
第一节:线性代数基础知识	136
第二节：SVD	138
第三节：隐语义索引.....	140
第四节：PLSA	147
第五节：LDA	148
第八章：文本情感分析	151
第一节：文本情感分类介绍.....	151
第二节：文档情感分类	153

第三节：句子级的情感分类.....	161
第四节：Aspects 级的情感分析.....	161
第五节：基于机器学习和基于词典的情感分析方法的讨论	163
第六节：实例：为评论预测星级评分.....	164
第九章：知识图谱.....	168
第一节：本体和知识库	168
第二节：语义网 Semantic Web.....	168
第三节：知识图谱	168
第十章：社会网络分析理论	169
第一节：简介.....	169
第二节：社会网络的基本知识和度量.....	170
第三节：节点级的 SNA	178
第四节：网络级的 SNA	185
第十一章：Gephi：社会网络可视化工具	193
第十二章：社会网络分析实践和案例	195
第一节：分析实例	195
第二节：案例 1：微信朋友圈的广告投放.....	206
第三节：案例 2：社交媒体上的争议分析	208
第四节：案例 3：Yelp 上的“社交商务”实证分析	216
附录 A：LDA 模型的数学推导.....	217

第一章：介绍

文本数据和网络（图）数据是在实施数据挖掘、分析任务时经常遇到的数据类型。它们也是 Web 数据中的两种重要数据类型（Web mining aims to discover useful information or knowledge from the **Web hyperlink structure, page content, and usage data**. 《**Web Data Mining**》, Bin Liu）。本课程主要讲述文本挖掘技术和社会网络分析的方法。

文本挖掘是数据挖掘技术的一个子领域。它以文本集合为考察分析对象，运用各种算法，探索从文本集合中发现模式、新知识。文本挖掘在广泛学科、领域都有成功的应用。

社会网络是一种复杂网络，但和网络科学中研究的复杂网络又有所不同。体现在网络的节点是“人”；节点可以拥有很多属性，节点本身也可以具有标签。因此社会网络的分析有其独特的研究问题。

本课程强调理论和实践并重。掌握理论知识，也掌握算法和实用工具，可以去解决实际问题。因此，提供了很多实际应用的 JAVA 代码。因为在生成词云图和社会网络分析部分，JAVA 实在是没有好的工具，因此我们使用 PYTHON。

讲义在不断更新过程中。

本课程主要参考了下列的书籍：

1. 信息检索导论，Manning
2. Sentiment Analysis and Opinion Mining, Bing Liu
3. Social Network Data Analysis,
4. Python for Graph and Network Analysis

第二章：文本处理

在进行文本分析、文本挖掘前首先应该能够获得要分析的文本。互联网是最大的数据源。很多的分析任务都需要从互联网去下载网页。例如，如果想了解论坛中谈论的话题，需要下载论坛的网页，如果想了解电商网站商品的信息，也可以下载商品的网页然后抽取相关信息。当然，有很多网页采取了特殊的技术防止用户去下载内容，我们这里就不讨论了。这一章，我们首先介绍网络爬虫（Web Crawler）的工作原理，然后用开源的 java API crawler4j 来实现简单的一个网络爬虫。

获得文本后，并不是所有的文本都可以马上拿来进行分析。还需要有相应的步骤来对文本进行处理。例如，如果想从一篇新闻网页中获得主要的新闻内容。我们在第二节介绍网页主要内容的抽取方法。第三节，我们还将介绍一些基本的文本处理的概念和方法。中文文本有特殊之处，我们在第四节介绍中文分词的技术。

第一节：网页爬虫

1. 工作原理

在进行 web 挖掘时，基本的任务是需要获得网页。网络爬虫就是从互联网下载网页的工具。现在各种开源的网络爬虫很多，包括功能很全，很强大的 Nutch。也包括一些简单易用的如 crawler4j。各个搜索引擎的背后都有一个庞大的爬虫集群。一个典型的大规模网页爬虫的工作流程如图 2-1 所示：

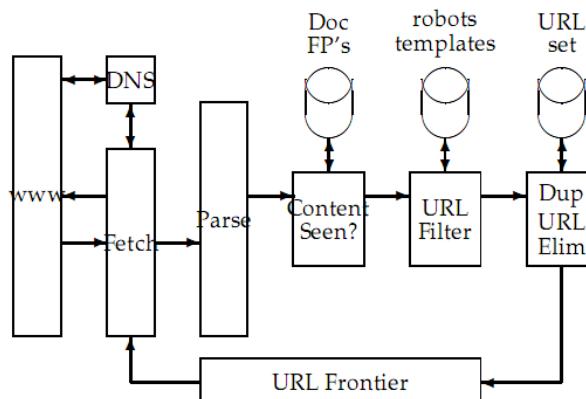


图 2-1 大规模爬虫的工作原理

URL Frontier 是待采集 URL 池。它包含了当前待采集的 URL

DNS 是 DNS 解析模块。它在根据 URL 抓取网页时用于确定其对应的 Web 服务器的 IP 地址。

Fetch 是抓取模块。获取 url 对应的网页。

Parse 是分析模块。从网页中抽取文本和超链接

在 Content Seen 通过文档指纹的方式确定是否具有相同内容的网页已经被抽取了。

URL 过滤器采用多个测试来确定某个抽取的 URL 是否应该被放在 URL 池。

Web 上的很多主机会在某些地方放置对本主机进行采集的限制条件，一些常规的实现措施就是采用**拒绝蜘蛛协议** (robots exclusion protocol)。它通过在网站服务器根目录下放置一个名为 robots.txt 的文件，来告诉访问该主机的网络蜘蛛那些可以下载或不能下载。一个 robots.txt 文件的示例如下：

```
User-agent: *
Disallow: /yoursite/temp/

User-agent: searchengine
Disallow:
```

该文件规定名为 searchengine 搜索引擎可以访问所有目录下的问题。其他搜索引擎不能访问 temp 文件夹下的内容。

2. 使用 jsoup 构造网页爬虫

Jsoup 是 java 的一个库 (library)。它完成 HTML 解析器 (parser) 的功能。使用 jsoup 可以下载网页。也可以从 HTML 文档中抽取数据。我们使用 jsoup 构造一个简单的爬虫，它包含的功能 (下载模式 mode) 如下：

(1) 给定种子网页，只下载种子网页

(2) 给定种子网页。从种子网页开始抽取超链，爬取超链对应的网页。并重复这个过程。因为我们实施的是一个简单的爬虫，因此我们还规定，只下载满足一定规则的网页。例如，URL 必须包含'swufe.edu.cn' 字符串。（不设定规则，我们很可能要爬取整个互联网）。

首先我们来看怎样用 jsoup 下载一篇网页。

```
String html;
```

```
String url="http://www.gov.cn/test/2006-02/16/content_200704.htm";  
try {  
    Document doc = Jsoup.connect(url).get();  
    html = doc.html();  
    System.out.println(html);  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

JSoup 类的静态方法 connect 可以获取一篇 url 地址对应的 html 网页的内容。

我们的网页爬虫项目名称为 Netcrawler。它包括下面的 java 文件：

- (1) Main.java : 主运行界面
- (2) Download.java : 各种下载模式的接口
- (3) DownloadMode1.java : 实现下载模式 1。给定种子网页，只下载种子网页
- (4) DownloadMode2.java : 实现下载模式 2。给定种子网页。从种子网页开始抽取超链，爬取超链对应的网页。并重复这个过程。
- (5) Parameter.properties : 设置参数的文件。参数包括 outdir 是下载网页的保存路径；mode 是下载模式；limits 是为了现在爬虫的搜索范围，设定的 url 地址的一级目录必须包含的字符串
- (6) Seeds.txt 是设置的种子 url。

因为其他的程序很简单，我们这里只介绍 DownloadMode2.java。

我们使用 LinkedList 类创建的对象作为一个队列。它存储了待下载的 url。有一个 HashSet 类的对象，存储访问过的 URL。我们必须把访问过的 url 保存起来，并在下载一条新的 url 之前，先判断是否该 url 已经被下载过了。

在 run 方法中，依次读取队列中的一条 url。如果它是已经访问过的（ HashSet 的对象 visited 包含该条 url ），这略过。否则就在 get 方法中下载该网页，并抽取出该网页中包含的 url。我们使用 jsoup 提供的下面方法来抽取一篇网页中的所有 url

```
String url = "http://it.swufe.edu.cn/";
```

```

try{

    Document doc = Jsoup.connect(url).get();

    Elements links = doc.select("a[href]");

    for (Element link : links) {

        System.out.println(link.attr("abs:href"));

    }

}catch(Exception e){

    e.printStackTrace();

}

```

我们应该检查每条抽取出的 url 会检查是否符合我们规定的 url 条件（即是否包含我们在参数 limits 中规定的必须包含的子串）和是否已经访问过了。如果符合规定，且没访问过，即把该抽取的 url 放入下载队列。下面是 DownloadMode2 的源码

```

/**
 * download mode 2 : download html file starting from seed url
 * */

public class DownloadMode2 implements Download{

    private HashSet<String> visited;

    @Override

    public void run(LinkedList<String> que){

        String url;

        int k = 1;

        visited = new HashSet<String>();




        while(!que.isEmpty()){

            url = que.removeFirst();

            if(visited.contains(url)) continue;

            visited.add(url);

            get(url, k+".html", que);

            k++;

        }

    }

}

```

```

    }

    private void get(String url, String fname, LinkedList<String>
que){

    String html;

    Parameters param = Parameters.getInstance();

    String file = param.outdir+"/"+fname;

    PrintWriter pw;

    try {

        Document doc = Jsoup.connect(url).get();

        pw = new PrintWriter(new File(file));

        html = doc.html();

        pw.println("<url>" + url + "</url>");

        pw.print(html);

        pw.close();

        extractURL(doc, html, que, param);

    } catch (IOException e) {

        System.out.println("Download "+url+" fail");

    }

}

private void extractURL(Document doc,
                        String str,
                        LinkedList<String> que,
                        Parameters param){

    Elements links = doc.select("a[href]");

    String url;

    for (Element link : links) {

        url = link.attr("abs:href");

        if(check(url,
param)&&!visited.contains(url)&&!que.contains(url)){

```

```

        que.add(url);
    }

}

}

/**

 * 检查超链是否符合设定的规则 ( parameter.properties中的limits )

 */

private boolean check(String url, Parameters param){

    String regex="https?://(.+?)/";
    Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
    Matcher m = p.matcher(url);
    String s;

    if(m.find()){

        s=m.group(1);

        if(s.contains(param.limits)){

            return true;
        }
    }

    return false;
}
}

```

第二节：网页中的主内容抽取

Web 内容挖掘通常关心 Web 的主要内容。导航条、广告、版权等不关心内容被视为噪声。噪声常常干扰 Web 挖掘的性能。例如，在 Web 分类和聚类中，如果从网页中提取的特征向量包含噪声，将使分类和聚类的精确度降低。在实际应用中，当收集了网页并需要从网页中提取新闻内容时，首先需要判断一篇网页是否含有主内容。我们将网页分为两类，一是噪声页，例如，一些门户网站的首页，这类网页上分布着种

类繁多的信息，并没有突出一个主题的整段文字内容，如图 2-2 (a) 所示；二是内容页，如一篇新闻网页，其中包含关于某个主题的整段的文字内容，如图 2-2 (b) 所示。图中框内的内容就是新闻内容。

与噪声页相比，内容页的明显特征是有一个内容块。若判别网页是否为内容页，则需要研究网页的属性，然后将网页判别的问题转换为分类问题。

规范化 HTML 文档可以使用辅助栈来正确获得网页块划分中的内容。“块”的定义如下：

定义 1 (块和子块)：设字符序列 S 表达一篇 HTML 文档，对于 S 中任意一对 HTML 标记 $\langle \text{TAG} \rangle \langle / \text{TAG} \rangle$ ，从标记 $\langle \text{TAG} \rangle$ 到 $\langle / \text{TAG} \rangle$ 是一个子序列 $s = \{\langle \text{TAG} \rangle, \dots, \langle / \text{TAG} \rangle\} \subset S$ 。对于任意一个子序列 $s \in S$ ，如果 $\exists s_j \in S$ ，在序列位置上满足 $s_j \subset s$ ，或者 $\neg \exists s_j \in S$ 满足 $s_j \subset s$ ，记 $s_j = \emptyset$ ，则我们将 $(s - s_j)$ 称为 **块**，记为 B ， s_j 称为 s 的 **子块**。

我的研究中提出了一个用块分布来获取网页组内容的方法。其工作原理如下：

提取块的内容时，遇到开标记，建一个块，将块插入到块链表中，同时相应的将标记和新块的引用入栈。遇到闭标记则将栈顶元素出栈。当遇到一个标记时，将上一个标记和当前标记之间的内容提取出来，插入到当前栈顶标记对应的块中。那么就可以将网页的内容正确的插入到块链表中。



图 2-2 (a) 无主内容网页 图 1 (b) 有主内容网页

定义 2 (网页块分布和块分布曲线)：设 BL 是由算法 1 获得的块链表， o 为 BL 中的一个结点， c 为结点 o 的内容属性。如果对于所有 $\forall o \in BL$ ，统计 o 的内容属性 c 的字符大小，记为 n ，那么将获得一个反映网页所有块大小的集合 $\{n_1, \dots, n_k\}$ 。将集合

元素按降序排序后的得到的序列 $D = (n_1, \dots, n_k)$ 称为网页的块分布。在块分布 $D = (n_1, \dots, n_k)$ 中，如果对于 $\forall n_i \in N$ ，以 n_i 的值为纵坐标， i 在块分布中的序号为横坐标，则将可以得到一条分段曲线，称之为网页块分布曲线。如图 2-3 所示

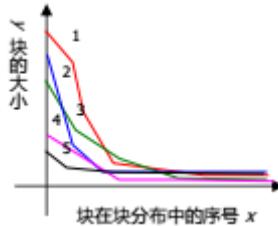


图 2-3 网页的块分布曲线

我们的实验显示，仅使用块分布的方差属性来进行分类时，分类精确度并不足够好。因此我们引入块分布的弯曲度属性。

定义 3 (块分布的弯曲度 β)：设 $D = (n_1, \dots, n_k)$ 是一篇网页的块分布；如果以 n_i 的值为纵坐标， i 在 D 中的序号为横坐标将可以得到分段曲线。设 α_i ($i=1, \dots, k-1$) 是每段曲线的斜率，我们将 $\beta(D) = |Max(\alpha_1, \dots, \alpha_{k-1}) - Min(\alpha_1, \dots, \alpha_{k-1})|$ 称作块分布 D 的弯曲度。

弯曲度属性可以反映内容页和噪声页的差别。

每篇网页可以获得块分布的方差和弯曲度两个属性。内容页判别就成了基于方差和弯曲度两个属性的网页分类问题。实验 2 证明，使用块分布的方差和弯曲度两个属性，用决策树方法进行分类，可以达到很好的网页判别效果。

内容页中，主要内容块少且大。相对于噪声块，内容块就是孤立点。因此可以采用孤立点分析的方法找出内容块。内容块中的内容就是最后提取出的新闻内容。

基于块分布的网页主内容程序我做成了一个 jar 文件 webextract.jar。把该 jar 文件和课件里给出的 cfile-1.4.jar 添加到工程中。parser.Main 类是调用程序的主界面，它包含三个方法

(1) static public String get(String filename)

使用默认弯曲度阈值抽取主内容

(2) static public String get(String filename, double th)

用户可以自己设置弯曲度阈值 0-1

(3) 如果上面的方法出现了中文乱码使用

```
static public String getChinese(String filename)
```

使用该程序的示例代码如下：

```
public static void main(String[] args) {  
    String file = "1.htm";  
    String str = parser.Main.getChinese(file);  
    System.out.println(str);  
}
```

程序调用 parser.Main 类中的 get 方法来获得主内容。这里没有给出弯曲度阈值的设定，程序里默认的阈值是 0.79。即块分布的弯曲度大于 0.79 时会被认为是有主内容的，否则就是无主内容的噪声页。用户也可以自己设定，如，设为弯曲度 0.6

```
String str = parser.Main.get(file, 0.6);
```

请访问网站 <http://www.biswufe.cn/maincontents/>，使用主内容抽取功能

第三节：使用正则表达式进行信息抽取

正则表达式 (Regular Expression) 就是按照一定语法规则写好的字符串，因为有语法规则，因此正则表达式也被看做是一种语言，可以灵活、有效、高效的处理文本。它已经被许多现代编程语言集成。例如，你可以在 JAVA、PYTHON、R 中使用正则表达式。正则表达式包含两类字符，特殊字符称作 meta characters，用于表达正则表达式的语法。另一类称为 literal 或字面字符，它们不是特殊字符，就是表达文字本身。

正则表达式通过按一定语法规则在一段文字中进行匹配，然后返回匹配结果。

这样的匹配包括：(1) 在一段文本中发现是否包含某个子串；(2) 抽取出符合正则表达式语法的子串；(3) 对符合正在表达式语法的子串进行替换等。(4) 返回一个布尔值，指示是否匹配。

在 Java 中使用正则表达式的方法如下：

Java 中关于正则表达式有两个类，一个接口和一个异常

```
Java.util.regex.Pattern; Java.util.regex.Matcher; Java.util.regex.MatchResult;  
Java.util.regex.PatternSyntaxException;
```

一个 Pattern 对象是一个编译的正则表达式，可以用于任何字符串。一个 Matcher 对象是一个单个的用于一个特别目标字符串的实例。MatchResult 封装了一次成功匹配的数据。

Pattern.compile()方法将一个字符串编译成正则表达式。即得到一个 Pattern 对象。

```
String regex="(<id?>.+?</id?>)";  
String str=<id>4654165</id><String>QWEHASOIUASC</string>;  
Pattern p = Pattern.compile(regex);  
Matcher m = p.matcher(str);  
String s;  
  
if(m.find()){  
    s = m.group(1);  
    System.out.println(s);  
}
```

上例中，字符串 regex 是我们写好的正则表达式的字符串。变量 str 是待匹配的字符串。

```
Pattern p = Pattern.compile(regex);
```

将正则表达式字符串编译，得到一个 Pattern 对象。

Pattern 对象下的 matcher()方法是使用当前编译的正则表达式去匹配字符串，然后得到一个 Matcher 对象。Matcher 对象的 find()方法判断是否匹配成功。正则表达式的一次匹配结果可以包含多个子串，用 Matcher 对象的 group 方法来获得匹配结果的多个子串。其参数是一次匹配中第几个匹配的子串。如上图是一次匹配中的第一个匹配的子串。（注：一个正则表达式可以从匹配的字符串中抽取多个子串，其中子串用括号来表示，如上例中有一对括号，因此就有一个匹配的子串。）

1. 正则表达式

一些 meta characters 如下：

(1) ^ 开始字符。例如，^cat 表示匹配一段文字。这一段文字如果以'cat'开始，则可以匹配

(2) \$ 结束字符。例如，cat\$，表示匹配以 cat 结尾的一段文字

(3) [] 称作字符类，它列举想匹配的字符。例如，gr[ea]y，表示匹配含有 grey 或 gray 子串的文字。

(4) 在字符类内部使用符合“-”表示范围。例如，<H[1-6]>表示<H1>, ..., <H6>
[0123456789abcdEFGH]可以写成[0-9a-dE-H]。注：“-”仅仅在字符类内部是 meta character。

(5) 在字符类内使用^表示非。[^1-6]表示匹配一个字符不是 1 到 6 的。

问：下面代码的运行结果是什么

```
String regex="[1-6];  
String str="hello 6 world";  
Pattern p = Pattern.compile(regex);  
Matcher m;  
String s;  
m = p.matcher(str);  
System.out.println(m.find());
```

这段正则表达式的含义是匹配包含数字的字符串。再看这一段代码

```
String regex="[^1-9]";  
String str="hello 6 world";  
Pattern p = Pattern.compile(regex);  
Matcher m;  
String s;  
m = p.matcher(str);  
System.out.println(m.find());
```

这一段代码运行结果为 true。因为上面的正则表达式的含义是，匹配含有不是 1-9 数字的字符串。

(6) 字符“.”匹配任意字符。但在字符类[.]内的“.”就是它的字面意思而不是特殊字符。例如正则表达式 03[.]16[.]76 可以匹配 03.16.76，03\16\76，03.16\76，03\16.76。

再看正则表达式 “[0-9].[0-9]” ，表达的是匹配包含下来子串的字符串 “**数字任意字符数字**”。即子串的第一个字符是数字，第二个字符是任意字符，第三个字符是数字。

(7) “|” 表示或，可以让用户联合多个表达式到一个表达式，例如，Bob|Robert 表示匹配 Bob 或 Robert。正则表达式 Gr[ea]y 可以写成 grey|gray。也可以写成 gr(e|a)y。但 gr[e|a]y 中的“|”不是特殊字符，是其字面意思。**gr(e|a)y 中括号约束了替换的范围，即两个字符 e 和 a 的替换。**

再举个例子：正则表达式“(first|1st).[Ss]treet”匹配字符串 first street, first Street, 1st street 和 1st Street

(8) \<和\>称作词的边界，匹配一个 word 的开始和结束位置。\\<cat 表示以 cat 开始的词，例如 cate。cat\\>表示以 cat 结束的词。注意：“<” 和 “>” 不是特殊字符，但 “\\<” 和 “\\>” 是。（**java 里应用没有成功**）

(9) ? 意味着选择，即在?前面的字符是可选择的。colou?r 即匹配 color 或者 colour。4(th)?表示匹配 4 或者 4th。此处括号对可选择的多个字符做了限制。

(10) +意味着一个或多个立即跟随的字符。“*”意味着任何数据（或没有）的字符。例如，[0-9]+表示任意长的数字，012，,91 等

(11) “\” 表示后面跟的字符是字面含义，而不是特殊字符。例如，ega\\.att\\.com。此处的“.”不是特殊字符而是 literal。匹配 ega.att.com。

(12) {min,max}定义匹配的范围。例如, [a-z]{1,3}表示字母 a-z 可以出现一次，最多三次。

(13) 括号()有两种含义，一是表示限定范围，见(7)(9)的例子。二是，当从字符串中提取子串时，括号规定了要提取的子串，见后面的例子(5)。

(14) 一些特殊字符：

\d 表示数字；

\D 表示非数字。它等于[^\\d]

\w 等同于[a-zA-Z0-9]

\W 等同于 [^\\w]

\s 空白符等同于[\f\n\r\t\v]

\S 非空白符

\b 匹配一个 backspace 或 tab 字符

例子：

下面用正则表达式进行匹配的结果是什么？

(1)

```
String regex="q[^u]";  
String str="Iraq";  
Pattern p = Pattern.compile(regex);  
Matcher m = p.matcher(str);  
System.out.println(m.find());
```

该结果是 false，为什么没有匹配上呢，是因为该正则表达式要匹配的是字符串包含字符“q”，并且后面必须跟上一个不是“u”的字符。

(2)

```
String regex="q[^u]";  
String str="Qantas";  
Pattern p = Pattern.compile(regex);  
Matcher m = p.matcher(str);  
System.out.println(m.find());
```

是 FALSE，因为正则表达式是大小写敏感的。

如果想大小写不敏感，编译正则表达式时用

```
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
```

(3) 两个正则表达式

^From|Subject|Date:

^(From|Subject|Date):

有什么不同？下面的正则表达式匹配字符串吗？

```
String regex="^(From|Subject|Date):";  
String str="hello Subject: dallas";  
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);  
Matcher m = p.matcher(str);  
System.out.println(m.find());
```

第一个正则表达式 “^From|Subject|Date:” , 表示匹配 “^From” (被匹配的字符串必须以 From 开头) 或者 “Subject” (包含子串 Subject) 或者“Date:” (包含字符串 Date:) 。

第二个正则表达式“^(From|Subject|Date):”表示匹配的字符串必须以 “From:” 或者 “Subject :”或者 “Date:” 开头。

(4) 下面的代码运行结果是什么

```
String regex="^h.+d$";
String str="hello world";
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
Matcher m = p.matcher(str);

System.out.println(m.find());
```

这段正则表达式匹配以 h 开头 , 以 d 结尾的字符串。 “.+” 表示匹配任意多个字符。

```
String regex="hello\\.world";
String str="Hi, hello.world, Yes";
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
Matcher m = p.matcher(str);

System.out.println(m.find())
```

这段代码中 , 我们要匹配一个子串“hello.world”。因为正则表达式中“.”是特殊字符。因此要匹配子串 , 我们写的正则表达式应该是“hello.\world” , 这里的“\”表示后面的“.”是其字面意思 , 不是特殊字符。而在 java 中“\”又是特殊字符 , 我们就再加上一个“\”字符。因此得到了最终的正则表达式“hello\\.world”。

(5) 我们要提取字符串 “price is 34\$, today is April 12” 中描述的价格。正则表达式 , 以及 Java 代码应该怎么写。

```
String regex="([0-9]+)\\$";
String str="price is 34$, today is April 11";
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
```

```

Matcher m = p.matcher(str);
String s;

if(m.find()){
    s=m.group(1);
    System.out.println(s);
}

```

美元符号\$在正则表达式中是特殊字符，当前的例子中，文本中包含了美元符号，因此在写正则表达式时，需要在它的前面加上\"表示其后的美元符号不是特殊字符。

另外，括号的作用在于指示这是要提取出的子串。因此程序运行结果是 34。

(6) 我想匹配上面的所有数字，即 34 和 11，正则表达式和 Java 代码如下

```

String regex="([0-9]+)";
String str="price is 34$, today is April 11";
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
Matcher m;
String s;
Integer location=0;
m = p.matcher(str);

while(m.find()){
    s=m.group(1);
    System.out.println(s);
}

```

匹配的结果中包含了匹配的子串的起始位置和结束位置，可以用 Matcher 对象的 start()方法和 end()方法来获得。定义一个 Integer 类型的变量 location，它记录每趟循环中匹配到的子串的结束位置。

find (location) 方法，表示从当前 location 的位置去寻找匹配的子串。因此在 while 循环中多次去匹配字符串，发现字符串多个与正则表达式匹配的子串。

(7) 对于包含小数点的数字怎么提取？例如， “price is 34.34\$”

```
String regex="([0-9]+\.\[0-9]+)";  
String str="price is 343.4$, today is April 11";  
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);  
Matcher m;  
String s;  
m = p.matcher(str);  
if(m.find()){  
    s=m.group(1);  
    System.out.println(s);  
}
```

再问：如果字符串中混合了小数和整数，应该怎么提取？例如，“price is 343.4\$，
today is April 11”

```
String regex="([0-9]+(\.\[0-9]+)?)";  
String str="price is 343.4$, today is April 11";  
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);  
Matcher m;  
String s;  
  
Integer location=0;  
m = p.matcher(str);  
  
while(m.find()){  
    s=m.group(1);  
    System.out.println(s);  
}
```

该正则表达式中有两对括号，其中一对嵌入在另一对中。里面的那一对后面有个？，
是用来指示括号里面的内容是“可选择的”。匹配的子串是外面的那一对括号。

(8) 提取出时间，例如“9:45 am”，“12:00 pm”

```

String regex="([0-9]?[0-9]:[0-9][0-9]\\s(am|pm)?)";
String str="It is 9:21 am";
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
Matcher m;
String s;
m = p.matcher(str);
if(m.find()){
    s=m.group(1);
    System.out.println(s);
}

```

(9) <http://www.xxx.com/hello/good.jpg>

从超链的地址中提取出图片名

```

String regex="([^\n]+\\.(jpg|jpeg|bmp|png))";
String str="http://www.xxx.com/hello/good.jpeg";
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
Matcher m;
String s;
m = p.matcher(str);
if(m.find()){
    s=m.group(1);
    System.out.println(s);
}

```

2. 贪婪模式和非贪婪模式

正则表达式默认的是最大匹配，即贪婪模式。例如，html 文档

"<p>hello</p><div>world</div>"，我们想对每一对标记进行匹配，即当前例子中匹配的结果应该是两个"<p>hello</p>"和 "<div>world</div>"

如果我们写正则表达式"<[a-zA-Z]+>.+</[a-zA-Z]+>"。匹配的结果只有一个

"<p>he11o</p><div>wor1d</div>"

这就是贪婪模式，如果我们想使用非贪婪模式，正则表达式中的“*”或“+”后加上一个?符号，称为惰性符号

```
"<[a-zA-Z]+>.+?</[a-zA-Z]+>"
```

匹配结果如下：

```
"<p>hello</p>"      "<div>world</div>"
```

练习 1：下载一篇网页，获取该网页中的所有超链

因为使用 `getURL` 下载中文网页时乱码，因此使用 `download.file` 来下载网页，但多了一个写网页到磁盘，再从磁盘读入的操作。

```
public class ExtractURL {  
    public static void main(String[] args) {  
        ExtractURL e=new ExtractURL();  
        String urlString="http://www.swufe.edu.cn";  
        try {  
            e.run(urlString);  
        } catch (Exception e1) {  
            e1.printStackTrace();  
        }  
    }  
  
    public void run(String urlString) throws Exception{  
        URL url = new URL(urlString); // Create the URL  
        InputStream in = url.openStream(); // Open a stream to it  
        String contents=getContent(in);  
        Vector<String> vec=extractURL(contents);  
  
        for(int i=0;i<vec.size();i++){  
            System.out.println(vec.get(i));  
        }  
        System.out.println(contents);  
    }  
  
    private String getContent(InputStream in) throws IOException{
```

```

int bytes_read;

byte[] buffer=new byte[4096];

ByteArrayOutputStream ba=new ByteArrayOutputStream();

StringBuilder sb=new StringBuilder();


while((bytes_read = in.read(buffer)) != -1){

ba.reset();

ba.write(buffer);

sb.append(ba.toString());

}

return sb.toString();

}

private Vector<String> extractURL(String str){

String regex=<a href=?(.+?)\?"?(\s|>)";

Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);

Matcher m;

String s;

Integer location=0;

Vector<String> res=new Vector<String>();

m = p.matcher(str);

while(m.find()){

s=m.group(1);

res.add(s);

}

return res;

}

```

上面的代码中，正则表达式"

的含义是匹配子串或者<a href=...空格

其中表示双引号是可选项。以">"或者空格结束。此处的括号内的内容是提取的子串。

练习 2：下载一篇网页中的所有图片

```
public class ExtractIMG {  
  
    public static void main(String[] args) {  
  
        ExtractIMG e=new ExtractIMG();  
  
        String urlString="http://www.swufe.edu.cn";  
  
        try {  
  
            e.run(urlString);  
        } catch (Exception e1) {  
  
            e1.printStackTrace();  
        }  
    }  
  
    public void run(String urlString) throws Exception{  
  
        URL url = new URL(urlString);  
  
        InputStream in = url.openStream();  
  
        String contents=getContent(in);  
  
        Vector<String> vec=extractIMGURL(contents);  
  
        for(int i=0;i<vec.size();i++){  
  
            System.out.println(vec.get(i));  
  
            saveIMG(vec.get(i));  
        }  
    }  
  
    private String getContent(InputStream in) throws IOException{  
  
        int bytes_read;  
  
        byte[] buffer=new byte[4096];  
  
        ByteArrayOutputStream ba=new ByteArrayOutputStream();  
  
        StringBuilder sb=new StringBuilder();
```

```

        while((bytes_read = in.read(buffer)) != -1){

            ba.reset();

            ba.write(buffer);

            sb.append(ba.toString());

        }

        return sb.toString();
    }

    private Vector<String> extractIMGURL(String str){

        String prefix="http://www.swufe.edu.cn/";

        String regex=<img src=?(.+?)\.(jpg|jpeg|bmp|png))?"?;

        Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);

        Matcher m;

        String s;

        Integer location=0;

        Vector<String> res=new Vector<String>();

        m = p.matcher(str);

        while(m.find()){

            s=m.group(1);

            res.add(prefix+s);

        }

        return res;
    }

    /**
     * urlString是图片的url地址，fname是要保存在本地的文件名
     */
    private void saveIMG(String urlString) throws Exception{

        URL url = new URL(urlString);

        InputStream in = url.openStream();

```

```

String fname;
String regex="([^\/]+\.\(jpg|jpeg|bmp|png\))";
Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
Matcher m;
String s;
m = p.matcher(urlString);
if(m.find()){
    fname="d:/"+m.group(1);
}else{
    return;
}

RandomAccessFile rout=new RandomAccessFile(fname,"rw");
int bytes_read;
byte[] buffer=new byte[4096];

while((bytes_read = in.read(buffer)) != -1){
    rout.write(buffer,0,bytes_read);
}
rout.close();
}
}

```

第四节：文本处理基础

1. 文本处理的一些相关概念

我们首先介绍文本处理的一些相关概念和技术术语，如下：

(1) 文本、文档和语料库：文本 (text) 是指数据格式，文档 (Document) 是指文本文件。在信息检索中，文档检索系统检索对象，或可以理解为保存在计算机上的一个

待检索文件（一个文本文件，一篇网页等）。语料库（corpus）：多个文档组成的文档集合。

（2）结构化数据和非结构化数据：信息可以划分为两大类。一类是没有清晰和明显语义结构的数据，称为非结构化数据；而与之相对应的另一类信息称之为结构化数据。如文本、图像、声音、网页等，我们称之为非结构化数据。结构化数据最典型的例子是数据库中的数据。如，

姓名	性别	年龄	婚姻
王五	M	20	未婚
李四	F	30	已婚

结构化的数据很方便进行分析，如上表中，如果需要查找未婚的男性，因为数据格式都是固定的，标准的很方便进行查找。甚至格式化的数据可以进行量化，然后进行数据分析。

文本是非结构化数据。例如，

“王五是个男的今年 20 岁了还没结婚。李四今年 30 岁是个女的已经结婚了”

我们不能从这段文本中直接获得被描述的人的姓名、年龄、婚姻状况等信息。我们是说通常文本数据是非结构化的数据。但是如果文本中的数据按照一定规律存放，很容易读取，我们也不说它是非结构化数据。比如一段文本内容

Name=王五 ; Age=20 ; sex=男 ; marriage=未婚

Name=李四 ; Age=30 ; sex=女 ; marriage=已婚

还有像 HTML、XML 等文件，我们说它们是半结构化数据。

传统的数据分析方法不能直接作用在文本数据上。因此文本数据必须有相应的处理方法，将其量化后才能进行分析。所以，非结构化数据分析的思想都是将它们进行量化后才能进行分析。

（3）词条(token)和词项 (term)：词条是指对文档进行分词操作得到的一个单元；词项是指词典中的一个词。这里词典的含义是进行文本分析时建立的一个词的集合。词条不一定会出现在词典中。

2. 一些文本处理的技术：

（1）词干化或词干还原：是指将英文中可以表达为多种形态的一个词，如一个词有多种派生形式，如名词化，动词分词，形容词等。演变成了含义相近的新的词。在处理时，需要用一个词干表示这些词。例如，are, is 词干化为 be；prices 和 pricing 可以词

干化为 price。有很多词干化的工具包，如最常见的 porter
(<http://www.tartarus.org/~martin/PorterStemmer/>)。斯坦福的自然语言工具箱中也有词干化的工具。 (<http://nlp.stanford.edu/software/tagger.html#Download>)。

当然，词干的精确形式并不重要，重要的是能够得到等价类。这句话的意思是，如果把 “quickly” 这个词词干化得到的是 “quic” ，这个操作得到的结果是不精确，甚至是错误的。但是，如果所有的 quick 的衍生词，包括 quick 本身的词干化的结果都是 quic，而其他词的词干化结果不好得到 quic，就没有关系。因为所有的 quick 和 quick 衍生词的词干化结果是相同的，它们得到了等价类。

(2) 词项归一化 (normalization) : 是将看起来完全不一致的多个词条归纳成等价类，以便在它们之间进行匹配。比如，查询 USA 时，肯定希望检索系统能够返回包含 U.S.A. 的文档。大小写转换也是一种词项归一化操作。

(3) 停用词表使用的探讨

一些词项在文档中出现的太频繁了，不能表达文档的主题信息，它们称为停用词。如 “the” , “and” , 啊，吧等。在文本挖掘中，我们更喜欢频繁出现在一篇文档中，而没有出现在其他文档中的词。它们更能描述文档的主题信息。而语料库中每篇文档中都出现的词，不具有描述文档含义的作用。在文本处理时，会预先建立一个停用词表，然后根据停用词表把文本中的停用词删除。

在一些特定的文本挖掘任务中，比如文本分类中，会使用停用词表，以提高系统的性能，以及效率。因为文本分类中对实词（名词、动词、形容词）更感兴趣，因为他们通常描述了文本的内容。

但在一些领域对虚词，即传统停用词表中的词更感兴趣。例如，在判断文章作者时，其实是在对文章的写作风格（体现在了作者对虚词，如语气词、助词等的使用）进行判断。例如，《联邦党人文集》，共计 85 篇文章，其作者是麦迪逊、汉密尔顿和杰伊 (Jay)。其中多数文章的作者是明确的，但有十几篇仍具争议。莫斯特勒 (Mosteller) 和华莱士 (Wallace) 对这些有争议文章进行鉴定时，把多数文章作为训练集建立一个模型，用来对有争议的文件做判别。在训练时，莫斯特勒和华莱士估算词汇表中的每个词的似然比： $\text{Pr}(\text{word}|\text{Madison})/\text{Pr}(\text{word}|\text{Hamilton})$ 。对有争议的文章通过文中每个词的似然比的乘积打分。

$$\text{Score}(\text{doc}) = \prod_{\text{word} \text{ in } \text{doc}} \frac{\text{Pr}(\text{word}|\text{author1})}{\text{Pr}(\text{word}|\text{author2})}$$

这样的做法可以在我们的生活中找到例子，例如，聊天中可以发现四川人喜欢以“哈”做结尾；陕西人喜欢以“嘞”做结尾。

对于短语查询来说，采用停用词策略会有问题。短语查询是按照整个短语来匹配文档。例如，在百度中提交加入了双引号的查询“中文分词系统”，返回的网页都是匹配整个短语“中文分词系统”的网页。例子：President of the United States 是个短语但含有两个停用词“of”，“the”。如果倒排索引中不含该两个词，则没办法进行短语查询。

信息检索系统的发展中，有从大停用词表（200-300）个词，到小停用词表（7-12）个词，最后到不用停用词表的趋势。现在大型、专业的搜索引擎中通常都不使用停用词表了。它们更关注如何利用语言的统计特性来更好的处理常见词的问题。

练习 1：使用 Stanford Tagger 做词干化

- (1) 下载 postagger 包 <http://nlp.stanford.edu/software/stanford-postagger-2016-10-31.zip>。解压后获得 `stanford-postagger.jar`
- (2) 建立 java 工程，将该包添加到该工程中。使用类 `Morphology` 的 `stem` 方法完成词干化的操作。

第五节：中文分词

由于中文的词之间没有分隔符，如果要进行中文文本处理，必须要用特殊的方法将中文的句子分词一个个的词条。中文分词是中文信息处理的最基本任务，几十年的研究里有丰富的成果，技术很成熟，而且有很多的开源中文分词的软件。

中文分词的技术分为两大类，基于词典的分词和基于机器学习的分词。2002 年以前的研究都是基于词典的分词方法。2003 年基于字标注（character-based Tagging）的机器学习分词开始崭露头角，后来不断发展成为了当今分词技术的主流。我们引用《中文分词十年回顾》一文中的结论：(1) 实践证明，基于手工规则的分词系统（注：基于词典的分词方法中再添加人工规则，如词法规则等，也可以是统计出的一些规则）在评测中不敌基于统计学习的分词系统（注：这里是指机器学习的方法）；(2) 未登录词（out-of-vocabulary）造成的分词精度失落至少比分词歧义大 5 倍；(3) 迄今为止的实验结果证明，能够大幅度提高未登录词识别性能的字标注统计学习方法优于以往的基于词典的方法，并使得分词精度达到新高（注：未登录词是造成分词精度低的主要原因，而基于字标注的机器学习方法提高了未登录词识别性能）。

1. 基于词典的机械分词

最初的分词技术就是采用基于词典的机械分词方法。它是按照一定的策略将待分析的汉字串与一个充分大的词典中的词项进行匹配，若在词典中找到了某个字符串，则匹

配成功（识别出一个词）。按照扫描方向的不同，该分词方法分为正向匹配和逆向匹配；按照不同长度优先匹配的情况，可以分为最大匹配或最小匹配。也有一些组合方法，如，将正向最大匹配和逆向最小匹配方法结合起来构成双向匹配法。由于汉语可以单字成词，正向最小匹配和逆向最小匹配一般很少使用。一般来说，逆向匹配的切分精度略高于正向匹配，遇到的歧义现象也较少。

下面介绍一个基本的基于词典的分词算法，正向减字最大匹配法。

正向减字最大匹配法分词的过程是从中文语句中提取出设定的长度字串，与词典比较，如果在词典中，就算一个有意义的词串。否则缩短字串，在词典中重新查找。

算法：正向减字最大匹配分词

输入：中文词典，中文句子 s ，最大词长 \max

步骤：

1. 对于 s ，从左向右以 \max 为界选出候选字串 w
2. 如果 w 在词典中，输入 w 。处理下一个长为 \max 的候选字串
3. 否则，将 w 最右边一个字去掉，继续与词典比较。直到 w 成为单字
4. 重复上面的步骤

因为要字串要频繁的与词典比较，字典上的匹配效率就很关键。一种成熟的技术是字典采用红黑树的平衡二叉搜索树的数据结构。

2. 基于字标注的机器学习分词方法

基于词典的分词方法需要预先有个词典，分词的过程就是通过词表和相关信息（如，还可以人工加入规则）来做出词语切分的决策。与此相反，基于字标注的分词方法实际上是构词法。即把分词过程视为字在字符串中的标注问题。由于每个字在构造一个特定的词语时都占据着一个确定的构词位置（即词位），假如规定每个字最多只有四个构词位置：即 B(词首), M(词中), E(词尾) 和 S(单独成词)，那么下面句子(甲)的分词结果就可以直接表示成如(乙)所示的逐字标注形式：

(甲)分词结果 :/上海/计划/到/本/世纪/末/实现/人均/国内/生产/总值/五千美元/。

(乙)字标注形式 :上/B 海/E 计/B 划/ E 到/S 本/S 世/B 纪/E 末/S 实/B 现/E 人/B 均/E 国/B 内/E 生/B 产/E 总/B 值/E 五/B 千/M 美/M 元/E 。/S

把分词过程视为字的标注问题的一个重要优势在于,它能够平衡地看待词表词和未登录词的识别问题。在这种分词技术中,文本中的词表词和未登录词都是用统一的字标注过程来实现的。在学习架构上,既可以不必专门强调词表信息,也不用专门设计特定的未登录词(如人名、地名、机构名)识别模块。这使得分词系统的设计大大简化。在字标注过程中,所有的字根据预定义的特征进行词位特性的学习,获得一个概率模型。然后,在待分字串上,根据字与字之间的结合紧密程度,得到一个词位的标注结果。最后,根据词位定义直接获得最终的分词结果。总而言之,在这样一个分词过程中,分词成为字重组的简单过程。然而这一简单处理带来的分词结果却是令人满意的。

现代机器学习的主要方法,包括隐马尔科夫模型(HMM)、最大熵和条件随机场,都已经被研究人员用于由字构词的词位学习中。我们下面以HMM为例。

首先,准备一个像上面例子中字标注的语料库。在标注好的语料库上学习HMM模型,结果包括:

StatusSet: 状态值集合, {B, M, E, S}。

ObservedSet: 观察值集合,所有观察到的汉字的集合

TransProbMatrix: 状态转移概率矩阵,描述了四个位置的状态转移概率,一个 4×4 的矩阵,即 $\{B,E,M,S\} \times \{B,E,M,S\}$;

EmitProbMatrix: 发射概率矩阵,一个字出现在四个构词位置的概率。因此该矩阵大小是状态的个数(此处是4)*字的集合的大小

InitStatus: 初始状态分布

状态转移矩阵图2-5所示如下:

	B	E	M	S
B	-3.14e+100	-0.511	-0.916	-3.14e+100
E	-0.590	-3.14e+100	-3.14e+100	-0.809
M	-3.14e+100	-0.333	-1.260	-3.14e+100
S	-0.721	-3.14e+100	-3.14e+100	-0.666

图 2-5 状态转移矩阵图示例

其中的一行,例如B所在的这一行。描述了前一个位置的字的词位是B,后一个字的词位是BEMS的概率取对数后的结果。

具体训练的过程就不讲了，简单的说，就是在标注的语料库上统计一个状态到下一个状态出现的次数，然后填表；统计一个字出现在每个位置上出现的次数，然后填表。

当新来了一个句子，分词时就是求，给句子中的每个字一个位置标签，得到的标注结果。Verbit 算法是在 HMM 上对序列进行标注的算法。得到的标注序列中标签 B 和 E 之间的字串就是分得的词。

分词工具

中文分词开源软件包有很多，如中科院，清华大学都推出了自己的分词工具。结巴分词（python）也是一款很受欢迎的工具。我的课堂上将使用 Stanford NLP 工具箱中包含的分词功能。

首先下载分词工具 <https://nlp.stanford.edu/software/segmenter.shtml#Download>。将该文件解压后，将 data 文件夹放到你的 eclipse 工程的根目录下。将文件 stanford-segmenter-X.X.X.jar 文件添加到你的工程。分词的方法可以参考该工具带的一个演示程序 SegDemo.java。代码如下：

```
public class SegDemo {  
  
    private static final String basedir =  
        System.getProperty("SegDemo", "data");  
  
    public static void main(String[] args) throws Exception {  
        System.setOut(new PrintStream(System.out, true, "utf-8"));  
  
        Properties props = new Properties();  
        props.setProperty("sighanCorporaDict", basedir);  
        props.setProperty("serDictionary", basedir + "/dict-  
            chris6.ser.gz");  
        if (args.length > 0) {  
            props.setProperty("testFile", args[0]);  
        }  
        props.setProperty("inputEncoding", "UTF-8");  
        props.setProperty("sighanPostProcessing", "true");  
    }  
}
```

```
CRFClassifier<CoreLabel> segmenter = new  
CRFClassifier<>(props);  
  
segmenter.loadClassifierNoExceptions(basedir + "/ctb.gz",  
props);  
  
for (String filename : args) {  
  
    segmenter.classifyAndWriteAnswers(filename);  
}  
  
  
String sample = "我住在美国。";  
  
List<String> segmented = segmenter.segmentString(sample);  
  
System.out.println(segmented);  
}  
}
```

第三章：文本分析

数据分析和数据挖掘是两个相互关联但又不同的概念。它们都是对数据集进行考察，但数据分析重点在于展现数据集的特征，如展现均值、中位数，数量大小等统计学特征；又或者将数据集可视化，通过展现数据集本身的特征，增加对数据集的了解。而数据挖掘的重点是从数据集合中发现知识、发现模式。这些知识和模式只有通过特殊的技巧和方法才能获得。例如，典型的啤酒和尿布的例子中，仅仅通过数据分析的方法不能获得啤酒和尿布之间的关联，而通过关联规则发现的算法才能洞察到这一有价值的信息，或称为知识。

面对文本数据对象，文本分析的工作我的理解就是对文档或文档集合本身所具有的特征的描述。通过一些方法来抽取文档的这些特征。这一章主要讲述面对单个文本，在无监督学习的情况下，在句子或词的级别上对单个文本进行的分析。展示单个文本的特性。

第一节：从文档抽取关键词

以下，我们将 key words（关键词）和 key phrase（关键短语）看做具有相同的含义的术语。

关键词抽取的任务是从一篇文档中抽取一个词项集合，这些词项最好的描述了这篇文档的内容或特征。关键词抽取是一个研究很广泛、全面的问题。许多研究将 key phrase 抽取看做是有监督学习的问题，针对有了语料库做训练集训练一个抽取模型，然后应用在新文本上来抽取关键词。本节我们面向实际应用，简化问题。在没有语料库的情况下抽取一篇文档的关键词。重点介绍下面的几种方法。

（1）词频

我们将问题转换成统计文档中的词频，排除停用词以后来获得文档的关键词集合。然后按照词频来挑选前 k 个词作为关键词。

问题：你们怎么来统计一篇文档的词频？

附件里给了两个 JAVA 文件，一个是从中文文档里按词频抽取关键词；一个是从 pdf 文档里按词频获取关键词。

(2) RAKE 算法

但只使用词来做为关键词还不够。更多时需要的是短语，即 keyphrase 关键短语。

RAKE (Rapid Automatic Keyword Extraction) 算法 (见 Text Mining: Applications and Theory 一书) 是一种简单易行的 keyphrase 抽取方法。这里的 keyphrase 可以是单个词或词组。RAKE 基于这样的观察，关键词频繁的包含多个词，而不会包含标点符号和停用词。

注：Python 有包实现了 RAKE 功能 (<https://pypi.python.org/pypi/python-rake>)，本讲义给出了它的 JAVA 实现版本。

RAKE 的工作原理如下：

RAKE 抽取 keyphrases 时将原始的文本分割成候选 keywords 集合。首先，将文本看做是一个长串。对于中文文档需要分词操作。这个长串然后被分割成序列的组合。分割原则是，**标点符号和停用词看做分割符**。例如，把“的，像”看做是停用词。句子“我们的祖国像花园”被分割成子序列的集合是：“我们”，“祖国”，“花园”

如此的操作将产生候选关键词集合。接着需要为候选关键词打分。每个候选的关键词确定后，根据候选关键词绘制 words 共现矩阵。例如，下面是从一个文档中抽取的候选 keywords 集合。

Compatibility – systems – linear constraints – set – natural numbers – Criteria – compatibility – system – linear Diophantine equations – strict inequations – nonstrict inequations – Upper bounds – components – minimal set – solutions – algorithms – minimal generating sets – solutions – systems – criteria – corresponding algorithms – constructing – minimal supporting set – solving – systems – systems

从 keywords 中 words 的共现绘制矩阵。

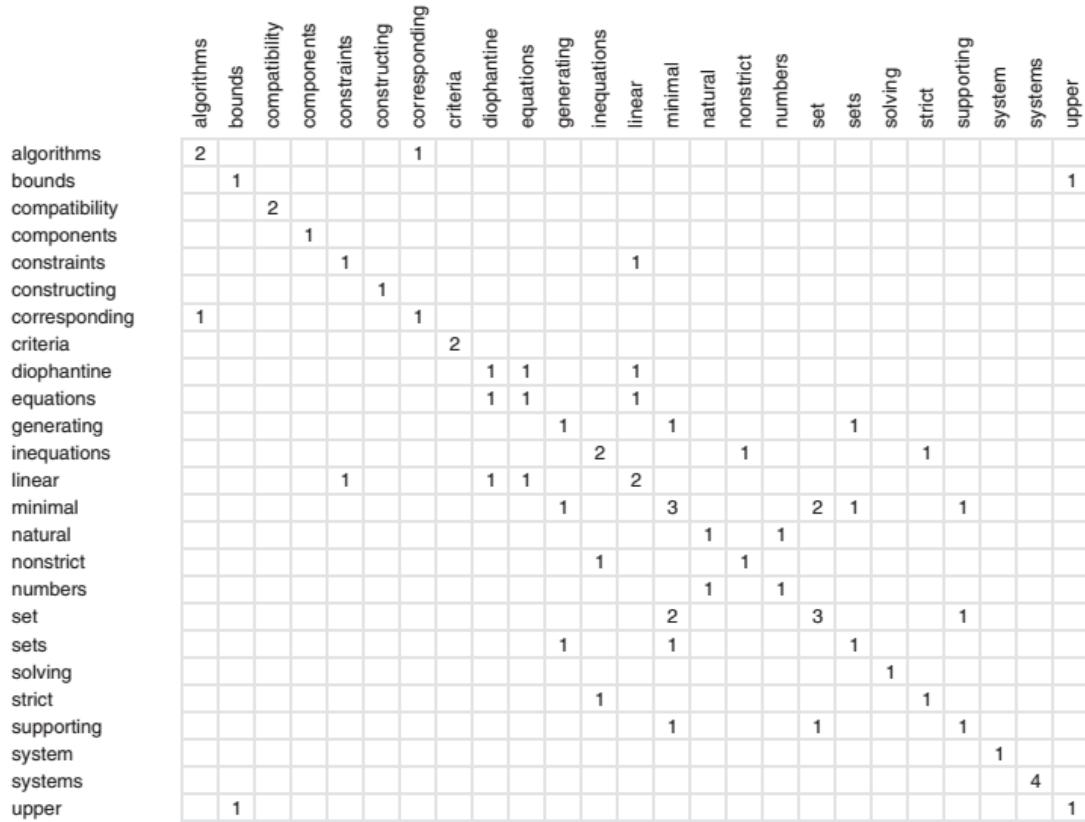


图 3-1 共现矩阵

矩阵中的对角线是一个词出现的次数，而其他位置的元素反映的是两个词的共现次数。

Keywords 的评分计算是词的度和词频的比率 $\text{deg}(w)/\text{freq}(w)$ 。词频 $\text{freq}(w)$ 就是一个词在对角线上的值。如果图 3-1 中词“algorithms”的词频是 2，词的度 $\text{deg}(w)$ 就是一个词的行向量的和。如“algorithms”的度是 3。图 3-2 显示了计算结果

	algorithms	bounds	compatibility	components	constraints	constructing	corresponding	criteria	diophantine	equations	generating	inequations	linear	minimal	natural	nonstrict	numbers	set	sets	solving	strict	supporting	system	systems	upper
$\text{deg}(w)$	3	2	2	1	2	1	2	2	3	3	3	4	5	8	2	2	2	6	3	1	2	3	1	4	2
$\text{freq}(w)$	2	1	2	1	1	1	1	2	1	1	1	2	2	3	1	1	1	3	1	1	1	1	4	1	
$\text{deg}(w) / \text{freq}(w)$	1.5	2	1	1	2	1	2	1	3	3	3	2	2.5	2.7	2	2	2	2	3	1	2	3	1	1	2

图 3-2 评分计算

这种计算方法的原则是，经常和其他词在 keywords 中共现的词，而又单独出现的次数少，这样的词应该很重要。根据此原则，计算每一个 keywords 的评分是它包含的所有 words 的 $\text{deg}(w)/\text{freq}(w)$ 评分的和。例如，上面的关键词的计算评分是

minimal generating sets (8.7), linear diophantine equations (8.5), minimal supporting set (7.7), minimal set (4.7), linear constraints (4.5), natural numbers (4), strict inequations (4), nonstrict inequations (4), upper bounds (4), corresponding algorithms (3.5), set (2), algorithms (1.5), compatibility (1), systems (1), criteria (1), system (1), components (1),constructing (1), solving (1)

但是实际应用中，用 RAKE 在中文文本中提取关键词效果不好，因为 RAKE 很大程度上靠停用词来划分短语，中文中使用的停用词没那么多，甚至一个句子中没有停用词。如此整个句子都做为 keyphrases。例如，

“全国居民人均可支配收入实际增长 8%”

整个句子没停用词，则整个句子作为了 keyphrases（这合理吗？）。另外 RAKE 算法累积一个 keyphrase 中所有词的 $\text{deg}(w)/\text{freq}(w)$ 评分。该计算方法则会鼓励长句子。我觉得不合理，所以我修改了别人实现的 RAKE。用下面的公式计算一个关键词 KP 的评分。

$$\text{Score(KP)} = \frac{1}{N} \sum_{w \in KP} \text{deg}(w) / \text{freq}(w)$$

N 是关键词 KP 中词的个数。

(3) TextRank

注：TextRank 有多种 Python 实现版本，本讲义实施了一个 JAVA 版本。

TextRank 是在 EMNLP2004 的一篇论文“TextRank: Bringing Order into Texts”中介绍的基于 PageRank 思想来实现关键词抽取的算法。它也是一种无监督的，基于单个文本的关键词抽取算法。该算法其实是把 PageRank 算法应用在了文本分析。我们先介绍一下 PageRank 算法。PageRank 最初由 Google 的创始人 Larry Page 发明。WWW 中，网页之间因为超链接而形成了互联的关系。这样的互联关系可以透露出网页的重要性。PageRank 算法就是从这样的互联关系中计算网页的重要性评分。这里简要介绍 PageRank，一些细节的讨论见第十章的 3.1 节。

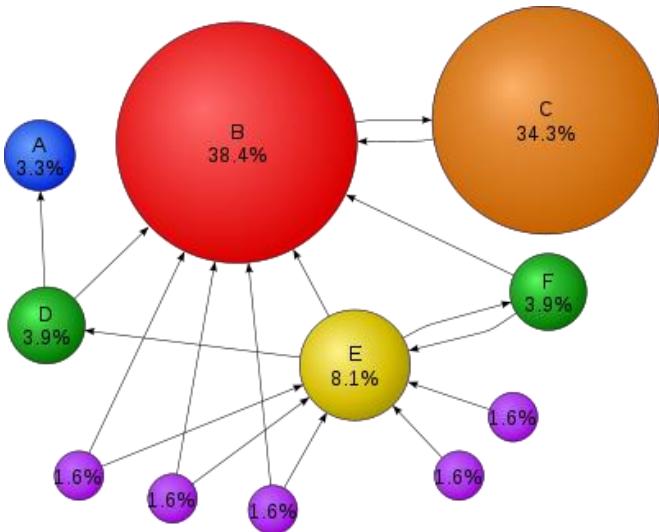


图 3-3 PageRank 计算示例

PageRank 的评分原理是，当 Web 冲浪者在 Web 上随机游走，他对某些节点的访问次数会比其他节点更多。PageRank 的思路就是，在随机游走过程中越频繁被访问的网页越重要。PageRank 算法基于下面的公式计算一篇网页的评分。

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

$PR(p)$ 是网页 p 的 PageRank 评分； d 是一个 damping 系数[0-1]； N 是 Page 数量； $M(p)$ 是链接到网页 p 的网页集合； $L(p)$ 是网页 p 的链出数量。

PageRank 算法如下：

```

PR0←E
loop:
    计算 PRi+1(p)
    g←||PRi||1-||PRi+1||1
    PRi+1← PRi+1+g*E
    delta←||PRi+1-PRi||1
    While delta<epsilon

```

E 是每个网页的初始得分，如可设置 $E=\{0.5,..,0.5\}$ ； PR_i 表示是在第 i 轮迭代中计算的 PR 评分； $||PR||_1$ 表示向量的 L1 范数。

$$||PR||_1 = \sqrt{|PR(1)| + \dots + |PR(N)|}$$

当把 PageRank 应用于文本分析时，步骤如下：

- (1) 将每个词作为网络的节点。
- (2) 确定词之间的关系，在有关系的两个词之间建立一条边。根据自己任务的需要这个图可以是有向图或无向图，有权重图或无权重图。
- (3) 使用 pagerank 算法在该图上为每个词计算一个评分。然后排序。根据需要选择前 k 个词作为 Keywords。
- (4) 进行关键词收缩 (collaps) 的操作。即从第三步的关键词列表中产生词组。作为 keyphrase。

对于第 (2) 步。如何确定两个词的关系，该文以一个窗口 (window) 内两个词的共现性来确定。例如，有一个短文本

“Matlab code for plotting ambiguity functions”

如果窗口大小是 4。从左到右一个词一个词的移动窗口，可以产生三组子串

“Matlab code for plotting”，“code for plotting ambiguity”，“for plotting ambiguity functions”

在每组子串中的任意一对词都是有关联的，都可以在网络的两个节点 (这一对词) 之间画一条边。

对于第 (4) 步。假设有一段文本

“Matlab code for plotting ambiguity functions”

如果在第 (3) 步挑选出的关键词列表中包含了两个词，“Matlab”，“code”。因为这两个词是有关系的 (假设我们以“邻近”作为两个词的关系)。就将它们合并成一个 keyphrase。得到 “Matlab code” 作为产生的 Keyphrase。

在我实际实施这个算法时，一些具体的实施细节如下：

- (1) 建立的是有权重的无向图。每个词作为了一个节点。只把相邻的两个词做为是有关联的词。即可以理解成窗口大小为 2。如果两个词多次出现相邻，累计共现次数作为网络的边的权重。两个词的关系是不分出现的前后次序的，所以是无向图。
- (2) 前面的 pagerank 算法中的评分是针对的有向无权重图。针对无向有权重图的计算评分公式是

$$R_{k+1}[i] = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} W(p_{ij}) \frac{R_k[j]}{W(p_j)}$$

$M(p_i)$ 是与节点 i 有链接的节点集合。 $W(p_{ij})$ 是节点 i 和 j 之间的边的权重。 $W(p_j)$ 是节点 p_j 的边的权重和。

- (3) 在进行 collapse 操作时，对于产生的前 K 个评分最高的 $terms=[t_1, \dots, t_K]$ 。产生所有可能的词对组合 $\langle t_i, t_j \rangle$ 。词对组合中词的顺序无关。如果节点 t_i 和 t_j 之间边的权重大于一个设定的阈值（如果 t_i 和 t_j 之间无边连接视为权重为 0）。则产生 keyphrase t_i, t_j 。

第二节：句子的语义分析

看我的笔记 2017/06/26

第三节：词的语义分析

在文档中某个词会和其他词存在语义关联的，我们称之为称为 word association，翻译做“词关联”。我们将被考察的词称为“目标词”，和目标词语义关联的词称为“关联词”。关联词的集合可以作为描述该词在文档中的语义。例如，在 2015 年人大政府报告中，抽取前 10 个和“习近平”这个词关联的词有{同志，总书记，党中央，贯彻落实，狠抓，系列，八项，重大，改革}。可以发现这些关联词提供了“习近平”这个词的语义描述。

关联词的最简单抽取方法可以找出和目标词在一个窗口中共现的频率最高的前 k 个词。还有研究采用了关联规则挖掘中常用的 Apriori 算法。

然而，还有一种情况。举个例子： $\{A, B\}$ 是在共现在一个窗口中的高频词对。它们被考虑为互为关联词。词对 $\{B, C\}$ 也是同样如此。但 $\{A, C\}$ 并不是这样的关联词。但如果 $\{A, B\}, \{B, C\}$ 的关联词之间的关联性很强，其实也可以说明 $\{A, C\}$ 之间存在着关联。按照社会网络分析的观点来理解，朋友的朋友也很可能是朋友。

因此，我提出一个在“词图”上发现关联词的方法。第一节中的 TextRank 算法，初始需要建立一个词的关联图，称为“词图”。词做为了图的顶点。如果一对词共现在一个

窗口内，则在“词图”上的两个顶点间绘制一条边。共线频率作为边的权重。通过在“词图”上的随机游走可以发现关联词。

图上的 t 步随机游走是两个节点间所有长度为 t 的路径的概率（两个节点间长度为 t 的路径数除以两个节点间的路径总数）。图上的随机游走包括两种：正向随机游走和逆向随机游走。正向随机游走，即从 j 点出发，在 k 点结束 t 步随机行走的概率 $P_{t|0}(k|j)$ （即已知 j ，所以 j 是条件； $t|0$ ，表示已知的是起点）；逆向随机游走，即当在节点 j 结束 t 步随机行走，起始点是 k 的概率 $P_{0|t}(k|j)$ 。其中 $P_{0|t}(k|j) \propto P_{t|0}(j|k)P_0(k)$ 。 $P_0(k)$ 是选择 k 点作为随机行走开始点的概率。我们的项目中，仅仅使用正向随机行走的方法。下面是计算 t 步随机游走转换概率的方法。

由无向权重连接图 G 获得一步随机行走转换概率矩阵 A ， A 是一个行随机矩阵，即 $\sum_j A_{ij} = 1$ 。 $A[j, k]$ 是从节点 j 到 k 的一步转换概率

$$P_{t+1|t}(k|j) = \begin{cases} (1-s)C_{jk} / \sum_{ij} C_{ji} & \forall k \neq j \\ s & k = j \end{cases}, \text{ 其中 } s \text{ 表示自转换概率, } C_{ji} \text{ 是节点 } j, i \text{ 之间的边的权重。}$$

求对角阵 Z ，其中 $Z_{jj} = \sum_i [A']_{ij}$ 。然后及时 t 步概率转换矩阵 $M \leftarrow A^t * Z^{-1}$

如此，矩阵 M 中的一个元素 $M[i, j]$ 描述的就是从节点 i 开始 t 步随机游走终止节点是 j 的概率。概率值越高，越说明词项 i 和词项 j 之间存在高的语义关联。一个计算一步转换概率的例子如下图：

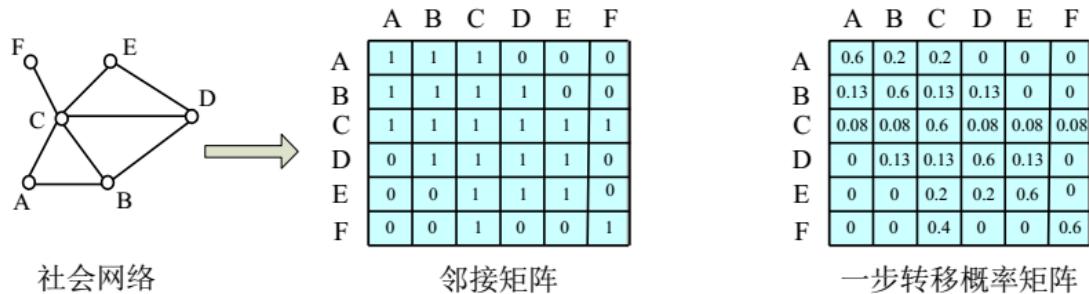


图 3-4 一步概率转移矩阵的计算

在具体实施时需要设置两个参数值：随机游走的步数理解为阶数，即如果只考察和核心词有直接关联的词，步数就等于 1；如果想考察朋友的朋友则步数就设为 2。在一个小的网络上随机游走步数太高，将会陷入到一个小的联系紧密朋友圈子里。因此，我们的实施中选择随机游走步数为 2 或 3。比较大的图选择 5-6。自转换概率设置为 0.4-0.5。

我对“关联词”的 JAVA 实施见课件。调用该程序的示例代码如下：

```

public static void main(String[] args) {
    Main m=new Main();
    m.setSteps(2);
    m.setSelf(0.5);

    /* 第一次调用了build程序后，除非是分析新的文档，可以第二次就不再
运行它就进行分析 */

//    m.buildEng("www2016.txt", 5, true, false);
//    m.buildChinese("rdbg2015.txt", 1);

    /* 分析 */
    m.get("习近平", 10);
}

```

Main.java 是调用该程序的界面。SetSteps 方法设置随机游走步数；setSelf 设置自转换概率。该程序给了两个方法 buildEng 和 buildChinese 来分别处理英文和中文文档。

public void buildEng(String fname, int min, boolean useStopList, boolean useStem);

buildEng 方法中的参数 fname 是待处理的文件名；min 是阈值。我们挑选词频大于 min 的词。useStopList 为 true 表示使用停用词表来过滤词项；useStem 为 true 表示要进行词干化处理。

public void buildChinese(String fname, int min)
buildChinese 处理中文文档，参数含义同上。

在第一次调用了 buildEng 或 buildChinses 方法后。t 步转换概率矩阵被串行化保存在磁盘上了。因此，想再运行程序分析别的词，不需要再重新建立 t 步转换概率矩阵。即第二次不用调用 buildEng 或 buildChinese 方法了。程序会读入串行化的 t 步概率转换矩阵，调用 get 方法在此基础上发现目标词的关联词。

public void get(String term, int k);

get 方法的两个参数 term 是目标词，k 是前 k 个和目标词语义最相关的词。

附：

下面是 python 的 word association 实现版本。需要安装 python 的中文分词包 jieba 和自然语言处理工具箱 NLTK

第四节：文本可视化:词云图

词云图即将关键词用图的方式展现出来的可视化方法。例如图 3-5，每个关键词会根据重要性或权重显示不同的大小。



图 3-5 词云图

词云图可以让用户很方便的了解信息。词云图的可视化工具很多，我们这里用 python 的包 word_cloud (https://github.com/amueller/word_cloud) 来绘制词云图。如果你在 windows 下安装该包遇到了问题，可以访问 <https://www.lfd.uci.edu/%7Eohlke/pythonlibs/#wordcloud>

使用已提供的 Wordcloud 的 XML 文件示例

网上的许多网站都提供了绘制词云图的功能。一个关于文本可视化的网站
<http://research.dbvis.de/text/>

安装 WordCloud 包如果让你安装 Visual C++ 的 build tools
请参考 [https://www.visualstudio.com/vs/installation/](#)

<http://landinghub.visualstudio.com/visual-cpp-build-tools>

http://amueller.github.io/word_cloud/网站有很多使用 wordcloud 绘制词云图的例子。

下面是绘制一个词云图的例子

```
#!/usr/bin/python3
```

```
# -*- coding:utf8 -*-
import codecs

from os import path
from wordcloud import WordCloud

d = path.dirname(__file__)

# Read the whole text.
text = open(path.join(d, 'example.txt'),encoding="utf8").read()
font = r'C:\Windows\Fonts\simsun.ttc'

# Generate a word cloud image
wordcloud = WordCloud(background_color="white",
                      prefer_horizontal=0.7,
                      font_path=font).generate(text)

# Display the generated image:
# the matplotlib way:
import matplotlib.pyplot as plt
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

该例子中，文件 example.txt 保存了文件。如果文件中包含中文，使用包 codecs。然后在打开文件时规定编码方式 encoding="utf8"。然后在创建 WordCloud 对象时，设定中文字体的路径 font_path=font。我们在变量 font 中给出了中文的仿宋字体 font = r'C:\Windows\Fonts\simsun.ttc'。wordcloud 的构造函数还有很多参数，详见 http://amueller.github.io/word_cloud/generated/wordcloud.WordCloud.html#wordcloud.WordCloud

练习：对比分析 2017 年和 2018 年的人大政府工作报告。把人大政府工作报告抽取关键词，用词云图进行展示。

第四章：信息检索系统

我们首先从一个问题开始。如果你的计算机上有一个文档集合（纯文本），你会怎样查找出包含了某个词的文档？一种简单的方法是依次打开每个文档，查找是否包含了某个词。但当面对文档集合很大，这样的查找操作频繁时，这种方法也是一种非常低效率的方法。

信息检索(Information Retrieval, 缩写 IR)这一术语始于二十世纪四十年代。当时，面对管理大量的科学文献的需求，需要有效的进行文档检索的技术。但当时计算机还没有诞生。信息检索的含义更多的是怎样手工为图书资料建立索引，便于用户通过索引找到相应的图书。现在信息检索这一术语的含义非常广泛，它的学术定义为“信息检索是从大规模非结构化数据（通常是文本）的集合（保存在计算机上）中找出满足用户需求的资料（文档）的过程”。我们首先了解一个简单的信息检索系统，布尔检索。

第一节: 布尔检索

一个最简单的信息检索系统构建步骤：

- (1) 将每篇文档分为一个个词条 (token)。这个过程成为词条化 (tokenization)。该过程还包括对词条的选择性处理，包括词干化，归一化，去除停用词等等。
- (2) 建立词典。
- (3) 建立词项-文档关联矩阵。纵向为词典中的词，横向为文档名称。下表是在莎士比亚文集上建立的一个词项文档矩阵的示例。

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

图 4-1. 词项-文档矩阵

(4) 用户提交一个查询 (query)，查询是表达用户需求的一个或多个词。比如，用户想查找关于苹果电脑价格的网页，则用户向信息检索系统提交的通常查询是词“苹果 电脑 价格”。这个查询（三个词）表达了用户的信息需求（检索需求）。

(5) 布尔检索。在图 1-1 的例子建立了莎士比亚文集的词项-文档关联矩阵，当提交了查询时，就会从词项-文档关联矩阵中抽取每个词对应的行向量。例如，想查找包含词项 Brutus 和 Caesar 而不包含词项 Calpurnia 的文档，其查询是 Brutus and Caesar and NOT Calpurnia。此处，and 是布尔逻辑运算符。那么可以抽取出 3 条行向量，三条行向量做逻辑运算即可得到，满足该查询的文档。

Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	0	0
Not Calpurnia	1	1	0	1	0	0
与操作	1	0	0	1	0	0

检索结果	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth

对信息检索系统性能的评价指标，即评价检索结果满足信息需求的程度，有很多。我们介绍两个在数据挖掘中也常用的指标：precision 查准率或正确率，recall 查全率或召回率。

Precision：返回的结果中真正和信息需求相关的文档所占的百分比。

recall：所有和信息需求真正相关的文档中被检索系统返回的百分比。

两个评价指标的图示如下：

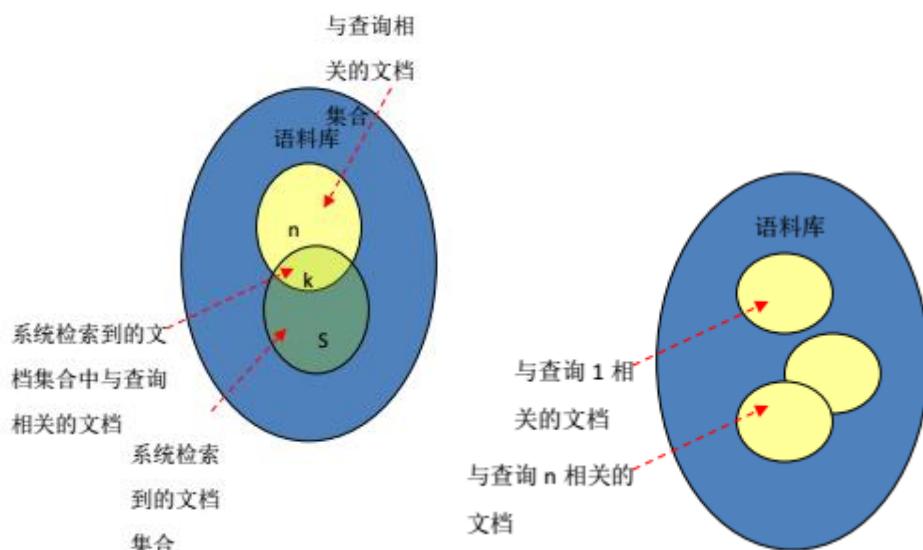


图 4-2 Precision 和 Recall 的图示说明

例子：已知一个语料库中有 m 篇文档，有一个查询 query，语料库中已知和该查询有关的文档有 n 篇。现在我们实施了一个信息检索系统。该系统查找到了 s 篇文档，其中 k 篇和 query 相关。Precision= k/s ；Recall= k/n

信息检索系统在做测评时需要给出准备好的“语料库”、“查询”。语料库中和查询相关的文档需要做标注。

查准率和查全率两个评价指标称为信息检索测评中的“无序检索集合的评价”。而更为重要的评价指标是“有序检索结果的评价”。例如，搜索引擎中，检索结果中的和查询最相关的文档总是排在最前面。具体的“有序检索”评价指标有 MAP， $p@k$ 等。信息检索系统的测评是一个重要的研究方向，我们这里不做过多的讨论。相关内容参见“信息检索导论”。

第二节：倒排索引

回到第一节的例子，检索一个语料库，有文档 100 万篇，而词项的个数是 50 万，可以试想一下，为了进行信息检索建立的词项-文档关联矩阵有多大？它的缺点和优点是什么？

观察词项-文档矩阵，我们可以发现该矩阵是高度稀疏的。即大部分的元素是 0，极少部分元素是 1。对上面的语料库粗略做个计算，由于每篇文档的平均长度是 1000 个单词，所以 100 万篇文档在词项-文档矩阵中最多对应 10 亿个 1，而在这个语料库上这个文档词项矩阵的大小是 $100 \text{ 万} * 50 \text{ 万} = 5000 \text{ 亿}$ 。也即 $1 - 1/500 = 99.8\%$ 的元素是 0。因此转换存储方式，只保存元素为 1 的信息将大大减小保存语料库信息所付出的空间开销。

上述思路引出信息检索中的一个核心概念：倒排索引（inverted index）。这里倒排的概念是建立从词项到文档的映射。其数据结构如图 4-3

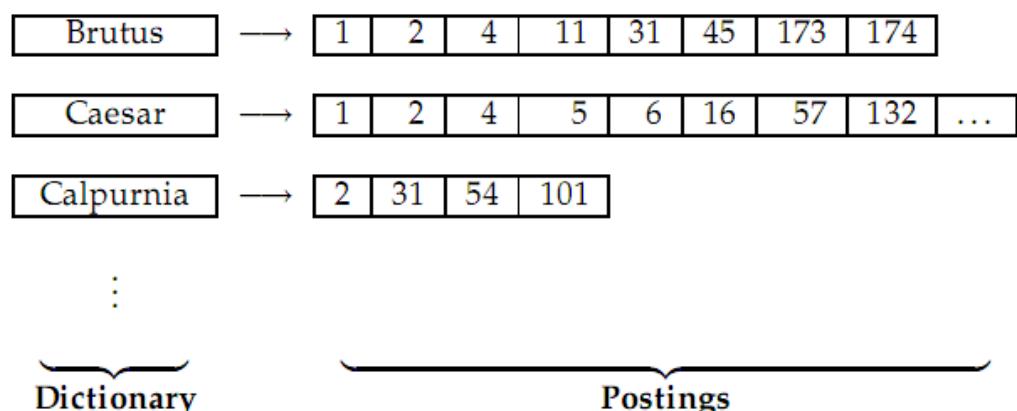


图 4-3 倒排索引

左边是词项词典 (lexicon, 有时也称作 dictionary, vocabulary)，右边是记录每个词项在文档中出现了一次的文档编号的列表，称作倒排记录表 (posting list)，表中的元素称作倒排记录 (posting)。词典按照字母顺序排序，倒排记录表按照文档编号进行排序。

问：从该图我们可以看出倒排索引比词项-文档关联矩阵的优点在哪里？

从语料库建立倒排索引的过程（原理）如下：

1. 文档集合中的每篇文档被分配唯一编号。

2. 对每篇文档产生一个预处理的（归一化）的词条列表，其中每个元素是一个二元组（词项和文档编号），如图 4-4 所示。

term	docID
I	1
did	1
enact	1
julius	1

图 4-4 产生词条列表

3. 将所有词条列表合并，然后排序（此处我们假设每个词条都是词典中的词项）。

term	docID	term	docID
I	1	ambitious	2
did	1	be	2
enact	1	brutus	1
julius	1	brutus	2
caesar	1	capitol	1
I	1	caesar	1
was	1	caesar	2
killed	1	caesar	2
i'	1	did	1
the	1	enact	1
capitol	1	hath	1
brutus	1	I	1
killed	1	I	1
me	1	i'	1
so	2	it	2
let	2	julius	1
it	2	killed	1
be	2	killed	1
with	2	let	2
caesar	2	me	1
the	2	noble	2
noble	2	so	2

图 4-5 排序后的词条列表

4. 合并后的词条列表中有许多相同的词项，他们表达了同一个词项在不同文档中的出现。为每个唯一词项建立一个链表（倒排记录表），该链表记录了每个词出现在文档中的文档编号。

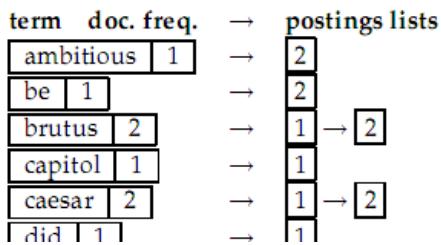


图 4-6 产生倒排索引

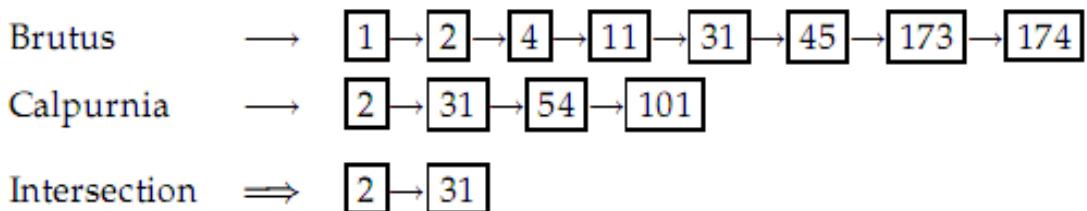
在词项-文档矩阵上非常方便进行布尔检索。那在倒排记录表上怎么进行布尔检索？

布尔检索模型是多个查询词项的布尔逻辑操作集合。例如，用户提交了一个布尔查询 Brutus AND Calpurnia。其含义是查找含有 Brutus 和 Calpurnia 两个词的文档。

倒排索引上的查询操作过程如下：

在词典 (Lexicon) 中找到这两个词，获得这两个词的倒排记录表。

其后的逻辑与操作，是求两个倒排记录表的交集。这里有专门的算法在两个链表中求交集。



两个倒排记录表的求交集算法，有时称为链表的合并 (merge) 算法：

```

INTERSECT( $p_1, p_2$ )
1  $answer \leftarrow \langle \rangle$ 
2 while  $p_1 \neq NIL$  and  $p_2 \neq NIL$ 
3 do if  $docID(p_1) = docID(p_2)$ 
4     then ADD( $answer, docID(p_1)$ )
5          $p_1 \leftarrow next(p_1)$ 
6          $p_2 \leftarrow next(p_2)$ 
7     else if  $docID(p_1) < docID(p_2)$ 
8         then  $p_1 \leftarrow next(p_1)$ 
9     else  $p_2 \leftarrow next(p_2)$ 
10 return  $answer$ 

```

示例如图 4-7：

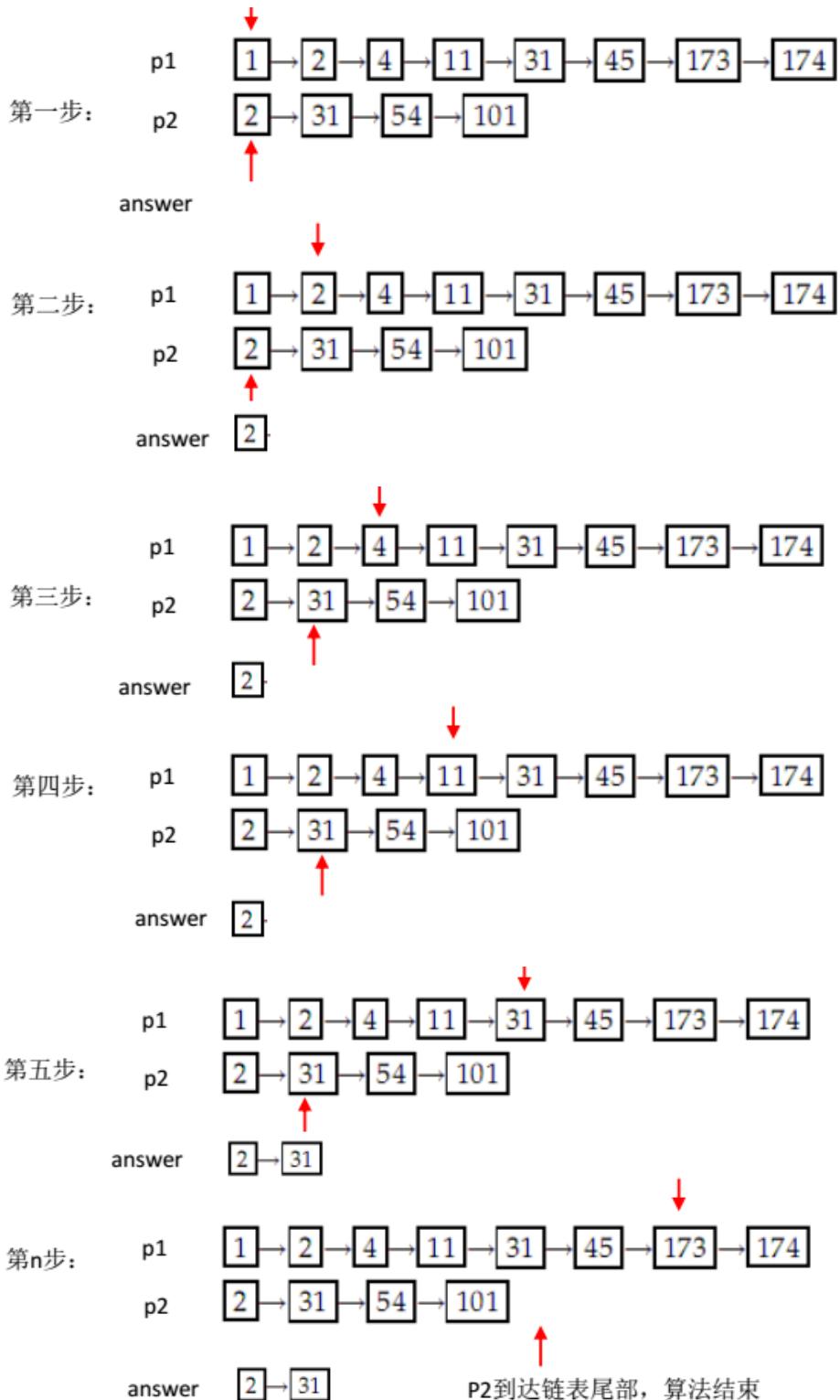


图 4-7 倒排记录表求交集算法

问：该算法可以求简单的“与操作”的查询。“或操作”的查询怎么处理？如，Brutus OR Caesar。“非操作”的查询怎么处理？如，Not Calpurnia

在信息检索系统中，为了进行短语查询，需要对倒排索引进行扩展。有多种扩展倒排索引的方法，我们这里仅介绍常用的位置信息索引，即建立倒排索引时，保存了词项出现在文档中的位置信息。如图所示。

```
to, 993427:  
  ( 1, 6: ⟨7, 18, 33, 72, 86, 231⟩;  
   2, 5: ⟨1, 17, 74, 222, 255⟩;  
   4, 5: ⟨8, 16, 190, 429, 433⟩;  
   5, 2: ⟨363, 367⟩;  
   7, 3: ⟨13, 23, 191⟩;... )  
  
be, 178239:  
  ( 1, 2: ⟨17, 25⟩;  
   4, 5: ⟨17, 191, 291, 430, 434⟩;  
   5, 3: ⟨14, 19, 101⟩;... )
```

图 4-8 保存了词项出现在文档中的位置信息的倒排记录表

一条倒排记录包含了词项出现在文档和文档中具体位置的信息。例如，词项 to 在文档 1 出现了 6 次，分别是在文档的位置⟨7, 18, 33, 72, 86, 231⟩。

在具有位置信息的倒排索引上进行短语查询。例如查询短语 “to be”。首先，获得两个词的倒排记录表，然后按照前面讲述的倒排索引合并算法，合并两个记录表。在合并过程中，如果两个词共同出现在了某个文档中，则在位置记录信息中查找是否有正确顺序的相邻词条位置。有则该文档属于检索结果。

```
to: ⟨..., 4: ⟨..., 429, 433⟩; ... ⟩  
be: ⟨..., 4: ⟨..., 430, 434⟩; ... ⟩
```

注：此处显示的信息和上图有点差别，此次的 4 是文档编号；没有给出词项出现在文档中的次数。

布尔检索模型普遍的问题是：采用 AND 操作符产生的结果查准率虽然高但是查全率偏低。而采用 OR 操作符的查全率虽然高但是查准率低。不可能找到一个折中方案。

布尔检索模型只是记录词项存在不存在，但是我们往往需要获得检索到的文档和需求是否相关的可信度。布尔检索结果返回的是无序的文档集，但我们需要对返回的结果排序。有序检索模型（ranked retrieval model）不是通过具有精确语义的逻辑表达式来构建查询，往往采用一个或多个词来构成自由文本查询（free text query）。该模型需要确认哪篇文档最能满足用户的需求，然后按照文档重要性排序后的结果返回给用户。

在搜索引擎中都是采用有序检索模型。

第三节：向量空间模型

1. tf-idf 词项权重的计算

在布尔检索中，给定一个布尔查询，一篇文档要么满足查询要么不满足。在文档集规模很大的情况下，满足布尔查询结果的文档数量可能非常多，往往会大大超过用户能够浏览的文档的数目。因此对于信息检索系统来说，对文档进行评分排序很重要。用户可以根据评分（文档和查询的相关度）进行检索。例如，搜索引擎就是这么做的。

布尔检索模型只考虑了词项在文档域中出现与否的情况。如果我们想更准确的给文档评分，需要换一种方式来描述文档（原始的文档描述就是词的集合）。粗略的说，文档内容（以下均简称为文档）中出现频率越高的词项，越能描述该文档（不考虑停用词）。因此可以统计每个词项在每篇文档中出现的次数，即词项频率，记为， $tf_{t,d}$ t 为词项， d 为文档。获得文档中每个词的 tf 权重，一篇文档则转换成了词-权重的集合，通常称为词袋模型（bag of words model）。我们用词袋模型来描述一篇文档。词袋的含义就是说，像是把一篇文档拆分成一个一个的词条，然后将它们扔进一个袋子里。在袋子里的词与词之间是没有关系的。因此词袋模型中，词项在文档中出现的次序被忽略，出现的次数被统计。例如，“a good book” 和 “book good a” 具有同样的意义。

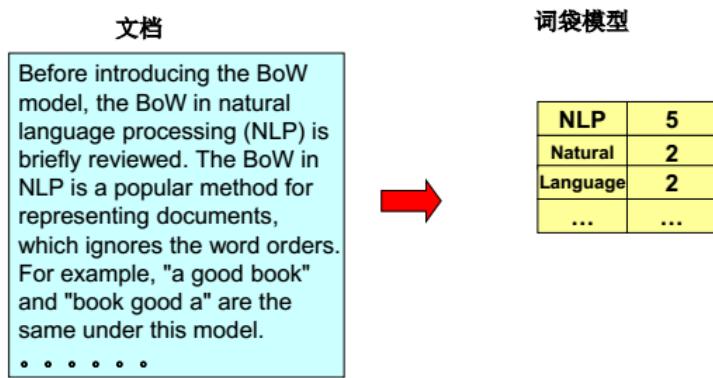


图 4-9 词袋模型

词袋模型中只为文档中的词计算了 tf 权重。tf 权重只考虑词在文档中出现的频率。如果一个词，只在某篇文档中出现，而没在文档集合中的其他文档中出现，则该词可以很好的区分描述这篇文档，则应该给该词更高的权重。例如，在描述汽车的文档集合中，几乎每篇文档都会出现 car, auto 这样的词，这样的词不具有区分描述能力也即，我们想计算词项的权重时也考虑是否该词具有很好的描述性。

因此，又会统计一个词项的文档频率 df ，即在文档集合中，出现该词项的文档的数目。在实际应用中会采用逆文档频率 idf 。

$$idf_t = \log \frac{N}{df_t}$$

N 是文档集合中的文档数。可以发现，一个词如果在文档集合中出现的次数越少，它的 idf 得分越高。

现在文档中的每个词可以计算两个权重 tf 和 idf ，单凭哪一个权重来衡量一个词项的重要性很多时候并不合适（注：也有的应用只单独根据 tf 或 idf 来作为权重）。所以，用 $tf-idf$ 权重来表示词项的权重。

$$tfidf_{t,d} = tf_{t,d} \times idf_t$$

词项的 $tf-idf$ 权重的含义如下：

- (1) 一个词在少数几篇文档中多次出现，它的权重越大（此时对文档能够提供最强的区分能力）
- (2) 但词项在一篇文档中出现次数少，或者在很多文档中出现，权重取值次之。
- (3) 如果词项在所有文档中都出现，那么它的权重值最小（为什么？）

如果把一篇文档看做是向量，其中每个分量都对应词典中的一个词项，分量是 $tf-idf$ 计算出的权重值。某词项在文档中没有出现，其对应的分量为 0。我们就可以用一个向量来描述文档。一系列文档在同一向量空间中表示，就称为向量空间模型（vector space model）VSM。同一向量空间的含义是，所有文档中相同位置的分量（元素）所对应的词是相同的。

	Doc1	Doc2	Doc3
car	0.88	0.09	0.58
auto	0.10	0.71	0
insurance	0	0.71	0.70
best	0.46	0	0.41

前面讲了，信息系统中很重要的是对检索的文档进行排序，即计算查询和文档的相关程度。在向量空间模型上，计算每篇文档和查询的相关程度（评分）的方法之一：重合度评分指标。即查询中所有词在一篇文档中 $tf-idf$ 权重值求和。

$$Score(q, d) = \sum_{t \in q} tfidf_{t,d}$$

t 是查询 q 中的词项

思考：

(1) 公式 $\text{idf}_t = \log \frac{N}{df_t}$ 中 log 运算的底是怎样影响评分计算的 $scoer(q, d) = \sum_{t \in q} \text{tfidf}_{t,d}$?

(2) log 运算的底是怎样影响两个文档在相同的一条的查询上的相对评分的？即会不会影响两篇文档的排序变化？

2. 余弦相似度

前面提出了文档向量的概念。其中每个分量代表词项在文档中的相对重要性。一系列文档在同一向量空间的表示称为 VSM (Vector Space Model)。VSM 是词袋模型。向量空间模型是信息检索、文本分析中基本的模型。通过该模型，可以进行有序文档检索、文档聚类、文档分类等。当然，现在的研究有新发展。出现了很多模型代替 VSM。

每篇文档在 VSM 中用向量表示，那么计算两篇文档的相似度自然的想到用两个向量的差值。但是，可能存在的情况是。如果两篇相似的文档，由于文档长度不一样。他们的向量的差值会很大。

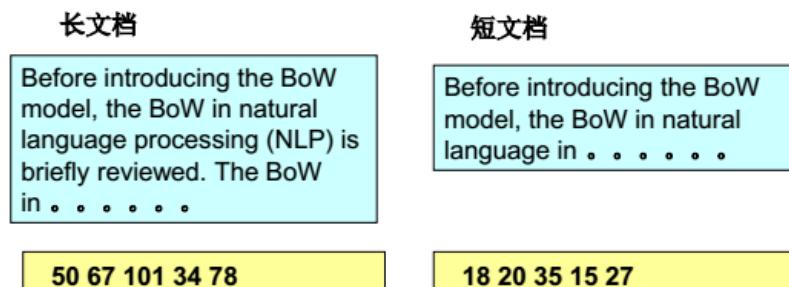


图 4-10 长文档和短文档的差异

用重合度评分指标就没有考虑文档长度的影响。余弦相似度是使用的非常广泛的计算两个向量相似度的公式，它可以去除文档长度的影响。

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)| |\vec{V}(d_2)|}$$

公式中 $\vec{V}(d_1)$ 和 $\vec{V}(d_2)$ 是文档 d_1 和 d_2 用向量形式的描述。 $\vec{V}(d_1) \cdot \vec{V}(d_2)$ 是两个向量的点积（内积）。两个向量的内积定义为

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^M x_i y_i$$

公式的分母是两个向量的欧几里得长度。文档 d 的向量表示为 $\vec{V}(d)$ ，它是一个 M 维向量 $\vec{V}_1(d) \dots \vec{V}_M(d)$ ， $\vec{V}_M(d)$ 是向量中的一个元素。文档 d 的欧几里得长度是 $\sqrt{\sum_{i=1}^M \vec{V}_i^2(d)}$

如果把每篇文档的向量除以该文档的欧式长度，得到的就是欧式归一化结果。也即前页的向量相似度公式是两个归一化向量的点积。

$$\text{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2)$$

该值也称作是两个向量的余弦夹角。因此，我们可以看到余弦相似度的计算是抵消了文档长度的影响。**向量余弦夹角相似度，不考虑向量的长度，只考虑两个向量的余弦夹角 $\cos(\theta)$ 大小。**

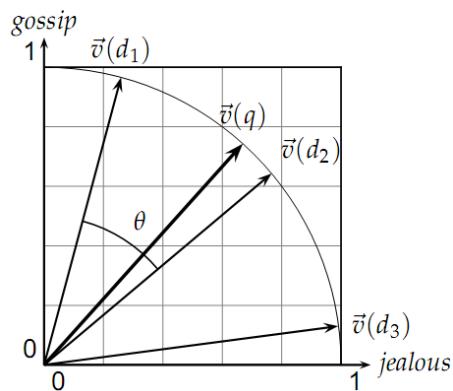


图 4-11 向量的余弦夹角

注：可以证明如果把查询向量和文档向量都规范化到单位向量（即前面讲余弦相似度提到的，将每个向量除以它的欧几里得长度（L2）范数，就是将向量规范化到单位向量），那么采用欧式距离计算查询和文档的距离，然后用该距离进行文档排序的检索结果和余弦相似度是一样的。（欧式距离 $|\vec{x} - \vec{y}| = \sqrt{\sum_{i=1}^M (x_i - y_i)^2}$ ）。因为单位向量之间的距离就是上面的一个圆上的两个节点间的距离。可以看到圆上两点的距离和向量的夹角是等价的。

3. 在 VSM 上的检索

给出一条查询 q ，为了在向量空间模型上进行检索（有序检索），需要将查询也转换成和文档集空间统一的向量。将查询看做是一个很短的文档，得到它在 VSM 相同空间下的向量。就可以根据查询和文档的余弦相似度来产生有序检索结果。

要得到查询 q 的向量 $\vec{V}(q)$ ，其中分量的值可以是 tf、或 idf 或 tf-idf 权重。

$$\text{score}(q, d) = \frac{\vec{V}(q) \cdot \vec{V}(d)}{|\vec{V}(q)| |\vec{V}(d)|}$$

用余弦相似度计算查询和文档向量的相似度与用重合度评分指标 $\text{score}(q, d) = \sum_{t \in q} t f i d f_{t,d}$ 进行计算两者有什么区别呢？

(1) 余弦相似度去除了文档长度的影响。

(2) 如果查询向量的词项频率都是 1，文档向量做了欧式归一化后使用重合度评分指标。则等价于余弦相似度的计算。

理想情况，我们计算查询和文档的相似度时，查询和文档都是保存在向量中，则可以直接计算。

词项文档矩阵

	Doc1	Doc2	Doc3	q
car	0.88	0.09	0.58	0
auto	0.10	0.71	0	1.3
insurance	0	0.71	0.70	2.0
best	0.46	0	0.41	3.0

但在实际中，采用的是倒排索引的数据结构。倒排索引中保存了词项在相应文档中的词频 tf。头部保存了 idf。如图 4-12 所示。

包含词权重的倒排索引

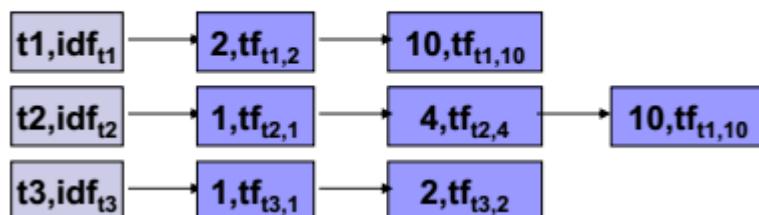


图 4-12 包含词权重的倒排索引

即逻辑上，我们为语料库建立了向量空间模型是矩阵形式。物理上，其实是按照倒排索引存储在词频上的。（注：有些软件就是用矩阵相似来保存 VSM，例如 R 的文本挖掘包 tm。当然，如此就限制了能处理的语料库的大小）。

在倒排索引上计算查询和文档的余弦相似度时，例如，查询 q 和文档 Doc3。找出每个查询词项的倒排记录表。将查询词项的权重和倒排记录表中对应的权重相乘。并利用一个辅助数组累加保存计算结果。例如，我们假设文档集合一共有 10 篇文档，建立的辅助数组的大小就是 10（下图的示例中假设数组的下标是从 1 开始）。相乘的结果就保存在对应的数组单元。

算法描述如下：

```

COSINESCORE( $q$ )
1 float Scores[N] = 0
2 Initialize Length[N]
3 for each query term  $t$ 
4 do calculate  $w_{t,q}$  and fetch postings list for  $t$ 
5 for each pair  $(d, tf_{t,d})$  in postings list
6 do Scores[d] += wft,d × wt,q
7 Read the array Length[d]
8 for each  $d$ 
9 do Scores[d] = Scores[d] / Length[d]
10 return Top K components of Scores[]

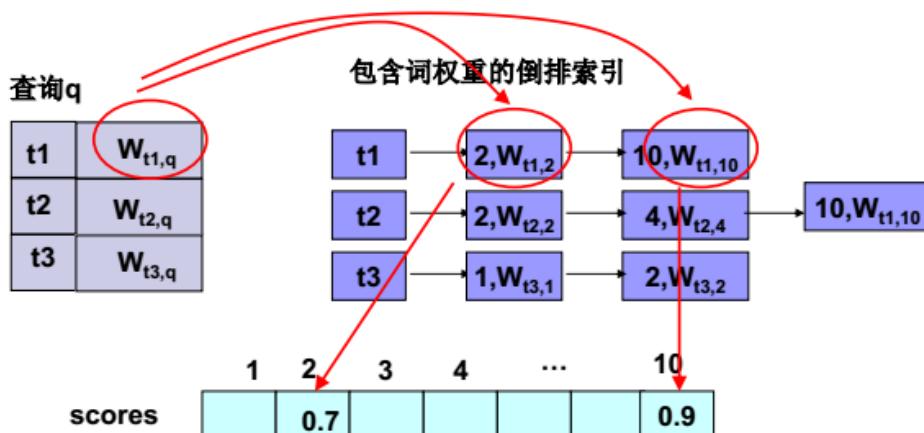
```

数组 length 存储的是每篇文档的长度。最后，返回评分最高的 k 篇文档。

从上面的计算过程，我们可以看到，它的计算余弦相似度计算时是除以的 L1 范数

$\sqrt{\sum_{i=1}^M |v_i(d)|}$ 。而不像前面的余弦相似度公式除以的是 L2 范数（欧几里得长度）；还有就是没有对查询向量进行规范化。这样的处理有下面的两点考虑：（1）如果要计算一篇文档的 L2 范数，代价会比较高，而计算 L1 范数就简单了，就是文档的长度。除以 L1 范数或 L2 范数对检索的排序结果也没有本质影响。（2）因为检索系统其实只关心排序的结果，当用一个查询和所有的文件计算相似度的时候，查询向量的欧式长度是一个定值，sim(q, d_1) 和 sim(q, d_2) 中，都除以或者不除以 $\|q\|_2$ 不会影响排序结果，因此也就没必要计算了。

算法中的第 4-6 步可以用图 4-13 解释工作原理。



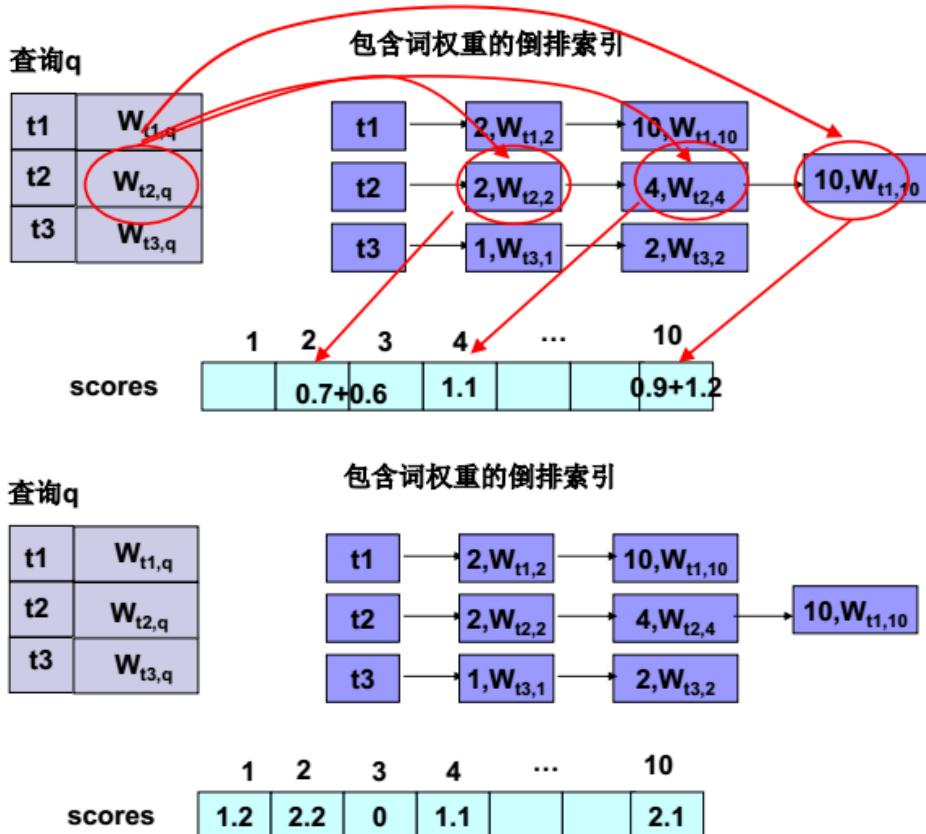


图 4-13 算法的图示

当计算下一个查询词项和对应文档中的词项的权重乘积后，在对应位置和原来已有的值进行累加。最后的辅助数组 score 中的值就是文档向量相对于查询向量 q 的点乘的和。但此时还没有做归一化。如果把向量做归一化，就得到了余弦相似度。

4. 其他的 tf-idf 权重计算方法

对每篇文档的每个词项赋予一个权重，除原始的 tf 及 tf-idf 之外还有很多其他方法。

(1) tf 的非线性尺度变换方法

显而易见，即使一个词项在文档中出现了 20 次，它所携带的信息的重要性也不可能是只出现 1 次词项的 20 倍。一个常用的修改 tf 权重的方法是计算原始词项频率的对数函数。

$$wf_{t,d} = \begin{cases} 1 + \log tf_{t,d} & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

则 tf-idf 权重变成 wf-idf 权重

$$wf\text{-}idf_{t,d} = wf_{t,d} \times idf_t$$

(2) 基于最大值的 tf 规范化

另一种 tf 权重归一化方法是，对于一篇文档，采用文档最大词项频率对所有词项的频率进行归一化。

$$ntf_{t,d} = a + (1 - a) \frac{tf_{t,d}}{\max_{t \in d}(tf_{t,d})}$$

公式中， a 是阻尼系数取值[0,1]。经验值取 0.4。阻尼系数其平滑 (smoothing) 作用，主要是抑制后一部分的贡献。也即，使用平滑技术来保证规范化引起的 $ntf_{t,d}$ 不会剧烈波动。我们发现长文档中词项反复出现的可能性大，所以长文档中的词项频率倾向于取更大的值。这显然不公平，而最大 tf 规范化方法的主要思路就是减少这种不公平带来的影响。我们采用余弦相似度来计算两个向量的相似度时已经从一方面抑制文档长度带来的影响；最大 tf 规范化是从另一个角度来抑制长文档带来的计算异常。基于最大值 tf 规范化使用时会受以下情况的影响。

(1) 停用词表的变化将引起词权重的显著变化，从而也造成文档排序的变化。这是因为，因为停用词表的变化将使得一些文档中最高 tf 值发生改变，而另一些文档又没有改变。因此不同文档的 tf 值，就发生相对变化，从而影响向量相似度计算结果。从而造成文档排序变化。从这个意义上来说，该方法不稳定。

(2) 某篇文档可能包含一个异常词项，它的出现次数非常多，但是它并不代表文档内容。如果这个异常词项不属于停用词，它将其他词的 tf 降低很多。检索时，因为该词不代表该文档，则与该文档相关的查询中，该文档的得分会很低。而有该词的检索将会把该文档排序在很靠前，但该文档又不和该查询相关。

(3) 词项分布均匀的文档和不均匀的文档应该区别对待。因为词项分布均匀的文档，在每项查询中的得分都会差不多。

常用的 tf, df 计算方法和归一化方法

Term frequency		Document frequency		Normalization	
n (natural)	$tf_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(tf_{t,d})$	t (idf)	$\log \frac{N}{df_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - df_t}{df_t}\}$	u (pivoted unique)	$1/u$ (Section 6.4.4)
b (boolean)	$\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^\alpha, \alpha < 1$
L (log ave)	$\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$				

第四节：概率检索模型和 BM25

概率检索模型最早于 1960 年提出，经过几十年的发展，概率检索模型已经从理论走向实际应用。基于概率检索模型的 OKPI 检索系统在多次 TREC 中取得了优异的成绩，另一个概率检索系统 INQUERY 也有不错的声誉。本节先介绍经典的二值独立概率模型（Binary Independence Retrieval Model, BIM），然后介绍在此基础上发展而来的 BM25 模型。BM25 可以称为是最成功的文本检索模型之一。概率模型是指的一类模型，我们仅介绍 BIM 和 BM25。

补充知识：

贝努力分布：考虑一个二值随机变量 $x \in \{0,1\}$ 。 x 可以描述了投掷一枚硬币， $x=1$ 表示正面朝上， $x=0$ 表示反面朝上。正常的硬币正面朝上或反面朝上的概率是 0.5，现在假设该硬币有破损，概率为 μ 。可以得到 $p(x=1|\mu) = \mu$ ； $p(x=0|\mu) = 1 - \mu$ 。因此 x 的概率分布可以写成

$$\text{Bern}(x|\mu) = \mu^x (1 - \mu)^{1-x}$$

这就是贝努力分布。

当我们有 K 个硬币，每个只投掷一次，能观察到每给硬币都朝上的概率是

$$\text{Bern}(x_1, \dots, x_K | \mu)$$

这是多元贝努力分布。如果满足每个事件是独立的，即第 i 个硬币投掷和第 j 个硬币的投掷是两个不相关的事件，则

$$\text{Bern}(x_1, \dots, x_K | \mu) = \prod_{i \in K} \mu_i^{x_i} (1 - \mu_i)^{1-x_i}$$

μ_i 是第 i 个硬币正面朝上的概率； $x_i = 1$ 是第 i 个硬币硬币朝上。当然如果两个事件是相关的，则多元贝努力公式就很复杂，不在本文讨论范围。

二项式分布：当我们想计算投掷一个硬币 N 次，能观察到 m 次正面朝上的概率。我们可以计算

$$\text{Bin}(m|N, \mu) = \binom{N}{m} \mu^m (1 - \mu)^{N-m}$$

其中

$$\binom{N}{m} \equiv \frac{N!}{(N-m)!m!}$$

这称作二项式分布。

概率排序原理

现在有一个排序式检索系统，我们考虑这个系统对文档的检索是二值的，即查询 q 和一篇文档 d 的相关性 $R_{d,q}$ 要么是相关的 $R_{d,q} = 1$ ，或者不相关 $R_{d,q} = 0$ 。 $R_{d,q}$ 简记为 R 。如果进行有序检索，于是可以利用概率模型来估计每篇文档和查询的相关概率 $P(R=1|d,q)$ ，然后按概率对文档进行排序，这是概率排序原理 PRP 的基本原理。

为了在实际中使用 PRP，则需要对 $P(R=1|d,q)$ 进行估计。二值独立模型 BIM (binary Independence Model) 提供了对 $P(R|d,q)$ 进行估计的方法。**BIM 中二值等于布尔值，即文档 d 表示为文档向量 x ，词项 t 出现在文档 d 中， $x_t=1$, 否则 $x_t=0$** 。此时的文档向量是二值向量，而不是有权重的向量。在 BIM 中假设各词项在文档中的出现是独立的。虽然现实中不是独立的。但 BIM 通常可以得到满意的结果。

我们假设，每篇文档的相关性和其他文档无关。计算 $P(R=1|x,q)$ 利用贝叶斯定理可得

$$\begin{aligned} P(R=1|\vec{x}, \vec{q}) &= \frac{P(\vec{x}|R=1, \vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})} \\ P(R=0|\vec{x}, \vec{q}) &= \frac{P(\vec{x}|R=0, \vec{q})P(R=0|\vec{q})}{P(\vec{x}|\vec{q})} \end{aligned}$$

该公式的推导过程如下：

$$P(R=1|x, q) = \frac{P(R=1, x, q)}{P(x, q)} = \frac{P(x|R=1, q)P(R=1|q)P(q)}{P(x|q)P(q)}$$

继续可以得到

$$\begin{aligned} P(R=1|\vec{x}, \vec{q}) &= \frac{P(\vec{x}|R=1, \vec{q})P(R=1|\vec{q})}{P(\vec{x}|\vec{q})} \\ P(R=0|\vec{x}, \vec{q}) &= \frac{P(\vec{x}|R=0, \vec{q})P(R=0|\vec{q})}{P(\vec{x}|\vec{q})} \end{aligned}$$

其中 $P(\vec{x}|R=1, \vec{q})$ 是当检索返回一篇相关文档时，文档表示为 x 的概率。 $P(\vec{x}|R=0, \vec{q})$ 是当返回一篇不相关文档时，文档表示为 x 的概率。我们可以这样理解这段话：有一个文档空间，有很多文档。和查询 q 相关的文档中包括 x 。现在给定查询 q ，返回相关文档 x 的概率就是 $P(x|R=1, q)$

$P(R=1|\vec{q})$ 对于查询 q 返回一篇相关文档的先验概率。 $P(R=0|\vec{q})$ 对于查询 q 返回一篇不相关文档的先验概率。

计算 $P(R=1|\vec{x}, \vec{q})$ 需要许多其他的概率值。它们不可能精确计算，需要进行估计。通常通过实际文档集分布的统计特性来估计。如果知道文档集合中相关文档的百分比

m ，就可以估计 $P(R=1|q)=m$, $P(R=0|q)=1-m$ 。另外对于某个查询，一篇文档或者和它相关或者不相关，因此 $P(R=1|x,q)+P(R=0|x,q)=1$

给定查询，按照二值独立模型进行检索时就是计算 $P(R = 1|\vec{x}, \vec{q})$ 。由于检索系统关心的是文档的相对次序，所以并不需要直接估计出这个值，而是采用其他更容易计算的排序函数，只要保证排序函数和直接计算概率所得到的文档次序一致即可。优势率（odds）公式如下：它提供了一种反映概率如何变化的放大器

$$O(A) = \frac{P(A)}{P(\bar{A})} = \frac{P(A)}{1 - P(A)}$$

文档相关性的优势率定义如下：

$$O(R|\vec{x}, \vec{q}) = \frac{P(R = 1|\vec{x}, \vec{q})}{P(R = 0|\vec{x}, \vec{q})} = \frac{\frac{P(R=1|\vec{q})P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|\vec{q})}}{\frac{P(R=0|\vec{q})P(\vec{x}|R=0,\vec{q})}{P(\vec{x}|\vec{q})}} = \frac{P(R = 1|\vec{q})}{P(R = 0|\vec{q})} \cdot \frac{P(\vec{x}|R = 1, \vec{q})}{P(\vec{x}|R = 0, \vec{q})}$$

因为 $P(R = 1|\vec{q})$ 对于查询 q 返回一篇相关文档的先验概率。 $P(R = 0|\vec{q})$ 对于查询 q 返回一篇不相关文档的先验概率。两个都是常数，它们是不影响排序的。因此可以把它忽略掉。我们只需要估计出

$$\frac{P(\vec{x}|R = 1, \vec{q})}{P(\vec{x}|R = 0, \vec{q})}$$

就可以使用优势率 $O(R|x, \vec{q})$ 对文档进行排序。

现在引入条件独立性假设，即给定查询的情况下，文档中一个词的出现与否是和其他词相互独立的。即设 A、B 是两个词， $P(AB)=P(A)P(B)$ 。当然实际中，不是这样。例如，一篇文档中出现了“检索”，那出现“信息”这个词出现的概率就很高

$$P(\vec{x} | R = 1, \vec{q}) = P(x_1, \dots, x_M | R = 1, \vec{q})$$

在条件独立的假设下

$$\frac{P(\vec{x}|R = 1, \vec{q})}{P(\vec{x}|R = 0, \vec{q})} = \prod_{t=1}^M \frac{P(x_t|R = 1, \vec{q})}{P(x_t|R = 0, \vec{q})}$$

即将求“给定一个查询当返回一篇相关文档时，文档表示为 x 的概率”，转换为求“当给定一个查询返回一个相关词项 x_t 的概率”，文档 x 中所有词项的概率乘积

因此有

$$O(R|\vec{x}, \vec{q}) = O(R|\vec{q}) \cdot \prod_{t=1}^M \frac{P(x_t|R=1, \vec{q})}{P(x_t|R=0, \vec{q})}$$

由于每个 x_t , 要么为 0 要么为 1 (因为按照前述, **BIM 中二值等于布尔值, 即文档 d 表示为文档向量 x , 词项 t 出现在文档 d 中, $x_t=1$, 否则 $x_t=0$**)。即给定查询 q 检索相关文档, 词项 $x_t=1$ 表示出现在了相关文档中, $x_t=0$ 表示没出现在相关文档中。所以有

$$O(R|\vec{x}, \vec{q}) = O(R|\vec{q}) \cdot \prod_{t:x_t=1} \frac{P(x_t=1|R=1, \vec{q})}{P(x_t=1|R=0, \vec{q})} \cdot \prod_{t:x_t=0} \frac{P(x_t=0|R=1, \vec{q})}{P(x_t=0|R=0, \vec{q})}$$

令 $p_t = P(x_t = 1|R = 1, \vec{q})$ 表示词项出现在一篇相关文档中的概率;

$u_t = P(x_t = 1|R = 0, \vec{q})$ 表示词项出现在一篇不相关文档中的概率;

$1 - p_t = P(x_t = 0|R = 1, \vec{q})$ 表示词项 x_t 没有出现在一篇相关文档的概率;

$1 - u_t = P(x_t = 0|R = 0, \vec{q})$ 表示词项 x_t 没有出现在一篇不相关文档的概率。

为简化做假设, 没有在查询中出现的词项在相关或不相关文档里出现的概率相等。可以简化成

$$O(R|\vec{q}, \vec{x}) = O(R|\vec{q}) \cdot \prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \cdot \prod_{t:x_t=0, q_t=1} \frac{1-p_t}{1-u_t}$$

$x_t=q_t=1$ 的含义是词项 x_t 在文档中出现了也在查询中出现了。即我们只需考虑在查询中出现的词项。

还有的推导步骤我们不详细阐述, 参见《信息检索导论》

我们可以得到用于排序的度量值, 称为检索状态值 RSV。一篇文档 d 的 RSV 是

$$RSV_d = \log \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} = \sum_{t:x_t=q_t=1} \log \frac{p_t(1-u_t)}{u_t(1-p_t)}$$

我们定义

$$c_t = \log \frac{p_t(1-u_t)}{u_t(1-p_t)} = \log \frac{p_t}{(1-p_t)} + \log \frac{1-u_t}{u_t}$$

则一篇文档 d 的 RSV 排序得分是

$$RSV_d = \sum_{x_t=q_t=1} c_t$$

由此可知，只要计算出查询词项的权重 c_t 。就可以进行查询排序。也即我们需要估计估计 pt 和 ut 值。

给定一个查询和文档集合，如果不知道相关文档有哪些，有下面的一些估计方法。

假设相关文档只占所有文档的极小一部分，那么可以通过整个文档集的统计数字来计算与不相关文档有关的量。本来 $u_t = (df_t - s) / (N - S)$ 。在该假设下，不相关文档中出现词项 t 的概率是。 $u_t = df_t / N$ 。于是有

$$\log[(1 - u_t) / u_t] = \log[(N - df_t) / df_t] \approx \log N / df_t$$

因此，我们可以用 idf_t 来代替 $\log \frac{1-u_t}{u_t}$

对于 pt ，可以采用如下估计方法：

(1) 如果知道某些相关文档，可以采用基于概率的相关反馈方法 (11.3.4，不讲)

(2) 有人提出用常数替代 p_t ，Greiff 证明更合理的计算 p_t 方法

$$p_t = \frac{1}{3} + \frac{2}{3} df_t / N$$

问题：(信息检索导论的习题 11.2)

What are the differences between standard vector space tf-idf weighting and the BIM probabilistic retrieval model (in the case where no document relevance information is available)?

我的思考：

1. 当只考虑 pt 为常数时，实际上的 BIM 是一个只考虑了逆文档频率的检索模型。

$$RSV_d = \sum_{x_t=q_t=1} c_t \quad ct = \text{常数} + idf_t$$

2. 向量空间模型采用的 tf-idf 权重计算。

3. 标准向量空间模型采用余弦相似度来计算查询向量和文档向量的相似度。BIM 用出现在一篇文档中的查询词项的权重和来计算。

概率检索模型发展

对于一个概率检索模型来说，对所需概率的合理近似需要一些重要假设。BIM 模型中这些假设包括：

(1) 文档、查询及相关性的布尔表示

(2) 词项的独立性

(3) 查询中不出现的词项不会影响最后的结果

(4) 不同文档的相关性之间是互相独立的

因为这些假设，检索很难达到较好水平。概率模型如果不知道部分的相关性信息，只能得到一个较差的由词项权重方式构成的模型。当 90 年代 BM25 模型出现后，概率检索模型出现了改观。现在概率检索模型和向量空间检索模型差别不大，只是在评分时，不是采用 tf-idf 权重和余弦相似度，而是采用由概率论得出的评分公式。有人已经通过引入概率模型的词项权重计算公式来改造向量空间模型 IR 检索系统。并取得了很好的效果。

BM25 模型是现在概率检索模型的基础。许多文献检索的研究做实验时，都以 BM25 做测试基础。BIM 模型最初为较短的编目记录和长度大致相当的摘要所设计。但是对现在的全文搜索文档集合来说，模型应该重视词项频率和文档长度（习题 11-2 证明 BIM 只采用了 idf）。

BM25 模型是基于词项频率、文档长度等因子来建立概率模型的一种方法，并且不会引入过多的模型参数。对于文档 d ，最简单的文档评分方法是给文档中的每个查询词项仅仅赋予一个 idf 权重。

$$RSV_d = \sum_{t \in q} \log \frac{N}{df_t}$$

通过引入词项频率和文档长度，可以将上述公式修改为

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{df_t} \right] \cdot \frac{(k_1 + 1)tf_{td}}{k_1((1 - b) + b \times (L_d / L_{ave})) + tf_{td}}$$

L_d 和 L_{ave} 是文档 d 的长度和整个文档集的平均长度。 k_1 是一个取正值的参数，用于对文档中的词项频率进行缩放控制。 $k_1=0$ 则对应 BIM 模型。 k_1 取较大值则对应于取原始词项频率。 $b(0 <= b <= 1)$ 决定文档长度的缩放程度。 $b=1$ 表示基于文档长度对词项进行完全的缩放； $b=0$ 表示不考虑文档的长度。

如果查询很长，对于查询词项也可以采用类似的权重计算方法。

$$RSV_d = \sum_{t \in q} \left[\log \frac{N}{df_t} \right] \cdot \frac{(k_1 + 1)tf_{td}}{k_1((1 - b) + b \times (L_d / L_{ave})) + tf_{td}} \cdot \frac{(k_3 + 1)tf_{tq}}{k_3 + tf_{tq}}$$

tf_q 是词项 t 在查询 q 中的权重。由此可知，此时的查询中如果有词项多次出现此公式才有意义。

对于参数 k_1 、 k_3 、 b 的估计通常是在一个已知查询、相关文档的训练集上。对参数进行估计。也有实验表明， k_1 和 k_3 的合理取值区间是 1.2~2， b 取的 0.75。

BM25F 是由微软 Cambridge 研究院在参加 Trec 2004 的 Web Track 竞赛时针对 web 文档检索的任务提出。一篇 Web 文档可以被分为多个 Field (域)。如 Title、Body。

首先累积一个词项在所有 Field 上的权重

$$weight(t, d) = \sum_{c \text{ in } d} \frac{occurs_{t,c}^d \cdot boost_c}{((1 - b_c) + b_c \cdot \frac{l_c}{avl_c})}$$

L_c 是域的长度， avl 是域 c 的平均长度； b_c 是与域 c 相关的一个系数； $Boost_c$ 是应用到该域 c 的提升因子； $Occurs_{t,c}^d$ 是词项 t 在文档 d 的 c 域的词频。

查询 q 和文档 d 的相关性评分是

$$R(q, d) = \sum_{t \text{ in } q} \frac{weight(t, d)}{k_1 + weight(t, d)} \cdot idf(t)$$

当 $k_1=0$ ，则是基于逆文档频率的排序。 K_1 调节 $weight(t, d)$ 对排序的影响。

lucene 实施了 BM25 模型

第五节：统计语言模型

统计语言模型（或简单的称为语言模型）是从词汇表中抽取的字符串到概率的映射函数。它可以为一个词的序列计算概率分布。（换做我的语言，语言模型为词汇表中的每个词分配了一个概率值。满足，所有词的概率值的和为 1。一元语言模型是 $p(w)$ ，二元语言模型是 $p(w_1|w_2)$ ）。例如，语言模型可以为下面的三个句子计算概率值

$$p(\text{"Today is Wednesday"}) = 0.001$$

$$p(\text{"Today Wednesday is"}) = 0.000000001$$

$$p(\text{"The equation has a solution"}) = 0.000001$$

一个语言模型是语境相关的。例如，上面的语言模型中，“The equation has a solution” 比 “today is Wednesday” 有更小的概率。这阵子我们的日常对话语境，

这是合理的。但是，我们的语境如果是在讨论数学领域的问题。“The equation has a solution”这句话应该出现的更频繁。

给定了一个语言模型，我们可以抽样词的序列，产生文本。在这个意义上，语言模型又称作文本的生成模型（The generative model for text）。

为什么语言模型是有用的？一个通常的回答是，它提供了一个方法对自然语言的不确定性进行定量。

最简单的语言模型是一元语言模型（unigram language model），即我们假定一个词序列中每个词的产生是独立的。因此这个词序列的概率是每个词的概率的乘积。我们描述一个词序列的概率，形式上，设 V 是词汇表； w_1, \dots, w_n 是一个词的序列， $w_i \in V$ 是一个词。可以得到

$$p(w_1 \dots w_n) = \prod_{i=1}^n p(w_i)$$

那么我们给定一个一元语言模型 θ ，产生两个不同的文档 D_1 和 D_2 的概率是不同的，即 $p(D_1|\theta) \neq p(D_2|\theta)$ 。

$$p(D|\theta) = p(w_1 \dots w_n|\theta) = \prod_{i=1}^n p(w_i|\theta)$$

那什么样的文档有更高的概率呢？直觉上应该是包含了很多语言模型 θ 上的高概率的词的文档。在这个意义上 θ 的高概率词，指示了的 θ 的话题 topic。例如，从图 4-14 中的两个语言模型，一个话题是关于 text mining 一个是关于 health。

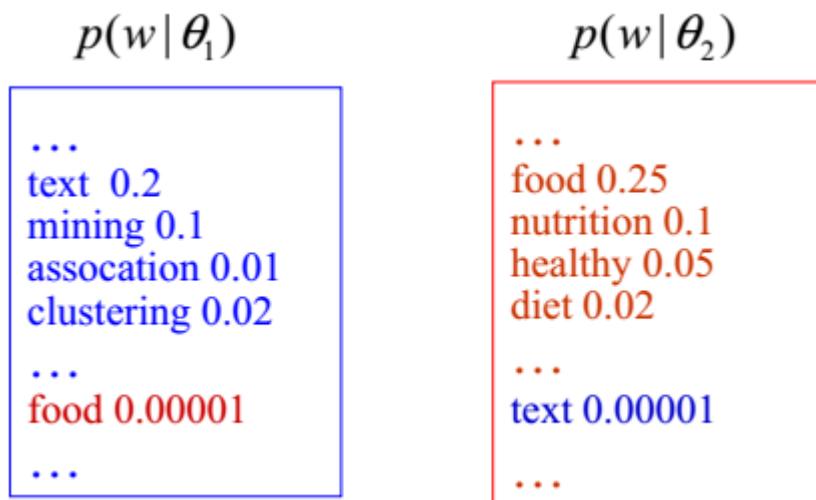


图 4-14 两个语言模型

如果 D 是一篇关于 text mining 的论文，我们可以预计 $p(D|\theta_1) > p(D|\theta_2)$ 。

现在假设我们有一篇文档 D (例如一篇关于 text mining 的摘要) , 它是用一元语言模型 θ 生成的。假设我们不知道 θ , 那么我们可以推理出 θ , 即基于 D 估计每个词 w 的概率 $p(w|\theta)$ 。这是一个标准的统计问题。一个通用的方法是最大似然法 (Maximum Likelihood) , 它寻找一个可以给观察到的数据 D 最高似然 (即最好的解释了这个数据) 的模型 $\hat{\theta}$ 。

$$\hat{\theta} = \operatorname{argmax}_{\theta} p(D|\theta)$$

一元语言模型的 ML 估计中 , 给每个 word 计算一个概率 , 它等于

$$p(w|\hat{\theta}) = \frac{c(w, D)}{|D|}$$

$c(w, D)$ 是词 w 在文档 D 中的计数 , $|D|$ 是文档的长度。 (这个公式的推导见 Chengxiang Zhai 的 Statistical Language Model for IR)

因为 ML 估计试图尽可能的拟合数据 , 如果数据是小样本 , 则可能发生过拟合。它会给予任何没出现在文档里的词一个概率 0。但这是不合理的。调整 ML 估计以便避免概率为 0 的情况 , 称为平滑 smoothing。平滑的方法很多可以参考 Chengxiang Zhai 的这本书。最简单的是上面公式的分子 $c(w, D)$ 加一个数 , 如 1 , $1/2$, 甚至更小的数。

一元语言模型强调每个词的产生是独立的 , 这在实际情况并不合理。比 n 元语言模型假定当前词的出现是更前 $n-1$ 个词的出现相关的 , 它可以捕获更多的词之间的依赖。二元语言模型的定义如下

$$p(w_1 \dots w_n) = p(w_1) \prod_{i=2}^n p(w_i | w_{i-1})$$

它可以捕获两邻近的词的依赖关系。

当然语言模型越复杂 , 它的计算代价就越高 , 我们在实际应用中需要对此作出权衡。而且数据稀疏会导致复杂的语言模型参数估计的不精确。迄今为止 , 一元模型对于信息检索已经是很有效的了。一个原因就是信息检索的任务相对于其他自然语言理解的任务是相对容易的。

查询似然模型

下面我们介绍一个语言模型在信息检索上应用的例子 , 查询似然模型。更多的基于统计语言模型的检索模型参看 Chengxiang Zhai 的书。

补充知识 :

1. 二项式分布

设一次试验的成功概率是 μ ， N 次试验成功 m 次的概率是一个二项式分布

$$Bin(m | N, \mu) = \binom{N}{m} \mu^m (1 - \mu)^{N-m}$$

$$\binom{N}{m} = \frac{N!}{(N-m)!}$$

是规范化因子

2. 多项式分布

设 x 是一个 K 维向量，满足

$$\sum_{k=1}^K x_k = 1$$

μ 也是一个 K 维向量， μ_k 是 $x_k=1$ 的概率。每次试验会得到 x 的一种可能取值。多项式分布描述了 N 次试验中， K 维向量中的元素各种可能出现次数的一个联合概率分布，其中

$$\sum_{k=1}^K m_k = N$$

多项式分布描述为

$$Mult(m_1, \dots, m_k | N, \mu) = \binom{N}{m_1, \dots, m_k} \prod_{k=1}^K \mu_k^{m_k}$$
$$\binom{N}{m_1, \dots, m_k}$$

是规范化因子

以投掷色子为例：

一个色子有 6 面。用 6 维向量 x 描述投掷色子的一次结果，满足

$$\sum_{k=1}^K x_k = 1$$

投掷色子 N 次，其中有 m_1 次 1 面朝上， m_k 次 k 面朝上的概率如下。设色子均匀则 $\mu_k = 1/6$ 。多项式分布为

$$Mult(m_1, \dots, m_k | N, \mu) = \binom{N}{m_1, \dots, m_k} \prod_{k=1}^K \mu_k^{m_k}$$

将文本考虑成一个向量，向量中每个分量是一个词项 t 的出现次数。在语言模型中已经描述了每个词项被抽取的概率 $P(t)$ 。那么描述一个文档被产生的概率分布是一个多项式分布。

$$P(d) = \frac{L_d!}{tf_{t_1,d}!tf_{t_2,d}!\dots tf_{t_M,d}!} P(t_1)^{tf_{t_1,d}} P(t_2)^{tf_{t_2,d}} \dots P(t_M)^{tf_{t_M,d}} \quad (4-1)$$

该多项式模型表达的含义是：一个文本中所有词项 t ，以 $P(t)$ 的抽取概率被生成出 tf_t 次的联合概率分布，即 $P(d)$ 是一个文本 d 被产生的概率。

IR 中最早使用也是最基本的语言模型是查询似然模型。在该模型中，我们对文档集中的每篇文档 d 构建其对应的语言模型 M_d 。检索的目标是将文档按照其与查询相关的概率 $P(d|q)$ 排序。

$$P(d|q) = P(q|d)P(d)/P(q)$$

为了求解 $P(d|q)$ ，将 $P(d)$ 看做是均匀分布的。因为，我们不是要求 $P(d|q)$ 的绝对值，而是求 $P(d_1|q)$ 、 $P(d_2|q)$ 的相对值。所以，可以忽略 $P(d)$ 。

均匀分布意味着

$$tf_{t_1,d_1} \approx, \dots, \approx tf_{t_n,d_1}$$

$$tf_{t_1,d_m} \approx, \dots, \approx tf_{t_n,d_m}$$

将 tf 按文档长度归一化

$$P(d_1) \approx P(d_m)$$

另外对于每篇文档， $P(q)$ 是相同的，也可以忽略。因此，最后的文档排序按照查询似然 $P(q|d)$ 来排序即可。它是在文档 d 对应的语言模型下生成查询 q 的概率 $P(q|M_d)$

因此，基于语言建模的 IR 方法实际上是在对查询的生成过程建模：首先每篇文档对应一个语言模型，然后计算查询被视为每个语言模型的随机抽样样本的概率，最后根据这些概率对文档排序。

$$P(q | M_d) = K_q \prod_{t \in V} P(t | M_d)^{tf_{t,q}}$$

该公式即套用多项式分布公式 4-1 而得到。即在一个文档语言模型 M_d 上，词汇表中每个词 t 在查询中产生 $tf_{t,q}$ 次的联合概率。

因为 K_q 是常数，且

$$\prod_{t \in V} P(t | M_d)^{tf_{t,q}} = \prod_{t \in q} P(t | M_d)$$

因此在最大似然估计，及一元语言模型假设的情况下，给定文档 d 的 LM M_d 产生查询 q 的概率是

$$P(q | M_d) = \prod_{t \in q} P_{mle}(t | M_d) = \prod_{t \in q} \frac{tf_{t,d}}{L_d}$$

但是该公式有个问题：当查询 q 中的词项未出现在文档中，计算出的概率值将为 0。这并不合理，因此需要采用一些平滑操作方法。如下，将基于文档的多项式分布和基于全部文档集估计出的多项式分布混合。

$$\hat{P}(t|d) = \lambda \hat{P}_{mle}(t|M_d) + (1 - \lambda) \hat{P}_{mle}(t|M_c)$$

M_c 是基于全部文档集构造的语言模型， λ 是平滑系数。

因此，基本查询似然模型是

$$P(q|M_d) = \prod_{t \in q} \lambda \hat{P}_{mle}(t|M_d) + (1 - \lambda) \hat{P}_{mle}(t|M_c)$$

最后，给出查询 q 的检索排序函数的定义如下

$$P(d|q) \propto P(d) \prod_{t \in q} \lambda \hat{P}_{mle}(t|M_d) + (1 - \lambda) \hat{P}_{mle}(t|M_c)$$

其中

$$P(t|M_c) = \frac{tf_{t,c}}{L_c}$$

$$P(t|M_d) = \frac{tf_{t,d}}{L_d}$$

平滑操作的作用实际上不仅仅是避免“估计”的问题。它可以带来检索性能的提高。

《Statistical Language Model for IR》一书的 3.4 节做了分析。结论是：经过变体，平滑的查询似然模型形式上类似 BM25。它惩罚文档频率高的词项，考虑了文档长度因素。

第六节：专题：使用 Lucene 构建文本检索系统

最新 lucene 版本是 7.2。由于 Lucene 每次更新后的前后版本变化有些大。我的课程使用 5.5 版本。我们只用 lucene 的 lucene-core-5.5.0.jar、lucene-queryparser-5.5.0.jar 和 analyzer.jar 库。但 lucene 还有很多功能扩展库，完成不同的功能，例如如果想建立具有 snippet 和高亮功能的搜索引擎，可以使用 highlighter.jar。

lucene 主要的类（分布在我们要用的那三个包中）

Analysis：转换文本到可索引和可搜索的 token

Document：对于一个用于索引和搜索的文档的逻辑描述

Index：维护和访问索引

Queryparser：查询分析器

Search：在索引上检索

Search.function：文档评分的程序控制

一个完整的检索系统如图图 4-15：

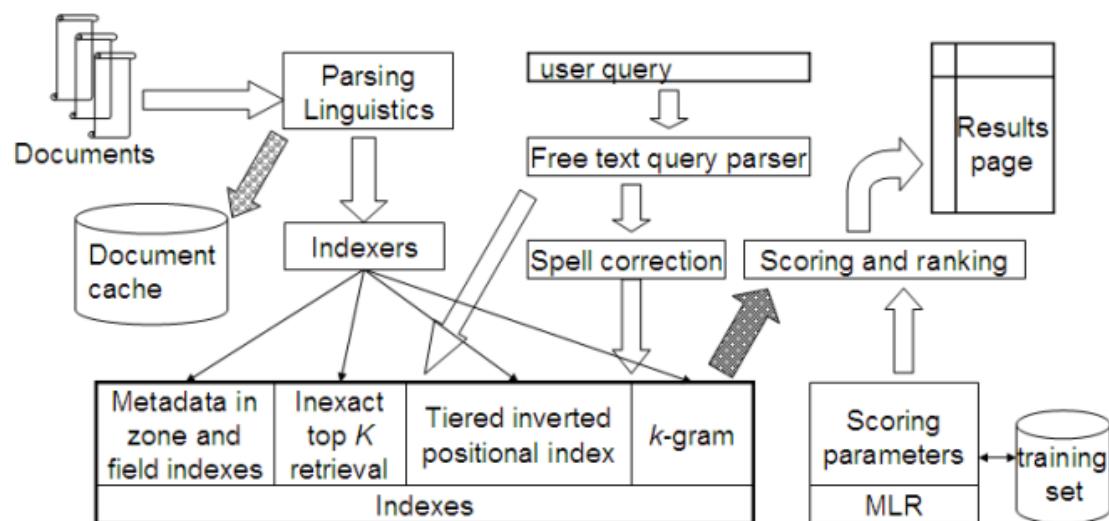


图 4-15 一个信息检索系统

使用 Lucene 建立一个 IR 系统的步骤如下：

(1) 建立分析器；(2) 基于分析器建立索引；(3) 在索引上进行检索。

1. 分析器

在构建信息检索系统的最初需要使用分析器 (analyzer) 对文档进行处理。检索时也需要使用相同的分析器对查询进行处理。Lucene 主要的分析器有：

(1) **StopAnalyzer**：可以去除停用词，同时完成小写转换。

(2) **StandardAnalyzer**：除了包含 StopAnalyzer 的功能，还可以完成数字、字母、IP 地址以及中文字符的分析处理（没有中文分词）

(3) **SimpleAnalyzer**：是一个具备西文字符词汇分析的分析器。处理时以非字母字符作为分隔符号。分析器不做词汇过滤。输出的词项完成小写转换。

(4) **WhitespaceAnalyzer**：使用空格符作为分隔符的分析器，不过滤、不做小写转换

(5) **KeywordAnalyzer**：把整个输入作为一个单独的词项，方便特殊的文本进行索引和检索。

(6) 我们前面使用的 **IKAnalyzer** 其实就是一款为 Lucene 处理中文准备的。我们直接构建 IKAnalyzer 类的对象，就可以建立 lucene 的分析器。Lucene 自带的 smartcn 也可以创建中文的 analyzer（不知道为什么，我使用 smartcn 建立索引检索时效果不好）。

在 Analysis 包下提供了各种分析器和构成分析器的各种方法，如分词、词干化等。用户也可以根据自己的需要构建自己的 Analyzer

创建分析器的方法很简单，例如只需要一条语句

```
Analyzer textAnalyzer=new SimpleAnalyzer();
```

注意，使用完分析器后应该关闭它

```
analyzer.close();
```

3. 建立索引

索引结构

我们先看一下 lucene 中索引的结构。

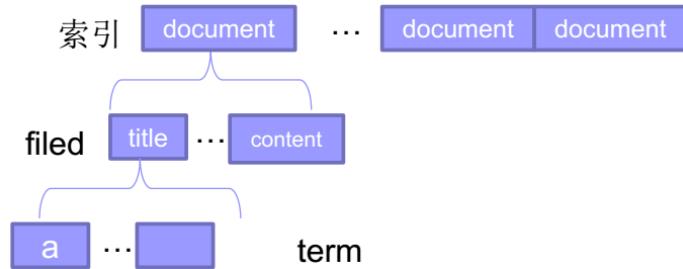


图 4-16 Lucene 的索引结构

Lucene 的索引文件中，一个索引是一个 document 序列；一个 document 是一个 fields 序列；一个 filed (域) 是一个赋予了名称的 term 序列；一个 term 是一个字符串，即词项。上图中的 filed 序列中的“title”, “content”是赋予的 filed 的名称。Field 的含义是，一个被索引的文档，还可以包含很多“域”，例如，文档标题、文档路径、文档正文或“作者”。Lucene 可以为这些域单独索引。如此，在检索时，可以根据正文进行检索，也可以根据标题，作者等进行检索。另外，我们也可以在检索结果 (document 对象的集合) 中抽出相应的域，如，根据正文 field 进行检索，我们得到了 document 对象，我们可以抽取出该文档所在的路径，标题等 (如果我们在建立索引时为一篇文档建立了正文，标题和路径的 field 域) 。

Lucene 的索引保存了正向信息。 即保存了从索引，一直到词的包含关系：索引(Index) → 文档(Document) → 域(Field) → 词(Term)。**也保存了反向信息，即采用倒排索引**，因此 lucene 可以列出包含一个 term 的文档。Lucene 实际上有个子索引 (segment) 的概念，我们这里不提了。

Fields 包含两种类型 stored 和 indexed，也可以既是 stored 又是 indexed。Stored 方式整个 fields 中的文本被作为一个索引项存储。例如，一篇文档可以包含 url，文件名，内容等 fields。文件名和 url 通常作为 stored 存储。Indexed 类型，Fields 中的文本被 tokenized 到 term，然后 term 被倒排索引。对于文档的内容和标题 field 都可以作为 indexed 类型。例如，我们给网页建立索引时，我们可以建立两个 field：“url”保存网页的 url，它是 stored 类型的 field；“content”保存网页内容，它是 indexed 类型的 field。

Lucene 中的索引维护下面的内容：

- (1) Filed Name。
- (2) Stored Field Value。前页的 Stored field 中存储的内容。

(3) Term Dictionary。词典包含了所有 document 中的所有 indexed Fields 的 term，也包含了含有该 term 的 document 的编号。还有指针指向 Term Frequency 和 Term Proximity。

(4) Term Frequency。词典中的每个 term 的文档频率和一篇文档中的词频，除非建立索引时，选择遗漏该信息。

(5) Term Proximit。Term 出现在每篇文档中的位置，除非建立索引时，选择遗漏该信息。 **(3 , 4 , 5 构成倒排索引)**

(6) Normalized factors。Document 中的每个 field 中一个值，该 Filed 参与评分时乘以该参数。前面关于信息检索的内容中我有部分内容没有将，称作“参数化”

(7) Term Vectors。每个 document 的每个 Field 可以存储 term Vector。一个 term text 和 term frequency。 (这个是正向索引)

创建索引

创建索引的过程是将每篇文档转化成 Document 对象，然后调用 IndexWriter 的 addDocument 方法将 document 对象添加到索引中的过程。

IndexWriter 类被用来创建或打开索引。在创建 IndexWriter 前还要做一些工作。

(1) 使用前面创建好的分析器，比如创建一个简单分析器 SimpleAnalyzer 的对象 textAnalyzer。

(2) 创建一个 Directory 类的对象，该对象指向了创建的索引保存的路径。

(3) 创建 IndexWriterConfig 类的个对象。该对象对 IndexWriter 的功能进行设置。 Analyzer 对象作为为了 IndexWriterConfig 类构造方法的参数。 IndexWriterConfig 对象的 setOpenMode 方法决定是创造一个新索引还是打开一个存在的索引。如果不调用 setOpenMode 方法进行设置，则是创建一个新的索引。如果是调用该方法且使用了 IndexWriterConfig.OpenMode.CREATE_OR_APPEND 作为参数，则是在设置的索引路径 dir 处寻找索引，找到则是添加新的内容到该索引，否则创建新索引。

当创建 IndexWriter 对象时，使用上面(2)、(3)步骤创建的对象作为参数。

```
Analyzer textAnalyzer=new SimpleAnalyzer();
IndexWriterConfig conf = new IndexWriterConfig(textAnalyzer);
conf.setOpenMode(IndexWriterConfig.OpenMode.CREATE_OR_APPEND)
;
```

```
Directory dir = FSDirectory.open(Paths.get("c:/qjt/index"));
IndexWriter writer=new IndexWriter(dir,conf);
```

这一段代码创建了一个简单的分析器，设置了索引被保存的路径 “c:/qjt/index” 。并设置如果在该路径没发现索引则创建新索引，发现了索引则添加新的内容到该索引。如果设置 conf.setOpenMode(IndexWriterConfig.OpenMode.CREATE);

则是在 dir 设置的位置创建新索引。

进一步 IndexWriter 对象为文档创建索引时，首先需要为被索引的文档创建 Document 对象。IndexWriter 类中的 addDocument 方法向索引中添加需要被索引的文档的 Document 对象，如此来建立索引。

创建索引时，我们应该先考虑好，我们的索引应该有哪些 field。Lucene 5.X 版本的 Field 类有很多的子类，具体为 document 对象创建 field 对象时是使用它的这些子类。我们介绍几个常用的子类，详细内容参考 Lucene 的 API。

如果我们是为文档正文创建 Field，应该采用 TextField 子类（保存长文），而且是一个 Indexed 类型的 field。为路径创建 Field，应该用 StringField（保存短文本），应该是一个 stored 类型的 field，如果文档标题能被抽取出来，也可以建立一个 indexed 类型的 field。如果有一些整数需要保存，可以使用 IntField 或 LongField。

我们介绍三个 Field 子类常用的构造方法：

(1) IntField(String name, int value, Field.Store stored)

创建一个 IntField 对象，name 是 Field Name。value 是要被创建的值。Filed.Store 如果值为 Filed.Store.YES。表示将该 field 整个作为一个单元建立索引。

IntField field = new IntField("id", 6, Field.Store.NO);

创建一个名字为“id”的 field，当前该 field 保存的值是 6，它不是 stored 类型的。

(2) public StringField(String name, String value, Field.Store stored)

A field that is indexed but not tokenized: the entire String value is indexed as a single token. For example this might be used for a 'country' field or an 'id' field, or any field that you intend to use for sorting or access through the field cache.

name - field name ; value - String value ; stored -对于很短的字符串，如只是“作者”，或者对于路径，URL 可以设为 Store.YES。其他情况应该设为 Store.NO

(3) public TextField(String name, String value, Field.Store store)

A field that is indexed and tokenized, without term vectors. For example this would be used on a 'body' field, that contains the bulk of a document's text.

name - field name ; value - string value ; store – 如果保存的文档的内容应该设为 Store.NO

```
public TextField(String name, Reader reader)
```

使用 reader 创建一个新的 un-stored TextField。

一段关于进行怎样考虑建立索引时选择 Store.YES 或 Store.NO 的例子如下：

say you index information about a book with the title: "Lucene in Action" with an ID and other information. searching for "Lucene" will find the book and will give you the book's ID. now if you used Store.YES -- then Lucene can also give you the full title, i.e. "Lucene in Action", but if you used Store.NO -- then you can not retrieve this information from Lucene. you will need to take the ID and retrieve the title from some other data store.

创建 document 对象后，添加 field 到该对象。下面是示例代码

```
Document doc=new Document();
Field pathField = new StringField("path", file.getName(), Field.Store.YES);
doc.add(pathField);
doc.add(new TextField("contents", new FileReader(file)));
writer.addDocument(doc);
```

该代码假设已经有了一个 File 对象 file。为该文件的文件名创建一个 StringField，称为 pathField；为该文件的内容创建一个 field 称为 content。然后添加这些 field 到 document 对象。

4. 查询

Lucene 使用 IndexSearcher 作为主要的检索工具。

```
IndexReader reader =
DirectoryReader.open(FSDirectory.open(Paths.get(indexPath)));
```

```
IndexSearcher is=new IndexSearcher(reader);
```

查询时，使用 IndexSearcher 类的 search 方法来得到检索结果。Search 方法重载了很多。基本的一个方法是：

```
TopDocs search(Query query, int n)
```

此时需要的一个参数是 Query 类的对象，返回的结果集是 TopDocs 类型。n 是返回的结果个数

创建 Query 类的对象有两种，我们介绍使用 QueryParser 类来构造的方法。

```
Analyzer analyzer=newSimpleAnalyzer()
```

```
QueryParser parser=new QueryParser("要检索的 field", analyzer)
```

```
Query query = parser.parse(查询)
```

注意：（1）这里的 analyzer 应该是你创建索引时建立的那个 analyzer 类型相同。

（2）布尔检索是 Lucene 的工作基础，lucene 的评分（或各种检索模型）是工作在布尔检索基础上的。此时给出的查询是布尔查询。即给出“老王 AND 小李”“老王 OR 小李”这样的查询。如果不给出布尔操作符，“老王 小李”，则默认是 OR 查询。查询返回的结果集是 TopDocs 类型，该类的 **scoreDocs** 属性是一个 ScoreDoc 类的数组。ScoreDoc 类有两个属性 doc 是文档在索引中的编号，score 是当前文档的评分。注意：此处的布尔运算符 AND，OR 一定要是大写。

从文档编号获得文档 Document 类型可以使用 IndexSearcher 类的 doc 方法。获得 Document 类的对象后可以使用 get 方法获得文件名。示例代码如下：

```
IndexReader reader =
DirectoryReader.open(FSDirectory.open(Paths.get("c:/qjt")));
IndexSearcher is=new IndexSearcher(reader);
Analyzer analyzer = new StandardAnalyzer();
parser = new QueryParser("content", analyzer);
QueryParser query=parser.parse("习近平");
TopDocs docs=is.search(query, 10);
ScoreDoc[] sdoc=docs.scoreDocs;
for(int i=0;i<sdoc.length;i++){
    doc = is.doc(sdoc[i].doc);
    String name = doc.get("path");
    System.out.println(name+" score:"+sdoc[i].score);
}
```

5. Lucene5.X 的评分机制

https://lucene.apache.org/core/5_0_0/core/org/apache/lucene/search/package-summary.html#scoring

https://lucene.apache.org/core/5_0_0/core/org/apache/lucene/search/similarities/TfIdfSimilarity.html

lucene 支持多种检索模型，如 VSM，概率模型和语言模型。Lucene 首先进行的是布尔检索，默认的是发现逻辑或操作的检索结果。例如，给出查询“A B C”，是首先进行布尔检索“A or B or C”。然后在此基础上应用各种检索模型进行评分计算。

Lucene 有三种层次的 Boosting

(1) Document Level Boosting。在一个 document 添加到索引中之前调用 document.SetBoost()。即给一篇文档分配权重

(2) Field Level Boosting. 在添加一个 field 添加到 document 之前，调用 field.setBoost(float boost)

(3) Query Level Boosting。检索时，在一个查询子句中设置 boost

VSM 评分计算

我们这里只介绍 Lucene 使用 VSM 进行评分计算。Lucene 改进了 VSM

(1) 对 $V(d)$ 规范化，用一个 normalized factor : doc-len-norm(d)

(2) 在索引阶段用户就可以通过使用 boosting 来规定哪一篇文档更重要。通过分配 document boost，此时该文档得分是乘上了 boost 值 doc-boost(d)

(3) Lucene 基于 field，每个 Query term 应用到一个单独的 field。Field 也可以 boosting。

(4) 检索时，用户可以对检索中的每个 query term 规定 boost。从而增加一个查询 term 对 document 的贡献。query-boost(q)

(5) 一个 document 可以并不完全匹配所有的查询 term，用户可以进一步规定匹配更多 query term 的 document 被奖励。通过一个 coordination factor 来奖励。一篇文档 匹配更多个 query term，coord (q,d) 越大

如此，Lucene 实际应用的评分公式如下

$$\text{Score}(q, d) = \text{coord}(q, d)$$

$$* \text{queryNorm}(q) \sum_{t \in q} \text{tf}(t \text{ in } d) * \text{idf}(t)^2 * t.\text{getBoost}() * \text{norm}(t, d)$$

- (1) $tf(t \text{ in } d)$ 是词项 t 在当前文档 d 中的词项频率的开方 $\sqrt{tf_t}$ 。在 lucene 中， $tf(t \text{ in } q)$ 即查询中的词项频率设为了 1。如果一个查询中一个词项有多个重复，则在公式的求和关系 Σ 中体现出了该词项的重要性，即该词项被计算多次。
- (2) $idf(t)$ 即逆文档频率。 $idf(t) = 1 + \log(\frac{N}{df_t + 1})$ 。 N 是文档集合中的文档数， df_t 是词项 t 的文档频率。
- (3) $coord(q, d)$ 是一个因子，基于查询 q 有多少查询词项出现在了一篇文档 d 中。通常我们认为查询词项在一篇文档中出现的越多它应该有更高的得分。
- (4) $queryNorm(q)$ 是一个规范化因子，它让不同的查询的评分是可比较的，它不影响排序。
- (5) $t.getBoost()$ 。如果为查询词项设置了 boost，获得它的 boost
- (6) $norm(t, d)$ 封装了创建索引时计算的一些 field boost 和长度因子 length norm。Length norm 是文档的长度因子，短的文档会有更高的评分。Field boost 在创建文档对象的多个 field 对象后，通过调用 $field.setBoost()$ 可以为每个 field 设置 boost。当前是在哪个 field 上进行检索，就乘上该 field 的 boost。

$$norm(t, d) = lengthNorm * f.boost()$$

Lucene 的其他评分机制如，BM25，语言模型等，参见 Similarity 类 API

https://lucene.apache.org/core/5_0_0/core/org/apache/lucene/search/similarities/package-summary.html

6. Highlighter

我们使用搜索引擎时，检索结果的每一条记录会包含一个短文本称为 snippet，显示检索结果的部分内容，同时在该 snippet 中包含的关键词会被高亮。Lucene 的扩展包 highlighter 包含了该功能。

[Lucene - Official Site](#) 翻译此页

Apache Lucene and Solr set the standard for search and ... hit highlighting and advanced analysis/tokenization ... example/files update-script.js to be Java 7 and ...
<https://lucene.apache.org> ▾

lucene5.X 提供了三个 package 来实施 highlight 功能。它们返回一个 snippet，即一段包含关键词，并使用 html 标记对其中的关键字进行 highlight 显示的文本。

- [org.apache.lucene.search.highlight](#)
- [org.apache.lucene.search.postingshighlight](#)

- [org.apache.lucene.search.vectorhighlight](#)

这里介绍 lucene5.X 的 Hightlighter 包中提供的一个简单的 Highlighter 功能， postinghighlight。

在索引阶段，一个方法 buildDoc 为一篇中文文档创建索引

```
private static Document buildDoc(File f) throws Exception{
    String fname=f.getPath();
    Document doc=new Document();

    FieldType offsetsType = new FieldType(TextField.TYPE_STORED);
    offsetsType.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS_AND_OFFSETS);
    Field body = new Field("contents", getContent(f), offsetsType);

    doc.add(body);
    doc.add(new StringField("path", fname, Field.Store.YES));
    return doc;
}

private static String getContent(File f) throws Exception{
    ChineseFileReader cfr=new ChineseFileReader(f);
    StringBuffer sb=new StringBuffer();
    String line;
    while((line=cfr.readLine())!=null){
        sb.append(line.trim());
    }
    return sb.toString();
}
```

这里我们用一个专门读写中文的包 ChineseFileReader 来将中文文档转换成一个字符串。然后，使用下面的代码创建 field，并添加到 document 对象中

```
FieldType offsetsType = new FieldType(TextField.TYPE_STORED);
```

```

offsetsType.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS_A
ND_OFFSETS);
Field body = new Field("contents", getContentType(f), offsetsType);

```

在查询阶段

```

public static void run(String indexPath, String query, int n) throws
Exception{

    IndexReader reader =
DirectoryReader.open(FSDirectory.open(Paths.get(indexPath)));

    IndexSearcher is=new IndexSearcher(reader);

    ScoreDoc sdoc[];

    PostingsHighlighter highlighter = new PostingsHighlighter();

    Query q = new TermQuery(new Term("contents", query));

    TopDocs topDocs = is.search(q, n);

    String highlights[] = highlighter.highlight("contents", q, is,
topDocs, 3);

    sdoc=topDocs.scoreDocs;

    Document doc;

    System.out.println("\n"+highlights.length);

    for(int i=0;i<highlights.length;i++){

        doc = is.doc(sdoc[i].doc);

        String name = doc.get("path");

        System.out.println(name);

        System.out.println(highlights[i]);

    }

}

```

创建 PostingHighlighter 的对象，创建查询对象，获得查询结果，获得该查询结果对应的高亮的 snippets。

```
highlighter.highlight("contents", q, is, topDocs, 3);
```

表示返回的 snippet 中最多包含 3 段包含了关键词的片段。

例如：

毛泽东同志在建国初期，特别在社会主义改造基本完成以后，再三指出要把工作中心转到经济方面和技术革命方面来。... 毛泽东同志认为，除了特殊的、局部的、暂时的现象以外，社会主义社会的矛盾不象资本主义社会的矛盾那样表现为剧烈的对抗和冲突，它可以通过社会主义制度本身而得到解决。为了充分发挥社会主义制度的这个优越性，我们应该继续努力提高人民特别是青年的马克思列宁主义、毛泽东思想的政治觉悟，继续努力加强社会主义民主和社会主义法制。

第五章：文本分类

第一节：朴素贝叶斯分类

“分类”是数据挖掘和机器学习中的基本任务。面向文本的分类任务则称为文本分类。文本分类的应用领域很广，如：垃圾网页检测，垃圾邮件过滤，垂直搜索引擎等。

分类问题的形式化描述：

有文档 $d \in X$ ， X 表示文档空间，和一个固定的类别集合 $C = \{c_1, \dots, c_l\}$ 。给定已经标注好的训练集 $D = \langle d, c \rangle$ ，其中 $\langle d, c \rangle \in X \times C$ ，利用某种学习算法，从训练集学习一个分类函数 γ ，它可以将文档映射到类别：

$$\gamma : X \rightarrow C$$

此类有训练集的学习方法称为，有监督学习。

有很多文本分类算法，朴素贝叶斯（Naïve Bayesain, 缩写 NB）是最基本的一种分类方法。它包含多项式朴素贝叶斯和贝努力朴素贝叶斯两种。多项式朴素贝叶斯模型是一种基于概率的学习方法。文档 d 属于类别 c 的概率 $P(c|d)$ ，可以如下计算：

$$P(c|d) = P(d|c)P(c)/P(d)$$

因为 $P(d)$ 是常数，我们可以得到

$$P(c|d) \propto P(d|c)P(c)$$

文档 d 是由一个词项集合 $\{t_1, \dots, t_D\}$ 构成

$$P(c|d) \propto P(t_1, \dots, t_D | c)P(c)$$

当我们不考虑词项之间的关系，即每个词项的出现是独立事件，可以得到

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k | c)$$

$P(t_k | c)$ 是 t_k 出现在类 c 文档中的条件概率； $P(c)$ 是文档出现在类 c 中的先验概率； $\langle t_1, t_2, \dots, t_{n_d} \rangle$ 是文档 d 中出现在词汇表中的词条。

例如，对于一篇文档 “Beijing and Taipei join the WTO” , 去掉停用词 and 和 the 后的词条是 Beijing, Taipei, join, WTO。

文本分类是找出文档最有可能属于的类别。对于 NB 分类来说，最可能的类是具有最大后验概率(MAP)的类别 c_{map} 。

$$c_{\text{map}} = \arg \max_{c \in C} \hat{P}(c|d) = \arg \max_{c \in C} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$

上述公式计算对应的条件概率的乘积，这可能会导致浮点数下界溢出，因此通过引入 log 对数，将上式转变成求对数和。

$$c_{\text{map}} = \arg \max_{c \in C} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)]$$

$\log \hat{P}(t_k|c)$ 表示词项 t_k 在类别 c 中的权重。类别的对数先验值和词项在类别中的权重累加求和后，就得到了文档属于类别的可能程度。求解该公式则需要对参数 $P(c)$ 和 $P(t_k|c)$ 进行估计。使用最大似然估计时，它实际上最后算出的是相对频率值 $\hat{P}(c) = \frac{N_c}{N}$ 。条件概率 $P(t_k|c)$ 的估计为 t 在 c 类文档中出现的相对频率

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

T_{ct} 是 t 在训练集合 c 类文档中出现的词项频率；分母 $\sum_{t' \in V} T_{ct'} =$ 是词汇表中所有词项在训练集合 c 类文档中出现的频率之和

MLE 通常会有一个问题，例如，对应没有在训练集中出现的<词项，类别>来说，其 MLE 估计值 $P(t_k|c)$ 为 0。 $\log(P(t_k|c))$ 的计算将失去意义。因此需要采用平滑方法。

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B}$$

产生 NB 分类器的算法如下：

```

TRAINMULTINOMIALNB(C, D)
1   $V \leftarrow \text{EXTRACTVOCABULARY}(D)$ 
2   $N \leftarrow \text{COUNTDOCS}(D)$ 
3  for each  $c \in C$ 
4  do  $N_c \leftarrow \text{COUNTDOCSINCLASS}(D, c)$ 
5     $prior[c] \leftarrow N_c / N$ 
6     $text_c \leftarrow \text{CONCATENATETEXTOFTALLDOCSINCLASS}(D, c)$ 
7    for each  $t \in V$ 
8      do  $T_{ct} \leftarrow \text{COUNTTOKENSOFTERM}(text_c, t)$ 
9      for each  $t \in V$ 
10     do  $condprob[t][c] \leftarrow \frac{T_{ct} + 1}{\sum_{t'} (T_{ct'} + 1)}$ 
11  return  $V, prior, condprob$ 
```

可算法需要两个参数类别集合 C 和文档集合 D。step 1 从训练集合抽取词汇表 V；step 2 统计训练集的文档数；step 3 到 10，考察每个类别 c，统计属于当前类别 c 的文档数 N_c ，然后可以计算向量 $p(c)$ 。第 6 步，将文档集合 D 中属于 c 类的文档拼接成一个长串。对于词汇表中的每个词 t，计算它在类别 c 文档中的词项频率 T_{ct} 。如此就可以计算条件概率 $p(t|c)$ 。最后产生的 NB 分类器实际上是：词汇表 V，类别的先验概率 prior 和每个词项属于每个类别的条件概率。

当新到来一篇文档 d，使用 NB 分类器进行分类的算法如下：

```
APPLYMULTINOMIALNB(C, V, prior, condprob, d)
1 W ← EXTRACTTOKENSFROMDOC(V, d)
2 for each c ∈ C
3   do score[c] ← log prior[c]
4     for each t ∈ W
5       do score[c] += log condprob[t][c]
6 return arg maxc ∈ C score[c]
```

例子：有一个文档集合如下

	docID	words in document	in c = China?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

它包含 4 篇文档，两个类别 $c=China$ 和 $\sim c$ 。建立一个多项式 NB 的步骤如下：

(1) 建立词汇表 $V=\{\text{Chinese}, \text{Beijing}, \text{Shanghai}, \text{Macao}, \text{Tokyo}, \text{Japan}\}$

(2) 统计文档集合文档数 $N=4$ ，属于类别 c 的文档数 $N_c=3$, $N_{\sim c}=1$ 。计算先验概率 $P(c)=3/4$, $P(\sim c)=1/4$ 。

(3) 为每个词计算条件概率

$$\begin{aligned} \hat{P}(\text{Chinese}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\ \hat{P}(\text{Tokyo}|c) &= \hat{P}(\text{Japan}|c) = (0+1)/(8+6) = 1/14 \\ \hat{P}(\text{Chinese}|\bar{c}) &= (1+1)/(3+6) = 2/9 \\ \hat{P}(\text{Tokyo}|\bar{c}) &= \hat{P}(\text{Japan}|\bar{c}) = (1+1)/(3+6) = 2/9 \end{aligned}$$

当新到来文档 d5。为它分类的步骤如下

对于类别 c，为 d5 中的每个出现在词汇表中的词项计算 $P(c)*P(t_1|c)...P(t_d|c)$

$$\hat{P}(c|d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003$$

对于类别 $\sim c$ ，为 d5 中的每个出现在词汇表中的词项计算 $P(\sim c)*P(t_1|\sim c)...P(t_d|\sim c)$

$$\hat{P}(\bar{c}|d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001$$

按照 MAP，挑选最大的后验概率对应的类别，为 c。即文档 d5 属于类别 c。

贝努力模型

建立 NB 分类器有两种不同的方法，上节介绍的是基于多项式的方法，另一种方法是贝努力模型。该模型中，对于词汇表中的每一个词项都对应一个二值变量，1 和 0 分别表示词项在文档中出现与否。

贝努力 NB 的训练算法如下：

```
TRAINBERNOULLINB(C, D)
1 V ← EXTRACTVOCABULARY(D)
2 N ← COUNTDOCS(D)
3 for each c ∈ C
4 do Nc ← COUNTDOCSINCLASS(D, c)
5 prior[c] ← Nc/N
6 for each t ∈ V
7 do Nct ← COUNTDOCSINCLASSCONTAININGTERM(D, c, t)
8 condprob[t][c] ← (Nct + 1)/(Nc + 2)
9 return V, prior, condprob
```

贝努力模型的分类算法如下：

```
APPLYBERNOULLINB(C, V, prior, condprob, d)
1 Vd ← EXTRACTTERMSFROMDOC(V, d)
2 for each c ∈ C
3 do score[c] ← log prior[c]
4 for each t ∈ V
5 do if t ∈ Vd
6 then score[c] += log condprob[t][c]
7 else score[c] += log(1 - condprob[t][c])
8 return arg maxc ∈ C score[c]
```

贝努力 NB 和多项式 NB 的区别：

从上面的算法中可以看出，贝努力模型在估算条件概率概率 $P(t|c)$ 时只考虑词项出现或不出现（即二值），并不考虑出现次数。而多项式模型考虑出现次数。在使用模型进行分类时，多项式模型中并不考虑未出现的词；而贝努力模型对词汇表中的所有词都参与 $P(c|d)$ 的概率估算。

另外，多项式模型适合处理更长的文档，而贝努力模型在短文档上表现的比较好。多项式模型可以处理特征数较多的模型，而贝努力适合特征数较少的模型。

对 NB 模型的总结：

(1) NB 的概率估计效果差，但分类决策效果出乎意料的好。

(2) NB 的速度快，加上不低的精确度使得 NB 模型通常作为基准分类器来使用。

对于第一条，我们可以从下面的这张表中进行理解

	c_1	c_2	class selected
true probability $P(c d)$	0.6	0.4	c_1
$\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k c)$ (Equation (13.13))	0.00099	0.00001	
NB estimate $\hat{P}(c d)$	0.99	0.01	c_1

这里文档 d 属于类别 c_1 和 c_2 的真实概率是 0.6 和 0.4。但用 NB 模型估计出的是 0.99 和 0.01。但是当我们用 NB 做分类决策时，却可以得到正确的答案。

第二节：特征选择

特征选择是指从训练集合出现的词项中选出一部分子集的过程。在文本分类中仅使用这个子集作为特征。特征选择主要有两个目的：

(1) 通过减小有效的词汇空间来提高分类器的训练和使用效率。

(2) 特征选择可以去除噪声特征，从而提高分类精度。噪声 特征是指那些加入文本表示后反而会增加新数据上的分类错误率的特征。

噪声特征是指，当把这些特征加入到文档的描述中，使得新数据的分类正确率降低的特征。例如，一个很少见的词 arachnocentric，对于 China 这个类别没有任何相关性。但在训练集中碰巧有词 arachnocentric 出现在了 China 这个类别的文档中。然后学习到的分类器包含了该特征。这就是学习算法的过拟合。

思考：为什么说，贝努利模型对噪声特征敏感，必须进行特征选择？

用一段伪代码来形式化描述特征选择，它是从训练集 D 中为类别 c 选择前 k 个最佳特征。

```
SELECTFEATURES(D, c, k)
1  V ← EXTRACTVOCABULARY(D)
2  L ← []
3  for each t ∈ V
4    do A(t, c) ← COMPUTEFEATUREUTILITY(D, t, c)
5      APPEND(L, ⟨A(t, c), t⟩)
6  return FEATURESWITHLARGESTVALUES(L, k)
```

可以看出，进行特征选择最关键的是计算词项 t 和类别 c 的效用指标 $A(t, c)$ 。计算效用指标有多种方法。其中一种是使用互信息 $I(t; c)$ 。互信息 $I(X; Y)$ 是一种计算两个随机变量 X, Y 之间共有信息的度量。当在文本分类中使用互信息做特征选择时，就是计算词项和类别之间的互信息。为某个类选出互信息高的词项。

$$I(U;C) = \sum_{e_t \in \{1,0\}} \sum_{e_c \in \{1,0\}} P(U = e_t, C = e_c) \log_2 \frac{P(U = e_t, C = e_c)}{P(U = e_t)P(C = e_c)}$$

为了理解互信息我们先来补充一下信息论的知识。

(注：这里的互信息是期望互信息，它和逐点互信息 pairwise mutual information 还不同)

1. 补充知识

信息论是由香农(Claude Shannon)在二十世纪四十年代创建的理论体系。最初的研究是如何在非理想化的通信通道中传播更多的信息。现在已经应用到很多涉及概率统计的领域。其包含的基本理论如下：

(1) 熵 (entropy) : 设 $p(x)$ 为随机离散变量 X 的概率密度函数 , x 属于离散集合 X 。
 $p(x)=P(X=x), x \in X$ 。熵表示单个随机变量的不确定性。随机变量的熵越大 , 它的不确定性越大。也就是说能正确估计其值的概率越小。熵的计算公式如下 :

$$H(X) = - \sum_{x \in X} p(x) \log p(x)$$

熵度量了包含的随机信息量的大小 , 它的单位是 bit。按照熵的定义 , 它具备以下三个属性 :

- $H(X) \geq 0$
- $H(X)=0$, 当且仅当随机变量 X 的值是确定的 , 没有任何信息量可言。
- 熵值随着信息长度的增加而增加 (这里是指在信息传输中的信息编码)

(2) 联合熵

如果 (X, Y) 是一对离散随机变量 , 其联合概率分布函数密度为 $p(x, y)$, (X, Y) 的联合熵 $H(X, Y)$ 定义为

$$H(X, Y) = - \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(x, y)$$

(3) 条件熵 :

如果离散随机变量 (X, Y) 的联合概率分布函数密度为 $p(x, y)$, 已知随机变量 X 的情况下随机变量 Y 的条件熵 $H(Y|X)$, 实际上表示的是在已知 X 的情况下 , 传输 Y 额外需要的平均信息量

$$H(Y|X) = - \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 p(y|x)$$

熵的链规则为： $H(X,Y) = H(X) + H(Y|X)$

(4) 互信息

根据熵的链规则，可以有如下的计算公式

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

这个差值称为随机变量 X 和 Y 之间的互信息。用 $I(X;Y)$ 表示，它实际上表示在已知 Y 的值后 X 的不确定性的减少量，即随机变量 Y 揭示了多少关于 X 的信息量。

我们可以用一张图（图 5-1）来描述上述概念之间的关系。

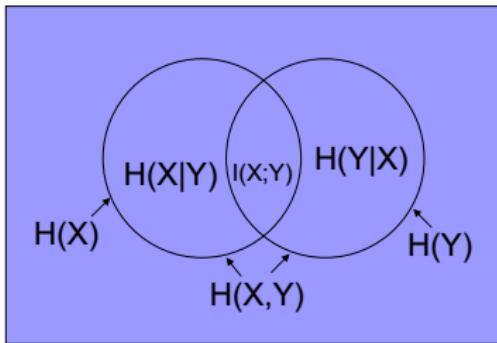


图 5-1：熵，条件熵，联合熵和互信息自己的关系

互信息的公式可以描述为

$$I(X;Y) = \sum_{x,y} p(x,y) \log(p(x,y)/p(x)p(y))$$

它是一种计算两个随机变量之间共有信息的度量。满足非负性和对称性两个特点。它实际上作为验证两个变量是否互不相关的手段。它的取值（1）当两个随机变量无关时，互信息为 0；（2）当两个变量之间存在依赖关系时，它们的互信息不仅和依赖程度相关，而且和变量的熵也相关。

(5) 相对熵和 Kullback-Leibler 距离

给定两个概率密度函数 $p(x)$ 和 $q(x)$ ，它们的相对熵又称为 Kullback-Leibler Divergence，或信息增益。有下面的公式

$$D(p||q) = \sum_{x \in X} p(x) \log(p(x)/q(x))$$

相对熵满足两个属性：(1) $D(p||q) >= 0$ 当且仅当 $p(x)=q(x)$ 时成立。(2) $D(p||q) <> D(q||p)$ 即非对称性。

相对熵表示了两个随机变量分布之间的差异程度。

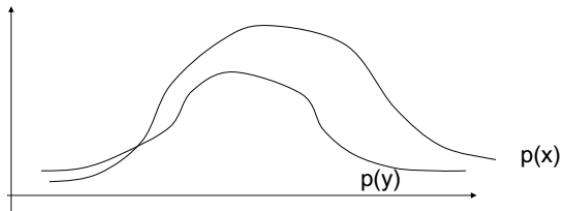


图 5-2：两个随机变量之间的分布

2. 互信息

使用互信息进行特征选择时，公式如下

$$I(U;C) = \sum_{e_t \in \{1,0\}} \sum_{e_c \in \{1,0\}} P(U = e_t, C = e_c) \log_2 \frac{P(U = e_t, C = e_c)}{P(U = e_t)P(C = e_c)}$$

e_t 是用出现或没出现来描述词项 t ，它有两种可能取值 {1,0}。 e_c 描述是不是类别 c ，它有两种选择 {1,0}。 $P(U=e_t, C=e_c)$ 则描述了词项 t 和类别 c 的可能取值的联合概率。当用 MLE 来估计概率参数时，上述公式等价于

$$\begin{aligned} I(U;C) &= \frac{N_{11}}{N} \log_2 \frac{NN_{11}}{N_{1.}N_{.1}} + \frac{N_{01}}{N} \log_2 \frac{NN_{01}}{N_{0.}N_{.1}} \\ &\quad + \frac{N_{10}}{N} \log_2 \frac{NN_{10}}{N_{1.}N_{.0}} + \frac{N_{00}}{N} \log_2 \frac{NN_{00}}{N_{0.}N_{.0}} \end{aligned}$$

N_{11} 表示类别为 c 的文档中出现词项 t 的文档数。 $N_{1.}=N_{10}+N_{11}$ 是词项 t 出现的文档数。 N 是所有文档数。 N_{00} 是类别 $\sim c$ 中没有出现词项 t 的文档数。 $N_{0.}=N_{00}+N_{01}$ 是没有出现词项 t 的文档数。

例子：在 Reuters-RCV1 语料库有一个 *poultry* 类，我们考察词项 *export*。词项和文档关系的统计数据，互信息 $I(e_t|e_c)$ 如下

	$e_c = e_{poultry} = 1$	$e_c = e_{poultry} = 0$
$e_t = e_{export} = 1$	$N_{11} = 49$	$N_{10} = 27,652$
$e_t = e_{export} = 0$	$N_{01} = 141$	$N_{00} = 774,106$

$$\begin{aligned} I(U;C) &= \frac{49}{801,948} \log_2 \frac{801,948 \cdot 49}{(49+27,652)(49+141)} \\ &\quad + \frac{141}{801,948} \log_2 \frac{801,948 \cdot 141}{(141+774,106)(49+141)} \\ &\quad + \frac{27,652}{801,948} \log_2 \frac{801,948 \cdot 27,652}{(49+27,652)(27,652+774,106)} \\ &\quad + \frac{774,106}{801,948} \log_2 \frac{801,948 \cdot 774,106}{(141+774,106)(27,652+774,106)} \\ &\approx 0.0001105 \end{aligned}$$

对互信息的理解：从信息论的角度，互信息度量的是词项是否被类别包含所带来的信息量。 $I(t,c)$ 互信息值越大，表明词项 t 越能判别归属类别 c 。互信息是从信息论的角度

度度量了一个词项被包含在一个类别中的信息量。如果在一个类 c 中某个词项 t 的分布和词项 t 在整个文档集上的分布是一样的，则互信息 $I(t;c)=0$ 。如果一个词项 t 是一个类别 c 的完美指示器，互信息 $I(t;c)$ 会达到最大值。也就是说该词项出现在一篇文档里，当且仅当该文档属于类别 c 。

Mutual information measures how much information – in the information-theoretic sense – a term contains about the class. If a term's distribution is the same in the class as it is in the collection as a whole, then $I(U;C) = 0$. MI reaches its maximum value if the term is a perfect indicator for class membership, that is, if the term is present in a document if and only if the document is in the class.

对于上面的文字，“如果在类 c 上一个词项的分布和在整个文档集上的分布相同，那么 $I(U;C) = 0$ ”。我们设想一个极端的例子，当前的文档集 D 中有两个类别集合 C_1 和 C_2 ，满足 $C_1 \cup C_2 = D$ 。 C_1 和 C_2 类中的文档数都是 15，即整个 D 上有 30 篇文档。词项 t 在两个类别上出现次数相同，都是：{1:15, 2:5, 3:0}。其含义是词项 t 出现了一次的文档有 15 篇，出现了 2 次的有 5 篇，出现了 3 次的有零篇。如此，词项 t 在 C_1, C_2 和整个文档集上的分布相同都是。

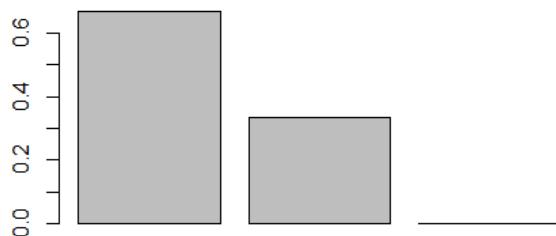


图 5-3：词项 t 在 C_1 和 C_2 文档集上的分布

而如果词项 t 没有出现在 C_2 类中，则在整个文档集上的分布是

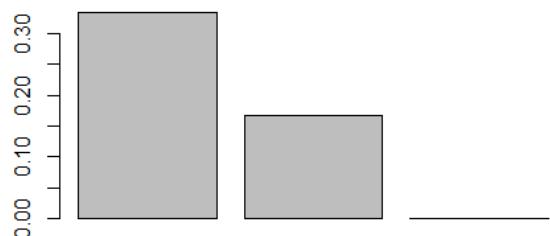


图 5-4：词项 t 在文档集合 D 上的分布

此时，词项 t 在类 C_1 和文档集合 D 上的两个分布是有很大差别的。

下图列出了 6 个类别中互信息值较高的部分词项列表。

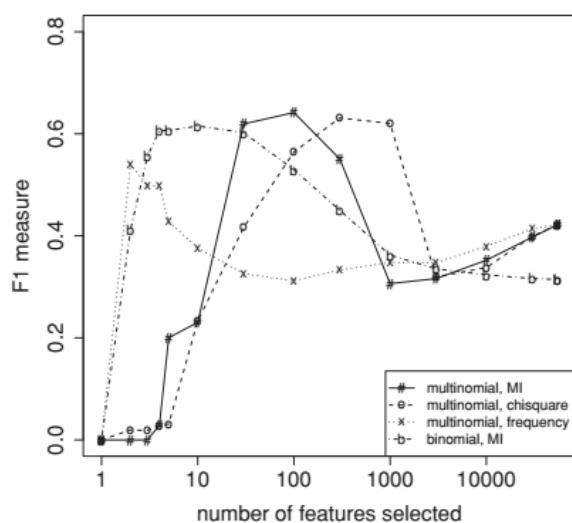
UK		China		poultry	
london	0.1925	china	0.0997	poultry	0.0013
uk	0.0755	chinese	0.0523	meat	0.0008
british	0.0596	beijing	0.0444	chicken	0.0006
stg	0.0555	yuan	0.0344	agriculture	0.0005
britain	0.0469	shanghai	0.0292	avian	0.0004
plc	0.0357	hong	0.0198	broiler	0.0003
england	0.0238	kong	0.0195	veterinary	0.0003
pence	0.0212	xinhua	0.0155	birds	0.0003
pounds	0.0149	province	0.0117	inspection	0.0003
english	0.0126	taivian	0.0108	pathogenic	0.0003
coffee		elections		sports	
coffee	0.0111	election	0.0519	soccer	0.0681
bags	0.0042	elections	0.0342	cup	0.0515
growers	0.0025	polls	0.0339	match	0.0441
kg	0.0019	voters	0.0315	matches	0.0408
colombia	0.0018	party	0.0303	played	0.0388
brazil	0.0016	vote	0.0299	league	0.0386
export	0.0014	poll	0.0225	beat	0.0301
exporters	0.0013	candidate	0.0202	game	0.0299
exports	0.0013	campaign	0.0202	games	0.0284
crop	0.0012	democratic	0.0198	team	0.0264

► Figure 13.7 Features with high mutual information scores for six Reuters-RCV1 classes.

图 5-5：特征选择出的 TOP 10 个词项

下图是在 Reuters-RVC1 语料库的实验结果，不同特征数目下多项式模型和贝努力模型的分类效果。F1 度量是对分类的评价指标查全率和查准率进行综合考察的一个分类性能指标：

$$F1 = 2 * Precision * Recall / (Precision + Recall)$$



► Figure 13.8 Effect of feature set size on accuracy for multinomial and Bernoulli models.

图 5-6：NB 和特征选择的实验结果

从图 5-6 可以得出结论：无论采用哪种模型，认真挑选出的部分特征而获得的分类效果都会优于全部特征。（对于该图有些疑问,当采用基于频率的特征选择,只需要不到 10

个词项就可以大大很高的 F 值?既然,卡方和互信息进行特征选择时,达到峰值时,词项数目的差别有些大,为什么 ?需要实验验证。

注: 实验的结果显示该图有问题啊 , 当继续增大特征数的时候 , F1 值没有下降 , 而是维持在最高值。按照词频挑选特征 , 也是在几百后达到最高值

3. 卡方 Chi-Square

在统计学中 , 卡方统计量常常用于检测两个事件的独立性。两个事件 A 和 B 独立是指 A、B 的概率满足 $P(AB)=P(A)P(B)$ 。在特征选择中 , 两个事件是指词项的出现和类别的出现。此时的卡方统计按如下公式计算 :

$$X^2(\mathbb{D}, t, c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}}$$

D 是文档集 ; N 是 D 中观测到的频率(文档频率) , E 是期望频率

$$\chi^2(D, t, c) = \frac{(N_{11} - E_{11})^2}{E_{11}} + \frac{(N_{10} - E_{10})^2}{E_{10}} + \frac{(N_{01} - E_{01})^2}{E_{01}} + \frac{(N_{00} - E_{00})^2}{E_{00}}$$

对于 $N_{et, ec}$, N_{11} 的含义是在类别 c 的文档集合中出现了词项 t 的文档数。 E_{11} 表示在词项 t 和类别 c 独立的情况下 , 在 c 类文档集合中出现了词项 t 的期望文档数

$$E_{11} = N * P(t|c) = N * P(t) * P(c)$$

$$\text{估计值 } P(t) = (N_{11} + N_{10}) / N$$

$$\text{估计值 } P(c) = (N_{11} + N_{01}) / N$$

因此可以得到卡方统计量的一个更简单的计算公式

$$X^2(\mathbb{D}, t, c) = \frac{(N_{11} + N_{10} + N_{01} + N_{00}) \times (N_{11}N_{00} - N_{10}N_{01})^2}{(N_{11} + N_{01}) \times (N_{11} + N_{10}) \times (N_{10} + N_{00}) \times (N_{01} + N_{00})}$$

4. 基于频率的特征选择

即选择那些在类别中频率较高的词项作为特征。这里的频率可以是文档频率 (类别 c 中包含某个词项 t 的文档数目) 也可以是文档集频率 (c 类别所有文档中 t 出现的总次数)。文档频率更适合贝努力模型 , 文档集频率更适合多项式模型。 (思考为什么 ?)

5. 多类分类问题的特征选择方法

前面的例子在为一个二分类器挑选特征集。但在多分类问题中，通常做法不是为每个分类器建立特征集。而是建立一个统一的特征集。一种普遍的做法，首先基于二分类问题对每个类计算特征指标，然后将这些指标组合，组合方法有两种：

- (1) 对于一个词项，在每个类上计算一个指标，然后求平均。
- (2) 对 n 个分类器中的每个分类器都选出 k/n 个特征，然后组成全局特征集。

6. 不同特征选择方法的比较

(1) MI 和卡方是两种完全不同的特征选择方法。但两者的分类精度并没有太大的不同。在文本分类中，强指示特征（指那些对确定类别具有很强导向性的特征）很少，大部分都是弱指示特征。只要所有的强特征和很多的弱特征被选出，那么分类的期望精度都不错。上述两种方法都能做到这一点。

- (2) 前页的图 5-6 显示了两种特征选择方法，两者的效果相当。
- (3) 不论是 MI 还是卡方，还是基于频率的方法。都是基于贪心的策略。即我们挑选出的特征不一定是使得分类器能达到最优的特征集合。但是寻找最优的特征集合所付出的代价，往往不能接受，因此才会接受贪心策略产生的次优结果。对于贪心策略不一定能在特征选择时，挑选出最优特征集合的例子：

图 13-7 中，kong 在特征选择中排名第 7，但是它和前面的 hong 高度相关。因此 kong 是冗余信息。即挑选了 kong 这个特征不能给分类器的性能带来提升。

第三节: KNN

向量空间模型将每篇文档表示成实数型的分量所构成的向量，每个分量对应一个词项的权重。其实也可以基于向量的距离来进行分类。这类型的分类的主要思路是邻近假设：同一类的文档构成一个邻近区域，而不同类的邻近区域之间是互不重叠的。

例如，在前面的展示挑选出的特征向量的图中，China 类的文档倾向于在诸如 Chinese, Beijing, Mao 之类的词所对应的维度上取较大值。UK 类中的文档在诸如 London, British 或 Queen 之类的词上取较大值。

如果在向量空间上表示两个类，可以发现两个类之间是有边界的。从而就可以通过确定边界对文档进行分类。这就是基于向量空间模型进行文本分类的原理。

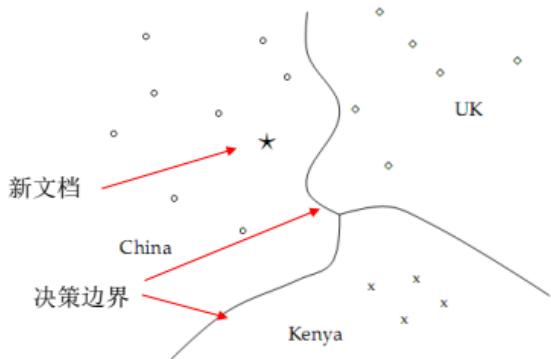


图 5-7：在向量空间模型上的分类

例如，图 5-7 中，根据训练集确定好类别的边界后，新来的文档落在哪个类别的区域，就属于哪个类。

基于向量空间的分类器在分类决策时用到距离的概念。在讲余弦相似度时提到，余弦相似度计算和欧氏距离的计算之间有相关性（转换为单位向量（欧式归一化后）计算的欧式距离等价于余弦相似度），因此分类时，无论采用欧式距离或者余弦相似度来计算距离都可以。

KNN 所依据的假设是：根据邻近假设，一篇测试文档 d 将和其邻域中的训练文档应该具有相同类别。KNN 通过局部信息来确定类别边界。对于 1NN 分类方法，将距离测试文档最近的文档的类别赋给该文档。KNN 将与测试文档最近的 k 篇文档所属的主类别赋给该文档。

K 的取值不同会影响分类效果。图中的测试文档，对于 1NN 来说属于圆圈类，对于 3NN 来说属于 X 类。 K 的取值取决于经验或分类问题本身的有关知识。 $K=3$ 和 $k=5$ 是常用的两组取值，但是 K 也常取到 50-100 间的更大奇数取值。

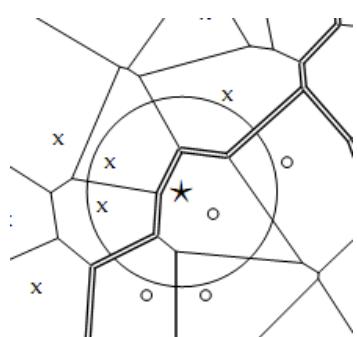


图 5-8:1NN 和 3NN 的分类

可以将 k 个近邻基于余弦相似度加权。这种情况下，文档 d 属于类别 c 的得分计算如下。最后将得分最高的类别赋予文档 d 。这种基于相似度加权投票的方法精度往往高于简单投票的方法。

$$\text{score}(c, d) = \sum_{d' \in S_k(d)} I_c(d') \cos(\vec{v}(d'), \vec{v}(d))$$

S_k 是文档 d 的 k 个近邻组成的文档集合。如果文档 d' 属于类别 c 则 $I_c(d')=1$, 否则=0。
我们可以看到如果使用基于相似度加权投票的 KNN , K 的值就不能取得太小。

实际上，如果把文档集合建立向量空间模型的过程和确定 K 值的过程看做是预处理，KNN 不需要训练的过程。且 KNN 分类器的精度可以与精度最高的文本分类器媲美（下一节的表）。但我们也发现 KNN 在做决策时的计算量和训练集相关，因为新来的文档和每篇文档要计算距离。如果训练集很大，则决策时计算量非常大。

多分类问题

如果我们建立分类器时有多个类别标签，即面对多分类问题怎么处理。多类问题包括非互斥类别上的多分类问题，称为多标签或多值分类，和互斥类别上的单标签分类。
多标签分类问题中，一篇文档可以属于多个类、只属于一个类或不属于任何类。在某个类的分类决策不影响它在其他类上的决策。单标签分类问题：这里的类别是互斥的，一篇文档只能属于一个类。

在多标签分类问题中，我们会学到 J 个不同的分类器 γ_j ，对于每篇文档 d ， γ_j 返回的类别结果要么是 c_j ，要么是 $\sim c_j$ 。通过二分类器来解决多标签问题的步骤如下：

- (1) 对每个类别建立一个分类器，此时训练集包含所有属于该类的文档（正例）和所有不属于该类的文档（反例）
- (2) 给定测试文档，分别使用每个分类器进行分类，每个分类器的分类结果不影响其他分类器的结果

单标签分类问题：这里的类别是互斥的，一篇文档只能属于一个类。形式化的，这种分类存在单个分类函数 γ ，其值域为 $C=\{c_1, \dots, c_J\}$ 。在实际中单标签问题不如多标签问题普遍，如一篇文档可以关于多个主题。在多类问题上，KNN 和 NB 是什么样的分类器？（KNN 是单标签，NB 是多标签）

通过二分类器来解决单标签问题多分类的步骤如下：

- (1) 对每个类别建立一个分类器，此时训练集包含所有属于该类的文档（正例）和所有不属于该类的文档（反例）
- (2) 给定测试文档，分别使用每个分类器进行分类，每个分类器的分类结果不影响其他分类器的结果
- (3) 将文档分配给得分最高的类。

线性及非线性分类器

线性分类器是文本分类中最重要的一类分类器。我们以二分类器为例。根据特征的线性组合和某个阈值的比较结果来确定类别归属的二分类器叫做线性分类器(linear classifier)。二维平面上线性分类器就是一条直线。在多维空间上的分类器就是一个超平面，称为决策超平面。 $w^T x = b$ 。此时的判别准则就是：如果 $w^T x > b$ ，将文档 x 归入 c 类，否则，归入 $\sim c$ 类。线性分类器的主要困难来自于训练，即基于训练集来确定参数 w 和 b 。

NB 和 KNN 是线性还是非线性分类器？NB 在对数空间也是一个线性分类器。

$$c_{\text{map}} = \arg \max_{c \in C} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k | c)]$$

根据本节前面的 KNN 的图，明显 KNN 的分类决策面是线段，因此它是非线性分类器。

线性分类器面临着两个问题：

(1) 如果真实的两个类别分布 $p(d|c)$ 和 $p(d|\sim c)$ 能够被一条直线分开。但存在的一些噪声点，是造成线性分类器训练困难的原因。如果选择分类器的决策超平面时过度关注噪声点，那么在新数据上会表现不佳。更根本的是，我们很难判断哪些文档才是会误导分类的噪声文档。

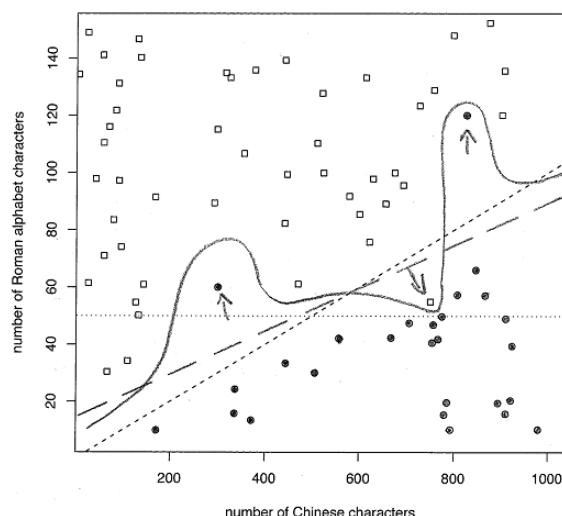


图 5-9：噪声对分类的影响

(2) 如果存在一个超平面能将两类完全分开，那么就称这两个类是线性可分的。实际上如果是线性可分的，那么就存在无穷多个线性分类器可将两类分开，如何选择最优的分类器它可以在新数据上也有很好的效果？

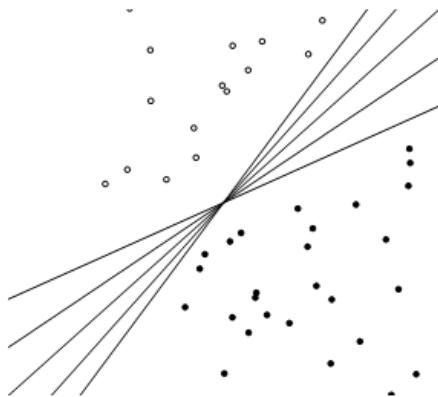


图 5-10：线性分类中的分类超平面

这里有个极端的例子。在下图，由于在左上角存在一个内嵌区域，不可能存在一个好的线性分类器可以将二类进行分类。但是如果采用 KNN，在训练数据足够大的情况下会取得高精度的结果。

那么我们是否就此可以得出结论：非线性分类器是不是一定比线性的好？

在选择分类器的时候有一个概念，偏差-方差折中准则。简单的说，在选择学习方法时，我们的目标是学习误差最小化。学习误差=偏差+方差。偏差反映了在训练集上学习方法能正确学习分类边界的能力。方差是在不同训练集上学习到的分类器，在一个测试集上做决策时的差异程度。反映了分类器对噪声文档的敏感程度。

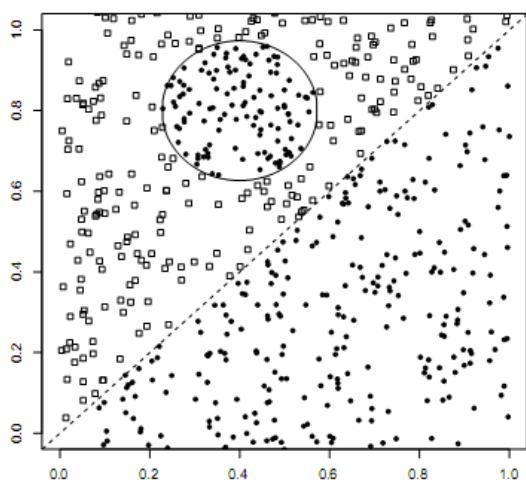


图 5-11：一个极端的例子

偏差：

图 5-9 是有噪声的训练集。线性分类器学习的结果不能把训练集中的数据完全正确区分开。但非线性分类器可以。所以非线性分类器的偏差小。（短虚线是真实边界，长虚线是线性分类器学习到的边界，曲线是非线性分类器学习到的边界）

方差：

假设从一个文本集合中抽样了多个训练集和一个测试集。在不同的训练集上产生的分类器，在最终测试集上得到测试的结果。非线性往往差异性要大。方差反映了学习方法的过拟合特性。

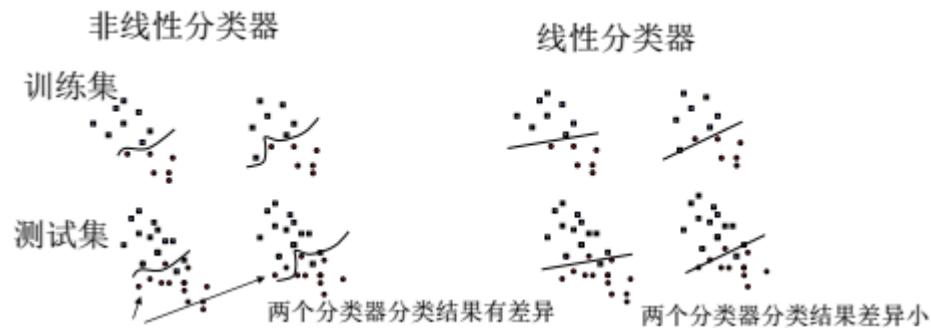


图 5-12：对于方差的图示

图 5-12 解释了方差的原理。有两组不同的训练集，有一组相同的训练集。当训练分类器时得到了不同的分类决策面。即学习到了分类器。把学到的分类器应用在相同的测试集上时，两个非线性分类器进行测试时，测试结果差异较大。而线性分类器测试结果差异较小。我们给出一个结论：非线性分类器因为容易过拟合，因此方差较大。而线性分类器方差较小。

往往 学习误差=偏差+方差。偏差和方差不会同时最小。因实际选择学习方法是根据应用需要对学习方法和方差进行加权求和，然后选择学习误差达到最小的学习方法。这种折中就称为偏差-方差折中。

如果一个问题是非线性的，即它的类别边界不能通过线性超平面来近似，那么此时使用非线性分类器往往会好于线性分类器。如果是一个线性问题，那么最好用简单的线性分类器。

第四节：支持向量机

1. 支持向量机原理

支持向量机（Support Vector Machine, SVM）由 Vapnik 等在 1995 年提出。它是在统计学习理论基础上发展出的一种新的模式识别方法。SVM 旨在解决小样本条件下的分类问题。SVM 在小样本情况下有明显优势；在文本分类方面有明显优势。

二类线性可分问题存在大量可能的线性分界面。直观的看，一个处于中间空白处的决策面比那些靠近某个类的决策面好。图 5-10 给出了例子。NB 分类方法按照某个准则

寻找最优线性分界面。对于 SVM 而言，它定义的准则是寻找一个离数据点最远的决策面（基于向量空间模型的机器学习方法其目标是找到两个类别之间的一个决策边界，使之尽量远离训练集上的任意一点）。从决策面到最近数据点的距离决定了分类器的间隔(margin)。这种思想意味着，SVM 的决策函数完全由部分的数据子集（通常的数量很小）确定，并且这些子集定义了分界面的位置。这些子集中的点称为**支持向量**。除支持向量外的其他数据点对最终分界面的确定不起作用。支持向量机就是寻找这样一个决策超平面，它可以使得两个类之间的分类间隔最大化。

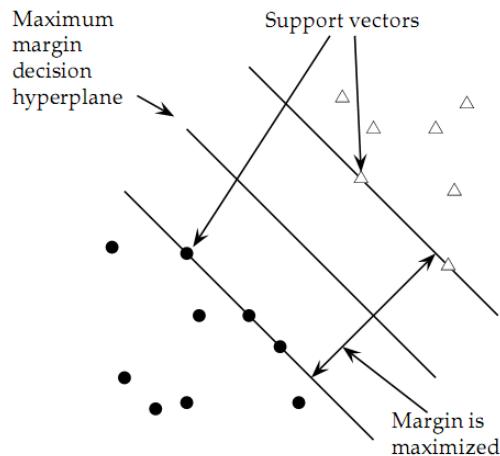


图 5-13：分类器间隔两端的 5 个点是支持向量

最大化分类间隔看上去很合理，这是因为在分界面附近的点代表了不确定的分类决策，分类器会以 50% 的概率做出决策。具有很大分类间隔的分类器不会做出确定性很低的决策，它给出了一个分类的安全间隔：度量中的微小错误和文档的轻微变化不会导致错误分类。

另一种 SVM 的直观解释参见图 5-14。在分类器构建的过程中，SVM 强调在分类决策面上下有一个大的分类间隔。如，在两个类之间放入一个最宽的矩形。

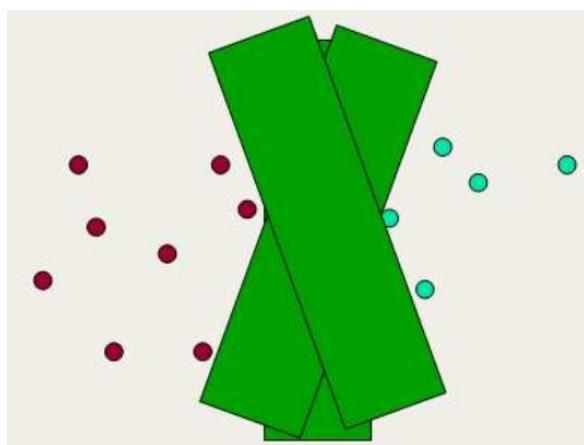


图 5-14：在类之间放入最大间隔的分类器示意图

支持向量机的形式化描述：

决策超平面的数学描述为 $w^T x + b = 0$ 。 w 为决策超平面的法向量， b 为截距。在决策超平面 H_0 上的所有点 x ，都满足 $w^T x + b = 0$ 。假定训练集上的所有点为 $D = \{(x_i, y_i)\}$ ，其中 x_i 是第 i 个样本数据， y_i 是类别标签 $\in \{+1, -1\}$ 。如此 SVM 的线性分类器可以表示成 $f(x) = \text{sign}(w^T x + b)$

这里 sign 函数是

$$\text{sign}(z) = \begin{cases} +1 & z > 0 \\ -1 & \text{else} \end{cases}$$

在图 5-15 所示的例子中， H_1 上的点就满足 $\text{sign}(w^T x + b) = +1$ 。在 H_2 上的点 $\text{sign}(w^T x + b) = -1$ 。训练 SVM 分类器就是由训练集来求解 w 。

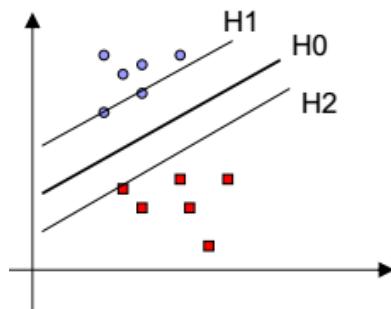


图 5-16：支持向量机示例

一个点 x 到决策超平面的距离记为 r ，由于点到超平面的最短距离垂直于该平面，也就是说和 w 平行。这个方向的单位向量是 $w/|w|$ 。图中的点线就是 $rw/|w|$ 的平移结果。
将超平面上距离 x 最近的点标记为 x' ，于是有 $x' = x - rw/|w|$

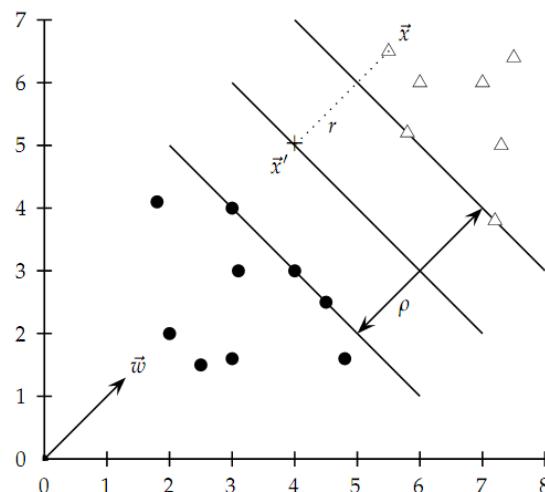


图 5-17：

由于 x' 在决策边界上，因此有 $w^T x' + b = 0$ ，于是 $w^T(x - y_i w / |w|) + b = 0$ ，对 r 求解得

$$r = y_i(w^T x + b) / |w|$$

前面提到，“支持向量机就是寻找这样一个决策超平面，它可以使得两个类之间的分类间隔最大化”。训练集中的每个点 (x_i, y_i) 的类别标签 y_i 是 +1 或 -1。把该点带入支持向量机时，如果 $|w^T x_i + b| < 1$ 则不可以做正确分类。因此，SVM 的约束条件是 $y_i(w^T x_i + b) \geq 1$ ，且若 x 为支持向量， $y_i(w^T x_i + b) = 1$ 。因为 $r = y_i(w^T x + b) / |w|$ ，对于支持向量应满足 $r = 1 / |w|$ 。那么使得支向量的 r （分类间隔/2）最大时的 $|w|$ ，就是我们需要求解的 w 。即寻找 w 和 b 来满足分类间隔 $\rho = 2 / |w|$ 极大化。

标准的支持向量机公式是用最小化形式描述的：寻找 w 和 b 使得

(1) $w^T w / 2$ 极小化

(2) 对所有 $\{(x_i, y_i)\}$ ， $y_i(w^T x_i + b) \geq 1$

上述问题是线性约束条件下的二次优化问题。可以转换为在约束 $a_i \geq 0$ 的条件下，求解使得

$$\sum a_i - \frac{1}{2} \sum_i \sum_j a_i a_j y_i y_j \vec{x}_i^T \vec{x}_j$$

取得最大值的 $a = [a_1 \dots a_n]$ 。除去支持向量对应的 a_i 是 > 0 ，其他数据对应的 a 均等于 0 于是最后的分类函数为

$$f(\vec{x}) = \text{sign}(\sum_i a_i y_i \vec{x}_i^T \vec{x} + b)$$

即只有支持向量参与了分类决策。 a_i 是按照对偶的方式求解二次优化问题时，将原始的约束条件 $y_i(w^T \vec{x}_i + b) \geq 1$ 对应成的拉格朗日因子。

软间隔分类

对于文本分类是高维空间的数据分类。有时数据是线性可分的，一般情况下不成立。而且即使线性可分，也会优先考虑那些能够将大部分数据分开而忽略一些噪声文档的方案。对应到支持向量机，就是说，即使当前的数据集合是完全线性可分的。我们不一定要寻找完全可以把所有文档分开的超平面，因为如此会过拟合。而是允许犯错误，尽量寻找允许错误范围内的最大间隔。

如果训练集是非线性可分的，常规的做法是允许决策间隔(margin)犯一些错误（有离群点或噪声点在决策间隔里或决策面错误一方）。于是需要根据每个错分的例子满足间隔的程度定义其惩罚代价。为实现这一目的，引入松弛变量 ξ ，一个非零的 ξ 表示允许 x_i 在未满足间隔需求下的惩罚量或代价因子。

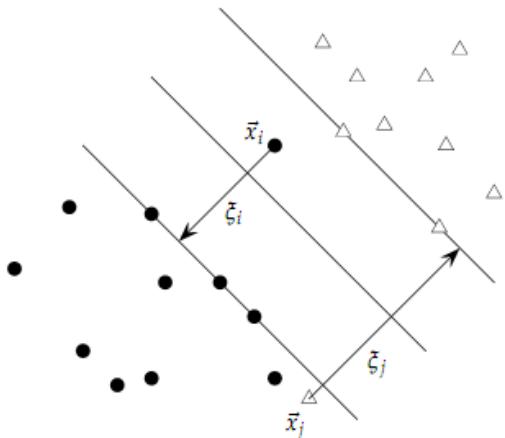


图 5-18：软间隔分类示例

如此，软间隔分类的问题就是：

Find \vec{w} , b , and $\xi_i \geq 0$ such that:

- $\frac{1}{2} \vec{w}^T \vec{w} + C \sum_i \xi_i$ is minimized
- and for all $\{(\vec{x}_i, y_i)\}$, $y_i(\vec{w}^T \vec{x}_i + b) \geq 1 - \xi_i$

C 是正则化因子，可以通过它来控制过拟合问题。如果 C 变大，会对出现在间隔内的点惩罚较大。它会更尊重数据本身，当然代价就是减小了分类间隔。当 C 很小，则容易通过松弛变量来考虑噪声点。可以对大部分数据建立更宽的间隔。

非线性支持向量机

上述介绍的 SVM 称为线性 SVM，因为它针对的数据是线性可分的情况。如果想进行非线性的分类，如图 5-19，需要使用核方法将数据映射到高维空间。在高维空间数据是线性可分的。

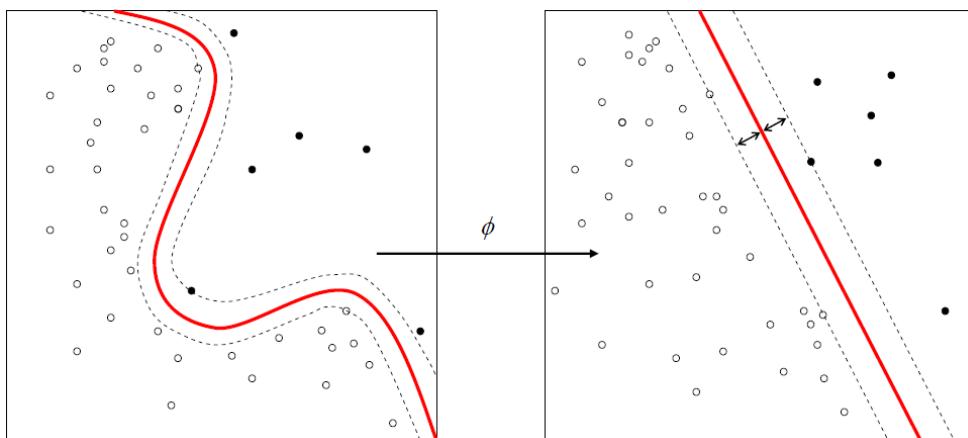


图 5-19：非线性和线性支持向量机

前面讨论的数据集都是线性可分的（最多包含少数离群点或噪声点）。如果数据集不允许线性分类器分类时，应该怎么办？

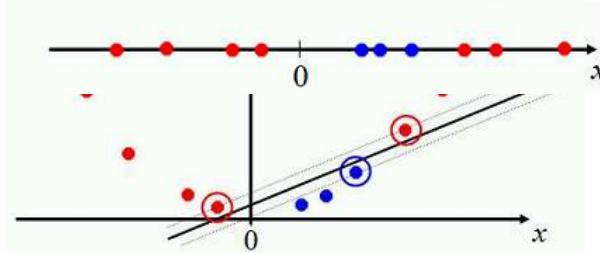


图 5-20：高维空间映射

将图 5-20 中线性不可分的数据映射到一个高维空间并在此空间上使用线性分类器将数据分开。

SVM 以及其他分类器可以通过核方法(kernel trick)非常有效的将数据映射到高维空间。用核函数 $K(x_i, x_j) = \Phi(x_i) \Phi(x_j)$ 替代 $x_i^T x_j$ ，支持向量机的分类器可以写成。 Φ 是映射函数。

$$f(\vec{x}) = \text{sign}(\sum_i \alpha_i y_i K(\vec{x}_i, \vec{x}) + b)$$

最常用的核函数是多项式核函数(polynomial kernels)和径向基核函数(radial basis functions)。多项式核函数的形式如下。当 $d=1$ 时是线性核函数，前面讲的 SVM 实际上就是线性核函数，+1 只是改变了阈值

$$K(\vec{x}, \vec{z}) = (1 + \vec{x}^T \vec{z})^d$$

在 libsvm 中的多项式核函数的公式是

$$K(\vec{x}, \vec{z}) = (\text{coef0} + \gamma \vec{x}^T \vec{z})^d$$

径向基核函数的一个最普遍形式是采用高斯分布

$$K(\vec{x}, \vec{z}) = e^{-(\vec{x} - \vec{z})^2 / (2\sigma^2)}$$

Libsvm 中的径向基函数是

$$K(\vec{x}, \vec{z}) = \exp\{-\gamma |\vec{x} - \vec{z}|^2\}$$

Libsvm 中还有个 sigmoid 核函数

$$K(\vec{x}, \vec{z}) = \tanh(\text{coef0} + \gamma \vec{x}^T \vec{z})$$

2. 几种支持向量机模型

C-SVC

C-SVC 是最常用的支持向量分类，它的数学描述如下：

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^l \xi_i \\ \text{subject to} \quad & y_i (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \dots, l. \end{aligned}$$

v-SVC

v-SVC 使用一个新的参数 v ，它控制支持向量的个数和训练误差。参数 $v \in (0,1]$ 。

$$\begin{aligned} \min_{\mathbf{w}, b, \xi, \rho} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} - v\rho + \frac{1}{l} \sum_{i=1}^l \xi_i \\ \text{subject to} \quad & y_i (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq \rho - \xi_i, \\ & \xi_i \geq 0, i = 1, \dots, l, \rho \geq 0. \end{aligned}$$

还有 one class-SVC 和 SVR 不在本课程中介绍，可以参考 libsvm 的帮助手册。关于参数调优，见下一节。

3. LibSVM

台湾大学的 Chih-Jen Lin 开发了一个著名的 SVM 工具箱 LIBSVM。该库实施了多种 SVM。大家可以参考课件中给的文件。了解更多的 SVM 种类和使用 LIBSVM 的方法。LibSVM 的使用步骤如下：

(1) 建立一个 Eclipse 工程

(2) 解压 Libsvm.rar 文件。将 java 文件夹下的 4 个 java 文件复制粘贴到该 Eclipse 工程中。将 libsvm.jar 文件添加到该工程中。

(3) 运行 svm_toy(applet 方式) 来先简单体会一下 svm

(4) 准备数据

对于分类，训练集格式如下

标签编号空格分量 1:值空格分量 2 ...

1 1:0.2 2:0.5 3:0.1

0 1:0.7 2:0.9 3:0.5

如果分量没有值可以不写

1 162:1.0 164:1.0
2 12:1.0 58:2.0 88:1.0 214:1.0

对于检验集也是如此。对于测试集可以先随便分配一个标签。对于 SVR，在准备数据集时，则将标签换成结果值

0.45 1:0.5 2:4.5

(5) 对数据进行标定

标定 (scaling) 的主要目的是避免值的变化范围太大的属性值他们主宰 (dominate) 取值范围窄的属性。另一个目的是避免进行计算时，一些数值在计算上的困难（太大或 log 计算时值为零等）。推荐为每个属性进行线性标定法，标定范围到 [1; +1] 或者 [0; 1]。当然对于训练集和测试集数据必须使用同样的标定法。

对数据进行标定：标定用的参数

- l lower : x scaling lower limit (default -1)
- u upper : x scaling upper limit (default +1)
- s save_filename : save scaling parameters to save_filename
- r restore_filename : restore scaling parameters from restore_filename
- f scalefile : 输出标定后的数据集到该文件

在程序中用一个数组来保存参数。前一个是参数类型，后一个是参数值，数组最后一个元素是待标定的文件名。

```
// 训练数据集标定
String[] argDScale={"-l","0","-f",scalefile,"-s",scaleparam,trainfile};

// 测试数据集标定
String[] argTScale={"-l","0","-f",tScaleFile,"-r",scaleparam,testfile};
```

训练集的标定和测试集的标定要对应。上例中，训练数据集标定后的参数保存在 scaleparam 文件。测试数据集标定时需要读取该文件，作为标定的参数。

(6) 设置 SVM 参数

-s svm_type

它是支持向量机的类型设置有下面几种选择，默认是 0

0 -- C-SVC

```
1 -- nu-SVC  
2 -- one-class SVM  
3 -- epsilon-SVR  
4 -- nu-SVR  
-t kernel_type
```

设置核函数的类型，默认是 2

```
0 -- linear: u'*v  
1 -- polynomial: (gamma*u'*v + coef0)^degree  
2 -- radial basis function: exp(-gamma*|u-v|^2)  
3 -- sigmoid: tanh(gamma*u'*v + coef0)  
4 -- precomputed kernel (kernel values in training_set_file)
```

-d degree : 设置核函数的度 (degree) 默认是 3

-g gamma : 设置核函数的 gamma 参数 (default 1/num_features)

-r coef0 : 设置核函数的 coef0 (default 0)

-c cost : 设置参数 C , 为 C-SVC, epsilon-SVR, and nu-SVR (default 1)

-n nu : 设置参数 nu , 为 nu-SVC, one-class SVM, and nu-SVR (default 0.5)

-p epsilon : 设置损失函数的 epsilon 参数 , 为 epsilon-SVR (default 0.1)

-m cachesize : set cache memory size in MB (default 100)

-e epsilon : set tolerance of termination criterion (default 0.001)

用一个数组存储设置的 SVM 参数

```
// svm 参数  
String[] argTrain={"-s","0","-t","2","-g","4","-c","2","-w1","2",scalefile,modelfile};
```

Scalefile 是训练集标定后的文件 ; modelfile 是训练后得到的模型文件。这些文件只需给文件名即可 , 他们是中间文件 , 你不需要知道内容。

(7) 训练一个 SVM

```
try{
```

```
    svm_scale.main(argDScale);
    svm_train.main(argTrain);
}catch(Exception e){
    e.printStackTrace();
}
```

(8) 使用 SVM 进行预测

给出预测参数，它是一个数组

```
String[] argPred={tScaleFile,modelfile,predfile};
```

tScaleFile 是测试集标定后的文件， modelfile 是 svm 训练后得到的模型文件。 predfiles 预测结果文件

```
svm_predict.main(argPred);
```

(9) 关于支持向量机的参数调优

K-fold validation K 折交叉确认函数网格搜索。

以常用的 10 折交叉确认为例，将训练集分成 10 份，循环 10 次，每次拿出没用过的一份作为测试集，剩下的 9 份作为训练集。然后 10 测试结果求平均值作为最后的测试结果。

对于网格搜索，以 c-svm , RBF 径向基核函数为例，有两个重要的参数 C 和 γ 。网格搜索即试着用各种(C , γ)的组合进行 k 折交叉确认的实验。选择实现性能最好的 (C 和 γ) 参数对。 Libsvm 建议了一个指数增长的 (C 和 γ) 序列

(C = $2^{-5}, 2^{-3}, \dots, 2^{15}$, $\gamma = 2^{-15}, 2^{-13}, \dots, 2^3$) 。

在 libsvm 提供的 guide.pdf 文件中提及建议初学者使用 SVM 的步骤：

- 转换数据到 SVM 需要的格式
- 对数据进行标定
- 考虑使用 RBF 核
- 使用交叉确认发现最好的参数 C 和 γ

- 使用最好的 C 和 γ 在整个训练集上训练一个 SVM 模型
- 测试

使用 LibSVM 的例子

在 java 中使用 LibSVM 时，为了把 LibSVM 集成到自己的 Java 程序中，我把 LibSVM 重新安排了一下，参见我给的 Eclipse 工程 adsFilter。该工程中把 LibSVM 中的程序安排到了两个包 `libsvm.classifier` 和 `libsvm.svm`。我们会调用 `libsvm.classifier` 包下面的程序建立支持向量机，而 `libsvm.classifier` 中的程序会调用 `libsvm.svm` 中的程序。另外，原来版本的文件将标定后的文件输出到控制台，必须用命令的方式保存。我修改了 `svm_scale.java` 程序，保存标定结果到文件。

`adsFilter` 是用 NB 和 SVM 实现的文本过滤（分类）的程序。`Vars.properties` 文件中进行参数设置。运行 `qjt.adsFilter` 包中的 `Main` 方法来运行程序。

以后大家如果构建自己的基于 LIBSVM 构建自己的应用。建议把我的 `adsFilter` 工程中的 `libsvm.classifier` 和 `libsvm.svm` 复制到你自己的工程下面。这样可以方便地使用 LibSVM。（我这种方法就是直接使用了 Libsvm 的源码，而没使用它提供的*.jar 文件）

第五节：文本分类器总结

1. 文本分类器的选择。

(1) 当建立一个分类器时，首先考虑的问题就是：训练数据有多少。如果拥有的训练数据非常少，而又要训练出一个有监督的分类器。机器学习理论指出，此时应该选择具有高偏差的分类器。例如，理论和经验都表明，这种情况下 NB 具有很好的效果。不管怎样此时如 KNN 的低偏差模型不可取。但无论采用何种模型，模型的质量始终都会因数据有限而受到不利影响。

(2) 如果拥有的数据很多，但标注的数据很少，可以采用半监督的训练方法。NB 的优点是可以直接扩展成半监督的方法。也有研究半监督 SVM 的工作，称为 transductive SVM。如果拥有较多的标注数据，可以使用前面讲述的任何文本分类技术。如果具有极大规模的数据，分类器的选择对最后的分类结果没有什么影响。一个通用的法则是，训练数据每增长一倍，那么分类器的效果将得到线性的提高。

(3) 无论是检索还是文本分类，其默认的特征都是词项。但对文本分类来说，如果对特定的问题加入额外的特征，那么分类效果会显著提高。例如，文档中是否含有化学

分子式，是否有时间日期，或 ISBN 等。另外，文档通常都包含了域。在文本分类中，利用这些域也可以提高分类性能。

2. 分类效果的提高

对于一个特定的应用来说，分类器的效果往往具有显著提升的空间。这可以通过使用领域知识，或数据集相关特征来实现。文档往往包含对分类特别有用的域，也往往可以通过对特定的子词汇表进行特殊对待，从而对分类器的效果进行优化。

	NB	Roc-	Dec.	kNN	linear SVM		rbf-SVM
		chio	Trees		C = 0.5	C = 1.0	$\sigma \approx 7$
earn	96.0	96.1	96.1	97.8	98.0	98.2	98.1
acq	90.7	92.1	85.3	91.8	95.5	95.6	94.7
money-fx	59.6	67.6	69.4	75.4	78.8	78.5	74.3
grain	69.8	79.5	89.1	82.6	91.9	93.1	93.4
crude	81.2	81.5	75.5	85.8	89.4	89.4	88.7
trade	52.2	77.4	59.2	77.9	79.2	79.2	76.6
interest	57.6	72.5	49.1	76.7	75.6	74.8	69.1
ship	80.9	83.1	80.9	79.8	87.4	86.5	85.8
wheat	63.4	79.4	85.5	72.9	86.6	86.8	82.4
corn	45.2	62.2	87.7	71.4	87.5	87.8	84.6
microavg.	72.3	79.9	79.4	82.6	86.7	87.5	86.4

图 5-21：各种分类器的实验比较结果

第六节：使用 mallet 进行文本分类实验

Mallet 是 Information Extraction and Synthesis Laboratory, Department of CS, UMass, Amherst 开发的一款用于统计自然语言处理的 Java-based package。可以进行文本分类，聚类，话题模型，信息抽取，和其他的机器学习在文本领域的应用。

使用 mallet 先去网站下载 mallet。新建一个 eclipse 项目，然后将 mallet 的 dist 包下的 mallet.jar 和 mallet-deps.jar 文件添加到新建的一个 eclipse 项目的 class path。

1. Mallet 处理的文件格式

Mallet 可以处理三种格式的数据：

- (1) 每个文件夹下保存相同类别标签的文件。文件夹名就是类别标签。
- (2) Trec 格式的数据。所有的数据集保持在一个文件中，一行即一个文件。一行的格式如下：

FirstToken\tSecondToken\tRest

FirstToken 是文件编号（或文件名）。SecondToken 是类别标签。Rest 是该行的文本内容。没有 FirstToken 也可以。

(3) Svmlight 格式：即 LibSVM 用的数据格式

-1 1:0.43 3:0.12 9284:0.2 # abcdef

2. 文档预处理

Mallet 将待处理的每篇文档转换成 instance 对象。一个 instance 对象包含四部分内容：

Name : 用于确定一条 instance 的字符串；Label : 类别标签；Data : 一个特征向量（一篇文档被转换成特征向量）；Source: 原始文档，经常是 null。

文档集中的每篇文档被处理后，会保存在一个 InstanceList 对象中。预处理就是将文档集合（训练集、测试集）转化成特征向量，再转换成 instance，得到 InstanceList。

Mallet 采用 pipe 类进行预处理。在 cc.mallet.pipe 包下提供了多种预处理方法。为每种预处理方法建立一个 pipe 类对象，例如：

```
Pipe p=new pipe(new Target2Label());
```

该 pipe 对象完成获得一篇文档类别标签的功能。当进行预处理过程很多时，需要遵循一个顺序。因此可以创建完成多种预处理的 pipe 对象数组，作为一个 SerialPipes 对象的参数，使得一个预处理的结果是下一个预处理步骤的输入。顺序决定了预处理步骤，因此顺序不能错。

通常 mallet 采用的代码是：

```
Pipe instancePipe = new SerialPipes (new Pipe[] {  
    // Target String -> class label  
    new Target2Label (),  
  
    // Data File -> String containing contents  
    new Input2CharSequence (),  
  
    // Data String -> TokenSequence  
    new CharSequence2TokenSequence (),
```

```

// TokenSequence words lowercased
new TokenSequenceLowercase (),

// Remove stopwords from sequence
new TokenSequenceRemoveStopwords (),

// Replace each Token with a feature index
new TokenSequence2FeatureSequence(),

// Collapse word order into a "feature vector"
new FeatureSequence2FeatureVector(),
});


```

InstanceList 构造方法的参数是上面创建的 pipe 对象。 InstanceList 对象下有个 addThruPipe 方法， 其参数是某一类待处理的文档集合，如此可以将每篇文档按照创建 pipe 时的预处理方顺序进行处理，从而获得一篇文档的 Instance，进而整个文档集合的 InstanceList。示例代码如下：

```

String trainfile="c:/qjt/data/reuter R8/r8-train-no-stop-id.txt";
InstanceList trainList = new InstanceList (instancePipe);
trainList.addThruPipe(new
CsvIterator(trainfile,"(\w+)\s+([\w-]+)\s+(.*", 3, 2, 1));

```

通常需要分别为训练集和测试产生一个 InstaceList。由训练集的 InstanceList 产生分类器，然后分类器在测试集上测试。

3. 创建分类器

首先创建分类器训练器，然后由训练器创建分类器。如创建一个 NB 分类器的训练器

```
ClassifierTrainer<NaiveBayes> naiveBayesTrainer = new NaiveBayesTrainer();
```

Mallet 可以创建多种的分类器，参看 mallet API 的 **Classifier** 类。

调用 Trainer 的 train 方法，以 InstanceList 对象作为参数，就可以训练得到一个分类器。

```
Classifier classifier=naiveBayesTrainer.train(trainList);
```

分类器包含的一些方法如下：

(1) Classify()，其参数可以是一个 instance，一个 instance 数组，或一个 InstanceList，返回分类结果。

(2) getAccuracy(InstanceList testlist)，对测试集进行分类测试，返回 accuracy 评测值。

(3) getF1 (testList, int index))，返回分类测试的 F 值，index 是当你的分类器是对多个类别建立分类器时，你的测试集是用那个分类器做分类测试的 F 值。index 的值是从 0 开始。

可以用下面的代码查看分类器中有哪些类别的分类器

```
LabelAlphabet la=classifier.getLabelAlphabet();
System.out.println(la.toString());
```

4. 特征选择

FeatureSelector 类可以创建特征选择器。其构造方法规定了选用的特征选择方法和选择个数。Mallet 有四种特征选择方法（参看 **Interface**

RankedFeatureVector.Factory）

ExpGain.Factory, FeatureCounts.Factory, GradientGain.Factory, InfoGain.Factory

```
FeatureSelector fselector=new FeatureSelector(new InfoGain.Factory(),300);
```

下面的代码从一个 InstanceList 获得选择的特征，并查看

```
fselector.selectFeaturesFor(trainList);
FeatureSelection fs=trainList.getFeatureSelection();
BitSet bs=fs.getBitSet();
System.out.println(bs.toString());
```

然而， InstanceList 中的特征向量并没有因进行了特征选择而改变。若想，根据选择的特征改变 InstanceList 中的特征向量可以进行如下操作。

```

int len=trainList.size();

Instance in;

for(int i=0;i<len;i++){

    in=trainList.get(i);

    FeatureVector f=(FeatureVector)in.getData();

    FeatureVector d=FeatureVector.newFeatureVector(f, f.getAlphabet(),
trainList.getFeatureSelection());

    in.unLock();

    in.setData(d);

    in.lock();

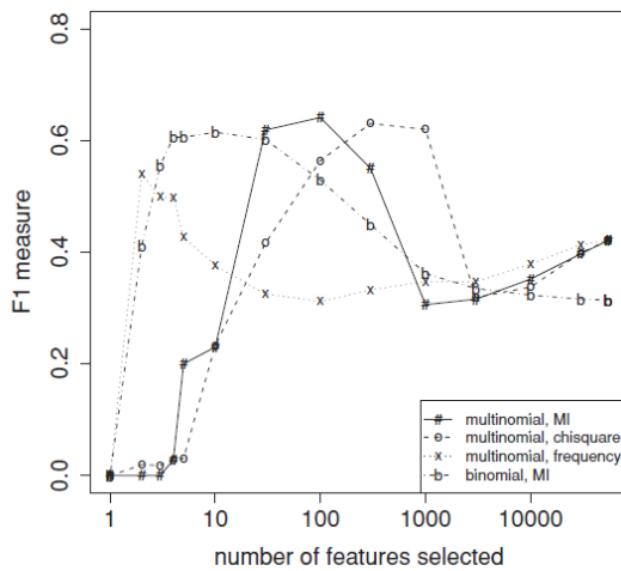
}

```

详细代码参看我给你们的 MalletTest.java 文件

实验：

用 mallet 的 NB 分类器、Reuters R8 语料库中的训练集文档通过特征选择后建立 NB 分类器。在测试集上进行操作，查看各种特征选择方法和特征数对分类性能的影响。看看能得到下面的图吗。



第六章：文本聚类

第一节：聚类算法介绍

聚类算法将一系列文档聚类成多个子集或簇（cluster），其目标是建立类内紧密类间分散的多个簇。即聚类的结果要求簇内的文档之间要尽可能相似，而簇间的文档则要尽可能不相似。乍一看，聚类和分类很相似。都是将文档分配到不同的组中。但分类任务需要使用训练集产生分类器，称为有监督学习。而聚类任务不需要训练集，称为无监督学习。

聚类同分类一样，关键是计算文档间的距离。通常采用欧式距离计算方法。聚类又分为扁平聚类和层次聚类。扁平聚类会给出数据的一个聚类结果；而层次聚类会展现一棵聚类树，在树的每个层次上，展现不同的聚类结果。

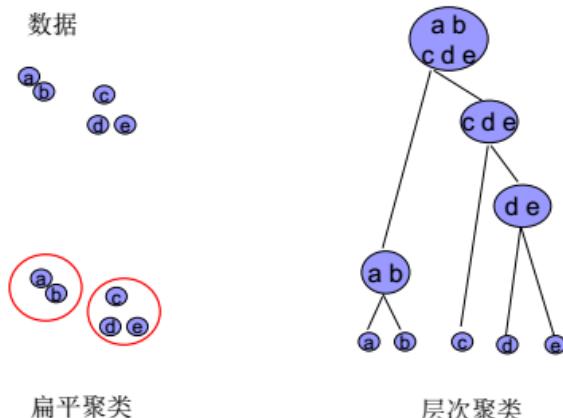


图 6-1：扁平与层次聚类算法

硬聚类和软聚类

硬聚类将每篇文档分配一个标签，即每个文档只能且必须属于一个类。而软聚类中，每篇文档以不同的隶属度属于每个类

聚类问题的描述

硬扁平聚类的目标可以定义如下：

(1) 给定一系列文档 $D=\{d_1, \dots, d_n\}$

(2) 给定期望的簇数目 K

(3) 给定用于评估聚类质量的目标函数

(4) 计算一个分配映射， $\gamma: D \rightarrow \{1, \dots, K\}$ ，该分配下的目标函数值极大化或极小化。

目标函数通常基于文档的相似度或者距离来定义。

通常，对于文档而言“相似度”是指文档在主题上的相似。采用前面介绍的向量余弦相似度或欧式距离的计算方法就可以计算。

如果计算的相似度不是主题上的相似度，需要采用别的计算方法。例如，如果是为了对不同语言的文档进行语言相似度的计算。会需要考虑停用词。将英语文档和法语文档区分开时，停用词很重要。

聚类的势—簇的数目

聚类算法中的一个难点是如何确定簇的数目或者说聚类的势。通常聚类的数目 K 是一个基于领域知识的经验值。

基于聚类的目的是对目标函数优化。因此聚类实际上是一个搜索问题。穷举法可以获得所有可能的聚类结果。但实际上不可行。因此大部分扁平聚类算法会在初始划分结果的基础上不断的迭代更新，直到获得最终结果。如果出现初始点选择不当，那么可能不会达到全局最优的结果。因此扁平聚类的另一个重要问题是选择好的初始点。

聚类算法的评价

聚类算法的典型目标函数是将簇内高相似度以及簇间低相似度的目标形式化后得到的一个函数。《信息检索导论》16.3 节介绍的是四种外部准则，即聚类结果和已有标准分簇结果的吻合程度。提个问题：如果已知分簇结果，为什么还要聚类操作？

这是在对聚类算法进行测评时，已有的训练集中的文档做了标注。

我们介绍其中三种聚类质量的外部准则

(1) 纯度 purity

纯度是一个简单明显的评价指标。计算纯度时，每个簇被分配给该簇当中出现数目最多的文档所在的类别，然后可以通过正确分配的文档数除以文档集中的文档总数 N 来得到该分配的精度。其公式如下：

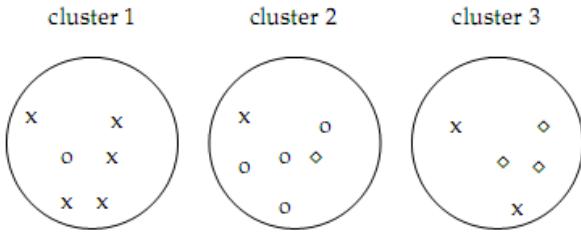


图 6-2 : 纯度计算示意图

$$\text{purity}(\Omega, \mathcal{C}) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|$$

其中 $\Omega = \{\omega_1, \omega_2, \dots, \omega_K\}$ 是聚类结果 , $\mathcal{C} = \{c_1, c_2, \dots, c_J\}$ 是类别的集合。完美的聚类结果的纯度计算应该为 1 , 差的结果趋近于 0。

图 6-2 的例子中 , 纯度为 $(5+4+3)/17 \approx 0.71$

(2) RI (Rand Index) 兰德指数

RI 计算的是正确决策的比例 , 实际上它就是 Accuracy。

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

这里 , TP (true-positive) 将两篇同一类的文档归入同一个簇。TN 将两篇不同类的文档归入不同的簇。FP 将两篇不同类的文档归入同一簇。FN 将两篇同一类的文档归入不同的簇。

以图 6-2 为例

$$TP = \binom{5}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2} = 20$$

$$TP + FP = \binom{6}{2} + \binom{6}{2} + \binom{5}{2} = 40$$

$$FP = 40 - 20 = 20$$

$$FN = C_5^1 + C_5^2 + C_4^1 + C_2^1 + C_3^1 = 24 ; TN = C_{17}^2 - TP - FP - FN = 72$$

$$RI = (20 + 72) / (20 + 20 + 24 + 72) \approx 0.68$$

(3) F 值

$$P = \frac{TP}{TP + FP} \quad R = \frac{TP}{TP + FN} \quad F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

TP , FN 和 FP 等的计算同里 RI 的计算。

第二节: K-Means

K-means 算法是最重要的扁平聚类算法，它的目标是最小化文档到其簇心的欧式距离平方的平均值。其中，簇心的定义为簇 ω 中文档向量平均值或者质心 $\vec{\mu}$ 。

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x}$$

这里文档向量为长度归一化的向量。RSS 残差平方和是 K-means 算法的目标函数。K-means 聚类的目的就是让这个函数取最小值。

$$RSS_k = \sum_{\vec{x} \in \omega_k} |\vec{x} - \vec{\mu}(\omega_k)|^2 ; \quad RSS = \sum_{k=1}^K RSS_k$$

K-means 算法初始随机选择 K 篓文档构成初始簇中心。然后，算法不断移动簇中心，以使得 RSS 极小化。图 16-5 的伪代码可以通过反复迭代执行下列两步直至满足停止条件：

(1) 先将文档重新分配到距它最近的质心所在的簇。

(2) 基于簇中目前的文档重新计算质心。

K-means 的算法描述如下：

```
K-MEANS( $\{\vec{x}_1, \dots, \vec{x}_N\}, K$ )
1  $(\vec{s}_1, \vec{s}_2, \dots, \vec{s}_K) \leftarrow \text{SELECTRANDOMSEEDS}(\{\vec{x}_1, \dots, \vec{x}_N\}, K)$ 
2 for  $k \leftarrow 1$  to  $K$ 
3   do  $\vec{\mu}_k \leftarrow \vec{s}_k$ 
4   while stopping criterion has not been met
5   do for  $k \leftarrow 1$  to  $K$ 
6     do  $\omega_k \leftarrow \{\}$ 
7     for  $n \leftarrow 1$  to  $N$ 
8       do  $j \leftarrow \arg \min_{j'} |\vec{\mu}_{j'} - \vec{x}_n|$ 
9          $\omega_j \leftarrow \omega_j \cup \{\vec{x}_n\}$  (reassignment of vectors)
10    for  $k \leftarrow 1$  to  $K$ 
11      do  $\vec{\mu}_k \leftarrow \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} \vec{x}$  (recomputation of centroids)
12  return  $\{\vec{\mu}_1, \dots, \vec{\mu}_K\}$ 
```

K-means 算法中迭代终止条件可以有多种考虑：

(1) 当迭代一个固定次数后停止。该条件可以限制算法的运行时间，但有些情况由于迭代次数不足，聚类的结果质量并不高。

(2) 当文档到簇的分配结果不再改变后停止。通常情况该方法可以产生较好的聚类结果。

(3) 质心向量不再改变后停止，等价于第二个方法。

(4) RSS 低于某个阈值时停止。该条件能够保证停止后的聚类结果具有一定的质量。在实际使用中，必须将它和迭代次数一起使用以保证迭代能够停止。

(5) 当 RSS 的减小值低于某个阈值停止。这意味着迭代接近收敛。同样这个条件需要和迭代次数限制条件一起使用，以避免运行时间太长

Kmeans 有它的局限：

(1) kmeans 是采用欧式距离来计算数据点到质心的距离，因此簇在空间上的形状是 hyper-spherical，kmeans 可以工作的很好，如果不是则 kmeans 不是一个好的选择。

(2) 需要设置簇的个数 k 。 k 的选择如果不合适可以产生很差的聚类结果。

(3) 如果 kmeans 的初始点选择不合适会陷入局部最优。图 6-3 是一个 Kmeans 算法陷入局部最优的例子。

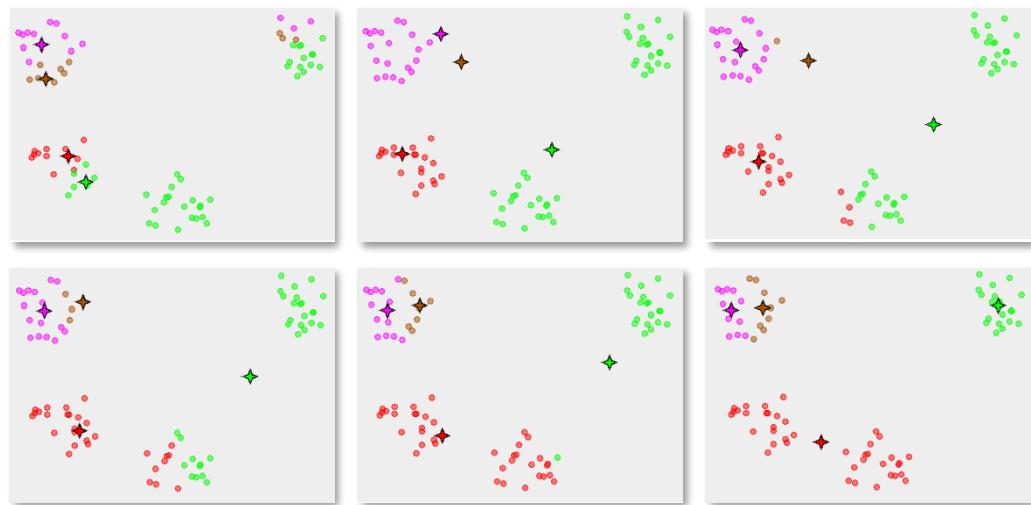


图 6-3：K-means 陷入局部最优的例子(执行顺序是从左到右，从上到下)

《信息检索导论》P251 证明了 K-means 算法的收敛性。即使用 K-means 算法进行迭代肯定能得到一个聚类结果。但该结果不一定能是全局最优的。特别是包含离群点（孤立点）。通常来说如果离群点被选作是初始的簇中心，那么在其后的迭代中不会

有任何其他的向量被分配到该簇中。所以，虽然最后算法收敛了，但最后得到的是一个只包含一篇文档的单点簇。

选取种子文档（初始簇心）的启发式策略

- (1) 从初始集合中排出离群点
- (2) 尝试多种可能初始点，并选作代价最小的聚类结果。
- (3) 借助其他方法，如层次聚类来获得种子。
- (4) 为每个簇选作 i 个（如 10 个）随机向量，然后将它们的质心向量作为该簇的初始质心。该方法在文档聚类中具有鲁棒性。

K-means 算法中簇的个数

对于扁平聚类算法簇的个数 K 是预先确定的，当不能合理的确认 K 的取值时，应该怎么处理？一个处理该问题的启发式方法是采用图 6-4 的方法。在每个 K 的取值上，计算 $RSS_{min}(K)$ 。 $RSS_{min}(K)$ 是说对某个 k 的取值，例如 $k=3$ 时，我们运行 i 次，例如 10 次。每次随机选择初始的质心。然后 10 次聚类中最小的 RSS 值，即 RSS_{min} 。画成图后，发现曲线变化平坦的拐点。

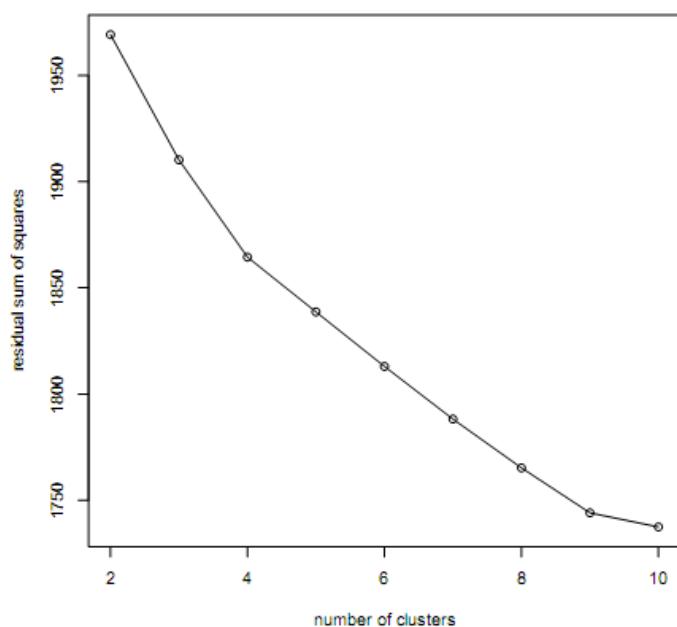


图 6-4：k-means 中，簇的个数选择

为什么要发现拐点？拐点的含义是 RSS 的减小趋于平缓的点。定性的分析也就是说，增加了一个 k 值后，不能让 RSS 明显下降。因为我们不能追求 RSS 越小越好，因此就试图发现拐点，作为选择的 K 值。

图 6-4 有两个曲线平缓的拐点，表明只存在一个最佳簇数目的情况并不常见。提个问题：既然按照此图，曲线所示的 RSS 一直在减小，为什么不选择最小的 RSS 值时对应的类的个数？

一个最自然的想法是，选作使得 RSS 极小化的 K 值。但是，当 K=N（文档的数目），RSS 会取得最小值 0。

而对应两个拐点中间的 k 值，因为它们不是拐点，因此还有让 RSS 再有意义的下降的空间（有意义的下降是说，增加 k 值可以得到明显的 RSS 的下降）。

第二种确定簇数目的准则是对每个新簇予一定的惩罚（聚类结果的簇越多惩罚越重）。这种做法引入一个包含两个要素的一般化目标函数：一个是失真率，衡量文档和他们的簇原型之间的偏离成都（比如 K-means 中的 RSS）；另一个要素是模型的复杂度（这里将聚类结果解释为一个模型，则簇的数目是模型复杂度）。对于 K-means 算法 K 的选择准则如下：

$$K = \arg \min_K [\text{RSS}_{\min}(K) + \lambda K]$$

Lambda 是权重因子，lambda 取较大值意味着簇的数目倾向于取较小的值，lambda=0 表示并不对簇的数目进行惩罚。但是，确定 lambda 是一个问题。

以我个人的经验，可以选择逐步增大 k 值，进行实验。对于每个 k 值，做多次实验，当聚类结果趋于不稳定时，可以停止实验，此时的 k 值，作为最终选择的值。

第三节：层次聚类

扁平聚类具有概念简单，速度快的优点，但是同时有很大缺点：它返回的是一个扁平簇集合；它需要预先设定簇的数目。层次聚类不需要事先指定簇的数目，可以输出一个层次结构的簇集合。

层次聚类可以是一个自底向上或自顶向下的过程。自底向上的算法一开始将每篇文档都看成是一个簇，然后不断的利用某种方法将簇进行合并，这种方法又称为凝聚式层次聚类。自顶向下方法，首先将所有文档看做是一个簇，然后不断的应用某种方法将簇进行分裂，直到每篇文档都成为一个簇。该方法又称为分裂式层次聚类。

凝聚式层次聚类 HAC 的结果往往采用如图 6-5 的树状图来描述。

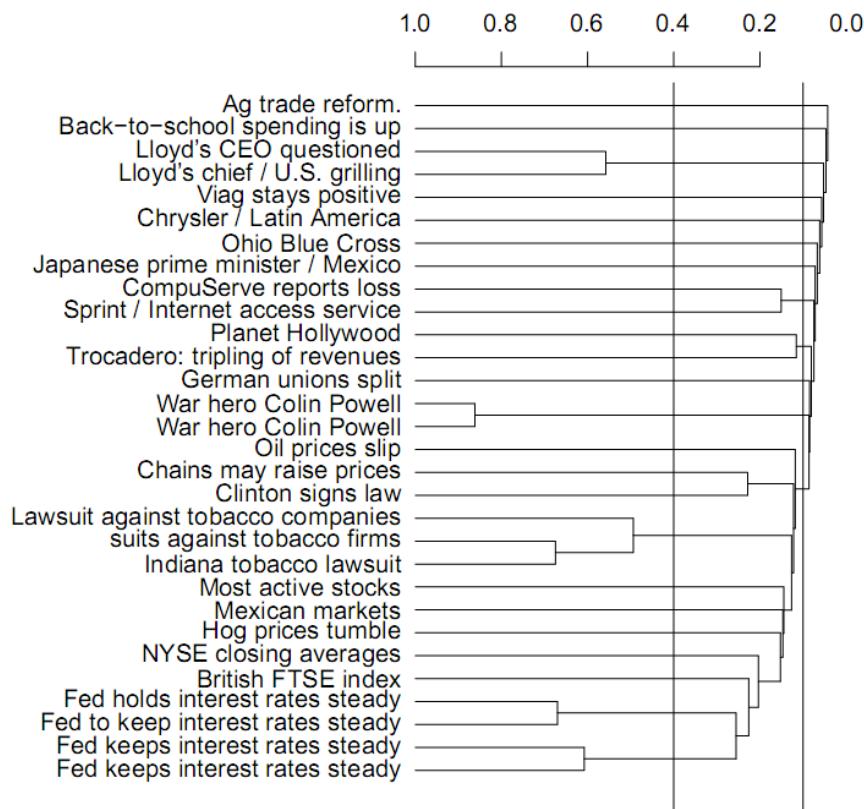


图 6-5：凝聚式层次聚类结果

凝聚式层次聚类，在每次迭代中找出当前相似度最高的两个簇合并。计算两个簇的相似度，有下面的方法

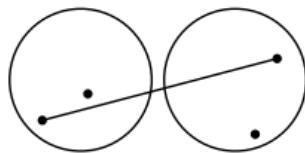
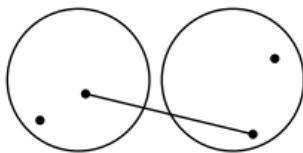
- (1) 单连接聚类，两个簇的相似度定义为两个最相似的成员间的相似度。
- (2) 全连接聚类，两个簇间相似度定义为，两个最不相似成员之间的相似度
- (3) 质心聚类，通过两个簇的质心相似度来定义簇的相似度
- (4) 组平均 GAAC 方法，通过计算两个组所有文档之间的相似度来计算簇的相似度

对于四种计算簇相似度的方法，我们不详细讨论，直接给出结论：

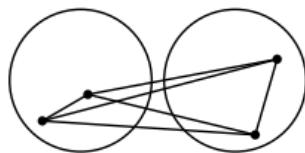
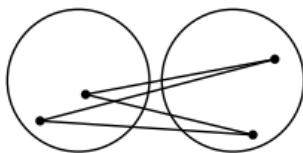
GAAC 通常产生的聚类结果最优。推荐在文档聚类中使用 GAAC 方法。

层次聚类不需要预先给出簇的数目，但有时，我们希望得到像扁平聚类的那样结果，即某一个聚类的划分。这时，需要在聚类树上的某一点进行截断（取其截面）。有多种做法：

(1) 在事先给定的相似度水平上进行截断。比如，我们希望簇的结合相似度不低于 0.4 的话，在 0.4 处的截面如图 6-5 所示。



(a) single-link: maximum similarity (b) complete-link: minimum similarity



(c) centroid: average inter-similarity (d) group-average: average of all similarities

图 6-6：簇的相似度计算

(2) 当两个连续的聚类结果的结合相似度之差最大时截断。这意味着多增加一个簇后聚类质量会显著下降。如图 6-5，0.4 处的截面会产生最大相似度差值。

(3) 使用公式

$$K = \arg \min_{K'} [\text{RSS}(K') + \lambda K']$$

(4) 预先给定聚类个数

第四节：高斯混合模型和有约束的高斯混合模型

以我的经验，高斯混合模型（Gaussian Mixed Model, GMM）比 kmean 性能要更好。在我给的 matlab 示例程序 test_gmm.m 可以演示 kmeans 和 GMM 在聚类上的性能差异。

GMM 形式化描述如下。设 $X=\{x_1, \dots, x_N\}$ 是一个数据集。GMM 是 K 个高斯密度函数的叠加。公式如下

$$p(x) = \sum_{k=1}^K \pi_k N(x|\mu_k, \Sigma_k)$$

每个高斯密度函数 $N(x|\mu_k, \Sigma_k)$ 称作混合模型的一个构件 (component)。每个构件都有一个均值 μ_k 和一个协方差矩阵 Σ_k 。参数 π_k 是混合系数，满足 $0 \leq \pi_k \leq 1$ ，且 $\sum_{k=1}^K \pi_k = 1$ 。高斯混合模型已经被证明在聚类任务中很优秀。

GMM 中一个构件描述一个聚类的簇。后验概率 $p(k|x)$ 指示数据点 x 属于簇 k 的概率。它们满足 $\sum_{k=1}^K p(k|x) = 1$ 。要从数据集 $X=\{x_1, \dots, x_N\}$ 学习一个 GMM，我们首先引入一个 K 维二元随机变量 z 。它是一个 1-of- K 描述，即 z_k 的值为 1，所有其他元素的值为 0。条件概率 $p(z_k=1|x)$ 用于描述样本数据点 x 被 GMM 的构件 k 产生的概率。使用最大似然法从数据集估计 GMM 的参数，我们可以使用期望最大化算法 (Expectation-Maximization, EM)。GMM 的似然函数如下

$$\ln p(\mathbf{X} | \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k N(x_n | \mu_k, \Sigma_k) \right\}$$

EM 算法包含一个 E 步骤和一个 M 步骤。当使用 EM 算法训练 GMM 时，EM 算法的 E 步骤计算条件概率 $p(z_k=1|x)$

$$\gamma(z_{nk}) = p(z_k = 1|x_n) = \frac{\pi_k N(x_n | \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j N(x_n | \mu_j, \Sigma_j)}$$

M 步骤使用当前的条件概率 $p(z_k=1|x)$ 重新计算三个参数：

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{new})(\mathbf{x}_n - \boldsymbol{\mu}_k^{new})^T$$

$$\pi_k^{new} = \frac{N_k}{N}$$

我们可以把 GMM 理解成功能多了一些的 k-means。多出的功能是计算协方差矩阵。在 kmeans 计算每个数据和质心的距离的步骤，GMM 计算后验概率 $p(z_k=1|x)$ ；在 kmeans 在分配每个数据到距离它最近的质心后，重新计算质心的步骤，GMM 计算均值 μ 、协方差 Σ 和簇的先验概率 π 。EM 算法适用于有隐变量的模型参数估计。GMM 中 z 就是隐变量。

有约束的 GMM

有约束的聚类也称为半监督聚类。通常有一些约束条件，它们被描述成 must-link 和 cannot-link 两个约束集合。即规定了哪些数据必须在一个簇中 (must-link)，哪些数据不能在一个簇中 (cannot-link)。有约束聚类根据约束条件将所有的数据聚成多个簇。我试过多个半监督聚类算法但他们都不能满足我们的任务，我自己开发一个有约束的 GMM。

传统的聚类任务中我们不知道每个 data point 的类别，即所有的数据是没有标签的。一个有约束的 GMM 是说，我们已经知道有一部分的数据需要满足一定的约束条件。在实施聚类的过程中，需要用这部分约束条件去影响聚类过程。这样的约束条件通常用 must-link 和 cannot-link 来描述。Must-link 和 cannot-link 是分别是一个 pairwise-relation 的集合。每条关系描述了一个数据对 $\langle I_1, I_2 \rangle$ 。Must-link 中，规定数据 I_1 和 I_2 必须属于同一个簇；cannot-link 中的一个数据对 $\langle I_3, I_4 \rangle$ 描述 I_3 和 I_4 不能属于同一个簇。

实施这样一个有约束的 GMM，我的思路是

约束条件是用 pairwise relation 来描述的。例如，有 must-link 约束 $C_{ML}=\{(x_i, x_j)\}$ 和 cannot-link 约束 $C_{CL}=\{(x_i, x_j)\}$ 。此处 $(x_i, x_j) \in C_{ML}$ 指示 x_i 和 x_j 必须属于同一个簇，而 $(x_i, x_j) \in C_{CL}$ 指示 x_i and x_j 必须属于不同的簇。

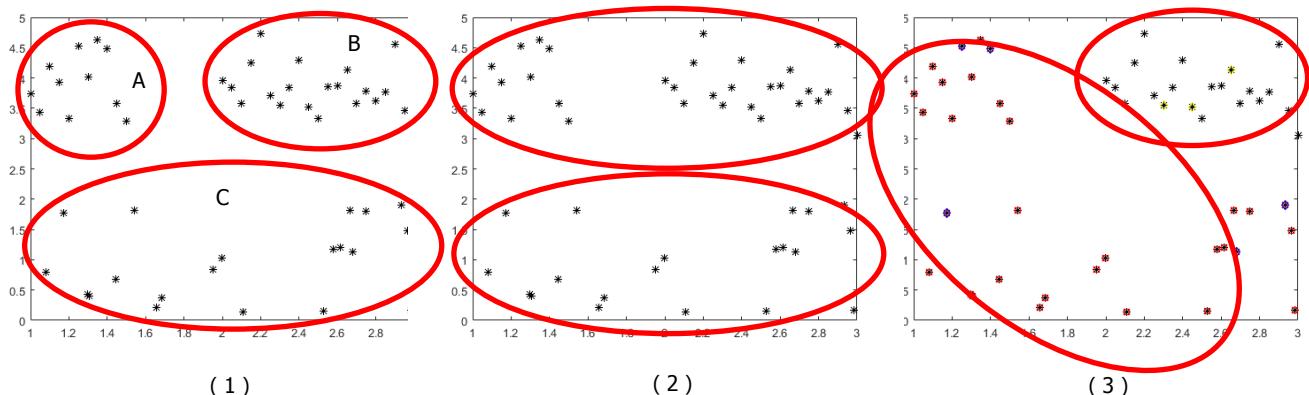


图 6-7：有约束的聚类示例

图 6-7 是有约束聚类的示例。数据集由 A、B、C 三部分数据构成。如果使用传统的 GMM 进行聚类操作，其中 $K=2$ 。GMM 将 A 和 B 数据子集聚为一类，C 数据子集聚为一类。聚类结果为图 6-7 (2) 所示。当我们想让 A 和 C 数据子集聚为一类，而让 B 数据子集单独聚为一类。这时，我们给 A 和 C 数据子集中的部分数据设置 pairwise constraints，要求它们在 must-link 约束中。而在 B 数据子集中选择一部分数据也设置 must-link 约束。同时设置两部分节点为 cannot-link 约束。最后聚类结果如图 6-7 (3) 所示。有约束的 GMM 将 A 和 C 数据子集聚为一类，而 B 数据子集单独为一类。图 6-7 (3) 中的蓝色和黄色数据的是被选择的约束数据点。相同的颜色指示数据点必

须在同一个簇中，不同颜色的数据点指示应该聚类在不同的簇。红色和黑色的数据点指示聚类结果。

在传统的 GMM 中，E 步骤计算 $\gamma(Z_{ki}) = p(Z_{ki} = 1|x_i)$ ，即一条数据 x_i 属于类别 k 的概率。在有约束条件的情况下，怎样计算约束条件中的数据点的 $\gamma(Z_{ki})$ ？我们的思考是，给这些约束条件中的数据点分配簇，在分配时尽量让它们不违反约束条件。分配完成后，例如，数据 x_i 当确定分配给簇 k 后，分配 $\gamma(Z_{ki}) = 1$ ；而 $\gamma(Z_{mi}) = 0, m \neq k$ 。

因此，E 步骤的关键是怎么分配 x_i 到哪个簇，而让分配尽量不违反约束条件。我们用 Gibbs 抽样的方法，即从一个关于数据 x_i 属于哪个簇的概率分布，抽样的方式确定 x_i 属于哪个簇，然后再去确定 x_{i+1} 。抽样 x_{i+1} 时是以 x_i 的分配结果来作为条件的，这就是 Gibbs 抽样。即当前数据的分配结果是以前面其他数据的分配结果为基础。当迭代这个过程，最后收敛时，得到的数据结构就是优化的，违反约束条件最少的分配结果。而抽样是基于这个分布公式

$$p(Z_i = k|Z_{-i}) = \frac{\gamma(Z_{ik}) \exp\{\sum_{j \in U} \delta(W_M^{ij}, 1) - \delta(W_C^{ij}, 1)\}}{\sum_{l=1}^K \gamma(Z_{il}) \exp\{\sum_{j \in U} \delta(W_M^{ij}, 1) - \delta(W_C^{ij}, 1)\}}$$

U 是显示在约束条件中数据点集合。 W_M^{ij} 是 must-link 中的逐对约束关系， W_C^{ij} 是 cannot-link 中的逐对约束关系。 $\delta(i, j)$ 是 kronecher delta 函数，满足

$$\delta(W_M^{ij}, 1) = \begin{cases} 1, & W_M^{ij} = 1 \\ 0, & \text{otherwise} \end{cases}$$

该公式计算了数据 x_i 属于每个簇（ k 的取值是从 1 到总共簇的个数）的概率。 $\gamma(Z_{ki})$ 是传统的 GMM 中计算的数据 x_i 属于簇 k 的概率。 $\exp\{\sum_{j \in U} \delta(W_M^{ij}, 1) - \delta(W_C^{ij}, 1)\}$ 是根据约束条件计算的把数据 x_i 分配给簇 k 的代价值。

算法：基于 Gibbs 抽样的有约束 GMM

Input : k

Output:

1. Repeat until convergence
 2. E step
 3. Initializing cluster label collection cl satisfied $\{cl_i = 0 : i = 1, \dots, U\}$
 4. Repeat until cl not change or
 5. Sample $z_1^{(t+1)} \sim p(z_1|z_{-1}^{(t)})$; $cl[1] = z_1^{(t+1)}$
- ...

<p>Sample $z_U^{(t+1)} \sim p(z_U z_{-U}^{(t)})$; $cl[U] = z_U^{(t+1)}$</p> <p>6. End Repeat</p> <p>7. For each data point in constraints U</p> <p>8. Set $\gamma(Z_{i,cl[i]})=1$ and $\gamma(Z_{i,-cl[i]})=0$</p> <p>9. End For</p> <p>10. M step</p> <p>11. End Repeat</p>
--

在我给的 matlab 示例程序 test_sgmm.m 和 sgmm.m 演示了有约束 GMM 的工作过程。

第五节：给文本聚类的簇贴标签

给聚类结果的簇贴标签，可以了解这个簇的文档在谈论什么。话题检测是一个热门的研究领域，它通常认为一篇文档中包含了多个话题，详细内容我们在第 7 章将详细讨论。而当我们假设一个文档只包含一个话题时，其实我们可以用文档聚类的方法来检测话题，也即给簇贴标签。

给簇贴标签有多种方法。一种可以用聚类结果的质心向量来描述当前簇谈论的话题；还可以特征选择的方法，如使用互信息或卡方挑选出当前簇的特征作为当前簇的话题（或标签）。使用质心向量中的高权重词项作为簇的标签会有个问题，他会挑选出一些无意义的词。特征选择是挑选那些可以将当前簇和其他簇差异化的词项。但也容易挑选到一些稀有的词项。将特征选择结合对稀有词项进行惩罚的方法往往可以得到较好的簇的标签。

我们可以这样理解用簇的质心描述簇的话题。有这样一个文档集合，其中的每篇文档用一个特征向量来描述，如果特征向量的权重是 TFIDF 权重。该权重描述了一个特征描述一篇文档的能力。那么，文档聚类的结果获得多个簇，每个簇的质心我们定义为一个话题（图 6-8 中白色的点），抽取质心向量的 top k 个词项，就描述了这个话题。

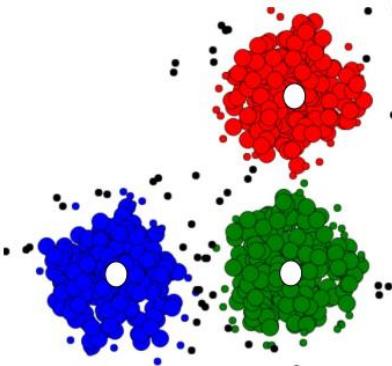


图 6-8：文本集合的聚类结果和质心

使用聚类的方法去发现话题会有一些问题，例如，如果多个簇的质心在某个词项上都会有较高的权重值，则这些簇的质心所描绘的话题会有相同词项。但该方法实现简单。

实践中，建议将每篇文档按照段落被划分成更小的文档。因为通常一个段落的内容是更相关的。如此，在分割后的小文档上实施聚类，再贴标签，更能准确的描述文档集合中的话题。

用 mallet 的 K-means 聚类从文档集合发现话题的步骤如下：

1. 文档集合预处理：将文档集合转换为 InstanceList

```

String trainfile="d:/qjt/data/reuter R8/r8-train-no-stop-id.txt";
Pipe instancePipe = new SerialPipes (new Pipe[] {
    new Target2Label (),// Target String -> class label
    new Input2CharSequence (),// Data File -> String containing
    contents
    new CharSequence2TokenSequence (), // Data String ->
    TokenSequence
    new TokenSequence2FeatureSequence(),// Replace each Token with
    a feature index
    new FeatureSequence2FeatureVector(),// Collapse word order into
    a "feature vector"
});
InstanceList trainList = new InstanceList (instancePipe);
try {
    trainList.addThruPipe(new
        CsvIterator(trainfile,"(\w+)\s+([\w-]+)\s+(.*")", 3, 2, 1));
}

```

```

} catch (FileNotFoundException e) {
    e.printStackTrace();
}

```

同样的它需要使用 Pipe 类，来建立文档预处理的模型。上面在 Pipe 类的对象 instancePipe 中，添加了下面的预处理方法：获得类别标签、将文档转化成词条序列 TokenSequence、转化成特征序列、转化成特征向量。

2. 建立 KMeans 对象

```

KMeans kmeans=new KMeans(instancePipe,8, new
NormalizedDotProductMetric());
Clustering clusters=kmeans.cluster(trainList);

```

KMeans 类在 cc.mallet.cluster.KMeans 包中。它的构造方法包括三个参数

第一个是上述建立的进行文档预处理的 Pipe 类的对象，k-means 中的 k 值（示例程序取值 8），和进行两个向量距离或相似性度量的方法的对象。Mallet 有两种计算方法：

[Minkowski](#), [NormalizedDotProductMetric](#)

两个向量的 Minkowske 距离

$$\left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

NormalizedDotProductMetric 就是余弦相似度

用 kmeans 对象的 cluster 方法

```
Clustering clusters=kmeans.cluster(trainList);
```

可以得到聚类结果，它的参数是预处理后的文档集合 InstanceList

3. 从聚类结果中获得，每个簇的质心（簇的均值）

```
ArrayList<SparseVector> alist=kmeans.getClusterMeans();
```

返回的结果是存储了每个簇的质心（SparseVector 类）的线性表（ArrayList 类）

4. 从一个质心中找出 Top k 个特征，他们就是一个话题

(1) 首先，我们获得每个词典

```
Alphabet al=trainList.getAlphabet();
```

文档集合预处理后建立的 InstanceList 类包含了一个词典。InstanceList 的 getAlphabet 方法可以获得词典。

(2) 获得质心向量

每个质心向量的长度是不一样的。因为该向量只存储向量分量的权重（特征的权重分值）大于 0 的向量。但我们可以获得一个索引数组，它描述了，该向量的每个特征的索引值，通过该索引值可以找到他们在词典中对应的词项。也可以找到每个特征的权重。

```
vec=alist.get(i);
indices=vec.getIndices();
val=vec.getValues();
```

(3) 找出 Top n 个特征作为话题

我另外写了一个 SortedList 类，它提供辅助方法，找出 Top n 个特征。其工作原理是：

建立一个线性表。用插入排序的方法向该线性表插入特征。插入后，如果线性表的长度大于了 n，则删除最后的那个特征。插入排序时，用折半查找的方式找到带插入的特征的位置。

下面的 topn 方法将特征插入 SortedList，获得 top n 个词项的索引

```
private int[] topn(int[] indice, double[] val, int n){
    SortedList sl=new SortedList(n);
    if(indice.length!=val.length){
        System.out.println("Error!");
        System.exit(1);
    }
    for(int i=0;i<indice.length;i++){
        sl.insert(indice[i], val[i]);
    }
    int[] r=new int[n];
    for(int i=0;i<sl.size();i++){
        r[i]=sl.get(i).index;
    }
}
```

```
    return r;  
}
```

topn 方法有三个参数：质心向量的索引数组，质心向量的特征权重数组，和 top n 中的 n 值。它返回 top n 个特征的索引。

(4) 根据词典，将 top n 个索引对应的词项显示出来，即获得了话题

```
private void getTopic(SparseVector svec, Alphabet al){  
    double[] val;  
    int[] indice;  
  
    val=svec.getValues();  
    indice=svec.getIndices();  
    indice=topk(indice,val,30);  
    for(int  
i=0;i<indice.length;System.out.print(al.lookupObject(ind  
ice[i])+" "), i++);  
        System.out.println();  
}
```

实验：

我们应用上述基于 kmeans 聚类来检测话题的方法，从路透社 R8 语料库 r8-train-no-stop-id.txt 检测八个话题。每个话题是 Top 10 的词项列表，其结果如下：

1. mln dlrs net year loss reuter cts shr profit kly
2. cts loss net mln shr reuter profit revs qtr hmy
3. oil opec prices dlrs mln crude bpd reuter pct avialable
4. shares dlrs stock company pct share reuter mln common cvt
5. offer usair american company gencorp dlrs twa group reuter terminating
6. pct bank billion year rate reuter banks rates mln midrate
7. trade japan japanese reuter agreement foreign year countries states steeply
8. company reuter corp dlrs mln unit sale merger acquisition

第六节：使用聚类算法的误区

聚类算法是无监督学习，它适合用来考察数据本身的特性。即，通过聚类算法来了解获得数据的特性。然而，我评审过的论文中发现一些应用聚类算法不正确的地方。例如，有一个客户购买的数据集。每条数据是一个购买者购买一件商品的记录，包括商品的特征。当前数据没有性别特征，有人认为，当前购买者无非是男性或女性。因此将该数据集聚类操作获得两个簇，则两个簇分别描述了男性和女性购买者。这对吗？

那我又说，这个数据集中的商品只有两类奢侈品和日用品（当前数据没有给出该特征）。那聚类的结果是反映的则两类商品吗？

到底聚成的两个簇反映的是什么特征不是使用聚类算法的人想当然认为的。需要考察簇的质心的特征结合应用背景来推断簇所描述的类。

常见的错误是拿到用户数据。在进行用户细分时，理论上认为用户可以分为四个类。那对用户数据进行聚类，获得四个簇，那每个簇对应一种用户细分。这种做法想当然了。

第七章：矩阵分解与话题模型

第一节:线性代数基础知识

本节简单回顾一下必要的线性代数背景知识。另 C 为一个 $M \times M$ 的词项文档矩阵，其中的元素都是非负实数。矩阵的秩 (rank) 是线性无关的行 (或列) 的数目，因此有 $\text{rank}(C) \leq \min\{M, N\}$ 。一个非对角线上元素均为零的 $r \times r$ 方阵被称为对角阵 (diagonal matrix)，它的秩等于其对角线上非零元素的个数。如果对角线上的 r 个元素都是 1，则称为 r 维单位矩阵 (identity matrix)，记为 I_r 。

对于 $M \times M$ 的方阵 C 及非零向量 \vec{x} ，有

$$C\vec{x} = \lambda\vec{x} \quad (7-1)$$

满足公式 7-1 的 λ 被称为均值 C 的特征值 (eigenvalues)。对于特征值 λ ，满足上面公式的 M 维非零向量 \vec{x} 称其为右特征向量。对应最大特征值的特征向量被称为主特征向量 (principal eigenvalues)。同样矩阵 C 的左特征向量是满足下列等式的 M 维向量 y

$$\vec{y}^T C = \lambda \vec{y}^T \quad (7-2)$$

C 的非零特征值的个数最多是 $\text{rank}(C)$ 。

例子：

考虑矩阵

$$S = \begin{pmatrix} 30 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

很明显，矩阵的秩是 3，并且有 3 个非零的特征值 $\lambda_1 = 30$, $\lambda_2 = 20$ 及 $\lambda_3 = 1$ ，它们对应的特征向量是

$$\vec{x}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \vec{x}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \text{ and } \vec{x}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

对每个特征向量而言，它与矩阵 S 相乘相当于用单位矩阵的某个倍数去乘以该特征向量，对于不同的特征向量，具体的倍数也有所不同。对于任意一个向量，例如

$$\vec{v} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

我们总是可以将 \vec{v} 表示成 S 的三个特征向量的线性组合，对于本例有

$$\vec{v} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix} = 2\vec{x}_1 + 4\vec{x}_2 + 6\vec{x}_3$$

假定用 S 乘以 \vec{v} ，则有

$$\begin{aligned} S\vec{v} &= S(2\vec{x}_1 + 4\vec{x}_2 + 6\vec{x}_3) \\ &= 2S\vec{x}_1 + 4S\vec{x}_2 + 6S\vec{x}_3 \\ &= 2\lambda_1\vec{x}_1 + 4\lambda_2\vec{x}_2 + 6\lambda_3\vec{x}_3 \\ &= 60\vec{x}_1 + 80\vec{x}_2 + 6\vec{x}_3. \end{aligned}$$

这个例子表明，即使 \vec{v} 是一个任意的向量，用 S 去乘以它的效果都取决于 S 的特征值及特征向量。另外，该式子有一个非常直观的结论，就是，相对而言， $S\vec{v}$ 的大小更不受 S 的小特征值影响。例子中， $\lambda_3 = 1$ ，所以公式中最右边的加数影响较小。实际上如果完全忽略公式最右边对应于的 $\lambda_3 = 1$ 特征向量，那么 $S\vec{v}$ 的结果就是 $(60, 80, 0)^\top$ ，而不是正确结果 $(60, 80, 6)^\top$ 。无论采用哪一种指标，来计算两个向量都相差比较近。这也意味着，**对于矩阵-向量的乘积来说，较小的特征值及其特征向量的影响也较小**。我们将带着这种直观来研究矩阵分解和低秩逼近的问题。

矩阵分解

将一个方阵分解成多个矩阵因子乘积，并且这几个矩阵因子都可以从方阵的特征向量导出。这个过程称为矩阵分解 (Matrix Decomposition)。

下面我们先给出将方阵分解成特殊矩阵乘积的两个定理。定理 1 是矩阵对角化定理，它给出了实方阵的基本因子分解法。定理 2 是对称对角化定理，它给出了实对称方阵的分解方法。这是后面我们将介绍的“奇异值分解定量”的基础。

定量 1 (矩阵对角化定理) 令 S 为 $M \times M$ 的实方阵，并且它有 M 个线性无关的特征向量，那么存在一个特征分解：

$$S = U\Lambda U^{-1} \tag{7-3}$$

其中，U 的每一列都是 S 的特征向量， Λ 是按照特征值从大到小排列的对角阵，即

$$\begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \dots & \\ & & & \lambda_M \end{pmatrix}, \lambda_i \geq \lambda_{i+1}$$

如果特征值都不相同，那么该分解是唯一的。

定理 2 (对称对角化定理) 假定 S 是一个 $M \times M$ 的实对称方阵，并且它有 M 个线性无关的特征向量，那么存在如下一个对角化分解

$$S = Q\Lambda Q^T \quad (7-4)$$

其中， Q 的每一列都是 S 的互相正交且归一化（单位长度）的特征向量， Λ 是对角矩阵，其每个对角线上的值都对应 S 的一个特征值。另外，由于 Q 是实矩阵，所以有 $Q^{-1} = Q^T$

我们后面讲基于该定理来建立词项-文档矩阵的低秩逼近矩阵。

第二节：SVD

前面我们介绍的分解都是基于方阵，然而我们感兴趣的是 $M \times N$ 的词项-文档矩阵 C 。 C 基本上不可能是对称矩阵。为此，我们先给出对称对角化分解的一个被称为 SVD (Singular Value Decomposition) 的扩展形式。

定理 3 令 r 是 $M \times N$ 矩阵 C 的秩，那么 C 存在如下形式的 SVD

$$C = U\Sigma V^T \quad (7-5)$$

其中：(1) CC^T 的特征值 $\lambda_1, \dots, \lambda_r$ 等于 C^TC 的特征值。(2) 对于 $1 \leq i \leq r$ ，令 $\sigma_i = \sqrt{\lambda_i}$ ，并且 $\lambda_i \geq \lambda_{i+1}$ 。 $M \times N$ 的矩阵 Σ 满足 $\Sigma_{ii} = \sigma_i$ ，其中 $1 \leq i \leq r$ ，而 Σ 中其他元素均为 0。其中， σ_i 就是矩阵 C 的奇异值。

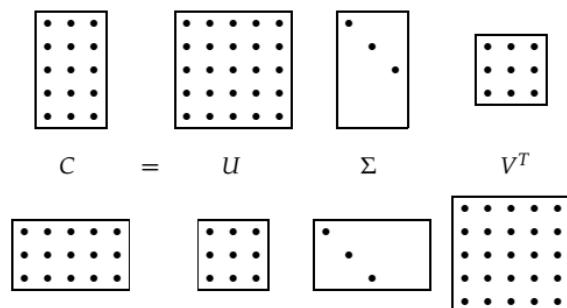


图 7-1 SVD 示意图

理解定理 2 和定理 3 之间的关系是非常有益的。将公式 7-5 与其转置相乘，有

$$CC^T = U\Sigma V^T V\Sigma U^T = U\Sigma\Sigma^T U^T \quad (7-6)$$

在公式 7-6 中，左边是一个实对称方阵，而右边正好是定量 2 给出的对称对角化分解形式。那么左边的 CC^T 代表什么呢？实际上它是一个方阵，其每行和每列都对应 M 个词项中的一个。矩阵中第 i 行，第 j 列的元素实际上是第 i 个词项与第 j 个词项基于文档共现次数的一个重合度计算指标。其精确含义依赖于构建 C 所使用的词项权重方法。

假定 C 是图 4-1 的词项文档矩阵（元素只有 0 或 1 值），那么 CC^T 的第 i 行，第 j 列的元素是词项 i 和词项 j 共现的文档数目。

当记录 SVD 分解的数值结果时，常规做法是将 Σ 描述成一个 $r \times r$ 的矩阵对角线上是奇异值。因为 Σ 的其他部分都是零（参见图 7-1）。同样，对应于 Σ 中被去掉的行，U 中最右 $M-r$ 列也被去掉。对应于 Σ 中被去掉的列，V 中最有 $N-r$ 列也被去掉。这种 SVD 的书写形式有时被称为简化的 SVD (reduced SVD) 或截断 SVD (truncated SVD)。

例子

给出一个矩阵 C

$$C = \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 1 \end{pmatrix}$$

我们使用 matlab 进行 SVD 分解 $[U,S,V] = \text{svd}(C)$ ，得到

$U =$	$S =$
$\begin{array}{cccccc} -0.6325 & 0.0000 & -0.4885 & 0.6012 & 2.2361 & 0 \\ 0.3162 & -0.7071 & -0.6109 & -0.1637 & 0 & 1.0000 \\ -0.3162 & -0.7071 & 0.6109 & 0.1637 & 0 & 0 \\ 0.6325 & -0.0000 & 0.1224 & 0.7649 & 0 & 0 \end{array}$	
$V =$	
$\begin{array}{cc} -0.7071 & -0.7071 \\ 0.7071 & -0.7071 \end{array}$	

此处的 S 是 Σ 。而当实施截断 SVD，

$S =$

$$\begin{matrix} 2.2361 & 0 \\ 0 & 1.0000 \\ 0 & 0 \\ 0 & 0 \end{matrix}$$

$U =$

$$\begin{matrix} -0.6325 & 0.0000 \\ 0.3162 & -0.7071 \\ -0.3162 & -0.7071 \\ 0.6325 & -0.0000 \end{matrix} \begin{matrix} -0.4885 & 0.6012 \\ -0.6109 & -0.1637 \\ 0.6109 & 0.1637 \\ 0.1224 & 0.7649 \end{matrix}$$

$V =$

$$\begin{matrix} -0.7071 & -0.7071 \\ 0.7071 & -0.7071 \end{matrix}$$

即

$$C = \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} -0.632 & 0.000 \\ 0.316 & -0.707 \\ -0.316 & -0.707 \\ 0.632 & 0.000 \end{pmatrix} \begin{pmatrix} 2.236 & 0.000 \\ 0.000 & 1.000 \end{pmatrix} \begin{pmatrix} -0.707 & 0.707 \\ -0.707 & -0.707 \end{pmatrix}$$

截断后的 U, S, V 三个矩阵的乘积仍然是 C 。

第三节：隐语义索引

隐语义索引 (Latent Semantic Index, LSI) , 也叫 Latent Semantic Analysis, LSA。它是基于词项文档矩阵的奇异值分解。即将原始的词项-文档空间转换到一个 LSI 空间。

按照 LSI 最初的提出者 Deerwester 的说法 , LSI 主要用于解决信息检索中一义多词的问题。一义多词 , 表现在两个词在文档中的共现性。

向量空间模型中 , 文档表示成向量。这种表示的优点包括 : 能够对不同的词项赋予不同的权重 , 除了文档检索之外还可以推广到聚类、分类等其他领域。但是向量空间模型没有能力处理自然语言中的两个经典问题 , 一义多词 (synonymy) 和一词多义 (polysemy) 。一义多词指的是不同的词 (比如 , car 和 automobile) 具有相同的含义。向量空间表示不能捕捉诸如 car 和 automobile 这类同义词之间的关系。将它们表示成了独立的两个维度。因此在信息检索中 , 计算查询 q(如 car) 和文档 d(只包含了 automobile) 的相似度就为零。而一词多义是某个词项 (如 change) 具有多个含义。因

此计算 $q.d$ 时就会高估了用户期望的相似度（用户查询中的 change 和文档中的 change 不是同一个意思）。一个自然的问题就是能否利用词项的共现情况（例如，charge 是和 legal 还是 electron 在某篇文档中共现），来获得词项的隐语义关联，从而减轻这些问题的影响。

在 LSI 中，使用 SVD 来构造矩阵 C 的一个低秩逼近 C_k 。下面我讲一下低秩逼近的概念。

低秩逼近是说：给定 $M \times N$ 矩阵 C 及正整数 k ，寻找一个秩不高于 k 的 $M \times N$ 矩阵 C_k ，使得两个矩阵的差 $X = C - C_k$ 的 F 范数最小

$$\|X\|_F = \sqrt{\sum_{i=1}^M \sum_{j=1}^N X_{ij}^2}$$

SVD 可以解决低秩逼近的问题。下面是使用截断 SVD 进行 k 秩逼近的操作方法。K 的取值往往在几百以内：

(1) 给定 C ，构造 SVD 分解，因此 $C = U \Sigma V^T$

(2) 把 Σ 中对角线上 $r-k$ (k 由用户自己定) 个最小奇异值对应的行和列截断，从而得到 Σ_k

(3) 令 U'_k 和 V'_k 分别表示保留 U 和 V 前 k 列后得到的矩阵， U'_k 是一个 $M \times k$ 矩阵， V'^T_k 是 $k \times N$ 矩阵，因此有 $C_k = U'_k \Sigma_k V'^T_k$

注意：在我们这里进行 k 秩逼近目的不是得到 C_k 。而是得到 U_k 和 V_k 。 因为这是将词项-文档矩阵 C 中的每行和每列（分别对应每个词项和文档）映射到一个 k 维空间。 U_k 是把原有词项空间往 k 维空间压缩得到的新的 k 维词项空间（矩阵）。 V_k^T 是把原来的文档空间往 k 维空间压缩得到的新的 k 维文档空间（矩阵）。

使用 LSI 进行信息检索

LSI 操作的通常步骤如下：

(1) 收集领域相关的文本集合，分割成一系列文档。大部分的应用中，每篇文档按照段落被划分成更小的文档。因为通常一个段落的内容是更相关的。

(2) 创建词项-文档矩阵。矩阵元素是词项在文档中的词频。也可以考虑其他权重计算方法。

(3) 调用 SVD 对词项文档矩阵进行分解。得到三个矩阵 U 是 $m \times r$ 的词项矩阵， V^T 是 $N \times k$ 的文档矩阵。 Σ 是 $r \times r$ 的对角阵。

(4) 给出一个参数 $k < r$ ，对 SVD 分解结果进行截断。得到 $m \times k$ 的矩阵 U_k 、 $k \times k$ 的对角阵 Σ_k 、 $N \times k$ 的文档矩阵 V_k^T 。

此时的文档矩阵 V_k^T 就是一个压缩了，或转换了的文档空间。

U_k 和 V_k^T 就是 LSI 空间。在 LSI 空间上进行信息检索时（已经把原始的词项-文档矩阵转换成了 LSI 空间），需要把原始的查询 q ，映射到 LSI 空间上进行检索。因为

$$U_k \Sigma_k \vec{q}_k \approx \vec{q}$$

这里 $U_k \Sigma_k \vec{q}_k$ 是 k 秩逼近的空间。由此，可以得到

$$\vec{q}_k \approx \Sigma_k^{-1} U_k^T \vec{q}$$

\vec{q}_k 就是映射到 LSI 空间上的查询向量。进行信息检索时就使用 \vec{q}_k 和文档空间 V_k 上的文档向量进行相似度计算来检索。

LSI 特性的实验分析

下面通过实验来总结 LSI 的特性，并分析原因。以图 7-2 为例。一义多词体现在了词项的共现性上。*Voyage* 和 *trip* 是一义多词。如果查询为 *trip*，那么包含 *voyage* 词项的文档，将获得高的排序位置。

	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
voyage	1	0	0	1	1	0
trip	0	0	0	1	0	1

图 7-2：词项文档矩阵

设查询向量为 $q = [0 \ 0 \ 0 \ 0 \ 1]'$

我们用下面的 matlab 程序来做实验

```
clc;clear;

C=[1 0 1 0 0 0;0 1 0 0 0 0;1 1 0 0 0 0;1 0 0 1 1 0;0 0 0 1 0 1];

q=[0 0 0 0 1]';

k=2;          % svd 的参数 k
```

```

[row,col]=size(C);
[U,S,V]=svd(C);
Uk=U(:,1:k); % 此方法称为截断 SVD
Vk=V(:,1:k)';
d=diag(S);
Sk=diag(d(1:k));

%将原始查询向量映射到 LSI 空间上
qk=inv(Sk)*Uk'*q;

% LSI 空间下，计算查询向量和文档空间的距离
qs=qk*ones(1,col);
t=(Vk-qs).^2;
t=sum(t,1);
t=sqrt(t);
disp(t);

%原始空间下
qk=q;
qs=qk*ones(1,col);
t=(C-qs).^2;
t=sum(t,1);
t=sqrt(t);
disp(t);

```

当 SVD 分解的 k 参数为 5 时的程序运行结果是(按照向量距离计算的结果)

1.2910	1.2910	1.2910	0.8165	1.4142	0.0000
2.0000	1.7321	1.4142	1.0000	1.4142	0

第一行是在 LSI 空间上进行的计算，第二行是在原始空间上的计算。此时的 C 的秩是 5，可以看出，进行与否 LSI 处理，对排序结果影响不大。d5 文档都排在了后面。而 voyage 和 trip 是一义多词。d5 本应排在前面。

设置 SVD 分解的 k 参数为 2，运行程序的结果如下。

0.9340	0.9474	0.5971	0.3925	0.2756	0.0000
2.0000	1.7321	1.4142	1.0000	1.4142	0

此时可以看出，进行 LSI 处理，D5 文档都排在了前面。我们可以得出结论：如果减低 k 值，LSI 可以提高检索的查全率。

LSI 的定性分析

LSI 处理后获得的文档空间是压缩了的文档空间。原来的原始文档空间，每个词项是一维。压缩后的空间把一义多词的词项合并到一个维。所以，在 LSI 空间上进行检索时。相当于在原始空间上，会在与查询的词项有相同含义的其他词项也算作是匹配词项。因此提高了查全率。

我们还可以看到 LSI 的几个特点：

- (1) 可以将原始向量空间（成千上万维）压缩到很小（几百维之内）
- (2) SVD 的计算开销很大。
- (3) 按照 Deerwester 的总结：LSI 很好的处理了一义多词的问题，对一词多义有部分解决。
- (4) LSI 仍然延续了 VSM 模型的两个缺点：无法表示否定，无法完成布尔查询。

使用 LSI 做话题检测

截断 SVD 时，U 空间描述的是压缩到 k 维的词项分布空间，具有相同共现性的词汇出现在同一列。一个列也意味着在原始文档空间谈论的话题。

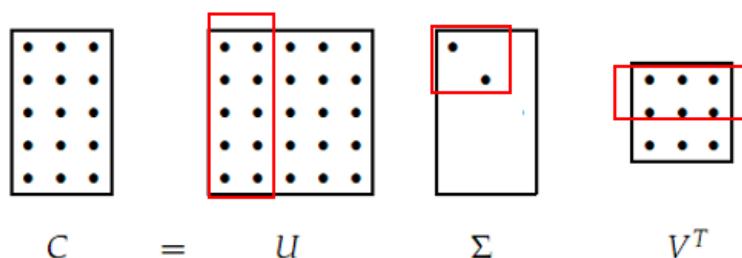


图 7-3：截断 SVD

上面的例子中，选择 k=2 时的截断 SVD 得到的 U_k 矩阵是

$U_k =$

0.4403	-0.2962
0.1293	-0.3315
0.4755	-0.5111
0.7030	0.3506
0.2627	0.6467

每一列是一个话题，我们选择 top n 个词项就可以获得该话题的描述。

用 LSI 做文档聚类

V_k 空间是压缩到 k 维的文档空间。把 k 维中的一维看做是一个簇，则 V_k 空间描述了文档聚类的结果。

上面的例子中，选择 k=2 时的截断 SVD 得到的 V_k 矩阵是

$V_k =$

0.7486	0.2797	0.2036	0.4466	0.3251	0.1215
-0.2865	-0.5285	-0.1858	0.6255	0.2199	0.4056

此时，每一行是一个簇。选择每列中最大值所在的行作为每篇文档的簇标签。例如， V_k 中的第一列是文档 d1 在两个簇上的评分计算。可以看出，d1 属于簇 1。文档 d5 在两个簇上评分相近。我们可以把 d5 分配到两个簇中，这时就是软聚类（一篇文档可以属于多个簇）。

话题模型的发展

话题模型 Topic Model 是文本挖掘中的一个研究热点。它主要用于从文档集合中发现话题。

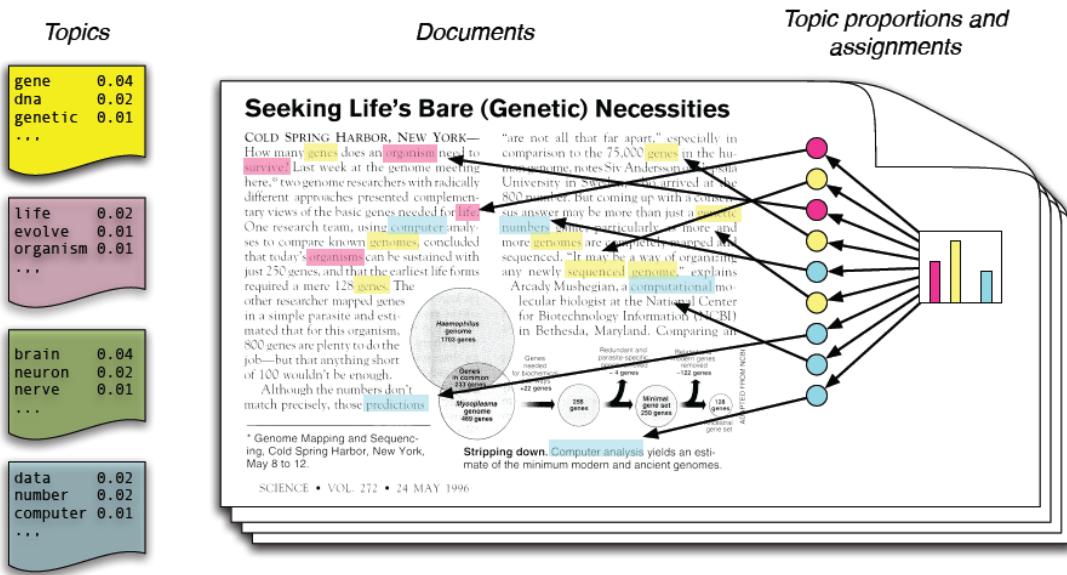


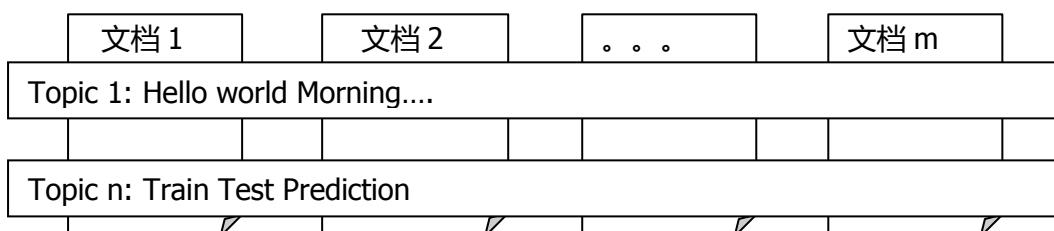
图 7-4 : 话题检测示例

话题 (Topic) 的定义是一个 words 的概率分布。简单的理解就是：话题就是一个词项的集合，这些词项以高的概率事件去描述文本中所谈论的某一方面的内容。

computer	chemistry	cortex	orbit	infection
methods	synthesis	stimulus	dust	immune
number	oxidation	fig	jupiter	aids
two	reaction	vision	line	infected
principle	product	neuron	system	viral
design	organic	recordings	solar	cells
access	conditions	visual	gas	vaccine
processing	cluster	stimuli	atmospheric	antibodies
advantage	molecule	recorded	mars	hiv
important	studies	motor	field	parasite

图 7-2 : 50-topic LDA 模型从 Science 的 1980~2002 期刊检测出的其中 5 个话题

Topic Model 通常中假设一篇文档包含多个话题。有很多 Topic Model，如 PLSA，LDA 等从文档集合中检测话题。



话题检测模型从 LSA 发展到了 PLSA，再到 LDA。下面将对 PLSA 和 LDA 详细介绍。

第四节：PLSA

LSI 的主要目的是建立压缩的文档空间（LSI 空间），为信息检索时，解决一义多词的问题。它可以提高信息检索的查全率。LSI 使用的是矩阵分解技术。Hofmann 提出了一个 LSI 技术的一个初始概率扩展方法 PLSA。它是一个概率模型。LSI 并没有解决一词多义的问题。而 PLSA 可以解决。有很多研究应用 PLSA 从文档集合中发现话题。

按照 PLSA 发明者 Thomas Hofmann 在他的论文中的描述，PLSA 可以达到和 LSI 同样的目标，PLSA 定义了一个 Generative Model，它有几个优点：

- (1) 这样的 Generative 概率模型，更适合对模型的拟合，模型的选择，模型复杂性的控制。
- (2) 概率模型可以引入领域知识到模型中。
- (3) PLSA 中因为是概率模型还可以引入隐变量，以产生更多复杂的隐语言结构。

PLSA 通过将一个未观察到的变量 z_k 和每个观察建立联系（一个观察即一个 word 出现在一篇文档中）。有几个参数：

$p(d_i)$ 选择一篇文档 d_i 的先验概率

$p(z_k|d_i)$ 选定文档 d_i 后，选择隐变量 z_k 的概率

$p(w_j|z_k)$ 选定隐变量 z_k 后，选择一个 word w_j 的概率

一个观察 $\langle d_i, w_j \rangle$ 的概率分布（**建立的概率模型是为观察建立的概率模型**）

$$p(d_i, w_j) = p(d_i)p(w_j|d_i)$$

其中对于 $p(w_j|d_i)$ 引入 K 个隐变量 z 后，可以得到

$$\sum_{k=1}^K p(w_j|z_k)p(z_k|d_i)$$

即 d_i 和 w_j 的联合概率分布是

$$p(d_i, w_j) = p(d_i) \sum_{k=1}^K p(w_j|z_k)p(z_k|d_i)$$

该模型是等价于

$$P(d, w) = \sum_{z \in \mathcal{Z}} P(z)P(d|z)P(w|z)$$

而它又等价于截断 SVD。见图 7-3。

$P(z)$ 对应 Σ_k , $P(d|z)$ 对应 V_k , $P(w|z)$ 对应 U_k 。这就是为什么说 PLSA 是 LSI 的概率扩展。

$p(w_j|z_k)$ 和 $p(z_k|d_i)$ 是模型的参数。在话题发现的问题上 z_k 可以理解为是话题。该模型对应于“一个文档包含多个话题，一个话题是 word 的概率分布”。

估计该概率模型，需要获得该模型的似然函数。（**写似然函数时，是根据概率模型来写的**）

$$\begin{aligned}\mathcal{L} &= \sum_{i=1}^N \sum_{j=1}^M n(d_i, w_j) \log P(d_i, w_j) \\ &= \sum_{i=1}^N n(d_i) \left[\log P(d_i) + \sum_{j=1}^M \frac{n(d_i, w_j)}{n(d_i)} \log \sum_{k=1}^K P(w_j | z_k) P(z_k | d_i) \right]\end{aligned}$$

该似然函数即把所有的观察 (d_i, w_j 对) 带入概率模型 $p(d_i, w_j)$ 。注：上面的公式给出的是 log 似然。然后使用 EM 算法可以估计参数。具体的参数估计，请参考论文 Thomas Hofmann, Probabilistic Latent Semantic Indexing, SIGIR, 1999。

在 E 步骤

$$P(z|d, w) = \frac{P(z)P(d|z)P(w|z)}{\sum_{z' \in \mathcal{Z}} P(z')P(d|z')P(w|z')}$$

在 M 步骤

$$\begin{aligned}P(w|z) &\propto \sum_{d \in \mathcal{D}} n(d, w) P(z|d, w), \\ P(d|z) &\propto \sum_{w \in \mathcal{W}} n(d, w) P(z|d, w), \\ P(z) &\propto \sum_{d \in \mathcal{D}} \sum_{w \in \mathcal{W}} n(d, w) P(z|d, w)\end{aligned}$$

第五节：LDA

LDA (Latent Dirichlet Allocation) 是一种文档主题生成模型，也称为一个三层贝叶斯概率模型，包含词、主题和文档三层结构。所谓生成模型，就是说，我们认为一篇文章的每个词都是通过“以一定概率选择了某个主题，并从这个主题中以一定概率选择

某个词语”这样一个过程得到。文档到主题服从多项式分布，主题到词服从多项式分布。 LDA 模型是对 PLSA 模型的发展。

LDA 是一种非监督机器学习技术，可以用来识别大规模文档集 (document collection) 或语料库 (corpus) 中潜藏的主题信息。它采用了词袋 (bag of words) 的方法，这种方法将每一篇文档视为一个词频向量，从而将文本信息转化为了易于建模的数字信息。但是词袋方法没有考虑词与词之间的顺序，这简化了问题的复杂性，同时也为模型的改进提供了契机。每一篇文档代表了一些主题所构成的一个概率分布，而每一个主题又代表了很多单词所构成的一个概率分布。 ·

代码：使用 mallet 实现的 LDA

```
public class LDATest {  
    String file="www2016.txt";  
    int numTopics=8;  
  
    public static void main(String[] args) {  
        LDATest lda=new LDATest();  
        try {  
            lda.run();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
    private void run() throws Exception{  
        Pipe instancePipe = new SerialPipes (new Pipe[] {  
            new Target2Label (),  
            new Input2CharSequence (),  
            new CharSequence2TokenSequence (),  
            new TokenSequenceLowercase (),  
            new TokenSequenceRemoveStopwords (),  
            new TokenSequence2FeatureSequence(),  
        }  
    }  
}
```

```
        new FeatureSequence2FeatureVector(),
    });

    InstanceList trainList = new InstanceList (instancePipe);

    try {

        trainList.addThruPipe(new
CsvIterator(file,"(\\w+)\\s+([\\w-]+)\\s+(.*", 3, 2, 1));

    } catch (FileNotFoundException e) {

        e.printStackTrace();

    }

    SimpleLDA lda = new SimpleLDA (numTopics, 50.0, 0.01);

    lda.addInstances(trainList);

    lda.sample(1000);

}

}
```

第八章：文本情感分析

第一节：文本情感分类介绍

文本情感分析是这样的一个研究领域，它从文本中分析人们的情感、观点、评价或态度。它是自然语言处理和数据挖掘中广泛研究的一个领域。事实上由于它的重要性，情感分析已经不单单是计算机领域的研究问题，它已经渗透到了管理科学和社会科学领域。如对商品评论的分析，可以帮助企业找出人们对商品一些特点的情感倾向，从而帮助企业改进产品质量。研究者从 twitter 和 facebook 中分析人们的政治观点来预测美国大选。

情感分析 (sentiment analysis) 和一些其他的名词术语相近。如，opinion extraction, sentiment mining, subjectivity analysis, affect analysis, emotion analysis, review mining。

我们可以把 Sentiment analysis 看做是一个大的框架，该框架包含了观点挖掘 (opinion mining) 、 opinion extraction 等。

1. 情感分析的应用

随着 Web 2.0 的爆发增长（例如，论坛、微博、网站中的评论、社交媒体），个人与组织日益使用这些用户产生内容做决策。现在，一个人想买一件商品，可以不再去问他的朋友，而是在 web 上去寻找相关产品的评论信息。一个企业进行市场调查，也不再局限于向公众发放调查表，而是可以从 web 上获取公众意见，从而获得市场信息。

另外在学术界有很多情感分析的应用研究，例如：

- 使用情感模型预测销售业绩
- 使用评论数据对商品、服务等进行排序。
- Twitter 中的情感分析用来做民意调查。
- 使用 Twitter 数据，评论数据和博客做电影票房的预测。
- 邮件中的情感表达被用于发现情感方面的性别差异。

- Twitter 中的公共情绪用于预测股票市场。
- 博客和新闻中的情感表达用于研究交易策略

2. 情感分析的研究

情感分析的不同分析层次。情感分析主要在三个层次上进行分析：

(1) 文档层次

这个任务中，对文档中表现出的整个的情感倾向做分类，分类为正向或负向的情感倾向。例如，给出一条商品评论，系统确定是否评论对该商品表达了正向的或负向的观点。这一任务通常可以看做是文档级的情感分类。这一层次上的分析假定每篇文档对一个单独的实体 (entity)，例如单个的商品，表达观点。因此，对于一篇文档中包含了对多个实体进行评估和比较的文档，该方法就不实用了。（适用于评论数据）

(2) 句子层次

这一任务针对句子，确定是否每个句子表达了一个正向的、负向的或者中立的观点。中立通常意味着没有观点。这一级别的分析和主观分类 (subjectivity classification) 相关。主观分类将句子区分为主观句子和客观矩阵。主观句子中表达主观的观点；客观句子中表达实际信息或客观事实。然而，需要说明，主观不等同于情感倾向，而有时客观句子中却可能包含观点。例如，“We bought the car last month and the windshield wiper has fallen off”。研究者已经在子句的级别上进行研究，但在子句级别的分析仍然不够。例如，“Apple is doing very well in this lousy economy.”

(3) 实体与aspect层次

文档级和句子级的分析没有发现某个人确切的喜欢什么和不喜欢什么。Aspect 级别的分析完成更精细粒度的分析。Aspect 级更早也称作特征 (feature) 级别的分析（基于特征的观点挖掘和摘要）。Aspect 分析不直接考察语言结构（文本、段落、句子、子句或词组）。它直接考察观点。它基于这样的思想：一个观点包含一个情感倾向（正向或负向）和一个观点表达的目标（target）。一个没有目标的观点它的应用范围有限。

情感分析中的一些术语和定义：

(1) 定义 1 (entity , 实体)：一个 entity 是一个产品、服务、话题、问题、人、组织或者事件。它用一个“对” (T, W) 来描述。这里 T 是“部件，子部件” (part, sub-

part) 的层次结构。W 是一个属性 (attribute) 集合。每个部件或子部件都有自己的属性集合。

例子：一个特别型号的照相机是一个实体，例如，Canon G12。它有一个属性集合，例如，图片质量、大小、重量。也有一些部件，例如，镜头、取景窗、电池等。而电池又有一个属性集合，例如电池寿命，电池重量。一个话题也可以是一个实体，例如，“增税”。而“穷人增税”，“中产增税”是它的部件。

(2) **定义 2 (opinion , 观点)** : 一个观点是一个五元组($e_i, a_{ij}, s_{ijkl}, h_k, t_l$)。 e_i 是一个实体的名称； a_{ij} 是实体的一个 aspect； s_{ijkl} 是一个实体 e_i 的 aspect a_{ij} 的情感倾向； h_k 是观点持有者； t_l 是观点表达的时间。情感倾向 s_{ijkl} 可以是正向或负向或中立的。或者可以用一个评分表达强度，例如，电商评论数据中常用 1-5 星的评分。当一个观点是表达在整个实体上（未针对属性或 aspect），可以使用一个特殊的 aspect，称作 GENERAL 来替代。这里， e_i, a_{ij} 一起描述观点的目标。

基于 Aspects 的情感分析的目标是给定一篇文档 d ，发现 d 中所有的观点五元组($e_i, a_{ij}, s_{ijkl}, h_k, t_l$)。

(3) **定义 3 (aspect, 方面)** : Aspect 指的是一个 entity 的各种属性。如果我们谈 General Aspect, 它包含 entity 和 aspects。

第二节：文档情感分类

情感分类可以说研究的非常广泛。它的目标是将一篇文档分类为表达了正向情感的文档或是负向情感的文档。此任务就是文档级的情感分类。已经有大量的研究对在线评论进行情感分类。此处，我们将问题定义按照“评论数据”进行定义。（因为评论数据中只有一个目标）。我们下面将讨论三种文档情感分类：(1) 有监督的文档情感分类；(2) 无监督的情感分类；(3) 基于词典的文档情感分类。

1. 有监督的文档情感分类

情感分类可以看做是一个二类分类问题，即正向或负向情感的情感类别。情感分类因此基本上是一个文本分类的问题。传统的文本分类方法主要是将文档按照不同的 topic 进行分类。例如，政治类、科学类或体育类。在这样的分类任务中，与 topic 相关的词就是关键特征。然而，在情感分类中，指示了观点倾向的情感词或观点词更重要一些。例如，great, excellent, amazing, horrible, bad, worst 等等。

因为文本情感分类本质上是一个文本分类问题，因此任何有监督的学习方法都可以用来实施文本情感分类。例如，朴素贝叶斯，SVM 等等。在一篇研究论文中 Bo Pang 用

这些方法将电影评论分类为两类，正向或负向的情感类。该研究显示使用一元语言模型（词袋模型）的 NB、SVM 很好的完成了工作。

就像其他的有监督学习模型一样，情感分类的关键是选择一系列有效特征的“特征工程”。一些特征的选择如下：

(1) 词项和词项频率。这些特征是单个的词 (unigram)，并统计了它们的词频。这是在传统的基于 topic 的文档分类中的最基本的特征。在一些情况下，词的位置也可以被考虑。信息检索模型中的 TF-IDF 权重也可以应用。就像在传统的文本分类中一样，这些特征已经被证明在情感分类中也非常有效。

(2) 词性 (part of speech, POS)。每个词的词性 (名词、动词等) 也非常重要。不同词性的词可以区别对待。例如，已经证明形容词是重要的“观点指示器”。因此，一些研究人员将形容词看做是特别的特征。然而，一个人也能使用所有的“词性”标签和对应的他们的词项作为特征。英文中，很通用的是词项表是 treebank。

https://ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

TABLE 3.1: Penn Treebank Part-Of-Speech (POS) tags			
TAG	DESCRIPTION	TAG	DESCRIPTION
CC	Coordinating conjunction	PRP\$	Possessive pronoun
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Adverb, comparative
EX	Existential <i>there</i>	RBS	Adverb, superlative
FW	Foreign word	RP	Particle
IN	Preposition or subordinating conjunction	SYM	Symbol
JJ	Adjective	TO	<i>to</i>
JJR	Adjective, comparative	UH	Interjection
JJS	Adjective, superlative	VB	Verb, base form
LS	List item marker	VBD	Verb, past tense
MD	Modal	VBG	Verb, gerund or present participle
NN	Noun, singular or mass	VBN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non-3rd person singular present
NNP	Proper noun, singular	VBZ	Verb, 3rd person singular present
NNPS	Proper noun, plural	WDT	Wh-determiner
PDT	Predeterminer	WP	Wh-pronoun
POS	Possessive ending	WP\$	Possessive wh-pronoun

(3) 情感词和词组。情感词是在某语言中能够表达正向或负向情感的词。例如，good, wonderful, 和 amazing 是正向情感词；bad, poor, and terrible 是负向的情感词。大部分的情感词是形容词或副词。但是，有时名词(例如, rubbish, junk, 和 crap) 和动词 (例如, hate 和 love) 也能用于表达情感。除了单个的词，也有情感词组或习语 (idiom)。例如，“cost someone an arm and a leg”。

除了上面的标准机器学习方法应用到文档的情感分类，也有很多研究发展了特定的面向情感分类的方法。

例 1：评论情感分类 (Mining the peanut gallery: Opinion extraction and semantic classification of product reviews , WWW2013)。该文提供了许多文本挖掘时的基本文本处理技巧。

该文的工作过程是这样的：

- (1) 准备好正例和负例的文档集合。
- (2) 挑选特征
- (3) 为每个特征计算一个评分 s in [-1,1]
- (4) 情感分类时，对一个 review，获取特征，累积，特征的评分

Step 1. 特征选择

- (1) 替换操作

将数字用 “NUMBER” 替代。数字在特定的环境下有特定的情感倾向，此方法可以降低错误分类率。

- (2) 引入 N-Grams。即该文不但考虑每个词项，还考虑了 2-gram (连续的两个词项组成的词组) 和 3-gram (连续三个词项组成的词组)。

<i>Unigrams</i>	<i>Bigrams</i>	<i>Trigrams</i>	<i>Distance 3</i>
Top positive features			
great camera best easy support excellent	easy to the best . great great camera to use i love	easy to use i love it . great camera is the best . i love first digital camera	. great easy to camera great best the . not easy use
back love not digital	love it a great this camera digital camera	for the price to use and is a great my first digital	.camera i love to use camera this
Top negative features			
waste tech	returned it after NUMBER	taking it back time and money	return to customer service
sucks horrible	to return customer service	it doesn't work send me a	poor quality . returned
terrible return worst customer returned poor	. poor the worst back to tech support not worth it back	what a joke back to my . returned it . why not something else . the worst	the worst i returned support tech not worth . poor back it

该文的实验证明，n-grams 的引入对情感分类效果很好。

(3) 特征挑选：仅仅挑选至少出现在了 3 篇文档中的特征。因此特征数量大大减少了。

Step 2. 为每个特征 f_i 计算评分 (这里 C 是正例 , C' 是负例)

$$score(f_i) = \frac{p(f_i|C) - p(f_i|C')}{p(f_i|C) + p(f_i|C')}$$

$p(f_i|C)$ 是规范化的词项频率，即一个特征 f_i 出现在类别 C 的次数。 $p(f_i|C)$ 减去 $p(f_i|C')$ ，然后除以 C 中的词项总数。一个特征的评分 $score(f_i)$ 的变化范围是 [-1, 1]。该文也探讨了使用信息增益来为每个特征计算评分。

Step 3. 分类

为每个特征计算了评分后，对一篇文档中的每个抽取出的特征 f 计算评分 $score(f)$ ，然后对所有评分求和。使用总的评分 $eval(d)$ 的极性作为类别标签。

$$class(d_i) = \begin{cases} C & eval(d_i) > 0 \\ C' & eval(d_i) < 0 \end{cases}$$

Where

$$eval(d_i) = \sum_j score(f_j)$$

2. 使用无监督的方法进行情感分类

因为情感词经常情感分类的主宰因素。我们可以想象情感词和词组可以用于无监督形式的情感分类。在论文《Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews》研究了一个无监督情感分类的方法。它基于固定的语法模式（它们很可能被用于表达情感）。语法模式基于词的词性。该算法包含三个步骤。

Step 1: 抽取词组

两个连续的词被抽取出，如果他们的词性符合表 8-1 中的模式。例如：模式 2 是说两个连续的词，如果第一个是副词，第二个词是形容词，第三个词不是名词，则将两个连续的词抽取出来（第三个词不抽取）。下面给出一个例子：句子

“This piano produces beautiful sounds”

中，“beautiful sounds” 被抽取出来，因为它满足第一个模式。

使用这些模式的原因是，JJ（形容词），RB（副词），RBR（比较级副词）和RBS（最高级副词）经常用于表达观点。而名词和动词经常被用作语境（context）。因为不同的语境下 JJ, RB, RBR 和 RBS 可以表达不同的情感。例如，形容词（JJ）“unpredictable” 在一条关于 car 的评论中可以表达负面情感，像 “unpredictable steering”。但在电影评论中“unpredictable plot” 表达的是正面情感。

表 8-1：语法模式

TABLE 3.2: Patterns of POS tags for extracting two-word phrases			
	FIRST WORD	SECOND WORD	THIRD WORD (NOT EXTRACTED)
1	JJ	NN or NNS	anything
2	RB, RBR, or RBS	JJ	not NN nor NNS
3	JJ	JJ	not NN nor NNS
4	NN or NNS	JJ	not NN nor NNS
5	RB, RBR, or RBS	VB, VBD, VBN, or VBG	Anything

Step 2: 计算词组的情感倾向

使用逐点互信息 (mutual information , PMI) 估计抽取出的词组的情感倾向 (sentiment orientation , SO) 。 PMI 计算公式如下

$$PMI(term_1, term_2) = \log_2 \left(\frac{\Pr(term_1 \wedge term_2)}{\Pr(term_1)\Pr(term_2)} \right)$$

PMI 度量两个词项的统计上的依赖性。 $\Pr(term_1 \wedge term_2)$ 是实际上的 term1 和 term2 的共现概率。 分母 $\Pr(term_1)\Pr(term_2)$ 是两个词如果满足独立性它们的共现概率。一个词组的情感倾向 SO 是基于它和正向的一个参考词“excellent”和负向的一个参考词“poor”的关联来计算。

$$SO(phrase) = PMI(phrase, "excellent") - PMI(phrase, "poor").$$

这些概率 ($\Pr(term_1)$, $\Pr(term_1 \wedge term_2)$ 等) 通过使用搜索引擎并收集 hits (击中) 数量来计算。对于每条查询，一个搜索引擎通常会返回相关文档数量，这就是 hits 的数量。因此通过一起或单独搜索两个词项，上面的 SO 公式概率值就可以被估计出来了。这篇论文中使用 AltaVista 搜索引擎，因为它有个 NEAR 操作符，来约束搜索文档必须在 10 个词的范围内包含两个词。设 $hits(query)$ 是返回的击中数 (hits) 。上面的 SO 计算公式可以写成。

$$SO(phrase) = \log_2 \left(\frac{hits(phrase NEAR "excellent")hits("poor")}{hits(phrase NEAR "poor")hits("excellent")} \right)$$

Step 3: 分类

给定一条评论，该算法计算评论中所有抽取出的词组的平均 SO。根据平均 SO 的正或负的分值给评论分类。

在我的实际操作中，将上述的无监督 SO 改成了有监督 SO，即建立训练集，从一个训练集中计算 PMI (term, 正例) 和 PMI (term, 负例) 。在我的实验中，我对 YELP 评论数据集进行情感分类，讲 4、5 星的评论作为正例，1、2 星的评论作为反例，3 星评论不用，建立训练集。 SO 方法的性能如下：

	SO
BAC	93%
ACC	92.1%

$$ACC(D_p) = \frac{TP}{TP + FN}$$

$$\text{ACC}(\text{D}_n) = \frac{TN}{TN + FP}$$

$$\text{ACC}(\text{D}) = \frac{TP + TN}{TP + FN + TN + FP}$$

$$\text{BAC}(\text{D}) = \frac{\text{ACC}(\text{D}_p) + \text{ACC}(\text{D}_n)}{2}$$

其中， D_p 表示正例文档集合， D_n 表示反例文档集合。ACC 表示整个文档集合上分类的精确度；BAC 表示平衡精确度，BAC 适用于度量类别不平衡的数据集。例如，如果一个数据集中极端不平衡 90% 为正例样本，10% 为反例样本。如果把所有样本都分类为正例，就会有 90% 的 ACC 精确度，但只有 50% 的 BAC。

3. 基于词典的情感分类

那些具有正向或负向情感的词和词组是文本情感分析的可利用的基本元素。另一种无监督的方法是基于词典的方法。它使用一个情感词典（给出了词和词组以及他们在情感倾向上的评分），并将表示程度的词和否定的词结合起来，来为文档计算情感评分 (Lexicon-Based Method for Sentiment Analysis)。

词典：基于词典的方法进行情感分析，其中的词典主要有两种方法来创建：人工和自动创建。自动创建中，有的使用种子词汇集合，然后从该集合扩展词典。种子词汇是一组具有强烈正或负向倾向的词的集合。如, excellent, abysmal。原则上，一个正向情感的词汇应该频繁地出现在正向种子词附近。负向词汇则反之。有研究采用互信息的方法，为这些词汇计算情感评分。

在使用词典的方法中，很多研究使用形容词作为文本的语义方向 (Semantic Orientation，也叫 SO 和 Sentiment Orientation 是一个意思) 的指示符。形容词和他们的 SO 评分，编辑到一个词典中。对于一个给定的文本，所有的形容词被抽取，并用词典做标注。累积形容词的 SO 评分，作为最后文本的 SO 评分。

在前述“基于词典的方法”的基础上，有很多研究发现，语言情境 (linguistic context) 对情感分析有重要的影响。很多研究已经指出，在评估 SO 时，词所处的局部的语境需要被考虑。这些语境中存在 valence shifter(一些词修饰词加上后使得本身的情感发生变化)。例如，否定词 (not, hardly) ；再比如，“他对人过于好了”。

“过于”这个修饰词，将“好”本身的原始极性给偏移了。

我们介绍一个称为 SO-CAL 的情感方向计算模型。它首先抽取情感表达的词，包括形容词、动词、名词和副词，然后使用它们来做事计算 SO 的基本元素，并充分考虑 Valence Shifter(intensifiers, negation and irealis marker) 的影响。

SO-CAL 计算 sentiment orientation 有两个基本的假设：(1) 单个词具有先验极性，即独立于 context 的 semantic orientation；(2) semantic Orientation 能被表达为一个数值。

步骤：

- (1) 手工创建词典，包括了形容词、副词、名词和动词。该文的研究中每个情感词会被分配从-5 到+5 的评分。

Table 2
Examples from the adverb dictionary.

Word	SO Value
excruciatingly	-5
inexcusably	-3
foolishly	-2
satisfactorily	1
purposefully	2
hilariously	4

- (2) 获得 Intensification (增加或减弱词的情感极性)。

Table 3
Percentages for some intensifiers.

Intensifier	Modifier (%)
slightly	-50
somewhat	-30
pretty	-10
really	+15
very	+25
extraordinarily	+50
(the) most	+100

如果“good”有一个 3 分的 SO 评分，然后 “really very good” 有一个 SO 评分 $3 \times 100\% + 25\% \times (100\% + 15\%) = 4.3$

- (3) 获得 negation (翻转词的极性)

- (4) 获得 Irealis (一些标记，可以指示句子里的一些词他们的情感极性不可靠，如条件句,情态动词 should have , 负极性的词 any , anything , 疑问句 , 一些动词 expect doubt)。该文的做法是忽略在 irrealis 范围内的情感词的 semantic orientation.

(5) 文本的特征。重复出现的 word 调低它的 SO 值，第 n 次出现的，SO 为原始 SO 的 $1/n$

第三节：句子级的情感分类

在很多应用中，文档级的情感分类太粗糙。例如，一篇文档表达的观点很多，可以有正、有负面倾向的表达。在句子的级别上做情感分析更合适。然而，句子级和文档级没有本质不同，句子可以看做是短文本。在做句子级的情感分析时通常做的假定是：一个句子通常只包含一个观点。

句子情感分类可以看做是一个三类分类（正向、负向和中性）或二分类问题。

在二分类中，则包含两个步骤，第一步需要判断该句子是否表达了观点（subjectivity classification）还是仅仅描述了客户信息；第二步判断该句子观点的正或负向。

第四节：Aspects 级的情感分析

Aspect 级的情感分析又称为 观点挖掘（opinion mining）。

在文档的级别上或句子级别上对有观点的文本进行分类经常不能满足应用需求。因为它们没有确定观点的目标或给目标分配表达的情感倾向。

即使我们假定，每篇文档只对一个实体进行评估（观点表达），对于该实体表达了正向观点的一篇文档并不意味着作者对于该实体的所有 aspect 都有正向的观点。同时，一个负向观点的文档也不意味着作者对每件事都是负面观点。对于更多完整的分析，我们需要发现 aspects 并确定每一个 aspects 的情感倾向。基于 aspects 的情感分析包含了实体和 aspects 的情感分析。

在 aspects 的级别上，目标是发现给定的文档 d 所包含的每个五元组($e_i, a_{ij}, s_{ijkl}, h_k, t_l$)。要达到这个目标，主要完成下面的两个任务

(1) Aspect 抽取: 此任务抽取 aspects。例如，有一个句子 “The voice quality of this phone is amazing,” 实体是“this phone”，aspect 是 “voice quality”。需要注意的是，这里实体 “this phone” 没有指示 aspect GENERAL（将实体和它的 aspects 作为一个整体来评价）。因为这里不是对 this phone 做一个整体评价，而是关于它的 voice quality。然而，句子 “I love this phone” 是将电话作为一个整体来评价，即 实体 “this phone”的 GENERAL aspect。需要记住的是，无论什么时候我妈妈谈论

“aspects”，我们必须知道它属于哪个实体。下面的讨论中，为了表达上的简便，我们经常忽略提及“实体”。

(2) Aspect 情感分类: 此任务确定是否不同 aspects 上的观点是正向、负向或中立的。在上面的例子中，在 aspect “voice quality” 上的观点是正向的。第二个例子中，在 aspect GENERAL 上，观点也是正向的。

1. Aspects 抽取

Aspects 抽取也能看做是一个信息抽取的任务。有四个主要的“明确地 aspects 抽取”方法：

- (1) 基于频繁的名词和名词词组
- (2) 利用观点和目标的关系
- (3) 使用有监督的学习
- (4) 使用 Topic modeling

第一种方法（基于频繁的名词和名词词组）。从某个特定领域的大量的评论数据中发现“明确的”aspect 表达，即名词和名词词组。名词和名词词组可以通过词性 POS 来确定。它们的出现频率被计数，仅挑选频繁出现的名词和名词词组，这个频率阈值可以通过实验来确定。

这种方法之所以能工作，是因为当人们评论一个实体的不同 aspects 时，它们使用的词汇表通常是一个有限的集合。因此，频繁被谈论到的名词通常是真正的重要 aspects。评论中的不相关内容通常涉及的范围很广，例如，不同的评论谈论的不相关内容差异很大。因此不频繁的名词很可能不是 aspects，或不那么重要的 aspects。虽然这个方法很简单，它在实际应用中该方法却很有效。一些商业公司使用的是这种方法的改进版本。

其余的方法，我们不做过多的讨论了。

2. Aspects 情感分类

Aspects 的情感分类有两个主要的方法，即，有监督的学习和基于词典的方法。

我们在前面讨论的句子级或子句级的基于有监督学习的方法也可以应用在 aspects 级别的情感分类。然而关键问题是怎样确定每个情感表达的范围，即，是否情感表达覆盖了句子中的 aspects。

第五节：基于机器学习和基于词典的情感分析方法的讨论

情感倾向的最重要的指示器是情感词，也称作观点词。这些词通常被用来表达正向或负向的情感倾向。例如，good, wonderful, amazing 都是正向的情感词。“bad, poor, terrible”都是负向的情感词。除了单个的词，一些词组、习语也具有情感倾向。例如，“cost someone an arm and a leg”。情感词和词组是进行情感分析的基本工具。情感词和词组的列表称作情感词典（sentiment lexicon）。虽然情感词和词组对于情感分析是重要的，但真正实际应用中，仅仅使用情感词典是不够的。这个问题很复杂，换句话说，情感词典是必须的，但仅仅使用情感词典是不够的。下面将强调这个问题：

1. 一个正向或负向的情感词在不同的领域或许会有相反的情感倾向。
2. 一个包括情感词的句子却不能包含任何情感倾向。例如：“Can you tell me which Sony camera is good？”
3. 讽刺句，不管它包含或不包含情感词，都很难处理。

许多句子没有情感词却包含了情感倾向。

1. 基于机器学习的方法

有监督的学习依赖于训练集。一个分类模型从某个领域的训练集中训练，它可以捕获词在特定领域的情感倾向。但是当把该分类器应用在其他领域时，分类的性能会很差。即有监督的方法是领域相关的。另外，基于机器学习方法不能捕获语言学上的特性，例如，否定词的存在可以翻转原有的词的情感极性。一些词可以使得情感转移等。

2. 基于词典的方法

如果没有标注的数据（训练集）基于词典的方法是一个进行情感分析的比较好的方法。此时可以建立一个通用的（领域独立的）情感分析模型。当然通用模型会有它的问题，下面会讨论到。基于词典的方法已经证明可以在大量的领域工作完成的很好。基于词典的学习是典型的非监督学习方法。它们使用情感词典、观点表达的规则、和句子的释义树（sentence parse tree）来确定一个句子中的每个 aspects 的情感倾向。这些工作也通常会考虑，情感转移词（sentiment shifters），但是子句和影响情感表达的句子构造。情感转移词（sentiment shifters）是能改变情感倾向的词，如否定词。

当然词典方法有它自身的缺点：

- (1) 在一些领域，有些词会具有独特的情感倾向。通用词典不能捕获词在特定领域的倾向。例如，恐龙、青蛙本是名词，不具有情感倾向。但在网络上，它们是具有负面情感的词。
- (2) 在社交媒体上，新词出现的太频繁。词典不能捕获这些新词的话，性能就差。

第六节：实例：为评论预测星级评分

Online reviews are an important asset for users deciding to buy a product, see a movie, or go to a restaurant, as well as for managers making business decisions. The reviews in the e-commerce websites are usually accompanied by ratings, facilitating users to learn the reviews. However, a lot of reviews spread across forums or social media are written in a plain text, which do not have ratings, called non-rated review in this paper. From the perspective of sentiment analysis, this study develops a predictive framework to calculate the ratings for non-rated reviews. The idea behind the framework begins at a couple of observations: (1) the rating of the review depends on sentiment score of aspects as well as the number of positive and negative aspects in the review; (2) the sentiment score of an aspect is determined by its context. Viewing term-pairs co-occurring with aspects as their context, we conceive of a variant of Conditional Random Field model, called SentiCRF, for generating term-pairs and calculating their sentiment scores from a train set. Then we develop a cumulative logit model that uses aspects and their sentiments in a review to predict ratings of the review. In addition, we meet a challenge of class imbalance on calculating sentiment scores of term-pairs. We also propose a heuristic re-sampling method to tackle class imbalance. Experiments conducted on the YELP dataset demonstrate the predictive framework is feasible and effective on predicting rating of reviews.

When we talk about the reviews in the e-commerce, they are usually referred to the texts posted under products, services, or businesses shown in the e-commerce website. Always, they are accompanied by star ratings varying from 1-star to 5-star, which may facilitate visitors to learn the reviews. 图 8-1 is an example of such a review on Amazon website, which involves the Samsung Galaxy S6. A 3-star rating is assigned to the item.

171 of 186 people found the following review helpful

★★★☆☆ A good phone but not an unqualified success

By M. Henson on May 8, 2015

Color: Black

I just moved from the Galaxy Note 3 to the S6 Edge so my review is colored by that comparison. This wasn't the unqualified success I'd hoped for.

图8-1 一条星级评论

However, other types of reviews are also widely spread across forums or social media. They don't have significant differences compared with those in the e-commerce website, except for the lack of the ratings. This paper calls them as the non-rated reviews. For instance, 图 8-2 is a tweet in Twitter, which talks about iPhone 7.



图8-2 一条tweet

Predicting ratings of the non-rated reviews are valuable for building business intelligence applications.

From the perspective of sentiment analysis, this study intends to build a predictive framework to provide star ratings for the non-rated reviews. To meet the challenge, we firstly examine reviews on the e-commerce website. It may be observed that every review involves at least one aspect; we may roughly get positive sentiments from all aspects of a 5-star rating review whereas all aspects in a review with 1-star rating tend to be negative sentiments. 图 8-3 shows

Cheesecake Factory is a solid choice, and you pretty much know what you're getting when you go. Mom and I visited on a Monday evening- no wait, seated right away. *Service* wasn't particularly friendly, and I did have a wait for a *refill*, but it was adequate. As always, there is a huge *menu* with lots of choices and large portions. My *meal* was a bit underseasoned, almost bland, but did the trick. As always, the cheesecakes are the stars, and the Red Velvet *Cheesecake* was delicious.

图 8-3 一条评论中的 aspects 和情感倾向

a 3-star rating review derived from the YELP website where aspects highlighted by a red color have positive sentiments and those highlighted by a yellow color have negative sentiments. Observing the review, we find the number of positive aspects is almost equal to that of negative aspects.

Motivated by the above observations, the task of predicting ratings can be formally decomposed into three steps: extracting aspects, obtaining their sentiments, and then predicting the rating based on the aspects.

To better understand this study, we firstly provide two definitions.

Definition 1: Aspects. We refer a collection of entities and their features in a review as to aspects of the review.

Definition 2: Context of Aspects. The context of an aspect, a , is a set of term-pairs, T . For one term-pair $(w_a, w_b) \in T$, it occurs with the aspect, a , in the same sentence s.t.

$w_i \neq w_j, w_i \neq a, w_j \neq a$. Each term-pair $(w_a, w_b) \in T$ is accompanied by both a positive and negative sentiment score $\{\lambda_p, \lambda_n\}$, respectively.

Following the motivation discussed in Section 1, this study develops a predictive framework for the purpose of predicting the ratings for non-rated reviews. The framework, illustrated in Fig. 4, works with the following steps.

- (1) Use the reviews derived from the e-commerce websites as a train set. In the preprocessing phase, we employ a list of stop words to delete the stop words from the reviews, using Stanford POS-Tagger [24] as a tool to pick up terms and generating term pairs.
- (2) Train a SentiCRF model, obtaining a collection of term-pairs that are attached to sentiment scores.
- (3) Develop a cumulative logit model (CLM).
- (4) Extract aspects of a non-rated review, building their context. Nouns and noun phrases (or groups) identified via a part-of-speech (POS) tagger are considered to be aspects in the review. Although this method is very simple, it is in fact quite effective. Some commercial companies are using this method with several improvements [19].

(5) Map the term-pairs in the context of the aspects to those in the collection generated in step (2). Calculate the sentiment for every aspect in the review based on its context. Then, build a feature vector for the review.

(6) Finally, employ CLM to predict the star rating for the non-rated review.

Both the SentiCRF model and the cumulative logit model are key components of the predictive framework. In the remaining part of this section, we give a detailed discussion about these two components.

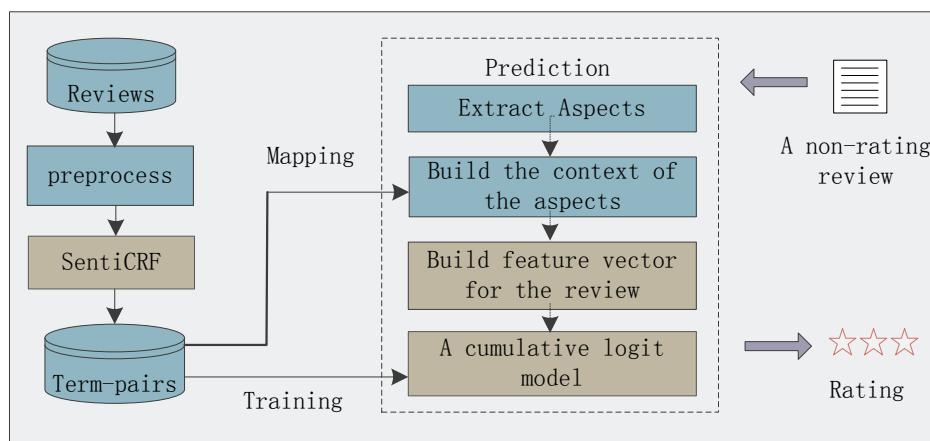


图8-4 工作框架

详细内容参见论文：

Jinagtao Qiu, et al., LEVERAGING SENTIMENT ANALYSIS AT THE ASPECTS LEVEL TO PREDICT RATINGS OF REVIEWS, *Information Science* (2018), DOI: 10.1016/j.ins.2018.04.009,

访问网站 <http://www.biswufe.cn/predratings/> 可以通过使用Demo来了解这项研究。

第九章：知识图谱

第一节：本体和知识库

第二节：语义网 Semantic Web

第三节：知识图谱

第十章：社会网络分析理论

第一节：简介

社会网络通常被定义为一个以人为节点，以人与人之间的交互或关系为边的网络。社会网络分析不是一个新鲜的话题。社会学的相关研究者很早就在社会网络分析领域展开研究。例如，二十世纪初始，社会学家已经开始关注相对小的、内在具有一致性的社交组。在二十世纪三十年代，有社会学家根据美国南方某个社区妇女的通信、参加社会活动的情况研究白人和黑人妇女的社会分层。

1909 年的诺贝尔奖得主 Guglielmo Marconi 提出了一个小世界猜想，即人类社会是一个小世界网络，网络中的平均路径很短。匈牙利作家 Frigyes Karinthy 根据这个猜想发布了一个挑战，即一个人通过另外 5 个人可以与一个他不认识的人建立连接。这是六度分割理论（**Six degrees of separation**）的最早起源。milgram 在 1967 年发表了一个报告“The Small World Problem”。这个报告描述了他做的进行“六度分割假设”检验的实验。他随机从 Nebraska 的 Omaha 城和 Kansas 的 Wichita 城选择个体作为实验的起始点。目的地是居住在波士顿的被选择的实验对象。从起始点，实验参与者通过转发邮件给他的亲戚朋友到的方式，尽量让邮件到达目的地。实验结果，从起始点发出的 296 封邮件最终有 232 封到达了目的地。平均路径长度是 5.5。



图 10-1 六度分割假设的实验

这是最早的六度分割理论的实验。当时这个实验还是在小样本上的检验，现在通过互联网，计算机的发展可以去建立社会网络，去验证该理论。有研究表明，在 MSN 中信息传递的平均路径长度是 6.6。这被广泛认为是六度理论在互联网的验证。

传统社会学的研究通常基于很小的数据集。而 web 的发展，涌现的海量数据，带来了新的挑战。过去几年，随着网络应用的发展，尤其是社交媒体的发展，社会网络吸引了很多来自计算机、管理学、物理学等领域的研究者。

第二节：社会网络的基本知识和度量

这一节主要讲图论的基础知识（2.1），对边和节点的度量（2.2），对静态网络的度量（2.3）和对动态网络的度量（2.4），以及介绍社会网络的基本类型（2.5）。

2.1 图论基础

网络或图是指一种数据结构 $G < V, E >$ 。V 是节点集合；E 是边或链接的集合。如果边是有箭头的，则描述的是有向图，否则是无向图。我们也可以给边分配权重。此时的图称为有权重图，否则称为无权重图。图 10-2 给出了示例。

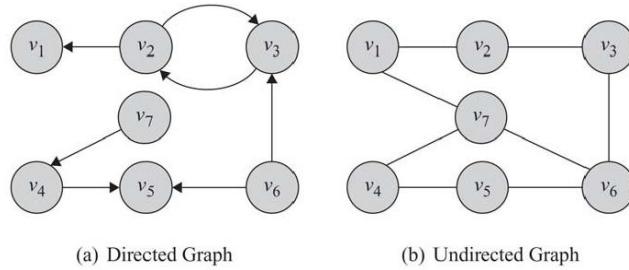


图 10-2：有向图 (a) 和无向图 (b)

(1) 度

连接到一个节点的边的数量称为节点的度。有向图中，度还可以分为入度和出度。入度是链接到该节点的边的数量，出度是从该节点链接出的边的数量。

(2) 度的分布

在一个大的图中，节点度的分布（简称度的分布）是一个重要的属性。它可以透露出图的一些特性。度的分布 P_d 是一个数据选择的节点 v ，它的度为 d 的概率。

$$p_d = \frac{n_d}{n}$$

n_d 是度为 d 的节点数量； n 是图中节点的数量。

要画出度的分布，通常是绘制一个直方图。X 轴描述度，y 轴描述 p_d 。例如，图 10-3 的度的分布是 $d = \{1, 2, 3, 4\}$ ； $p_1 = 1/7$; $p_2 = 4/7$; $p_3 = 1/7$; $p_4 = 1/7$

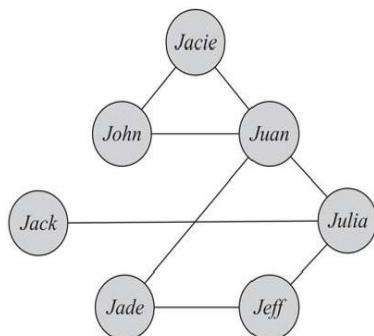


图 10.3：一个示例图

这是 facebook 度的分布图。X 轴是朋友的数量，即度；y 轴是概率。如此，图 10-4 中的一个节点描述 facebook 中的一个用户有确定的朋友数的概率。

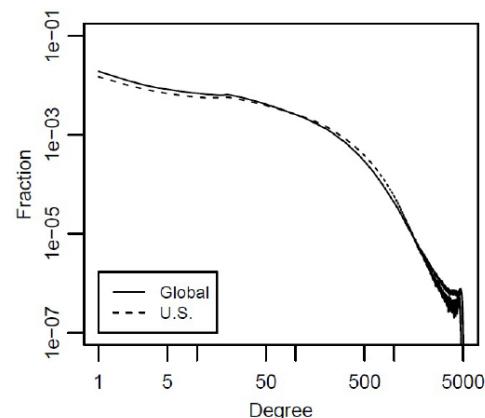


图 10-4 facebook 中社会网络的度的分布图

(3) 二部图

有人将社会网络分成一模 (one-mode) 的和多模 (multimode) 的。一模网络是指所有的网络节点都是一个类型的。大部分的社会网络都是一模的。而多模的网络四章网络节点可以属于不同类型。如，一类是商品、一类是客户。二部图是多模网络。

一个二部图 $G(V; E)$ 中节点集合可以划分成两个部分。如此所有的边它的一端（节点）是在一个集合，而另一端是在另一个集合。换句话说，边连接了两个集合中的节点；不存在边的两个节点属于同一个集合。

二部图在推荐系统中应用的比较多。如图 10-5 所示的二部图，蓝色节点是用户节点；黄色节点指代商品。节点之间的边描述了购买关系。推荐系统需要预测未来不存在边的两个节点之间是否会产生新的边，即会发生购买。

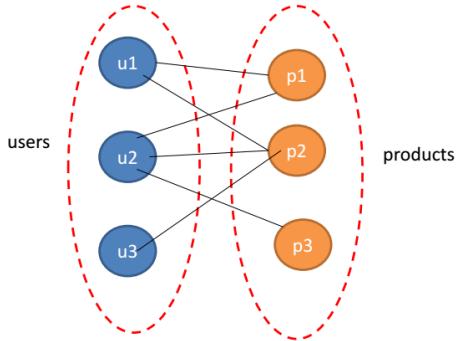


图 10-5 用户-商品构成的二部图

(4) 最短路径

在图中，任何一对节点之间可能存在多条路径。我们经常对最短路径感兴趣。最短路径的应用领域广泛包括在 GPS 导航中规划行程等。

2.2 节点和边的度量

边的度量用于解释节点间相互作用。

(1) Tie strength

两个节点的连接强度取决于他们的邻居的重叠程度。用于共同的邻居越多，连接越强。

(2) Edge Betweenness

一条边的 Edge Betweenness 是指，网络中所有的节点对之间的最短路径通过这条边的数量。Newman 的社区发现算法利用了这个度量。

(3) Centrality

Centrality 定义了网络内的一个节点的重要性。有几种度量方法：degree centrality, betweenness centrality, closeness centrality。

在真实世界中，我们通常认为有很多连接的人，他的重要性高。**Degree Centrality** 就是基于这个思想。一个无向图中，节点 v_i 的 degree centrality 是 $C_d(v_i) = d_i$ 。 d_i 是节点 v_i 的度。有向图中可以使用入度、出度或两者的组合作为 degree centrality 的值。

(4) Closeness centrality 度量一个节点是多接近其他节点。如果一个节点到图中其他节点的距离是最短的，它在这个网络中的位置也是最重要的。

$$C(i) = \sum_j d_{ij}$$

d_{ij} 是从节点 i 到节点 j 的最短路径的链接数。Closeness centrality 对于理解网络中的信息传播很重要。

(5) Betweenness Centrality. 另一种评价节点重要性的方法是考虑，如果图中的最短路径通过一个节点越多，该节点越重要。Betweenness Centrality 的计算公式如下：

$$C_b(v_i) = \sum_{s \neq t \neq v_i} \frac{\sigma_{st}(v_i)}{\sigma_{st}}$$

这里 σ_{st} 是从节点 s 到节点 t 的最短路径数量。 $\sigma_{st}(v_i)$ 是从节点 s 到节点 t 的最短路径中通过节点 v_i 的数量。换句话说，我们是在度量节点 v_i 在连接任意节点对 (s, t) 所扮演的角色的重要性。这个度量称为 betweenness centrality。

2.3 静态网络的度量 (Network Measurement)

网络有很多属性。网络的属性度量可以帮助更好的理解这些网络。本节我们讨论下面几个度量：传递性，聚类系数，中心度 (centrality)、度分布 (degree distribution) 和平均最短路径 (Average shortest-path)。后两种度量与网络的类型相关。

(1) 传递性 Transitivity

Transitivity 是指两个节点之间的关系是“传递的”的程度。这个属性在社会网络中很重要，但在其他网络中不那么重要了。社会网络中，“传递性”可以理解为“朋友的朋友也是朋友”这样的概念。图 10-6 中假设节点 A 和节点 B 相连，节点 B 和节点 C 相连，那么这意味着节点 A 和 C 也是相连的吗？如果回答也是则三者之间 triad 具有很好的传递关系 transitive relationship。如果回答 NO，则这个 triad 缺乏内在的凝聚性。

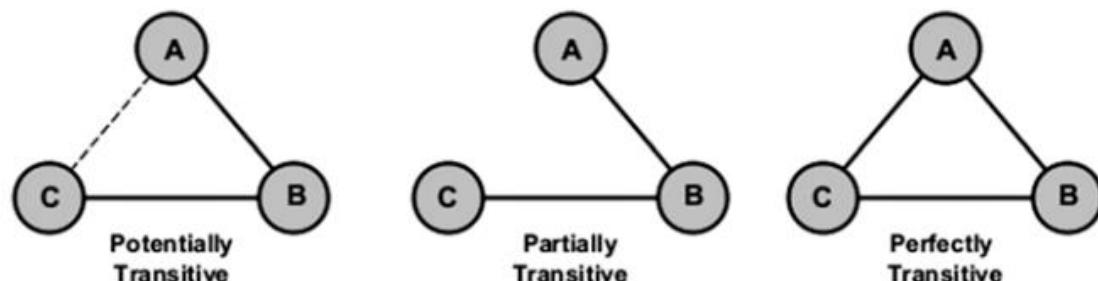


图 10-6 节点之间的传递性

社会网络的平均传递性定义为

$$Transitivity = \frac{3 * No. of triangles}{No. connected triples}$$

Triangle 是是网络中的三角形。Connected triples 是连通的三个节点构成的子图。如图 10-6 中，中间的图是一个 connected triples，但不是 triangles。右边的图是一个 connected triples 也是一个 triangle。如果一个网络的 Transitivity 值为 1，意味着该网络包含所有可能的边。真实的社会网络中通常的 Transitivity 值是 0.3-0.6。即这是正常的值。一个网络的 Transitivity 与聚类系数 clustering coefficient 高度相关。

(2) Clustering coefficient 聚类系数

聚类系数度量多少节点趋于形成一个紧密的子图。对于社会网络来说，聚类系数可以解释成一个节点的两个朋友也趋于是朋友的概率。聚类系数的值为 0-1。高的值指示网络有小世界现象（小世界网络中大部分节点是同质的，他们之间可以在比较少的步数被连接）。聚类系数可以用于区分社会网络和随机网络。

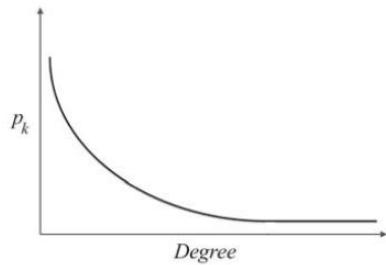
一个节点 i 的聚类系数 C 指示了节点 i 的邻居之间的互相连通性。Social Media Mining 一书中的聚类系数公式就等于上面的 transitivity 公式。

实际应用中，是通过重复 n 次如下的步骤估计图 G 的平均聚类系数：(1) 随机选择一个节点；(2) 随机选择它的两个邻居；(3) 检查这两个节点是否是连通的。

近似的聚类系数是在 n 次操作中，发现的 triangles 的数量。

(3) 度分布

社交媒体中我们可以观察到一个现象。少部分人有上千的粉丝，大部分人粉丝数比较少。这个观察和社交媒体中节点的度分布相关。社交媒体中的节点的度，就是朋友或粉丝的数量。社交媒体中的度分布符合幂律分布 **power-law distribution**。



(a) Power-Law Degree Distribution

图 10-7：幂律分布

图 10-7 中所示的幂律分布有一个长的“尾巴”，因此幂律分布又称为长尾分布 heavy-tailed distribution。符合幂律分布的网络通常是 scale-free networks。

补充：长尾理论

这个词可以说是“逆 20/80 法则”。20/80 法则是说，传统图书商店 20% 的畅销书带来了 80% 的利润。因此，传统书店尽量把畅销书摆上书架。
而电商网站 Amazon 可以保存无限的书。一年只卖出几本的非畅销书非常多。而这些非畅销书的销售总和超过了畅销书。

(4) 平均最短路径

平均最短路径是网络中所有节点之间的最短路径的长度平均值。如果真实的网络，任意两个成员之间的最短路径值比较小，这就称为小世界现象 small-world phenomenon。观察到了小世界现象的网络称为小世界网络 **small-world network**，即小世界网络中的平均最短路径比较小。小世界网络中包含社区结构。

Stanley Milgram 在 1960 年代进行小世界网络实验。他推测世界上的人可以通过最多六个人建立连接，即六度分割理论。

许多研究在社会网络中观察到了小世界现象。例如，图 10-8 显示 2011 年 5 月，Facebook 的社会网络中的平均最短路径是 4.7；美国的社会网络的平均最短路径是 4.3（不知道怎么建立的这个社会网络）

Web	Facebook	Flickr	LiveJournal	Orkut	YouTube
16.12	4.7	5.67	5.88	4.25	5.10

图 10-8：各种社会网络的平均最短路径

(5) 图的直径

图的直径定义为图中任意一对节点间最长的最短路径。最长的最短路径是指：图中的任意一对节点都有一个最短路径。所有的“节点对”的最短路径中的最长的那个称为最长最短路径。数学描述为 $\text{Max}_i (\min(\text{paths}_i))$ ， i 表示一对节点。

以图 10.6 为例：Jacie-Jack: 3, 4, 5，Shortest path: 3。Jade-John: 2, 3, 4，Shortest path: 2。The longest shortest path: 3。考察所有的可能，该图的直径是 3。

(6) 图的密度 Density

图的密度描述了网络中节点之间互相联系的紧密程度。在一个完全图中，图的密度是 1。图 G 的密度 $D(G)$ 定义为

$$D(G) = \frac{2m}{n(n - 1)}$$

密度的值 0-1。公式中 m 是图中边的数量，n 是节点数量， $n(n-1)/2$ 是无向图中最多边的数量。

2.4 动态网络的度量 : Influence and Homophily

社交力量 Social forces 是指连接社会网络中的个体的力量。有三种社交力量 influence, homophily (同质性) 和 confounding.

Influence 是指一个过程：一个个体 (有影响力的人)影响其他个体，使得被影响的个体与有影响力的个体成为越来越相似。

Homophily 是指相似的个体，由于它们的高的相似性，它们最后成为了朋友。

Confounding 是指环境影响个体成为越来越相似。一个例子是生活在俄罗斯的外国人因为环境的影响俄语越来越流利，因此在语言上和俄罗斯人相似。

图 10-9 说明了 Influence 和 homophily 的过程

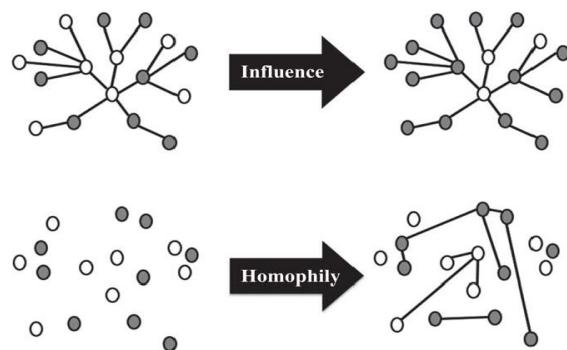


图 10-9 : Influence 和 homophily 的过程

(1) assortativity

Social force 在不同的方向将社会网络中的个体建立连接。当个体之间建立了连接，可以从社会网络上观察到一个有趣的模式，称为 assortativity。也称为社会相似性。例如，社会网络中可以观察到朋友之间在行为、兴趣、属性上高的相似性。

度量 assortativity 可以帮助定量分析 influence 和 homophily。图 10-10 描述了 1994 年美国一个高中的朋友网络。不同的颜色指代不同的种族。白色指代白人，黑人是黑色，拉美裔是灰色。



图 10-10：一所美国高中的朋友网络

从图中可以观察，相同种族的人有很高的 assortativity 现象。特别是在白人和黑人内部，他们的朋友趋近于属于相同的种族。拉美裔趋近于和白人成为朋友。

Newman (Assortative Mixing in Networks) 定义了两种计算 assortativity 系数的方法：一个是基于类别标签，此时给网络中的节点分配了类型标签，如性别，种族等。

$$r = \frac{\sum_i e_{ii} - \sum_i a_i b_i}{1 - \sum_i a_i b_i}$$

此处 e_{ij} 是连接类型 i 和类型 j 的顶点的边的分数。 $a_i = \sum_j e_{ij}$, $b_j = \sum_i e_{ij}$

assorativity 系数的值从 -1 到 1 变化。'1' 指示网络是完美的 assortativite，即所有的边只出现在同类型的节点之间。'-1' 指示网络是完美的 disassortative，即所有边出现在不同类型的节点。

另一个 assortativity 系数计算方法是基于数值属性 numerical attributes，即分配给顶点的一个数值。

$$r = \frac{1}{\sigma_q^2} \sum_{jk} jk(e_{jk} - q_j q_k)$$

对于无向图 $q_i = \sum_j e_{ij}$, 如此

$$r = \frac{1}{\sigma_o \sigma_i} \sum_{jk} jk(e_{jk} - q_j^o q_k^i)$$

对于有向图 $q_i^o = \sum_j e_{ij}$, $q_i^i = \sum_j e_{ji}$

(2) Homophily

Homophily 是由于两个节点之间的相似性导致两个节点建立连接。从时间上看网络表现出了 assortative 属性。要度量 homophily, 即度量网络的 assortativity 系数的随时间变化。设网络有两个 t_1 和 t_2 时刻的 snapshots : $G_{t_1}(V; E_{t_1})$ 和 $G_{t_2}(V; E_{t_2})$, $t_2 > t_1$ 。

Homophily 指数定义为 $H=Q^{t_2}-Q^{t_1}$ 。 Q 是网络的 assortativity 系数。这个公式表示当网络随时间成为了更加 assortativity 时，网络有个高的 Homophily 指数值。我们可以看到 assortativity 的度量是基于静态忘了，而 Homophily 是考察的动态网络。

2.5 社会网络的基本类型

(1) 无尺度网络 Scale-Free

网络的度分布遵守幂律分布的网络。许多网络包括 citation 网络、生物网络、WWW 网络和社会网络都是幂律分布。幂律分布是指少数的节点涉及了链接的大部分，而大部分链接只有很少的链接。

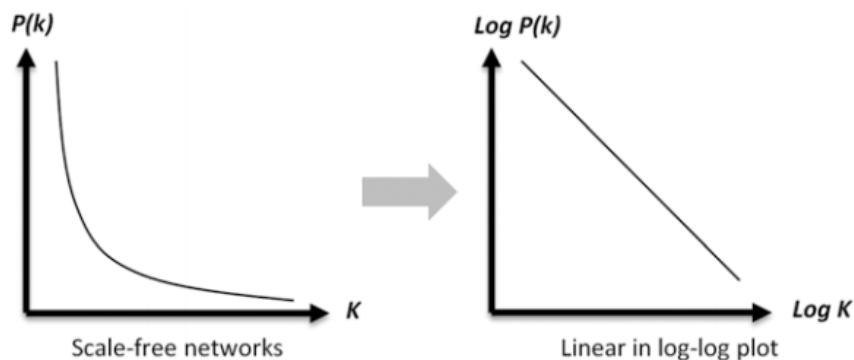


图 10-11 无尺度网络的幂律分布

(2) 小世界网络 small-world network

小世界网络中节点是同质的，节点之间通过少的链接就可以连接。这意味着每个节点有大致相当数量的链接。典型的，小世界网络有高的 clustering coefficient，短的平均最短路径，很多的 hub 节点。小世界网络包含有社区结构。

第三节：节点级的 SNA

节点级的 SNA 除了可以用第二节的度量方法对节点和边进行度量，也可以用一些算法对每个节点计算评分。也包含对链接的预测，即未来两个节点是否会发生关系的预测。节点分类是社会网络中，一些节点有标签。使用这些有标签的节点去预测没有标签的节点的类别。

我们首先看一下对边和节点之间影响力的度量

3.1 PageRank

pagerank 的评分原理是，当 Web 冲浪者在 Web 上随机游走，他对某些节点的访问次数会比其他节点更多。PageRank 的思路就是，在随机游走过程中越频繁被访问的网页越重要。PageRank 算法基于下面的公式：

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

$PR(p)$ 是网页 p 的 PageRank 评分； d 是一个 damping 系数[0-1]； N 是 Page 数量； $M(p)$ 是链接到网页 p 的网页集合； $L(p)$ 是网页 p 的链出数量。

PageRank 算法如下：

1. $PR_0 \leftarrow E$
2. loop:
3. 计算 $PR_{i+1}(p_j)$
4. $g \leftarrow ||PR_i||_1 - ||PR_{i+1}||_1$
5. $PR_{i+1} \leftarrow PR_{i+1} + g * E$
6. $\text{delta} \leftarrow ||PR_{i+1} - PR_i||_1$
7. While $\text{delta} < \text{epsilon}$

E 是每个网页的初始得分，如可设置 $E=\{0.5, \dots, 0.5\}$ ； PR_i 表示是在第 i 轮迭代中计算的 PR 评分； $||PR||_1$ 表示向量的 L1 范数。

$$||PR||_1 = \sqrt{|PR(1)| +, \dots, |PR(N)|}$$

PageRank 更新公式中 damping 系数的解释：

When the random surfer walks on the link graph, she/he does not necessarily always follow the existing hyperlinks. There is a small probability that she/he will jump to any other page uniformly. This small probability can be represented by $(1 - d)$, where d is called the damping factor.

我的解释就是：可以理解的，节点 p_i 的 pagerank 评分是来自于链接到 p_i 节点的其他节点的 pagerank 评分平均分配给 p_i 的评分，再求的和。还有一种可能是，网上冲浪时，用户以一种随机方式访问到了 p_i 节点，而并不是跟随超链（如在地址栏输入 p_i 的网址）。从网络中的每个节点都有这样的可能。因此这个概率是 $(1-d)$ 。此种情况下分配给 p_i 的评分是 $1/N$ 。damping 的取值有一个经验值是 0.85（听一个报告介绍的，我还没找到出处）

3.2 Hits 算法

Hyperlinks Induced Topic Search (Hits)算法发现 Hubs 和 Authorities 网页，或评价每篇网页的 Hubs 和 Authorities 特性。一个 Hub 网页可以通过它导向许多其他网页；一个好的 Authority 网页被许多 Hub 网页链接；Hits 认为网页的重要性体现在它的 Hub 和 Authority 特性上。

PageRank 和 Hits 都是网页搜索算法。相对于 PageRank，Hits 算法有下面的特性：

(a) Hits 是查询依赖的，即 Hits 算法根据查询的检索结果的网页集合构建有向图。而 PageRank 是在全局网页有向图上进行计算。

(b) Hits 为每篇网页计算 Hub 和 Authority 两个得分。而 PageRank 只是计算一个得分。

Hits 算法的步骤：

(1) 根据查询获得一个结果集，由该结果集建立有向图。

(2) 每个节点被分配一个 Hub 和 Authority 初始得分 1。

(3) 在迭代中，每趟根据更新规则更新节点的 Hub 和 Authority 得分。

- Authority 更新规则： $\text{auth}(p)=\sum_{i=1}^n \text{hub}(i)$ ，即连接到 p 页面的 n 个页面的 hub 分值之和。
- Hub 更新规则： $\text{hub}(p)=\sum_{i=1}^n \text{auth}(i)$ ，即 p 链出到的 n 个页面的 authority 分值之和。

(3) 对所有节点构成的 auth 向量和 hub 向量规范化，满足 $\sum_{i=1}^n a_i = \sum_{i=1}^n h_i = 1$

3.3 链接预测

链接预测是在当前的社会网络中预测两个没有边相连的顶点在未来会发生连接。也可以是推理是否观察到的社会网络中有丢失的连接。一个例子是我们观察到了一个恐怖组织的社会网络，我们可以推测是否没有在社会网络上观察到有互动的两个成员其实是一起工作的。在知识图谱中，知识图谱补全任务就是考虑，初始建立的知识图谱不是完备的，需要通过推理来建立实体（节点）之间的关系（对应我们这里的链接预测）。

关于链接预测的研究有很多。有基于概率模型的（参考 Social Network Data Analysis 一书），也有将链接预测看做是一个有监督的分类任务。

3.3.1 基于特征集合的有监督的链接预测

构建该分类任务的训练集有两种考虑。 (1) 如果我们有一个动态的社会网络。为在 t1 时刻没有边的链接 , 但在 t2 时刻 (注 : $t_2 > t_1$) 有边的链接的的两个顶点 $\langle u, v \rangle$ 分配一个类别标签 +1 。 (2) 《 Social Network Data Analysis 》一书中说 , 对于一个静态社会网络 $G \langle V, E \rangle$ (注 : V 是网络的顶点集合 , E 是网络的边的集合)中的一对顶点 $\langle u, v \rangle$ 。这条数据被分配一个标签

$$y^{\langle u, v \rangle} = \begin{cases} +1, & \text{if } \langle u, v \rangle \in E \\ -1, & \text{if } \langle u, v \rangle \notin E \end{cases}$$

我的理解是 , 可以这么建立训练集 , 但下面要讨论两个顶点的拓扑统计特征时 , 要对 $\langle u, v \rangle$ 之间如果有边的链接忽略。知识图谱推理上的路径排序算法 Path Ranking Algorithm 就是这么做的。路径排序算法是通过考察网络拓扑推理是否两个实体 (节点) 之间存在关系 , 对应社会网络中两个节点间是否存在链接。

使用该数据集可以训练一个分类模型 , 用来预测一对顶点 $\langle u, v \rangle$ 未来是否会发生连接。许多分类模型可以完成此任务 , 例如 SVM , NB 等。因此 , 该方法中最重要的是如何构建特征集合。我们下面通过网络拓扑结构可以构建的两个节点 $\langle u, v \rangle$ 之间的各种统计特征。它的优点是只需要考察网络的拓扑结构 , 不需要领域知识。 (参看论文 : David Liben-Nowell, The Link-Prediction Problem for Social Networks, JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE AND TECHNOLOGY—May 2007, DOI: 10.1002/asi)

(a) 共同邻居数 : $|\Gamma(x) \cap \Gamma(y)|$ 。 $\Gamma(x)$ 是节点 x 的邻居

(b) 杰卡德系数

$$\text{Jaccard coefficient}(x, y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

(b) 路径特征 : 两个节点间的最短路径长度。

(c) Hitting time: 图上的两个节点 x, y 的 Hitting time $H(x, y)$ =一个从 x 开始的随机游走到达 y 的期望步数。

(d) 基于顶点和边的属性

$$\text{preferential attachment score}(x, y) = \Gamma(x) \cdot \Gamma(y)$$

(e) SimRank : 两个节点的相似度 , 由他们的邻居的相似性确定。

$$\text{simRank}(x,y) = \begin{cases} 1 & \text{if } x = y \\ \gamma \cdot \frac{\sum_{a \in \Gamma(x)} \sum_{b \in \Gamma(y)} \text{simRank}(a,b)}{|\Gamma(x)| \cdot |\Gamma(y)|} & \text{otherwise} \end{cases}$$

3.3.2 协同过滤

在二部图上预测边的链接是推荐系统的协同过滤算法基本原理。我们下面介绍协同过滤算法。

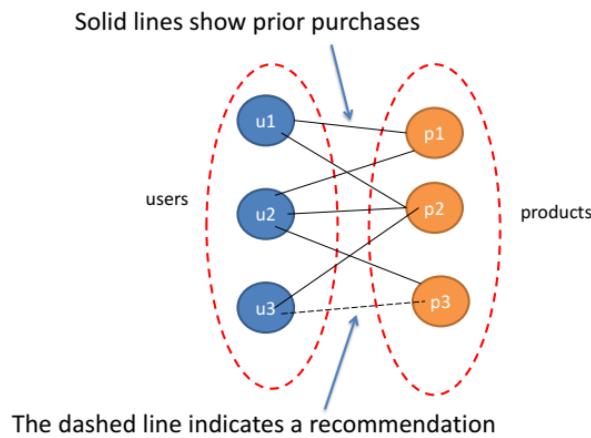


图 10-12 协同过滤的基本原理

图 10-12 的二部图中，顶点分成两个集合：一边是用户，一边是商品。实线的边描述了客户购买了商品。虚线的边是待预测的边，如果可以预测到该链接，则可以将商品推荐给相应的用户。在协同过滤中，可以将上述的二部图转换成一个“客户-商品”矩阵。矩阵的元素是客户对商品的评分。

	Lion King	Aladdin	Mulan	Anastasia
John	3	0	3	3
Joe	5	4	0	2
Jill	1	2	4	2
Jane	3	?	1	0
Jorge	2	2	0	1

图 10-13：客户-商品矩阵

图 10-13 矩阵中的“？”表示待预测的评分。这个预测工作可以使用矩阵中的其他评分来完成，称为基于内存的协同过滤（memory-based collaborative filtering）。基于内存的协同过滤中基于下面的假设：

- (1) 之前在商品上有相似评分的用户，很可能未来在同一商品上仍有相似的评分。

基于这一假设的 memory-based CF 技术称作 **user-based CF**。

(2) 之前接受了用户的相似评分的商品很可能继续接受未来用户的相似的评分。

基于这一假设的 memory-base CF 技术称作 **item-based CF**.

User-based Collaborative Filtering

基于用户的协同过滤中的两个用户的相似度是

$$sim(U_u, U_v) = \cos(U_u, U_v) = \frac{U_u \cdot U_v}{\|U_u\| \|U_v\|} = \frac{\sum_i r_{u,i} r_{v,i}}{\sqrt{\sum_i r_{u,i}^2} \sqrt{\sum_i r_{v,i}^2}}$$

它计算两个用户 u 和 v 的相似度。 $r_{u,i}$ 是用户 (user) u 分配给商品(item) i 的评分.

例如 , 图 10-13 中 john 和 jane 的相似度是

$$sim(Jane, John) = \frac{3 \times 3 + 1 \times 3 + 0 \times 3}{\sqrt{10} \sqrt{27}} = 0.73$$

Jane 和 Jorge 的相似度是

$$sim(Jane, Jorge) = \frac{3 \times 2 + 1 \times 0 + 0 \times 1}{\sqrt{10} \sqrt{5}} = 0.84$$

在 user-based collaborative filtering, 预测用户 u 给商品 i 的评分是

$$r_{u,i} = \bar{r}_u + \frac{\sum_{v \in N(u)} sim(u, v)(r_{v,i} - \bar{r}_v)}{\sum_{v \in N(u)} sim(u, v)}$$

\bar{r}_u 是 user u 给出的平均评分, \bar{r}_i 是 item i 接收的平均评分。 用户 u 的最相似的用户称作 u 的邻居, **N(u)**。 这里的 u 的邻居数量预先设定, 例如前 10 个最相似的邻居。 上图中, 当设邻居数为 2, 预测 jane 给 Aladdin 的评分是

$$\begin{aligned} r_{Jane, Aladdin} &= \bar{r}_{Jane} + \frac{sim(Jane, Joe)(r_{Joe, Aladdin} - \bar{r}_{Joe})}{sim(Jane, Joe) + sim(Jane, Jorge)} \\ &\quad + \frac{sim(Jane, Jorge)(r_{Jorge, Aladdin} - \bar{r}_{Jorge})}{sim(Jane, Joe) + sim(Jane, Jorge)} \\ &= 1.33 + \frac{0.88(4 - 2.75) + 0.84(2 - 1.25)}{0.88 + 0.84} = 2.33 \end{aligned}$$

Item-based Collaborative Filtering

基于商品的协同过滤发现最相似的商品。 图 10-14 显示了商品 items 的向量。 user u 给 item i 的评分计算如下

$$r_{u,i} = \bar{r}_i + \frac{\sum_{j \in N(i)} sim(i, j)(r_{u,j} - \bar{r}_j)}{\sum_{j \in N(i)} sim(i, j)}$$

	Lion King	Aladdin	Mulan	Anastasia
John	3	0	3	3
Joe	5	4	0	2
Jill	1	2	4	2
Jane	3	?	1	0
Jorge	2	2	0	1

图 10-14 item 向量

$$\bar{r}_{Lion\ King} = \frac{3 + 5 + 1 + 3 + 2}{5} = 2.8$$

$$\bar{r}_{Aladdin} = \frac{0 + 4 + 2 + 2}{4} = 2$$

$$sim(Aladdin, Lion\ King) = \frac{0 \times 3 + 4 \times 5 + 2 \times 1 + 2 \times 2}{\sqrt{24} \sqrt{39}} = 0.84$$

现在假设邻居数为 2。通过计算可以得到 Lion King 和 Anastasia 是和 Aladdin 最相似的两个邻居。使用 item-based collaborative filtering 可以预测 Jane 对 Aladdin 的评分

$$\begin{aligned}
 r_{Jane, Aladdin} &= \bar{r}_{Aladdin} + \frac{sim(Aladdin, Lion\ King)(r_{Jane, Lion\ King} - \bar{r}_{Lion\ King})}{sim(Aladdin, Lion\ King) + sim(Aladdin, Anastasia)} \\
 &\quad + \frac{sim(Aladdin, Anastasia)(r_{Jane, Anastasia} - \bar{r}_{Anastasia})}{sim(Aladdin, Lion\ King) + sim(Aladdin, Anastasia)} \\
 &= 2 + \frac{0.84(3 - 2.8) + 0.67(0 - 1.6)}{0.84 + 0.67} = 1.40. \tag{9.24}
 \end{aligned}$$

3.4 节点分类

节点分类就是：在一个社交网络，网络上的节点是有属性的例如，人口统计学特征，兴趣、宗教信仰等。有些部分节点是有 label 的。节点分类的问题就是用这些 label 的节点去判断未 label 的节点的 label。这样的 label 可以是对某事物有兴趣，或是否是某类人。

节点分类有很多应用，如在线广告，节点分类用来判断是否成员对某类广告感兴趣，推荐系统判断对某类商品是否感兴趣。

节点分类中，每个节点的特征包括自身的属性特征，如，如果节点是人，即性别、年龄、兴趣等。还有一部分是链接特征，即该节点和邻居节点构成的链接特征。如，它的邻居节点中，某个标签的节点数。一个基本的节点分类方法为每个节点建立特征向量。特征可以包括节点的链接特征，也可以包括节点本身的属性特征。

节点分类采用迭代的方法：

(a) 初始给定一个社会网络，其中有标签的节点集合是，用有标签节点集合训练一个分类器（如 NB , SVM 等）。

(b) 选择一个 unlabeled 节点分类，分类后重新为每个节点计算链接特征。重新训练分类器，再重复上面的步骤。

图 10-15 显示在一个简单的图上的局部迭代的两个步骤。深色的节点是贴了标签的节点。在第一步，节点 X 通过分类器贴标签'18'。基于这个新的链接特征，在第二次迭代中这个标签‘传播’到节点 Y。

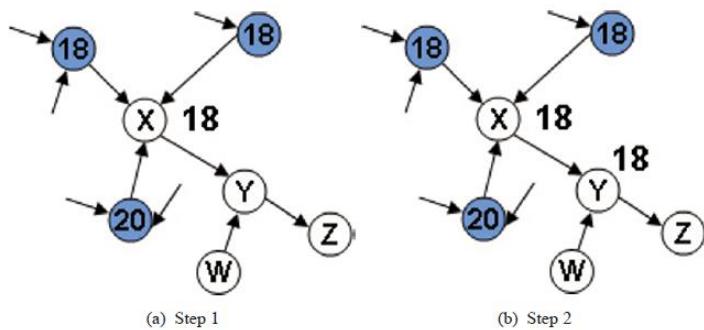


图 10-15：节点分分类示例

第四节：网络级的 SNA

网络级的 SNA 包括：社区发现和 blockmodel。

4.1 社区发现

社区发现是社会网络分析中最重要的问题之一。社区没有一个明确的定义，一个被普遍接受的概念是：一个社区是网络的一个子图，子图中的顶点更倾向于和子图内的顶点有边相连。社区内成员之间联系比与社区外成员的联系更紧密。社区内成员之间联系比与社区外成员的联系更紧密。准确地发现社区可以很好的描述社会网络中一些成员的共同属性，如共同爱好、社会功能等。图 10-16 中不同颜色的节点表示不同的社区。社会网络中的个体经常基于兴趣会形成“组”。研究社会网络时，检测社区可以帮助确定这些兴趣组。例如，在线图书销售中，确定兴趣组可以帮助进行图书推荐；可以帮助检测社交网络中的恐怖组织等。

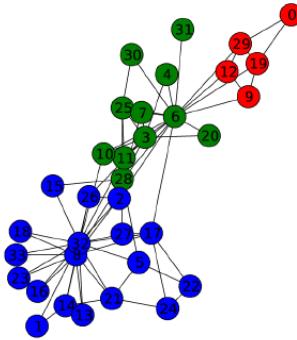


图 10-16：一个含有社区结构的社会网络

社区检测是这样一个任务，对于社会网络 $G(V; E)$ ，社区检测是发现一个社区集合 $\{C_i\}_{i=1}^n$ ，且 $\cup_{i=1}^n C_i \subseteq V$

有很多社区检测算法。图划分的方法是较早，较成功的社区发现方法。其中 MinCut 是最简单的图划分算法，该算法寻找移除最少的边，以将连通图划分成两个子图，此时的子图即从网络中获得的社区。MinCut 在一些情况对图划分不均衡，经常其中一个子图很小。一些改进算法如，Normalized Cut、Ratio Cut、Min-Max 等试图解决此问题。

iGraph 包（Python 和 R 都有该包）提供了 9 种社区检测方法，包括基于 edge-betweenness 的图划分方法、基于随机游走的方法、最大化 modularity 度量的方法。我们不一一介绍。

4.1.1 Edge-betweenness 方法

社会网络中的每条边计算一个 betweenness 评分。它是社会网络中最短路径集合中通过该边的数量。使用 edge-betweenness 进行社区检测的思想是，连接不同社区的边应该有更高的 edge-betweenness 评分。例如图 10-17 中连接社区的边有高的 edge-betweenness 评分。

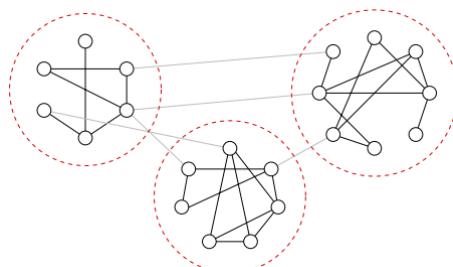


图 10-17 一个社会网络的社区结构

如果逐步移走这些高 edge-betweenness 评分的边，会最终得到一个层次图，如图 10-18 所示。图的根节点是所有节点属于同一个社区；叶子节点是每个节点属于只有它自己的社区。

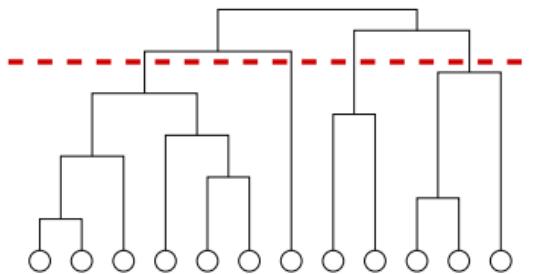


图 10-18：一个社区结构的层次图

算法的工作步骤：

- (1) 计算社会网络中所有边的 edge-betweenness 评分。
- (2) 移走最高评分的边，重新计算所有边的 edge-betweenness 评分。
- (3) 重复步骤 1
- (4) 最后得到层次聚类图 dendrogram

该算法的详细内容请见 newman 的论文 “Finding and evaluating community structure in networks”

4.1.2 图上的随机游走

一个图上的 t 步随机游走对两个节点间的所有长度为 t 路径的概率求和。它度量了两个节点间的路径的数量。两个节点间的路径越多，两个节点间的转移概率越高。图上的随机游走有两种类型：正向随机游走 (forward random walk) 或反向随机游走 (backward random walk)。正向随机游走 $P_{t|0}(k|j)$ 发现当从节点 k 开始一个 t 步随机游走结束在节点 j 的概率。反向随机游走 $P_{0|t}(k|j)$ 发现当在节点 j 结束一个 t 步随机游走，而起始节点是 k 的概率。正向和反向随机游走间存在一个规则 $P_{0|t}(k|j) \propto P_{t|0}(j|k)P_0(k)$ 。 $P_0(k)$ 是选择 k 作为起始节点的概率。

 Social Network	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th></th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th></tr> </thead> <tbody> <tr> <th>A</th><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr> <th>B</th><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr> <th>C</th><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr> <th>D</th><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> <tr> <th>E</th><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> <tr> <th>F</th><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr> </tbody> </table> Adjacency Matrix		A	B	C	D	E	F	A	1	1	1	0	0	0	B	1	1	1	1	0	0	C	1	1	1	1	1	1	D	0	1	1	1	1	0	E	0	0	1	1	1	0	F	0	0	1	0	0	1	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th></th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th></tr> </thead> <tbody> <tr> <th>A</th><td>0.6</td><td>0.2</td><td>0.2</td><td>0</td><td>0</td><td>0</td></tr> <tr> <th>B</th><td>0.13</td><td>0.6</td><td>0.13</td><td>0.13</td><td>0</td><td>0</td></tr> <tr> <th>C</th><td>0.08</td><td>0.08</td><td>0.6</td><td>0.08</td><td>0.08</td><td>0.08</td></tr> <tr> <th>D</th><td>0</td><td>0.13</td><td>0.13</td><td>0.6</td><td>0.13</td><td>0</td></tr> <tr> <th>E</th><td>0</td><td>0</td><td>0.2</td><td>0.2</td><td>0.6</td><td>0</td></tr> <tr> <th>F</th><td>0</td><td>0</td><td>0.4</td><td>0</td><td>0</td><td>0.6</td></tr> </tbody> </table> Transition Probability Matrix		A	B	C	D	E	F	A	0.6	0.2	0.2	0	0	0	B	0.13	0.6	0.13	0.13	0	0	C	0.08	0.08	0.6	0.08	0.08	0.08	D	0	0.13	0.13	0.6	0.13	0	E	0	0	0.2	0.2	0.6	0	F	0	0	0.4	0	0	0.6
	A	B	C	D	E	F																																																																																														
A	1	1	1	0	0	0																																																																																														
B	1	1	1	1	0	0																																																																																														
C	1	1	1	1	1	1																																																																																														
D	0	1	1	1	1	0																																																																																														
E	0	0	1	1	1	0																																																																																														
F	0	0	1	0	0	1																																																																																														
	A	B	C	D	E	F																																																																																														
A	0.6	0.2	0.2	0	0	0																																																																																														
B	0.13	0.6	0.13	0.13	0	0																																																																																														
C	0.08	0.08	0.6	0.08	0.08	0.08																																																																																														
D	0	0.13	0.13	0.6	0.13	0																																																																																														
E	0	0	0.2	0.2	0.6	0																																																																																														
F	0	0	0.4	0	0	0.6																																																																																														

图 10-19：社会网络和转移概率矩阵

对于一个社会网络，我们可以使用图上的随机游走计算该网络的 t 步转移概率矩阵。从节点 j 到到节点 l 的一步转移概率 $P_{t+1|t}(l|j)$ 是

$$P_{t+1|t}(l|j) = \begin{cases} (1-p)M_{jl} / \sum_i M_{ji} & \forall l \neq j \\ p & l = j \end{cases}$$

M 是一个邻接矩阵； M_{ij} 指示节点 i 和节点 j 之间的边； p 是自转移概率。用矩阵 A 描述这个一步转移概率。此处 $A[j, l]$ 是从节点 j 到节点 l 的转移概率 / $P_{t+1|t}(l|j)$ 。从一步转移概率矩阵 A 可以计算 t-step 转移概率矩阵 $A^t = A(A \dots (A))$ 。按照 $A^t = Z^{-1} \times A^t \times Z$ 对 A^t 规范化，Z 是对角阵。图 10-19 展示了一个一步转移概率矩阵，这里自转移概率是 p=0.6。

4.1.3 walktrap 基于随机游走的方法

基于随机游走的社区检测方法基于一个直觉：社会网络中的或图上的随机游走会通常陷入到社会网络中的紧密连接的部分，即社区。

Walktrap 定义了两个节点 i 和 j 之间的距离

$$r_{ij} = \sqrt{\sum_{k=1}^n \frac{(P_{ik}^t - P_{jk}^t)^2}{d(k)}}$$

P_{ik}^t 是从节点 i 到节点 k 的 t 步转移概率。d(k) 是节点 k 的度。由一个社区 C 到一个节点 j 的距离定义为

$$P_{Cj}^t = \frac{1}{|C|} \sum_{i \in C} P_{ij}^t$$

由此也可以获得两个社区间的距离。

$$r_{C_1 C_2} = \left\| D^{-\frac{1}{2}} P_{C_1}^t \cdot - D^{-\frac{1}{2}} P_{C_2}^t \cdot \right\| = \sqrt{\sum_{k=1}^n \frac{(P_{C_1 k}^t - P_{C_2 k}^t)^2}{d(k)}}$$

如此，Walktrap 就可以利用自底向上层次聚类算法来发现社区。初始，所有节点自己就是一个社区，每一次合并距离最近的两个社区。最终可以获得一个聚类树 dendrogram。

详细描述请参见论文“Computing communities in large networks using random walks”

4.1.4 最大化 modularity

Newman 在他的论文 Finding and evaluating community structure in networks 中定义了一个社区划分结果的评价标准，称为 modularity。其公式如下：

$$Q = \sum_i (e_{ii} - a_i^2) = \text{Tr } \mathbf{e} - \|\mathbf{e}^2\|$$

假设一个图被划分成了 k 个社区。设 \mathbf{e} 是一个 $k \times k$ 对称矩阵， e_{ij} 是所有的连接社区 i 和社区 j 的边的数目除以网络中的边的总数（或称为连接社区 i 和 j 的边的分数）。

$\text{Tr } \mathbf{e} = \sum_i e_{ii}$ 是社区 i 内部的边的分数。 $a_i = \sum_j e_{ij}$ 描述了连接到社区 i 的边的分数。 $\|\mathbf{x}\|$ 指示矩阵 \mathbf{x} 的元素和。 $Q=1$ 表示社区划分结果最好。

由 modularity 的定义出发，许多研究者研究使得 modularity 最大化的社区检测方法。

4.1.5 基于非负矩阵分解的社区发现

我的研究提出了一种基于随机游走和非负矩阵分解的社区发现方法。详细内容如下：

一个网络通常可以用一个邻接矩阵 G 来描述。如果 $G[i, j]$ 是 0，指示节点 i 和 j 没有边的链接。然而节点 i 和 j 存在路径时，表示两个节点间有一个潜在的关系。我们使用 4.1.2 节的图上的随机游走为图 G 建立一个 t 步转移概率矩阵 V 。

有研究显示非负矩阵分解可以用于“发现社区”的工作。我们首先简单介绍一下非负矩阵分解（NMF）。给定一个目标函数

$$J = \|V - WH\|_F^2 \quad (1)$$

where V is an $n \times m$ matrix with $V_{ij} \geq 0$ and k is a pre-specified positive integer with $k < \min(n, m)$, the objective of NMF is to find two non-negative matrices $W \in \mathbb{R}_{n \times k}$ and $H \in \mathbb{R}_{k \times m}$ that satisfy $\min_{W, H} \|V - WH\|_F^2$. In a usually implementation of NMF, W and H is initiated to a randomly generated matrix with $W_{ij} \geq 0$ and $H_{ij} \geq 0$, respectively. W and H are then improved or updated with alternating iterations of the algorithm. Generally, the results of NMF, matrices W and H , are not unique. The multiplicative update rules [17] are written as

$$\begin{aligned} H_{a\mu} &\leftarrow H_{a\mu} \frac{(W^T V)_{a\mu}}{(W^T W H)_{a\mu}}; \\ W_{ia} &\leftarrow W_{ia} \frac{(V H^T)_{ia}}{(W H H^T)_{ia}} \end{aligned} \quad (2)$$

When applying NMF for community discovery, V is an adjacency matrix where elements represent links among nodes. It is straightforward to interpret the factorization due to non-negativity. For example, with respect to $m \times n$ matrix W

obtained from NMF, m represents the number of community, n indicate the number of member and $W[i,j]$ refer to the extent of member i belonging to community j. Matrix H can also be interpreted in this way.

设 V 是这个 t-step 转移概率矩阵. When employing NMF to factorize V , we obtain two matrices W and H , which can be interpreted as community division based on forward random walk and backward random walk, respectively. To obtain the cores of overlapping community from matrix W and H , we need to divide matrix W and H further. 表 10.1 illustrates an example of matrix H .

表 10.1 矩阵 H 的一个例子

Members	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Community C_1	0.11	0	0	0.1	0.14	0	0.87	1.08	1.01	1.46	1.13	1.03	1.59	0.41
Community C_2	1.21	1.05	1.05	1.22	1.17	1.70	1.06	0.20	0	0	0.18	0.24	0.06	0

观察表 10.1 我们可以发现矩阵 H 透露出了关于社区结构的丰富的信息。矩阵的元素是一个指示符, 显示一个节点属于一个给定的社区(行向量)的程度。例如, 表中节点 7 在 C_1 和 C_2 的值是 0.87 和 1.06。比较到行向量中的其他节点这两个值都很大。这意味着节点 7 在一个很大的程度上同时属于两个社区。节点 14 比向量 C_1 和 C_2 中的其他元素有一个很小的值。将节点 14 看作是离群点, 不属于任何社区, 这样也是合理的。通过观察图 10-20, 我们可以确定上述分析的合理性。当分配成员 i 只能到一个社区时, 即它的最高评分对应的社区, 我们可以得到一个“硬划分”的社区结构。

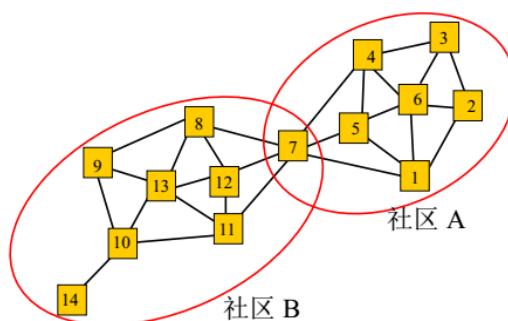


图 10-20 社区检测结果

4.2 Stochastic Block Model

Stochastic Block Model (SBM)可以比作是社会网络中的层次聚类。Block Modeling 的目标是将一个大的, 内在不一致的网络约减成小的可解释的网络。它基于这样的思想, 首先给予等价性一个定义, 然后网络中的节点可以按照等价程度组成组。如图 10-21 所示。

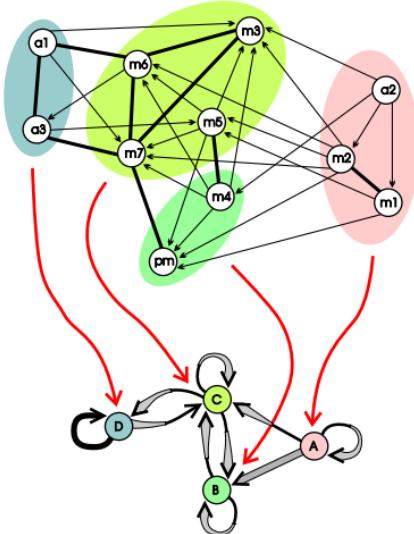


图 10-21 Blockmodeling 示意图

Blockmodeling 的主要问题就是在一个网络中确定节点的簇 clusters，它们共享了定义好的某种特性。形式化定义如下：

在一个网络 $N = (U, R)$, $R \subseteq U \times U$, U 是 unit 即网络的节点, R 是网络节点的关系。节点的簇共享按照 R (链接或边) 定义的结构上的特征。簇内的节点有着相同或相似的与其他节点的连接模式。然后可以形成一个簇的集合 $C = (C_1, \dots, C_k)$ 。 C 是节点集合 U 的划分。一个聚类结果 C 也将边的集合 R 划分成块 blocks。 $R(C_i, C_j) = R \cap C_i \times C_j$ 。每个 block $R(C_i, C_j)$ 包含属于 C_i 和 C_j 的节点和连接 C_i 和 C_j 的边。

有两个基本的关于等价性的定义。

- 等价的两个节点有相同的和“它们的相同的邻居”连接的模式。称作结构性等价
- 等价的节点和不同的邻居有相同或相似的连接模式。称作 general 等价

形式上，block model 和社区比较像。但社区是基于这样的假设，社区内的成员联系紧密，社区间的成员联系松散。而 block model 放松了这个假设，它只要求节点在链接到其他 block 的方式是等价的，这些节点就可以划分到同一个 block。例如，二部图中一个节点集合只和另外的节点集合发生连接。

SBM 是一个 generative model 生成模型，即一个 SBM 假定了一个网络的潜在结构，并给出了建立一个有这样结构的网络的过程。通过分配所有的节点到无交集的 (disjoint) 集合 (这些集合称为 block) 和设定 block 内部和 block 之间边的数量，一个 SBM 被定义。在一个 block 内的所有节点具有和其他 block 链接的相同的概率，也包括和自己 block 链接的相同的概率。

可以说，SBM 按照网络内节点的角色将他们放到不同的组。

第十一章：Gephi：社会网络可视化工具

1. 按照边的属性给边点上颜色

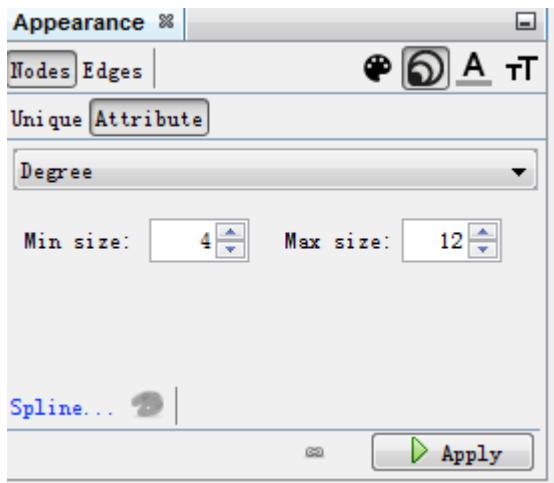
(1) 在 Data Laboratory , 选择 Edges

The screenshot shows the Gephi interface with the 'Data Laboratory' tab selected. A workspace named 'Workspace 1' is open. In the 'Data Table' view, there is one row under the 'Edges' tab. The columns are 'Source', 'Target', and 'Type'. The data row shows '1' in 'Source', '2' in 'Target', and 'Undirected' in 'Type'. Other tabs like 'Nodes' and 'Configuration' are visible at the top.

点击该视图底下的 “Add Column” 按钮。增加一列名叫 Appearance , 可以复制 label 的内容。然后在 overview 的 Appearance 中的 attribute , 选择 appearanc. 设置不同的颜色即可

The screenshot shows the 'Appearance' panel in the Gephi interface. It has tabs for 'Nodes' and 'Edges', with 'Edges' selected. Under 'Unique' and 'Attribute', the 'Appearance' dropdown is chosen. Two categories are listed: 'A' (represented by a red square) and 'B' (represented by a black square). Each category has a percentage value of '(50%)'. At the bottom right of the panel is an 'Apply' button.

2. 按照 degree 设置节点的大小。



选择 nodes->Attribute, 点击下拉菜单 , 选择 degree , 然后点击图标  , 在出现的 Min Size 和 Max Size 中选择数值 , 然后点击 Apply

第十二章：社会网络分析实践和案例

这一章我们使用 Python 的 NetworkX 包来分析社会网络。用 Python 分析社会网络的优点是简单方便。当前 NetworkX 的 2.0 版本不知道为什么不能在 spyder 中运行。下面的代码是 python 3.5 和 networkX2.0 版本。

第一节：分析实例

1. 绘制网络的度分布

```
import networkx as nx
from collections import Counter
from operator import itemgetter

g = nx.karate_club_graph()
nlist = g.nodes()
dlist = g.degree(nlist)
clist=[]

for node in nlist:
    clist.append(dlist[node])

r = Counter(clist)

for degree in r:
    r[degree]=r[degree]*1.0/len(nlist)

d = r.items ()
```

```
c= sorted (d , key =itemgetter (0))
print(c)
```

NetworkX 包自带了 karate 网络。nx.karate_club_graph()创建这个网络。nodes()方法获得网络所有的节点。degree 方法获得一组节点的度。

2. 计算网络的直径

方法 shortest_path(g,source=n1,target=n2)展现节点编号为 n1 和 n2 之间的节点的最短路径。Karate 图中节点的下标是从 0 开始

```
import networkx as nx

g = nx.karate_club_graph()
nlist = g.nodes()

idx1 = list(range(0, len(nlist)-2))
slist = []

for n1 in idx1:
    idx2 = list(range(n1+1, len(nlist)-1))
    for n2 in idx2:
        slist.append(len(nx.shortest_path(g,source=n1,target=n2)))

print(max(slist))
```

3. 计算 degree centrality

```
import networkx as nx
```

```
import operator

G = nx.karate_club_graph()
nodes = nx.degree_centrality(G);
sorted_n = sorted(nodes.items(), key=operator.itemgetter(1),
reverse=True)
print(sorted_n)
```

4. 计算 betweenness centrality

```
import networkx as nx
import operator

g = nx.karate_club_graph()
nodes = nx.betweenness_centrality(g)
sorted_n = sorted(nodes.items(), key=operator.itemgetter(1),
reverse=True)

print(sorted_n)
```

5. 计算聚类系数

average_clustering(G, trials=1000)

估计图 G 的平均聚类系数。通过重复 trials 次如下的步骤：

(1) 随机选择一个节点； (2) 随机选择它的两个邻居； (3) 检查这两个节点是否是连通的。

近似的聚类系数是在 trial 次操作中，发现的 triangles 的数量。

```
import networkx as nx
from networkx.algorithms.approximation.clustering_coefficient
import average_clustering
```

```

g = nx.karate_club_graph()
c = average_clustering (g, 50)
print(c)

```

注意：因为是上面的操作是随机选择节点。因此每次计算的结果可能不一样。

6. Pagerank

为 wikipedia 上的一张图，参见图 12-1，计算图中节点的 pagerank 评分

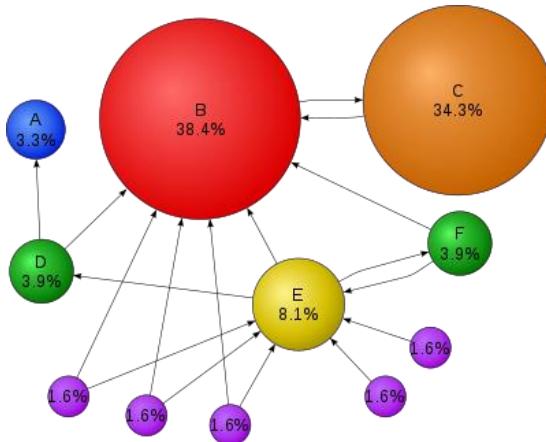


图 12-1 一个有向图

我们建立一张邻接表文件 wikipagerank.adj 来描述该图

B C
C B
D A B
E B D F
F B E
G B E
H B E
I B E
J E
K E

使用方法 pagerank 来计算评分。Alpha 是 damping 系数；max_iter 表示最大迭代次数；tol 表示用于判断收敛的 error tolerance；weight 指示边的数据是否使用了权重，设为 1 时表示没有。

```
import networkx as nx
from operator import itemgetter

G=nx.read_adjlist('wikipagerank.adj')
# G = nx.karate_club_graph()
nodes = nx.pagerank(G, alpha=0.95, max_iter=100, tol=1e-08,
weight=1)
c = nodes.items()
c= sorted (c , key =itemgetter(1), reverse=True)
print(c)
```

计算结果如下：

```
[('E', 0.2615718615847963), ('B', 0.22775272287674098), ('D',
0.10131289486466047), ('H', 0.06651641651245176), ('I',
0.06651641651245176), ('G', 0.06651641651245176), ('F',
0.06651641651245176), ('A', 0.03662786771986229), ('K',
0.03560711584622608), ('J', 0.03560711584622608), ('C',
0.03545475521168023)]
```

7. 计算平均最短路径

使用函数 nx.average_shortest_path_length(g)就可以为图 g 计算平均最短路径

```
import networkx as nx

g = nx.karate_club_graph()
avg = nx.average_shortest_path_length(g)
print(avg)
```

练习：分析 karate 网络是小世界网络吗，是无尺度网络吗？

8. 社区检测

NetworkX 提供了三个社区检测的方法 : Asynchronous label propagation algorithms、k-clique 和 Asynchronous Fluid。

asyn_lpa_communities(G, weight=None)

weight 是描述了边的权重属性。如果为 None 表示边的权重都为 1.

k_clique_communities(G, k)

k-clique 算法是通过在图上发现大小为 k 的完全子图来检测社区。

asyn_fluidc(G, k[, max_iter])

是 Asynchronous Fluid 社区检测算法。K 是用户设定要发现的社区数。

```
import networkx as nx
from networkx.algorithms.community import asyn_fluidc
g = nx.karate_club_graph()
res = list(asyn_fluidc(g, 3))

for r in res:
    for one in r:
        print( '%s ' % str(int(one)+1), end="")
    print("")
```

不过这三个社区检测算法效果都不好。例如在 Karate club 数据集 (图 12-2) 上面的运行结果是

```
33 34 9 15 16 19 21 23 24 27 30 31
1 2 4 5 6 7 8 11 12 13 14 17 18 20 22
3 10 25 26 28 29 32
```

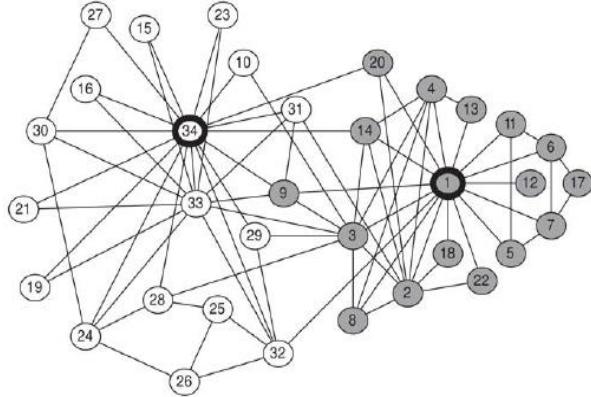


图 12-2 Karate Club 数据集

而 1 5 6 7 11 12 17 应该为第三个社区。

这里给出我在第五节介绍的联合 NMF 和随机游走的社区检测的 Python 程序

myCommFind.py

```
import numpy
import networkx as nx

class MyCommFind:
    sp = 0.5
    step = 6

    def find(self, g, sp, step, numofcomm):
        ''' sp: self-transfer probability
            step: the number of steps
            numofcomm: the number of communities
        ...
        matrix = nx.to_numpy_matrix(g)
        self.sp=sp
        self.step=step

        size = len(g.nodes())
```

```

# -- generate one step transition matrix

mone = numpy.sum(matrix, 1)
ones = numpy.ones((1,len(mone)), dtype=None )
mone = (1-self.sp)*matrix/(mone*ones)

seye = numpy.eye(size, size) * self.sp
mone=seye+mone

# -- transition matrix

sm = mone

for i in range(1,step):
    sm = numpy.dot(sm,mone)

(W,H)=self.mynmf(sm,numofcomm)
r=H.T

#-- obtain dividing result

(row, col)=r.shape

r = numpy.argmax(r, axis=1)
comm={}
for i in range(col):
    comm[i]=[]

for i in range(0, row):
    comm[r[i,0]].append(i)

return comm

def mynmf(self, V,k):
    ''' k is rank. return (W,H)
    '''

    (row,col)=V.shape

```

```

EPS=0.000001

W=numpy.random.rand(row,k)

H=numpy.random.rand(k,col)

d1=2

delta=1

Hk=H

Wk=W

while (abs(delta-d1)>EPS):

    d1=delta

    Hkt=Hk.T

    V1=numpy.dot(V,Hkt)

    W1=numpy.dot(numpy.dot(Wk,Hk),Hkt)+EPS

    vv=V1/W1

    Wk=Wk/(1/vv)

    Wkt=Wk.T

    V2=numpy.dot(Wkt,V);

    W2=numpy.dot(numpy.dot(Wkt,Wk),Hk)+EPS

    vv=V2/W2

    Hk=Hk/(1/vv)

    delta=numpy.linalg.norm(V-numpy.dot(Wk,Hk),ord='fro')

W=Wk;H=Hk

return(W,H)

```

Test.py

```

import networkx as nx

import myCommFind as my

f=my.MyCommFind()

g = nx.karate_club_graph()

```

```

nlist=g.nodes()
p=0.5
step=6
numofcomm=3

comm = f.find(g,p,step,numofcomm)
for a in comm:
    clist = [i+1 for i in comm[a]]
    clist.sort()
    print(str(a)+":"+str(clist))

```

运行 test.py 得到结果：

```

0:[5, 6, 7, 11, 17]
1:[9, 10, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34]
2:[1, 2, 3, 4, 8, 12, 13, 14, 18, 20, 22]

```

9. 计算 Homophily

要度量一个动态网络的 homophily 特性，需要获得该网络的在 t1 和 t2 时刻的两个快照。Homophily 定义为了

$$H = Q^{t2} - Q^{t1}$$

Q 是网络的 assortativity 度量。NetworkX 提供了多种 assortativity 度量。包括在第 10 章第 3 节中介绍的基于类别节点属性和数值节点属性的方法。同时也提供了基于节点的度来计算的方法。

只要我们知道网络的结构,我们总是可以知道一个顶点的度。进一步，我们可以问是否“度”高的节点之间是否有链接，喜欢社交的人是否和其他喜欢社交的人存在联系？ NetworkX 提供了一个方法

`degree_assortativity_coefficient(G)`

可以计算一个网络的 assortativity

```

import networkx as nx

g = nx.karate_club_graph()
r=nx.degree_assortativity_coefficient(g)
print("%3.1f"% r)

```

上面的计算结果是-0.5。对此我的理解是，在 karate 俱乐部中的两个组，每个组有自己的中心人物，中心人物有很高的度。然而在两个组的中心人物之间有很少的连接。我们可以理解这两个组是保持分隔状态，甚至是对立的。

我们下面计算 enron 数据集的 Homophily。Enron 公司在 2004 年爆出丑闻后，公司内部邮件被公布，形成了公开的 enron 数据集。通过邮件之间的通信关系可以构建社会网络。我们分别构建了 2000、2001 和 2002 三年的通信网络。然后分别计算每个网络的 assortativity

```

import networkx as nx

g1=nx.read_edgelist('enron2000.txt');
g2=nx.read_edgelist('enron2001.txt');
g3=nx.read_edgelist('enron2002.txt');

r=nx.degree_assortativity_coefficient(g1)
print("%3.1f"%r)
r=nx.degree_assortativity_coefficient(g2)
print("%3.1f"%r)
r=nx.degree_assortativity_coefficient(g3)
print("%3.1f"%r)

```

其结果是

0.1
-0.0
-0.1

你可以发现 Homophily 现象吗？有什么思考？

第二节：案例 1：微信朋友圈的广告投放

实际上每个互联网公司都有自己的广告投放平台。广告主向这个平台提交自己的广告需求，平台会给广告主圈定一部分潜在用户。一般的平台在圈定用户时会有两种做法。

一种是显性的定位，广告主根据用户的标签直接定位，比如平台给广告主提供用户的标签选项，广告主选择“年龄”、“性别”、“区域”等标签。这个时候的关键是平台进行用户画像，即给用户贴各种标签。如此，平台把广告投放给贴有相应标签的用户。

第二种是基于预测模型的方法。此时广告主需要提供一个用户清单，称为种子用户，平台根据该组种子用户寻找和种子用户相似的潜在用户。如此，如何选取特征和构建预测模型是关键。各广告平台有自己的方法。

这里只介绍第二种方法。微信朋友圈的广告平台定位潜在用户充分考虑到社会网络的社交同配性（assortativity）。其核心是计算两个用户的相似度。具体的技术细节如下：

1. 根据种子用户可以构建社交网络。根据社交网络计算两个用户（节点）的相似度。两个节点的相似度可以根据把每个节点转换成一个向量，然后计算向量的相似度。
2. 使用 Word Embedding 的思想，构建 node embedding。即为网络中的每个节点构建一个节点向量。Word embedding 是深度学习应用在自然语言处理时的一个技术。可以计算词向量（参见我的讲义《深度学习及其在文本挖掘中的应用》）。
3. 在 word embedding 中需要句子作为基本的单位。把句子作为输入，带入学习模型来建立词向量。因此要建立 node 向量，需要从社交网络中产生 node 序列。此时采用了图上的有偏随机游走的方法。如图 12-3 所示。

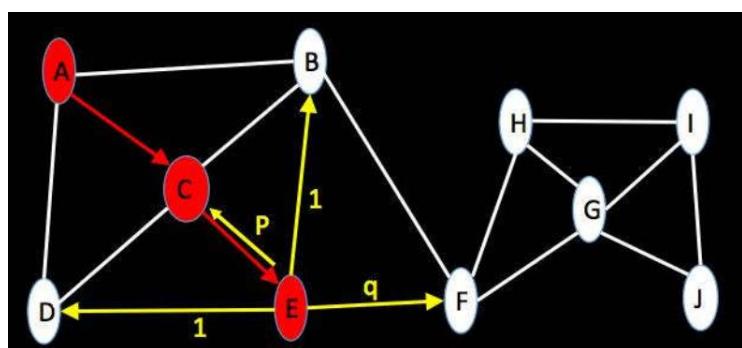


图 12-3 有偏的随机游走

例如，此时随机游走从 A、C 走到节点 E，当考虑如何往下游走时，给走回到节点 C 一个边权重 p ，给走到下一个社区的节点 F 一个权重 q ，给走到本社区的其他节点分配权重 1。因此，从 E 走到哪里，收到了 p ， q 值的影响。

在图上进行有偏随机游走后会得到一个节点的序列，例如[A, C, E, C, B, ...]。把该序列带入 word2vec 算法（产生词向量的算法，详见我的深度学习讲义）。就为图上的每个节点计算了节点向量。

4. 两个节点基于它们的节点向量可以计算节点的相似度。然后，可以为一个种子节点，例如 A，产生一个相似好友列表。

定性解释如下：

在微信的好友中，一些好友是事务性的不能表达亲密度。例如，客户，服务中介，因为它们没有形成社交圈。在产生节点向量的过程中，产生的节点序列因为共现次数少（序列中相邻的两个节点称为共现），那么产生的节点向量，在计算亲密度排序时就会排在后面。而相对来说，亲人，同学，会排在前面。

微信的广告投放平台建立了一个排序的框架。它根据广告主给出的种子用户，找出潜在用户。该框架的输入是种子用户列表，输出是排序的潜在用户。

现在有两个用户集合：种子用户 S ，朋友集合 P 。该框架的工作步骤如下：

1. 通过给出的种子用户和他们的朋友构建四个网络：朋友圈的好友网络；沟通频次网络；共同阅读相同文章、点赞等构成的二分网络；共同关注公众号构成的二分网络。
2. 在上面的的每个网络中都进行 node embedding，建立节点向量。
3. 为每个 $\langle s, p \rangle$ 对，其中 $s \in S, p \in P$ ，计算一个评分。这个评分是用一个 SVR 模型来计算。SVR 模型的输入是四个值，用户 s 和朋友 p 在步骤 1 中的四个网络上的节点向量的相似度。输出则是 s 和 p 在广告上的偏好相似度 $score(s,p)$ 。
4. 最后，为每个朋友 p ，累加它和每个种子用户计算的 $score(s,p)$ ，然后进行排序。

注：在训练 SVR 模型时，训练集中的目标值是用 $Score(s,p)=历史上 s 和 p 的 “共同点击广告数/共同曝光” 来计算。$

补充知识：有偏的随机游走 A biased random walk

我们在第四节讲社区发现时提到的随机游走，相对于有偏随机游走称为 fair random walk。而不是 fair random walk 就是有偏的随机游走

一个无向图上的有偏随机游走的通常描述如下：

假设每个节点都有一个属性 α_i ，游走者从当前节点 j 游走一步到节点 i 的概率是

$$T_{ij}^\alpha = \frac{\alpha_i A_{ij}}{\sum_k \alpha_k A_{kj}}$$

A_{ij} 是节点 j 到 i 的权重。有偏表现在每个节点有不同的 α_i 。不同的网络， α_i 含义不同。如社交网络中一个人的吸引力，也可以是 betweenness centrality，甚至可以理解为节点的内在特性。相对于有偏的随机游走，fair random walk 中，每个节点的 $\alpha_i = 1$

第三节：案例 2：社交媒体上的争议分析

3.1 简介

争议是涉及公共事务时，公众具有的不同的观点和想法。社交媒体上由于用户可以自由的发表观点，因此我们经常可以看到社交媒体上的某个话题可以引发整个舆论界的热烈讨论。考察社交媒体上的争议可以帮助了解公众对某个事件或公共事务的观点。例如，可以分析人们的政见观点，帮助预测大选。

我们收集社交媒体上的 UGC 数据，包括，帖子、评论、回复、转发等，发现它们有两个特点：（1）这些 UGC 数据表现出了社交媒体上用户之间的互动；（2）这些 UGC 数据往往具有情感倾向。表 12-1 是我们收集的一个关于转基因话题的帖子下的评论、回复、转发等数据的情感倾向。

表 12-1 UGC 数据的情感倾向

sentiments	Positive	Neutral	negative	Total
Number	985	1766	2694	5445

“转基因”在中国总是一个很具有争议性的话题。可以看见，争议话题中用户之间的互动频繁，且更倾向于表达负面的情感（大家忙着在网上吵架）。我们也可以看到，在互联网上的争议中，可以将用户划分成三个组：正向、负向和中立。

我们的研究将通过分析社交媒体中的 UGC 数据，为“争议”建立观点分布。即划分参与讨论的用户到三个组。我们的研究建立了一个分析框架 IODO (Investigating Opinion Distribution)，如图 12-4 所示。



图 12-4 IODO 的工作流程

这个分析框架将考察“争议”的问题抽象为，建立情感社会网络，划分“派别”(faction)任务。首先我们给出几个形式定义：

定义 12-1 (观点分布). All participants fall into three groups according to their opinions on a topic in the controversy: the positive, negative and neutral group. A discrete distribution that contains three values in a sequence of pos, neg and neu corresponding to the percentage of the sizes of three groups is called an opinion distribution.

定义 12-2 (情感社会网络 Emotional Social Network). Let $G = (V, E)$ denote a graph, V be a set of nodes, E be a set of edges. For each $e_{ij} \in E$, a 2-tuple of $\langle f, s \rangle$ describes weights of e_{ij} in which f , an integer satisfied $f \geq 1$, denotes the frequency of interaction and s , a real, denotes the strength of sentiments between node i and j .

定义 12-3 (情感派别 Emotional Factor). Let $F = (V_F, E_F)$ denote an emotional factor, which is a sub-graph of an emotional social network $G = (V, E)$ where $V_F \in V$, $E_F \in E$. Edges inside the same emotional factor tend to have positive sentiments whereas those among emotional factors tend to have negative sentiments.

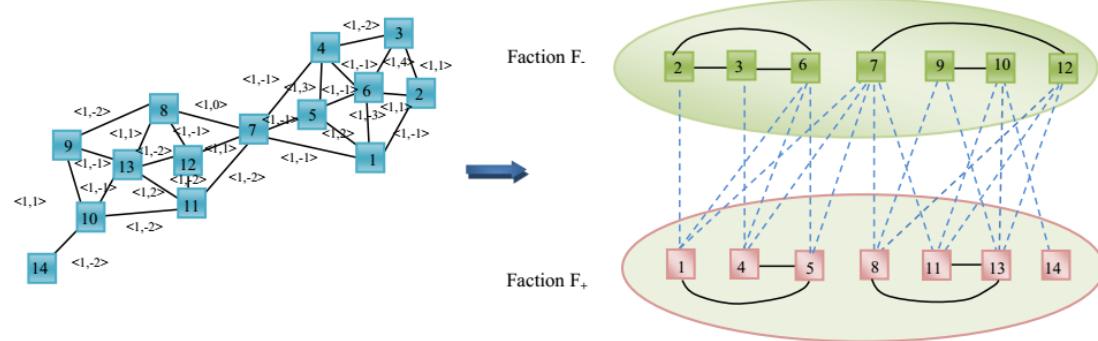


图 12-5 情感社会网络和“派别”的示例

3.2 分析框架

IDO 完成下面的步骤：

(1) 收集 UGC 数据。比如，收集社交媒体上涉及某个话题的帖子下是所有评论、转发、回复等数据。

(2) 建立情感社会网络。所有参与讨论的用户是情感社会网络的顶点，根据 UGC 数据的类型绘制边。计算 UGC 内容的情感倾向，分配一个-2 到 2 的评分，对应着负向情感到正向情感。具体绘制边和分配表的权重的策略见表 12-2。

表 12-2 绘制边的策略

Type of UGC	Edge	Weights
Reply	在发帖人和回复人之间绘制一条边	f 加 “1” 累积交互频率. S 是 UGC 内容的情感评分累积.
Retweet with comment in which there is a @username	绘制两条边。一条连接回复者和发帖人；一条回复者和被@的用户	f 加 1 累积交互频率. 发帖者和回复者的这条边, s 按照 UGC 情感评分累积。另一条边, s 加 1 累加
Retweet with comment in which there is no a @username	在发帖人和回复人之间绘制一条边	f 加 1 累加; s 按 UGC 情感评分累加
Retweet without comment	在发帖人和回复人之间绘制一条边	f 加 1 累积. s 加 1 累积

按照上面的步骤构建了一个情感社会网络 ESN，所有边的 s 权重是均值 $s=s/f$.

(3) 划分派别 faction。将 ESN 划分成两个情感派别：一个是支持者，一个是反对者

(4) 修剪派别。将两个派别中年保持中立或态度比较弱的用户移到中立组

(5) 定量计算“争议性”

步骤 3, 4, 5 将在下面详细展示。

3.2.1 检测 ESN 中的情感派别

社交媒体上涉及到一个话题时，用户通常有正向、负向或者中立的态度。我们的研究首先将用户划分成正向和负向的两个派别。划分派别是一个优化的问题，试图可以将话题的支持者和反对者完美的放到不同的组。模拟退火算法 Simulated Annealing (SA) 是一个解决离散优化问题的工具，SA 已经在很多领域得到成果应用。SA 应用在一个离散优化问题时，在算法的每一次迭代，新、旧两个解的目标函数会被比较。如果新

解更优，则会被接受；如果旧的解更优，则以一定的概率接受新解。这样可以有助于逃离局部最优。接受性能不优的新解的概率取决于“温度”参数。在每一次迭代中，这个文档参数会下降。

SA 算法的主要部分包括：解（solution）的描述，初始解的构造，一个新解的产生，目标函数和迭代终止条件。当我们应用 SA 来检测一个情感社会网络 ESN 的“派别”划分，SA 算法的这些部分描述如下：

- (1) 解的描述。ESN 中一个派别划分就是一个解。使用 R 指示一个解。
- (2) 初始解的构造。我们首先建立两个空的派别 F_+ 和 F_- ，然后将 ESN 的每个节点随机地放到其中一个 faction。然后得到初始解： $R = \{F_+, F_-\}$
- (3) 一个新解的产生。我们随机的选择 ESN 的一个节点，将它从当前的 faction 移动到另一个 faction。这样就获得一个新的解。
- (4) 目标函数。ESN 中每条边有两个权重： f 和 s 。 s 指示两个用户的关系。 $s > 0$ 表示两个用户有正向的关系，即，就一个话题有相同的观点。算法试图将有正向关系的两个用户放到同一个 faction。 $s < 0$ 表示两个用户有负向关系，表示他们有不同的观点。算法试图放两个用户到不同的组。如果我们可以获得一个 faction 的划分，所有 $s > 0$ 的边在同一个 faction 内，所有 $s < 0$ 的边发生在两个 faction 之间，我们可以确定这是最优的解。同时，算法考虑用户之间的较好频率 f 。它对情感有增强的作用。我们开发的一个目标函数如公式(1)

$$Q(R) = \sum_{e \in E_+} p(e) + \sum_{e \in E_-} p(e) - \sum_{e \in E_*} p(e) \quad (1)$$

where

$$p(e) = s(e)(\log_{10}(f(e) + 1)) \quad (2)$$

$s(e)$ 和 $f(e)$ 注释边 e 的权重 s 和权重 f ； E_+ , E_- 和 E_* 分别指示在 F_+ , F_- 和在 F_+ 、 F_- 之间的边的集合。我们称 $p(e)$ ‘relational power’，它描述了两个用户之间的社交影响力。 Q 值越大意味着一个更好的解，即更好的 faction 划分。

- (5) 迭代终止条件。我们的研究中当温度 T_i 小于一个阈值 ε 时迭代终止。

基于 SA 的 Facts 划分算法如下：

Algorithm 1. Partitioning emotional factions in a ESN

Input: An Emotional Social Network G

Output: Factions F_+ and F_-

1. Randomly assign one initial label in $\{+, -\}$ for every node in G
2. Put nodes having same label into the same faction and obtain a partitioning, $R_{\text{old}} = \{F_{\text{old}+}, F_{\text{old}-}\}$
3. Set initial temperature T_i , step temperature T_s , cooling rate α , shock rate φ and ε
4. **While** $T_i > \varepsilon$
5. **For** $j=1 \sim \text{the number of nodes}$
6. Randomly select a node from G , revert its label, and consequently obtain a new partition of faction F_{new}
7. Calculate $\Delta Q = Q(R_{\text{new}}) - Q(R_{\text{old}})$
8. **IF** ($\Delta Q > 0$) accept current partitioning $R_{\text{old}} = R_{\text{new}}$
9. **ELSE** accept current partitioning $R_{\text{old}} = R_{\text{new}}$ with a probability $p = \exp(\Delta Q/T_i)$
10. **END IF**
11. **END FOR**
12. $T_i = \alpha \times T_i$
13. **IF** ($Q(R_{\text{new}}) - Q(R_{\text{old}}) == 0$)
14. $T_s = \varphi \times T_s$, $T_i = T_i + T_s$
15. **END IF**
16. **END WHILE**

为了避免算法陷入局部最优解，当目标函数收敛到一个局部最优解时，我们证据 T_i 的值。这样的操作可以增加算法的收敛时间，但也增加了寻找到更优的解的机会。 T_a 是温度的步长满足 $T_a > 0$ ， φ 是 shock rate 满足 $1 \geq \varphi \geq 0$ 。我们使用 φ 来改变 T_s (i.e., $T_s = \varphi \times T_s$)，然后再改变 T_i (i.e., $T_i = T_i + T_s$)。

3.2.2 建立中立 Faction

算法 1 中，我们从一个 ESN 获得了两个 Faction: F_+ 和 F_- 。他们是一个争议话题中两个观点对立的组。这个算法强制约束每个节点必须放到一个组，即使许多用户其实对话题没有观点。如果我们可以识别这些用户，就可以建立一个中立 Faction。假设存在一离心力试图将 F_+ 和 F_- 中的每个节点从它们所在的 Faction 分离，我们视一个节点对目

标函数 Q 的贡献作为向心力。如果一个节点的向心力不能大于离心力，则该节点分配到中立 Faction。我们给出一个节点的“separation power”的计算公式。

$$S(v) = \beta - \left(\sum_{p \in M_+(v)} p(e_{vp}) - \sum_{p \in M_-(v)} p(e_{vp}) \right) \quad (2)$$

$p(e)$ 是 ‘relational power’; β 是用户设定的离心力，满足 $\beta \geq 0$. $\sum_{p \in M_+(v)} p(e_{vp}) - \sum_{p \in M_-(v)} p(e_{vp})$ 节点 v 对目标函数 Q 的贡献，即向心力。此处 $M_+(v)$ 注释有边链接到 v 且和 v 发生在同一个 faction 的节点集合。 $M_-(v)$ 注释有边链接到 v 且和 v 发生在对立 faction 的节点集合（现在先不考虑中立 faction）。

我们为每个节点计算 separation power，将 separation power 大于等于零的节点移到中立 faction。这个过程就称为修剪 Fraction。图 12-6 描述了一个例子。

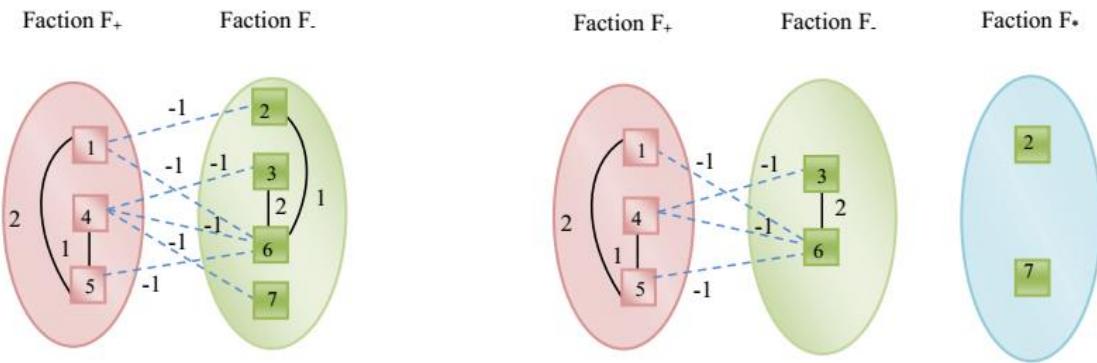


图 12-6 An example of pruning fractions

图 12-6 中节点 2 和 7 对目标函数的贡献是 2 和 1。当设置 $\beta = 2$, separation power 是 $S(2)=0$ 和 $S(7)=1$ 。相应的 fraction 修剪操作得到一个中立 Fraction $F^* = \{2, 7\}$.

3.2.3 定量争议性

要给一个话题的争议性分配一个定量的评分，我们从获取前面获得三个 faction 的大小，即三个值 pos, neg and neu，满足约束

$$\forall v \in \{pos, neg, neu\}, 0 \leq v \leq 1 \text{ and } \sum_{v \in \{pos, neg, neu\}} v = 1 \quad (6)$$

我们使用一个前馈神经网络来计算这个争议性评分。如图 12-7 所示。

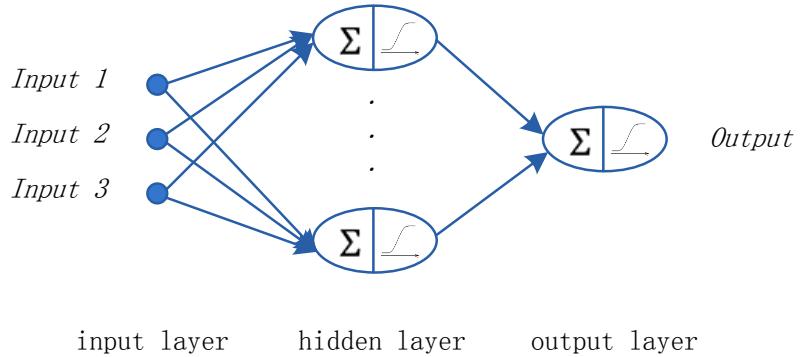


图 12-7 计算 c-Score 的前馈神经网络

3.3 案例研究

我们使用 IODO 来考察中国的电影。IODO 参数设置如下 $T_0=100$, $\alpha = 0.7$, $T_a=9$, and $\varphi = 0.9$. 我们从豆瓣收集电影片评论数据。图 12-8 展示了一个回复网络，它是由豆瓣的一个关于电影“战狼 2”的帖子下的所有回复和评论构建的。其中一个节点指示一个用户，一条边指示两个用户间的回复关系。‘A’注释发布这个帖子的用户。我们可以观察到该帖子下的用户可以分为两个组：一组用户只是对用户 A 进行回复；另一组用户之间有激烈的争论。矩形区域展示了争论最热烈的部分。我们可以观察到，这部中国票房最高的电影，观众的争论很激励。

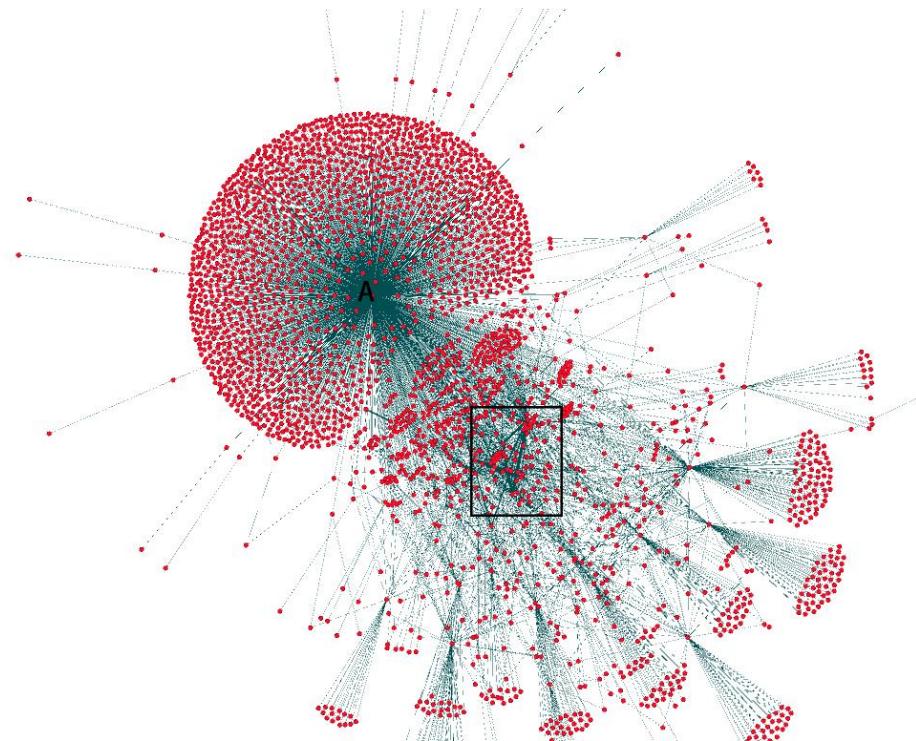


图 12-8 A network built using replies in movies ‘Wolf Warriors 2’

豆瓣上的每部电影都计算了一个 1-10 的评分。更高的评分意味着电影有很好的声誉。我们选择了最高和最低评分的几部电影。表 12-3 列出了使用 IODO 为这些电影计算的争议分布和评分。

表 12-3 The movies and their controversy

No.1	Movies	ratings	Opinion distribution of the movies	c-Score
1	The Shawshank Redemption	9.6	{0.78, 0.02, 0.20}	0.10
2	Farewell My Concubine	9.5	{0.70, 0.03, 0.27}	0.13
3	Zootropolis	9.2	{0.65, 0.04, 0.31}	0.11
4	Tiny Times 1.0	4.7	{0.11, 0.76, 0.13}	0.11
5	A Woman, a Gun and a Noodle Shop	4.6	{0.10, 0.74, 0.16}	0.10

我们也得出一个结论：最高分和最低分的电影有很少的争议性。

进一步我们选择当前中国市场最高票房的电影前 10 部电影中的 6 部，考察他们的争议性。结果列在了表 12-4。

表 12-4 Top list of Box office

No.	Movies	Ratings	Top	Distribution	c-Score
1	Wolf Warriors 2	7.3	1	{0.40, 0.14, 0.46}	0.30
2	Mermaid	6.8	2	{0.54, 0.10, 0.36}	0.22
3	Monster Hunt	6.8	5	{0.36, 0.20, 0.44}	0.51
4	Kung Fu Yoga	5.0	8	{0.46, 0.16, 0.37}	0.36

5	Mojin - The Lost Legend	7.5	9	{0.44, 0.15, 0.41}	0.33
6	Journey to the West: Demon Chapter	5.6	10	{0.40, 0.16, 0.44}	0.38

我们可以得出几个结论：

(1) 和最高评分和最低评分的电影比较，所有高票房的电影都是高争议性的。这种争议性实际上产生了一个好的传播效果。争议性可以获得更多观众的注意力。

(2) 高票房的电影并不意味着高口碑。这里最高评分是 7.5，最低是 5.0。我们得出结论票房和口碑之间没有正相关性。

(3) 在争议性分布中，这些高票房电影的支持者总是比反对者比例大。这意味着正面的口碑胜过了负面口碑。

我们总结一个高票房的电影的成果因素包括：高争议性；正面口碑显著胜过负面口碑。

更多内容请参看 demo: <http://www.biswufe.cn/controversy/index.jsp>

第四节：案例 3：Yelp 上的“社交商务”实证分析

附录 A : LDA 模型的数学推导