**相关文档整理**

# 1 文本分词

参考地址：http://www.lanceyan.com/tech/datadig\_segment\_step1.html

常见的分词算法有：

1、基于词典的分词，需要先预设一个分词词典，比如上面句子切分出来的“假如、上午”这些词先存放在词典，然后把句子切分成单字组合成词语去词典里查找，匹配上了就挑选出来一个词。没有匹配上的就切分成单字。

2、基于统计的分词，需要先获取大量的文本语料库（比如新闻、微博等），然后统计文本里相邻的字同时出现的次数，次数越多就越可能构成一个词。当达到一定次数时就构成了一个词即可形成语料概率库。再对上面句子进行单字切分，把字与字结合后在语料概率库里查找对应的概率，如果概率大于一定值就挑选出来形成一个词。这个是大概描述，实际生产环境中还需要对句子的上下文进行结合才能更准确的分词。

3、基于语义的分词，简而言之就是模拟人类对句子的理解来进行分词。需要先整理出中文语句的句法、语义信息作为知识库，然后结合句子的上下文，对句子进行单字切分后组合成词逐个带入知识库进行识别，识别出来就挑选出一个词。目前还没有特别成熟的基于语义的分词系统。

为了让大家快速的了解分词技术，我们采用第一个方式来做测试：基于词典的分词，这种方式简单暴力可以解决百分之七八十的问题。基于词典的分词大概分为以下几种方式：

1、正向最大匹配，沿着我们看到的句子逐字拆分后组合成词语到词典里去匹配，直到匹配不到词语为止。举个实际的例子：“人民大会堂真雄伟”，我们先拆分为单字“人”去词典里去查找，发现有“人”这个词，继续组合句子里的单字组合“人民”去词典里查找，发现有“人民”这个词，以此类推发现到“人民大会堂”，然后会结合“人民大会堂真”去词典里查找没有找到这个词，第一个词“人民大会堂”查找结束。最终分词的结果为：“人民大会堂/真/雄伟”。

2、逆向最大匹配，这个和上面相反，就是倒着推理。比如“沿海南方向”，我们按正向最大匹配来做就会切分成 “沿海/南方/向”，这样就明显不对。采用逆向最大匹配法则来解决这个问题，从句子的最后取得“方向”这两个字查找词典找到“方向”这个词。再加上“南方向”组成三字组合查找词典没有这个词，查找结束，找到“方向”这个词。以此类推，最终分出“沿/海南/方向”。

3、双向最大匹配，顾名思义就是结合正向最大匹配和逆向最大匹配，最终取其中合理的结果。最早由哈工大王晓龙博士理论化的取最小切分词数，比如“我在中华人民共和国家的院子里看书”，正向最大匹配切分出来为“我/在/中华人民共和国/家/的/院子/里/看书”工8个词语，逆向最大匹配切分出来为“我/在/中华/人民/共/和/国家/的/院子/里/看书”共11个词语。取正向最大匹配切出来的结果就是正确的。但是如果把上面那个例子“沿海南方向”双向切分，都是3个词语，改如何选择？看第4个《最佳匹配法则》。

4、最佳匹配法则，先准备一堆文本语料库、一个词库，统计词库里的每一个词在语料库里出现的次数记录下来。最后按照词频高的优先选出，比如“沿海南方向”，正向切分为：“沿海/南方/向”，逆向切分为：“沿/海南/方向”。其中“海南”的频度最高，优先取出来。剩下“沿”、“方向”也就正常切分了。是不是这就是基于词典分词的最佳方案？比如数学之美中提到的：“把手抬起来” 和 “这扇门的把手”，可以分为“把”、“手”、“把手”，不管怎么分总有一句话的意思不对。后续再介绍如何通过统计的分词处理这些问题。

## 1.1 scws

SCWS 是 Simple Chinese Word Segmentation 的首字母缩写（即：简易中文分词系统）。

官方地址：<http://www.xunsearch.com/scws/index.php>

项目地址：https://github.com/hightman/scws

## 1.2 cppjieba

项目地址：https://github.com/yanyiwu/cppjieba

# 2 观点抽取

利用Stanford Parser（JAVA语言）进行中文观点抽取。

参考：http://blog.sina.com.cn/s/blog\_a16534260100z7hw.html

## 2.1需求问题

所谓的观点抽取就是从文本中获取关于某个特征词的观点词语。特征词在句子结构中通常为主语或者宾语，从词性上看一般为名词或者形容词，而观点词通常为带有情感色彩的形容词或者副词。观点词的抽取在用户对产品评价分析中非常有用。

例如：在句子“卖家 的 服务 态度 不错 ， 快递 也 很 迅速”这个句子中，“服务”和“快递”是两个描述卖家的特征词，而“不错”和“迅速”则是这两个词的观点词。

## 2.2解决方法

1.选择文本数据（数据源，如产品评论文本等）

2.对文本进行断句和分词

3.筛选相关句子（找出含有特征描述对象的句子，直接匹配）

5.语法分析（Stanford Parser)

6.抽取观点词（遍历stanford-parser生成的语法结构树，找到离特征词节点最近的观点词节点，具体参见下面代码）

# 3 文本相似度

simhash地址：https://github.com/yanyiwu/simhash

## 3.1 海量数据相似度计算之simhash和海明距离

参考：http://www.lanceyan.com/tech/arch/simhash\_hamming\_distance\_similarity.html

通过采集系统我们采集了大量文本数据，但是文本中有很多重复数据影响我们对于结果的分析。分析前我们需要对这些数据去除重复，如何选择和设计文本的去重算法？常见的有余弦夹角算法、欧式距离、Jaccard相似度、最长公共子串、编辑距离等。这些算法对于待比较的文本数据不多时还比较好用，如果我们的爬虫每天采集的数据以千万计算，我们如何对于这些海量千万级的数据进行高效的合并去重。最简单的做法是拿着待比较的文本和数据库中所有的文本比较一遍如果是重复的数据就标示为重复。看起来很简单，我们来做个测试，就拿最简单的两个数据使用Apache提供的 Levenshtein for 循环100w次计算这两个数据的相似度。代码结果如下：

|  |
| --- |
| String s1 = "你妈妈喊你回家吃饭哦，回家罗回家罗" ;  String s2 = "你妈妈叫你回家吃饭啦，回家罗回家罗" ;  long t1 = System.currentTimeMillis();  for (int i = 0; i < 1000000; i++) {  int dis = StringUtils .getLevenshteinDistance(s1, s2);  }  long t2 = System.currentTimeMillis();  System. out .println(" 耗费时间： " + (t2 - t1) + " ms "); |

耗费时间： 4266 ms

大跌眼镜，居然计算耗费4秒。假设我们一天需要比较100w次，光是比较100w次的数据是否重复就需要4s，就算4s一个文档，单线程一分钟才处理15个文档，一个小时才900个，一天也才21600个文档，这个数字和一天100w相差甚远，需要多少机器和资源才能解决。

为此我们需要一种应对于海量数据场景的去重方案，经过研究发现有种叫 local sensitive hash 局部敏感哈希 的东西，据说这玩意可以把文档降维到hash数字，数字两两计算运算量要小很多。查找很多文档后看到google对于网页去重使用的是simhash，他们每天需要处理的文档在亿级别，大大超过了我们现在文档的水平。既然老大哥也有类似的应用，我们也赶紧尝试下。simhash是由 Charikar 在2002年提出来的，参考 《Similarity estimation techniques from rounding algorithms》 。 介绍下这个算法主要原理，为了便于理解尽量不使用数学公式，分为这几步：

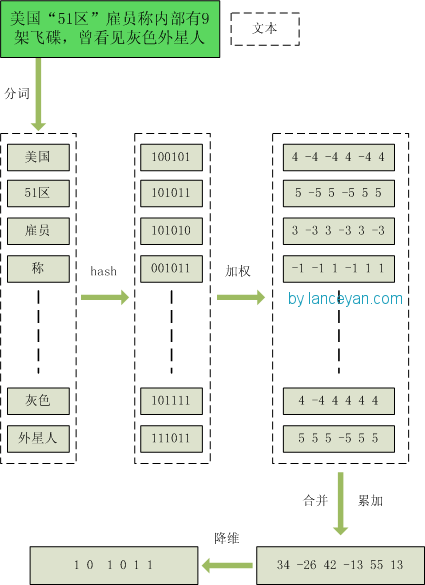
1、分词，把需要判断文本分词形成这个文章的特征单词。最后形成去掉噪音词的单词序列并为每个词加上权重，我们假设权重分为5个级别（1~5）。比如：“ 美国“51区”雇员称内部有9架飞碟，曾看见灰色外星人 ” ==> 分词后为 “ 美国（4） 51区（5） 雇员（3） 称（1） 内部（2） 有（1） 9架（3） 飞碟（5） 曾（1） 看见（3） 灰色（4） 外星人（5）”，括号里是代表单词在整个句子里重要程度，数字越大越重要。

2、hash，通过hash算法把每个词变成hash值，比如“美国”通过hash算法计算为 100101,“51区”通过hash算法计算为 101011。这样我们的字符串就变成了一串串数字，还记得文章开头说过的吗，要把文章变为数字计算才能提高相似度计算性能，现在是降维过程进行时。

3、加权，通过 2步骤的hash生成结果，需要按照单词的权重形成加权数字串，比如“美国”的hash值为“100101”，通过加权计算为“4 -4 -4 4 -4 4”；“51区”的hash值为“101011”，通过加权计算为 “ 5 -5 5 -5 5 5”。

4、合并，把上面各个单词算出来的序列值累加，变成只有一个序列串。比如 “美国”的 “4 -4 -4 4 -4 4”，“51区”的 “ 5 -5 5 -5 5 5”， 把每一位进行累加， “4+5 -4+-5 -4+5 4+-5 -4+5 4+5” ==》 “9 -9 1 -1 1 9”。这里作为示例只算了两个单词的，真实计算需要把所有单词的序列串累加。

5、降维，把4步算出来的 “9 -9 1 -1 1 9” 变成 0 1 串，形成我们最终的simhash签名。 如果每一位大于0 记为 1，小于0 记为 0。最后算出结果为：“1 0 1 0 1 1”。



大家可能会有疑问，经过这么多步骤搞这么麻烦，不就是为了得到个 0 1 字符串吗？我直接把这个文本作为字符串输入，用hash函数生成 0 1 值更简单。其实不是这样的，传统hash函数解决的是生成唯一值，比如 md5、hashmap等。md5是用于生成唯一签名串，只要稍微多加一个字符md5的两个数字看起来相差甚远；hashmap也是用于键值对查找，便于快速插入和查找的数据结构。不过我们主要解决的是文本相似度计算，要比较的是两个文章是否相识，当然我们降维生成了hashcode也是用于这个目的。看到这里估计大家就明白了，我们使用的simhash就算把文章中的字符串变成 01 串也还是可以用于计算相似度的，而传统的hashcode却不行。我们可以来做个测试，两个相差只有一个字符的文本串，“你妈妈喊你回家吃饭哦，回家罗回家罗” 和 “你妈妈叫你回家吃饭啦，回家罗回家罗”。

通过simhash计算结果为：

1000010010101101**1**11111100000101011010001001111100001**0**0101**1**001011

1000010010101101**0**11111100000101011010001001111100001**1**0101**0**001011

通过 hashcode计算为：

1111111111111111111111111111111110001000001100110100111011011110

1010010001111111110010110011101

大家可以看得出来，相似的文本只有部分 01 串变化了，而普通的hashcode却不能做到，这个就是局部敏感哈希的魅力。目前Broder提出的shingling算法和Charikar的simhash算法应该算是业界公认比较好的算法。在simhash的发明人Charikar的论文中并没有给出具体的simhash算法和证明，“量子图灵”得出的证明simhash是由随机超平面hash算法演变而来的。

现在通过这样的转换，我们把库里的文本都转换为simhash 代码，并转换为long类型存储，空间大大减少。现在我们虽然解决了空间，但是如何计算两个simhash的相似度呢？难道是比较两个simhash的01有多少个不同吗？对的，其实也就是这样，我们通过海明距离（Hamming distance）就可以计算出两个simhash到底相似不相似。两个simhash对应二进制（01串）取值不同的数量称为这两个simhash的海明距离。举例如下： 10101 和 00110 从第一位开始依次有第一位、第四、第五位不同，则海明距离为3。对于二进制字符串的a和b，海明距离为等于在a XOR b运算结果中1的个数（普遍算法）。

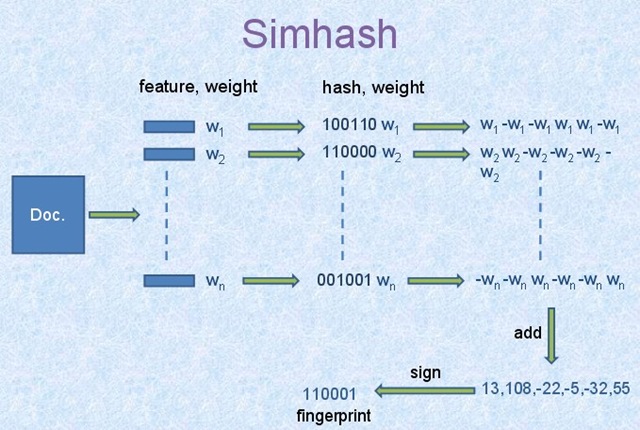
## 3.2 simhash算法原理及实现

参考：http://yanyiwu.com/work/2014/01/30/simhash-shi-xian-xiang-jie.html

simhash是google用来处理海量文本去重的算法。 google出品，你懂的。 simhash最牛逼的一点就是将一个文档，最后转换成一个64位的字节，暂且称之为特征字，然后判断重复只需要判断他们的特征字的距离是不是<n（根据经验这个n一般取值为3），就可以判断两个文档是否相似。

### 3.2.1原理

**1、simhash值的生成图解如下**



大概花三分钟看懂这个图就差不多怎么实现这个simhash算法了。特别简单。谷歌出品嘛，简单实用。

算法过程大概如下：

（1）将Doc进行关键词抽取(其中包括分词和计算权重)，抽取出n个(关键词，权重)对， 即图中的(feature, weight)们。 记为 feature\_weight\_pairs = [fw1, fw2 … fwn]，其中 fwn = (feature\_n,weight\_n`)。

（2）hash\_weight\_pairs = [ (hash(feature), weight) for feature, weight in feature\_weight\_pairs ] 生成图中的(hash,weight)们, 此时假设hash生成的位数bits\_count = 6（如图）;

（3）然后对 hash\_weight\_pairs 进行位的纵向累加，如果该位是1，则+weight,如果是0，则-weight，最后生成bits\_count个数字，如图所示是[13, 108, -22, -5, -32, 55], 这里产生的值和hash函数所用的算法相关。

（4）[13,108,-22,-5,-32,55] -> 110001这个就很简单啦，正1负0。

到此，如何从一个doc到一个simhash值的过程已经讲明白了。 但是还有一个重要的部分没讲，

**2、simhash值的海明距离计算**

二进制串A 和 二进制串B 的海明距离 就是 A xor B 后二进制中1的个数。

举例如下：

|  |
| --- |
| A = 100111;  B = 101010;  hamming\_distance(A, B) = count\_1(A xor B) = count\_1(001101) = 3; |

当我们算出所有doc的simhash值之后，需要计算doc A和doc B之间是否相似的条件是：

**A和B的海明距离是否小于等于n，这个n值根据经验一般取值为3,**

simhash本质上是局部敏感性的hash，和md5之类的不一样。 正因为它的局部敏感性，所以我们可以使用海明距离来衡量simhash值的相似度。

**3、高效计算二进制序列中1的个数**

|  |
| --- |
| /\* src/Simhasher.hpp \*/  bool isEqual(uint64\_t lhs, uint64\_t rhs, unsigned short n = 3)  {  unsigned short cnt = 0;  lhs ^= rhs;  while(lhs && cnt <= n)  {  lhs &= lhs - 1;  cnt++;  }  if(cnt <= n)  {  return true;  }  return false;  } |

由上式这个函数来计算的话，时间复杂度是 O(n); 这里的n默认取值为3。由此可见还是蛮高效的。

## 3.3 fingerprint的Hamming Distance

参考地址：http://leoncom.org/?p=650607

**问题：**一个80亿的64-bit指纹组成的集合Q，对于一个给定64-bit的指纹F，如何在a few millionseconds中找到Q中和f至多只有k(k=3)位差别的指纹。

**思想：**

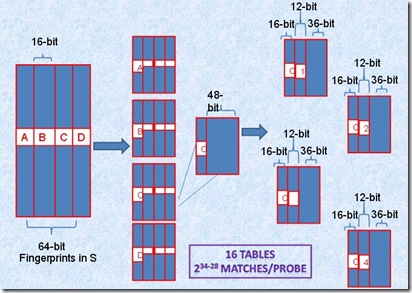
1. 对于一个具有2^d个记录的集合，只需要考虑d-bit hash。

2. 选取一个d’使得|d’-d|十分小，因此如果两fingerprint在d’-bits上都相同，那么在d-bits也很可能相同。然后在这些d-bit match的结果中寻找整个f-bit的Hamming Distance小于k的fingerprint。 简单的说，就是利用fingerprint少量特征位数比较从而首先缩小范围，然后再去确定是否差异小于k个bit。

**算法：**

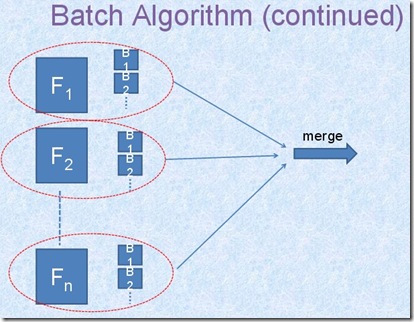
|  |
| --- |
| 1. 首先对于集合Q构建多个表T1，T2…Tt，每一个表都是采用对应的置换函数π(i)将64-bit的fingerprint中的某p(i)位序列置换换到整个序列的最前面。即每个表存储都是整个Q的fingerprint的复制置换。  2.对于给定的F，在每个Ti中进行匹配，寻找所有前pi位与F经过π(i)置换后的前pi位相同的fingerprint。  3.对于所有在上一步中匹配到的置换后的fingerprint，计算其是否与π(i)(F)至多有k-bit不同。 |

算法的重点在于对于集合Q的分表以及每个表所对应的置换函数，假设对于64-bit的fingerprint，k=3，存储16个table，划分参考下图:



将64-bit按照16位划分为4个区间，每个区间剩余的48-bit再按照每个12-bit划分为4个区间，因此总共16个table并行查找，即使三个不同的k-bit落在A、B、C、D中三个不同的区块，此划分方法也不会导致遗漏。

以上方法是对于online的query，即一个给定的F在集合中查找相似的fingerprint。如果爬虫每天爬取了100w个网页，快速的查找这些新抓取的网页是否在原集合中有Near-duplication，对于这种batch-query的情况，Map-Reduce就发挥它的威力了。



不同的是，在batch-query的处理中，是对待查集合B(1M个fingerprint)进行复制置换构建Table而非8B的目标集合，而在每一个chunkserver上对Fi(F为整个8B的fingerprint)在整个Table(B)中进行探测，每一个chunkserver上的的该Map过程输出该Fi中与整个B的near-duplicates，Reduces过程则将所有的结果收集、去重、然后输出为一个sorted file。

**Haffman编码压缩**

上述的查询过程，特别是针对online-version的算法，可以看出需要对8B的fingerprint进行多表复制和构建，其占据的容量是非常大的，不过由于构建的每一个置换Table都是sorted的，因此可以利用每一个fingerprint与其前一个的开始不同的bit-position h(h∈[0,f-1]) 来进行数据压缩，即如果前一个编码是11011011，而自身是11011001，则后一个可以编码为(6)1，即h=6，其中6表示从第6位(从0开始编号)开始和上一个fingerprint不相同(上一个为1，这个必然为0)，然后再保存不相同位置右侧的编码，依次生成整个table。

Google首先计算整个排序的fingerprint表中h的分布情况，即不同的h出现次数，依据此对[0,f-1]上出现的h建立Haffman code，再根据上述规则生成table（例如上面的6就表示成对应的Haffman code）。其中table分为多个block，每一个block中的第一个fingerprint保存原数据，后面的依次按照编码生成。

将每一个block中所对应的最后一个fingerprint保存在内存中，因此在比对的时候就可以直接根据内存中的fingerprint来确定是哪一个block需要被decompress进行比较。

8B个64-bit的fingerprint原占据空间大约为64GB，利用上述Haffman code压缩后几乎会减少一般，而内存中又只对每一个block保存了一个fingerprint。

# 4 C语言正则表达式详解 regcomp() regexec() regfree()详解

标准的C和C++都不支持正则表达式，但有一些函数库可以辅助C/C++程序员完成这一功能，其中最著名的当数Philip Hazel的Perl-Compatible Regular Expression库，许多Linux发行版本都带有这个函数库。

C语言处理正则表达式常用的函数有regcomp()、regexec()、regfree()和regerror()，一般分为三个步骤，如下所示：

C语言中使用正则表达式一般分为三步：

编译正则表达式 regcomp()

匹配正则表达式 regexec()

释放正则表达式 regfree()

下边是对三个函数的详细解释

1、int regcomp (regex\_t \*compiled, const char \*pattern, int cflags)

这个函数把指定的正则表达式pattern编译成一种特定的数据格式compiled，这样可以使匹配更有效。函数regexec 会使用这个数据在目标文本串中进行模式匹配。执行成功返回０。

参数说明：

①regex\_t 是一个结构体数据类型，用来存放编译后的正则表达式，它的成员re\_nsub 用来存储正则表达式中的子正则表达式的个数，子正则表达式就是用圆括号包起来的部分表达式。

②pattern 是指向我们写好的正则表达式的指针。

③cflags 有如下4个值或者是它们或运算(|)后的值：

REG\_EXTENDED 以功能更加强大的扩展正则表达式的方式进行匹配。

REG\_ICASE 匹配字母时忽略大小写。

REG\_NOSUB 不用存储匹配后的结果。

REG\_NEWLINE 识别换行符，这样'$'就可以从行尾开始匹配，'^'就可以从行的开头开始匹配。

2. int regexec (regex\_t \*compiled, char \*string, size\_t nmatch, regmatch\_t matchptr [], int eflags)

当我们编译好正则表达式后，就可以用regexec 匹配我们的目标文本串了，如果在编译正则表达式的时候没有指定cflags的参数为REG\_NEWLINE，则默认情况下是忽略换行符的，也就是把整个文本串当作一个字符串处理。执行成功返回０。

regmatch\_t 是一个结构体数据类型，在regex.h中定义：

typedef struct

{

regoff\_t rm\_so;

regoff\_t rm\_eo;

} regmatch\_t;

成员rm\_so 存放匹配文本串在目标串中的开始位置，rm\_eo 存放结束位置。通常我们以数组的形式定义一组这样的结构。因为往往我们的正则表达式中还包含子正则表达式。数组0单元存放主正则表达式位置，后边的单元依次存放子正则表达式位置。

参数说明：

①compiled 是已经用regcomp函数编译好的正则表达式。

②string 是目标文本串。

③nmatch 是regmatch\_t结构体数组的长度。

④matchptr regmatch\_t类型的结构体数组，存放匹配文本串的位置信息。

⑤eflags 有两个值

REG\_NOTBOL 按我的理解是如果指定了这个值，那么'^'就不会从我们的目标串开始匹配。总之我到现在还不是很明白这个参数的意义；

REG\_NOTEOL 和上边那个作用差不多，不过这个指定结束end of line。

3. void regfree (regex\_t \*compiled)

当我们使用完编译好的正则表达式后，或者要重新编译其他正则表达式的时候，我们可以用这个函数清空compiled指向的regex\_t结构体的内容，请记住，如果是重新编译的话，一定要先清空regex\_t结构体。

4. size\_t regerror (int errcode, regex\_t \*compiled, char \*buffer, size\_t length)

当执行regcomp 或者regexec 产生错误的时候，就可以调用这个函数而返回一个包含错误信息的字符串。

参数说明：

①errcode 是由regcomp 和 regexec 函数返回的错误代号。

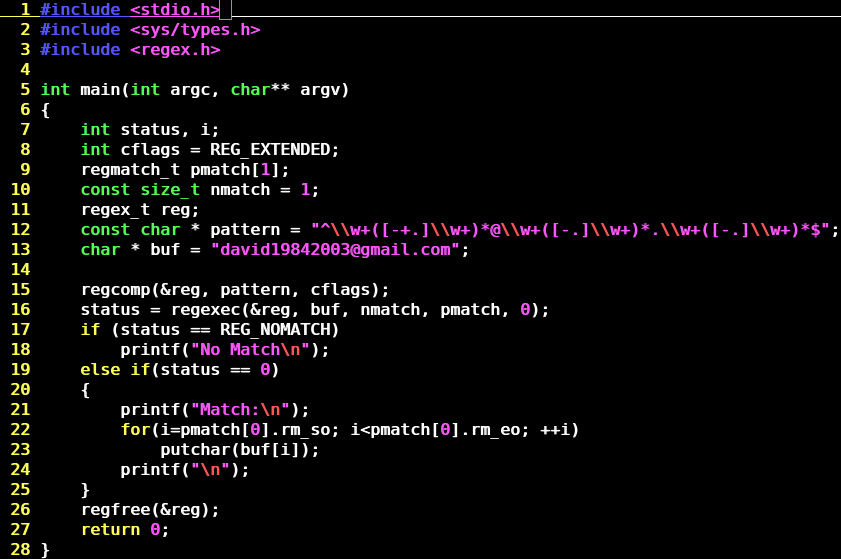
②compiled 是已经用regcomp函数编译好的正则表达式，这个值可以为NULL。

③buffer 指向用来存放错误信息的字符串的内存空间。

④length 指明buffer的长度，如果这个错误信息的长度大于这个值，则regerror 函数会自动截断超出的字符串，但他仍然会返回完整的字符串的长度。所以我们可以用如下的方法先得到错误字符串的长度。

size\_t length = regerror (errcode, compiled, NULL, 0);

下边是一个匹配Email例子，按照上面的三步就可以。



# 5 网络舆情分析中的文本聚类算法

网络舆情分析和预警系统通过对互联网中的各类信息，包括各大论坛、百度贴吧、微博等进行24小时监控，实时采集话题性强、时效性新的各类内容和消息。系统架构主要包括舆情信息采集、信息预处理、信息分析、舆情信息上报四个模块。舆情分析模块主要对处理后的网络文本数据进行分析、挖掘，以实现热点话题和敏感信息的识别和跟踪。文本聚类作为一种无监督的机器学习方法，由于不需要进行训练，并且不需要预先对网络文本手工标注类别，因此具有较高的灵活性和自动化处理能力，成为网络舆情分析系统中组织文本信息、挖掘热点话题的主要技术手段。

进行文本聚类之前，首要问题是对文本内容建立文本信息特征，即使用特定形式的特征向量（T1， V1， T2， V2， …， Tm， Vm）来表示文档，其中Ti表示特征词，由分词后获取，Vi表示第i个词在文档中的权重，m表示特征词的个数。

(1)确定信息特征的常用技术是：先对文本信息进行词性标注、语义标注等预处理，构建统计词典，然后对文本进行分词处理和特征词权重的确定。

(2)计算权重的方法包括布尔权重、词频权重和TFI-DF权重三种。布尔权重是最简单的表示方法，若第i个特征词在本篇文档中出现，则其权重为1，否则为0；词频权重将第i个特征词在本篇文档中的出现概率作为对应的权值；**TFI-DF权值**由两部分组成，一部分是TF，表示一个词在本篇文本中出现的次数，即词频。另一部分IDF表示所有文本中有多少篇包含了这个词，即逆向文档频率，通常由公式IDFi=log（（1+|D|）/|Di|）计算得到，其中|D|表示文本总数，|Di|表示包含第i个关键词的文本数量。 TFI-DF的权值就是由TF和IDF相乘得到，被证明是最有效的确定特征词权重的方法。   
　　由于在建立文本信息特征的过程中，许多关键词构成了海量特征词集合，因此导致了文档特征向量维数的增加，在影响计算效率的同时，也为稀疏文本向量的区分和比较带来困难。因此，在对文本进行聚类处理之前， 应对文本信息特征向量进行缩减。通常的方法是对所有特征词的权重排序，选取预定数目的最佳特征词作为结果的特征子集。特征词的个数以及权值评价标准需要根据具体问题来分析决定。第二种常用方法是由网络信息的特点决定的，网络信息一般带有标题，并且在标题中总结概括了正文的主要内容，因此也可将标题内容表示为文本信息特征。此外，在对正文统计特征词时，可以只针对具有实际意义的名词和动词，这既减少了特征词数量，也保留了关键信息。将标题向量的相似度和正文向量的相似度进行加权求和就得到了两篇文本的相似度。   
　　完成以上步骤后，就将文档表示成向量空间中的一个点，然后通过计算点之间的距离来对文本进行聚类。相似度是定义聚类算法的基础，它主要用来比较衡量两个对象的相似程度。在网络舆情分析中，两个文本特征对象的相似度量标准对大多数聚类算法都是必不可少的。计算相似度的方法有多种，在文本向量计算中一般使用**余弦相似度公式**，而在几何中常用欧几里得距离。**余弦相似度不会放大数据对象重要部分的作用**，而欧几里得距离的度量方法则在一定程度上放大了较大元素误差在距离测度中的作用。目前，**文本聚类的方法大致可以分为两种类型：层次聚类法和平面划分法。**   
　　层次聚类法将所有文本的特征向量组织成一棵聚类的树，有自底向上的凝聚型和自顶向下的分裂型两种。凝聚型开始将所有文本向量作为一个单独的对象，然后不断相继合并相似的对象和类簇，直到到达终止条件或者合并为一个类簇。分裂型的聚类方法过程与之相反，开始将所有文本向量置于同一个类簇中，然后通过不断迭代将类簇分解为更小的类簇，直到到达终止条件或者每个向量在单独的一个类簇中。常用的层次凝聚法包括Single-link和 group-average方法。   
　　平面划分法中有两个经典算法：k平均值算法和k中心点算法。K平均值算法首先把所有文本特征随机地分配到 k个非空的类中，然后根据类中所有的文本特征计算每个类的平均值，并用该平均值表示相应的类，根据每个对象到各个类中心的距离，将其重新分配到与它最接近 的类中。重复执行计算和分配步骤，直到不再有新的分配发生。K中心点算法随机选择k个文本特征对象作为初始的类中心点，然后对每一组非中心对象j和中心对 象i，计算i被j替代的总代价，若代价<0，i被替换，然后根据所有非中心点到中心点的距离重新分配非中心点的所属类。重复执行替换和分配的过程直 到不再有新的分配发生。   
　　针对网络舆情分析中用到的聚类算法，有很多研究者提出了改进和优化方案。为了更好地发现热点话题，提出了两层聚类算 法，其设计思路是先确定每日热点簇，然后再利用增量聚类算法发现热点事件。也有研究结合K-means聚类算法和基于密度的聚类算法的优点，改进了K- means算法中随机选取初始聚类中心的问题。此外，基于主题的聚类方法是检测热点新闻的一个研究思路，有研究提出对新闻主题或新闻事件等具有较强表达能 力的文本域进行加权处理以突出该主题或事件，然后再对无序的网络舆情信息进行聚类处理，达到热点话题探测的目的。应用LDA、LSI等主题模型进行新闻聚 类是当前发现新闻主题的重要研究方向。