

目錄

Introduction	1.1
第1章 文本数据分析	1.2
1.1 什么是文本数据分析	1.2.1
1.2 文本数据分析相关概念	1.2.2
1.3 常用文本数据分析工具	1.2.3
1.4 小结	1.2.4
1.5 深度阅读材料	1.2.5
1.6 练习	1.2.6
第2章 Python 入门	1.3
2.1 Python简介	1.3.1
2.2 相关工具/包的安装说明	1.3.2
2.3 Jupyter Notebook 快速入门	1.3.3
2.4 Python 基础	1.3.4
2.5 小结	1.3.5
2.6 深度阅读材料	1.3.6
2.7 练习	1.3.7
第3章 文本数据来源	1.4
3.1 公开数据源	1.4.1
3.2 自有数据	1.4.2
3.3 网络抓取数据	1.4.3
3.4 文本数据读取	1.4.4
3.5 小结	1.4.5
3.6 深度阅读	1.4.6
3.7 练习	1.4.7
第4章 分词与词性标注	1.5
4.1 中文分词及词性标注	1.5.1
4.2 英文分词及词性标注	1.5.2
4.3 小结.md	1.5.3
4.4 深入阅读材料.md	1.5.4
4.5 练习.md	1.5.5

第5章 文本数据结构化处理	1.6
5.1 文档-词项矩阵	1.6.1
5.2 词频-逆向文档频率	1.6.2
5.3 词向量	1.6.3
5.4 文本数据结构化处理实例	1.6.4
5.5 小结	1.6.5
5.6 深度阅读材料	1.6.6
5.7 练习	1.6.7
第6章 文本分类、聚类	1.7
6.1 文本分类算法简介	1.7.1
6.2 文本聚类算法简介	1.7.2
6.3 常用文本分类工具	1.7.3
6.4 常用文本聚类工具	1.7.4
6.5 文本分类、聚类分析实例	1.7.5
6.6 小结	1.7.6
6.7 深度阅读材料	1.7.7
6.8 练习	1.7.8
第7章 主题模型	1.8
7.1 LSI	1.8.1
7.2 LDA	1.8.2
7.3 主题模型实例	1.8.3
7.4 小结	1.8.4
7.5 深度阅读材料	1.8.5
7.6 练习	1.8.6
第8章 文本数据可视化	1.9
8.1 基于词频的文本内容可视化	1.9.1
8.2 文本关系可视化	1.9.2
8.3 基于时间信息的文本数据可视化	1.9.3
8.4 小结	1.9.4
8.5 深度阅读材料	1.9.5
8.6 练习	1.9.6
TODo	1.10

Python 文本数据分析初学指南

Begining Text Mining with Python

项目代号: begining-text-mining-with-python

效果预览

<https://datartisan.gitbooks.io/begining-text-mining-with-python/content/>

此版本库书写约定

- 遵循 gitbook 的结构约定
- README.md 文件是此版本库说明文件，不属于图书内容
- 图书内容使用 markdown 语法格式
- 图书目录写到 SUMMARY.md 文件中
- 其他内容写到对应的 .md 文件中
- 图片统一放置在根目录下的 `图片资源` 文件夹内，正文内使用相对路径对图片进行引用

工作流程

- 参数书写的人首先 fork 本版本库到各自的账号中
- 在各自的账号中进行修改
- 修改后发送 pull request 到此库，审核合并

稿件导出

导出 word 格式稿件:

- 使用 gitbook 导出 epub 格式
- 使用工具转换 epub 为 word 格式

第1章 文本数据分析

完成一个完整的文本数据分析过程，要求满足以下几个要素条件：一是明确的待分析的文本数据，二是要掌握自然语言处理、文本数据分析相关理论知识，三是要学会使用分析工具进行实现。在在进行复杂的文本数据分析之前，本章将重点向读者介绍自然语言处理、文本数据分析相关理论知识，包括文本数据分析的相关思想、概念，文本数据分析的意义，以及常用的文本数据分析工具等，这些概念以及工具在后续章节都会有所涉及，在此一并明晰，后续不多做说明。本章的目的在于：界定后续章节将涉及的相关概念，统一介绍本书用到的分析工具。通过阅读本章，文本数据分析的初学者将对文本数据分析有更加清晰的认识，对相关的分析流程及工具有系统的认知，同时本章也将为读者后续章节的阅读、理解奠定基础。本章具体内容安排如下：1.1节首先介绍什么是文本数据及文本数据的特点，文本数据分析的含义、基本内容，通过列举文本数据分析在不同领域的应用，来展示文本数据分析的价值所在，最后描述了文本数据分析在未来一段时间内的发展趋势；1.2节重点说明文本数据分析过程中涉及的相关操作，包括分词、词性标注、命名实体识别、句法分析、词向量等，为后续章节的学习构建基础；最后，1.3节将一一介绍常用的文本数据分析工具，包括分析环境、常用自然语言处理工具、科学计算库、机器学习库、词典资源等等。

1.1 什么是文本数据分析

1.1.1 文本数据

直观来讲，“文本”（text）即文字、话语，是语言的书面表现形式，可以是句子、段落或者篇章等，其存储方式和常见的数字不同。一般根据存储方式的不同，可以将数据划分为结构化数据、非结构化数据和半结构化数据。所谓结构化数据，指的就是可以用二维表组织、分析处理过程较为简单的信息，可以将这种结构化的二维表组织方式理解为一个文件夹，文件夹里的每一个文件都被明确标记并很容易被识别，数字、符号等属于结构化数据；与结构化数据相对的即为非结构化数据，非结构化数据是多种信息的无结构混合，通常无法直接知道其内部结构，只有经过识别、有条理的存储分析后才能体现其价值，图片、声音、视频等属于非结构化数据，理想条件下，所有的非结构化数据都可以结构化处理，但是实际上，有些类型的非结构化数据很难结构化处理，比如不包含结构化字段的纯文本（如文章摘要）就很难切分和分类；介于结构化数据和非结构化数据两者之间的数据称为半结构化数据，大多数文本，既包含标题、作者、分类等结构字段，又包含非结构化的文字内容，这类文本均属于半结构化数据，即半结构化文本数据。

1.1.2 文本数据特点

1.1.2.1 半结构化

正如上文 1.1.1 所提到的，大多数文本数据属于半结构化数据，例如一篇我们常见的网络新闻，其标题、作者、分类、来源等信息通常都会以某种特定的格式标注出来，这样的信息可以称为是结构化内容，而新闻的主体部分譬如摘要和正文等则是由连续的文字与标点组成的纯粹文本，即非结构化内容。用于处理大量数据的计算机并非人类，本身并不具备解读非结构化自然语言的能力，因此，文本数据半结构化的特点阻碍了传统数据分析方法对其的直接应用，对于文本数据的结构化处理的有关方法以及文本数据的表示形式也成为近年来学界关注的一大焦点。

1.1.2.2 数据量大

一般的文本库至少包含数千个文本样本，此外，随着网络信息技术的飞速发展，数据获取、传输、存储的速度大幅增长，人们越来越多的依赖网络进行信息沟通，比如在网上发表意见、讨论问题、交流情感等，使得文本数据呈现出高速膨胀的态势。鉴于文本数据基数大与高增长的特征，需要采用特殊的方法对这些数据进行分析处理以提高分析效率。

1.1.2.3 高维稀疏性

一般文本数据结构化处理后得到的文本向量都会面临维度过高和稀疏的问题，维数一般都高达数千甚至上万维，如果不进行处理，则会导致文本挖掘算法计算量大，资源消耗高，同时严重影响相关挖掘算法的准确性，所以有必要进行特征筛选等降维处理。

1.1.2.4 蕴含语义、情感

文本是语言的书面表现形式，其内在蕴含了不同语言环境下复杂的语义关系，如一词多义、起承转合、时间关系等。此外，文本内容由特定的人编写，因此，除传递所表达的基本信息外，不可避免地会隐含着表述者的态度或情感等。

1.1.3 文本数据分析

1.1.3.1 文本数据分析含义

Seth Grimes 曾指出“80% 的商业信息来自非结构化数据，主要是文本数据”，这一表述可能夸大了文本数据在商业数据中的占比，但是文本数据的蕴含的信息价值毋庸置疑。文本数据数据量大的特点，使得人工信息处理变得效率低下，必须借助计算机来完成相关工作，但是文本数据蕴含着复杂的语义关系和情感倾向，计算机无法直接识别、处理，所以需要将文本数据进行相应的转化处理。从狭义的角度来讲，将文本数据转化为计算机可以识别处理的结构化数据的科学抽象过程即为文本数据分析过程，其首要目标就是利用自然语言处理和分析方法将“文本”转换为“数据”，具体会涉及到词频分布研究、模式识别、关联分析、信息提取、可视化和预测分析等等，通过文本数据分析，可以初步推断文本的主要含义和文本提供者的意图。从广义的角度来讲，文本数据分析既涵盖前文提到的文本数据结构化的前置处理，又

囊括抽取文本数据蕴含的深层次、高质量信息等进一步的研究内容，即文本数据挖掘（text data mining），也常被表述为文本挖掘（text mining），传统的文本数据挖掘包括文本分类、文本聚类、命名实体识别、情感分析、建立实体关系模型等。本书内容从广义文本数据分析出发，除介绍文本数据结构化处理外，也涉及文本聚类、命名实体识别等文本挖掘内容。

1.1.3.2 文本数据分析基本内容

（1）获取文本数据：即获取文本数据分析的对象，属于文本数据分析的准备阶段，可以从公开数据源下载，或者利用自有数据集，或者按照分析需求从网络抓取。（2）自然语言处理：虽然一些文本数据分析会涉及到较高级的统计方法，但是部分分析还是会更多的涉及到自然语言处理过程，如分词、词性标注、句法分析等。（3）命名实体识别：即利用词典或统计方法识别命名的文本特征，如：人名、地名、组织机构、特定的缩写等。（4）模式识别：文本中可能会出现像电话号码、邮箱地址这样的有正规表示方式的实体，通过这些特殊的表示方式或者其他模式来识别这些实体的过程就是模式识别。（5）关系识别：识别代指同一对象的不同词汇。（6）文本聚类：即运用无监督机器学习手段归类文本，适用于海量文本数据的分析，在发现文本话题、筛选异常文本资料方面应用广泛。（7）文本分类：即在给定分类体系下，根据文本特征构建有监督机器学习模型，达到识别文本类型或内容主旨的目的。（8）文本关联：传统关联规则挖掘方法在文本特征上的直接应用，包含文档类型关联、词汇关联、实体关联等内容。（9）情感分析：包括识别文本隐含的主观内容、挖掘不同形态的观点信息，如：情绪、情感、语气、观点等，目前的文本数据分析技术可以细化到实体、概念、话题等級的情感分析。（10）定量文本分析：人为或者通过机器学习来挖掘词汇间的语义、语法关系，进而识别一段文本的含义、文体。

1.1.3.3 文本数据分析的价值

文本数据分析涵盖诸多的研究方向，而这些研究方向又可以被应用于不同的领域，下面从不同的应用领域出发，分别介绍文本数据分析技术在各个领域发挥重要的作用。

在商业实践中，通过分析客户和竞争对手相关文本数据可以提高企业自身竞争力。客户分析方面，企业可以从客户关系数据、社交媒体、电子商务平台等渠道获取相关文本数据，通过自然语言处理和相关分析揭示隐含在文本背后的商业信息，进而进行产品分析、客户关系管理、客户流失预测、企业的风险和机会分析，总结产品优缺点、把握客户情感和需求、了解舆论导向，为商业决策、行业趋势研究等提供有力支持。比如，了解未购买产品的客户对该种产品的情感是正面还是负面，可以在一定程度上判断要说服这名客户购买该产品的难易程度；通过分析电影预告片评论来预测电影受欢迎程度，进而迅速地调整推广策略；首次发布的产品，针对开始出现的投诉进行文本数据分析，可以快速识别出产品存在的问题，以便更快、更积极地避免未来产品中出现同样的问题；分析竞争对手产品信息及评价，可以及时了解市场需求和走势，知己知彼。

在欺诈识别中，比如健康险投诉事件，使用文本数据分析技术可以解析出客户的评论和理由，进而识别出欺诈模式，标记出风险的高低，将更多的资源投入高风险的投诉中。

信息检索里的许多任务都可以归结为文本分类问题，包括搜索引擎对网页的相关性排序、垃圾邮件的过滤、文档的组织等，网页检索方面也越来越多地引入信息检索和文本分类的技术，以更好地理解用户的搜索需求，提供给更优的信息处理服务。

在安全监控领域，很多文本数据分析软件包都被设计用于监测和分析纯文本数据，比如互联网新闻、博客等等，当中也会涉及到情感分析、文本加密或解密技术等。将这些技术用于追踪跨境的有组织犯罪，可以提高在跨境执法方面组织效率；用于分析罪犯(或嫌疑犯)的真实供述，可以研发出预测模型以区分谎言和实话，与测谎仪等其他测谎技术相比，避免了过多的中介物的干扰；用于监控情感信息，可以识别消极情感信息的突然增加。

除此之外，文本数据挖掘也被用于生物医学、化学、金融市场、社会科学等研究中。

1.1.3.4 文本数据分析的发展趋势

上文已经提到，文本数据分析应用领域非常广泛，在各个领域的发展程度势必存在一定差异，但总体上仍然存在较大的发展空间，可以是技术上的创新，也可以是应用领域的拓展，综合来讲，文本数据分析在未来短时间内有以下发展趋势：（1）信息提取 大数据环境下，信息密度低是一个共性问题，如何在海量的文本数据中过滤掉无用信息并快速提取所需的要素一直以来都是一个研究热点，目前虽然在文本数据结构化处理方面已经取得了一定的进展，但是信息要素的提取仍然是一个研究热点和难点。

（2）多语言分析 一般情况下，文本数据分析研究多围绕一种语言展开，其中针对英文的研究更加完善，但是随着机器翻译和机器学习等技术的不断发展，不同语言之间的差异已经逐渐得到弥补，为文本数据分析在多种语言上的发展提供了支持。（3）情感分析 “情感引导决策”的理念长期深入人心，关于文本情感的研究一直是大家关注的热点，但是近期关于情感分析更加广泛、系统的研究热潮仍然超出了正常预期。除从文本数据中提取情感状态外，将情感倾向量化处理将是近期的研究热点，在广告商、新闻传媒、市场营销、代理机构等应用领域的发展也将更加成熟。（4）表情符号分析 随着网络信息技术的飞速发展，人们越来越多的依赖网络进行信息沟通，而在网上发表意见、讨论问题、交流情感的过程中又会利用许多表情符号来直观的表达个人情感，如果能对表情符号进行识别、划分，将得到更加有价值的情感信息，目前已经有一部分学者开启了这方面的研究，未来的研究成果也是十分值得期待的。（5）自然语言生成 所谓自然语言生成（natural language generation，NLG），指机器通过上下文、事先设定的规则等形成算法进而生成文本内容，如邮件、即时信息、翻译等，适用于生成大量的、重复度高的内容，如体育、天气预报相关的内容，在人机对话方面（如智能机器人）也是一个研究热点。

1.2 文本数据分析相关概念

虽然文本数据的分析流程与传统数据挖掘相似，但文本数据表现为非结构性、自由形态的文字，或者是由许多符合特定计算机语言的语法及语法规则构成的文字和语句的字符串，利用现有数据挖掘方法是无法直接进行分析的。抛开词频之外的复杂语义结构，挖掘过程首先要考虑将这种非结构化的数据结构化处理，常规做法就是分词、生成文本-词频矩阵，后续可能涉及到高维矩阵的处理等问题。所以，在进行复杂的文本数据分析之前，本节先对文本数据结构化处理会涉及到的概念及相关操作一一作简要说明，主要包括分词、构建“文档-词项”矩阵（Document-Term Matrix，DTM矩阵）、TF-IDF、词向量等。

1.2.1 分词（Word Segmentation）

分词，是将由连续字符组成的语句按照一定规则划分成一个一个独立词语的过程。不同的语言有不同的句法结构和行文方式，在实际分析中，中文和英文分词是经常要处理的两种情况。在英文中，单词之间是以空格作为自然分界符的，所以英文文本属于词干化（word stemming），可以指定“空格(space)”符作为分词标记，而中文词没有形式上的分界符，只有字、句和段能通过明显的分界符来简单划界，所以中文比之英文要复杂的多、困难的多。如对中文表述“中文分词”和英文表述“Chinese Word Segmentation”进行分词，后者按照空格符直接提取词“Chinese”、“Word”及“Segmentation”即可，而前者则需指定更复杂的分词规则。自20世纪80年代以来，中文分词就成为研究热点，由于中文语言的复杂性也使之一直处于发展阶段。本部分重点介绍中文分词相关内容。现有的分词算法可分为三大：基于规则的分词方法、基于统计的分词方法、基于字符串匹配的分词方法，**基于字符串匹配**的分词方法就是**基于词典分词**。当前**基于规则**的分词方法都是基于人工标注的词性和统计特征对中文语料进行训练，得到对每一个字的类别标注，根据标注结果来进行分词，同时通过模型计算各种分词结果出现的概率，将概率最大的分词结果作为最终结果。基于统计的分词关注的是文本本身的词项构成，其基本思想是**字符串频数分析**。分词过程可表述为：将文本中所有相邻汉字按照某一长度构成字符串（按照中文词构成规范，最小组合长度为两个汉字），**遍历所有字符串组合并统计其出现的频数，字符串出现的频数越高表明其为固定搭配词的可能性越大，设定某一频数阈值，超过阈值时则将该字符串划分为固定搭配词**，接着，再继续按照其他长度进行搜索。该方法优势在于无需与词典做比照，分词效率较高，且利用了上下文信息，缺点则是未充分利用常用词信息。基于词典分词，是应用词典匹配、汉语词法或其它汉语语言知识进行分词的方法，使用的词典可以是庞大的统一化词典，或者是分行业的垂直词典，如中科院开发的汉语语法分析系统 ICTCLAS（Institute of Computing Technology, Chinese Lexical Analysis System）、知网词库等，该方法简单、分词效率较高，设计和操作也较为简单，易于实现。基本分词原理是：文本输入后，依据一定策略将待分析的文本与词典进行词项匹配，匹配成功则提取该词。匹配策略又有不同的分类方法：按照扫描方向可分为**正向匹配**（从左至右）、**逆向匹配**（从右至左），正向匹配时，从若干汉字组合成的字符串开始搜索，若匹配则完成分词，若不匹配则划除字符串的最右一个汉字重新搜索，逆向匹配相反；按照匹配长度差异又可分为最大匹配、最小匹配，二者的基本区别在于不同长度字符串组合

的搜索顺序，前者首先选取词典中最长的词长度为输入文本中字符串长度，判断是否匹配，再搜索次长字符串，直至完成全部匹配，而最小匹配法则相反，实际应用中主要使用最大匹配。常用的基于词典分词的方法主要有正向最大匹配、逆向最大匹配、全二分匹配及逐词遍历等。在中文分词过程中，有两个颇具挑战性的问题，一是歧义词识别，二是未登录词识别。

1.2.1.1 歧义词

一般当一个字可以同时作为两个词的组成部分，并且这两个词按序同时出现在一个语句中时，就可能会出现歧义的现象。目前的歧义一般分为三种：交叉歧义，组合歧义，真歧义。

- (1) **交叉歧义**：语句 ABC 中，AB 和 BC 都能组成汉语词汇，就会造成交叉歧义，如“产品质量”一句中，“产品”、“品质”、“质量”都可以构成常用汉语词汇，属于交叉型歧义片段。
- (2) **组合歧义**：如果两个词汇 A、B 既可以单独成词又可以组合在一起以 AB 的形式成词的话，在分词时两个词汇同时连续出现就会导致组合歧义的问题。如“以我个人的名义”和“我一个人在家”，前者是“个人”是一个词，后者是“个”、“人”是两个词。
- (3) **真歧义**：如果一句话有多重切分方式，那么就会导致真歧义的问题。如“乒乓球拍卖完了”，切分胃“乒乓球拍/卖/完了”和“乒乓球/拍卖/完了”都是合理的。

1.2.1.2 未登录词

未登录词指在分词词典中没有收录但是已被公认为词语的词汇，也可以称之为新词。最典型的未登录词的例子就是人名，此外，未登录词的一个重要来源就是互联网，人们在网络交流过程中会创造很多新鲜词汇，如“神马”、“细思极恐”等等，这些都给分词任务带来了很大的困难。

1.2.2 停用词（Stop Words）

停用词指在文本中不影响文本主要意思表达的“无用”内容，通常为在人类自然语言中常见且无具体意义的助词、虚词、代词，如英文词“I”、“this”以及中文词“的”、“啊”等，停用词的存在直接增加了文本数据的特征维度，提高了文本数据分析过程中的成本，若直接以包含大量停用词的文本为分析对象，还可能会导致数据分析的结果存在较大偏差。因此，停用词去除也是文本数据分析中的重要一环，常用的方法有**词表匹配法、词频阈值法、权重阈值法**等。

1.2.3 “文档-词项”矩阵（DTM矩阵）

DTM 矩阵即为文本数据的信息阵，是以文档为样本、各文档分词得到的词汇项并集为变量集合、词频为变量观测值的矩阵。分词后得到 DTM 矩阵就初步实现了对文本数据的结构化处理。由 N 个词汇项 @todo 带下标公式 构成的包含 M 个文档 @todo 带下标公式 的文档集合 D，其 DTM 矩阵为：@todo DTM 矩阵表格 其中，@todo 带下标公式 为第 j 个词项在第 i 个文档中出现的频数。

1.2.4 词频-逆向文档频率(Term Frequency–Inverse Document Frequency，简称 TF-IDF)

文档集由若干文档组成，文档又由若干词汇组成，词汇的重要性随着它在文档中出现的次数成正比，但同时会随着它在文档集中出现的频率成反比，也就是说一个词汇如果在所有的文档中都频繁出现，在某种程度上意味着这个词不能很好地表示文本的特征，如一些没有实际意义的停用词。为了降低这样的词汇对文本数据分析的影响，需要降低这些词汇 DTM 矩阵的系数，也就是需要进行词频矩阵的核心信息提取，使得词汇能突出所属文档的个性化，对单独文档而言，又能够体现相对重要性，从而方便后续分类、检索、提取规律等。“词频-逆向文件频率”是 Salton 等提出的最为常用的提取核心信息的统计方法。基本思想是：若词项 @todo 带下标公式 在文本 @todo 带下标公式 中出现的频率高，且在其他文本中出现频率低，则认为该词具有良好的文本特征表示能力，赋予其较高的权重。其本质上是按照逆文本频率 idf 对词频加权，一定程度上体现了文档集合中各文档的区别，将词频的绝对多少转化为相对多少，最终特征空间由加权后的词频即 $tf \cdot idf$ 值构成。TF-IDF 方法在一定程度上解决了停用词和常用词被划归为关键特征的问题，更能区分词项对于文档的重要程度，从而准确表达文档特征。其优点在于原理直观、易解释；计算较简单、效率高。作为重要的文本特征表示方法，目前 TF-IDF 在语义识别、文本聚类、文本分类、信息检索、推荐系统中都有广泛应用。TF-IDF 的计算流程可参见 1.5 节深度阅读材料 1.5.1。

1.2.5 词向量 (Distributed Representation)

要将自然语言理解的问题转化为机器学习的问题，首先要找一种方法把文字符号数学化。所谓词向量，直观来讲就是将自然语言中的词汇用数值向量进行表示的方式，其发展过程经历了从 one-hot representation 到 distributed representation 的过程。one-hot representation 是指用一个很长的向量来表示一个词，向量的长度为词典的大小，向量的分量只有一个 1，其他全为 0，1 的位置对应该词在词典中的位置。one-hot representation 是 NLP 中最直观，也是到目前为止最常用的词表示方法，但该种表达方式存在两个缺点：一是容易受维数灾难的困扰，特别是将其用于 Deep Learning 的一些算法时；二是不能很好地刻画词与词之间的相似性。为了克服 one-hot representation 的不足，Hinton 于 1986 年提出了 Distributed Representation 方法。该方法通过训练（常用神经网络的方法）将某种语言中的每一个词映射成一个固定长度的短向量（相对于 one-hot representation 的“长”而言，维度以 50 维和 100 维比较常见），我们将这样的向量称为词向量。将所有这些向量放在一起形成一个词向量空间，而每一向量则为该空间中的一个点，在这个空间上引入“距离”，可以用最传统的欧氏距离来衡量，也可以用 cos 夹角来衡量，就可以根据词之间的距离来判断它们之间的（词法、语义上的）相似性，所以说词向量可以提取词与词之间的深层语义关系。关于词向量的发展历程以及训练模型的构建可以参见深度阅读材料 1.5.2 提供的几篇论文。

1.2.6 词性标注(Part-of-speech Tagging，POS)

所谓词性，就是对词语的一种分类方式。现代中文词汇大致可以分为名词、动词、形容词、数词、量词、代词、介词、副词、连词、感叹词、助词和拟声词等12种，英文词汇基本分类包括名词、形容词、动词、代词、数词、副词、介词、连词、冠词和感叹词等10种。词性作为对词的一种泛化，在语言识别、句法分析、信息抽取等任务中有重要作用。词性标注是给句子中每个词一个词性类别的任务，属于自然语言处理中的基本操作，是信息检索等领域不可或缺的步骤。如“我喜欢音乐”中，“我”为人称代词（r），“喜欢”为动词（v），“音乐”为名词（n）。具有两个或两个以上词性的词，即兼类词的存在是词性标注的难点。目前，针对兼类词的歧义排除，**比较经典的算法可以归纳为以下三类：基于规则的算法、基于概率统计模型的算法、规则和统计相结合的算法。**

1.2.7 命名实体识别（Named Entity Recognition）

命名实体识别指识别文本中具有特定意义的实体，如人名、机构名、地名等专有名词和有意义的时间等，是信息检索、问答系统等技术的基础任务。如在“小明在夏威夷度假。”中，命名实体有：“小明——人名”、“夏威夷——地名”。

@todo:此处需"命名实体"配图 1.2.7

相对于英文，中文命名实体没有明显的形式标志，还存在分词的干扰，导致中文命名实体识别难度也高于英文。目前，命名实体识别的方法有基于规则的方法、基于统计的方法及混合方法。

1.2.8 句法分析(Dependency Parsing，DP)

句法分析，指根据给定的语法体系下分析句子的句法结构，划分句子中词语的语法功能，并判断词语之间的句法关系。短语结构和依存结构是目前句法分析研究中应用最广泛的两类语法体系，不同语法在句法分析的过程中采用的分析算法差别不大，算法的发展经历了从基于规则算法到基于统计算法的过程。如“我喜欢音乐”一句中，“我”为主语，“喜欢”为谓语，“音乐”为宾语；“我”和“喜欢”为主谓关系（SBV），“喜欢”和“音乐”为动宾关系（VOB）。

@todo:此处需"句法结构"配图 1.2.8

1.2.9 情感分析（Sentiment Analysis）

所谓情感分析，也可以称为意见挖掘(Opinion Mining)，是通过计算机语言技术、自然语言处理方法整理和分析相关的文本数据，对主观情感性文本进行分析和推理的过程，是一个分析人们书面语言情绪、情感或态度的领域。按照应用领域的不同，可以分为褒贬情感倾向分类、主观性内容识别和在线评论经济价值挖掘等几个方面；按照文本类型的不同，可以分为产品评论的情感分析、新闻评论的情感分析等；按照分析粒度的不同，可以分为篇章级、词语级、语句级三个不同粒度层次的情感分析。综合国内外的研究成果，从技术方法来看，有两类方向：一类是基于语义分析，另一类是基于机器学习。

@todo 其他自然语言处理相关

1.3 常用文本数据分析工具

一个完整的文本数据分析过程，除了适宜的开发环境外，还要借助很多自然语言处理以及数据分析的工具，譬如对中英文来说，都需要基本的分词处理，进一步，可能还需要词性标注，句法分析，关键词提取，文本分类，情感分析等等，在这些方面，特别是面向英文领域，有很多优秀的 Python 工具包。本节结合本书的内容安排，整合了常用的文本数据分析工具、库等，方便读者后续的学习和应用。

1.3.1 Jupyter Notebook

Jupyter Notebook（原 IPython Notebook），是一个开源的、交互式的科学计算 Web 应用，它支持实时的代码执行，支持文本内容嵌入，包括数据公式、数据可视化图表、视频等，通常被用于数据清洗和转换、数值拟合、统计建模、机器学习等，是数据科学相关研发人员的得力工具。简单来讲，Jupyter Notebook 可以执行并保存代码的执行结果，下一次打开时无需重新运行即可查看，是一种可以实现可读性分析的灵活工具。Jupyter Notebook 具有以下几个优势：（1）支持 40 多种程序语言，包括数据科学分析中常用的 Python、R、Julia 和 Scala；（2）可以通过邮件、Dropbox、GitHub 以及 nbviewer（Jupyter Notebook 查看器，用于渲染.ipynb 文件，并支持 HTML 和 PDF 输出）共享 Notebook 文件；（3）交互式窗口，代码输出结果可以是图片、视频、LaTeX、JavaScript 等，有助于数据的实时处理及可视化展示。（4）支持多种类似 Apache Spark 的大数据处理工具以及相关库，如 pandas、scikit-learn、ggplot2 等。想更加深入的了解 Jupyter Notebook，可以到 1.5.3 提供的网址阅读相关文档，具体的安装、使用说明见第二章 2.2、2.3 节。

1.3.2 Python 文本处理工具

1.3.2.1 jieba

jieba，中文表述为“结巴”，是用 Python 编写的最好用的中文分词组件之一，具备中文分词、关键词提取、词性标注等功能，代码清晰，扩展性、兼容性较好。总体上有以下几种特点：
(1) 支持繁体中文分词。(2) 可以添加自定义词典。(3) 支持三种分词模式，分别为：精确模式、全模式、搜索引擎模式，不同的分词模式有不同的优势和适用场景，其中，**精确模式**可以将句子精确地切分开，适用于对分析精度要求较高的文本数据分析；**全模式**把句子中所有的可以成词的词语都扫描出来，优势是分词速度快，但是不能解决歧义。**搜索引擎模式**是在精确模式的基础上，对长词再次切分，适合用于搜索引擎分词。

1.3.2.2 NLTK

NLTK 是利用 Python 程序处理文本数据的免费开源工具，提供了 50 多个如 WordNet 的语料和词典资源接口以及一系列的文本处理库，支持分词、标记、语法分析、语义推理、分文本类等文本数据分析需求，适合语言学家、工程师、学生等各种有分析需求的群体，支持 Windows, Mac OS X 和 Linux 系统。想更加深入的了解 NLTK，1.5.4 提供了官方文档的网址，并为刚刚接触 NLTK 或者需要详细了解 NLTK 的读者推荐了两本书籍。

1.3.2.3 CoreNLP

CoreNLP 是斯坦福大学自然语言处理小组研发的一整套开源自然语言处理工具，有以下几个主要特点：（1）是自然语言处理的集成工具，对于自然语言处理的标识解析、句子切分、词性标注、词元分析等各个方面应用都有具体的实现；（2）分析快速、稳健；（3）提供当下多种流行通用的开发语言接口，覆盖面较广；（4）可以很方便的选择需要使用或不使用的语言分析工具模块，有较强的灵活性和可扩展性；（5）支持多国语言的分析，各项功能及支持的语言见下表：@todo 表格 <http://stanfordnlp.github.io/CoreNLP/#about>

1.3.2.4 Pattern

Pattern 是由比利时安特卫普大学 CLIPS 实验研发的一套基于 Python 的 web 数据挖掘工具，包括数据抓取模块（Google、Twitter、维基百科的 API，以及网络爬虫和 HTML 分析器）、自然语言处理模块（分词、词性标注、句子切分、语法检查、拼写纠错、情感分析、句法分析等）、机器学习模块（向量空间模型、聚类、SVM）以及可视化模块等。

1.3.2.5 langid.py

langid.py 是一个独立的语言识别工具，支持 97 种语言的快速检测，并且可以作为网络服务。

1.3.2.6 TextBlob

TextBlob 是基于上面提到的两个 Python 工具包 NLKT 和 Pattern 封装的文本处理工具包，同时提供了很多文本处理功能的接口，包括词性标注、名词短语提取、情感分析、文本分类、拼写检查等，还包括翻译和语言检测功能。

1.3.2.7 SnowNLP

SnowNLP 是受到了 TextBlob 的启发而写的处理中文文本的 Python 库，可以方便的处理中文文本内容，和 TextBlob 不同的是，SnowNLP 没有用 NLTK，所有的算法都是自行实现的，并且自带了一些训练好的词典。

1.3.2.8 PyNLPI

PyNLPI 是“Python Natural Language Processing Library”的缩写，是一个用于自然语言处理的 Python 库，可以用于处理常规的自然语言处理任务，如特征提取、建立语意模型，此外，还提供文件格式解析器，其最大的特点是提供了处理 FoLiA XML 的丰富库。

1.3.3 Python 科学计算工具库

1.3.3.1 Matplotlib

Matplotlib 是一个科学计算领域广泛使用的基于 Python 的二维绘图库，风格跟 Matlab 相似。它包含了大量的工具，使用这些工具可以创建各种图形，包括简单的散点图、直方图、频谱图等常用统计图像，还提供了丰富的附加工具，如可以绘制地图的 basemap 和 cartopy，绘制 3D 图的 mplot3d，以及更加高级的绘图接口，如 seaborn、holoviews、ggplot 等。此外，在图像美化方面，Matplotlib 可以自定义线条的颜色和样式，支持在一张绘图纸上绘制多张小图，也可以在一张图上绘制多条线进行对比分析。输出的图像具有可复制、质量高（达到了科技论文中的印刷质量）、格式多样等特点。@todo 示例图片 1.3.3.1-1、1.3.3.1-2、1.3.3.1-3 可以访问 <http://matplotlib.org/gallery.html#> 浏览更加全面的图库内容，1.5.6 也为读者提供了 Matplotlib 的深度阅读材料，感兴趣的读者可以自行选择阅读。

1.3.3.2 Numpy

NumPy 是 Numerical Python 的简称，是基于 Python 的高性能计算和数据分析第三方库，提供了快速高效的多维数组对象 ndarray，其操作也是围绕 ndarray 进行。NumPy 针对数组运算提供大量的数学函数库，包含线性代数、傅里叶变换和随机数生成函数，严格的数据处理能力使得其为很多大型金融公司以及核心的科学计算组织使用。想更多的了解 Numpy 支持的数据结构及相关操作，请读者登录 1.5.7 提供的网址阅读快速入门教程。

1.3.3.3 SciPy

SciPy 是基于 Python 的开源数学算法和科学计算工具包的集合，通过提供高水平的数据处理和可视化功能能增强了 Python 的交互性，使 Python 成为像 MATLAB 一样的数据处理环境。具体包含最优化、线性代数、积分、内插法、快速傅里叶变换、信号和图像处理等模块。SciPy 构建于 Numpy 之上，Numpy 为其提供了方便快捷的 N 维数组相关操作，Numpy 和 Scipy 适用于多种系统，开源、免费，且安装简单，实际应用中常常将二者相结合，Python 大多数机器学习库都依赖于这两个工具库。

1.3.4.4 Pandas

Python 一直以来都是数据预处理和加工的首选工具，但是缺乏数据分析和建模的功能，Pandas 的出现弥补了这一不足，使得可以利用 Python 完成一系列的数据处理、分析流程，而无需借助像 R 这样的分析语言，促使 Python 被更加广泛的用于学术研究和商业分析等领域。

域，如金融、统计、广告、网站分析等。Pandas 的名称源于“panel data”，是基于 NumPy 和 Matplotlib 开发的数据处理和数据分析库，特别的是，Pandas 提供了可用于处理时间序列的高级数据结构和相关工具，使得在 Python 中处理数据非常快速和简单，具体支持分析的数据类型有：（1）包含多列的表格式数据，如 SQL 表和 Excel 表格；（2）有序或无序的时间序列（等间隔或不等间隔）；（3）任意的矩阵数据；（4）其他形式的统计数据集。Pandas 显著功能特点总结如下：（1）支持一系列的数组数据结构，其中两个重要的数据结构是：Series 和 DataFrame，DataFrame 和 R 语言里的 data.frame 很像；（2）通过标签操作实现大型数据集的切分、多方式索引及子集构造；（3）支持数据集的聚合和灵活转换；（4）利用时间处理函数，可以生成自定义间隔的序列、进行移动窗口（滑窗）统计分析和时间滞后处理等；（5）利用输入和输出工具可以实现不同存储格式的外部数据的读取及分析结果的展示和保存；（6）提供智能的数据分组和缺失值处理方式。想深入学习 Pandas 支持的数据结构及其数据处理、分析功能可以阅读 1.5.8 给出的文档。

1.3.4 Python 机器学习、数据挖掘工具库

1.3.4.1 scikit-learn

scikit-learn 是基于 NumPy、SciPy 和 Matplotlib 构建的开源 Python 机器学习库，名称 "SciKit" 源于 SciPy Toolkit。scikit-learn 为用户提供各种机器学习算法接口，可以让用户简单、高效地进行数据挖掘和数据分析，基本功能主要被分为六个部分：分类、回归、聚类、降维、模型选择、数据预处理，每一部分都涵盖不同的算法说明、方法调用说明及应用案例。scikit-learn 有非常完善的官方说明文档，包括机器学习理论的介绍、scikit-learn 使用指南、其他相关资源等等，有需求的读者可以前往学习。

1.3.4.2 Gensim

Gensim 的雏形是几个用于提取跟目标文章相近文章的 Python 脚本，gensim 即 “generate similar”的缩写，到目前为止，Gensim 已经发展成一个免费的 Python 自然语言主题模型库，旨在将文本中涵盖的语意统计量化、分析纯文本文件的语义结构、检索语意相近的文件，具体功能可以概括为：构建主题模型、文件检索、相似度查询。Gensim 具有以下特点：（1）跨平台：Gensim 基于 Python，所以同样支持 Windows、Mac、Linux/Unix 系统以及其他支持 Python 和 Numpy 的系统；（2）稳健：长期以来，Gensim 已经被各类使用者应用于多种系统，稳健性得到了较好的验证；（4）开源；（5）高效：通过使用最优的核心算法以及分布式计算提高计算速度；（6）支持相似性查询：顺应主题模型发展趋势，Gensim 提供语意相似性检索工具。（7）包含多种时下流行算法，如潜在语义分析（Latent Semantic Analysis (LSA/LSI/SVD)）、狄利克雷分布（Latent Dirichlet Allocation (LDA)）、随机映射（Random Projections (RP)）、分层狄利克雷过程（Hierarchical Dirichlet Process (HDP)）、word2vec 深度学习等。想系统学习 Gensim 提供的算法及使用方法可以直接访问官方主页 <https://radimrehurek.com/gensim/>。

1.3.4.3 PyBrain

PyBrain 是 “Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library” 的简写，是一个模块化的开源 Python 机器学习库，旨在提供灵活的、易用的、高效机器学习算法以及预先定义的环境来进行算法对比。相较于其他机器学习库，PyBrain 更加适合于初学者，但是也能支持高水平的研究需求。读者可以访问官方网站 <http://pybrain.org/> 做进一步了解，也可以找到官方提供的使用文档。

1.3.5 词典资源

1.3.5.1 WordNet

WordNet 是由普林斯顿大学认知科学实验室开发的一个大型英语词汇数据库，其中名词、动词、形容词以及副词各自都按照词义进行分组，形成一个同义词语意网络，每一个具有相同意义的词组成的同义词集合（synset）都代表一个基本的语义概念，WordNet 为其提供了简短、概要的定义，并且这些集合之间也根据各种关系相连接，WordNet 记录了不同同义词集合之间的语义关系。由于它包含了语义信息，所以有别于通常意义上的字典，可以说 WordNet 是一部语义词典。

许多国家在筹划和建立与英文 WorNet 兼容的本国语言 WordNet 系统，如欧洲基于 WordNet 的 EuroWordNet，韩国的 KoreanWordNet 等。

关于 WordNet 的详细介绍，可以阅读 1.5.9 给出的书籍或者登陆官方网站学习。

1.3.5.2 HowNet

HowNet 是中国以 WordNet 为框架的概念词典，中文名称为知网，是一个以汉语和英语的词语所代表的概念为描述对象，是把概念与概念之间的关系以及概念的属性与属性之间的关系形成一个网状的知识系统，而不是一部语义词典。

1.3.5.3 中文概念辞书

中文概念辞书(Chinese Concept Dictionary , CCD)是北京大学计算语言学研究所构建的 WordNet 框架下的现代汉英双语概念词典，同时提供汉英双语概念的语义知识表达。CCD 从关系语义学的观点出发，用同义词集合（SynSet）来描述概念，用概念间的关系来描述语义，方便语义关系的表示和检索，有利于简单地实现语义距离的计算，可以直接应用于机器翻译、自动文摘、文本分类、概念检索和信息提取等方面的语义理解。

1.3.5.4 Mindnet

Mindnet 是微软研究院自然语言处理小组研发的大规模语义关系知识库。具体结构和构建方式可以阅读 1.5.17 提供的文章。

1.3.5.5 CSC

CSC 是一个规模较大的中文语意语库，该词库目前收录了 14 万以上的书面形式的词条，包括单词、固定词组、成语、少量在中文文章中较常见的英文缩写等等，每个词条通过关系比较密切的相关词与其它词条相连结，整个词库呈现为比较复杂的网络结构，并带有多种检索手段和显示方式。它可用于搜索引擎、全文检索等检索工具中，帮助用户选择关键词、帮助系统提供相关搜索词或进行其它智能处理，例如语义搜索、精准匹配等。也可用于字处理、写作助理等办公软件中，丰富的相关词能为写作中的词语优化提供较有力的支持。还可作为自然语言处理的资源或汉语教学的辅助工具。该词库已在有些企业和科研机构中得到应用。

1.3.5.6 中文词汇网络

中文词汇网络(Chinese WordNet,简称 CWN)是台湾中央研究院中文词网小组(Chinese WordNet Group)结合分析详尽的中文词汇词义数据与网络科技技术开发的中文词汇网络。

1.3.5.7 自然语言处理词典(The Natural Language Processing Dictionary)

自然语言处理词典是澳大利亚新南威尔士大学的教授 Bill Wilson 为所开设的“人工智能”课程编制的，除此之外还编制了 Prolog 词典(The Prolog Dictionary)、人工智能词典(The Artificial Intelligence Dictionary) 和机器学习词典(The Machine Learning Dictionary)，词典 URL 请见 1.5.18。

1.3.5.8 其他词典资源

除上述词典外，还有很多其他相关资源，如：中文同义词词林扩展版、基于框架语意学以语料库为基础建立的英语词汇资源库 FrameNet、VerbNet、CYC、英国剑桥综合语言知识库 ILD、日本电子词典研究院开发的日文处理方面的主要语意词典 EDR 等，感兴趣的读者可以进一步学习。

1.3.6 在线自然语言处理 API

1.3.6.1 哈工大语言云 (<http://www.ltp-cloud.com/>)

哈工大语言云是基于云计算技术的中文自然语言处理服务平台，以哈工大社会计算与信息检索研究中心研发的“语言技术平台（LTP）”为基础，为用户提供高效精准的中文自然语言处理云服务，包括中文分词、词性标注、命名实体识别、依存句法分析、语义角色标注等，用户只需要根据 API 参数构造 HTTP 请求即可在线获得分析结果。

1.3.6.2 腾讯文智 (<http://nlp.qq.com/>)

腾讯文智是基于并行计算系统和分布式爬虫构建的中文语义开放平台，结合独特的语义分析技术，一站式满足用户自然语言处理、转码、抽取、全网数据抓取等中文语义分析需求，用户能够基于平台对外提供的 API 实现搜索、推荐、舆情、挖掘等语义分析应用，也能够定制产品特色的语义分析解决方案。

1.3.6.3 linguaKit（<https://linguakit.com/es/>）

linguaKit 是由西班牙自然语言研究团队 Cilenis 研发的一套自然语言与文本分析工具包，主要功能包括单词形式转换、文本情感分析以及文本关键字抽取，其基本功能主要面向西班牙文和葡萄牙文，其中情感分析以及关键字抽取部分模块也支持英文，除此之外，该工具包也同时包含了词频统计、绘制词云图等辅助功能。

1.3.6.4 NLPIR 大数据搜索与挖掘共享平台（<http://ictclas.nlpir.org/nlpir/>）

NLPIR 是由北京理工大学海量语言信息处理与云计算工程研究中心大数据搜索与挖掘实验室（Big Data Search and Mining Lab.BDSM@BIT）开发的大数据搜索与挖掘共享平台，NLPIR 能够多角度满足应用者对大数据文本的处理需求，包括大数据完整的技术链条：网络抓取、正文提取、中英文分词、词性标注、实体抽取、词频统计、关键词提取、语义信息抽取、文本分类、情感分析、语义深度扩展、繁简编码转换、自动注音、文本聚类等。

1.3.6.5 玻森中文语义开放平台（<http://bosonnlp.com/>）

玻森中文语义开放平台提供中文自然语言分析云服务，专注中文语义分析，提供多个语义分析 API 满足情感分析、实体、分类、聚类等分析需求，并可定制数据分析模型和解决方案。

1.4 小结

作为本书的开端，本章从文本数据分析初学者的视角出发，向读者介绍自然语言处理及文本数据分析的基础知识。首先以文本数据作为切入点，界定了文本数据的含义，总结了文本数据的特点；在此基础上介绍了文本数据分析的含义、基本内容、意义和发展趋势，使读者快速、全面的了解文本的数据分析理论；接着，梳理了文本数据分析过程中涉及到的概念；最后，统一介绍了文本数据分析过程中常用的分析工具，包括自然语言处理工具、科学计算库、机器学习库、词典资源等等。完成本章的学习后，文本数据分析初学者将对文本数据分析领域有了更加明确、清晰的认识，对文本数据分析过程中常用工具与资源有所了解，想进一步学习相关内容的读者可以参考 1.5 节提供的深度阅读材料继续学习。

1.5 深入阅读材料

由于篇幅限制，很多文本数据分析相关概念、工具都不能做细致的解释说明，笔者将部分参考文档和资源网站等集中列于本节，有需要的读者可以自行选择阅读。

1.5.1 词频-逆向文件频率(**TF-IDF**)的计算原理在这篇文章中有相近的说明：

Salton G, Fox E A, Wu H. Extended Boolean information retrieval[J]. Communications of the ACM, 1983, 26(11): 1022-1036.

1.5.2 关于词向量的发展历程以及模型的训练原理可以阅读以下几篇经典论文：

- (1) Bengio Y, Schwenk H, Senécal J S, et al. Neural probabilistic language models[M]//Innovations in Machine Learning. Springer Berlin Heidelberg, 2006: 137-186.
- (2) Collobert R, Weston J. A unified architecture for natural language processing: Deep neural networks with multitask learning[C]//Proceedings of the 25th international conference on Machine learning. ACM, 2008: 160-167.
- (3) Mnih A, Hinton G E. A scalable hierarchical distributed language model[C]//Advances in neural information processing systems. 2009: 1081-1088.
- (4) Mikolov T, Karafiat M, Burget L, et al. Recurrent neural network based language model[C]//INTERSPEECH. 2010, 2: 3.

1.5.3 **Jupyter Notebook** 的官方文档对 **Jupyter Notebook** 的功能特点、安装使用等都给出了较详细的说明，还给出了许多应用实例，读者可以访问以下网址学习：

<http://jupyter-notebook.readthedocs.io/en/latest/>

1.5.4 读者可以登录 <http://www.nltk.org/> 阅读 **NLTK** 的官方文档了解 **NLTK** 的功能、提供的模块等，下面这两本书推荐给刚刚接触 **NLTK** 或者需要详细了解 **NLTK** 的读者。

- (1) 《Natural Language Processing with Python》
- (2) 《Python Text Processing with NLTK 2.0 Cookbook》

1.5.5 Stanford CoreNLP 官方网站：

<http://stanfordnlp.github.io/CoreNLP/>，CoreNLP 的功能、使用都有详细的说明。

1.5.6 想更全面的掌握 Matplotlib 的使用，推荐阅读以下两本书籍：

- (1) 《Mastering matplotlib》 by Duncan M. McGregor
- (2) 《Graphics with Matplotlib》 by David J. Raymond

以及官方使用指南：<http://matplotlib.org/users/index.html>

1.5.7 Numpy 快速入门教

程：<https://docs.scipy.org/doc/numpy-dev/user/quickstart.html>，对 Numpy 支持的数据结构及相关操作均有解释说明。

1.5.8 Pandas 说明文档：<http://pandas.pydata.org/pandas-docs/stable/index.html#>，对数据结构及数据处理、分析功能均有解释说明。

1.5.9 关于 WordNet 的详细介绍，可以阅读书籍《WordNet: An Electronic Lexical Database》或者登陆官方网站 <https://wordnet.princeton.edu/> 学习。

1.5.10 Mindnet 结构和构建方式说明文章：

Richardson ,S. D. ,William B. Dolan , and Lucy Vanderwende. MindNet : acquiring and structuring semantic information from text[A]. In : Proceedings of COLING'98[C] , 1998 ,1098 - 1100.

1.5.11 自然语言处理相关词典网址：

- (1) The Natural Language Processing Dictionary
– URL: <http://www.cse.unsw.edu.au/~billw/nlpdict.html>
- (2) The Prolog Dictionary
– URL: <http://www.cse.unsw.edu.au/~billw/prologdict.html>
- (3) The Artificial Intelligence Dictionary

– URL: <http://www.cse.unsw.edu.au/~billw/aidict.html>

(4) The Machine Learning Dictionary – URL:

<http://www.cse.unsw.edu.au/~billw/mldict.html>

1.6 练习

1.6.1 回顾文本数据及文本数据分析相关理论知识。

1.6.2 了解 **jieba**、**NLTK**、**CoreNLP** 等 **Python** 文本处理工具，并对比算法原理差异。

1.6.3 进一步了解 **WordNet**、**HowNet**、**FrameNet**、**MindNet** 等资源，并进行对比。

1.6.4 登录 **1.3.6** 所提供的在线自然语言处理平台，了解各个平台的功能、特点并进行对比。

第 2 章 Python 入门

完成一个完整的文本数据分析而过程，除了准备待分析的文本数据和掌握自然语言处理、文本数据分析理论外，还要会使用合适的分析工具进行实现。本书在第 1 章已经介绍了文本数据分析的相关理论知识，本章将针对文本数据分析工具语言 Python 进行讲解，向 Python 的初学者介绍 Python 相关的基本概念与操作。Python 是一种用于浏览和处理文本数据的优秀工具，其语法清晰简洁、易用，可扩展性以及丰富庞大的库深受广大开发者喜爱，其内置的非常强大的机器学习代码库和数学库，使 Python 理所当然成为自然语言处理的开发利器。此外，要使用第三方文本数据分析工具库之前，需要在 Python 环境下安装，本章后半部分将讲解主要工具库的安装步骤。本章具体内容安排如下：2.1 节为 Python 入门章节，从 Python 初学者的角度出发，分别介绍了什么是 Python 和 Python 解释器，列举了用 Python 进行文本数据分析的优势，在此基础上讲解了 Python 安装与使用，具体包括 Python 的基本操作、数据结构、基本语句等；2.2 节将会对本书涉及的文本数据分析工具的安装进行统一说明，包括 Jupyter Notebook、jieba、NLTK、Matplotlib、Numpy、Pandas 等。

2.1 Python 简介

2.1.1 什么是 Python

Python 是由 Guido van Rossum 于 20 世纪 90 年代初开发的面向对象的编程语言，通过 Python 语言可以实现对计算机的控制，类似于 Perl、Ruby、Scheme、Java 等。Python 有以下显著特征：（1）语法优雅，程序可读性强；（2）简单易用，非常适合于原型开发以及其他专门的开发任务；（3）提供了非常完善的标准代码库，支持连接 web 服务器、搜索文本、阅读和修改文件等基础编程任务；（4）交互模式方便测试代码片段；（5）有很强的扩展性和嵌入性；（6）适用于多种系统，包括 Mac OS X、Windows、Linux 和 Unix；（7）下载和使用完全免费，开源；（8）除了内置的库外，Python 还有大量的第三方库供直接使用；（9）支持多种基本数据结构，如数值（浮点数、复数、长整型）、字符串（ASCII 和 Unicode 编码均可）、列表和字典。Python 程序简单易懂，初学者入门容易，可深入性强。经过 20 多年的发展，Python 已经成为重要的编程语言之一，许多大型网站就是用 Python 开发的，例如全球最大的视频网站 YouTube，Instagram，国内的豆瓣，谷歌和国内的易度云计算开发平台也都选择了 Python 语言，NASA（美国航空航天局）、JPL 等都大量地使用 Python 实现科学计算。

2.1.2 Python 解释器

计算机不能直接认识并执行我们写的编程语言，它只能认识机器语言，而解释器是一种翻译程序，能完成从编程语言到机器语言这一翻译过程。Python 解释器就是将 Python 语言翻译成计算机语言并执行的翻译程序，也就是说当我们写完 Python 语句时，我们得到的是一个包含 Python 代码的以“.py”为扩展名的文本文件，要运行代码，需要 Python 解释器去解释、执行该文件。由于实现的方式不同，存在多种 Python 解释器，下面一一向读者介绍。

2.1.2.1 CPython

CPython 是用 C 语言实现的 Python 解释器，也是官方的并且是最广泛使用的 Python 解释器，从 Python 官方网站下载并安装好 Python 后，就直接获得了 CPython 解释器。

2.1.2.2 iPython (An interactive Python kernel and REPL)

iPython 是基于 CPython 的实现一种交互式解释器，支持变量自动补全、自动缩进，内置了许多很有用的功能和函数。

2.1.2.3 Jython

Jython 旧称 JPython，是 Python 的 Java 实现。

2.1.2.4 PyPy

PyPy 时用 rPython 实现的 Python，rPython 是 Python 的一个子集。

2.1.2.5 IronPython

IronPython 是运行在微软 .Net 平台上的 Python 解释器。

2.1.3 用 Python 进行文本数据分析的优势

作为一种通用语言，Python 可以被用于编写任何类型的程序，但是在编写脚本、网站开发、科学计算以及文本处理领域最为常用，在文本处理领域，Python 的优势体现在以下几个方面。相较于其他流行的计算机编程语言，Python 内置的字符串处理函数使其字符串处理方面具有天然优势，其灵活多样的数据格式也更方便于文本数据的结构化。同时，Python 平台还有拥有诸如 NLTK 等相对完善的自然语言分析库，以及 scipy、sklearn 等实用数据挖掘模块，适合文本数据分析初学者的使用与操作。除此之外，对比于 R 等统计分析编程语言，Python 虽然没有内置强大的矩阵计算函数和精美的可视化包，但是更加通用、直观与容易，其 Notebook 的组织形式能够很容易地与他人沟通共享，既适合于有一定编程背景的数据分析初学者，也在从事数据分析与统计工作的人员中广受欢迎。文本数据分析工作往往会涉及多个学科与行业，而 Python 好上手与易交流的特性能使我们更容易地将不同行业的人集合在一起，这也是本书选择它的重要原因之一。

2.1.4 Python 安装与使用

2.1.4.1 Python 版本介绍

目前 Python 官网上可以下载到的有两个版本：Python 2.x 和 Python 3.x。

Python 2.x 是旧版本，但是当前各个领域主要使用的版本，大多生产环境也会应用 2.x 版本，Python 官方核心团队表示 2.7 是 Python 2.x 最后的一版本，但是因为使用规模的比较大的问题，2.7 的小版本也一直在维护中，2016 年末最后发布的正式版本为 2.7.13。

Python 3.x 是 2008 年发布的版本，一直在持续更新迭代，2012 年发布 3.3，2014 年发布 3.4，2015 年发布 3.5，2016 年末最后发布的正式版本为 3.6.0。Python 3.x 是未来 Python 核心开发团队主要发展和维护的版本，是未来的发展方向。

Python 2.x 与 Python 3.x 这两个版本是存在不相兼容的部分的，相对于 2.x 版本，部分第三方库还暂时无法在 3.x 上使用，但是近期标准库的更新只体现在版本 3.x 中，二者具体差异请阅读 2.4.1 提供的官方说明文档。

那么要使用哪个版本呢？这取决于使用者的用途。使用 3.x 版本会面临一些小问题，比如部分库不能使用、目前发行的一些 Linux 和 Mac 仍然默认使用 2.x，但是如果你已经安装了 Python 3.x，并且不会用到 Python 2.x 的模块，那么 3.x 是很好的选择，而且大多数目前发行的 Linux 也正在逐步淘汰 Python 2.x。但是，以下几种情况建议使用 Python 2.x：（1）如果需要部署到不可控环境，比如需要利用特定版本；（2）如果需要使用一些包或工具，但是这些包或工具还未发行适用于 Python 3.x 的版本，而且移植需要大量的工作。

更加详细的对比请见，官方文档 <https://wiki.python.org/moin/Python2orPython3>。

本书以 Python 2.7.x 版本为基础。

2.1.4.2 Python 的安装和运行

Python 的安装十分简单，并且目前发行的很多 Linux 和 UNIX 都已经安装了最近版本的 Python，甚至有些 Windows 系统也已经安装了 Python。下面就简单说明下不同系统下 Python 的安装步骤与运行。

（1）Windows

安装步骤如下：

第一步，从 Python 的官方网站（<https://www.python.org/downloads/windows/>）下载所需的 Python 版本；

The screenshot shows the Python 2.7.13 Windows download page. It lists several download links for different Python versions and architectures. A red box highlights the 'Python 2.7.13 - 2016-12-17' section, which includes links for 'Download Windows x86 MSI installer' and 'Download Windows x86-64 MSI installer'. Below this section, there are links for 'Download Windows help file', 'Download Windows debug information files for 64-bit binaries', and 'Download Windows debug information files'.

第二步，运行下载的 MSI 安装包，弹出以下对话框



在选择安装组件的一步时，勾上所有的组件，特别时 pip 和 Add python.exe to Path ，然后点“Next”完成安装，默认会安装到 C:\Python27 目录下。

最后，打开命令提示符窗口，输入 `python` 后，出现以下结果：



A screenshot of a Windows command prompt window titled "管理员: C:\Windows\system32\cmd.exe - python". The window shows the Python 2.7.6 interpreter running. The text in the window reads:

```
C:\>管理员: C:\Windows\system32\cmd.exe - python
Microsoft Windows [版本 6.1.7601]
版权所有 © 2009 Microsoft Corporation。保留所有权利。
C:\>Users\G_will>python
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

说明已经成功安装了 Python，提示符 `>>>` 表示处于 Python 交互式环境，输入任何 Python 代码，回车后即可立刻得到执行结果，想要退出 Python 交互式环境，可以输入 `exit()` 并回车，也可以直接关掉命令行窗口。

另外，开源社区还有专门为 Windows 用户打包定制的 Python 发行版 WinPython，已经集成了很多常用的 Python 科学计算相关第三方程序包，方便大家安装使用，详见：
<http://winpython.sourceforge.net/>。

(2) Mac OS X

Mac OS X 系统自带 Python，打开终端应用程序直接输入 Python 代码即可，如果安装了 iPython，在终端输入 `ipython`，即可进入 `ipython` 解释器。如果要安装更新的 Python 版本，可按<https://www.python.org/downloads/mac-osx/> 给出的说明操作。

(3) Linux

常见的 Linux 发行版都内置了 Python，所以 Linux 系统很可能已经安装了 Python，可打开命令行窗口并输入 `python` 进行确认。

具体如何安装因 Linux 系统而异，读者可以访问 <https://www.python.org/downloads/> 了解如何在 Linux 系统上安装 Python。

2.2 相关工具/包的安装说明

2.2.1 Jupyter Notebook

如果你使用的是面向数据科学的集成 Python 环境，比如 Anaconda 或者 winpython，那么 Notebook 就已经安装好了。

如果你使用的单独安装或者系统自带的 Python 环境，可以这样来安装 Jupyter Notebook：

```
pip install jupyter
```

当然，Jupyter Notebook 为了获得更大的自定义便利，也可以通过源代码安装，这个就看使用者的个人习惯了，本书不做讨论。

2.2.2 jieba

可以直接通过 pip 来安装：

```
pip install jieba
```

2.2.3 NLTK

在 Linux、Mac 系统中可以直接通过 pip 来安装：

```
pip install nltk
```

在 Windows 中官方提供了打包好的二进制安装程序，可以直接在 <https://pypi.python.org/pypi/nltk> 下载安装。

2.2.4 Matplotlib

Matplotlib 也可以通过使用 pip 来进行安装：

```
pip install matplotlib
```

但是因为 Matplotlib 绘图会涉及到很多绘图系统类库，所以对于依赖要求较多，所以建议使用集成环境安装或者，在 Linux 系统中，可以使用系统包管理工具进行安装，例如在 Ubuntu/Debian Linux 发行版中，可以执行如下命令来安装：

```
apt install python-matplotlib
```

2.2.5 Numpy

可以通过 pip 直接安装：

```
pip install numpy
```

因为 Numpy 会依赖很多其他语言的科学计算类库，所以需要系统中有一些相关依赖包，可以根据安装反馈进行补充与安装，具体安装依赖软件的方式与操作系统有关。

2.2.6 Pandas

首先，可以通过 pip 来安装：

```
pip install pandas
```

但是 pip 安装 pandas 需要依赖 numpy 等需要涉及到编译的第三方包，所以需要电脑中有相关的编译工具环境以及依赖软件。

另外，一个简单的办法就是直接使用 Anaconda 基础环境，pandas 已经默认包含在其中了。

2.2.7 Gensim

Gensim 需要依赖 Numpy 与 Scipy，通过 pip 安装：

```
pip install gensim
```

2.2.8 scikit-learn

scikit-learn 需要依赖 Numpy 与 Scipy，通过 pip 安装可以直接安装，并且会安装相关依赖包：

```
pip install scikit-learn
```

2.3 Jupyter Notebook 快速入门

上一节已经讲解了相关工具的安装步骤，其中包括 Jupyter Notebook，本书后续涉及到的 Python 代码都是在该应用环境下运行的，所以本节在讲解 Python 基础知识之前，会先介绍 Jupyter Notebook 的界面、操作等基本内容。

2.3.1 启动 Jupyter Notebook 主界面

Jupyter Notebook 安装完毕后，打开命令提示符工具输入命令“jupyter notebook”并运行，即可在本地端口运行 Jupyter Notebook 服务并自动打开浏览器，如图 @todo 图编号 所示，由以下几个部分组成 @todo 补充 Jupyter Notebook 主界面截图 2.3.1-1 （1）文件列表选择区域：点击“Files”显示全部文件列表；点击“Running”显示正在运行的 Notebook；点击“Clusters”设置并行计算。（2）文件批量处理区域：有全部文件夹（Folders）、全部 Notebook（All Notebooks）、所有在运行 Notebook（Running）、所有文件（Files）四个选项。@todo 补充 截图 2.3.1-2 （3）文件列表：显示所有 Notebook 文件和文件夹。（4）上传、新建文件：点击 Upload 按钮可以上传本地文件；点击 New 按钮建立新文件，包括文本文件、文件夹、Python 文件等。@todo 补充 截图 2.3.1-3

2.3.2 创建一个 Python Notebook

点击主界面右上角 New 按钮，选择 Python2，创建一本新的 Notebook，新的 Notebook 会自动打开，Notebook 界面如图 @todo 图编号 所示，由以下几个部分组成：@todo 补充 Notebook 界面截图 2.3.2-1 （1）Notebook 名称：新建的 Notebook 名称默认为“Untitled”，直接点击名称即可重新命名。@todo 补充命名图片 2.3.2-2 （2）主工具栏：包含 File、Edit、View、Insert、Cell、Kernel、Help 七个按钮，提供保存、导出、重载以及重启内核等选项，如果想要详细了解 Notebook 或相关库的具体情况，可以使用 help 帮助菜单。（3）常用工具栏：从左至右依次为保存、下方插入空单元格（cell）、剪切选中的单元格、复制选中的单元格、粘贴单元格、上移单元格、下移单元格、运行选中的单元格并移至下一个单元格、中断运行、重新启动、单元格格式选择（代码、markdown、标题等）、打开命令面板、显示工具栏更新。（4）Notebook 的代码单元格：是 Notebook 的主要区域，使用者在单元格编辑指定格式的内容并运行输出相应的结果，每个 Notebook 由多个代码单元格构成，而每个单元格又可以有不同的用途。（5）状态栏：显示 Notebook 类型及运行状态。

2.3.3 上传文件

Jupyter Notebook 提供了非常便捷的文件上传功能，利用该功能使用者可以将分析过程中会使用到的数据上传到 Notebook 文件列表，在使用时直接调用即可。

点击界面右上角的 Upload 按钮，即可浏览并选择需要上传的文件 @补充上传文件操作截图

2.3.3-1

确认上传后即可在文件列表看到已经上传的文件 @todo补充文件列表截图 2.3.3-2

2.3.4 Jupyter Notebook 环境下 Python 的基本操作

本书在 1.3.1 部分已经对 Jupyter Notebook 做了介绍，Jupyter Notebook 支持多种程序语言，其中包括 Python。简单来讲，创建一个 Python 格式的 Notebook，就是为 Python 代码的运行提供了一个更加方便、友好的交互式环境，这种环境不会影响 Python 语言的语法及支持的数据结构等。下面用一些例子为读者介绍 Jupyter Notebook 以及 Jupyter Notebook 环境下 Python 的一些常用基本操作。

2.3.4.1 输入并执行代码

首先，选中待编辑的单元格，此时单元格周围会出现框线表示该单元格被选中；其次，在常用工具栏单元格格式选项中选择“code”（一般默认单元格格式即为“code”），设置单元格输入格式为 Python 代码，此时选中的单元格前端会显示“In []”，其中，“In”表示输入，运行单元格后“[]”中会显示数字。表示该单元格是第几个运行的单元格。

@todo 插入单元格截图 2.3.4.1-1

将光标移至单元格输入框内并单击，就可以编辑 Python 代码了。利用 Python 可以实现很多复杂问题，下面先从几个简单例子出发感受 Python 的魅力。

(1) 整数运算

Python 可被用作计算器，支持 4 种基本算术运算，即：+（加）、-（减）、（乘）和 /（除），此外，还可以使用 * 和 % 来分别表示乘方和求余。

例 1 简单加法计算 在单元格中输入“3+2”，点击运行按钮或者直接使用快捷键“Shift+Enter”运行程序，即可在单元格下方得到计算结果。

@todo 插入截图 2.3.4.1-2

可以看到，在输出结果前显示“Out [1]”，其中“Out”表示输出，“[]”内的数字表示是第几个输出结果。

例 2 乘方运算 运行后，会在已有单元格下方自动生成新的单元格；快捷键“Ctrl+Enter”运行程序则不会在下方自动生成新的单元格，使用者也可以点击常用工具栏中的“下方插入空单元格”按钮添加。

@todo 插入截图 2.3.4.1-3

在新的单元格输入算式并运行得到计算结果

@todo 插入截图 2.3.4.1-4

可以看到，与其他大多数编程语言不同，Python 对整数的长度没有限制，使用者可以执行数十位甚至数百、数千位的整数运算。

@todo 插入截图 2.3.4.1-5

(2) 语句

例 1 和例 2 给出的都是 Python 表达式，下面例 3 和例 4 给出的是 Python 语句，即给计算机指令做某件事情。

例 3 赋值语句 Python 可以给变量赋值并对已赋值变量进行操作。如将数值 8 赋值给变量 a：

```
In [4]: a=8
```

使用变量 a 进行计算

```
In [5]: a+5  
Out[5]: 13
```

例 4 print 语句 print 语句可以用于将字符串打印到屏幕：

```
In [6]:print " Text Mining with Python"  
Text Mining with Python
```

2.3.4.2 增加其他类型单元格

正像 notebook 的名称一样使用者可以以笔记的形式记录下编码的思维过程，因为除了代码外，Jupyter Notebook 提供了其他不同的单元格格式来丰富、美化 Notebook，如 Markdown，选定待编辑单元格后，在常用工具栏单元格格式选项中选择“Markdown”，在单元格中按照 markdown 语法输入需要的内容，对 Notebook 进行编辑排版，如输入标题“我的 notebook”

@todo 插入截图 2.3.4.1-6

运行后可以得到相应格式的输出

@todo 插入截图 2.3.4.1-7

2.3.4.3 导入模块

模块是 Python 最高级别的程序组织单元，它将程序代码和数据封装起来以便重复使用。简单来讲，将写好的一段代码进行分组，分别放到不同的文件中，这样的每一个文件都是一个模块，包含了一系列相关的函数和变量，要要使用模块需要先导入，导入后模块内的函数才可以被调用。Python 自带了很多实用模块，如 `math`、`cmath` 等，只要安装了 Python，都可以导入适当地模块来使用，这组模块称为标准库（standard library）。在 Python 中导入模块用 `import`，查看模块的所有函数用 `dir`（模块），调用模块里的函数用“模块.函数”。

例 5 导入 `math` 模块

比如要做乘方计算，需要调用 `math` 模块的 `pow` 函数。首先导入 `math` 模块：

```
In [7]:import math
```

列出 `math` 模块所有函数

```
In [8]:dir(math)
Out[8]:['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
        'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp',
        'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan',
        'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
        'sqrt', 'tan', 'tanh', 'trunc']
```

调用 `pow` 函数

```
In [9]:math.pow(2, 3)
Out[9]:8.0
```

模块 `math` 包含的函数及参数格式可以通过 `help` 函数查看：

```
In [10]:help(math)

Help on module math:

NAME
    math

FILE
    /home/datartisan/.pyenv/versions/2.7.12/envs/dataacademy-lesson-27/lib/python2.7/li
b-dynload/math.so

MODULE DOCS
    http://docs.python.org/library/math

DESCRIPTION
    This module is always available. It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

.....
DATA
    e = 2.718281828459045
    pi = 3.141592653589793
```

如果只需要用模块中的某个函数，可以通过“`from 模块 import 函数`”实现，使用这种方式导入的函数可以直接使用，而不用写模块名称。

例 6 利用 `math` 模块下函数进行计算

```
In [10]:sqrt(9)
NameErrorTraceback (most recent call last)
<ipython-input-27-c836dfef5db4> in <module>()
----> 1 sqrt(9)

NameError: name 'sqrt' is not defined

In [11]:from math import sqrt
In [12]:sqrt(9)
Out[12] : 3.0
```

如果想一次性引入 `math` 中所有的东西，可以通过“`from math import *`”实现

2.3.4.4 注释

为了增加程序的可读性和可维护性，有必要在编写代码时添加相应的注释。Python 中用符号“#”标记注释，注释部分的内容不会被 Python 解释。注释的位置没有限制，可以从行首开始，也可以在空白或代码之后，但是不出现在字符串中，字符串中的“#”仅表示 #。

例 7 添加注释

```
In [13]:#计算0.5的sin值  
math.sin(0.5)  
Out[13] : 0.479425538604203
```

2.3.4.4 一些快捷操作

在使用 Jupyter Notebook 过程中，掌握一些快捷操作可以大大提升工作效率，下面列举一些常用的快捷操作：

(1) 调出命令面板：Ctrl + Shift + P (2) 同时选择多个命令单元格：向下选择 Shift + J 或 Shift + Down 向上选择 Shift + K 或 Shift + Up (3) 合并命令单元格：Shift + M

2.4 Python 基础

2.4.1 Python 基本数据结构：序列

所谓数据结构，指数据的组织方式，这里的数据包括数值、字符等。Python 中最基本的数据结构是 **sequence**，翻译为“序列”，是一组按顺序排列的元素，每一个元素都有一个编号，对应元素在序列中的位置，第一个元素的编号为 0，以第一个元素为基准，向后第二个元素编号为 1，向前即倒数第一个元素编号为 -1，其他元素编号按照位置以此类推，这个编号称为索引。

Python 包含 6 中内建序列，常用的有字符串、列表和元组，本节后续内容将分别介绍这三种数据结构及相应的操作。

2.4.1.1 字符串（string）

(1) 什么是字符串 字符串即由数字、字母、标点符号等组成的一串字符，在 Python 程序中最主要的用法就是表示一些文本，主要使用单引号''、双引号""或者三引号"""表示字符串。三引号多用于表示非常长需要跨很多行的字符串或者包含多种特殊字符的字符串。

例 8 创建字符串

```
In [14] : 'Text Mining with Python'
Out[14] : 'Text Mining with Python'

In [15] : "Text Mining with Python"
Out[15] : 'Text Mining with Python'

In [16] : """Text Mining with Python"""
Out[16] : 'Text Mining with Python'
```

如果需要表示的字符串内包含双引号或者单引号，定义字符串时内外引号不得相同，即内有单引号，外用双引号，内有双引号，外用单引号，否则需要在引号前加“\”进行转义

```
In [17] : "there's a book"
Out[17] : "there's a book"

In [18] : 'there\'s a book'
Out[18] : "there's a book"
```

函数 **str()** 可以将其他数据结构转换为字符串

```
In [19]: str(23)
Out[19]: '23'
```

(2) 字符串拼接

所谓拼接，即将两个独立的字符串连接起来，创建一个新的字符串，直接用加号“+”即可实现。

例 9 拼接字符串

```
In [20]: "Text"+"Mining"
Out[20]: 'TextMining'

In [21]: a="Text"
          b="Mining"
In [22]: a+b
Out[22]: 'TextMining'
```

也可以对空格进行拼接

```
In [23]: "Text"+" "+"Mining"
Out[23]: 'Text Mining'
```

如果需要重复拼接一个字符串，可以利用乘号“*”实现。

```
In [24]: 5*"Text"
Out[24]: 'TextTextTextTextText'
```

(3) 常用字符串方法

方法，直观来讲就是针对于某个对象的函数，对象的类型没有严格限制，比如字符串、列表、字典都有对应的方法。当需要对某个对象进行某种方法操作时，可以通过“对象.方法（参数）”实现，与调用函数类似，只是“对象”被放在“方法”前，并用“.”隔开。

(3.1) join 和 split

join 和 split 互为逆方法，join 可以用来实现字符串的拼接，将多个字符串拼接为一个字符串；相反的，split 可以用来将一个长字符串切分为多个较短的字符串，并保存在一个列表中。下面结合实例来学习这两个方法的调用。

join：“拼接符号”.join（待拼接字符串）

例 10 使用空格“ ”对字符串“a”、“b”、“c”进行拼接，得到字符串“a b c”

```
In [25]: s1="a", "b", "c"
         " ".join(s1)
Out[25]:'a b c'
```

split：待划分字符串.**split**（划分符号）

例 11 将字符串“a b c”按照空格划分为多个字符串

```
In [26]: s2="a b c"
         s2.split(" ")
Out[26]:['a', 'b', 'c']
```

(3.2) **strip**

strip 方法用于去除字符串首尾两端的空格，调用方式为：字符串.**strip**（）

例 12 去除字符串两端空格

```
In [27]: "    Text Mining with Python    ".strip()
Out[27]:'Text Mining with Python'
```

(3.3) **find**

find 方法用于在较长的字符串中查找其子字符串，调用方式为：长字符串.**find**(目标子字符串)，返回目标子字符串最左端字母的索引。

例 13 查找子字符串“with”

```
In [28]:"Text Mining with Python".find("with")
Out[28]:12
```

(3.4) **lower**

lower 方法用于返回字符串对应的小写字母，调用方式为：字符串.**lower**()

例 14 返回“TEXT”的小写形式

```
In [29] :"TEXT".lower()
Out[29]:'text'
```

(3.5) replace

replace 方法用于查找并替换字符串中部分内容，调用方式为：字符串.replace（查找内容，替换内容）

例 15 将“with”替换为“abcd”

```
In [30]: "Text Mining with Python".replace("with", "abcd")
Out[30]: 'Text Mining abcd Python'
```

2.4.1.2 列表（list）

(1) 什么是列表

在实际分析过程中，列表相当于一个存储空间或者存储形式，使用者可以把要分析的数据读入到列表，也可以把分析结果添加到列表，列表用方括号“[]”括起，其中的元素用逗号分隔，元素可以是任何类型的值，如数字、字符串等，也可以是列表。

(2) 创建列表

(2.1) 直接创建列表

例 16 创建一个空列表

```
In [30]: list1=[]
In [31]: list1
Out[31]: []
```

例 17 创建一个包含数值、字符串元素的列表

```
In [32]: list2=[1, 2, "a", "b"]
In [33]: list2
Out[33]: [1, 2, 'a', 'b']
```

例 18 创建一个包含列表的列表

```
In [34]: list3=[[{"c": "a"}, {"r": "b"}]
In [35]: list3
Out[35]: [{"c": "a"}, {"r": "b"}]]
```

(2.2) 使用 list 函数创建列表

创建列表的另一个方法是用 `list()` 函数，即把括号里的序列转换成列表。

例 19 将字符串转换为列表

```
In [36]: list4=list("text mining")
In [37]: list4
Out[37]:['t', 'e', 'x', ' ', 'm', 'i', 'n', 'g']
```

从结果可以看出，`list` 函数将字符串的每一个元素都作为列表元素保存在列表中。

例 20 将元组转换为列表（元组的概念将在下一部分介绍）

```
In [38]: list5=list((1,2,3))
In [39]: list5
Out[39]:[1, 2, 3]
```

(3) 改变、删除列表元素

列表不同于其他序列（字符串、元组）的一个特点是：列表可以改变、删除其中的元素。

(3.1) 改变列表元素：列表[元素索引]=更改目标

例 21 将列表 `list5` 中元素“2”改为“6”

```
In [40]: list5[1]=6
In [41]: list5
Out[41]:[1, 6, 3]
```

(3.2) 删除列表元素：`del` 列表[元素索引]

例 22 删除 `list5` 中元素“3”

```
In [42]: del list5[2]
In [43]: list5
Out[43]: [1, 6]
```

(4) 常用列表方法

(4.1) 添加列表元素的方法：`append`、`extend`、`insert`

`append` 方法用于给列表末端增加新的元素，调用方式为：列表.append(待添加元素)

例 23 在 `list5` 末端增加元素

```
In [44]:list5.append(0)
In [45]:list5
Out[45]:[1, 6, 0]
In [46]:list5.append([0,1])
In [47]:list5
Out[47]:[1, 6, 0, [0, 1]]
```

`extend` 方法用于在列表末端同时添加另一个序列中的多个元素，调用方式为：列表.`extend`（待添加序列）

例 24 在 `list5` 末端添加 `list4` 中的元素

```
In [48]:list5.extend(list4)
In [49]:list5
Out[49]:[1, 6, 0, [0, 1], 't', 'e', 'x', 't', ' ', 'm', 'i', 'n', 'i', 'n', 'g']
```

`insert` 方法用于将指定元素插入到列表的指定索引位置，调用方式为：列表.`insert`（索引，待插入元素）

例 25 在 `list5` 索引位置 4 插入字符串“Python”

```
In [50]:list5.insert(4,"python")
In [51]:list5
Out[51]:[1, 6, 0, [0, 1], 'python', 't', 'e', 'x', 't', ' ', 'm', 'i', 'n', 'i', 'n', 'g']
```

(4.2) 删除列表元素的方法：`pop`、`remove`

`pop` 方法用于删除列表中指定索引位置的元素，默认认为最后一个元素，返回值是被删除的元素，调用方式为：列表.`pop`（待删除元素索引位置）

例 26 删除 `list5` 中最后一个元素

```
In [52]:list5.pop()
Out[52]:'g'
In [53]:list5
Out[53]:[1, 6, 0, [0, 1], 'python', 't', 'e', 'x', 't', ' ', 'm', 'i', 'n', 'i', 'n']
```

例 27 删除 `list5` 索引位置 2 的元素

```
In [52]:list5.pop(2)
Out[52]:0
In [53]:list5
Out[53]:[1, 6, [0, 1], 'python', 't', 'e', 'x', 't', ' ', 'm', 'i', 'n', 'i', 'n']
```

`remove` 方法用于删除列表中指定元值的元素，若有两个相同的元素，则删除第一个，调用方式为：`列表.remove(待删除元素值)`

例 28 删除 `list5` 中的字符串“t”

```
In [54]:list5.remove("t")
In [55]:list5
Out[55]:[1, 6, [0, 1], 'python', 'e', 'x', 't', ' ', 'm', 'i', 'n', 'i', 'n']
```

(4.3) index

`index` 方法用于查找列表中某一个元素的索引位置，若有多个相同的元素，则返回第一个元素的索引位置，调用方式为：`列表.index(待查找元素)`

例 29 查看 `list5` 中元素 `[0,1]` 和 “n”的索引位置

```
In [56]:list5.index([0,1])
Out[56]:2
In [57]:list5.index("n")
Out[57]:10
```

(4.4) 列表元素排序方法：sort、reverse

`sort` 方法用于将列表元素升序排列，调用方式为：`列表.sort()`

例 30 将列表 `list6=[9,3,6]` 元素升序排列

```
In [58]:list6=[9,3,6]
In [59]:list6.sort()
In [60]:list6
Out[60]:[3, 6, 9]
```

`reverse` 方法用于反转列表元素原有排序，调用方式为：`列表.reverse()`

例 31 反转列表 `list6` 元素排序

```
In [61]:list6.reverse()
In [60]:list6
Out[60]:[9, 6, 3]
```

(4.5) count

count 方法用于统计列表中某一元素出现的次数，调用方式为：列表.count（待计数元素值）

例 32 统计 list5 中元素 “n” 出现的次数

```
In [61]:list5.count("n")
Out[61]:2
```

2.4.1.3 元组 (tuple)

(1) 什么是元组

与列表不同，元组是不能修改的序列。元组用括号“（）”括起，其中的元素用逗号分隔，元素可以是任何值，如数值、字符串，或者序列。

(2) 创建元组

(2.1) 直接创建元组

例 33 创建一个空元组

```
In [62]: ()
Out[62]: ()
```

例 34 创建一个包含数值 1、2、3 的元组

```
In [63]: (1, 2, 3)
Out[63]: (1, 2, 3)
```

(2.2) 使用 tuple 函数创建元组

创建元组的另一个方法是用 tuple（）函数，即把括号里的序列转换成元组。

例 35 将列表 list5 转换为元组

```
In [64]:tuple1=tuple(list5)
In [65]:tuple1
Out[65]:(1, 6, [0, 1], 'python', 'e', 'x', 't', ' ', 'm', 'i', 'n', 'i', 'n)
```

例 36 将字符串“text mining”转换为元组

```
In [66]:tuple2=tuple("text mining")
In [67]:tuple2
Out[67]:('t', 'e', 'x', 't', ' ', 'm', 'i', 'n', 'i', 'n', 'g')
```

(3) 元组的用途

在处理文本数据时，元组结构用的不多，倒是一些函数或方法的返回值常以元组的形式存在，下面举一个简单的实例帮助读者理解：在做基于词性识别的情感分析时，采用 jieba 进行词性标注，结果保存在列表中，列表中的每一个元素都是一个元组，形如 pair(u'', u")，元组里的元素用 unicode 编码展示，前者为词汇，后者表示词性。

(4) 常用元组方法

相比于字符串和列表，元组的方法比较少。

(4.1) count

与列表一样，count 方法用于统计元组中某一元素出现的次数，调用方式为：元组.count(待计数元素值)

例 37 统计元组 tuple2 中字符串“t”出现的次数

```
In [68]:tuple2.count("t")
Out[68]:2
```

(4.2) index

与列表一样，index 方法用于查找元组中某一个元素的索引位置，若多个相同的元素，则返回第一个元素的索引位置，调用方式为：元组.index(待查找元素)

例 38 查找元组 tuple1 中字符串“n”的索引位置

```
In [69]:tuple1.index("n")
Out[69]:10
```

2.4.1.4 序列的通用操作

2.4.1.1 - 2.4.1.3 部分已经介绍了 Python 中三种基本序列类型（字符串、列表和元组）及其常用的方法，此外，序列还有一些通用的操作，包括索引、运算、计算长度等，下面将向读者一一介绍。

(1) 索引

本节开端已经介绍过，序列中每一个元素都有对应的索引位置，根据这个索引位置可以访问序列中相应的元素值。

例 39 访问序列某个元素：序列[索引位置]

```
In [70]:tuple1[0]
Out[70]:1

In [71]:"Text Mining with Python"[-2]
Out[71]:'o'
```

例 40 访问序列多个元素：序列[初始索引:结束索引]

```
In [72]:list5[2:5]
Out[72]:[[0, 1], 'python', 'e']

In [73]:"Text Mining with Python"[-5:-1]
Out[73]:'ytha'
```

可以看出，该种索引方式包括初始索引位置的元素，不包括结束索引位置的元素。

例 41 访问序列全部元素：序列[:]

```
In [74]:list6[:]
Out[74]:[6, 3, 9]
```

(2) 序列运算

(2.1) 序列相加

使用加法运算符号“+”可以对同类型序列进行连接

例 42 同类型序列链接

```
In [75]:list5+list6
Out[75]:[1, 6, [0, 1], 'python', 'e', 'x', 't', ' ', 'm', 'i', 'n', 'i', 'n', 6, 3, 9]

In [76]:"text"+"mining"
Out[76]:'textmining'
```

(2.2) 序列乘法

进行“ n^* 序列”的乘法操作，可以得到一个新序列，新序列为原序列重复 n 次。

例 43 创建重复序列

```
In [77]:"text"**3
Out[77]:'texttexttext'

In [78]:[1,2]**4
Out[78]:[1, 2, 1, 2, 1, 2, 1, 2]
```

(3) `len()`返回序列元素数量

例 44 统计序列元素数量

```
In [79]:len("text")
Out[79]:4

In [80]:len(list6)
Out[80]:3
```

(4) `max()`和`min()`返回序列中最大和最小元素

例 45 查找序列最大值和最小值

```
In [81]:max(list6)
Out[81]:9

In [82]:min(list6)
Out[82]:3
```

2.4.2 字典 (`dict`)

2.4.2.1 什么是字典

前面已经介绍过，不同的数据结构有不同的表示形式，列表用方括号“[]”括起，元组用圆括号“()”括起，而本部分将要介绍的字典，是用大括号“{}”括起的，并且里面的元素以“键（key）：值(value)”的形式成对出现，如 {"Lucy": "92", "David": "86", "Bob": "100"} 就是一个存储学生姓名及成绩的字典，学生姓名为“键”，成绩为“值”，中间用冒号“：“表示对应关系，每一对“键”和“值”之间用逗号隔开，可以通过学生姓名访问其成绩。直观来讲，字典就像小型得数据库，可以使用键高效地存储和检索数据。

2.4.2.2 创建字典

(1) 直接创建字典

例 46 创建一个空字典

```
In [83]: dict()
Out[83]: {}
```

例 47 创建学生成绩字典

```
In [84]: score={"Lucy": "92", "David": "86", "Bob": "100"}
In [85]: score
Out[85]: {'Bob': '100', 'David': '86', 'Lucy': '92'}
```

可以看出，键、值的顺序在字典中并不重要，只要保持对应关系，就不会影响字典的访问。

(2) 用 dict 函数创建字典

创建字典的另一个方法是用 `dict()` 函数，一种创建方式是：`dict([(key,value)])`，即用 `dict` 函数作用于以键、值构成的元组为元素的列表。

例 48 创建顾客电话号码字典

```
In [86]: tel=dict([('Alice', '123456'), ('Eve', '456789')])
In [87]: tel
Out[87]: {'Alice': '123456', 'Eve': '456789'}
```

另一种创建方式是：`dict(name=value)`。

例 49 创建病人年龄字典

```
In [88]:age=dict(Alex="15", Jack="33")
In [89]:age
Out[89]:{'Alex': '15', 'Jack': '33'}
```

2.4.2.3 基本字典操作

(1) 访问字典中键对应的值：字典[键]

例 50 查询字典 tel 中名为 Alice 的顾客的电话

```
In [90]:tel["Alice"]
Out[90]:'123456'
```

(2) 为键赋值：字典[键]=值

例 51 将字典 tel 中键 Alice 对应的值修改为 “654321”

```
In [91]:tel["Alice"]="654321"
In [92]:tel
Out[92]:{'Alice': '654321', 'Eve': '456789'}
```

例 52 为字典 tel 新增一个键值对：“Tracy”：“987654”

```
In [93]:tel["Tracy"]="987654"
In [94]:tel
Out[94]:{'Alice': '654321', 'Eve': '456789', 'Tracy': '987654'}
```

(3) 删除键值对：del 字典[键]

例 53 删除字典 score 中 Bob 的成绩

```
In [95]:del score["Bob"]
In [96]:score
Out[96]:{'David': '86', 'Lucy': '92'}
```

(4) 统计字典中键值对的个数：len(字典)

例 54 统计字典 age 中有多少个病人的年龄记录

```
In [97]:len(age)
Out[97]:2
```

2.4.2.4 常用字典方法

与上文介绍过其他数据结构类似，字典也有相应的方法，下面选取常用的字典方法进行简单介绍。

(1) get

get 方法用于访问字典中键对应的值，调用方式为：字典.get(键)

例 55 访问字典 score 中的值

```
In [98]:score.get("Lucy")
Out[98]:'92'

In [99]:print score.get("Jay")
None
```

第 99 条访问命令没有任何输出结果，这是因为字典 score 中不存在键 “Jay”，利用 print 得到 None 值。

(2) has_key

has_key 方法用于检查字典是否含有某个键，返回值“True”代表包含，“False”代表不包含。调用方式为：字典.has_key (键)

例 56 检查字典 tel 是否含有某些键

```
In [100]:tel.has_key("Eve")
Out[100]:True
In [101]:tel.has_key("X")
Out[101]:False
```

(3) items

items 方法的功能与 dict 函数相反，前文介绍过，dict 函数可以将包含（键，值）元组的列表转换为字典，**items** 方法则可以实现这一过程的逆操作，但是这一操作并不改变原有字典结构，只是返回相应的列表结果，如果有需要可以将结果赋给新的变量。调用方式为：字典.items ()

例 57 输出字典 tel 的列表形式

```
In [102]:tel.items()
Out[102]:[('Tracy', '987654'), ('Alice', '654321'), ('Eve', '456789')]
In [103]:tel
Out[103]:{'Alice': '654321', 'Eve': '456789', 'Tracy': '987654'}
```

(4) keys

keys 方法可以以列表形式返回字典的键，结果不包含重复的键，每个键有且只有一个，调用方式为：字典.keys（）

例 58 创建一个新字典 d，并查看字典包含的所有键

```
In [104]:d={"one":1,"two":2,"three":1}
In [105]:d.keys()
Out[105]:['three', 'two', 'one']
```

(5) values

与 **keys** 类似，**values** 方法可以以列表形式返回字典的值，但是与 **keys** 方法不同的是，**values** 方法返回的值可以重复，调用方式是：字典.values（）

例 59 返回字典 d 包含的所有值

```
In [106]:d.values()
Out[106]:[1, 2, 1]
```

(6) pop

pop 方法用于删除指定的键值对，返回值为指定删除的值，调用方式为：字典.pop（键）

例 60 删除字典 d 中 "one":1 键值对

```
In [107]:d.pop("one")
Out[107]:1
In [108]:d
Out[108]:{'three': 1, 'two': 2}
```

2.4.3 集合（set）

集合内包含了一系列元素，兼具列表和字典的某些特性：集合和列表一样，可以用于存储一系列元素，但是集合的元素没有重复，而且是无序的，这点和字典的键很像。在 notebook 中，集合和字典一样用大括号“{}”括起，括号中的元素用逗号分开。

2.4.3.1 创建集合

创建集合需要调用 `set()` 函数，调用方式是：`set(序列或其他可迭代对象)`

例 61

```
In [109]: set1=set("text")
In [110]: set1
Out[110]: {'e', 't', 'x'}
```



```
In [111]: set2=set([1,2,3])
In [112]: set2
Out[112]: {1, 2, 3}
```

2.4.3.2 常用集合方法与运算

(1) `union` 和 “|”

与数学中“集合”的概念类似，Python 中集合结构的数据也可以进行集合运算，如求并集、交集和差集。求集合的并集，可以使用 `union` 方法，或者直接只用“|”运算符，调用方式为：集合`1.union(集合2)` 或 集合`1|集合2`

例 62 求两个集合的并集

```
In [113]: set3=set([3,4,5])
In [114]: set2.union(set3)
Out[114]: {1, 2, 3, 4, 5}
In [115]: set2|set3
Out[115]: {1, 2, 3, 4, 5}
```

(2) `intersection` 和 “&”

求集合的交集，可以使用 `intersection` 方法，或者直接只用“&”运算符，调用方式为：集合`1.intersection(集合2)` 或 集合`1&集合2`

例 63 求两个集合的交集

```
In [116]: set2.intersection(set3)
Out[116]: {3}
In [117]: set2&set3
Out[117]: {3}
```

(3) difference 和 “-”

求集合的差集，可以使用 `difference` 方法，或者直接只用“-”运算符，调用方式为：集合`1.difference(集合2)` 或 集合`1-集合2`

例 64 求两个集合的差集

```
In [118]: set2.difference(set3)
Out[118]: {1, 2}
In [119]: set2-set3
Out[119]: {1, 2}
```

(4) add

`add` 方法用于给集合中添加新元素，调用方式为：集合`.add(待添加元素)`

例 65 为集合 `set2` 添加元素 4

```
In [120]: set2.add(4)
In [121]: set2
Out[121]: {1, 2, 3, 4}
```

(5) remove

`remove` 方法用于删除集合中的元素，调用方式为：集合`.remove(待删除元素)`

例 66 删除集合 `set2` 中的元素 4

```
In [122]: set2.remove(4)
In [123]: set2
Out[123]: {1, 2, 3}
```

@todo 补充数据结构的方法说明对照表 2.4.3.2

2.4.4 常用语句

2.4.4.1 if 条件语句

if 语句可以根据给定的条件，自动判断该是否执行指定的语句。if 条件语句的基本结构为：

if 条件 : 执行语句

if 语句以 if 开头，if 后面紧跟着判断条件，第一行以冒号结尾。执行语句部分需换行并缩进四个空格，在 Jupyter Notebook 中回车换行后会自动缩进光标。如果 if 后的条件为真，则执行冒号后的语句，如果 if 后的条件为假，则跳过该语句，永远不执行。

例 67 如果 "Eve" 是字典 tel 中的键，则打印其对应的值

```
In [124]:if "Eve" in tel:  
    print tel["Eve"]  
456789
```

还可以在 if 语句后增加 else 子句，在 if 条件不满足时提供其他选择：

if 条件 : 执行语句1 else : 执行语句2

也就是说，当 if 后的条件不满足时，执行语句2。

例 68 如果 "x" 是字典 tel 中的键，则打印其对应的值，否则打印 "no such key"

```
In [125]:if "x" in tel:  
    print tel["x"]  
else:  
    print "no such key"  
no such key
```

如果是有多个判断条件的复杂决策，可以在 if 语句后增加 elif 子句：

if 条件1 : 执行语句1 elif 条件2 :

执行语句2 else : 执行语句3

例 69 如果字典 tel 长度大于 3 ，打印 a ，如果字典 tel 长度小于 3 ，打印 b ，否则，打印 len tel = 3

```
In [126]:if len(tel)>3:
    print "a"
elif len(tel)<3:
    print "b"
else:
    print "len tel = 3"
len tel = 3
```

2.4.4.2 循环语句

循环语句用于重复执行满足循环条件的语句，Python 主要的两个循环语句分别为 `while` 循环和 `for` 循环

(1) `while` 循环

`while` 循环的基本结构为：

`while` 循环条件：执行语句

`while` 循环语句以 `while` 开头，后面跟循环条件，以冒号结尾。执行语句另起一行并缩进四个空格，如果满足循环条件的话则执行语句，并在此检查是否满足循环条件，直到不满足条件为止并跳出循环，执行循环语句后的其他语句。

例 70

```
In [127]:a=1 # 初始化语句，给变量 a 赋值 1
while a < 5: # 如果满足条件 a<5
    print a # 打印 a 值
    a+=1 # 递增语句，令 a=a+1，即将 a 的值加一
1
2
3
4
```

(2) `for` 循环

`for` 循环适用于为一组元素循环执行语句的情况，如一个序列中的所有元素，基本结构为：

`for` 循环变量 `in` 元素集合：执行语句

`for` 循环语句以 `for` 开头，后面跟循环变量，此处循环变量代指元素集合中的元素，常用变量 `i`、`j`、`k` 等表示，后跟关键字 `in` 和待循环的元素集合，第一行以冒号结束。执行语句同样需要换行并缩进四个空格，整个循环语句表示：对于元素集合中的每一个元素都执行第二行的语句。

例 71

```
In [127]:for i in [1,3,5]:
    print i
1
3
5
```

2.4.5 常用 Python 标准库模块

前面已经介绍过，在安装 Python 的时候也会获得一组有用的模块，即标准库，提供了丰富的外部函数为使用者自己的程序所用。本部分将为读者介绍一些常用的标准库模块。

2.4.5.1 time

time 模块下有很多函数可以实现对时间的处理，如转换常见日期格式、获取当前时间、格式化时间为字符串等，要查看 time 模块下所有的函数，可以执行 `dir(time)` 语句实现，下面列举几个常用函数及调用方法：

(1) time

函数 `time()` 可以返回当前时间，时间格式为新纪元开始后的秒数，常用来做日期运算，调用方式为 `time.time()`

例 72

```
In [128]:import time
In [129]:time.time()
Out[129]:1476239582.4023
```

(2) localtime

函数 `localtime()` 可以将当前时间（新纪元开始后的秒数）转换为元组结构的时间表示，调用方式为：`time.localtime(新纪元开始后的秒数)`，若参数为空，则默认为 `time()` 返回的当前时间。

例 73

```
In [130]:time.localtime()
Out[130]:time.struct_time(tm_year=2016, tm_mon=10, tm_mday=12, tm_hour=10, tm_min=29,
tm_sec=51, tm_wday=2, tm_yday=286, tm_isdst=0)
```

(3) asctime

函数 `asctime()` 可以将当前时间用字符串表示出来，调用方式为：`time.asctime(时间元组)`，若参数为空，则默认为 `localtime()` 返回的当前时间元组。

例 74

```
In [131]: time.asctime()
Out[131]: 'Wed Oct 12 10:33:14 2016'
In [132]: time.asctime(time.localtime(time.time()))
Out[132]: 'Wed Oct 12 10:34:35 2016'
```

(4) strftime

函数 `strftime()` 可以将时间元组转换为指定的字符串时间格式，调用方法为：`time.strftime(格式字符串, 时间元组)`，如果时间元组参数为空，则默认为 `localtime()` 返回的当前时间元组。

例 75

```
In [133]: time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
Out[133]: '2016-10-12 10:58:53'
```

除 `time` 模块外，Python 还提供了 `datetime`、`calendar` 等时间处理模块，读者可以通过 2.6.3 提供的官方文档详细阅读相关内容。

2.4.5.2 random

`random` 模块提供了多种分布的伪随机数生成函数，如返回指定区间的随机实数、随机返回序列中的元素等，要查看 `random` 模块下所有的函数，可以执行 `dir(random)` 语句实现，下面列举几个常用函数及调用方法：

(1) random

函数 `random()` 返回 0~1 之间的伪随机数，调用方式为：`random.random()`

例 76

```
In [134]: import random
In [135]: random.random()
Out[135]: 0.5336585385012483
```

(2) uniform

函数 `uniform()` 返回指定区间的平均分布的随机实数，调用方式为：`random.uniform(区间下限, 区间上限)`

例 77 返回 0~90 度之间的随机角度值

```
In [136]: random.uniform(0, 90)
Out[136]: 27.722651669371373
```

(3) choice

函数 `choice()` 可以随机返回指定序列中的元素，调用方式为：`random.choice(序列)`

例 78

```
In [137]: random.choice([1, 2, 3])
Out[137]: 1
```

(4) sample

函数 `sample()` 可以从指定序列中随机选择若干个不同元素，调用方式为：`random.sample(序列, 返回元素个数)`

例 79

```
In [138]: random.sample([1, 2, 3], 2)
Out[138]: [1, 3]
```

2.4.5.3 collections

上文介绍的字符串、列表、元组、字典等等都是 Python 内置的数据结构，`collections` 模块在这些内置数据结构的基础上，又提供了额外的特殊数据结构 `Container`，可以简单翻译为容器数据结构，该模块中定义了 `Counter`、`Hashable`、`Mapping`、`MutableSet` 等类，其中 `Counter` 是一个非常常用的计数器，可以用于计算序列中元素出现的次数，其返回值以字典结构存在，字典中键即为元素，值对应元素出现的次数，所以常用的字典方法也适用于 `Counter` 类对象。`Counter` 类实例化方式为：`实例=Counter(序列或其他可迭代对象)`

例 80

```
In [139]:from collections import Counter
In [140]:c=Counter("text mining with python")
          print c

In [141]:Counter([1,2,3,4,3,2,1])
Out[141]:Counter({1: 2, 2: 2, 3: 2, 4: 1})
Counter({'t': 4, ' ': 3, 'i': 3, 'n': 3, 'h': 2, 'e': 1, 'g': 1, 'm': 1, 'o': 1,
         'p': 1, 'w': 1, 'y': 1, 'x': 1})
```

Counter 对象还支持以下三种方法：

(1) elements

elements 方法可以按照 Counter 实例中各个元素出现的次数连续重复列举所有的元素，调用方式为：实例.elements()

例 81

```
In [142]:c=Counter([1,2,3,4,3,2,1])
In [143]:list(c.elements())
Out[143]:[1, 1, 2, 2, 3, 3, 4]
```

(2) most_common

most_common 方法可以计算 Counter 实例中各个元素出现的次数，并按照出现次数从大到小的顺序返回，调用方式为：实例.most_common(n)，其中参数 n 表示出现次数最多的前 n 个元素

例 82

```
In [144]:Counter("text mining with python").most_common(3)
Out[144]:[('t', 4), (' ', 3), ('i', 3)]
```

(3) subtract

subtract 方法可以对两个 Counter 实例进行减法运算，即对元素出现的次数相减，对于未出现的元素会出现计数为负数的情况。调用方式为：实例1.subtract(实例2)

例 83

```
In [145]:a=Counter([1,2,3,3,2])
    b=Counter([1,1,2,"t"])
    a.subtract(b)
In [146]:a
Out[146]:Counter({1: -1, 2: 1, 3: 2, 't': -1})
```

2.5 小结

本章从 Python 基本数据结构以及模块、相关工具安装和使用等几个方面向读者介绍了 Python 入门知识，并辅以相应的应用实例，通过学习本章内容，Python 初学者对 Python 应有较清晰认知并能独立书写、执行 Python 命令。现将本章内容总结如下：

- (1) Python：面向对象的编程语言，通过 Python 语言可以实现对计算机的控制，类似于 Perl、Ruby、Scheme、Java 等。
- (2) Python 解释器：一种翻译程序，能完成从编程语言到机器语言这一翻译过程。存在多种 Python 解释器，如 CPython、iPython、Jython 等。
- (3) 用 Python 进行文本数据分析的优势
- (4) Python 安装与使用
- (5) 相关工具/包的安装：Jupyter Notebook、jieba、NLTK、Matplotlib、Numpy、Pandas、Gensim、scikit-learn 等。
- (6) Jupyter Notebook 的使用：启动主界面、创建 Python notebook、Jupyter Notebook 基本操作
- (7) Python 基础：数据结构（序列、字典、集合）、常用语句（条件语句、循环语句）、常用 Python 标准库模块（time、random、collections）

2.6 深度阅读材料

对于未能做细致解释说明的内容，笔者将部分参考文档和资源网站等集中列于本节，有需要的读者可以自行选择阅读。

2.6.1 相对于 **2.0** 版本，**Python 3.0** 具体更新的内容，可以阅读官方文档《**What's New In Python 3.0**》，网址为：<https://docs.python.org/3/whatsnew/3.0.html>。

2.6.2 关于 **Python** 版本选择的问题，可以阅读官方文档《**Should I use Python 2 or Python 3 for my development activity?**》了解具体内容，网址为：<https://wiki.python.org/moin/Python2orPython3>。

2.6.3 Jupyter Notebook 官方网站 <http://jupyter.org/> 提供了详细的安装说明及相关资源、文档等，有需求的读者可以进行选择性阅读。

2.6.4 Python 基本数据结构官方文档网址：

(1) 序列：<https://docs.python.org/2/library/stdtypes.html#sequence-types-str-unicode-list-tuple-bytearray-buffer-xrange> (2) 字符串：<https://docs.python.org/2/library/string.html>
(3) 集合：<https://docs.python.org/2/library/sets.html> (4) 数据结构汇总：<https://docs.python.org/2/tutorial/datastructures.html>

2.6.5 想深入学习常用的 **Python** 模块可以阅读官方文档，以下是相关模块及文档网址：

(1) 时间处理模块：time 模块：<https://docs.python.org/2/library/time.html> datetime 模块：<https://docs.python.org/2/library/datetime.html#module-datetime> calendar 模块：<https://docs.python.org/2/library/calendar.html#module-calendar> (2) random 模块：<https://docs.python.org/2/library/random.html> (3) collections 模块：<https://docs.python.org/2/library/collections.html>

2.7 练习

2.7.1 启动 Jupyter Notebook 主界面，创建一个名为“练习”的 Python Notebook

2.7.2 熟悉 Python 中的常用运算符，利用 Python 作为科学计算器，并分别求出：

- (1) $22+30/5-24$ (2) $(123/25)^*8$

2.7.3 利用 math 模块进行数学计算，分别求出：

- (1) 以 10 为底 2 的对数，即 $\log_{10} 2$ @todo 替换公式 (2) 0.6 的 sin 和 cos 值 (3) -7 的绝对值

2.7.4 请判断以下语句的输出结果

```
age = 19 if age <= 8: print 'kid' elif age >= 18: print 'adult' else: print 'teenager'
```

2.7.5 输出列表 names = ['Bob','Tracy','Michael'] 中的所有名字

2.7.6 使用 for 循环语句计算 1-10 的整数之和

2.7.7 创建一个字典，将以下姓名和成绩保存到字典中：

names = ['Bob','Tracy','Michael'] scores = [92, 60, 83] 尝试使用列表 index 方法实现在两个列表之间查询学生成绩

2.7.8 使用符号“/”对字符串“a”、“b”、“c”进行拼接

2.7.9 将字符串 "text mining with Python" 按照空格划分为多个字符串

2.7.10 将字符串 "Text Mining with Python" 转换为小写形式

2.7.11 将序列 **[1,2,3,4]** 和 **[3,4,5,6]** 转化为集合数据结构，并进行并集、交集、差集运算

2.7.12 生成 **0~360** 度之间的随机角度值

2.7.12 统计序列 **["a","b","c","a","a","c","d"]** 中各个元素出现的次数，并列出出现次数最多的两个元素

第 3 章 文本数据来源

俗话说，巧妇难为无米之炊，掌握了自然语言处理、文本数据分析的理论知识，会使用相关的分析工具，没有用于分析的文本数据对象也是无法开展文本数据分析的。概括来讲，文本数据来源有三个方面：一是从公开的文本数据库下载所需的文本数据，二是日常业务等留存在本地的文本资料，此外，就是根据实际分析需求从网络抓取文本数据，本章将从这三个方面出发，分别向读者介绍相应的文本数据获取资源及方式。本章具体内容安排如下：3.1 节为读者列举了比较常用的公开文本数据库，均附有网址及简单说明；自有数据的说明放在 3.2 节；3.3 节重点介绍网络文本数据抓取相关内容，包括网络文本数据抓取的概念、原理、流程、实例等；最后，3.4 节讲解了如何在 Python 语言下读取所获得的文本数据以便进行后续的分析流程。

3.1 公开数据源

公开的文本数据资源很多，范围广泛内容丰富，可以满足不同的分析研究需求。下面列举一下较为常用的公开文本数据资源，其中大部分是可免费使用的。

3.1.1 nltk 提供的数据集

http://www.nltk.org/nltk_data/

作为世界知名的自然语言处理工具平台，NLTK（Natural Language Toolkit）不但打造了一系列可用的文本分析工具，也为其使用者提供了一份用于工具学习与文本数据分析的样例数据集，这其中不乏例如WordNet、布朗语意库（Brown Corpus）、路透社语料库（The Reuters-21578 benchmark corpus）、古腾堡项目电子文本档案节选（Project Gutenberg Selections）等近百份精品素材。另外，除了通过浏览网站下载，我们还可以在利用 NLTK 库中的 download 函数来直接下载这些语料素材，这为初学者学习文本数据分析提供了很大的便利。

3.1.2 20 Newsgroups 数据集

<http://qwone.com/~jason/20Newsgroups/>

20 Newsgroups 数据集整合了分别来自于20个不同网络新闻组的约20000个新闻组文档，并被广泛的应用于如今学界流行的机器学习文本聚类或分类研究中。除了通过浏览网站下载，我们还可以在 Python 中通过 sklearn 库直接导入这份数据集。

3.1.3 哈工大讯飞

<http://hfl.iflytek.com/chinese-rc/>

中文阅读理解语料集，其中内容包括人民日报与儿童童话

3.1.4 搜狗实验室

<http://www.sogou.com/labs/>

由搜狗团队组建的权威中文信息处理数据提供和评测平台，该平台对外免费共享了多项互联网语料与评测集合数据，分别对应有大小不同的版本，其中包含了2012年来自搜狐以及其他多家新闻站点的近20个栏目的分类新闻数据，同时我们也能在平台上下载到搜狗团队积累的互联网词库（2006年版本），用于提高在对中文互联网文本数据进行分析时的针对性。

3.1.5 数据堂

<http://datatang.com/>

通过获取线下大数据、行业大数据以及政府大数据，数据堂整合了涵盖科技、信用、交通、医疗、卫生、通信等数十大领域的大规模数据，为客户提供专业数据采集处理、共享交易及数据云服务。数据堂将所有数据分为语音识别、健康医疗、交通地理、电子商务、社交网络、图像识别、统计年鉴、研发数据共八个类别，每个类别下都有文本格式数据，部分免费。

3.1.6 The Blog Authorship Corpus

<http://u.cs.biu.ac.il/~koppel/BlogCorpus.htm>

The Blog Authorship Corpus 是一个英文博文语料库，包括 19320 名博主一个月内发布的所有博文，总计 681288 篇，超过一亿个英文词汇。每一篇博文都保存为独立的文件，至少涉及 200 个常用英文词汇，并提供相应博主的性别、年龄、职业等信息。

3.1.7 CLiPS

<http://www.clips.uantwerpen.be/datasets>

CLiPS (Computational Linguistics & Psycholinguistics) 是隶属于安特卫普大学语言学系的研究中心，前身是 CNTS 和 CPL 研究中心，其官方网站提供了 TwiSty Corpus、CSI corpus、Personae corpus 等语料数据集。

3.1.8 Lemur

<http://lemurproject.org/>

Lemur 项目致力于开发搜索引擎、浏览器工具栏、文本数据分析工具等来支持信息检索与文本挖掘软件的研究开发，该项目提供了两个文本数据集：ClueWeb09 和 ClueWeb12，ClueWeb09 数据集包含十亿网页，涉及十种语言；ClueWeb12 包含超过七亿的英文网页，是 ClueWeb09 进一步发展的结果。

3.1.9 Project Gutenberg

http://www.gutenberg.org/wiki/Main_Page

Gutenberg 提供了超过五万本免费电子书资源，支持在线阅读和下载。

3.1.10 Machine Comprehension Test (MCTest)

<http://research.microsoft.com/en-us/um/redmond/projects/mctest/index.html>

MCTest 包含 660 篇阅读理解文章，通常用于机器语意理解的研究，可以免费使用。

3.1.11 Saudi Newspapers Corpus

<https://github.com/ParallelMazen/SaudiNewsNet>

包含 31030 篇阿拉伯新闻报道，摘自多个在线沙特报刊。

3.1.12 NLPIR 语料库

<http://www.nlpir.org/?action-category-catid-28>

提供了较为丰富的中文语料资源，包括微博语料、中文新闻分类语料、中文情感挖掘语料等。

3.2 自有数据

随着互联网技术的快速发展以及数据传输、存储效率的飞速提升，不论是个人在日常生活工作中，还是政府、企业在经营运作过程中，都会产生并积累大量的数据，比如，个人博客中的文章、社交网络通信文本，企业客户资料、协议合同、顾客反馈意见以及政府文件等，这些自有数据中包含了大量的有用信息，是不可忽略的重要文本数据来源之一。对个人文章或社交网站留言进行分析可以了解个人的关注热点及情感倾向；对企业客户文本资料进行分析，有助于深入了解客户，协助客户关系管理；对顾客反馈意见进行分析，可以识别顾客关注热点，挖掘顾客需求，有助于客户细分，实现精准营销，改进产品设计和服务等。可以说，利用好自有文本数据资源是推进文本数据分析工作的第一步，自有数据的重要性不容忽视，企业在日常经营过程中应重视数据的整理和积累。

3.3 网络抓取数据

在进行某些特定的文本数据分析时，不论是自有数据还是公开的文本数据资源都无法提供满足分析需求的文本数据，比如需要分析某一社交网站上关于最近某一热门话题的评论文本，或者需要分析某一电子商务平台上某一类商品的顾客评价文本，这些文本数据都存储在网页上，并没有现成的数据资源，通过人工获取的方式费时费力，并不可行，这时就要借助网络文本数据抓取技术来获取所需文本。

3.3.1 什么是网络数据抓取

网络数据抓取即依赖网络爬虫技术获取所需数据的数据收集方式。网络爬虫（Web crawler），也被称为网络蜘蛛（Web spider），根据字面意思，可以将互联网就比作一张大网，网络爬虫就是在网上爬行的蜘蛛，遇到资源就会抓取下来。严格来讲，网络爬虫是指，按照一定的规则自动的抓取互联网信息的程序或者脚本，可以实现对网络内容的自动浏览，对于能够访问的页面内容均可自动采集，进而获得所需数据资料。

3.3.2 网络爬虫基本原理

从网站某一个页面开始，读取网页内容，并找到网页中其他链接地址，通过链接地址访问新的网页并读取网页内容复制到本地，此过程不断循环直至所有的网页都抓取下来为止。

3.3.3 网络数据抓取流程

1. 明确需要抓取数据源的 URL
2. 构造 HTTP 请求获取对应 URL 的返回响应，一般这样的返回结果为 HTML 文档
3. 通过解析得到的 HTML 或者其他格式的文本信息，获得其中的所需的数据
4. 处理保存这些数据

这样就是一个典型的网络数据抓取流程

3.3.4 爬虫相关包

在 Python 中与爬虫有关的包有很多，比如标准库中的 urllib 与 urllib2 都提供了很多方法来方便的处理网络抓取中的各种问题

另外第三方包中，requests 是一个非常好用的处理网络请求方面的包，封装优美，调用灵活。

另外，不能不提的有 Python 中爬虫方面的集大成框架包 Scrapy，Scrapy 是一个用来开发爬虫程序的框架，提供了用以做网络爬虫方方面面的支持与工具，适合用来做比较大型的、流程复杂的，需要大规模部署使用的爬虫程序。

3.3.5 数据存储

一般而言，因为我们在抓取前已经计划好抓取的数据形式，所以，一般我们使用普通的关系型数据库来保存抓取并解析处理好的数据，比如使用 MySQL，当然如果数据量不大，也可以直接使用 SQLite 来进行数据保存，这样不管在数据的写入、读取还是查询方面都很方便。

3.3.6 爬虫效率的提升

爬虫程序的运行过程中，最主要效率消耗就是在网络 IO 过程中的等待，所以提升爬虫效率，最重要点就是提高爬虫程序的网络 IO 效率，但是我们知道网络请求的延时是客观网络环境造成的，很难从硬件上立即提升，所以一般提升爬虫的效率都是从提高爬虫的执行体的并发度这个角度进行的。

可以这样理解这个思路，网络 IO 的等待过程中，计算机是闲置的，如果我们能利用起来这个计算机的空闲，让各个执行体的等待时间重叠，这样就提升了整体的处理效率，也就是串行执行变为并行执行。

具体可以使用的方法有很多，比如可以将爬虫程序多进程启动，利用多线程模型来开发爬虫程序，或者利用 Python 中的协程的特性来提高程序的并发度，另外也可以直接使用一些高性能网络库来提高抓取效率，比如 Gevent、Twisted 等等。

3.3.7 网络数据抓取实例

3.3.7.1 爬取静态网站

抓取静态网站比较简单，因为网页的全部内容会随着请求相应返回，直接处理相应的返回结果，解析获得数据。

3.3.7.2 爬取异步动态网站

因为现在很多网站因为客户体验的需要，很多网页内容的加载是异步进行的，也就是一次的请求无法获得完成网页内容，完整的网页内容需要页面中的 JavaScript 代码在浏览器中运行并获取其他数据后才能得到。

对于这样的类型的网页处理的办法会根据目标网页的具体情况来定夺。比较通用的办法是通过无头浏览器（Headless Browser）进行渲染后，得到最终结果的 HTML 后在进行页面解析处理。一般用的比较多的无头浏览器是 PhantomJS。

3.4 读写文本数据

不论是以何种方式获取的文本数据，在进行分析之前，都要读取到分析环境中。Python 提供了必要的函数和方法进行文件基本读写操作：利用 `open` 函数打开文件，创建 `file` 对象，通过 `file` 对象方法来实现大部分的文件操作。

3.4.1 open 函数

`open` 函数用来打开文件，并返回 `file` 对象，最简单的调用方式为：`open(file, mode)`。

参数说明：

(1) `file`：要打开的文件路径+文件名，若是在当前目录下，只需传入文件名，且该参数是 `open` 函数唯一强制传入的参数 (2) `mode`：文件打开模式，参数取值选项及含义见下表：
@todo 插入参数说明表

常用的模式组合有：

@todo 插入模式组合表格 3.4.1

3.4.2 file 对象方法

Python 为 `open` 函数返回的 `file` 对象提供了一系列方法，让使用者更轻松的对文件进行操作

3.4.2.1 write

以写模式打开的文件，可以通过调用 `write()` 方法将任何字符串写入文件，且不会在字符串的结尾添加换行符('\'n')，调用方式为：`file.write(string)`

例 在当前目录下创建新文件并写入

```
In []:f=open("test.txt","w") # 以只写模式打开文件 test.txt
        f.write("text""\n""mining""\n""with""\n") # 在文件中写入字符串并换行 ("\"n"为换行符)
        f.close()
```

最后一行为文件对象的 `close` 方法，此处调用该方法是因为：在此我们写文件时，系统往往不会立刻把数据写入磁盘，而是放到内存缓存起来，空闲的时候再慢慢写入，通过调用 `close()` 方法，系统会把没有写入的数据全部写入磁盘，此时打开文件才能看到写入的内容，如果不执行 `close` 方法，很可能打开文件后看不到任何内容。后续在 3.4.2.5 部分会进一步介绍该方法。

执行完以上语句后，在 notebook 文件列表中即可看到新建的 test.txt 文件，打开后文件内容如下如所示：

@todo 插入文件截图 3.4.2.1-1

例 在已有文件末端追加内容

```
In []:f=open("test.txt","a")
f.write("python"\n")
f.close()
```

执行完以上语句后，在 notebook 文件列表中重新打开 test.txt 文件，可以看到最后一行追加了“python”字符串

@todo 插入文件截图 3.4.2.1-2

例 打开已有文件并写入新内容覆盖原内容

```
In []:f=open("test.txt","w")
f.write("1\n""2\n""3\n")
f.close()
```

执行完以上语句在 notebook 文件列表中重新打开 test.txt 文件，可以看到之前写入的内容已经被覆盖了：

@todo 插入文件截图 3.4.2.1-3

3.4.2.2 read

已经存在的并以读模式打开的文件，可以调用 `read()` 方法读取文件内容，文件内容以一个字符串的形式返回，调用方式为：`file.read(size)`,参数 `size` 为可选数值型参数，省略或为负数时将读取全部文件内容，除非文件内容超过内存。

例 读文件

```
In []:f=open("test.txt")
f.read()
Out[]:'1\n2\n3\n'
In []:f.close()
```

3.4.2.3 readline

`readline` 方法每次读取文件的一行，若返回空字符串则表示已经读取到文件结尾，调用方式为：`file.readline()`

例

```
In []:f=open("test.txt")
      f.readline()
Out[]:'1\n'
In []:f.readline()
Out[]:'2\n'
In []:f.readline()
Out[]:'3\n'
In []:f.readline()
Out[]:''
In []:f.close()
```

3.4.2.4 `readlines`

`readlines` 方法读取文件的所有行，并返回一个字符串列表，调用方式为：`file.readlines()`

例

```
In []:f=open("test.txt")
      f.readlines()
Out[]:[ '1\n', '2\n', '3\n']
In []:f.close()
```

如果需要将文件内容读取到列表中，除了调用 `readlines` 方法外，还利用 `list()` 函数处理 `file` 对象：

例

```
In []:f=open("test.txt")
      l=list(f)
      print l
      ['1\n', '2\n', '3\n']
In []:f.close()
```

3.4.2.5 `close`

在以上的示例中，多次出现 `close` 方法，在反复调用 `write()` 方法来写入文件的同时，也务必要调用 `close()` 来关闭文件。这是因为，当我们写文件时，系统往往不会立刻把数据写入磁盘，而是放到内存缓存起来，空闲的时候再慢慢写入，调用 `close` 方法后，操作系统才能保证把没有写入的数据全部写入磁盘，如果忘记调用 `close` 方法，可能会导致只写了一部分数据到磁盘。此外，打开的文件对象会占用系统的资源，系统同一时间能打开的文件数量也是有限的。所以要适时调用 `close` 方法，调用方式为：`file.close()`

但是如果每次读写文件都要执行 `close` 方法，不仅麻烦也容易忘记，幸运的是，Python 的 `with` 语句可以很好的解决这一问题，它会在语句执行完后，自动执行 `close()`，避免了不断调用 `close` 方法的麻烦，使用 `with` 语句更加方便简洁也更加保险。调用方式为：

`with open(file,mode) as *: 执行语句`

参数说明：

(1) `file`：要打开的文件路径+文件名 (2) `mode`：文件打开模式 (3) `*`：文件变量，可以自行指定变量名称

例

```
In []:with open("test.txt","r") as f:  
    data=f.readlines()  
    print data  
['1\n', '2\n', '3\n']
```

3.4.3 csv 模块

CSV 即逗号分隔值（Comma-Separated Values），是一种以纯文本形式存储表格数据的存储形式。CSV 文件由任意数目的记录组成，记录间以某种换行符分隔；每条记录内部常以逗号分隔，在 excel 中显示为不同列。

比如，现在有一份以 CSV 格式存储的数据 `data.csv`

@todo 插入csv数据截图 3.4.3-1

将 `data.csv` 上传到 Jupyter Notebook 文件列表中，打开文件可以看到 CSV 文件中有四条记录，每条记录都换行存储，记录内用逗号分隔，这些分隔的内容在 excel 中显示为三列。

@todo 插入文件截图 3.4.3-2

按照本节前半部分介绍的方法直接使用 `open` 函数打开文件

```
In []:with open("data.csv") as f:
    for i in f:
        print i
a,b,c
1,2,3
4,5,6
7,8,9
```

去除数据之间的逗号需要逐行处理

```
In []:data=[]
with open("data.csv") as f:
    for i in f:
        row = i.split(',')
        data.append(row)
In []:data
Out[[['a', 'b', 'c\r\n'],
      ['1', '2', '3\r\n'],
      ['4', '5', '6\r\n'],
      ['7', '8', '9\r\n']]
```

我们可以利用以上方式实现 CSV 文件的处理，但是如果元素内部包含逗号或者某些字段包含引号，仍然使用这种方法就需要处理较多的细节问题，而不能更专注于后续的分析过程，幸运的是 Python 提供了内建 csv 模块，实现 csv 文件的读写和处理，csv 模块的两个主要函数分别为 reader 和 writer，分别用于读取和写入 CSV 文件数据，要查看 csv 模块下所有的函数，可以执行 dir(csv) 语句实现，下面主要说明 CSV 格式文件的读取和写入。

3.4.3.1 reader

reader 函数可以遍历给定文件的每一行，每一行都读取为一个字符串列表，最终返回一个 reader 对象，调用方法为：csv.reader(csvfile, dialect='excel', **fmtparams)

参数说明：

csvfile：需要处理的文件，必须是可迭代的对象，可以是 file 文件对象或者 list 列表对象，如果是 file 对象，要以二进制模式 "b" 打开。**dialect**：可选参数，用于设定编码风格，默认为 excel 的风格，也就是用逗号分隔。另外，csv 模块也支持 excel-tab 风格，也就是制表符(tab) 分隔。其它的方式需要自行定义，此处不做详细说明。**fmtparam**：格式化参数，用来覆盖之前 dialect 对象指定的编码风格。

例

```
In []:import csv
with open("data.csv","rb") as cf:
    rows=csv.reader(cf)
    for row in rows:
        print row
['a', 'b', 'c']
['1', '2', '3']
['4', '5', '6']
['7', '8', '9']
```

首先，导入 `csv` 模块，利用 `open` 函数打开欲读取的“`data.csv`”文件，返回了文件对象 `cf`；接着给 `csv.reader()` 传入了第一个参数，即文件对象名称，另外两个参数采用默认值，即以 `excel` 风格读入；最后 `reader()` 函数返回一个 `reader` 对象 `rows`,`rows` 是一个列表，逐行打印列表中的元素即可得到以上输出结果。

3.4.3.2 writer

`writer` 函数用于将待输入的数据转换为分隔的字符串并写入 `csv` 文件，返回一个 `writer` 对象，调用方式为：`csv.writer(csvfile, dialect='excel', **fmtparams)`，参数解释说明参见上方 `reader` 函数参数说明。

`writer` 对象有两个方法可以用于文件内容的写入：`writerow` 和 `writerows`，`writerow` 用于写入一行，`writerows` 用于写入多行。

例 利用 `writerow()` 方法写入一行内容

```
In []:with open("data.csv","wb") as cf:
    mywriter=csv.writer(cf)
    mywriter.writerow(["A", "12", "34"])
In []:with open("data.csv","rb") as cf:
    rows=csv.reader(cf)
    for row in rows:
        print row
['A', '12', '34']
```

首先，利用 `open` 函数以写模式打开当前路径下的名字为“`data.csv`”的文件，若不存在这个文件，则创建它，并返回文件对象 `cf`；接着给 `csv.writer()` 传入了第一个参数，即文件对象名称，另外两个参数采用默认值，即以 `excel` 风格写入，返回 `writer` 对象 `mywriter`；利用 `writerow()` 方法写入一行内容；最后重新打开当前路径下的名字为“`data.csv`”的文件，可以看到原来文件的内容已经被覆盖。

例 利用 `writerows()` 方法写入多行内容

```
In []::with open("data.csv", "wb") as cf:  
    mywriter=csv.writer(cf)  
    mywriter.writerows([["A", "12", "34"], ["B", "45", "78"]])  
In []::with open("data.csv", "rb") as cf:  
    rows=csv.reader(cf)  
    for row in rows:  
        print row  
['A', '12', '34']  
['B', '45', '78']
```

3.5 小结

继文本数据分析理论知识和 Python 入门之后，本章向读者介绍了文本数据的三个来源：一是从公开数据源获取数据，读者可以在 3.1 节找到几个常用的公开数据源；二是充分利用自有数据；三是利用网络爬虫按照需求制定相应的规则来抓取所需的文本数据，未接触过网络爬虫的读者可以仔细阅读 3.3 节的内容，学习网络爬虫的原理及实现。此外，本章最后向大家说明了 Python 文本数据的读写操作，并特别讲解了 CSV 格式数据的读写模块，掌握这些内容有助于后续数据的快速熟练读取和处理。到本章为止，文本数据分析基础知识部分已经介绍完毕，后面将正式进入文本数分析的实践部分，准备好了吗？

3.6 深度阅读

3.6.1 更多关于 **file** 对象的读写操作可以阅读官方文档进一步了解，文档网

址：<https://docs.python.org/2/tutorial/inputoutput.html#reading-and-writing-files>

3.6.2 csv 模块 **reader** 函数 **dialect** 和 **fmtparams** 参数的详细说明，可参阅

<https://docs.python.org/2/library/csv.html#dialects-and-formatting-parameters>

3.7 练习

3.7.1 熟悉常用公开文本数据源，尝试收集更多的文本数据资源，下载并查看部分数据。

3.7.2 随机选取百度贴吧的某一帖子进行抓取，尝试只抓取楼主发帖内容，并将抓取到的内容保存到文件。

3.7.3 抓取新浪新闻中心 <http://news.sina.com.cn/> 最近一周的体育新闻

3.7.4 上传 **txt** 格式文件到 **Jupyter Notebook** 文件列表，在只读模式下打开文件并打印文件的前五行，并在文件末尾添加新内容。

3.7.5 在 **Jupyter Notebook** 文件列表新建 **CSV** 格式文件，一次写入多行内容。

3.7.6 上传 **CSV** 格式文件到 **Jupyter Notebook** 文件列表，在只读模式下打开文件并打印文件的后两行。

第 4 章 分词与词性标注

只有让机器理解我们的文字，才能利用机器进行文本数据分析，而这样的过程也是自然语言处理中的重要一环。自然语言处理过程一般包括分词、词性标注、命名实体识别、句法分析等，其中分词是自然语言处理的基础，是文本数据结构化处理的关键步骤，搜索引擎、机器翻译、自动摘要生成等技术都涉及到分词技术，合理的分词结果对于后续的文本数据分析效果至关重要，所以说分词是文本数据分析过程中不容忽视的重要环节。词性标注是给句子中每个词一个词性类别的任务，同样属于自然语言处理中的基本操作，是信息检索等领域不可或缺的步骤。鉴于中英文分词原理的差异，本章将分别进行说明，并结合实例向读者介绍多种分词、词性标注工具的使用方法，并对比不同工具的效果。

本章具体内容安排如下：4.1 节围绕中文相关处理展开，包括 `jieba`、`Yaha`、`Genius`、`finalseg`、`scseg`、`pynlpir` 等工具包的分词及词性标注功能的实现，4.2 节则主要介绍英文文本处理工具包 `NLTK`。

4.1 中文分词与词性标注

从形式上看，每一段中文文本都可以被认为是由汉字与标点符号组成的一个字符串。由字可组成词，由词可组成词组，由词组可组成句子，进而由一些句子组成段、节、章、篇。在我们的日常生活中，一个中文文本或是几个汉字的组合能够具有被赋予多种含义，作为人类，我们通常可以很容易地根据语境或是场景来对不同文本的语义进行理解，并从中分离出具有实际意义的词语，例如我们能够一眼看出“小明吃了水果然后去超市”这段话中并不含有“果然”一词的语义，但对于计算机来说，能做到这一点则需要更多复杂的工作。在接下来的内容中，我们将向大家介绍几类 Python 中常用的中文分词与词性标注工具，这些工具结合词库与算法在很大程度上解决了计算机的中文分词问题，掌握它们的使用方法，能给我们的文本数据分析带来极大的便利。

4.1.1 jieba

4.1.1.1 了解 jieba

(1) 分词及词性标注原理

对于给定的待分词文本，`jieba` 主要分词处理思路如下：

(1.1) 加载模块自带 `dict.txt` 词典，从自带的词典中构建待分词的语句的有向无环图 (`DAG`)；

(1.2) 对于词典中未收录的词，使用 `HMM` 模型的 `Viterbi` 算法尝试分词处理；

(1.3) 已收录词和未收录词全部分词完毕后，根据有向无环图的最大概率路径使用 Python 的 `yield` 语法生成一个词语生成器，逐词语返回。

词性标注的基本原理可以概括为：对于需要标注的词，如果词典中包括该词，就从词典中读取该词的词性；如果词典中没有该词，则用 Viterbi 算法来进行词性估计。

(2) 特点

(2.1) 支持多种分词模式

`jieba` 支持以下三种分词模式：精确模式，对语句进行最精确地切分，只用于文本数据分析；全模式，把句子中所有的可以成词的词语都扫描出来，该模式速度非常快，但是不能解决歧义；搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。

(2.2) 支持繁体分词

(2.3) 支持自定义词典

(2.4) 待分词的字符串可以是 unicode 或 UTF-8 编码

4.1.1.2 `jieba` 中文分词函数

安装了 `jieba` 后，直接导入 `jieba` 模块，即可调用该模块下的函数，主要用于分词的函数及调用方式见下表：

@todo 补充函数说明表 4.1.1.2

(1) `jieba.cut`

`jieba.cut` 是 `jieba` 模块下进行中文语句分词的主要函数，调用方式为：`jieba.cut(sentence, cut_all=False, HMM=True)`

参数说明：

`sentence`:需要分词处理的字符串

`cut_all`:分词模式，`True` 代表全模式，`False` 代表精确模式，默认缺失值为 `False`

`HMM`:是否使用 HMM 模型，默认缺失值为 `True`

`jieba.cut` 返回的结果是一个可迭代的 `generator`，可以使用 `for` 循环来获得分词后得到的每一个词语，也可以用 `list()` 函数进行转化

例 1 利用 `jieba.cut` 对语句“我爱文本数据分析”进行分词处理

```
In [1]:import jieba
seg_list = jieba.cut("我爱文本数据分析") # 默认是精确模式
seg_list
Out[1]:<generator object cut at 0x7f30d0270140>
In [2]:for i in jieba.cut("我爱文本数据分析"):
    print i
我
爱
文本
数据分析
In [3]:for i in jieba.cut("我爱文本数据分析",cut_all=True): # 全模式
    print i
我
爱
文本
本数
数据
数据分析
分析
In [4]:list(jieba.cut("我爱文本数据分析"))
Out[4]:[u'\u6211', u'\u7231', u'\u6587\u672c', u'\u6570\u636e\u5206\u6790'] # 列表元素为
中文词汇对应的 Unicode 编码
```

可以看到，`jieba.cut` 直接返回的是 `generator`，为了查看分词结果，使用 `for` 循环语句打印得到的每一个词语，如果直接对 `generator` 列表处理，返回的列表元素为分词结果对应的 `Unicode` 编码。

(2) `jieba.cut_for_search`

`jieba.cut_for_search` 是适用于搜索引擎构建倒排索引（`Inverted index`）的分词函数，调用方式为：`jieba.cut_for_search(sentence, HMM=True)`

参数说明：

`sentence`:需要分词处理的字符串

`HMM`:是否使用 HMM 模型，默认缺失值为 `True`

和 `ieba.cut` 一样，`jieba.cut_for_search` 返回的结果也是一个可迭代的 `generator`。

例 2 利用 `jieba.cut_for_search` 对语句“我爱文本数据分析”进行分词处理

```
In [5]:seg_list = jieba.cut_for_search("我爱文本数据分析")
    seg_list
Out[5]:<generator object cut_for_search at 0x7f30d02703c0>
In [6]:for i in jieba.cut_for_search("我爱文本数据分析"):
    print i
我
爱
文本
数据
分析
数据分析
```

与 `jieba.cut` 函数的分词结果进行对比，可以发现多出了“数据”和“分析”两个分词结果。

(3) `jieba.lcut`

`jieba.cut` 返回的结果是 generator，`jieba.lcut` 可以直接返回列表结果，直接使用 `jieba.lcut` 进行分词可以省去对 `jieba.cut` 分词结果的列表化处理。

例 3 利用 `jieba.lcut` 对语句“我爱文本数据分析”进行分词处理

```
In [7]:jieba.lcut("我爱文本数据分析")
Out[7]:[u'\u6211', u'\u7231', u'\u6587\u672c', u'\u6570\u636e\u5206\u6790']
In [8]:print "/".join(jieba.lcut("我爱文本数据分析"))
我/爱/文本/数据分析
```

(4) `jieba.lcut_for_search`

`jieba.lcut_for_search` 函数的作用与 `jieba.lcut` 相似：`jieba.cut_for_search` 返回的结果是 generator，`jieba.lcut_for_search` 可以直接返回列表结果，直接使用 `jieba.lcut_for_search` 进行分词可以省去对 `jieba.cut_for_search` 分词结果的列表化处理。

例 4 利用 `jieba.lcut_for_search` 对语句“我爱文本数据分析”进行分词处理

```
In [9]:jieba.lcut_for_search("我爱文本数据分析")
Out[9]:[u'\u6211',
        u'\u7231',
        u'\u6587\u672c',
        u'\u6570\u636e',
        u'\u5206\u6790',
        u'\u6570\u636e\u5206\u6790']
In [10]:print "/".join(jieba.lcut_for_search("我爱文本数据分析"))
我/爱/文本/数据/分析/数据分析
```

(5) jieba.tokenize

jieba.tokenize 函数可以对指定语句进行切分并以元组的形式返回词语在原文的起止位置：

(词语，开始位置，结束位置)，返回结果是 generator，调用方式为：

```
jieba.tokenize(sentence, mode=u'default', HMM=True)
```

参数说明：

sentence:需要分词处理的字符串，必须是 Unicode 形式,在字符串引号前加一个 u 即可

mode：取值为 "default" 或者 "search", "search" 表示精细切分

HMM:是否使用 HMM 模型，默认缺失值为 True

例 5 利用 jieba.tokenize 对语句“我爱文本数据分析”进行分词处理

```
In [11]:jieba.tokenize(u"我爱文本数据分析")
Out[11]:<generator object tokenize at 0x7f30d01f85a0>
In [12]:for i in jieba.tokenize(u"我爱文本数据分析"):
    print i
(u'\u6211', 0, 1)
(u'\u7231', 1, 2)
(u'\u6587\u672c', 2, 4)
(u'\u6570\u636e\u5206\u6790', 4, 8)
```

从输出结果可以看到，每一个分词结果都保存在一个元组中，元组中另外两个元素分别为该词汇在语句中的起始位置。

(6) jieba.load_userdict

jieba 分词的一个特点就是支持自定义词典，**jieba.load_userdict** 函数即可实现这一功能。虽然 **jieba** 有新词识别能力，但是使用者通过自定义的词典，可以包含 **jieba** 自带词库里没有的词汇，从而保证更高的正确率。调用方式为：**jieba.load_userdict(file)**

参数说明：

file：词典文件对象或自定义词典的路径，文件必须为 UTF-8 编码

注意：自定义词典格式需要和 **jieba** 自带词库 **dict.txt** 一样，每一行从左至右分别为词语、词频、词性三部分，各部分之间用空格分开，其中词性部分可以省略，词频越大成词的概率越大。

(7) jieba.add_word

jieba.add_word 函数用于给词典中增加新的词汇，调用方式为：**jieba.add_word(word, freq=None, tag=None)**

参数说明：

word：需要增加到词典的词汇

freq：词频，可省略

tag：词性，可省略

例 6

```
In [13]:print "/".join(jieba.cut("江州市长江大桥参加了长江大桥的通车仪式"))
江州/市/长江大桥/参加/了/长江大桥/的/通车/仪式
In [14]:jieba.add_word("江大桥", freq =50000)
In [15]:print "/".join(jieba.cut("江州市长江大桥参加了长江大桥的通车仪式"))
江州/市长/江大桥/参加/了/长江大桥/的/通车/仪式
```

(8) jieba.del_word

`jieba.del_word` 函数用于删除词典中的词汇，调用方式为：`jieba.del_word(word)`

(9) jieba.suggest_freq

`jieba.suggest_freq` 函数用于调节词语的词频，用于将一个词汇中的字符分开或者合并以增加该切分出该词汇的可能性。调用方式为：`suggest_freq(segment, tune=True)`

参数说明：

`segment`：某一词汇预期被切分的形式

`tune`：取值为 `True` 时，调整词频

注意：HMM 新词发现功能可能会影响到 `suggest_freq` 函数的结果，如果调用该函数后分词结果没有改变，调用分词函数时可以设置 `HMM=False`。

例 7

```
In [16]:print "/".join(jieba.cut("结巴分词是很好的中文分词工具"))
结巴/分词/是/很/好/的/中文/分词/工具
In [17]:jieba.suggest_freq(("结巴分词"), True)
Out[17]:1
In [18]:print "/".join(jieba.cut("结巴分词是很好的中文分词工具"))
结巴分词/是/很/好/的/中文/分词/工具
```

例 8

```
In [16]:print "/".join(jieba.cut("在不断的练习中将有所进步"))
在/不断/的/练习/中/将/有/所/进/步
In [17]:jieba.suggest_freq(("中","将"), True)
Out[17]:494
In [18]:print "/".join(jieba.cut("在不断的练习中将有所进步"))
在/不断/的/练习/中/将/有/所/进/步
```

4.1.1.3 jieba 中文词性标注

`posseg` 包是 `jieba` 中实现词性标注功能的包，具体词性类别见下表：

@todo 补充词性对照表

`posseg` 包中定义了以下词性标注函数：

(1) cut

`cut` 函数可以同时实现分词和词性标注，以 `pair(u'词', u'词性')` 的形式返回标注结果，即将分词结果和对应的词性保存为一个 `pair` 对象，调用方式为：`posseg.cut(sentence, HMM=True)`

参数说明：

`sentence`：需要词性标注的文本

`HMM`：是否使用 `HMM` 发现新词，默认缺失值为 `True`

例 9

```
In [19]:import jieba
from jieba import posseg
pos = list(jieba.posseg.cut("我爱文本数据分析"))
pos
Out[19]:[pair(u'\u6211', u'r'),
pair(u'\u7231', u'v'),
pair(u'\u6587\u672c', u'n'),
pair(u'\u6570\u6362\u5206\u6790', u'l')]
In [20]: for i in pos:
    print i
我/r
爱/v
文本/n
数据分析/l
```

(2) lcut

lcut 函数与 cut 函数的作用一样，但是可以直接返回结果列表，调用方式为：

```
posseg.lcut(sentence, HMM=True)
```

例 10

```
In [21]:jieba.posseg.lcut("我爱文本数据分析")
Out[21]:[pair(u'\u6211', u'r'),
          pair(u'\u7231', u'v'),
          pair(u'\u6587\u672c', u'n'),
          pair(u'\u6570\u636e\u5206\u6790', u'l')]
In [22]:for a,b in jieba.posseg.lcut("我爱文本数据分析"):
          if b=="n":
              print a
文本
```

4.1.1.4 jieba 中文分词与词性标注案例

案例一

从数据堂（<http://more.datatang.com/>）网站下载新浪社会新闻 2013 年 4 月-7 月数据 new.txt（数据网址：<http://more.datatang.com/data/44293>），其中包括新浪社会新闻的标题、url、发布时间，共 2 万条数据。

@todo 插入数据文件截图 4.1.1.4

将数据上传到 Jupyter Notebook 文件列表，新建一个 Python notebook，用于运行相关命令并保存分析结果。首先读取文本数据：

```
In [1]:with open ("new.txt") as f:
    read=f.readlines()
    read[0:3]
Out[1]:['http://news.sina.com.cn/c/2013-07-12/134927651359.shtml|\xe7\xbd\xae\x4\xba|(07\xe6\x9c\x8812\xe6\x97\xaa 13:49)\n',
        'http://news.sina.com.cn/c/2013-07-12/134227651346.shtml|\xe6\xb9\xbb\x0\x91|(07\xe6\x9c\x8812\xe6\x97\xaa 13:42)\n',
        'http://news.sina.com.cn/c/2013-07-12/131527651106.shtml|\xe8\xb4\xbc\xaa|(07\xe6\x9c\x8812\xe6\x97\xaa 13:15)\n']
```

从返回的结果 Out[1] 可以看出，每一条新闻数据都以一个字符串的形式保存，字符串中包括新闻网址、标题、发布时间三个元素，并以“|”隔开。下面利用 split 函数去除字符串中的“|”，同时将字符串中不同成分分离开来，并提取标题文本存放在列表 title 中：

```
In [2]:title=[]
    for i in read:
        title.append(i.split("|")[1].decode("utf-8")) # 去除"|"后标题索引位置为[1]，并
利用 decode 函数进行解码处理
In [3]:title[0:3]
Out[3]:[u'\u7f51\u6c11\.....\u516c\u793a',
         u'\u6e56\u5357\.....\u79fb\u6c11',
         u'\u8d35\u5dde\.....\u6b7b3\u4f24']
In [4]:for i in title[0:3]:
    print i
网民质疑广东江门核燃料项目:4月动工7月公示
湖南家谱解密:六成人口是江西移民
贵州六盘水吊车侧翻致2死3伤
```

直接对新闻标题进行分词处理，并将分词后结果保存在列表 new_seg 中：

```
In [3]:import jieba
new_seg=[]
for i in title:
    seg=jieba.lcut(i)
    new_seg.append(seg)
```

查看部分分词结果：

```
In [4]:for i in new_seg[0:5]:
    print "/".join(i)
网民/质疑/广东/江门/核燃料/项目/:/4/月/动工/7/月/公示
湖南/家谱/解密/:/六成/人口/是/江西/移民
贵州/六盘水/吊车/侧翻/致/2/死/3/伤
北京公交/9/月/将/推/可/定制/商务/班车/ /保证/一人/一座
美国/总统/奥巴马/会见/汪洋/和/杨洁篪
```

进一步采用搜索引擎模式对标题进行分词，将分词结果保存在列表 new_seg_for_search 中：

```
In [5]:new_seg_for_search=[]
for i in title:
    seg=jieba.lcut_for_search(i)
    new_seg_for_search.append(seg)
```

查看部分分词结果：

```
In [6]:for i in new_seg_for_search[0:5]:
    print "/" .join(i)
网民/质疑/广东/江门/燃料/核燃料/项目/:/4/月/动工/7/月/公示
湖南/家谱/解密/:/六成/人口/是/江西/移民
贵州/六盘/六盘水/吊车/侧翻/致/2/死/3/伤
北京/公交/北京公交/9/月/将/推/可/定制/商务/班车/ /保证/一人/一座
美国/总统/奥巴/巴马/奥巴马/会见/汪洋/和/杨洁篪
```

对分词结果进行词性标注，标注的结果保存在列表 new_posseg 中：

```
In [7]:from jieba import posseg
new_posseg=[]
for i in title:
    seg=jieba.posseg.lcut(i)
    new_posseg.append(seg)
```

直接筛选所有出现的形容词，保存到列表 mew_adj 中：

```
In [8]:mew_adj=[]
for i in title:
    for a,b in jieba.posseg.lcut(i):
        if b=="a":
            mew_adj.append(a)
```

统计各个形容词出现的次数，查看出现次数最多的前五个词：

```
In [9]:from collections import Counter
c=Counter(mew_adj)
for i in c.most_common(5):
    print "word:",i[0],"count:",i[1]
word: 大 count: 196
word: 新 count: 163
word: 最高 count: 143
word: 严重 count: 129
word: 高 count: 110
```

案例二

随着网络信息以及仓储物流等行业的迅速发展，网购已经成为现代人的重要购物方式，伴随而来的是网民在各种网络平台上所发表的购物观点、意见等评论文本数据的激增，这些评论文本包含了消费群体对所购买的商品或者服务的情感态度等信息，反映了用户通过互联网对产品各方面发表的看法，对电子商务平台销售者以及个体消费者都有重要的分析价值。通过

评论分析，商家能够了解消费者对产品的看法，发现与竞争对手的差异，为产品改进、价格优化提供有价值信息；另外与商家的促销信息相比，在线评论具有独立性、非商业性，因此深得用户信赖，通过查看商品评价，消费者可以迅速了解到其他顾客对商品的评价、锁定关注属性的相关评价，支持消费决策，有效提高决策效率。鉴于网络评论文本数据分析的重要价值，本书多个分析实例都是围绕电子商务平台评论文本展开的，具体的评论文本抓取过程在案例中未做说明。

抓取亚马逊中国站上热门商品“kindle”电子书阅读器下“kindle”、“kindle paperwhite”、“kindle voyage”三个子类产品的商品评论语料作为分析实例，抓取的期间为 2014 年 10 月 3 日至 2015 年 8 月 24 日，共抓取评论文本 5632 条，保存为 CSV 格式文件 kindle_corpus.csv，并上传到 Jupyter Notebook 文件列表，新建一个 Python notebook，读取文本数据，并将数据保存到列表 corpus 中：

```
In [1]:import csv
      import jieba
In [2]:corpus=[]
      with open("kindle_corpus.csv") as f:
          reader=csv.reader(f)
          for i in reader:
              corpus.append(i)
```

查看 corpus 内元素形式：

```
In [3]:corpus[0:3]
Out[3]:[['', 'project_id', 'source_id', 'item_id', 'content', 'meta', 'pubdate'],
         ['0',
          '1',
          '1',
          'RI04UTN75BRKS',
          '\xe5\x90\x8c\xe6\x9f\xef\xbc\x9f',
          '{"rating": "3"}',
          '2015-07-11'],
         ['1',
          '1',
          '1',
          'R4B71L8JMVEZ3',
          '\xe5\x88\x9a\xe5\x86\xe3\x80\x82',
          '{"rating": "5"}',
          '2015-07-08']]
```

可以看到 corpus 第一个列表元素即为 CSV 文件的第一行，即抓取的字段，包括：空格、project_id、source_id、item_id、content、meta、pubdate，分别代表序号（从 0 开始）、商品所属大类 id、商品所属子类 id（共三个子产品，取值为 1、3、4，分别对应“kindle

“paperwhite”、“kindle voyage”和“kindle”）、商品 id、商品评论文本、商品评分（5分、4分、3分、2分、1分）和评论日期。将商品所属子类 id 为 3 的商品评论文本保存到列表 review 中：

```
In [4]:review=[]
    with open("kindle_corpus.csv") as f:
        reader=csv.reader(f)
        reader.next() # 跳过第一行字段内容
        for i in reader:
            if i[2]=="3":
                review.append(i[4])
In [5]:len(review) # 查看评论数
Out[5]:1682
In [6]:for i in review[101:104]:
    print i + "\n"
    电子书阅读器着实不错啊，很顺手。就是套餐里面的电子书购书券不知道为什么迟迟不到账？！！
```

1. 价格太贵太离谱 2. 软件缺乏人性化 3. 翻页键不灵敏，基本上就是个摆设，毫无用处 4. 每天阅读半小时也就2周的待机时间而已，待机数周纯粹骗人 建议入手499或者899

不知道是不是我自己对这个小物寄予太大期望了，拿到手之后就用了。手感很棒，背面的logo太丑了。不伤眼，但是一翻页就闪屏，刷新闪屏，各种闪屏，体验不好，略失望

添加评论中频繁出现的新词到 jieba 自带词典：

```
In [7]:for i in ['学生党', '纸质书', '电源键', '电子屏', '阴阳屏', '墨水屏', '电纸书', '电子墨水']:
    jieba.add_word(i, freq =50000)
In [8]:print "/".join(jieba.cut("学生党买电纸书遇到阴阳屏"))
    学生党/买/电纸书/遇到/阴阳屏
```

jieba 中并未直接提供停用词处理函数，需要使用者自行定义停用词词典对分词结果进行过滤操作。

加载停用词典：

```
In [9]:with open("stopwords.txt") as f: # 打开停用词典
    read=f.read().decode('utf-8') # 以 utf-8 编码格式解码字符串
    stop_words = read.splitlines() # 返回结果 read 为一个包含换行符的字符串，利用 splitlines 方法去掉输出结果里的换行符，或者用 split("\n")
In [10]:stop_words
Out[10]:[u',',
          u'?',
          u'\u3001',
          u'\u3002',
          ....]
```

分词并过滤停用词：

```
In [11]:review_segs=[] # 创建空列表 review_segs 用于存放所有评论的分词结果
for i in review:
    review_seg=[] # 创建空列表 review_seg 用于存放每条评论的分词结果
    segs = jieba.cut(i)
    for seg in segs:
        if seg not in stop_words:
            review_seg.append(seg) # 对于每条评论分词后的词汇，如果不在停用词表中就添加到该条评论的分词列表中，也就是说，如果是停用词就过滤掉
    review_segs.append(review_seg) # 将每条评论的分词结果添加到列表 review_segs 中
```

随机选取评论文本，查看分词结果：

```
In [12]:print "/".join( review_segs[850])
屏幕/不错/阴阳屏/限量版/好像/选/配套/保护套/颜色
```

直接对分词结果进行词性标注，同时过滤掉停用词，将词性标注结果保存在列表 review_pos 中：

```
In [13]:from jieba import posseg
review_pos=[]
for i in review:
    for a,b in posseg.lcut(i):
        if a not in stop_words:
            review_pos.append((a,b))
```

查看部分词性标注结果：

```
In [14]:for i in review_pos[0:5]:
    print i[0],i[1]
pw eng
对比 v
来说 u
优势 n
亮度 n
```

分别将标注结果中的名词和形容词保存到列表 review_n 和 review_a，进行统计汇总后，查看出现次数较多的词汇：

```
In [15]:review_n=[]
    review_a=[]
    for i in review_pos:
        if i[1]==“n”:
            review_n.append(i[0])
        elif i[1]==“a”:
            review_a.append(i[0])
In [16]:from collections import Counter
    c1=Counter(review_n)
    for i in c1.most_common(5):
        print i[0],i[1]
屏幕 531
感觉 484
电子书 362
问题 333
书 333
```

出现次数最多的名词是“屏幕”，可以推测“屏幕”在所有评论中被谈及的次数最多，是所有买家最关注的产品属性，“电子书”等也是大家关注的焦点。

```
In [17]:c2=Counter(review_a)
    for i in c2.most_common(5):
        print i[0],i[1]
好 669
不错 396
方便 225
明显 151
舒服 139
```

从出现次数较多的形容词可以在一定程度上初步了解买家对产品的评价，如“方便”、“舒服”等。

4.1.2 Yaha 中文分词

4.1.2.1 了解 Yaha 中文分词

(1) 分词原理

Yaha（“哑哈”）中文分词的词典是直接从 `jieba` 项目拷贝直接过来的，其算法的核心与我们之前讲到的 `jieba` 分词类似，同样基于对句子的最大概率路径的计算。与 `jieba` 不同的是，Yaha 分词在分词算法过程中去掉了很消耗内存的 Trie 树结构以及被项目研究者认为表现较差的 HMM 模型，利用“**最大熵**”算法实现大段文字的新词发现能力，同时使用者可以对分词的各个阶段进行定义。`Yaha` 分词的具体分词流程如下：

- a. 根据自定义的正则表达式规则切分目标语句，提取切分得到的独立词汇（如数字、英文单词等），这些词汇将被固定，不在之后的环节中被切分；
- b. 对目标语句进行预扫描，在词典中加入一些可能形成的新词，并确定其概率值；
- c. 结合内置词典与相关匹配模式创建有向无环图，对图中的词汇顶点赋予对应的概率值；
- d. 根据最优化算法（动态规划或Dijkstra算法）实现最大概率路径计算，二次处理其中不能成词的单字，或是输出多条分词路径，结合用户兴趣获得所需的分词结果。

(2) 特点

(2.1) 支持多种分词模式

和 `jieba` 分词一样支持以下三种分词模式：精确模式，对语句进行最精确地切分，只用于文本数据分析；全模式，把句子中所有的可以成词的词语都扫描出来，该模式速度非常快，但是不能解决歧义；搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。此外还支持备选路径，可生成最好的多条切词路径，在此基础上根据其它信息可得到更精确的分词模式。

(2.2) 分词规则的调控插件

用户可通过添加插件的方式调整分词过程中的正则表达式与匹配模式，默认的 `Yaha` 分词插件有正则表达式插件、人名前缀插件以及地名后缀插件，同时用户也可以通过定制插件的方式在分词的分宜阶段加入个人的定制。

(2.3) 新词学习功能

`Yaha` 分词自带的新词学习功能可以根据输入的大段文字学习自动，通过词语组成的规律得到文本当中的专业名词、名字、地点名词等等词语，可用于创建自定义词典，或在SNS等场合进行数据挖掘的工作。

(2.4) 获取大段文本的关键字和摘要

`Yaha` 中的有关函数能够基于词汇在文本段落中的权重提取关键词，并根据关键词的分布情况对断句进行打分，由截取的断句拼接为文本的摘要。

(2.5) 词语纠错功能

`Yaha` 库中的新功能，可以用于在搜索里对用户的错误输入进行纠正。

4.1.2.2 Yaha 中文分词方法

在调用 `Yaha` 分词函数之前，要先安装 `Yaha` 模块，在 notebook 代码单元格输入“`!pip install yaha`”，运行后在单元格下方可以看到如所示的安装进度提示：

```

Collecting yaha
  Downloading yaha-0.02.tar.gz (1.1MB)
  .....
Successfully built yaha
Installing collected packages: yaha
Successfully installed yaha-0.2

```

在 Yaha 模块中，Cuttor 类下定义了多个分词相关的方法，在调用类中的函数前需要将类“实例”化，即创建一个对象，类中的方法均可以作用于该对象。过程类似导入模块的操作，实例化类后即可调用类中的方法。

(1) cut

cut 方法是精确模式分词函数，返回一个 generator ，调用方式为：实例.cut(sentence)

例 1

```

In [1]:import yaha
        cutter=yaha.Cuttor() # 实例化 Cuttor 类
        cutter.cut("我爱文本数据分析") # 调用 cut 进行分词
Out[1]:<generator object cut at 0x7f51ac4de9b0>
In [2]:print "/".join(cutter.cut("我爱文本数据分析"))
        我/爱/文本/数据分析

```

(2) cut_all

cut_all 方法是全模式分词函数，返回一个 generator ，调用方式为：实例.cut_all(sentence)

例 2

```

In [3]:print "/".join(cutter.cut_all("我爱文本数据分析"))
        我/爱/文本/数据分析/数据/分析

```

(3) tokenize

与 jieba 分词一样，Yaha 中 tokenize 方法在分词的同时返回词汇起始位置，调用方式为：实例.tokenize(unicode_sentence, search=False)

例 3

```
In [4]:for i in cutter.tokenize(u"我爱文本数据分析"):
    print "word:"+i[0]+" "+"srart:"+str(i[1])+" "+"end:"+str(i[2])
word:我 srart:0 end:1
word:爱 srart:1 end:2
word:文本 srart:2 end:4
word:数据分析 srart:4 end:8
```

除 Cutter 之外，Yaha 模块中还定义了 RegexCutting、SuffixCutting、SurnameCutting 等类，感兴趣的读者可以进一步了解。

4.1.3 Genius

4.1.3.1 了解 Genius 中文分词

Genius 是一个开源的 Python 中文分词组件，基于 wapiti 采用 CRF(Conditional Random Field) 条件随机场算法实现分词功能，其支持用户自定义合并词典，采用 trie 树进行合并词典查找，并且支持词性标注功能。

4.1.3.2 Genius 中文分词函数

在调用 Genius 分词函数之前，要先安装 Genius 模块，在 notebook 代码单元格输入“!pip install Genius”，运行后在单元格下方可以看到如所示的安装进度提示：

```
Collecting genius
  Downloading genius-3.1.6.tar.gz (18.7MB)
  .....
Successfully built genius libwapiti
Installing collected packages: nose, libwapiti, genius
Successfully installed genius-3.1.6 libwapiti-0.2.1 nose-1.3.7
```

(1) seg_text

seg_text 是 genius 模块中定义的分词函数，调用方式为：

```
genius.seg_text(text,use_break,use_combine,use_tagging,use_parse_pinyin)
```

参数说明：

text: 需要分词的语句，必填参数且必须是 unicode

use_break: 代表对分词结构进行打断处理，默认值True

use_combine: 代表是否使用字典进行词合并，默认值False

`use_tagging`: 代表是否进行词性标注，默认值False

`use_parse_pinyin`: 代表是否对拼音进行分词处理，默认值False

例 1

```
In [1]:import genius
seg_list = genius.seg_text(u"我爱文本数据分析",use_tagging=True)
for i in seg_list:
    print i.text, i.tagging
我 r
爱 v
文本 n
数据分析 v
```

输出结果的第二列即为相应词汇的词性。

(2) `seg_keywords`

`seg_keywords` 是 `genius` 模块中保留歧义的适用于搜索引擎的分词函数，调用方式为：
`genius.seg_keywords(text, use_break, use_tagging, use_parse_pinyin)`

参数说明：

`text`: 需要分词的语句，必填参数且必须是 `unicode`

`use_break`: 代表对分词结构进行打断处理，默认值True

`use_tagging`: 代表是否进行词性标注，默认值False

`use_parse_pinyin`: 代表是否对拼音进行分词处理，默认值False

例2

```
In [2]:seg_list = genius.seg_keywords(u"我爱文本数据分析")
print('/'.join([word.text for word in seg_list]))
我/爱/文/文本/本/数据/分析
```

4.1.4 `finalseg`

`finalseg` 是基于 HMM 模型的中文分词模块，采用的算法是 Viterbi，有新词识别功能，并可以选择性打开该功能。在线分词效果展示地址：<https://finalseg.appspot.com/>。执行语句
`!easy_install finalseg` 即可完成 `finalseg` 安装，`cut` 函数为 `finalseg` 中的分词函数，调用方式为：`finalseg.cut(sentence, find_new_word=False)`

参数说明：

sentence：待分词语句

find_new_word：是否开启新词识别功能，缺失值默认为 False

例

```
In [2]:import finalseg
In [3]:print "/".join(finalseg.cut("我爱文本数据分析"))
我/爱/文本/数据/分析
```

4.1.5 scseg

scseg 中文分词，是基于 mmseg 的简单分词模块，支持汉字数字和拼音分词，使用者可以自定义词典。执行语句 “!pip install scseg” 即可完成 scseg 安装。该模块主要包括以下两个分词函数：

(1) seg_text

seg_text 为 scseg 模块的分词函数，调用方式为：scseg.seg_text(text, ext_dict_words=set([]), use_combine=True)

参数说明：

text：需要分词的语句

ext_dict_words：用户自定义的扩展字典

use_combine：是否需要合并处理，默认缺失值为 True

例 1

```
In[1]:import scseg
seg_list = scseg.seg_text(u'我爱文本数据分析，wenben shuju')
print '/'.join(seg_list)
我爱/文本/数据/分析/wenben/shuju
```

(2) seg_keywords

seg_keywords 为 scseg 模块中保留歧义的适用于搜索引擎的分词函数，可以枚举出所有可能的切分方式，调用方式为：scseg.seg_keywords(text)

例 2

```
In [2]:seg_list = scseg.seg_keywords(u'我爱文本数据分析，wenben shuju')
print '/'.join(seg_list)
我爱/分析/爱文/shuju/文本/数据/wenben
```

4.1.6 pynlpir

pynlpir 中文分词是利用 NLPIR 进行中文分词的 Python 包，支持新词发现与自适应分词功能，从较长的文本内容中，基于信息交叉熵自动发现新特征语言，并自适应测试语料的语言概率分布模型，实现自适应分词。

其中 `segment` 是利用 NLPIR 进行中文分词的函数，分词结果以列表形式返回，若只设置分词参数则列表元素为分词后词汇的字符串，如：“我”，“是”，……；若分词的同时进行词性标注，则列表元素为有分词后词汇字符串和相应的词性构成的元组，如：

[(“我”, “pronoun”） , (“是”, “verb”） ,]。调用方式为：`pynlpir.segment(s, pos_tagging=True, pos_names=u'parent', pos_english=True)`

参数说明：

`s`: 需要分词的中文语句，必须为 `Unicode` 或者 `UTF-8` 编码的字符串

`pos_tagging`: 是否使用词性标注，默认缺失值为 `True`

`pos_names`: 返回何种类型的词性标注，即词性划分的层次，只有当参数 “`pos_tagging`” 取值为 “`True`” 时才需要进行设置。有以下可选值，取值为字符串类型：

- (1) `data`: 返回 NLPIR 原始的词性编号
- (2) `parent`：默认缺失值，表示使用常用词性表示方式，如将“国家科学基金会”这类词归为“`noun`”
- (3) `child`：表示使用专用的、准确的词性表示方式，如将“国家科学基金会”这类词归为“`transcribed toponym`”
- (4) `all`：表示综合使用多种表示方式分层标注词性，如将“国家科学基金会”这类词归为“`noun:toponym:transcribed toponym`”

`pos_english`: 使用英文还是中文显示词性标注结果，默认取值为“`True`”，即英文表示，也仅在参数 “`pos_tagging`” 取值为 “`True`” 时才需要进行设置。

4.1.7 SnowNLP

SnowNLP 是基于 Python 的中文文本处理库，自带已经训练好的字典，支持中文分词、词性标注、情感分析等文本数据分析功能。执行语句 “`!pip install snownlp`” 即可完成安装，该库定义的 `SnowNLP` 类包含了分词、词性标注、情感分析等方法，分词方法 `words` 的调用方式为：

```
from snownlp import SnowNLP  
实例 = SnowNLP(text) # 实例化, text 为需要分词处理的字符串  
实例.words # 调用分词方法, 得到分词结果列表
```

4.1.8 smallseg

smallseg 是一个开源的、基于 DFA 的轻量级中文分词模块，使用 RMM 字符串分割算法。其特点在于：可自定义词典，分词后可以返回登录词列表和未登录词列表，并且有一定的新词识别能力。smallseg 模块的安装也十分简单，将下载包中 `smallseg.py` 拷贝到 Python 安装目录的 `Lib` 文件夹中即可完成安装。

smallseg 模块中常用的分词方法是类 `SEG` 下定义的 `cut` 方法，该方法可以实现中文语句的切分，返回识别出的登录词列表和未登录词列表，实例化类 `SEG` 后，通过“`实例.cut(sentence)`”即可实现对该分词方法的调用。

4.1.9 snailseg

snailseg 是一个基于 Python 的简单的中文分词库，通过统计单字在词语中出现的概率大小进行分词，选择最大可能的分词方案。snailseg 模块的安装也十分简单，将 `snailseg` 目录放置于当前目录或者 `site-packages` 目录即可完成安装。模块中的 `cut` 函数可以实现中文语句的切分，调用方式为：`snailseg.cut(sentence)`。

4.2 英文分词及词性标注

与中文分词相比，英文分词要简单很多。在英文中，单词之间是以空格作为自然分界符组成语句，语句之间再利用标点分隔组成大篇幅文本，所以我们可以简单的利用标点进行分句处理，利用空格进行分词处理。设计分句函数的思路很简单，英文文本中出现的标点一般为逗号“,”、句点“.”和问号“?”，假设我们有一段英文文本，可以先将文本按照句点分割成若干小段的文本，再将各小段文本按照内部出现的逗号或者问号再次切分。基于这样的思路，我们即使不利用外部的工具包，也能够通过调用 Python 中的内置函数来构建一个简单的英文分词器。

例 1 按标点分句

```
In [1]:def sentence_split(sentence): # 编写分句函数
    text_sen = []
    for s in sentence.split('.'):
        if '?' in s:
            tsxt_sen.extend(s.split('?'))
        elif ',' in s:
            tsxt_sen.extend(s.split(','))
        else:
            text_sen.append(s)
    return tsxt_sen
In [2]:text="text minming with python,TEXT MINING WITH PYTHON.Text Mining?With Python."
sentence_split(text)
Out[2]:['text minming with python',
       'TEXT MINING WITH PYTHON',
       'Text Mining',
       'With Python',
       '']
```

例 2 按空格分词

```
In [3]:text_word=[]
for i in sentence_split(text):
    if i != "":
        text_word.append(i.split(" "))
In [4]:text_word
Out[4]:[['text', 'minming', 'with', 'python'],
         ['TEXT', 'MINING', 'WITH', 'PYTHON'],
         ['Text', 'Mining'],
         ['With', 'Python']]
```

对于上例中结构比较简单的文本，可以自行编写函数进行处理，但是对于结构、内容更加复杂的英文文本，类似的操作就很难进行全面、细致的处理了，如对下例所示语句进行分句处理：

例 3

```
In [5]:text= "Good muffins cost $3.88\nin New York. Please buy me two of them.\nThanks."
         sentence_split(text)
Out[5]:['Good muffins cost $3',
        '$88\nin New York',
        'Please buy me two of them',
        '\nThanks',
        '']
```

由于句中存在“\$3.88”、“\n”等特殊表述，直接用自定义函数进行分句，并不能达到理想效果，本节将介绍几个英文分词工具，利用这些工具可以自动、快速实现分词处理，无需自行编写复杂的分句、分词函数，进而将更多的注意力集中在更加复杂的文本数据分析过程。

4.2.1 NLTK

作为基于 Python 的自然语言处理前沿平台，NLTK 为我们提供了一套更为专业的英文分词工具，相比于调用 Python 的内置函数，NLTK 的英文分词工具模式更加丰富，并且在去除停用词、词干化处理方面更为优秀。

4.2.1.1 tokenize 分词包

tokenize 是 NLTK 的分词包，其中的函数可以识别英文词汇和标点符号对文本进行分句或分词处理。

(1) sent_tokenize

sent_tokenize 为 tokenize 分词包中的分句函数，返回文本的分句结果，调用方式为：
sent_tokenize(text, language='english')

参数说明：

text: 需要分句处理的文本

language:nltk.tokenize.punkt 中包含了很多预先训练好的分词模型，参数 language 即为模型的名称

例 1

```
In [1]:import nltk
    from nltk.tokenize import sent_tokenize
    text= "Good muffins cost $3.88\nin New York. Please buy me two of them.\nThanks."
In [2]:sent_tokenize(text)
Out[2]:['Good muffins cost $3.88\nin New York.',
        'Please buy me two of them.',
        'Thanks.']

```

(2) word_tokenize

`word_tokenize` 为 `tokenize` 分词包中的分词函数，返回文本的分词结果，调用方式为：
`word_tokenize(text, language='english')`

参数说明：

`text`: 需要分词处理的文本

`language`: `nltk.tokenize.punkt` 中包含了很多预先训练好的分词模型，参数 `language` 即为模型的名称

例 2

```
In [3]:from nltk.tokenize import word_tokenize
    word_tokenize(text)
Out[3]:['Good',
        'muffins',
        'cost',
        '$',
        '3.88',
        'in',
        'New',
        'York',
        '.',
        'Please',
        'buy',
        'me',
        'two',
        'of',
        'them',
        '.',
        'Thanks',
        '.']
In [4]:[word_tokenize(t) for t in sent_tokenize(text)]
Out[4]:[['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.'],
        ['Please', 'buy', 'me', 'two', 'of', 'them', '.'],
        ['Thanks', '.']]
```

(3) regexp 模块：正则表达式分词

对于包含比较复杂词型（如 \$10、10%）的字符串，以上的分词算法往往不能实现精确分割，此时需要借助正则表达式来完成分词任务。所谓正则表达式（Regular Expression），就是一个描述指定的规则的字符序列，可以用来检查一个字符串是否与该规则匹配，用正则表达式分词就是按照正则表达式指定的规则对字符串进行分割的过程，可以根据实际情况自行编写正则表达式。正则表达式的内容本书不做特别的介绍，感兴趣的读者可以参看深度阅读部分推荐的读物学习。NLTK 提供了 `regexp` 模块支持正则表达式分词，其中包括 `regexp_tokenize`、`wordpunct_tokenize`、`blankline_tokenize` 等正则分词函数。

(3.1) RegexpTokenizer 类

`RegexpTokenizer` 是 `regexp` 模块下一个类，可以自行定义正则表达式进行分词，调用该类下的分词方法需要先实例化，实例化方式为：实例=`RegexpTokenizer(pattern, gaps, discard_empty, flags)`

参数说明：

`pattern`：必填参数，构建分词器的模式，即正则表达式字符串

`gaps`：可选参数，设置为 `True` 时，正则表达式指定识别标识符之间的间隔，默认缺失值为 `False`，即正则表达式用来识别标识符本身

`discard_empty`：可选参数，设置为 `True` 时，去除任何由分词器产生的空符“”，只有当参数“`gaps`”取值为“`True`”时分词器才会差生空符

`flags`：可选参数，编译分词器模式的正则标识，默认使用的是 `re.UNICODE | re.MULTILINE | re.DOTALL`

实例化后即可利用该类下的分词方法进行分词处理，分词方法调用方式为：实例.`tokenize` (`text`)

例 3

```
In [5]:from nltk.tokenize import RegexpTokenizer
       text="The four-poster canopy bed made in U.S.A. costs $600. The seller stake ou
t 40% of the profit."
       tokenizer = RegexpTokenizer('\w+|\$[\d\.\.]+|\S+')
       "/".join(tokenizer.tokenize(text))
Out[5]:'The/four/-poster/canopy/bed/made/in/U/.S.A./costs/$600./The/seller/stake/out/4
0/%/of/the/profit./'
```

也可以直接调用函数 `regexp_tokenize` 实现同样的分词效果，调用方式为：

`regexp_tokenize(text, pattern, gaps=False, discard_empty=True, flags=56)`

参数说明：

text：必填参数，需要分词的字符串

pattern：必填参数，构建分词器的模式，即正则表达式字符串

gaps：可选参数，设置为 `True` 时，正则表达式指定识别标识符之间的间隔，默认缺失值为 `False`，即正则表达式用来识别标识符本身

discard_empty：可选参数，设置为 `True` 时，去除任何由分词器产生的空符“”，只有当参数“`gaps`”取值为“`True`”时分词器才会差生空符

flags：可选参数，编译分词器模式的正则标识，默认使用的是 `re.UNICODE | re.MULTILINE | re.DOTALL`

例 4

```
In [6]:from nltk.tokenize import regexp_tokenize
In [7]:"/".join(word_tokenize(text))
Out[7]:'The/four-poster/canopy/bed/made/in/U.S.A./costs/$/600/.The/seller/stake/out/4
0/%/of/the/profit/.'
In [8]:pattern = r"""(?x)
                # 设置以编写较长的正则条件
               (?:[A-Z]\. )+
                # 缩略词
                | \$?\d+(?:\.\d+)?%
                # 货币、百分数
                | \w+(?:[-']\w+)*
                # 用连字符链接的词汇
                | \.\.\.
                # 省略符号
                | (?:[.,;'"?():-_`])
                # 特殊含义字符
                """
In [9]:"/".join(regexp_tokenize(text,pattern))
Out[9]:'The/four-poster/canopy/bed/made/in/U.S.A./costs/$600/.The/seller/stake/out/40
%/of/the/profit.'
```

从上例中两个分词函数返回的结果可以看出，利用正则表达式进行分词可以更加有针对性的解决待分词语句中有特殊格式的词汇。

(3.2) RegexpTokenizer 子类：用预先定义好的正则表达式分词

(a) WhitespaceTokenizer

`WhitespaceTokenizer` 子类，可以直接将字符串按照空格（包括 `space`, `tab`, `newline`）分词，效果相当于利用字符串 `split` 方法。调用方式为：实例.`tokenize` (`text`)

例 5

```
In [10]:from nltk.tokenize import WhitespaceTokenizer
        text="The four-poster canopy bed made in U.S.A. costs $600.\nThe seller stake
out 40% of the profit." # 第二句换行
In [11]:print text
        The four-poster canopy bed made in U.S.A. costs $600.
        The seller stake out 40% of the profit.
In [12]:tokenizer_space=WhitespaceTokenizer()
        "/".join(tokenizer_space.tokenize(text))
Out[12]:'The/four-poster/canopy/bed/made/in/U.S.A./costs/$600./The/seller/stake/out/40
%/of/the/profit.'
```

(b) WordPunctTokenizer

WordPunctTokenizer 子类，用正则表达式 “`\w+|\w\s+`” 将字符串切分成字母和非字母字符，分词方法调用方式为：实例.tokenize (text)

例 6

```
In [13]:from nltk.tokenize import WordPunctTokenizer
        tokenizer_punct=WordPunctTokenizer()
        "/".join(tokenizer_punct.tokenize(text))
Out[13]:'The/four/-/poster/canopy/bed/made/in/U./S./A./costs/$/600./The/seller/sta
ke/out/40/%/of/the/profit./'
```

也可以直接调用函数 wordpunct_tokenize 实现同样的分词效果，调用方式为：
wordpunct_tokenize (text)

例 7

```
In [14]:from nltk.tokenize import wordpunct_tokenize
        "/".join(wordpunct_tokenize(text))
Out[14]:'The/four/-/poster/canopy/bed/made/in/U./S./A./costs/$/600./The/seller/sta
ke/out/40/%/of/the/profit./'
```

(c) BlanklineTokenizer

BlanklineTokenizer 子类，将空行作为分隔符进行分词，空行是指不包含任何字符的行，空格 space 和制表符 tab 除外，相应的正则表达式为：“`\s\n\s\n\s*`”。分词方法调用方式为：实例.tokenize (text)

例 8

```
In [15]:from nltk.tokenize import BlanklineTokenizer
        text="The four-poster canopy bed made in U.S.A. costs $600.\n\nThe seller stake out 40% of the profit." # 第二句与第一句间空一行
In [16]:print text
        The four-poster canopy bed made in U.S.A. costs $600.

        The seller stake out 40% of the profit.
In [17]:tokenizer_blank = BlanklineTokenizer()
        tokenizer_blank.tokenize(text)
Out[17]:['The four-poster canopy bed made in U.S.A. costs $600.',
         'The seller stake out 40% of the profit.']

```

也可以直接调用函数 `blankline_tokenize` 实现同样的分词效果，调用方式为：

`blankline_tokenize (text)`

例 9

```
In [18]:from nltk.tokenize import blankline_tokenize
        blankline_tokenize(text)
Out[18]:['The four-poster canopy bed made in U.S.A. costs $600.',
         'The seller stake out 40% of the profit.']

```

(4) stanford 模块

`tokenize` 包中的 `stanford` 模块是 `NLTK` 的 `StanfordTokenizer` 接口，模块中定义的 `StanfordTokenizer` 类提供了利用分词工具 `PTBTokenizer` 进行分词的方法，调用方式为：实例.`tokenize(text)`

例 10

```
In [19]:from nltk.tokenize import StanfordTokenizer
        tokenizer_stan = StanfordTokenizer()
        tokenizer_stan.tokenize("Good muffins cost $3.88\nin New York. Please buy me\nntwo of them.\nThanks.")
Out[19]:['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.', 'Please', 'buy',
        'me', 'two', 'of', 'them', '.', 'Thanks', '.']
```

(5) `sexpr` 模块

`tokenize` 包中的 `sexpr` 模块是用来识别字符串中的带括号的表示形式，特别之处在于，该模块可以将字符串同时按照空格和括号切分。可以通过该模块下的 `SExprTokenizer` 类的方法实现括号表示的切分，首先将类实例化：实例 = `SExprTokenizer(paren='()', strict=True)`

参数说明：

parens：设置识别的括号形式，默认为（ 和 ）

strict：若括号表示不完全（如只包含半个括号）则会返回错误提示信息，参数 **strict** 取值为 **False** 时，则会进行正常切分处理，切分时将不完全的括号当做字符处理，默认缺失值为 **True**

实例方法调用方式为：实例.tokenize(text)

例 11

```
In [20]:from nltk.tokenize import SExprTokenizer
        tokenizer_sexpr=SExprTokenizer()
        tokenizer_sexpr.tokenize("ab(c,d)((e f) g) h")
Out[20]:['ab', '(c,d)', '((e f) g)', ' h']
In [21]:tokenizer_sexpr.tokenize("a b)(c d e f g h")
        ValueError: Un-matched close paren at char 13
In [22]:tokenizer_sexpr=SExprTokenizer(strict=False)
        tokenizer_sexpr.tokenize("a b)(c d e f g h")
Out[22]:['a', 'b', ')', '(c d e f g h']
In [23]:tokenizer_sexpr.tokenize("ab{c,d}{{e f} g} h")
Out[23]:['ab{c,d}{{e f} g} h']
In [24]:tokenizer_sexpr=SExprTokenizer(paren="{}")
        tokenizer_sexpr.tokenize("ab{c,d}{{e f} g} h")
Out[24]:['ab', '{c,d}', '{{e f} g}', ' h']
```

也可以直接调用函数 **sexpr_tokenize** 达到相同的切分效果，调用方式为：
sexpr_tokenize(text)

例 12

```
In [25]:from nltk.tokenize import sexpr_tokenize
        sexpr_tokenize('(a b (c d)) e f (g)')
Out[25]:['ab', '(c,d)', '((e f) g)', ' h']
```

(6) util 模块

util 模块提供了几个可以返回分词结果在原语句起始位置的函数，包括：
regexp_span_tokenize、**string_span_tokenize** 等。

(6.1) regexp_span_tokenize

regexp_span_tokenize 函数按照给定的正则表达式对字符串进行分词处理，并返回分词后各个词的起始位置，形式如 **(start, end)** 的元组。调用方式为：

regexp_span_tokenize(text,regexp)

参数说明：

text：需要分词的字符串

regexp：规定分割标识的正则表达式，不能为空

例 13

```
In [26]:from nltk.tokenize.util import regexp_span_tokenize
        text="Good muffins cost $3.88\nin New York. Please buy me\nntwo of them.\nThan
ks."
        list(regexp_span_tokenize(text, r'\s'))
Out[26]:[(0, 4),(5, 12),(13, 17),(18, 23),(24, 26),(27, 30),(31, 36),(38, 44),(45, 48)
,(49, 51),(52, 55),(56, 58),(59, 64),(65, 72)]
```

(6.2) string_span_tokenize

string_span_tokenize 函数按照给定的分隔符进行分词，并返回分词后各个词的起始位置，形式如 (start, end) 的元组。调用方式为：`regexp_span_tokenize(text,sep)`

参数说明：

text：需要分词的字符串

sep：分隔符，即分词依据

例 14

```
In [27]:from nltk.tokenize.util import string_span_tokenize
list(string_span_tokenize(text, " "))
Out[27]:[(0, 4),(5, 12),(13, 17),(18, 26),(27, 30),(31, 36),(37, 37),(38, 44),(45, 48)
,(49, 55),(56, 58),(59, 72)]
```

4.2.1.2 去除停用词

文本经过简单的分词处理后，还会包含大量的无实际意义的通用词，需要过滤掉，NLTK 提供了一份英文停用词词典供使用者直接使用，可以通过以下方式查看停用词词典：

例 15

```
In [28]:from nltk.corpus import stopwords
    english_stopwords = stopwords.words("english")
    print english_stopwords[0:10]
    [u'i', u'me', u'my', u'myself', u'we', u'our', u'ours', u'ourselves', u'you',
u'your']
In [29]:len(english_stopwords)
Out[29]:153
In [30]:for i in word_tokenize("Everything is OK. I can do it all by myself."):
    if i not in english_stopwords:
        print i
Everything
OK
.
I
.
```

从上面的结果可以看出，对于停用词典中未涵盖的大写停用词和标点并未过滤掉，这就要求在去除停用词前进行小写化处理，后续再过滤掉多余的标点符号。

例 16

```
In [31]:english_punctuations = [',', '.', ':', ';', '?', '(', ')', '[', ']',
'!', '@', '#', '%', '$', '*'] # 自定义英文表单符号列表
for i in word_tokenize("Everything is OK. I can do it all by myself."):
    if i.lower() not in english_stopwords: # 过滤停用词
        if i not in english_punctuations: # 过滤标点符号
            print i
Everything
OK
```

4.2.1.3 词干化处理

词干化处理(Stemming)就是去除形态词缀得到对应词根的过程，是英文特有的处理过程，比如说同一个英文单词有单数复数的变形（如 apple 和 apples ）、ing 和 ed 等时态的变形（ doing 和 did ）、人称代词不同谓语的变形等（ like 和 likes ），这些词虽然形式上有细微差别，但是都对应着相同的词根，在某些情况下应该当做相同的词处理（比如计算相关性），这就需要进行词干化处理。

NLTK 的 stem 包提供了几个相关模块进行词干化处理，包括 Lancaster Stemmer, Porter Stemmer （词干化处理有三大主流算法：Porter Stemming 、 Lovins stemmer 和 Lancaster Stemming ）。

(1) lancaster 模块

lancaster 模块是基于 Lancaster Stemming 算法的词干分析模块，该模块下定义了 LancasterStemmer 类，通过调用该类下的 stem 方法可以实现英文词汇的词干化处理，调用方式为：实例.stem(word)

例 17

```
In [32]:from nltk.stem.lancaster import LancasterStemmer
        st1 = LancasterStemmer()
        words=['fishing', 'crying', 'likes', 'meant', 'owed', 'was', 'did', 'done', 'wo
men', "avaliable"]
        for word in words:
            print word,st1.stem(word)
fish fishing
cry crying
lik likes
meant meant
ow owed
was was
did did
done done
wom women
avaliable avaliable
```

(2) porter 模块

porter 模块是基于 Porter Stemming 算法的词干分析模块，该模块下定义了 PorterStemmer 类，通过调用该类下的 stem 方法可以实现英文词汇的词干化处理，调用方式为：实例.stem(word)

例 18

```
In [33]:from nltk.stem.porter import PorterStemmer
        st2 = PorterStemmer()
        for word in words:
            print word,st2.stem(word),st1.stem(word)
fish fishing fish
cry crying cri cry
lik likes like lik
meant meant meant
ow owed owe ow
was wa was
did did did
done done done
wom women wom
avaliable avaliable avaly
```

(3) regexp 模块

`regexp` 模块是基于正则表达式识别词缀模式进行词干处理的模块，该模块下定义了 `RegexpStemmer` 类，通过调用该类下的 `stem` 方法可以实现英文词汇的词干化处理，实例化方式为：实例 = `RegexpStemmer(regexp, min=0)`

参数说明：

`regexp`：用来识别词缀的正则表达式

`min`：需要词干化处理词汇的最小长度，默认缺失值为 0

`stem` 方法调用方式为：实例.`stem(word)`

例 19

```
In [34]:from nltk.stem.regexp import RegexpStemmer
st3 = RegexpStemmer('ing$|s$|e$|able$', min=4)
for word in words:
    print word,st3.stem(word),st2.stem(word),st1.stem(word)
fish fishing fish fish
cry crying cry cry
like likes like like
meant meant meant meant
owe owed owe ow
was was wa was
did did did did
done done don done
wom women women women
avaliable avali avali avaly
```

4.2.1.4 NLTK 提供的语料库

(1) 下载语料库

`nltk.download()` 是 `Downloader` 类下的一个方法，可以用于下载 NLTK 语料和相关的包，利用该方法下载 NLTK 语料的步骤如下：

In [1]:`nltk.download()`

@todo 插入下载截图 4.2.1.4-1

在“Downloader>”后输入框输入“d”，回车确认后在下方出现新的 `Identifier` 输入框：

@todo 插入截图 4.2.1.4-2

此时可以直接输入需要下载的包的名称，也可以输入“!”查看所有包的列表：

@todo 插入截图 4.2.1.4-3

在 `Identifier` 输入框输入“book”，回车确认下载 NLTK 图书语料库合集

@todo 插入截图 4.2.1.4-4

在 Identifier 输入框输入“all-corpora”，回车确认下载 NLTK 全部语料库

(2) 查看下载的语料

执行 `nltk.download()` 命令后，在“Downloader>”后输入框输入“l”，回车确认后可以查看已经下载的包：

@todo 插入截图 4.2.1.4-5

在上图 list 的第九行可以看到已经下载的 Brown 语料，继续回车可以继续向下查看。对于已经下载的语料库，可以通过以下方式查看：

```
In [2]:from nltk.corpus import brown
print brown.readme() # readme 方法可以返回语料库内容
BROWN CORPUS

A Standard Corpus of Present-Day Edited American
English, for use with Digital Computers.

by W. N. Francis and H. Kucera (1964)
Department of Linguistics, Brown University
Providence, Rhode Island, USA

Revised 1971, Revised and Amplified 1979

http://www.hit.uib.no/icame/brown/bcm.html

Distributed with the permission of the copyright holder,
redistribution permitted.

In [3]:from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

以上执行结果列举了几个 NLTK Book 的示例，输入语料名称即可直接查看语料：

In [4]:text1 Out[4]:

text1 属于 Text 类，在 Text 类中，NLTK 提供了一些常用的文本搜索方法：

(2.1) collocations

collocations 方法用于搜索 Text 文本中的去除停用词后的固定搭配词组，可以自主设置固定搭配的限制条件，调用方式为：Text.collocations(num=20, window_size=2)

参数说明：

num：输出搭配词组个数的最大值，默认缺失值为 20

window_size：形成固定搭配词组中的单词之间可以间隔的距离，即间隔超过该参数值的单词就不能形成词组，默认缺失值为 2，也是可以设置的最小值

```
In [5]:text1.collocations(num=10, window_size=2)
Sperm Whale; Moby Dick; White Whale; old man; Captain Ahab; sperm
whale; Right Whale; Captain Peleg; New Bedford; Cape Horn
```

(2.2) common_contexts

common_contexts 方法可以查找给定单词的上下文，并给出最相似的频繁出现的上下文结构，调用方式为：Text.common_contexts(words, num=20)

参数说明：

words：用于做相似检索的单词，多个单词需放在同一个列表中并用逗号间隔

num：允许生成的词汇数，默认缺失值为 20

```
In [6]:text1.common_contexts(["very", "great"], num=10) #查找使用"very"或"great"的相同结构的
上下文
the_body a_white a_long
```

(2.3) concordance

concordance 方法可以搜索文本中指定词语出现的位置，输出词语所在上下文，调用方式为：Text.concordance(word, width=79, lines=25)

参数说明：

word：需要检索的词语

width：输出的上下文的长度，默认缺失值为 79

lines：输出的上下文的行数，默认缺失值为 25

```
In [7]:text1.concordance("very",lines=5)
Displaying 5 of 322 matches:
horse - whales , which had bones of very great value for their teeth , of whi
n inward bruise ." -- KING HENRY . " Very like a whale ." -- HAMLET . " Which
itself , " said Mr . Webster , " is a very striking and peculiar portion of the
egree , some time or other , cherish very nearly the same feelings towards the
o eat and sleep meanwhile . It was a very dubious - looking , nay , a very dar
```

(2.4) count

count 方法用于统计文中某个单词出现的次数，调用方式为：Text.count(word)，参数 “word” 即为需要统计的单词。

```
In [8]:text1.count("very")
Out[8]:311
```

(2.5) similar

similar 方法用于查找与指定单词有相同使用语境的单词，会先列出相似度最高的单词，调用方式为：Text.similar(word, num=20)，参数 num 设置生成的单词数。

```
In [8]:text1.similar("very")
so a same the but last first too and as in pretty only other white that is stra
nge now entire
```

(2.6).findall

.findall 方法用于找出文中符合指定正则表达式形式为文本内容，正则表达式中的符号需用尖角括号 (<>) 括起，调用方式为：Text.findall(regexp)

```
In [9]:text1.findall("<a>(<. *>)<man>")
monied; nervous; dangerous; white; white; pious; queer; good;
mature; white; Cape; great; wise; wise; butterless; white; fiendish;
pale; furious; better; certain; complete; dismasted; younger; brave;
brave; brave; brave
```

对于不属于 NLTK 文本库的其他字符串，可以利用 Text() 函数将其转换为 Text 对象，转换为 Text 对象后上述方法都可以用于字符串的分析。

4.2.1.5 tag 词性标注包

tag 包除定义了一些词性标注的类外，还提供了部分词性标注的接口。它定义的几个词性标注器均以分词结果列表作为输入，对应返回每一个分词结果的词性，多数标注器都是根据训练语料构建的，比如一元语法模型（unigram）词性标注器，对于给出的词汇，该标注器会在训练语料中查找每个词汇出现最多的词性并对其进行相应的标注，对于训练集中不存在的词汇，其词性会被标注为“None”。

(1) 词性标注函数 pos_tag

函数 pos_tag 是利用 NLTK 推荐的词性标注器对指定词汇列表进行词性标注的函数，词性标注结果以列表形式返回，列表元素为词汇和对应词性构成的元组。调用方式为：

`pos_tag(tokens, tagset=None)`

参数说明：

`tokens`：需要进行词性标注的词汇列表

`tagset`：使用的词性标记集。同一词性可以有不同的标注词，利用该参数可以进行标注词的规约，例如 universal 标记集，其标注词及含义如下表所示：

@todo 插入标注词含义表

例 1

```
In [1]:from nltk.tag import pos_tag
        from nltk.tokenize import word_tokenize
        pos_tag(word_tokenize("Good muffins cost $3.88\nin New York. Please buy me\ntwo of them.\nThanks."))
Out[1]:[('Good', 'JJ'), ('muffins', 'NNS'), ('cost', 'VBP'), ('$', '$'), ('3.88', 'CD'), ('in', 'IN'), ('New', 'NNP'), ('York', 'NNP'), ('.', '.'), ('Please', 'NNP'), ('buy', 'VB'), ('me', 'PRP'), ('two', 'CD'), ('of', 'IN'), ('them', 'PRP'), ('.', '.'), ('Thanks', 'NNS'), ('.', '.')]
In [2]:pos_tag(word_tokenize("Good muffins cost $3.88\nin New York. Please buy me\ntwo of them.\nThanks."), tagset='universal')
Out[2]:[('Good', u'ADJ'), ('muffins', u'NOUN'), ('cost', u'VERB'), ('$', u'.'), ('3.88', u'NUM'), ('in', u'ADP'), ('New', u'NOUN'), ('York', u'NOUN'), ('.', u'.'), ('Please', u'NOUN'), ('buy', u'VERB'), ('me', u'PRON'), ('two', u'NUM'), ('of', u'ADP'), ('them', u'PRON'), ('.', u'.'), ('Thanks', u'NOUN'), ('.', u'.')]
```

词性标注的一个比较重要的应用领域就是智能朗读，即将文本转换为语音，如果一个词汇同时有多种词性并且有不同的发音，那么准确识别词性对于语音准换是至关重要的，如下例展示的情况。

例 2

```
In [3]:pos_tag(word_tokenize("They desert the treasure in the desert."), tagset='universal')
Out[3]:[('They', u'PRON'), ('desert', u'VERB'), ('the', u'DET'), ('treasure', u'NOUN'), ('in', u'ADP'), ('the', u'DET'), ('desert', u'NOUN'), ('.', u'.')]
```

语句"They desert the treasure in the desert."中，词汇“desert”出现了两次，但是两次的词性都不相同，前一次是动词，发音为“dɪ'zɜ:t”，后一次是名词，发音为“dɛzət”。利用词性标注函数 pos_tag 可以正确识别该词的词性，如果需要转换为语音的话就不会出现读音错误。

(2) 词性标注函数 pos_tag_sents

函数 pos_tag_sents 是利用 NLTK 推荐的词性标注器对指定语句列表进行词性标注的函数，每一个语句都由词汇列表构成。调用方式为：pos_tag_sents(sentences, tagset=None)

参数说明：

sentences：需要进行词性标注的语句列表，每一个语句都由词汇列表构成

tagset：使用的词性标记集，如 universal, wsj, brown 等

例 3

```
In [4]:from nltk.tag import pos_tag_sents
        from nltk.tokenize import sent_tokenize
        sents="Good muffins cost $3.88\nin New York. Please buy me\nntwo of them.\nThanks."
        pos_tag_sents([word_tokenize(i) for i in sent_tokenize(sents)]) #利用 sent_tokenize 进行分句后再通过 word_tokenize 进行分词，得到分词后的语句列表
Out[4]:[[('Good', 'JJ'), ('muffins', 'NNS'), ('cost', 'VBP'), ('$', '$'), ('3.88', 'CD'), ('in', 'IN'), ('New', 'NNP'), ('York', 'NNP'), ('.', '.'), [('.', 'PRP'), ('buy', 'VB'), ('me', 'PRP'), ('two', 'CD'), ('of', 'IN'), ('them', 'PRP'), ('.', '.')], [('.', 'NNS')], ('.', '.')]]
In [5]:pos_tag_sents([word_tokenize(i) for i in sent_tokenize(sents)], tagset='universal')
Out[5]:[[('Good', u'ADJ'), ('muffins', u'NOUN'), ('cost', u'VERB'), ('$', u'.'), ('3.88', u'NUM'), ('in', u'ADP'), ('New', u'NOUN'), ('York', u'NOUN'), ('.', u'.')], [('.', u'NOUN'), ('buy', u'VERB'), ('me', u'PRON'), ('two', u'NUM'), ('of', u'ADP'), ('them', u'PRON'), ('.', u'.')], [('.', u'NOUN')], ('.', u'.')]]
```

(3) StanfordPOSTagger 类

stanford 是 tag 包中提供 Stanford 标注器接口的模块，StanfordPOSTagger 是该模块下定义的词性标注类。要使用该模块，需要事先下载标注模型，下载网址为：

<http://nlp.stanford.edu/software>。该类实例化方式为：实例 =

`StanfordPOSTagger(model, path, encoding="UTF-8")`

参数说明：

`model`：基于训练集的模型

`path`：可选参数，Stanford 标注器文件路径

`encoding`：可选参数，训练集编码，默认缺失值为 "UTF-8"

实例化后可以调用 `tag` 方法进行词性标注，调用方式为：实例.`tag(tokens)`，其中参数 `tokens` 即为需要词性标注的词汇列表。

例 4

```
In [6]:from nltk.tag import StanfordPOSTagger
       st = StanfordPOSTagger('english-bidirectional-distsim.tagger')
       st.tag('what is the airspeed of an unladen swallow ?'.split())
Out[6]:[('what', 'WP'), ('is', 'VBZ'), ('the', 'DT'), ('airspeed', 'NN'), ('of', 'IN'),
       ('an', 'DT'), ('unladen', 'JJ'), ('swallow', 'VB'), ('?', '.')]
```

此外，`tag` 包还提供了多个词性标注模块，包括：基于统计词性标注器 TnT 的模块 `tnt`、基于转换规则的词性标注模块 `brill`、基于 CRFSuite 的词性标注模块 `crf` 等，想进一步学习使用相关模块可以参见 4.4 节提供的深度阅读材料。

(4) 读取已标注语料

在 NLTK 提供的部分语料中已经标注了词性，使用 `tagged_words` 方法即可查看已经标注的词性。

例 5

```
In [7]:nltk.corpus.brown.tagged_words(tagset='universal')
Out[7]:[(u'The', u'DET'), (u'Fulton', u'NOUN'), ...]
```

4.2.1.6 亚马逊英文评论语料分析实例

(1) 数据抓取

抓取亚马逊美国站上一款鞋子（<https://www.amazon.com/FAYALE-Driving-Cowhide-Leather-Lace-Up/dp/B01ASME6VI>）的评论语料作为分析实例，共抓取评论文本 187 条，保存为 csv 格式文件 `shoes_review.csv`，并上传到 Jupyter Notebook 文件列表，新建一个 Python notebook，读取文本数据，并将数据保存到列表 `corpus` 中：

```
In [1]:import csv
corpus=[]
with open("shoes_review.csv") as f:
    reader=csv.reader(f)
    for i in reader:
        corpus.append(i)
```

查看 corpus 内元素形式：

```
In [2]:corpus[0:3]
Out[2]:[['id','product_id','author_name','author_id','helpful','rating','has_purchased',
'summary','content','published_at','created_at','updated_at','meta'],
['R3BANSXTDHC16U','B017D40RMM','mari','A1SOIEM6ZWUK9B','','4','TRUE','great shoes',
'Great color ..... trendy style.','2016-07-16 00:00:00','2016-07-19 15:47:56','2016-07-
19 15:47:56','{"Size": "6 B(M) US", "Color": "Wine Red", "variation_attribute": "Size:
6 B(M) US|Color: Wine Red"}'],
['R37EXA4Y7A32CS','B017D40RMM','JR Taylor','A1QHRBBD0063F1','','2','TRUE','no return
information and feel like an 8',"I wear ..... at all.",'2016-07-12 00:00:00','2016-07-19
15:47:56','2016-07-19 15:47:56','{"Size": "9 B(M) US", "Color": "White", "variation_a
ttribute": "Size: 9 B(M) US|Color: White"}']]
```

可以看到 corpus 第一个列表元素即为 csv 文件的第一行，即抓取的字段，包括：id、product_id、author_name、author_id、helpful、rating、has_purchased、summary、content、published_at、created_at、updated_at、meta，分别代表评论id、商品id、评论者姓名、评论者id、评论是否有帮助、评论得分、是否真实购买、评论摘要、评论内容、评论发布时间、评论创建时间、评论更新时间、商品属性，利用这些丰富的信息可以得到很有价值的分析结论。

(2) 分词处理

(2.1) 提取评论文本

将商品所属子类 id 为 3 的商品评论文本保存到列表 review 中：

```
In [3]:review=[]
    with open("shoes_review.csv") as f:
        reader=csv.reader(f)
        reader.next() # 跳过第一行字段内容
        for i in reader:
            review.append(i[8].decode('utf-8'))#以文件保存格式对内容进行解码，获得unicode
字符串
In [4]:len(review) # 查看评论数
Out[4]:187
In [5]:for i in review[25:28]:
    print i + "\n"
    They fit very comfortably. I am a nurse working 12 hour shifts and these are ve
ry easy on my feet. They are as close to barefoot and still wear shoes. No arch suppor
t, but great for wide feet.

    would buy another pair! Very comfortable!!!

    Wore it 2 times and the stitching started coming undone.
```

(2.2) 分句

对提取的英文评论文本进行分句处理，使用的是 `sent_tokenize` 函数，分句的结果保存在列表 `sent` 中。

```
In [6]:import nltk
    from nltk.tokenize import sent_tokenize
    sent=[]
    for i in review:
        sent.append(sent_tokenize(i))
In [7]:sent[0:3]
Out[7]:[[u'Great color, wear well but bought a size larger as recommended, but should
have bought my reg.',
         u'shoe size.',
         u'Very comfortable but had to put in some inner soles to make it fit better.'
```

`,`

```
         u'Would like to try another color but will buy my size.',
         u'Many compliments on this trendy style.'],
         [u'I wear a nine- period.',
          u'I read the reviews and ordered my exact size.',
          u'The shoes arrived with no order form, no return information and feel like a
n 8.'],
         u'I will never be able to wear them.',
         u'"FYI, they are as cute in person as online, but I'm stuck with a shoe I can'
t wear...at all."],
         [u'Very comfortable']]
```

从分句的结果可以看出，每一条评论文本都被分割成若干句，且保存在一个列表中。

(2.3) 分词

对以上分句的结果进一步做分词处理，这里使用最常用的分词函数 `word_tokenize`，分词的结果保存在列表 `words` 中。

```
In [8]:from nltk.tokenize import word_tokenize
words=[]
for i in sent:
    for j in i:
        words.extend(word_tokenize(j))
In [9]:words[0:3]
Out[9]:[u'Great', u'color', u',']
```

(2.4) 小写处理

分词结果中有部分大写的字母，为了提高后续去除停用词、语意分词等过程的处理、分析效果，需要先进性小写处理，直接利用字符串的 `lower` 方法即可，将处理结果保存在列表 `words_lower` 中。

```
In [10]:words_lower=[i.lower() for i in words]
words_lower[0:3]
Out[10]:[u'great', u'color', u',']
```

(2.5) 去除标点符号和停用词

分词结果列表中还存在大量的标点符号和停用词，利用自定义标点符号列表以及 NLTK 提供的停用词典进行过滤，过滤后的分词结果保存在 `words_clear` 列表中。

```
In [10]:from nltk.corpus import stopwords
english_stopwords = stopwords.words("english")
english_punctuations = [',', '.', ':', ';', '?', '(', ')', '[', ']', '!', '@',
'#', '%', '$', '*', '...'] # 自定义英文表单符号列表
words_clear=[]
for i in words_lower:
    if i not in english_stopwords: # 过滤停用词
        if i not in english_punctuations: # 过滤标点符号
            words_clear.append(i)
In [11]:print "/".join(words_clear[0:10])
great/color/wear/well/bought/size/larger/recommended/bought/reg
```

(2.6) 词干化处理

利用 porter 模块即 Porter Stemming 算法进一步进行词干化处理，将词干化后的结果保存在列表 words_stem 中。

```
In [12]:from nltk.stem.porter import PorterStemmer
st = PorterStemmer()
words_stem=[st.stem(word) for word in words_clear]
```

(2.7) 简单的统计汇总

经过以上几步的处理，可以初步得到一份比较清爽的分词结果，读者可以根据实际需求选择合适的分词、词干化等方法，并且可以在此结果基础上进一步过滤掉较短的单词、纠正拼写错误的单词等等，下面仅利用分词的结果做几个简单的统计汇总。

利用函数 Text() 将分词结果转换为 Text 格式，名称为 word_text

```
In [13]:from nltk.text import Text
word_text=Text(words_stem)
```

识别评论文本中常用固定词组搭配：

```
In [14]:word_text.collocations(num=20, window_size=2)
arch support; love shoe; true size; mani compliment; differ color;
read review; fit perfect; super comfort; everi color; wide feet; dark
blue; well made; second pair; appreci fayal; dissip quickli; skirt
short; someon said; extrem comfort; car around; skinni jean
```

结果显示，出现次数最多的词组为 arch support（足弓垫），此外 fit perfect、super comfort、well made、extrem comfort 等经常出现的好评词组显示了评论者较高的评价。

利用 Counter 计数器统计出现次数最多的前 20 个单词：

```
In [15]:from collections import Counter
words_counter=Counter(words_stem)
words_counter.most_common(20)
Out[15]:[(u'shoe', 152),(u'comfort', 93),(u'love', 63),(u'color', 47),(u'fit', 43),(u'wear', 41),(u'order', 40),(u'size', 39),(u'cute', 37),(u'pair', 35),(u'n't', 32),(u'like', 31),(u'look', 28),(u'great', 23),(u'day', 22),(u'feet', 22),(u'realli', 22),(u'buddy', 22),(u'comfi', 20),(u'one', 19)]
```

计数结果显示，出现次数最多的词是“shoe”，共出现了 152 次，其次是 comfort、love、color 等。通过查看高频词上下文相关内容，可以了解评论的具体内容：

```
In [16]:word_text.concordance("comfort",lines=10)
Displaying 10 of 93 matches:
comfort put inner sole make fit better wou
l
son onlin 'm stuck shoe ca n't wear comfort 've gotten plenti compliment shoe
c
t 've gotten plenti compliment shoe comfort wear day wear true 9 long toe foun
d
orter toe hit end rather wide still comfort bought arch insert arch support co
m
ort bought arch insert arch support comfort shoe bought love help prevent feet
r
rch shoe meet qualif worth tri cute comfort shoe hesit order review state righ
t
shoe imposs keep side shoe lace tie comfort fit great rub even lot walk love m
u
t skinni jean etc feel like slipper comfort love first pair black order order
p
pair packag came felt like noth bag comfort order differ style color ca n't be
a
ld back case return love hope order comfort love size fit perfect need buy dif
f
```

如果想直接查看原始评论文本，可以通过索引查看。

(3) 词性标注

```
In [17]:from nltk.tag import pos_tag
pos_tag(words_stem,tagset='universal')
Out[17]:[(u'great', u'ADJ'),
          (u'color', u'NOUN'),
          (u'wear', u'NOUN'),
          ...]
```

筛选出形容词和名词，分别保存在列表 ADJ 和 NOUN 中：

```
In [18]:ADJ=[]
    NOUN=[]
    for a,b in pos_tag(words_stem,tagset='universal'):
        if b=="ADJ":
            ADJ.append(a)
        elif b=="NOUN":
            NOUN.append(a)
In [19]:len(ADJ) # 查看形容词个数
Out[19]:523
In [20]:len(NOUN) # 查看名词个数
Out[20]:1203
```

查看出现次数较多的形容词和名词：

```
In [21]:c1=Counter(ADJ)
    for i in c1.most_common(10):
        print i[0],i[1]
    fit 39
    cute 27
    great 23
    super 13
    nice 13
    perfect 11
    big 10
    red 10
    right 9
    much 9
In [22]:c2=Counter(NOUN)
    for i in c2.most_common(10):
        print i[0],i[1]
    shoe 131
    comfort 88
    color 42
    order 40
    size 39
    love 33
    pair 32
    day 22
    feet 22
    comfi 19
```

从词性标注统计结果可以初步推断，购买者最常用来描述商品的形容词是“fit”，即“合脚”，此外“cute”、“great”等词出现的次数也比较多，除了名词“shoe”以外，出现次数较多的名词有“comfort”、“color”、“size”等，可以初步推断购买者更加关注鞋子的舒适度、颜色以及尺寸。

4.2.2 Pattern

Pattern 工具库中的 `en` 模块提供了英文词性标注、情感分析、动名词变换等工具，其中 `Parser` 类及相关方法可以实现英文分词和词性标注处理，使用的是一个含有 100000 个词汇及对应词性的词典，对于词典中未包含的词汇，根据词汇后缀及上下文词汇进行判断，准确率在 95% 左右，对于用词不规范的情境下准确率会更低。词性简写对照表可以参见官方文档（<http://www.clips.ua.ac.be/pages/mbsp-tags>）。

4.2.2.1 tag 函数

通过调用函数 `tag` 可以直接将英文语句切分并以元组形式返回词语及词语的词性标注结果，调用方式如下：`tag(string, tokenize=True, encoding='utf-8', **kwargs)`

参数说明：

`string`：待处理英文语句

`tokenize`：在分词时是否将标点符号和单词分开

`encoding`：所输入的英文语句的编码

例 1

```
In [1]:import pattern
        from pattern.en import tag
tag("I eat *pizza !with a fork.", tokenize=False)
Out[1]:[(u'I', u'PRP'),
        (u'eat', u'VBP'),
        (u'*pizza', u'NN'),
        (u'!with', u'IN'),
        (u'a', u'DT'),
        (u'fork.', u'NN')]
In [2]:tag("I eat *pizza !with a fork.")
Out[2]:[(u'I', u'PRP'),
        (u'eat', u'VBP'),
        (u'*', u'SYM'),
        (u'pizza', u'NN'),
        (u'!', u'.'),
        (u'with', u'IN'),
        (u'a', u'DT'),
        (u'fork', u'NN'),
        (u'.', u'.')]
```

4.2.2.3 parse 函数

通过调用函数 `parse` 可以直接将英文语句切分并返回词性标注结果，除此之外还可以识别句子成分、进行词干化处理等等。调用方式如下：`parse(string, tokenize=True, tags=True, chunks=True, relations=False, lemmata=False, encoding='utf-8', **kwargs)`

参数说明：

`string`：待处理英文语句

`tokenize`：在分词时是否将标点符号和单词分开

`tags`：是否进行词性标注

`chunks`：是否切分词组

`relations`：是否识别句子成分，如主语、宾语

`lemmata`：是否进行词干处理

`encoding`：所输入的英文语句的编码

例 2

```
In [3]:from pattern.en import parse
        parse('I eat pizza with a fork.')
Out[3]:u'I/PRP/B-NP/O eat/VBP/B-VP/O pizza/NN/B-NP/O with/IN/B-PP/B-PNP a/DT/B-NP/I-PN
P fork/NN/I-NP/I-PNP ././O/O'
```

输出结果为字符串形式，每个词的分析结果用空格分开，以“`I`”为例，结果“`I/PRP`”中，第一项“`PRP`”表示词性“pronoun, personal”，即人称代词，其余表示词组类型。`Pattern` 提供了 `pprint` 函数美化以上输出结果。

```
In [4]:from pattern.en import pprint
        pprint(parse('I eat pizza with a fork.', relations=True, lemmata=True))

      WORD   TAG    CHUNK   ROLE    ID     PNP    LEMMA
      I     PRP     NP     SBJ     1      -      i
      eat   VBP     VP     -       1      -      eat
      pizza  NN     NP     OBJ     1      -      pizza
      with  IN     PP     -       -      PNP    with
      a     DT     NP     -       -      PNP    a
      fork  NN     NP ^   -       -      PNP    fork
      .     .     -     -       -      -      .
```


4.3 小结

本章重点介绍自然语言处理的基础步骤——分词，鉴于中文分词和英文分词在分词原理及分词工具上的差异，分别结合实例向读者介绍了中文分词、英文分词的常用 Python 工具包，演示了一般的中英文分词步骤，对分词中常见的自定义词典、停用词处理等问题给出了相应的解决方案。按照书中的操作步骤，初学者可以很快的掌握并实现简单的分词过程。下面一起回顾一下本章的主要内容：

- (1) 中文分词工具包：jieba、Yaha、Genius、finalseg 等
- (2) 英文分词工具包：NLTK

各工具包中常用的模块、函数、类方法等汇总表如下：

@todo 插入汇总表

(3) NLTK 语料库的下载及查看，包括提取固定搭配词组、查找使用情境（上下文）相近的词语、计数等

- (4) 亚马逊中国站中文评论文本、亚马逊美国站英文评论文本分词实例

4.4 深入阅读材料

4.4.1 本章介绍的分词工具在 **github** 都有比较详细的说明，在此统一列出各项目主页地址方便读者查阅：

- (1) jieba 分词：<https://github.com/fxsjy/jieba>
- (2) Yaha 分词：<https://github.com/jannson/yaha>
- (3) Genius 分词：<https://github.com/duanhongyi/genius>
- (4) finalseg 分词：<https://github.com/fxsjy/finalseg>
- (5) scseg 分词：<https://github.com/duanhongyi/scseg>
- (6) pynlpir 分词：<https://github.com/tsroten/pynlpir>
- (7) smallseg 分词：<https://github.com/wangjun/smallseg>
- (8) snailseg 分词：<https://github.com/Abioy/snailseg>

4.4.2 《Natural Language Processing with Python》一书对于 **MLTK** 有着较为基础、详尽的介绍，对于想进一步了解 **NLTK** 的读者是不错的学习资料，官方网页版书籍地址：<http://www.nltk.org/book/>

4.4.3 **NLTK** 相关包、模块文档网址：

- (1) tokenize 包文档：<http://www.nltk.org/api/nltk.tokenize.html>
- (2) stem 包文档：<http://www.nltk.org/api/nltk.stem.html>
- (3) text 模块文档：<http://www.nltk.org/api/nltk.html#module-nltk.text>
- (4) tag 包文档：<http://www.nltk.org/api/nltk.tag.html>

4.4.4 在关于自然语言处理的书籍中，都会将词性标注单列一章重点讲解，对此有兴趣的读者可参考《自然语言处理综论》第一版第**8**章或《统计自然语言处理基础》第**10**章

4.4.5 关于Brwon语料库标记集的详细信息可参考：<http://www.comp.leeds.ac.uk/amalgam/tagsets/brown.html>

4.4.6 关于计算所汉语词性标记集的详细信息可参考：http://www.ictclas.org/ictclas_docs_003.html

4.5 练习

4.5.1 总结一个完整的中文分词过程应该包括哪些具体步骤

4.5.2 从公开数据源下载中文文本数据，运用 **jieba** 分词进行分词处理，必要情况下根据实际数据特征添加自定义词典、设置指定词语在词典中的词频等，最后根据分词结果进行简单的统计分析

4.5.3 对比 **4.1** 节给出的不同中文分词工具以及同一分词工具下不同分词函数的分词原理、特点，并用文本数据对比实际分词效果

4.5.4 利用网络爬虫技术自行抓取英文文本数据，利用 **NLTK** 提供的不同的分词函数进行分词处理，并进行停用词过滤、大小写转换等基本操作，总结一个完整的分词过程应该包括哪些具体步骤并按步骤执行

4.5.5 下载 **NLTK** 提供的语料库，并对 **Text** 文本进行计数、查询等操作

第 5 章 文本数据结构化处理

虽然文本数据的分析流程与传统数据挖掘相似，但文本数据表现为非结构性、自由形态的文字，或者是由许多符合特定计算机语言的语法及语法规则构成的文字和语句的字符串，利用现有数据挖掘方法是无法直接进行分析的。抛开词频之外的复杂语义结构，挖掘过程首先要考虑将这种非结构化的数据结构化处理，常规做法就是分词、生成文档-词项矩阵，后续可能涉及到高维矩阵的处理等问题。所以，文本挖掘需要建立在文本结构化转换基础上，于结构化框架下才能进行分析，本章将重点讲解几种常用的文本数据结构化处理的方法，包括文档-词项矩阵（DTM）、词频-逆向文件频率(TF-IDF)和词向量，具体内容安排如下：5.1节主要介绍文档-词项矩阵的思想和构建方法，包括 `sklearn` 和 `gensim` 库的相关模块、类方法等；5.2节进一步讲解最为常用的提取文本数据核心信息的统计方法：词频-逆向文件频率，分别介绍了使用 `sklearn` 和 `gensim` 库计算词频-逆向文件频率的方法；在 5.3 节引入词向量这一更加高级的文本数据结构化处理方法，重点说明利用 `word2vec` 工具包进行词向量训练的方法；最后 5.4 节结合实际案例讲解了相关 Python 工具的使用方法。

5.1 文档-词项矩阵

5.1.1 词袋模型简介

“词袋模型”一词源自“Bag of words”，简称 BOW，是构建文档-词项矩阵的基本思想。对于给定的文本，可以是一个段落，也可以是一个文档，该模型都忽略文本的词汇顺序和语法、句法，假设文本是由无序、独立的词汇构成的集合，这个集合可以被直观的想象成一个词袋，袋子里面就是构成文本的各种词汇。例如，文本内容为“经济发展新常态研究”的文档，用词袋模型可以表示为[经济，发展，新常态，研究]四个独立的词汇。**词袋模型对于词汇的独立性假设，简化了文本数据结构化处理过程中的计算，被广泛采用，但是另一方面，这种假设忽略了词汇之间的顺序和依赖关系，降低了模型对文本的代表性。**

5.1.2 文档-词项矩阵

“文档-词项矩阵”一词源自“Document-Term Matrix”，简称 DTM，DTM 矩阵转置后即为 TDM。我们在第一章简单介绍过文档-词项矩阵的构成，直观来看，矩阵的行代表文档，列代表词汇，矩阵元素即为文档中某一词汇出现的次数。例如，有以下两个文档：文档一[经济，发展，新常态，研究]，文档二[大数据，安全，隐私，保护]，基于这两个文档构造一个词典：{1：“经济”，2.“发展”，3.“新常态”，4.“研究”，5.“大数据”，6.“安全”，7.“隐私”，8.“保护”}，这个词典一共包含 8 个不同的词汇，利用词典的索引号，上面两个文档都可以用一个 8 维的向量表示：(1,1,1,1,0,0,0,0) 和 (0,0,0,0,1,1,1,1)，向量元素表示对应维度的词汇在文档中出现的次数，两个向量合并在一起即得到**文档-词项矩阵**。

虽然文档-词项矩阵没有考虑到词汇之间的依存关系，但是这一简单假设也大大简化了后续文本挖掘的计算过程，利用结构化处理的文档-词项矩阵已经可以实现很多有意义的分析过程，如计算文档之间的相关性、文本分类、文本聚类等等。

5.1.3 利用 scikit-learn 库构建文档-词频矩阵

除了常用的机器学习算法外，scikit-learn 库还提供了很多数据结构化处理的工具，将这类结构化处理统称为“Feature Extraction”，即“特征抽取”，文本中的词汇出现的次数就属于“特征”中的一种。通过 `sklearn.feature_extraction` 包实现相关操作，该包包括从文本和图像中进行特征抽取的方法。

5.1.3.1 `sklearn.feature_extraction.text.CountVectorizer`

`sklearn.feature_extraction.text` 是 `sklearn.feature_extraction` 包中进行文本数据结构化处理的模块，其中定义的 `CountVectorizer` 类可以同时实现分词处理和词频统计，并得到文档-词频矩阵。实例化方式为：实例 = `CountVectorizer(input=u'content', encoding=u'utf-8', decode_error=u'strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop_words=None, token_pattern=u'(?u)\\b\\w\\w+\\b', ngram_range=(1, 1), analyzer=u'word', max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=)`

参数说明：

`input`：有以下三种取值类型

- (1) `filename`：文本内容的文件名
- (2) `file`：有“read”方法的对象，如 `file` 对象
- (3) `content`：需要处理的文本

`encoding`：解码参数，默认取值为“utf-8”

`decode_error`：若需要分析的字符串中包含未能解码字符，可以利用该参数设置处理方案，有以下三种方案：

- (1) `strict`：默认缺失值，出现异常报错
- (2) `ignore`：忽略异常情况
- (3) `replace`

`analyzer`：指定特征项为词(word)还是 n-grams 字符（按照 n 个字符对语句进行划分），有以下几种取值：

- (1) `word`：指定特征项为词
- (2) `char`：指定特征项为 n-grams 字符
- (3) `char_wb`：仅从文本中词边界创建 n-gram 字符

如果传递一个用来提取特征的可调用函数，那么就按照被传递的函数进行处理

preprocessor：利用可调用函数改写预测处理函数，同时保留分词和 n-grams 的处理过程，
默认缺失值为“None”

tokenizer：利用可调用函数改写分词步骤，同时保留预处理和 n-grams 的处理过程，
默认缺失值为“None”

ngram_range：设置 n-gram 字符中 “n” 上下界的参数，取值类型为数组(min_n, max_n)，所有 $\min_n \leq n \leq \max_n$ 的 n 值都会被使用

stop_words：停用词设置参数，有以下三种取值：

- (1) 字符串“english”：使用内建的英文停用词表
- (2) 自定义停用词列表：列表中词汇将会从分词结果中删除，只有当参数 `analyzer == 'word'` 时才可以进行此项设置
- (3) None：不使用停用词，可以将参数 `max_df` 取值设置为 [0.7, 1.0) 基于内部语料库词频自动识别、过滤停用词

lowercase：在分词前是否将所有字符都转换为小写形式，默认缺失值为“True”

token_pattern：规定分词原理的正则表达式，仅在 `analyzer == 'word'` 时才可设置。默认的正则表达式是选择两个或者两个以上的字符（忽略标点符号，将其作为分词依据）

max_df：阈值参数，构建字典时，忽略词频明显高于该阈值（语料库的停用词）的词项。如果参数取值是浮点数，则代表了文档比例，如果是整数，则代表计数值。当字典非空时，这个参数会被忽略。

min_df：阈值参数，构建字典时，忽略词频明显低于该阈值的词项，也被成为截止值。如果参数取值是浮点数，则代表了文档比例，如果是整数，则代表计数值。当字典非空时，这个参数会被忽略。

max_features：如果该参数取值非 None，构建词典的时候仅仅考虑语料库里词频最高的那些特征，如果词典非空，这个参数将被忽略。

(1) `fit_transform` 方法

对 CountVectorizer 类调用 `fit_transform` 方法可以得到文档词项矩阵，调用方式为：实例.`fit_transform(raw_documents)`，`raw_documents` 即为需要结构化处理的字符串或 file 对象。

下面将以 20 Newsgroups 数据库 (<http://www.qwone.com/~jason/20Newsgroups/>) 的数据进行说明，20 Newsgroups 数据库中包括大概 20000 个新闻报道文档，约覆盖 20 类不同的新闻报道，常用于进行文本分类和聚类分析。各位读者可以在先从网上下载数据集，再进行解压处理。下面的例子中，我们将使用 sklearn 内建函数 `fetch_20newsgroups` 加载 20 newsgroups 数据集，该函数定义在 `sklearn.datasets.twenty_newsgroups` 模块下，调用方式为：`fetch_20newsgroups(data_home=None, subset='train', categories=None, shuffle=True, random_state=42, remove=(), download_if_missing=True)`

参数说明：

`data_home`：可选参数，用于指定数据集下载和缓存的文件夹。默认缺失值为 `None`，数据将保存在 '`~/scikit_learn_data`' 子文件夹。

`subset`：可选参数，用于选择训练数据或测试数据，“`train`”表示训练集，“`test`”表示测试集，“`all`”表示两者均有，且为乱序。

`categories`：有以下两种取值情况

- (1) `None`（默认缺失值）：下载所有类别的新闻数据
- (2) 类别名称列表：只下载列表中类别的新闻数据，其他类别不下载

`shuffle`：可选参数，用于指定是否随机抽取数据，在模型假设样本独立同分布（i.i.d.）时，比如随机梯度下降算法，需要进行随机抽取。

`random_state`：`numpy` 随机数生成器，用于随机抽取数据。

`download_if_missing`：可选参数，默认缺失值为 `True`，若取值为 `False`，当数据本地不可用时，将产生 `IOError` 错误，而不是继续尝试从源站点下载数据。

`remove`：取值可以是元组（'`headers`', '`footers`', '`quotes`'）的任何子集，子集中的元素将会被识别并从数据集中移除，防止直接使用元数据进行分类造成的过拟合。其中。“`headers`”移除头文件，“`footers`”用于移除数据结尾类似签名的部分，“`quotes`”用于移除引用另一篇文章的部分。

例 1 从 20 Newsgroups 数据库下载实验数据

```
In [1]:import sklearn
    from sklearn.datasets import fetch_20newsgroups
    dataset = fetch_20newsgroups(shuffle=True, random_state=1, remove=('headers', 'footers', 'quotes'))
    dataset
Out[1]:{'DESCR':None,
    'data': [u"Well i'm not sure ..... It is unfortunate.\n",
        u"\n\n\n\n\n\n\nYeah, ..... III",
        ...],
    'description':'the 20 newsgroups by date dataset',
    'filenames':array(['.....'], dtype='|S97'),
    'target':array([17, 0, 17, ..., 9, 4, 9]),
    'target_names':['alt.atheism', 'comp.graphics', 'talk.politics.misc', 'talk.religion.misc']} # 新闻文本保存在键 "data" 对应的列表内，"target" 对应新闻类型，"target_names" 为不同新闻类型名称
In [2]:dataset.keys() # 查看数据集字典的键
Out[2]:['description', 'DESCR', 'filenames', 'target_names', 'data', 'target']
In [3]:len(dataset.data) # 查看文本数
Out[3]:11314
In [4]:data_samples = dataset.data[:10] # 选取 10 条新闻文本作为分析实例
In [5]:data_samples
Out[5]:[u"Well i'm not sure ..... It is unfortunate.\n",
    u"\n\n\n\n\n\n\nYeah, ..... III",
    ....
    u"\n I was wondering ..... for any insight.\n"]
```

例 2 获得 data_samples 文档-词频矩阵

```
In [6]:from sklearn.feature_extraction.text import CountVectorizer
    dtm_vectorizer = CountVectorizer()
    dtm = dtm_vectorizer.fit_transform(data_samples)
    dtm
Out[6]:<10x845 sparse matrix of type '<type 'numpy.int64'>' with 1133 stored elements in Compressed Sparse Row format> # dtm 为 10x845 的稀疏矩阵
In [7]:print dtm
    (0, 797) 1
    (0, 498) 3
    (0, 712) 1
    :
    (9, 153) 1
    (9, 684) 1
    (9, 726) 1
    (9, 359) 1
```

在以上输出结果中，每一行都以“ $(i, j) \ x$ ”的形式存在，其中， i 表示第 i 条新闻， j 表示词汇 id， x 表示词汇 j 在新闻 i 中出现的次数。转换为常用的矩阵形式如下：

```
In [8]:dtm.toarray()
Out[8]:array([[0, 0, 0, ..., 0, 0, 0],
              [0, 0, 0, ..., 0, 0, 0],
              [0, 0, 0, ..., 0, 0, 0],
              ...,
              [0, 0, 0, ..., 0, 0, 0],
              [0, 0, 0, ..., 0, 0, 0],
              [0, 0, 0, ..., 0, 0, 0]])
```

由显示的元素值均为 0 也可以看出矩阵的稀疏性

(2) vocabulary_ 属性

可以利用字典属性 vocabulary_ 查看词汇对应的 id，查看方式与字典一致。

例 3

```
In [9]:dtm_vectorizer.vocabulary_.get("sure")
Out[9]:712
```

(3) get_feature_names 方法

利用 get_feature_names 方法可以获得所有特征项，即文档-词频矩阵列对应的所有词汇

例 4

```
In [10]:dtm_vectorizer.get_feature_names()
Out[10]:[u'0fhmt',
         u'0jf',
         ...,
         u'zv',
         u'zviq']
```

(4) inverse_transform 方法

inverse_transform 方法用于返回文档-词频矩阵中每个文档的非零的词项，调用方式为：实例.inverse_transform(DTM)，DTM 即需要统计文档非零词项的文档-词频矩阵

例 5

```
In [11]:dtm_vectorizer.inverse_transform(dtm)
Out[11]:[array([u'well', u'not', u'sure', ...,u'power', u'unfortunate'], dtype='|<U25'),
...
array([u'about', u'the', u'it', ..., u'thanks',u'insight'], dtype='|<U25')]
```

词袋模型的一个缺点就是其忽略了语句中词汇的顺序及语法关系，如以下两句话：“This is a text mining book.” 和 “Is this a text mining book?”，两句话由相同的词汇组成，文档-词频矩阵中对应的向量表示也应该是一样的，但是表达的语意不同，为了避免丢失这部分信息，我们可以同时采用 1-grams（单个词汇）和 2-grams（两个词汇组成的词组）的分词方法来构建文档-词频矩阵。

例 6

```
In [12]:corpus = ['This is a text mining book.',
                 'Is this a text mining book?',
                 'Text mining with python.']
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
X
Out[12]:<3x7 sparse matrix of type '<type 'numpy.int64'>'  
with 14 stored elements in Compressed Sparse Row format>
In [13]:X.toarray()
Out[13]:array([[1, 1, 1, 0, 1, 1, 0],
               [1, 1, 1, 0, 1, 1, 0],
               [0, 0, 1, 1, 1, 0, 1]])
```

从以上文档-词频矩阵结果可以看出，前两行向量完全相同，下面同时采用 1-grams 和 2-grams 构建文档-词频矩阵：

```
In [14]:bigram_vectorizer = CountVectorizer(ngram_range=(1, 2))
X_2 = bigram_vectorizer.fit_transform(corpus).toarray()
X_2
Out[14]:array([[1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0],
               [1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1]])
In [15]:bigram_vectorizer.get_feature_names()
Out[15]:[u'book',u'is',u'is text',u'is this',u'mining',u'mining book',u'mining with',u'python',u'text',u'text mining',u'this',u'this is',u'this text',u'with',u'with python']
```

从以上文档-词频矩阵结果可以看出，引入 2-grams 后，前两行向量得以区分开来，但是向量的维度也有所增加。

5.1.3.2 sklearn.feature_extraction.text.HashingVectorizer

利用 CountVectorizer 类构建文档词频矩阵时，需要调用两次文档集合，一次用于创建词典，一次用于创建每个文档对应的词频向量，两次调用会导致内存消耗较大。HashingVectorizer 类通过哈希（hashing）技巧，不创建字典，有效的缓解了这一问题。HashingVectorizer 类实例化方式为：实例=HashingVectorizer(input=u'content', encoding=u'utf-8', decode_error=u'strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop_words=None, token_pattern=u'(?u)\\b\\w\\w+\\b', ngram_range=(1, 1), analyzer=u'word', n_features=1048576, binary=False, norm=u'l2', non_negative=False, dtype=)

部分参数说明：

n_features：用于设置输出矩阵的列数，数值过小可能会引起哈希冲突，数值过大导致维度过高

norm：指定标准化矩阵的方式，有以下三种取值

- (1) l1:利用 l1 范数进行标准化
- (2) l2:利用 l2 范数进行标准化
- (3) None : 不进行标准化处理

non_negative：输出矩阵中是否只包括非负值，取值为 True 时，矩阵元素可以理解为频率，取值为 False 时，输出结果期望值为零

其余参数说明可以参考 5.1.3.1 CountVectorizer 类实例化参数。

对 HashingVectorizer 类调用 fit_transform 方法即可得到哈希文档词频矩阵

例 7

```
In [16]:from sklearn.feature_extraction.text import HashingVectorizer
        hashing_vectorizer = HashingVectorizer(n_features=500)
        hashing_dtm=hashing_vectorizer.fit_transform(data_samples)
        hashing_dtm

Out[16]:<10x500 sparse matrix of type '<type 'numpy.float64'>'>
          with 955 stored elements in Compressed Sparse Row format>

In [17]:hashing_dtm.toarray()

Out[17]:array([[0.          , 0.          , 0.          , ..., 0.          , 0.          , 0.          ],
   [0.          , 0.          , 0.          , ..., 0.          , 0.          , 0.          ],
   [0.          , 0.          , 0.          , ..., 0.17556172, 0.          , 0.05852057],
   ...,
   [0.          , 0.          , 0.          , ..., 0.          , 0.          , 0.          ],
   [0.          , 0.          , 0.          , ..., 0.          , 0.          , 0.          ],
   [0.          , 0.          , 0.          , ..., 0.          , 0.05652334, -0.05652334]])
```

5.1.3.3 sklearn.feature_extraction.DictVectorizer 模块

有时对文本数据进行分词和词频统计汇总后，得到的结果会直接以键、值的形式存储为字典格式，例如文档“text mining text analysis”，可以存储为 `{'text': 2, 'mining': 1, 'analysis': 1}`，如何将此种类型的文本分析结果转换为 DTM 呢？

DictVectorizer 模块下定义的 DictVectorizer 类可以将字典形式的特征表示转换为 Numpy 数组形式，对于分类变量采用“one-hot coding”表示。对于“one-hot coding”可以这样理解：如果分类变量有 A、B、C 三个取值，利用“one-hot coding”可以依次表示为(1,0,0)、(0,1,0)、(0,0,1)，相当于统计学中的虚拟变量。DictVectorizer 类实例化方式为：实例
`=DictVectorizer(dtype=, separator='=', sparse=True, sort=True)`

参数说明：

`dtype`：可选变量，特征值数据类型，通过该参数传入 Numpy array 或 `scipy.sparse` 矩阵构造器

`separator`：可选变量，构建新的“one-hot coding”特征值时使用的分隔符

`sparse`：可选变量，是否生成 `scipy.sparse` 矩阵

`sort`：可选变量，是否输出 `featurenames` 和 `vocabulary` 两个属性，属性 `feature_names` 是特征名称列表，对应文档中的词汇项，`vocabulary_` 是特征名称与相应 id 的字典

(1) `fit_transform` 方法

对 DictVectorizer 类调用 `fit_transform` 方法可以实现特征表示的数组形式转换，调用方式为：
 实例.`fit_transform(X)`，X 即为需要转换的字典类型的特征表示。比如在下例中，
`measurements` 是以字典存储的特征表示，其中“city”属于分类变量，“temperature”属于数值型变量，现要将其转换为数组形式。

例 8

```
In [18]:measurements = [{'city': 'Dubai', 'temperature': 33.},
                       {'city': 'London', 'temperature': 12.},
                       {'city': 'San Francisco', 'temperature': 18.}]
from sklearn.feature_extraction import DictVectorizer
vec = DictVectorizer(sparse=False)
measurements_vec=vec.fit_transform(measurements)
measurements_vec
Out[18]:array([[ 1.,  0.,  0.,  33.],
               [ 0.,  1.,  0.,  12.],
               [ 0.,  0.,  1.,  18.]])
In [19]:vec.vocabulary_
Out[19]:{'city=Dubai': 0, 'city=London': 1, 'city=San Francisco': 2, 'temperature': 3}
```

例 9

假设有 A、B 两个文档，文档 A 用词袋模型表示为 `{'text': 1, 'mining': 2}`，即文档中“text”出现了一次，“mining”出现了两次，类似的，文档 B 表示为 `{'Python': 3, 'text': 1}`，现需要得到两个文档的文档-词项矩阵：

```
In [20]:D = [{'text': 1, 'mining': 2}, {'Python': 3, 'text': 1}]
X = vec.fit_transform(D)
X
Out[20]:array([[ 0.,  2.],
               [ 3.,  0.]])
```

得到的结果即为 DTM，查看每个维度的词汇项：

```
In [21]:vec.vocabulary_
Out[21]:{'Python': 0, 'mining': 1, 'text': 2}
```

(2) inverse_transform 方法

`inverse_transform` 方法是 `fit_transform` 的逆方法，调用方式为：实例`.inverse_transform(Y)`，其中 Y 是需要转换为字典类型特征表示的数组

例 10

```
In [22]:vec.inverse_transform(X)
Out[22]:[{'mining': 2.0, 'text': 1.0}, {'Python': 3.0, 'text': 1.0}]
```

5.1.4 利用 gensim 库构建文档-词频矩阵

gensim 库 corpora 包的 `dictionary` 模块提供了文本数据结构化处理的一系列工具，其中 `Dictionary` 是 `dictionary` 模块下定义的类，可以实现词汇和词汇 id 之间的映射，即词典，该类的方法 `doc2bow` 可以将文本词汇集合转换为词袋模型表示形式，以列表形式返回，列表元素为 (词汇 id, 词频) 的元组。要将原始文档转换为词频矩阵，首先要将各文档分词，从字符串转化为单词列表，再统计各文档单词，生成词典(Dictionary)，最后利用词典方法 `doc2bow` 将文档转化成词频表示的向量。

5.1.4.1 构建词典

在构建词典之前，首先需要对文档进行分词、停用词过滤、词干化等处理得到各个文档的词汇集合，在此基础上构建词典。本部分以例 6 中的文本集合 `corpus` 为例说明词典的构建步骤。

```
In [1]:import gensim
from gensim import corpora
corpus = ['This is a text mining book',
          'Is this a text mining book',
          'Text mining with python']

# 此处只进行了简单的分词处理
texts = [[word for word in document.lower().split()] for document in corpus]

# 生成词典
dictionary = corpora.Dictionary(texts)
```

(1) 词典查看方法

生成词典后，利用 `keys` 方法可以查看所有词汇 id，利用 `values` 方法可以查看所有词汇

```
In [2]:dictionary.keys()
Out[2]:[0, 1, 2, 3, 4, 6, 5, 7]
In [3]:dictionary.values()
Out[3]:[u'a', u'mining', u'this', u'text', u'is', u'python', u'book', u'with']
In [4]:print dictionary.items()
Out[4]:[(0, u'a'), (1, u'mining'), (2, u'this'), (3, u'text'), (4, u'is'), (6, u'python'), (5, u'book'), (7, u'with')]
```

(2) 词典过滤方法

对于构建好的词典，可以对其中不满足某些条件的词汇过滤掉，常用的方法有以下几种：

(2.1) filter_tokens

`filter_tokens` 方法可以用于删除或保留词典中指定 id 的词汇，并重新分配词汇 id，调用方式为：`Dictionary.filter_tokens(bad_ids=None, good_ids=None)`

参数说明：

`bad_ids`：需要从词典中删除的词汇 id 的集合

`good_ids`：需要保留的词汇 id 集合，同时删除其他词汇

```
In [5]:dictionary.filter_tokens(bad_ids=[0,4])
      print dictionary.items()
Out[5]:[(0, u'mining'), (1, u'this'), (2, u'text'), (3, u'python'), (4, u'book'), (5,
u'with')]
```

(2.2) filter_n_most_frequent

`filter_n_most_frequent` 方法用于删除文档中出现次数最多的几个词汇，调用方式为：

`Dictionary.filter_n_most_frequent(remove_n)`，其中参数 `remove_n` 即为需要删除的出现次数最多的词汇的个数。

```
In [6]:dictionary.filter_n_most_frequent(1)
      print dictionary.items()
Out[6]:[(0, u'this'), (1, u'text'), (2, u'book'), (3, u'with'), (4, u'python')]
```

(2.3) filter_extremes

在 `filter_extremes` 方法中可以定义更加复杂的过滤方式，其调用方式为：

`Dictionary.filter_extremes(no_below=5, no_above=0.5, keep_n=100000)`。

参数说明：

`no_below`：若包含某词汇的文档少于 `no_below` 个，该词汇将被删除

`no_above`：若包含某词汇的文档在所有文档中占比大于 `no_above`，该词汇将被删除

`keep_n`：经过参数 `no_below` 和 `no_above` 的过滤后，保留频率最高的前 `keep_n` 个词汇，取值为 `None` 时，表示保留全部词汇

(3) 词典扩充方法

(3.1) merge_with

`merge_with` 方法用于字典合并，调用方式为：`Dictionary.merge_with(new_dictionary)`，其中参数 `new_dictionary` 即为其待合并新词典，两个词典中共同出现的词汇将会映射到相同的 `id`，新词典中新出现的词汇将会映射到新 `id`，

```
In [7]:corpus1 = ['This is a text mining book',
                 'Is this a text mining book',
                 'Text mining with sklearn']
texts1 = [[word for word in document.lower().split()] for document in corpus1]
dict1 = corpora.Dictionary(texts1)
corpus2 = ['That is a text mining book',
           'Is that a text mining book',
           'Text mining with gensim']
texts2 = [[word for word in document.lower().split()] for document in corpus2]
dict2 = corpora.Dictionary(texts2)
dict2_to_dict1 = dict1.merge_with(dict2)
dict2_to_dict1
Out[7]:<gensim.models.VocabTransform at 0x7f5bab786e50>
```

(3.2) add_documents

`add_documents` 方法可以用新的文档内容更新现有的词典，同样，新的文档形式需为词汇列表，调用方式为：`Dictionary.add_documents(documents, prune_at=2000000)`

参数说明：

`documents`：待更新文档，需转化为词汇列表

`prune_at`：用于限制词典大小，保证总词汇数小于等于 `prune_at`，当文档规模极大时，是一种节省内存的好方法，取值为 `None` 时，保留所有词汇。

```
In [8]:documents1 = ['This is a text mining book',
                   'Is this a text mining book']
dict = corpora.Dictionary([[word for word in document.lower().split()] for document in documents1])
dict.values()
Out[8]:[u'a', u'mining', u'this', u'text', u'is', u'book']
In [9]:documents2 = ['Text mining with sklearn',
                   'Text mining with gensim']
dict.add_documents([[word for word in document.lower().split()] for document in documents2])
dict.values()
Out[9]:[u'a', u'mining', u'this', u'text', u'is', u'book', u'with', u'gensim', u'sklearn']
```

5.1.4.2 构建文档-词频矩阵

利用 `doc2bow` 方法可以在构建的词典基础上得到文档词频矩阵，调用方式为：

`Dictionary.doc2bow(document, allow_update=False, return_missing=False)`

参数说明：

`document`：需要结构化处理的文档，需转化为词汇列表的形式

`allow_update`：取值为 `True` 时，允许在处理过程中更新词典，为新词汇映射新的 `id`

`return_missing`：是否返回缺失值

```
In [10]:corpus = ['This is a text mining book',
                 'Is this a text mining book',
                 'Text mining with python']
texts = [[word for word in document.lower().split()] for document in corpus]
dictionary = corpora.Dictionary(texts)
word_count = [dictionary.doc2bow(text) for text in texts]
word_count
Out[10]:[[[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1)],
           [(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1)],
           [(1, 1), (3, 1), (6, 1), (7, 1)]]]
```

以上得到的结果并不是常见的矩阵形式，即行代表文档，列代表词汇，元素为词频值，
`gensim` 库 `matutils` 模块提供了 `corpus2dense` 函数对上述结果进行转换，需要注意的是，该
 函数转换结果的列代表文档，需要进一步进行转置处理。调用方式为：
`corpus2dense(corpus, num_terms, num_docs=None, dtype=)`

参数说明：

`corpus`：需要转换的矩阵

`num_terms`：特征值个数，即词汇数

`num_docs`：文档数

```
In [11]:from gensim.matutils import corpus2dense
corpus_matrix=corpus2dense(word_count, len(dictionary))
corpus_matrix
Out[11]:array([[ 1.,  1.,  0.],
               [ 1.,  1.,  1.],
               [ 1.,  1.,  0.],
               [ 1.,  1.,  1.],
               [ 1.,  1.,  0.],
               [ 1.,  1.,  0.],
               [ 0.,  0.,  1.],
               [ 0.,  0.,  1.]], dtype=float32)
# 转置处理
In [12]:corpus_matrix.T
Out[12]:array([[ 1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.],
               [ 1.,  1.,  1.,  1.,  1.,  1.,  0.,  0.],
               [ 0.,  1.,  0.,  1.,  0.,  0.,  1.,  1.]], dtype=float32)
```


5.2 词频-逆向文档频率

5.2.1 词频-逆向文档频率

在大量的文本数据中，通常会存在一些出现频率极高但是并无实际意义的词汇，如部分停用词，如果直接用这些所谓的高频词对文档进行进一步的分析处理很可能会忽略某些出现频率没有那么高的重要词汇，所以需要在词频的基础上对各个词汇的频数进一步调整，“词频-逆向文档频率”就是一种常用的调整方式。“词频-逆向文档频率”一词源自“Term Frequency–Inverse Document Frequency”，简称 TF-IDF。TF-IDF 算法建立在以下假设之上：对某个文档最有代表性的词汇或者说对区别文档最有意义的词汇应该是那些在某个文档中出现频率高，而在整个文档集合的其他文档中出现频率少的词汇。我们在第一章简单介绍过词频-逆向文档频率的构造原理，直观来看，词频-逆向文档频率就是对文档词频矩阵的核心信息进行提取的结果，其目的在于使词汇能突出所属文档的个性化。常用的计算方式就是在文档词项矩阵 (tf) 的基础上进行权数 (idf) 调整。假设现在有一个包含 1000 个文档的文档集合，其中包括文档：[经济，发展，新常态，研究，……]，该文档总词汇数是 100，“经济”这个词汇出现了 4 次，则“经济”一词的词频 (TF) 为 $4/100 = 0.04$ ，如果在 1000 个文档中有 100 个文档出现过“经济”一词，则逆向文档频率 (IDF) 为 $\log(1000/100) = 1$ ，那么在 TF-IDF 矩阵中，该文档中“经济”一词对应的权数应为 $0.04 * 1 = 0.04$ 。

5.2.2 利用 scikit-learn 库构建词频-逆向文档频率

5.2.2.1 sklearn.feature_extraction.text.TfidfTransformer

`sklearn.feature_extraction.text` 模块下定义的 `TfidfTransformer` 类可以将词频矩阵转换为标准化的 TF 或 TF-IDF 矩阵，用于计算文档 `d` 中词汇 `t` 的 tf-idf 值的公式为：

@todo 补充公式 (1)

类实例化方式为：实例 = `TfidfTransformer(norm=u'l2', use_idf=True, smooth_idf=True, sublinear_tf=False)`

参数说明：

`norm`：指定标准化方式，有以下三种取值

- (1) `l1`:利用 `l1` 范数进行标准化
- (2) `l2`:利用 `l2` 范数进行标准化
- (3) `None` : 不进行标准化处理

`use_idf`：是否使用 `idf` 值对文档词频矩阵进行权数调整而得到 TF-IDF 矩阵，取值为 `False` 时得到文档词频矩阵，即 TF 矩阵。

`smooth_idf`：通过给文档频率加 1 来平滑 idf 权数，相当于假设有一个文档包含了所有的词汇，进而避免 idf 权数计算过程中分母为零的情况

`sublinear_tf`：对 tf 进行线性变换，如 $1+\log(tf)$

`TfidfTransformer` 在默认参数设置下， $\text{idf}(d,t)$ 计算公式为：`@todo 补充公式 (2)`

`TfidfTransformer` 类的 `fit_transform` 方法可以将文档词频矩阵转化为 TF-IDF 矩阵，调用方式为：实例.`fit_transform(X)`，其中 X 即为需要转化的文档词频矩阵。

例 1 利用 `TfidfTransformer` 将文档词频矩阵转化为 TF-IDF 矩阵

```
In [1]:from sklearn.feature_extraction.text import TfidfTransformer
        transformer = TfidfTransformer()
        transformer
Out[1]:TfidfTransformer(norm=u'l2', smooth_idf=True, sublinear_tf=False,
                       use_idf=True)
In [2]:tf = [[4, 0, 0],
            [3, 2, 0],
            [3, 0, 0],
            [3, 0, 2]]
        tfidf = transformer.fit_transform(tf)
        tfidf
Out[2]:<4x3 sparse matrix of type '<type 'numpy.float64'>'!
         with 6 stored elements in Compressed Sparse Row format>
In [3]:tfidf.toarray()
Out[3]:array([[ 1.          ,  0.          ,  0.          ],
               [ 0.61638324,  0.78744632,  0.          ],
               [ 1.          ,  0.          ,  0.          ],
               [ 0.61638324,  0.          ,  0.78744632]]))
```

可以看到，在原 `tf` 矩阵中，第一列，即第一个词汇在不同文档中出现的次数均比较多，实际分析时应减小这类词的系数，而扩大第二个词在第二个文档、第三个词在第四个文档中的系数，通过转换为 TF-IDF 矩阵即可实现这一调整。

5.2.2.2 `sklearn.feature_extraction.text.TfidfVectorizer`

5.1 节介绍的 `CountVectorizer` 类可以将文档集合转换为文档词频矩阵，`TfidfTransformer` 类可以将文档词频矩阵转换为 TF-IDF 矩阵，而 `TfidfVectorizer` 类将这两个过程合并在一起，即可将原始文本数据直接转换为 TF-IDF 矩阵。实例化方式为：实例

```
=TfidfVectorizer(input=u'content', encoding=u'utf-8', decode_error=u'strict',
                 strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None,
                 analyzer=u'word', stop_words=None, token_pattern=u'(?u)\\b\\w\\w+\\b', ngram_range=(1, 1),
                 max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=,
                 norm=u'l2', use_idf=True, smooth_idf=True, sublinear_tf=False)
```

参数说明可以参考 6.1.3.1 CountVectorizer 类实例化参数及 6.2.2.1 TfidfTransformer 类实例化参数。

(1) fit_transform 方法

fit_transform 方法可以直接将文本数据转换为 TF-IDF 矩阵，调用方式为：实例.fit_transform(raw_documents)，参数 raw_documents 即为需要转换的原始文档。

例 2 将 data_samples 转换为 TF-IDF 矩阵

```
In [4]:from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
data_samples_tfidf=vectorizer.fit_transform(data_samples)
data_samples_tfidf

Out[4]:<10x845 sparse matrix of type '<type 'numpy.float64'>'>
        with 1133 stored elements in Compressed Sparse Row format>

In [5]:data_samples_tfidf.toarray()

Out[5]:array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
   [ 0.,  0.,  0., ...,  0.,  0.,  0.],
   [ 0.,  0.,  0., ...,  0.,  0.,  0.],
   ...,
   [ 0.,  0.,  0., ...,  0.,  0.,  0.],
   [ 0.,  0.,  0., ...,  0.,  0.,  0.],
   [ 0.,  0.,  0., ...,  0.,  0.,  0.]])
```

(2) get_feature_names 方法

get_feature_names 方法可以获得所有特征项，即文档-词频矩阵列对应的所有词汇

例 3

```
In [6]:vectorizer.get_feature_names()
Out[6]:[u'0fhmt',
       u'0jf',
       ....
       u'zv',
       u'zviq']
```

5.2.3 利用 gensim 库构建词频-逆向文档频率

gensim 库 models 包中的 tfidfm model 模块下定义了构建词频-逆向文档频率矩阵的类——TfidfModel，同样通过词频（TF）和逆向文档频率（IDF）乘积得到，并对乘积进行标准化处理，对于文档 j 中词汇 i，未经标准化处理的 tf-idf 计算公式为：weight*i,j* =

`wlocal(frequency{i,j}) * wglobal(documentfreq{i}, D)`，其中使用者可以自行设置 `wlocal` 和 `wglobal` 函数，默认 `wlocal` 为恒等式，使用者可以自行定义为平方根、对数等，默认 `wglobal` 为 `log_2(total_docs / doc_freq)`。

该类的实例化方式为：实例 = `TfidfModel(corpus=None, id2word=None, dictionary=None, wlocal=, wglobal=, normalize=True)`

参数说明：

`corpus`：训练文档，取值需为 `doc2bow` 方法返回的类型，即以元组（词汇 `id`，词频）为元素的列表，基于该参数包含的词汇构建矩阵

`dictionary`：取值类型需为 `Dictionary` 类，如果通过该参数指定构建的词典，则直接使用该词典构造逆向文档频率映射，忽略已经设定的 `corpus` 参数

`wlocal`：设置词频（TF）计算函数

`wglobal`：设置逆向文档频率（IDF）计算函数

`normalize`：是否进行标准化处理，取值为 `True` 时，默认标准化为单位长度

实例化后通过调用“`[]`”方法即可得到用 `tf-idf` 值表示的文档向量，调用方式为：实例[`corpus`]，其中参数 `corpus` 即为需要结构化处理的文本，取值需为 `doc2bow` 方法返回的类型

例 4

```
In [7]:import gensim
    from gensim import corpora,models
    documents = ['This is a text mining book',
                  'Is this a text mining book',
                  'Text mining with python']
    texts = [[word for word in document.lower().split()] for document in documents
    ]
    # 创建词典
    dictionary = corpora.Dictionary(texts)
    # 创建文档词频向量
    corpus = [dictionary.doc2bow(text) for text in texts]
    # 计算 tf-idf 值
    tfidf_model = models.TfidfModel(corpus)
    corpus_tfidf = tfidf_model[corpus]
    for i in corpus_tfidf:
        print i
Out[7]:[(0, 0.5), (2, 0.5), (4, 0.5), (5, 0.5)]
[(0, 0.5), (2, 0.5), (4, 0.5), (5, 0.5)]
[(6, 0.7071067811865475), (7, 0.7071067811865475)]
```

利用 corpus2dense 转换矩阵形式

```
In [8]:from gensim.matutils import corpus2dense
    corpus_matrix=corpus2dense(corpus_tfidf, len(dictionary))
    corpus_matrix
Out[8]:array([[ 0.          ,  0.5         ,  0.          ,  0.          ],
               [ 0.          ,  0.          ,  0.          ,  0.          ],
               [ 0.5         ,  0.5         ,  0.          ,  0.          ],
               [ 0.          ,  0.          ,  0.          ,  0.          ],
               [ 0.5         ,  0.5         ,  0.          ,  0.          ],
               [ 0.5         ,  0.5         ,  0.          ,  0.          ],
               [ 0.          ,  0.          ,  0.70710677 ,  0.          ],
               [ 0.          ,  0.          ,  0.70710677 ,  0.70710677]], dtype=float32)
```

转置处理

```
In [9]:corpus_matrix.T
Out[9]:array([[ 0.5 ,  0. ,  0.5 ,  0. ,  0.5 ,  0.5 ,  0.5 ,  0. ,  0. ],
               [ 0.5 ,  0. ,  0.5 ,  0. ,  0.5 ,  0.5 ,  0.5 ,  0. ,  0. ],
               [ 0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0. ,  0.70710677 ,  0.70710677]], dtype
               =float32)
```

5.3 词向量

5.3.1 词向量

不知道读者是否还有印象，作为阅读全书的基本知识准备，我们在 1.2 节曾简单介绍过文本数据分析的相关概念，其中就包括词向量的基本概念，为了使读者对词向量有更加直观和深入的理解，本部分将结合具体的例子对词向量的由来和原理做进一步的说明。

本章的前两节提供了文本数据结构化处理的两种方式，一个是文档-词项矩阵，另一个是词项-逆文档频率矩阵，这两种方式的共同点在于：把文档看成词汇的集合，对于每个词汇赋予一个系数，可以是词汇出现的次数，也可以是每个词汇对应的 tf-idf 值，最终每一个文档都可以表示成由这些系数构成的数值向量。下面要介绍的词向量与以上两种方式有所不同，词向量是将每一个词汇用数值向量进行表示的文本数据结构化方式。

词向量的发展过程经历了从 one-hot representation 到 distributed representation 的过程。
one-hot representation 将词汇表示为数值向量，该向量的长度为文档集合中所有词汇构成的词典的大小，向量的元素只有一个为 1，其他全部为 0，1 的位置对应该词在词典中的位置。

比如，有以下文档集合：

文档一：我/爱/文本/分析

文档二：Python/文本/数据/分析

文档三：文本/数据/分析/指南

该文档集合对应的词典为{我，爱，文本，分析，Python，数据，指南}，词典中的每个词对应的 one-hot representation 为：

我 ——> (1, 0, 0, 0, 0, 0, 0)

爱 ——> (0, 1, 0, 0, 0, 0, 0)

文本 ——> (0, 0, 1, 0, 0, 0, 0)

分析 ——> (0, 0, 0, 1, 0, 0, 0)

Python ——> (0, 0, 0, 0, 1, 0, 0)

数据 ——> (0, 0, 0, 0, 0, 1, 0)

指南 ——> (0, 0, 0, 0, 0, 0, 1)

one-hot representation 是自然语言处理中最为直观和常用的词汇表示方法，但该种表达方式存在两个缺点：一是容易受维数灾难的困扰，二是不能很好地刻画词与词之间的关系。为了克服 one-hot representation 的不足，Hinton 于 1986 年提出了 Distributed Representation 方法。该方法通过训练（常用神经网络的方法）将某种语言中的每一个词映射成一个固定长度的短向量（相对于 one-hot representation 的“长”而言，维度以 50 维和 100 维比较常见），将所有这些向量放在一起形成一个词向量空间，而每一向量则为该空间中的一个点，在这个空间上引入“距离”，可以用最传统的欧氏距离来衡量，也可以用 cos 夹角来衡量，就可以根据词之

间的距离来判断它们之间的（词法、语义上的）相似性，也就是说，如果两个词在语义上接近，那么这两个词的词向量的夹角越小，夹角余弦值越大；另一个是 **Distributed Representation**

词向量具有“可加性”，经过训练得到的词向量可以得到类似这样的效果：“英国-伦敦=法国-巴黎”、“女王-女=国王-男”。所以说 **distributed representation** 在维度和语义两个层面上都弥补了 **one-hot representation** 的不足。用 **Distributed representation** 表示词汇的方式，通常被称为“**Word Representation**”或“**Word Embedding**”，也就是中文俗称的“词向量”。本书提到的所有“词向量”均指用 **Distributed Representation** 表示的词向量。

5.3.2 word2vec

word2vec 是 Google 于 2013 年发布的一个用于获取 Word Embedding 的工具包，它简单、高效，使训练速度大为提升，因此引起了很多人的关注。word2vec 的实现模型有两种：连续词袋模型(Continuous Bag of Words，简称 CBOW)和 Skip-Gram 模型，每个模型的训练方法又分别有两种，hierarchical softmax 与 negative sampling。算法上这两个模型是相似的，都利用人工神经网络作为它们的分类算法，最初每个单词都是一个随机 N 维向量，训练时，利用 CBOW 或者 Skip-Gram 获得每个单词的最优向量，差别在于，CBOW 是根据上下文来预测当前词语的概率，Skip-Gram 刚好相反，根据当前词语来预测上下文的概率。Skip-Gram 模型在处理大规模数据集时结果更为准确。

5.3.3 利用 gensim 库训练词向量

gensim 库 models 包提供的 word2vec 模块下定义的 Word2Vec 类可以快速、方便的实现词向量的训练，其最初的训练算法移植自 Google word2vec C 语言包，并拓展了更多的功能。该类的实例化或者说模型的初始化方式为：实例=Word2Vec(sentences=None, size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001, seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5, cbow_mean=1, hashfxn=, iter=5, null_word=0, trim_rule=None, sorted_vocab=1, batch_words=10000)

常用参数说明：

sentences：参数取值应为可迭代对象，其中的元素为字符串列表，利用该参数对模型进行初始化，若不提供该参数，则不对模型进行初始化处理

size：用于定义词向量维度，默认取值为“100”

alpha：初始学习率，在训练过程中会随参数“**min_alpha**”线性递减

window：用于设置一个文档中目标词汇和预测词汇之间的最大距离

min_count：在训练过程中忽略所有出现频数小于该值的词汇，默认值为“5”

max_vocab_size：利用该参数控制词汇数量，以便在训练过程中对内存进行控制，如果词汇数超出该值，则删除出现频数少的词汇，一般情况下，每 1000 万单词大概需要占用 1GB 内存，默认取值为“None”，即不限制词汇数上限

sample：控制高频词汇随机采样概率的阈值

seed：随机数生成器，初始化词向量

workers：控制训练的并行数，相当于使用多核机器加速训练过程

sg：用于选择训练的模型，默认取值为“0”，即使用 CBOW 模型，取值为“1”时，使用 skip-gram 模型

hs：取值为“1”时，在训练过程中采用 hierarchical softmax 训练模型；默认取值为“0”，即采用 negative sampling 训练模型

negative：用于 negative sampling 的负例数目，默认取值为“5”

cbow_mean：取值为“0”时，模型隐层为输入层的向量之和，取值为“1”时，模型隐层为输入层向量的均值。只有在使用 CBOW 算法时才需要设置。

iter：遍历语料的次数，默认取值为“5”

trim_rule：用于指定词汇修剪规则，即那些特定的词汇需要保留、删除或使用默认规则进行处理（删除出现频数小于 min_count 参数的词汇），取值为“None”时，表示使用默认规则（min_count）

sorted_vocab：默认取值为“1”，表示在给词汇分配索引之前将词汇按照词频降序排列

例 1 以 NLTK 提供的 brown 语料为例训练 word2vec 模型

```
In [1]:import nltk
       import gensim
       from nltk.corpus import brown
       sentences=[[j.lower() for j in i] for i in brown.sents()]
       sentences
Out[1]:[[u'the', u'fulton', u'county', ..., u'took', u'place', u'.'], ...]
In [2]:model = gensim.models.Word2Vec(sentences, size=100, window=5, min_count=5)
In [3]:set(model.vocab.keys()) # 查看模型中所有词汇
Out[3]:{u'raining', u'writings', u'fig.', ...}
In [4]:model['raining'] # 查看训练得到的词向量
Out[4]:array([ 0.02810971,  0.05340325, -0.0142033 ,  0.02279282, -0.01404482,
       -0.00586316, -0.02212046, -0.06923713, -0.01595714,  0.04950027,
       ....,
       -0.0636242 , -0.03947466, -0.02702851, -0.06154357, -0.05934817,
       -0.05101492,  0.01445699, -0.00774932,  0.02016927,  0.00441761], dtype=
float32)
```

5.3.3.1 `doesnt_match` 方法

Word2Vec 类的 `doesnt_match` 方法可以识别给定词汇列表中存在差异的词汇，调用方式为：
实例.`doesnt_match(words)`，参数 `words` 即为需要识别的词汇列表

例 2

```
In [5]:model.doesnt_match(['breakfast', 'cereal', 'dinner', 'lunch'])
Out[5]:'cereal'
```

5.3.3.2 most_similar 方法

`most_similar` 方法用于识别与给定词汇最相似的词汇，调用方式为：实例`.most_similar(positive=[], negative=[], topn=10, restrict_vocab=None)`

参数说明：

`positive`：用于设置正向词汇列表，作为相似性的正向参照

`negative`：用于设置负向词汇列表，作为相似性的负向参照

`topn`：设置输出的词汇数量，当该参数取值被设置为“`False`”时，该方法直接返回所有词汇的相似性（夹角余弦值）构成的向量

`restrict_vocab`：可选参数，用于控制检索词汇的范围，例如当该参数取值为 `10000` 时，只计算词汇列表中前 `10000` 个词汇的相似性，一般用于词汇列表按照词频降序排列的情况

该方法的计算原理是：计算模型中每个词向量与给定词汇词向量简单平均的夹角余弦值，筛选出夹角余弦值最大的几个词汇。

例 3

```
In [6]:model.most_similar(positive=["wonderful", "happy"], negative=["scared"], topn=5)
Out[6]:[(u'fun', 0.8787847757339478),
         (u'yours', 0.8648930788040161),
         (u'collectors', 0.8642285466194153),
         (u'none', 0.8616389036178589),
         (u'expressed', 0.8604250550270081)]
In [7]:model.most_similar(positive=["wonderful", "happy"], negative=["scared"], topn=False)
Out[7]:array([0.09320489, 0.3076005, ..., 0.66536468, 0.62079042], dtype=float32)
```

5.3.3.3 n_similarity 方法

`n_similarity` 方法用于计算两组词汇之间的相似性，即夹角余弦值，调用方式为：实例`.n_similarity(ws1, ws2)`，参数 `ws1` 和 `ws2` 为需要对比的两组词汇列表。

例 4

```
In [8]:model.n_similarity(['old', 'lady'], ['young', 'boy'])
Out[8]:0.93014280423014239
```

5.3.3.4 similar_by_word 方法

similar_by_word 方法用于查找与指定词汇最相近的词，调用方式为：实例.

similar_by_word(word, topn=10, restrict_vocab=None)，其中参数“word”即为指定的词汇，另外两个参数含义同方法“most_similar”

例 5

```
In [9]:model.similar_by_word('nice', topn=5)
Out[9]:[(u'pleasure', 0.9374653100967407),
         (u'happy', 0.9323044419288635),
         (u'funny', 0.9282171726226807),
         (u'wonderful', 0.9254555702209473),
         (u'lady', 0.9230206608772278)]
In [10]:model.similar_by_word('nice', topn=False)
Out[10]:array([ 0.17490953,  0.43710032,  0.26169139, ...,  0.69598532,
               0.7782889,  0.7407155], dtype=float32)
```

5.3.3.5 similarity 方法

similarity 方法用于计算两个词向量之间的夹角余弦值，调用方式为：实例.similarity(w1, w2)，参数 w1、w2 即为需要计算的两个词汇

例 6

```
In [11]:model.similarity('pleasure', 'happy')
Out[11]:0.90909800835209531
In [12]:model.similarity('pleasure', 'pleasure')
Out[12]:1.0
```

5.4 文本数据结构化处理实例

5.4.1 英文文本结构化处理实例

本部分以 20 Newsgroups 数据库（<http://www.qwone.com/~jason/20Newsgroups/>）的新闻数据为例，说明英文文档-词项矩阵、词项-逆向文档频率矩阵以及词向量的构建方式。

使用 sklearn 内建函数 `fetch_20newsgroups` 加载 20 newsgroups 数据集 'rec.sport.baseball' 类新闻数据

```
In [1]:import sklearn
from sklearn.datasets import fetch_20newsgroups
dataset = fetch_20newsgroups(shuffle=True, random_state=1, remove=('headers', 'footers', 'quotes'), categories=['rec.sport.baseball'])
dataset
Out[1]:{'DESCR':None,
'data': [u"Hello All,\n\n..... ,\n\nMike",
u'\n',
...],
'description':'the 20 newsgroups by date dataset',
'filenames':array(['.....'], dtype='|S97'),
'target':array([0, 0, 0, ..., 0, 0, 0]),
'target_names':['rec.sport.baseball']}
In [2]:len(dataset.data) # 查看文本数
Out[2]:597
```

构建 `dataset.data` 文档-词项矩阵

```
In [3]:from sklearn.feature_extraction.text import CountVectorizer
dtm_vectorizer = CountVectorizer(stop_words="english")
dtm = dtm_vectorizer.fit_transform(dataset.data).toarray()
dtm
Out[3]:array([[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0],
...,
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0],
[0, 0, 0, ..., 0, 0, 0]])
```

在文档-词项矩阵 `dtm` 基础上构建 TF-IDF 矩阵

```
In [4]:from sklearn.feature_extraction.text import TfidfTransformer
       transformer = TfidfTransformer()
       tfidf = transformer.fit_transform(dtm).toarray()
       tfidf

Out[4]:array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]])
```

在训练词向量之前先进行分词和停用词过滤等处理

```
In [5]:import nltk
       from nltk.tokenize import sent_tokenize
       from nltk.corpus import stopwords
       english_stopwords = stopwords.words("english")
       english_punctuations = [',', '.', ':', ';', '?', '(', ')', '[', ']', '!', '@',
       '#', '%', '$', '*'] # 自定义英文表单符号列表
       words=[word_tokenize(t) for t in dataset.data]
       words_lower=[[j.lower() for j in i] for i in words] # 小写处理
       words_clear=[]
       for i in words_lower:
           words_filter=[]
           for j in i:
               if j not in english_stopwords: # 过滤停用词
                   if j not in english_punctuations: # 过滤标点符号
                       words_filter.append(j)
           words_clear.append(words_filter)
       words_clear

Out[5]:[[u'hello', u'"d", u'like', ..... , u'use', u'appreciate', u'mike'],
       .....
       [u'someone', u'sabr', u'actually', ..... , u'4', u'5', u'runs']]
```

使用初步处理后的词汇列表 words_clear 训练词向量

```
In [6]:import gensim
       model = gensim.models.Word2Vec(words_clear, size=100, window=5, min_count=5)
```

查看模型中所有词汇

```
In [7]:set(model.vocab.keys())
Out[7]:{u'inning', u'saves', u'foul', ...}
```

查看训练得到的词向量

```
In [8]:model['sorry']
Out[8]:array([-0.14678614, -0.12967731, -0.08349328, -0.12420464, -0.04189511,
              -0.03648274,  0.05199799,  0.05459175,  0.10753562,  0.00180915,
              ....
              -0.03970845,  0.00807121, -0.12356181, -0.20895423, -0.14045434,
              0.08667169, -0.17613558, -0.21308741, -0.048795,  0.03967994]
              , dtype=float32)
```

6.4.2 中文文本结构化处理实例

中文实例采用和 4.1.1.4 节相同的新闻文本作为结构化分析实例，该数据是从数据堂（<http://more.datatang.com/>）网站下载的新浪社会新闻。

首先加载数据

```
In [1]:with open ("new.txt") as f:
    read=f.readlines()
```

提取前 500 个新闻标题，保存在列表 title 中

```
In [2]:title=[]
for i in read[:500]:
    title.append(i.split("|")[1].decode("utf-8")) # 去除"|"后标题索引位置为[1]，并
利用 decode 函数进行解码处理
```

利用 jieba 中文分词直接对新闻标题进行分词处理，并过滤停用词

```
In [3]:import jieba
    with open("stopwords.txt") as f: # 打开停用词典
        read=f.read().decode('utf-8') # 以 utf-8 编码格式解码字符串
        stop_words = read.splitlines()
    texts=[]
    for i in title:
        title_seg=[]
        segs = jieba.cut(i) # 分词处理
        for seg in segs:
            if seg not in stop_words: # 过滤停用词
                title_seg.append(seg)
        texts.append(title_seg)
    texts

Out[3]:[[u'\u7f51\u6c11', u'\u8d28\u7591', ..., u'\u6708', u'\u516c\u793a'], ...]
```

利用 gensim 库构建文档-词项矩阵

```
In [5]:import gensim
    from gensim import corpora
    dictionary = corpora.Dictionary(texts)
In [6]:word_count = [dictionary.doc2bow(text) for text in texts]
    word_count
Out[6]:[[0, 1), (1, 1), (2, 1), ..., (6, 1), (7, 1), (8, 1)], ...]
In [7]:from gensim.matutils import corpus2dense
    dtm_matrix=corpus2dense(word_count, len(dictionary))
    dtm_matrix.T
Out[7]:array([[ 1.,  1.,  1., ...,  0.,  0.,  0.],
               [ 0.,  0.,  0., ...,  0.,  0.,  0.],
               [ 0.,  0.,  0., ...,  0.,  0.,  0.],
               ...,
               [ 0.,  0.,  0., ...,  0.,  0.,  0.],
               [ 0.,  0.,  0., ...,  1.,  1.,  0.],
               [ 0.,  0.,  0., ...,  0.,  0.,  1.]], dtype=float32)
```

利用 gensim 库构建词项-逆文档频率矩阵

```
In [8]:from gensim import models
        tfidf_model = models.TfidfModel(word_count)
        tfidf = tfidf_model[word_count]
        tfidf_matrix=corpus2dense(tfidf, len(dictionary))
        tfidf_matrix.T
Out[8]:array([[0.3563464, 0.18751191, 0.24476767, ..., 0., 0., 0.],
               [0., 0., 0., ..., 0., 0., 0.],
               ...,
               [0., 0., 0., ..., 0.34610009, 0.34610009, 0.],
               ],
               [0., 0., 0., ..., 0., 0., 0.50546187
                ]], dtype=float32)
```

利用 gensim 库训练词向量

```
In [9]:model = gensim.models.Word2Vec(texts, size=100, window=5, min_count=5)
```

查看部分词汇训练结果

```
In [10]:model[u'北京']
Out[10]:array([-3.26092471e-03,  1.06515212e-03,  4.17973427e-03,
               3.81700741e-03,  2.17494601e-03,  1.85929902e-03,
               ....
               -2.32837372e-03, -1.31162710e-03,  4.11018310e-03,
               2.64904671e-03], dtype=float32)

In [11]:model.similarity(u'台风', u'暴雨')
Out[11]:0.13788290850313972
```

5.5 小结

本章从文档-词项矩阵、词频-逆向文档矩阵、词向量三个角度介绍了文本数据结构化处理的三种方式，相关工具主要来自 `sklearn` 和 `gensim` Python 库，现汇总如下：

@todo 补充汇总表

本章的最后，结合中英文实际数据，分别介绍了文本数据结构化处理的整个流程。

5.6 深度阅读材料

5.6.1 信息检索相关文章

(1) R. Baeza-Yates and B. Ribeiro-Neto (2011). Modern Information Retrieval. Addison Wesley, pp. 68-74.

(2) C.D. Manning, P. Raghavan and H. Schütze (2008). Introduction to Information Retrieval. Cambridge University Press, pp. 118-120.

5.6.2 word2vec 相关文章

(1) Distributed Representations of Words and Phrases and their Compositionality 网址：<http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>

(2) Efficient Estimation of Word Representations in Vector Space 网址：<https://arxiv.org/pdf/1301.3781.pdf>

5.6.3 Google word2vec C 语言包网

址：<https://code.google.com/archive/p/word2vec/>

5.7 练习

5.7.1 叙述词袋模型的思想。

5.7.2 对比词项-文档矩阵、词频-逆向文档频率矩阵、词向量三种文本数据结构化方式的原理差异及优缺点。

5.7.3 从 **20 Newsgroups** 数据库下载 '**alt.atheism**' 类新闻数据

(1) 利用 **scikit-learn** 库 **feature_extraction.text** 模块相关类方法构建文档-词项矩阵，尝试不同参数设置，对比输出结果的差异。

(2) 叙述 **scikit-learn** 库词频-逆向文档频率的计算原理，分别利用 **TfidfTransformer** 类和 **TfidfVectorizer** 类为 '**alt.atheism**' 类新闻数据构建 TF-TDF 矩阵。

(3) 训练词向量模型，领用词向量训练结果计算不同词汇之间的相似性。

5.7.4 将不同文档的词频统计结果列表 **[{'a': 1, 'b': 3}, {'b': 2, 'c': 1}]** 转化为常见 **DTM** 形式，即行表示文档、列表示词汇的矩阵形式，并对结果进行逆转换。

5.7.5 载入 **NLTK** 提供的 **brown** 语料

(1) 利用 **gensim** 库相关类方法构建文档词项矩阵。

(2) 叙述 **gensim** 库词频-逆向文档频率的计算原理，利用 **gensim** 库相关类方法构建 TF-IDF 矩阵，尝试不同的参数设置对比结果差异。

第 6 章 文本分类、聚类

分类和聚类都是数据挖掘领域非常重要的任务，在文本数据分析过程中也有重要的应用。文本分类可以预测文本的类别，用于垃圾邮件的过滤、网页分类、个性化新闻提供等；文本聚类对结构内容相近的文本进行归类，用于实现文档集合的自动整理、搜索信息定位、用户兴趣模式识别等。本章内容主要从文本分类和文本聚类两个角度出发，分别介绍了两种技术在文本数据分析过程中的应用流程、常见算法以及 Python 实现等，具体安排如下：前两节重点介绍文本分类和聚类的算法，6.1 节为文本分类算法，除分类思想及一般分类步骤外，主要介绍了朴素贝叶斯、最邻近、支持向量机、决策树、神经网络等常用分类算法；6.2 节为文本聚类算法，包括 K-means、BIRCH、GMM、GAAC 等，已经掌握了算法原理的读者可以忽略这两部分直接阅读后面的章节；6.3 节和 6.4 节重点介绍各种算法的实现方式，涉及到的工具库有 scikit-learn、NLTK、TextBlob、Pattern 等，此外还特别对部分聚类效果评价指标进行了说明；6.5 节将之前的内容加以整合，以实际分析案例的形式展示了文本分类、聚类的一般流程。

6.1 文本分类算法简介

6.1.1 分类思想

分类和聚类最重要的区别在于训练样本是否有类别标注。分类模型的构建基于有类别标注的训练样本，属于有监督学习，即每个训练样本的数据对象已经有对应的类标识，一般形式可以表示为： $(v_1, v_2, \dots, v_n; c)$ ，其中， v_i 表示样本属性或特征， c 表示对应的类别。通过分类学习可以形成一个分类函数或分类模型，也就是常说的分类器，该分类器能把数据项映射到给定类别中的某一个类中，进而预测测试数据的类别。文本分类，就是根据提取到的文本属性或特征，以及人工分标注的文本类别，构建分类器，将文本划分到已有的类别中。

一般的分类方法都可用于文本分类，常用的文本分类算法包括：决策树、神经网络、朴素贝叶斯（naïve bayes）方法、支持向量机（SVM）等等。

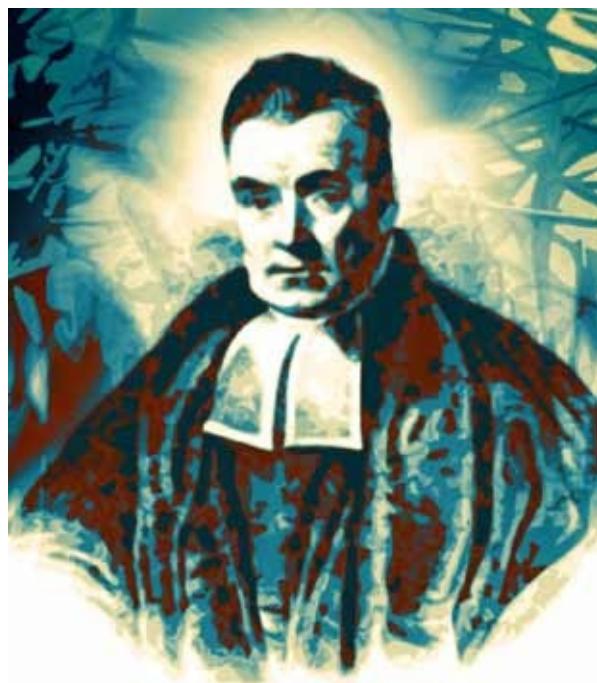
6.1.2 文本分类一般步骤

典型的文本分类过程可以归纳为 **文本类型标注、文本数据结构化处理、分类器构建和分类效果评价三个步骤**。文本类型标注一般指利用人工对一批文档进行准确分类的过程，标注的结果作为训练集；文本数据结构化处理是为了把文本表示成分类器能够处理的形式，如上一章介绍的三种文本数据结构化的方式；分类器的构建涉及分类特征的筛选和构建方法的选择，需要根据实际问题的特点来选择相应的特征和方法，在训练集上构建分类器；将基于训练集构建的分类器应用于测试集数据，得到预测的分类结果，并对该结果进行评价，即所谓的分类效果评价，评价指标一般包括查全率、查准率、F1 值等等。

6.1.3 常用文本分类算法

为了帮助大家更好的利用各类工具实现对于文本数据的分析，接下来我们将向大家介绍几种最为常用的文本分类算法，这些算法虽然基础，但对于初学者来说也是最为实用的。

6.1.3.1 朴素贝叶斯分类（Naive Bayes）



托马斯·贝叶斯(Thomas Bayes,1701—1761)，是18世纪一位广受爱戴的英格兰长老会牧师。为了证明上帝的存在，贝叶斯全身心地投入到了概率与统计的研究之中，尽管他美好的初衷至死都未能实现，但其研究极大地推进了概率论与数理统计的发展，成为了概率统计学原理的奠基。贝叶斯所采用的许多术语被沿用至今，而贝叶斯分类器，正是基于其提出的贝叶斯公式而产生的机器学习分类方法。

贝叶斯公式：设 D_1, D_2, \dots, D_n 为样本空间中的划分，以 $P(D_i)$ ($i=1, 2, \dots, n$) 表示事件 D_i 发生的概率，且有 $P(D_i) > 0$ 。则对于任意事件 x ， $P(x) > 0$ ，有：

$$P(D_j|x) = \frac{P(x|D_j)P(D_j)}{\sum_{i=1}^n P(x|D_i)P(D_i)}$$

在贝叶斯分类大家庭中，朴素贝叶斯分类是最为简单的，但同时也十分的常见与实用，广泛地被应用于一些业界基础的文本分类场景中。我们可以顾名思义，朴素贝叶斯的方法思想真的很朴素：对于一个待分类的样本，我们依次求出在它的对应特征条件下每个分类出现的概率，那个分类的概率最大，我们就认为这个样本属于哪个分类。在文本数据分析中，我们样本的分类特征通常对应于文本数据结构化处理后的文本特征项，运用朴素贝叶斯分类的过程可以简单分为以下几个步骤：

1.准备工作：根据处理后的文本数据训练集，确定分类划分 $D=\{y_1, y_2, \dots, y_n\}$ 与文本特征项 $x=\{a_1, a_2, \dots, a_m\}$ ；

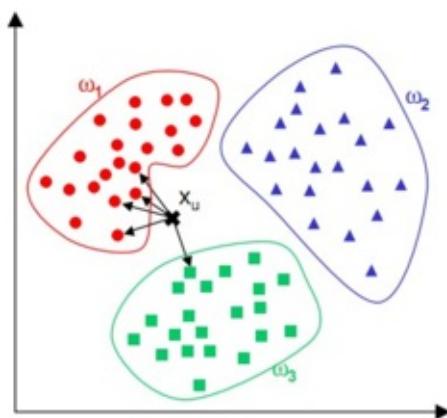
2. 分类器训练：计算训练集中所有特征情况的先验概率 $P(a_1), P(a_1), \dots, P(a_m)$ ，每个分类出现条件下各个文本特征情况的条件概率 $P(a_1|y_1), P(a_2|y_1), \dots, P(a_m|y_n)$ ；

3. 分类器应用：对于一个新的文本样本 $x'=\{a_1', a_2', \dots, a_m'\}$ ，我们根据贝叶斯公式，计算得在其文本特征出现的条件下每个分类出现的条件概率 $P(y_i|x')=P(x'|y_i)P(y_i)/P(x')$ ，这个概率也被称为后验概率，比较每个分类对应的后验概率大小，进而得到新文本样本所属的分类 y' 。

简单的朴素贝叶斯分类中包含了一个最重要的假设，那就是假设样本的特征之间是相互独立的，这个假设是也是整个朴素贝叶斯分类的基础，在我们日常的生活中，文本中字词的前后关联是显而易见的，并非完全的相互独立，因此在进行朴素贝叶斯分类器的应用前，准备工作阶段的文本特征提取是尤为重要的，如果能够提取得文本中相互独立的特征项，那么朴素贝叶斯分类的效果将会有显著的提高。

6.1.3.2 最邻近分类 (Nearest Neighbors)

最邻近分类，或称 K 最邻近分类 (K-Nearest Neighbor Classification)，也是机器学习分类中最为简单的算法之一。1973年，高德纳·克努特(Donald Ervin Knuth)在其经典巨著《计算机程序设计艺术》中首次提及了最邻近分类算法，并将它称为“邮局问题”，即将其比喻为市井居民去寻找离家最近的邮局。值得一提的是，正是凭借这本书在计算机科学理论与技术方面的贡献，高德纳于1974年成功荣获了万众瞩目的图灵奖。



“近朱者赤，近墨者黑”是 K 最邻近分类的指导思想，其中的逻辑便是通过观察你的邻居，来推断你所属的类别。进一步来说，在 K 最邻近分类算法中，我们通常要进行以下几个步骤：

1. 测量距离：这里所谓的“距离”，指的是对样本点之间相似程度的刻画，根据不同的样本特征，常用的距离度量包括欧氏距离 (Euclidean metric)、马氏距离(Mahalanobis distance)、夹角余弦 (Cosine) 等。而对于文本分类来说，在最邻近分类中使用夹角余弦值来计算文本数据样本点相似程度会比直接计算欧式距离来得更为合适；

2. 确定邻近点：在计算了新样本点与所有原始样本点之间的相似度之后，我们要做的便是确定其中几个与新样本点最为相似的样本，作为“邻居”。在最邻近分类中，样本点的个数通常由英文 K 表示，这也是为什么该类算法会被成为“K 最邻近分类”的原因。在一般的最邻近分类算法

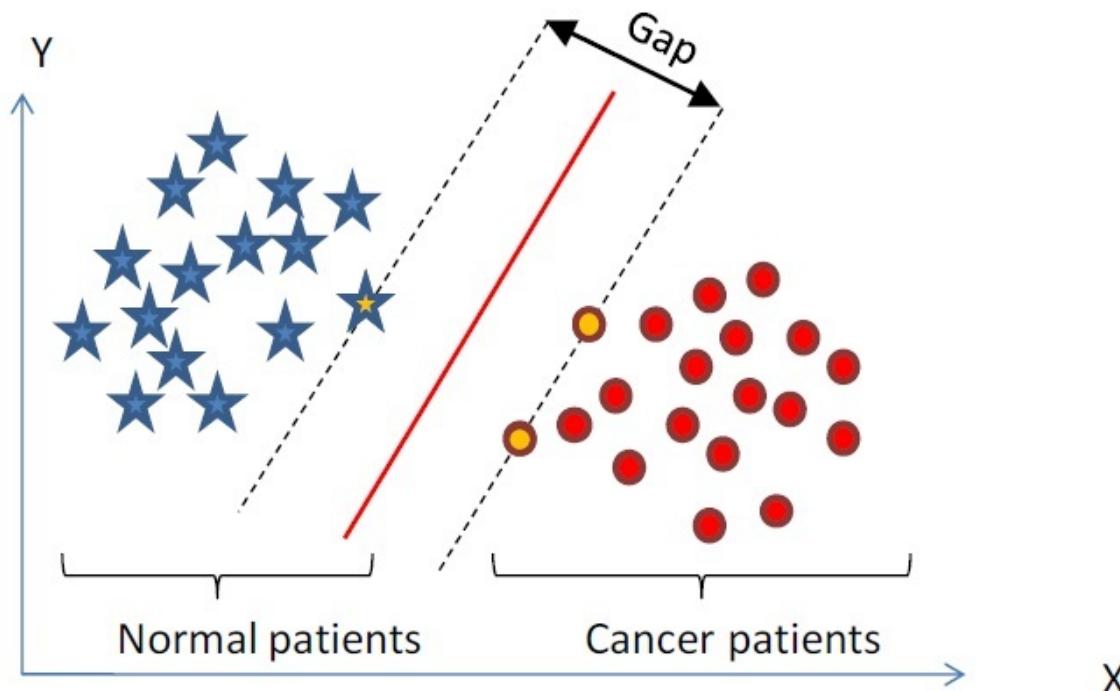
中，邻近点个数 K 都作为一个由人为设定的参数—— K 太小，分类结果易受噪声点影响； K 太大，近邻中又可能包含太多的其它类别的点。我们可以通过反复多次的交叉验证来确定 K 的最优值（因训练集而异），而从经验来看， K 的值一般低于训练样本数的平方根。

3. 判断分类：由于样本的邻近点可能属于不同的类别，我们需要根据一定的规则来确定样本的类别，通常情况下，最邻近分类的分类判别基于加权或等权的投票。

作为一个看似十分“懒惰”的算法，最邻近分类算法的优点在于简单且易于实现，在算法运行过程中，我们无需估计任何的参数，所以也可以说，这是一种“**无需训练**”的分类算法。在文本数据分析中，最邻近分类算法适用于那些分类类别数繁多，且存在一些稀有类别（如不常见的特殊文本种类）时的场景。但同时，它也存在计算量大、内存开销大、效率较低的缺点，不适用于样本量规模过大的文本数据集。

6.1.3.3 支持向量机（Support Vector Machines）

支持向量机算法，简称SVM算法，是当下数学方法和最优化技术在机器学习分类中的最典型、最直接的应用。1995年，由俄罗斯统计学家万普尼克（Vladimir Vapnik）领导的 AT&T Bell 实验室研究小组首先提出了这一种十分具有潜力的分类技术，但由于当时研究本身不够完善，且数学上比较艰涩，一直没有得到学界充分的重视。不过 SVM 算法的寒冬不算太久，随着神经网络等当时新兴的机器学习方法在研究中开始遇到局部极小点问题等重要的困难，SVM 便迎来了迅速发展的春天，于90年代末在生物信息学、文本和手写识别等方面取得了一系列成功的应用。



与我们之前谈及的几种分类器算法不同，支持向量机从一开始便瞄准了不同类别样本点之间可能存在的分类边界，其解决分类问题的逻辑为构造一个用于分割样本类别的超平面，并以规划的思维来解决对样本类别的判断。简单来说，支持向量机有以下三大组成部分：

1. 线性分类器：解决最优化问题——以函数间隔大于1为条件，最大化样本点至分类平面的几何间隔
2. 核函数：低维空间向量集通常难于划分，解决的方法是将它们映射到高维空间。但这个办法带来的困难就是计算复杂度的增加，而核函数正好巧妙地解决了这个问题。也就是说，只要选用适当的核函数，就可以得到高维空间的分类函数。
3. 软间隔：允许一些点游离并在模型中违背限制条件（函数间隔大于1），处理规则化与不可分情况

@todo 详细补充（最优化问题理解+公式；核函数通俗意义+常用例子；软间隔说明）

支持向量机算法所构造的超平面就像一把超越维度的利剑，如果在低维的空间中无法划分黑白，那么它就会上升到更高维度的空间，将这混沌一分为二。这把利剑的优势在于相比于其他分类算法有着更稳定的性能与判断效率，但在文本分类问题中，由于文本特征向量具有稀疏性大、维数高、特征之间有较大的相关性的特点，文本数据中的噪音可能对支持向量机的训练速度、分类准确度产生较大影响。

6.1.3.4 决策树（Decision Tree）

作为一种常见且典型的分类方法，决策树算法产生于上个世纪的70年代，罗斯昆（J. Ross Quinlan）于1975年在悉尼大学提出的 ID3 算法象征着决策树算法的正式成型。简单来说，决策树算法的核心思路类似于我们日常生活中的判断与选择，例如某毕业生寻找工作，在个人逻辑上将可能会思考以下问题：该单位工作地点在哪？如果地点不错，那该单位工资水平如何？如果工资可接受，那么我在该单位有多少晋升的空间？如果还有不错的晋升空间的话，那这就是我想要的工作了。决策树算法分类的内核就是这样的一套策略逻辑，唯一不同的是，决策树算法的模型为特定的树结构（可以是二叉树或非二叉树），并且在所有的判定条件上都进行了量化。



一颗决策树由决策结点、分支和叶子组成。决策树中最上面的结点为根结点，每个分支是一个新的决策结点，或者是树的叶子。每个决策结点代表一个问题或决策，通常对应于待分类对象的属性。每一个叶子结点代表一种可能的分类结果。

沿决策树从上到下遍历的过程中，在每个结点都会遇到一个测试，对每个结点上问题的不同的测试输出导致不同的分支，最后会到达一个叶子结点，这个过程就是利用决策树进行分类的过程，利用若干个变量来判断所属的类别。

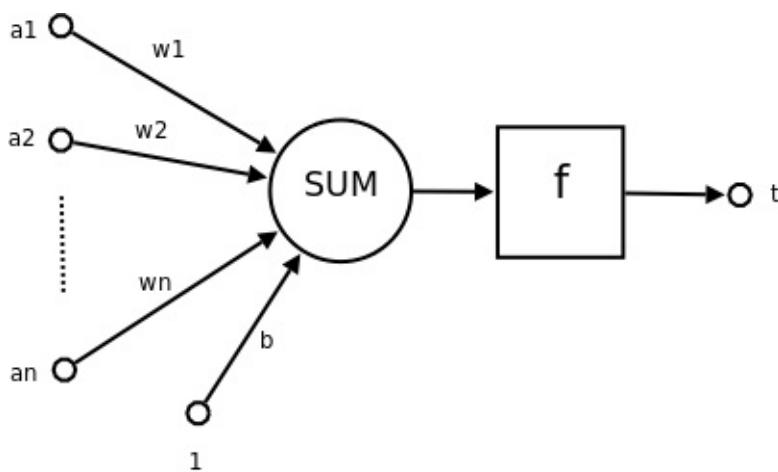
决策树的构建主要基于逼近离散函数值的方法，常用的算法为 ID3、C4.5 与 CART 等。其基本步骤如下：

- 1) 树以代表训练样本的单个结点开始；
 - 2) 如果样本都在同一个类，则该结点成为叶子，并用该类标记，否则在该结点上分裂；
 - 3) 算法选择最有分类能力的属性作为决策树的当前结点的分裂属性，其目标是让各个分裂子集尽可能地“纯”，即尽量让一个分裂子集中待分类项属于同一类别。在这一步骤上，不同算法所使用的划分准则指标不同，例如 ID3 算法评价的是不同特征属性分裂的信息增益（Information Gain），而其后继算法 C4.5 则考虑使用增益率（gain ratio）指标；
 - 4) 对于离散值类型的属性，决策树的构建过程会根据当前决策结点属性取值的不同，将训练样本数据集分为若干子集，每个取值形成一个分枝，有几个取值形成几个分枝；而对于连续值类型的属性，构建过程将会根据自身的划分准则确定合适的分裂点，从而确定划分的子集。针对上一步得到的一个子集，重复进行先前步骤1至3，形成每个划分样本上的决策树。同时，在一般的决策树构建过程中，一旦一个属性出现在一个结点上，就不必在该结点的任何后代考虑它；
 - 5) 递归划分步骤仅当下列条件之一成立时停止：
 - ①给定结点的所有样本属于同一类。
 - ②没有剩余额性可以用来进一步划分样本。在这种情况下，使用多数表决，将给定的结点转换成叶子，并以样本中元组个数最多的类别作为类别标记，同时也可以存放该结点样本的类别分布。
 - ③如果某一分枝tc，没有满足该分支中已有分类的样本，则以样本的多数类创建一个叶子。
- 另外，在实际构造决策树时，我们通常要考虑对决策树进行剪枝，用于处理有数据中的噪声或离群点导致的过拟合问题，常用的剪枝逻辑有以下两种：
- 1) 先剪枝——在构造过程中，当某个节点满足剪枝条件，则直接停止此分支的构造。
 - 2) 后剪枝——先构造完成完整的决策树，再通过某些条件遍历树进行剪枝。

决策树优点其分类精度较高，且能够生成易于理解的分类模式，对于噪声数据，决策树算法的稳健性较好，在文本分类中能够有效减少对训练集样本过拟合的可能性。

6.1.3.5 神经网络（Neural Network）

神经网络算法，也称人工神经网络（Artificial Neural Network，ANN），是如今在人工智能广受追捧的深度学习的算法基础，也是文本数据分析常用的一类分类算法。神经网络的构筑理念是受到生物（人或其他动物）神经网络功能的运作启发而产生的。1943年，沃伦·麦卡洛克（Warren Sturgis McCulloch）和沃尔特·皮茨（Walter Pitts）运用数学技巧对阈值逻辑的算法进行改进，从而创造了最初的神经网络的计算模型，这项创举使得被注明为“神经网络”一词的研究从此分裂为完全不同的两个方向——一种关注大脑中的生物学过程，而另一种则主要关注机器学习与人工智能里的应用。



在人工神经网络中最简单的人工节点被称作神经元（neurons），神经元相互连接从而形成一个类似生物神经网络的网状结构。在上图中， $a_1 \sim a_n$ 为神经元输入向量的各个分量， $w_1 \sim w_n$ 为神经元各个突触的权重值， b 为偏置， f 为神经元的传递函数， t 为神经元输出。在数学上，一个神经元的功能可以理解为求得输入向量与权重向量的内积后，经一个非线性传递函数得到一个标量结果。同时，输入向量至输出标量的映射也被称为激励函数（Activity Rule）。

一般来说，最基础的神经网络模型可以被划分为以下三层：1) 输入层（Input layer）：众多神经元（Neuron）接受大量特征输入，这些输入的特征被称为输入向量。文本数据分析中，这些特征可以等同于文本数据样本的特征项。2) 隐层（Hidden layer）：神经网络的隐层由相互连接的神经元构成，一个神经网络模型中可以拥有多个隐层，每个隐层中的神经元数目不定，但数目越多神经网络的非线性越显著，从而增加神经网络的稳定性。在习惯上，我们通常将单个隐层的神经元个数设置为输入层结点的1.2至1.5倍。3) 输出层（Output layer）：即神经网络模型结果的输出层，输出的结果也被称为输出向量，向量的每一个维度对应具体数据的分类划分可能性。

神经网络模型的训练过程也是对所有网络中每一层所有神经元权重及偏置的参数估计过程。常用的神经网络训练方法有梯度下降法等，该类方法需要我们规定网络中的权重如何随着时间推进而调整的学习规则（Learning Rule）。在规则的设定下，根据逐条或批量样本估计结果的误差损失，模型在训练过程中将逐步调整每一个神经元的权重与偏置参数，最终实现模型拟合。

作为广泛应用于人工感知领域的算法，神经网络在文本数据方面的应用相对来说在学术界较受重视。由于特殊的神经元权重结构，神经网络在对文本数据进行分析的同时体现了对文本知识的提取过程。但由于神经网络算法中的参数繁多，相对来说更容易出现过拟合的情况，因此在使用该类模型时应合理运用交叉验证等分类器评价技术。

6.2 文本聚类算法简介

6.2.1 聚类思想

分类作为一种监督学习方法，要求必须事先明确知道各个类别的信息，并且断言所有待分类项都有一个类别与之对应。但是很多时候上述条件得不到满足，尤其是在处理海量数据的时候，如果通过预处理使得数据满足分类算法的要求，则代价非常大，这时候可以考虑使用聚类算法。

聚类是一种无监督学习。也就是说，聚类是在预先不知道欲划分类的情况下，根据信息相似度原则进行信息聚类的一种方法。聚类的思想是使得属于同类别的对象之间的差别尽可能的小，而不同类别上的对象的差别尽可能的大。与分类规则不同，**进行聚类前并不知道将要划分分成几个组和什么样的组，也不知道根据哪些空间区分规则来定义组。**常见的聚类算法包括：**K-均值聚类算法、K-中心点聚类算法、CLARANS、BIRCH、CLIQUE、DBSCAN等。**

6.2.2 文本聚类一般步骤

6.2.2.1 文本表示（Text Representation）

把文档表示成聚类算法可以处理的形式。所采用的技术请参见文本分类部分。

6.2.2.2 聚类算法选择或设计（Clustering Algorithms）

算法的选择，往往伴随着相似度计算方法的选择。在文本挖掘中，最常用的相似度计算方法是余弦相似度。聚类算法有很多种，但是没有一个通用的算法可以解决所有的聚类问题。因此，需要认真研究要解决的问题的特点，以选择合适的算法。后面会有对各种文本聚类算法的介绍。

6.2.2.3 聚类评估（Clustering Evaluation）

因为没有训练文档集合，所以评测聚类效果是比较困难的。常用的方法是：选择人工已经分好类或者做好标记的文档集合作为测试集合，聚类结束后，将聚类结果与已有的人工分类结果进行比较。常用评测指标也是查全率、查准率及F1值。

6.2.3 常用文本聚类算法

6.2.3.1 K-means

作为基于距离的典型聚类算法，“K-means”一词最早于1967年被加州大学的詹姆斯麦奎恩（James MacQueen）首次使用，而其算法思想则可以追溯到术语提出的十年之前——1957年，斯图尔特劳埃德（Stuart Lloyd）在研究一种脉冲码调制技术时首先研发了 K-means 的标准算法，遗憾的是，其学术成果直到1982年才被贝尔实验室公开出版。在此之间的1965年，福吉（E.W.Forgy）在《Biometrics》发表了本质上相同的方法，因此，K-means 算法有时也被人们称为 Lloyd-Forgy 方法。

已知数据集 $D=\{x_1, x_2, \dots, x_n\}$ ，其中每一个样本都可以由一个 d 维实向量表示，K-means 聚类的目的便是要将数据集中的这 n 个样本划分到 k 个集合之中（ k 小于等于 n ），使得各个集合的组内平方和（Within-Cluster Sum of Squares）最小。该问题也可以由下式表示：

$$\arg \min_{S} \sum_{i=1}^k \sum_{x \in S_i} \|x - \mu_i\|^2$$

其中， S 为样本的聚类， μ_i 则为 S_i 中所有点的均值向量。

在文本聚类中，文本数据集中的每一个样本（我们将其简单称为“文档”）都可以由一个文档特征向量表示，被划分为同一个集合的文档在 K-means 中也被称之为属于同一个簇（cluster），而用于规定簇的中心点则本称为质心（centroid），当一个向量到某个质心的距离小于其至其他所有质心的距离时，这个向量对应的文档将被划分入质心所对应的簇中。为了在文本数据分析中达到文本聚类的目标，K-means 聚类的算法过程通常分为以下步骤：

- 1) 文本数据集中随机选取 K 个文档，作为初始的质心；
- 2) 对剩余的每个文档测量其到每个质心的距离，并把它归到最近的质心对应的簇；
- 3) 通过求中心向量的方式重新计算已经得到的各个簇的质心；
- 4) 迭代 2~3 步直至新的质心与原质心相等或小于指定阈值（或是迭代次数达到外生给定的最大次数），算法结束。

K-means 算法的主要优势在于快速简单、对大数据集有较高的效率，在分析大量文本数据时相比与其他聚类算法更加实用，而其缺点在于容易受 K 值、初始类质心样本选择或初始类划分的影响，在进行文本聚类时，确定最优的 K 值往往需要花费大量的时间资源。

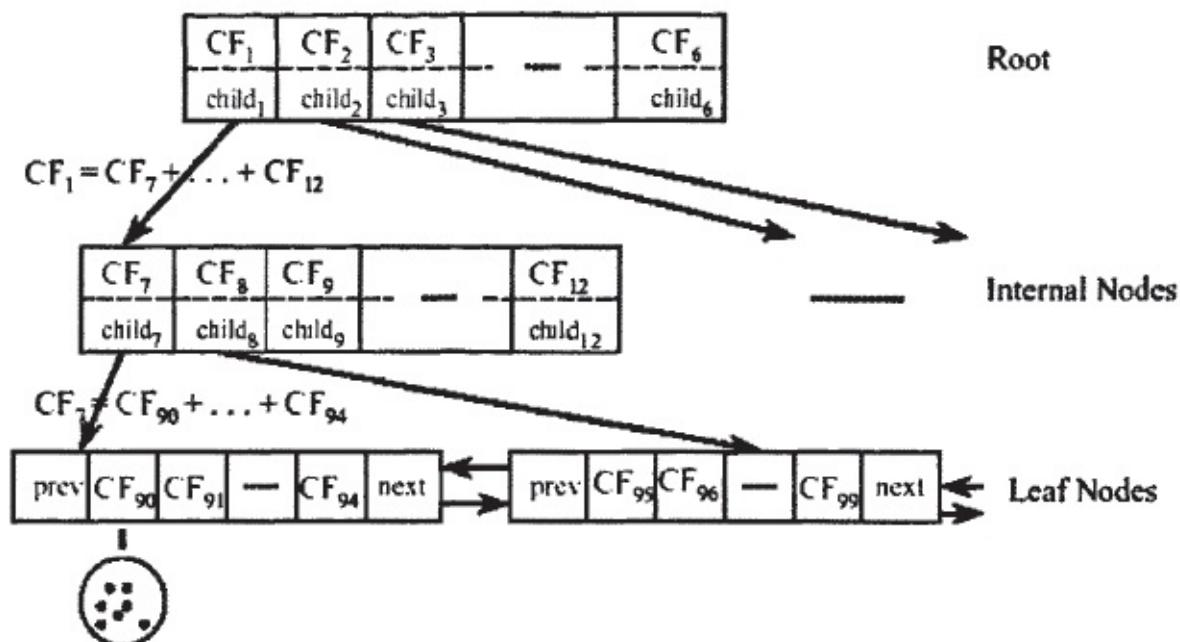
6.2.3.2 BIRCH

BIRCH 算法，全称为利用层次方法的平衡迭代规约和聚类（Balanced Iterative Reducing and Clustering Using Hierarchies），这一在1996年由 Tian Zhang 提出来的聚类算法。虽然名字冗长拗口，但 BIRCH 算法却是广受学界认可的一种节约内存资源的高质量聚类方法。

相比于其他多遍扫描的聚类算法，BIRCH 算法利用了树结构来帮助我们快速的聚类，我们一般将 BIRCH 算法中的这种树结构称之为聚类特征树(Clustering Feature Tree, CF Tree)。聚类特征(Clustering Feature, CF)是聚类特征树的重要概念，它可以被理解为在聚类特征树某一节点上对样本划分的一种状态，其定义可由下式表示：

$$CF = (N, LS, SS)$$

可以看到，我们将聚类特征表示为一个三元组结构，其中 N 为这个聚类特征中拥有的样本点的数量， LS 为这个聚类特征中拥有的样本点各特征维度的和向量， SS 则代表这个聚类特征中拥有的样本点各特征维度的平方和向量。聚类特征 CF 有一个很好的性质，就是满足线性关系，也就是说两个聚类特征可以进行相加，且有 $CF_1 + CF_2 = (N_1+N_2, LS_1+LS_2, SS_1+SS_2)$ 。



作为 BIRCH 算法中的核心部分，聚类特征树，也称 CF 树，是由多个不同层次下 CF 组成的节点构成的树结构，除了聚类特征 CF 以外， CF 树还涉及另外三个重要概念：是每个内部节点的最大 CF 数 B 、每个叶子节点的最大 CF 数 L ，叶节点中每个 CF 的最大样本半径阈值 T ，通过控制这三个参数，在数据集准备充分的情况下，我们就能够构建出一棵聚类特征树。

聚类特征树的构建过程如下：

- 1) 从根节点开始，自上而下选择最近的子节点；

2) 到达叶子节点后，检查最近的元组 $\$CF_i\$$ 能否在大样本半径阈值 $\$T\$$ 范围内吸收此数据点，如果可以，则更新 CF 值；若不能，则考虑是否可以添加一个新的元组，若无法添加则分裂最远的一对元组，作为种子，按最近距离重新分配其它元组；

3) 更新每个非叶节点的 CF 信息，如果分裂节点，在父节点中插入新的元组，检查分裂，直到 $root$ 。

在此基础上，BIRCH 算法的流程分为以下4个阶段：

1) 在内存中构建 CF 树；

2) 对第1阶段中的 CF 树进行瘦身（可选步骤）；

3) 以 CF 叶子节点对应的子簇为基础，实现数据点的聚类；

4) 对第3阶段的聚类结果进行精修（可选步骤）。

@todo 补充算法流程

在文本聚类中，BIRCH 算法的主要优势体现在它对系统内存的节约，在算法执行过程中，聚类特征树仅仅保存了 CF 节点和对应的指针；树状的结构使得 BIRCH 算法更适用于在分析那种实际种类繁多的文本聚类场景之中；由于可以识别噪音点，BIRCH 算法常用于对大量数据集进行初步分类的预处理。但同时，BIRCH 算法由于在执行过程中对每个节点的 CF 个数加以了限制，故可能导致聚类的结果因此与真实的类别分布不同，另外文本数据高维特征的特点也会使 BIRCH 算法的聚类效果大打折扣，在使用 BIRCH 算法进行文本聚类之前，我们应首先进行一定的降维处理。

6.2.3.3 GMM（高斯混合模型聚类）

学界对高斯混合模型（Gaussian mixture model，GMM）的研究动力最早来自于动物学，1893年，动物学家瓦尔特·弗兰·克拉斐尔·韦尔登（Walter Frank Raphael Weldon）在观察前人对雌性滨蟹前额身长比的研究报告时突发奇想，推测这些分布直方图的不对称性可能来自于两个不同的进化分歧，关于混合模型的研究也由此启程。

随着后人在数学计算与统计学方面的努力，今天高斯混合模型已经成为了数据挖掘中广泛应用的聚类模型之一，特别是在图像处理方面广受欢迎。作为一个典型的概率模型类统计学习方法，高斯混合模型的基本思路便是将聚类问题看作为对每个样本的概率密度分布进行估计的问题，且根据中心极限定理，所有的样本都被假设服从于某一参数条件下的高斯分布。

高斯分布的概率密度分布函数：

$$\$phi \left(y \mid \theta \right) = \frac{1}{\sqrt{2\pi} \sigma} e^{-\frac{(y-\mu)^2}{2\sigma^2}}$$

简单来说，高斯混合模型的聚类过程分为以下步骤：

- 1) 假设数据集样本整体都服从某个由 K 个高斯分布加权构成的整体混合分布，该分布被定义为： $P(y|\theta) = \sum_{k=1}^K \alpha_k \phi(y|\theta_k)$ ；
- 2) 运用极大似然估计的方法对高斯分布概率密度函数中的参数 μ_k 、 σ_k 以及权重 α_k 进行估计；
- 3) 得到明确的概率密度函数后，对数据集中的样本分别在每一个高斯分布上投影；
- 4) 选取对应概率最大的分布，作为该样本点所属的类。

在求解极大似然估计时，实际执行中为了解决计算机函数求导的困难，一般会采用 **EM 算法**：第一步，对各个高斯模型的参数进行初始化；第二步，基于当前参数估计每个高斯模型的权重；第三步，基于估计的权重，重新确定高斯模型的参数。重复这二三两个步骤，直到两次参数估计的结果差异足够小，即近似达到极值（EM 算法存在陷入局部最优的风险，因此该估计值不一定就是最优值）。

我们可以发现，**EM 算法**的思想与 **K-means** 聚类中的不断迭代过程似乎有异曲同工之妙，与 **K-means** 相比，高斯混合模型与之的共同点还在于二者都需要指定固定的类别数 K ，并且都运用到了初始化的过程，其中 **K-means** 初始化中心，高斯混合模型则初始化密度函数参数；而二者的区别则在于优化的目标不同，**K-means** 强调最短距离，而高斯混合模型则基于极大似然估计，同时二者在最终的类别划分上也不相同，依据分别为距离与概率。

@todo 补充文本数据分析中 GMM 的优势

劣势：类别数 K 不能设定过多，不适用于类别数繁多的文本数据分析场景。

6.2.3.4 GAAC（凝聚层次聚类）

GAAC (Group-average Agglomerative Clustering)，也称凝聚层次聚类法，是一种基于层次聚类法 (Hierarchical Clustering) 思想的无监督学习方法。GAAC 的核心思路是簇与簇之间的合并，在聚类的过程中，所有的样本点在开始时各成一簇，之后不断重复合并两个距离最近的簇，直至簇的个数达到指定的数量。

GAAC 法聚类的关键点在于明确两个簇之间的相似度度量，常用的度量方法有以下三种：

- 1) 单链(MIN): 将不同两个簇的两个最近的点之间的距离作为两个簇的相似度。
- 2) 全链(MAX): 将不同两个簇的两个最远的点之间的距离作为两个簇的相似度。
- 3) 组平均：取来自两个不同簇的所有点组合，计算点与点之间的距离，以其平均值作为两个簇的相似度。

同时，在 GAAC 法中，为了防止出现过度簇与簇合并的现象，在算法执行时会规定一个退出条件，如当前簇数为初始簇数的 10% 等等，以保证凝聚层次聚类法得到理想的聚类结果。

在文本数据分析中，我们很少会对 GAAC 法进行直接的应用，这是因为 GAAC 在计算簇相似度时往往需要一次性计算大量的样本点间距离，这使得在处理文本数据这一类高维度数据时将耗费大量的资源与时间。因此，当我们面对数据规模较大的文本数据分析时，对 GAAC 法进行使用前一般会结合有效的抽样技术，并将它作为其他聚类方法的铺垫或补充。

6.3 常用文本分类工具

6.3.1 使用 scikit-learn 进行文本分类

scikit-learn 库提供了丰富的分类算法，本部分选取几个常用分类方法进行说明。

6.3.1.1 朴素贝叶斯分类器（Naive Bayes）

scikit-learn 库的 `naive_bayes` 模块提供了朴素贝叶斯分类器的训练算法。朴素贝叶斯方法是基于贝叶斯定理的有监督学习算法，由于特征值之间相互独立的朴素假设，称之为“朴素贝叶斯”。给定类别变量 y 以及相互独立的特征向量 (x_1, \dots, x_n) ，根据贝叶斯公式可以得到以下计算关系：

@todo 补充公式

根据独立性假设以上公式可以简化为

@todo 补充公式

由于分母项恒定，可以使用以下分类规则：

@todo 补充公式

不同朴素贝叶斯分类器的差别主要在于 $P(x_i|y)$ 的假设分布不同，scikit-learn 提供了 Gaussian Naive Bayes、Multinomial Naive Bayes、Bernoulli Naive Bayes 三种假设，后续会一一进行介绍。

尽管假设过于简化，朴素贝叶斯分类器在很多实际分类问题中取得了较好的分析效果，尤其是在文档分类和垃圾邮件过滤方面，且相对于其他复杂算法，朴素贝叶斯分类器训练速度更快，但是该算法分类结果的预测概率比较不准确。

常见有两种模型，分别是多项式模型(multinomial model)和伯努利模型(Bernoulli model)。

(1) 高斯朴素贝叶斯分类器（Gaussian Naive Bayes）

高斯模型即假设 $P(x_i|y)$ 服从高斯分布，并使用极大似然进行参数估计，模型训练主要利用 `naive_bayes` 模块下的 `GaussianNB` 类。

`GaussianNB` 类实例化方式为：`实例=GaussianNB(priors=None)`，参数 `priors` 为类别先验概率，如果在实例化过程中人为设定，则先验概率值不会根据实际数据进行调整。

(1.1) fit 方法

`fit` 方法根据给定的数据训练高斯朴素贝叶斯模型，调用方式为：`实例.fit(X, y, sample_weight=None)`

参数说明：

X：数组型数据，形如 (n_samples, m_features) 的训练集，表示 n 个样本及其对应的 m 个特征项

y：数组型数据，长度为 n，表示 n 个样本的分类情况

sample_weight：可选参数，用于设置每个样本在训练过程中的权重，默认取值为“None”，即不做任何加权处理

(1.2) predict 方法

predict 方法利用训练好的模型对测试集进行分类，调用方式为：实例.predict(X)，其中参数 X 即为测试集数据，数据类型为数组，该方法直接返回测试集的分类结果

(1.3) predict_proba 方法

predict_proba 方法以数组形式返回测试集样本属于各个类别的概率，数组的列表示类别，调用方式为：实例.predict_proba(X)，其中参数 X 即为待预测测试集数组

(1.4) score 方法

score 方法可以用于计算分类结果的平均准确率，调用方式为：实例.score(X, y, sample_weight=None)

参数说明：

X：数组型数据，形如 (n_samples, m_features) 的测试集，表示 n 个样本及其对应的 m 个特征项

y：数组型数据，长度为 n，表示测试集 n 个样本的真实分类情况

sample_weight：样本权重

例 1

```
In [1]:import sklearn
    from sklearn.naive_bayes import GaussianNB
    import numpy as np
    X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
    Y = np.array([1, 1, 1, 2, 2, 2])
    clf = GaussianNB()
    clf.fit(X, Y) # 以 X、Y 为训练集训练分类模型
Out[1]:GaussianNB()
In [2]:clf.predict(np.array([-5, -0.1], [5, 0.1])) # 预测测试集分类
Out[2]:array([1, 2])
In [3]:clf.predict_proba(np.array([-5, -0.1], [5, 0.1])) # 预测测试集样本被分到各个类别的概率

Out[3]:array([[ 1.00000000e+00,   2.81846254e-14],
               [ 2.81846254e-14,   1.00000000e+00]])
In [4]:X_test= np.array([-5, -0.1], [5, 0.1]) # 测试集特征值
y_test=np.array([1, 2]) # 测试集真实分类
clf.score(X_test,y_test) # 计算平均准确率
Out[4]:1.0
```

(1.5) partial_fit 方法

当训练集的规模非常大不适合放在内存中时，可以利用 `partial_fit` 方法进行训练，该方法可以以动态增加的形式使用训练数据，或者说是一个在线学习过程。调用方式为：实例`.partial_fit(X, y, classes=None, sample_weight=None)`

参数说明：

`X`：数组型数据，形如 `(n_samples, m_features)` 的训练集，表示 `n` 个样本及其对应的 `m` 个特征项

`y`：数组型数据，长度为 `n`，表示训练集 `n` 个样本的分类情况

`classes`：可选参数，类别列表，包括 `y` 中可能出现的所有类别，第一次调用时一定要提供该参数

`sample_weight`：样本权重

(1.6) 属性

`classprior` 属性返回各个类别的概率

`classcount` 属性返回各个类别包含的样本数

`theta_` 属性返回各个类别样本属性均值

`sigma_` 属性返回各个类别样本属性方差

例 2

```
In [5]:clf.class_prior_
Out[5]:array([ 0.5,  0.5])

In [6]:clf.class_count_
Out[6]:array([ 3.,  3.])

In [7]:clf.theta_
Out[7]:array([[ -2.          , -1.33333333],
              [ 2.          ,  1.33333333]])

In [8]:clf.sigma_
Out[8]:array([[ 0.66666667,  0.22222223],
              [ 0.66666667,  0.22222223]])
```

(2) 多项式朴素贝叶斯分类器(Multinomial Naive Bayes)

多项式分类器基于多项式分布执行朴素贝叶斯算法，适用于特征值为离散值的训练集，是贝叶斯分类器中适用于文本分类的分类器之一，因为文本分类过程中常常提取词频作为分类特征，虽然多项式分布一般要求特征值为整数，但是在实际分析过程中，如 tf-idf 这样的非整数值也可以取得较好的训练效果。

多项式分类器将每一个类参数化为向量

@todo 补充公式

其中， n 为特征数，对应到文本分类中即为词汇总数，

@todo 补充公式

可以使用以下方式进行参数估计：

@todo 补充公式

在 `sklearn` 库中，多项式模型训练主要利用 `naive_bayes` 模块下的 `MultinomialNB` 类，该类实例化方式为：`实例=MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)`

参数说明：

`alpha`：设置平滑参数，默认缺失值为 1，即 Laplace 平滑

`fit_prior`：是否学习类别先验概率，默认取值为“True”，若取值为“False”，将使用均匀先验概率

`class_prior`：类别概率，默认值为“None”

与 `GaussianNB` 类类似，`MultinomialNB` 类也通过 `fit` 方法进行训练集的训练、通过 `predict` 方法进行测试集类别预测、通过 `predict_proba` 方法返回测试集属于各个类别的概率、通过 `score` 方法计算分类器的平均准确率、利用 `partial_fit` 方法进行大型训练集的训练，具体调用

方式及参数说明可以参考 GaussianNB 类。

例 3

```
In [9]:from sklearn.naive_bayes import MultinomialNB
      X = np.random.randint(5, size=(6, 100))
      y = np.array([1, 2, 3, 4, 5, 6])
      clf = MultinomialNB()
      clf.fit(X, Y) # 以 X、Y 为训练集训练分类模型
Out[9]:MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
In [10]:X_test=np.random.randint(5, size=(2, 100))
         clf.predict(X_test) # 预测测试集分类
Out[10]:array([1, 2])
```

(3) 伯努利朴素贝叶斯分类器(Bernoulli Naive Bayes)

伯努利分类器假设数据服从多元伯努利分布，也就是说，可能训练集包含多个特征项，但是每个特征项都是布尔型变量，即只有两种取值情况，因此要使用该种分类器需要先将样本特征项转换为布尔型，也可以通过参数设置将原始数据直接转换为布尔型。在文本分类中，如果文档词——词项矩阵元素不是词频而是词汇是否出现这样的二值变量，就可以利用伯努利分类器进行文本分类，特别的，当文档长度比较短时，伯努利朴素贝叶斯分类器可以达到较好的分类效果。

naive_bayes 模块下的 BernoulliNB 类可以实现伯努利朴素贝叶斯分类器的训练，实例化方式为：BernoulliNB(alpha=1.0, binarize=0.0, fit_prior=True, class_prior=None)，其中参数 binarize 用于设置二值变量的阈值，如果取值为“None”则默认输入训练集已经为二值型。相关类方法可以参考 GaussianNB 、MultinomialNB 类。

例 4

```
In [11]:from sklearn.naive_bayes import BernoulliNB
      X = np.random.randint(2, size=(5, 20)) # 随机生成二值型训练集
      Y = np.array([1, 2, 3, 4, 4])
      clf = BernoulliNB()
      clf.fit(X, Y)
Out[11]:BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
In [12]:X_test=np.random.randint(2, size=(2, 20))
         clf.predict(X_test) # 预测测试集分类
Out[12]:array([1, 1])
```

6.3.1.2 K-最邻近分类器 (KNN)

最邻近分类器的基本思想可以概括为：对于一个新的待分类的数据点，找到离其较近的已知分类的数据点，这些数据点中大多数属于哪一类，新的数据点就被归为哪一类。可见最邻近分类器是一种基于距离计算的分类器，并不涉及模型的拟合。`sklearn` 库提供了两种不同的最邻近分类器，分别是 **K**-最邻近分类器：`KNeighborsClassifier` 和 半径最邻近分类器：`RadiusNeighborsClassifier`。两个分类器的差别在于临近点的选取规则不同，`KNeighborsClassifier` 选取离待分类点最近的 **K** 个点进行距离计算，`RadiusNeighborsClassifier` 选取以待分类点为中心、以 r 为半径的圆所覆盖的所有点进行距离计算，其中 **K** 和 r 都是使用者自行定义的值。两种方法中 **K** 最邻近分类器最为常用，**K** 的选择很大程度上取决于实际数据情况，一般来讲，**K** 值越大，分类器受噪音的影响越小，分类界限也越模糊。当样本点分布比较不均匀时，可以考虑使用半径最邻近分类器，

`sklearn` 库 `neighbors` 包的 `classification` 模块下定义了 `KNeighborsClassifier` 类进行 **K** 最邻近分类器的训练，`KNeighborsClassifier` 类实例化方式为：实例

```
=KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
```

参数说明：

n_neighbors：用于设置临近点个数 **K** 取值的参数，默认取值为 5

weights：用于设置临近点的权重，有三种取值形式：

- (1) “uniform”：所有临近点都有相同的权重
- (2) “distance”：有时需要给距离待分类点距离近的点较大的权重，可以选取该参数值实现这一需求，即临近点的权重是与待分类点距离的倒数
- (3) 自定义函数：使用者可以根据需求自行定义权重设置函数，函数输入值为点与点之间的距离数组，返回包含权重的同形数组

algorithm：对于临近点的计算，`neighbors` 包提供了 `BallTree`、`KDTree`、`brute-force` 三种算法，“algorithm” 参数就是用于算法的选择，有以下四种取值：

- (1) “ball_tree”：使用 `BallTree` 算法
- (2) “kd_tree”：使用 `KDTree` 算法
- (3) “brute”：使用 `brute-force` 算法
- (4) “auto”：将根据实际数据情况选择最优算法

leaf_size：传给 `BallTree` 或 `KDTree` 算法的叶子节点个数，默认取值为 30

p：用于设置距离计算方法，不同取值及对应的计算方法如下：

- (1) $p = 1$ ：使用曼哈顿距离（`manhattan distance`，l1）
- (2) $p = 2$ ：使用欧式距离（`euclidean distance`，l2），默认取值为 2
- (3) 其他任意 p 值：使用闵可夫斯基聚力（`minkowski distance`，l_p）

metric：算法树的距离测度，默认取值为“`minkowski`”，即闵可夫斯基测度，当参数“ $p=2$ ”时，即为欧式测度，其他可用测度可以查看 `DistanceMetric` 类说明文档（<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html>）

`n_jobs`：搜索临近点的并行任务的数量，默认取值为“1”，当取值为“-1”时，并行任务数取决于CPU内核数

(1) `fit` 方法

`fit` 方法根据实例化时 `algorithm` 的取值，使用训练集数据进行训练生成一个 `kd_tree` 或者 `ball_tree`，如果 `algorithm='brute'`，则什么都不做。调用方式为：实例.`fit(X, y)`，其中 `X` 为训练集样本特征值矩阵，`y` 为对应的分类。

(2) `kneighbors` 方法

`kneighbors` 方法可以找到待分类点的 `K` 个最邻近点，直接返回临近点的标签和距待分类点的距离，调用方式为：实例.`kneighbors(X=None, n_neighbors=None, return_distance=True)`

参数说明：

`X`：待分类样本特征矩阵，若不指定该参数，则默认输入训练样本数据

`n_neighbors`：返回的临近点个数，默認為实例化时设置的 `K` 值

`return_distance`：是否返回距离值，默認為“True”，即返回

(3) 其他方法

`predict`：对测试集进行分类，调用方式为：实例.`predict(X)`，参数 `X` 即为测试集数据

`predict_proba`：用于估计测试集被划分到各个类别的概率，调用方式为：实例.`predict_proba(X)`，参数 `X` 为测试集数据

`score`：计算分类器的平均准确率，调用方式为：实例.`score(X, y, sample_weight=None)`，参数 `X`、`y` 为测试集数据，`sample_weight` 为样本权重

例 5

```
In [13]:from sklearn.neighbors import KNeighborsClassifier
X = np.array([[0,0], [1,0.5], [2,4], [3,3.7]])
y = np.array([0, 0, 1, 1])
clf_knn = KNeighborsClassifier(n_neighbors=2)
clf_knn.fit(X,y) # 使用训练数据进行训练
Out[13]:KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',metric_params=None, n_jobs=1, n_neighbors=2, p=2,weights='uniform')
In [14]:clf_knn.predict([[0.5,0.5],[5,5]]) # 预测测试集类别
Out[14]:array([0, 1])
In [15]:clf_knn.predict_proba([[0,3],[1,5]])
Out[15]:array([[ 0.5,  0.5],
               [ 0. ,  1. ]])
```

6.3.1.3 支持向量机（SVM）

另外一个常用的分类器是支持向量机，支持向量机在数据维度较高或者特征项维度高于样本数的情况下也能得到较好的分类效果，并且在计算决策函数过程中只使用支持向量，简化了计算量，此外使用不同的核函数可以得到不同的决策函数。

sklearn 库 svm 包下定义了支持向量机的算法，其中类 SVC、NuSVC 和 LinearSVC 都可以进行多类别分类，SVC 和 NuSVC 方法相似，但是在参数设置和计算原理上有所不同，而 LinearSVC 是使用线性核函数的分类器，故在参数设置时不用考虑核函数的设置。此外，在进行多类别分类时，SVC 和 NuSVC 默认使用“一对一”（“one-against-one”）的方式对任意两个类构建分类器，若一共有 n 个类别，则需要构建 $n * (n - 1) / 2$ 个分类器，但是也提供了“一对多”的参数选项，为每一类和其余的类集合构建分类器，若一共有 n 个类别，则需要构建 n 个分类器。而 LinearSVC 默认使用“一对多”的方式。关于三种分类器原理上的差异可以阅读 sklearn 官方说明文档 (<http://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation>)。

(1) SVC

SVC 基于 libsvm 算法库，数据拟合的时间复杂度大于样本数量的二次方，使得该方法很难扩展到包含 10000 个样本的数据集。

该类实例化方式为：实例=SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None)

参数说明：

C：误差项的惩罚系数，用以平衡分类间隔和误判样本，惩罚系数越大，被误判的样本点越少，但可能存在噪音样本过拟合的情况，默认取值为 1

kernel：用于设置核函数类型，必须在以下核函数中进行选

择：‘linear’、‘poly’、‘rbf’、‘sigmoid’、‘precomputed’，默认取值为‘rbf’

degree：当核函数为多项式核函数‘poly’时，该参数用于设定多项式的最高次幂

gamma：当核函数为‘poly’、‘rbf’、‘sigmoid’时，该参数用于设定核函数系数，默认取值为“auto”，表示使用 $1/n_features$ 作为系数

coef0：当核函数为‘poly’和‘sigmoid’时，该参数用于设定核函数中的独立项

probability：是否进行概率估计

tol：停止计算的最小差值

cache_size：设置训练过程所需的内存

class_weight：用于设置各个类的权重，各个类别的惩罚系数 C 随权重改变

max_iter：用于设置最大迭代次数，默认取值为 -1，即无限制

`decision_function_shape`：解决多分类问题，需要组合多个二分类器来实现多类别分类器的构造，使用该参数选择组合方式，有以下三种取值：

- (1) ‘ovo’：“一对一”
- (2) ‘ovr’：“一对多”
- (3) None：默认取值，为了后续兼容一般执行‘ovo’

`random_state`：概率估计时数据重新排序依照的伪随机数的生成器种子

关于核函数的形式和部分参数关系，可以阅读官方文档（<http://scikit-learn.org/stable/modules/svm.html#svm-kernels>）。

(1.1) `decision_function` 方法

`decision_function` 方法返回样本点到各个分割超平面之间的距离，即决策函数值，调用方式为：实例.`decision_function(X)`，其中参数 X 即为样本特征矩阵。

(1.2) `fit` 方法

和其他分类器一样，`fit` 方法用训练集数据训练支持向量机，调用方式为：实例.`fit(X, y, sample_weight=None)`，其中 X、y 为训练集数据，`sample_weight` 是调节惩罚系数 C 的权重

(1.3) 其他方法

`predict` 方法：对给定的数据进行分类，调用方式为：实例.`predict(X)`，其中参数 X 即为样本特征矩阵，直接返回各个待分类样本的类别标签。

`predict_proba` 方法：计算样本被划分到各个类别的概率，调用方式为：实例.`predict_proba(X)`

`score` 方法：计算分类器平均精度，调用方式为：实例.`score(X, y, sample_weight=None)`，其中 `sample_weight` 为样本权重

(1.4) 部分属性

`support_`：返回支持向量的标签

`supportvectors_`：返回支持向量

`nsupport`：返回每一类支持向量的个数

`dualcoef`：返回支持向量在决策函数中的系数

`intercept_`：返回决策函数中的常数项

例 6

```
In [16]:from sklearn.svm import SVC
        X = np.array([[-1, -1, -1], [-2, -1, -2], [1, 1, 1], [2, 1, 2]])
        y = np.array([1, 1, 2, 2])
        clf_svc = SVC()
        clf_svc.fit(X, y)
Out[16]:SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovo', degree=3, gamma='auto', kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)
In [17]:clf_svc.support_vectors_
Out[17]:array([[-1., -1., -1.],
               [-2., -1., -2.],
               [1., 1., 1.],
               [2., 1., 2.]])
In [18]:clf_svc.decision_function(X)
Out[18]:array([-0.99974151, -0.99977318, 0.99975735, 0.99975734])
In [19]:X_test = np.array([[-3, 0, -1], [5, 4, 3]])
        clf_svc.predict(X_test)
Out[19]:array([1, 2])
```

(2) NuSVC

NuSVC 与 SVC 相似，同样也是基于 libsvm 算法库，但是在分类器训练过程中 NuSVC 通过参数设置控制支持向量数量。该类实例化方式为：实例=NuSVC(nu=0.5, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovo', random_state=None)，其中参数“nu”为误判样本占比上界、支持向量占比下界，其他参数含义均与 SVC 相同，相关属性及方法也可参照 SVC。

(3) LinearSVC

LinearSVC 与参数 kernel='linear' 的 SVC 相似，但是 LinearSVC 基于 liblinear 算法库，而不是 libsvm，所以惩罚函数、损失函数的选择都更加灵活，并且对于大样本数据具有更好的伸缩性。该类实例化方式为：实例=LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000)

部分参数说明：

loss：用于设置损失函数，有以下两种取值：

- (1) ‘hinge’：标准 SVM 损失函数
- (2) ‘squared_hinge’：标准 SVM 损失函数的平方，为默认值

penalty：用于设置惩罚项中的范数类型，有‘l1’和‘l2’两种取值，默认取值为 l2 范数

6.3.1.4 决策树（Decision tree）

决策树是一个非参数的有监督学习过程。决策树在学习之前不需要过多的数据预处理，并且可以处理数值型变量和分类型变量，可视化的特性使其便于理解和解释说明，但是比较容易出现过拟合的情况，易受样本数据变动的影响。

sklearn 库中 tree 包中定义的 DecisionTreeClassifier 类可以实现多类别的决策树分类，默认使用的是 CART 算法。该类的实例化方式为：实例.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_split=1e-07, class_weight=None, presort=False)

参数说明：

criterion：设置节点特征信息量度量方法，“gini”表示基尼指数，“entropy”表示信息增益，默认取值为“gini”

splitter: 设置节点特征选择规则，“best”表示选择最优的特征，“random”表示随机的在部分特征中找局部最优的特征。

max_features：寻找节点特征时考量的特征个数，有以下几种取值情况：

- (1) 整数：在每一个节点都考量 max_features 个特征
- (2) 浮点数：max_features 值为考量特征在全部特征数 n_features 中的占比，在每一个节点考量(max_features * n_features)个特征
- (3) “auto”：max_features=sqrt(n_features)
- (4) “sqrt”：max_features=sqrt(n_features)
- (5) “log2”：max_features=log2(n_features)
- (6) None : max_features=n_features

max_depth：树的最大深度，默认取值为 "None"，树将扩展到直至所有的叶节点为单一类别或者包含的样本数小于参数 “min_samples_split” 界定的样本数为止。

min_samples_split：样本数阈值，停止划分一个中间节点的最小样本数，有以下两种取值方式：

- (1) 整数：最小样本数为 min_samples_split
- (2) 浮点数：min_samples_split 为考量的样本数在全部样本数 n_samples 中的占比，故每个节点最小样本数为 (min_samples_split * n_samples)

min_samples_leaf：确认一个叶子节点的最小样本数，若小于该值，则被剪枝，同样有整数和浮点数两种取值方式：

- (1) 整数：最小样本数为 min_samples_leaf
- (2) 浮点数：min_samples_leaf 为考量的样本数在全部样本数 n_samples 中的占比，故每个节点最小样本数为 (min_samples_leaf * n_samples)

min_weight_fraction_leaf：确认一个叶子节点的所有样本权重之和的最小值，若小于该值，则被剪枝，默认取值为“0”，即不考虑权重问题

max_leaf_nodes：允许拟合的最大叶子节点个数，避免过拟合问题，默认取值为“None”，即不限制叶子节点个数

class_weight：类别权重，若不指定该参数，所有的类别权重均为 1，有三种取值方式：

- (1) 利用形如 {class_label: weight} 的字典设置类别权重
- (2) “balanced”：根据各个类别样本数自动进行权重调整
- (3) “None”

min_impurity_split：控制决策树增长的阈值，如果一个节点的信息量（基尼系数、信息增益）大于该值，则继续划分，否则为叶子节点。

presort：是否对样本数据进行排序以加快节点特征筛选速度，对于数据量较大的样本进行排序可能会降低训练速度，但是当数据量较小或者预设深度较小时，排序可以加快训练速度。

(1) `apply` 方法

`apply` 方法返回每个样本预测叶子节点的标签，调用方式为：实例`.apply(X, check_input=True)`，参数 `X` 即为样本特征矩阵，参数 `check_input` 用于设置可以忽略的输入检查，一般不清楚具体需求时，不对该参数进行设置。

(2) `fit` 方法

`fit` 方法用于训练决策树分类器，调用方式为：`fit(X, y, sample_weight=None, check_input=True, X_idx_sorted=None)`，在对同一个样本集合进行多次训练时，需要设置参数 `X_idx_sorted` 对多个决策树进行标记，一般情况不需要进行设置。

(3) 其他方法

`predict`：预测测试集类别，调用方式为：实例`.predict(X, check_input=True)`

`predict_proba`：预测测试集样本被划分到各个类别的概率，调用方式为：实例`.predict_proba(X, check_input=True)`

`score`：利用测试集计算决策树分类器的平均准确率，调用方式为：实例`.score(X, y, sample_weight=None)`

例 7

```
In [20]:from sklearn import tree
        X = [[0, 0, -1], [1, 0, 1]]
        Y = [0, 1]
        clf_DecisionTre = tree.DecisionTreeClassifier()
        clf_DecisionTre.fit(X, Y)
Out[20]:DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,max_
_features=None, max_leaf_nodes=None, min_samples_leaf=1,min_samples_split=2, min_weigh_
t_fraction_leaf=0.0,presort=False, random_state=None, splitter='best')
In [21]:clf_DecisionTre.predict([[2, 2, 0]])
Out[21]:array([1])
```

6.3.1.5 神经网络（Neural network）

0.18 版本的 scikit-learn 库 neural_network 包提供了两种神经网络分类器，分别是 BernoulliRBM 和 MLPClassifier，而 0.17 版本只支持 BernoulliRBM 算法，本部分主要介绍多层感知器 MLP (multi-layer perceptron)。

scikit-learn 库定义了 MLPClassifier 类利用 BP 算法训练多层神经网络，仅支持交叉熵 (Cross-Entropy) 损失函数和分类概率估计，类实例化方式为：实例
`=MLPClassifier(hidden_layer_sizes=(100,), activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08)`

参数说明：

hidden_layer_sizes：设置各个隐层的单元数，取值类型为元组，元组长度为神经网络总层数减 2，元组中第 i 个元素就代表第 i 个隐层的单元数，默认取值为“(100,)”

activation：设置隐层的激活函数 $f(x)$ ，提供了以下四种取值：

- (1) ‘identity’：无激活函数，只进行线性转换，即 $f(x) = x$
- (2) ‘logistic’：logistic sigmoid 函数，即 $f(x) = 1 / (1 + \exp(-x))$
- (3) ‘tanh’：双曲函数，即 $f(x) = \tanh(x)$
- (4) ‘relu’：修正线性函数，即 $f(x) = \max(0, x)$

默认取值为 ‘relu’

solver：设置寻找最优权重的算法，支持以下三种算法：

- (1) ‘lbfgs’：属于拟牛顿算法的一种最优化算法，对于较小数据集训练效果较好
- (2) ‘sgd’：随机梯度下降法
- (3) ‘adam’：Kingma 等人提出的基于梯度搜索的最优化算法，对于较大的数据集该算法不论是训练时间还是交叉验证结果都有较好的表现，
默认取值为 ‘adam’

`alpha`：设置正则化项的惩罚系数。默认取值为 0.0001

`learning_rate_init`：初始学习率，默认取值为 0.001

`power_t`：当参数 `learning_rate` 取值为 ‘`invscaling`’ 时，用于设置缩放指数

`tol`：寻找最优权重过程中的最优化阈值，如果迭代过程中损失函数或者交叉验证得分变动未满足该参数，则对学习率进行调整

`learning_rate`：设置学习率变化趋势，有以下三种取值：

(1) ‘`constant`’：始终保持初始学习率 `learning_rate_init`

(2) ‘`invscaling`’：随着迭代次数 `t` 的增加在初始学习率的基础上逐渐减小学习率，减小规则为 `learning_rate_init / pow(t, power_t)`，其中 `power_t` 参数需另外设置

(3) ‘`adaptive`’：只要损失函数取值随着迭代次数的增加而减小就保持初始学习率不变，如果连续两个迭代的损失函数减小没有超过参数 `tol`，或者交叉验证得分的增加没有超过参数 `tol`，学习率就缩小为原来的 1/5

默认取值为 ‘`constant`’

`max_iter`：最大迭代次数，默认取值为 200

`warm_start`：取值为“True”时，表示使用之前调用的参数进行初始化，否则覆盖掉之前的结果

`nesterovs_momentum`：是否使用 Nesterov momentum 优化算法

`momentum`：用于设置优化梯度下降算法的 `momentum` 值，取值范围在 0-1 之间，默认取值为 0.9

`early_stopping`：当交叉验证得分没有提升时是否提前终止迭代，如果取值为“True”时，会自动预留 10% 的训练集作为验证样本，当连续两次迭代交叉验证得分增加不超过参数 `tol` 时，终止训练

`validation_fraction`：在需要提前终止训练时，该参数用于设置用来验证的训练样本的比重，

(1) 相关方法

@todo 补充方法列表

(2) 相关属性

@@todo 补充属性列表

6.3.2 使用 `TextGrocery` 进行文本分类

`TextGrocery` 是基于 `LibLinear` 和 `jieba` 分词的 Python 短文本分类工具，同时支持中文和英文语料，安装和使用都较为简单，可访问官方文档进行详细了解

<http://textgrocery.readthedocs.io/zh/latest/index.html#>。

6.3.2.1 安装 TextGrocery

```
In [22]:!pip install tgrocery
Collecting tgrocery
  Downloading tgrocery-0.1.4.tar.gz
Requirement already satisfied: .....
Building wheels for collected packages: .....
Successfully built tgrocery
Installing collected packages: tgrocery
Successfully installed tgrocery-0.1.4
```

6.3.2.2 Grocery 类

`tgrocery` 模块下的 `Grocery` 类定义了短文本分类器的训练方法，`Grocery` 类实例化方式为：
`实例=Grocery(name, custom_tokenize=None)`

参数说明：

`name`：自定义分类器名称 `custom_tokenize`：指定分词器，默认为 `jieba` 分词，使用者可以自行指定分词函数

(1) train 方法

`train` 方法使用训练集训练样本，调用方式为：实例.`train(train_src, delimiter='\t')`

参数说明：

`train_src`：用于训练分类器的训练集，有两种取值方式：

- (1) 语料列表：列表元素为 (类别，文本) 元组
- (2) 文件路径：一行为一个训练样本，类别标签在前、语料文本在后，默认分隔符是\t

`delimiter`：解析训练集文件的分隔符，默认为"\t"，仅在 `train_src` 为文件路径时有用

(2) predict 方法

`predict` 方法用于预测单一文本类别，调用方式为：实例.`predict(single_text)`，参数 `single_text` 为待预测文本

(3) test 方法

`test` 方法使用测试集测试分类器的准确性，调用方式为：实例.`test(test_src, delimiter='\t')`

参数说明：

`test_src`：用于测试分类器的测试集，有两种取值方式：

- (1) 语料列表：列表元素为 (类别，文本) 元组
- (2) 文件路径：一行为一个测试样本，类别标签在前、语料文本在后，默认分隔符是\t

delimiter：解析测试集文件的分隔符，默认为“\t”，仅在 `test_src` 为文件路径时有用

例 8

```
In [23]:from tgrocery import Grocery
grocery = Grocery('sample')
train_src = [('财经', '格力电器继续排名深股通十大成交活跃股榜首'),
             ('财经', '全国玉米市场化电子交易平台正式启动'),
             ('体育', '英超-桑切斯进球阿森纳1-2遭逆转 14轮不败终结'),
             ('体育', '当年湖人因对抗不足弃波神 斯科特调侃成笑柄')]
grocery.train(train_src)
Out[23]:<tgrocery.Grocery at 0x7fd101daba90>
In [24]:print grocery.predict('吉林东北虎一战重返前八 卡姆拉尼弑杀旧主')
Out[24]:体育
In [25]:test_src = [('财经', '发改委今日调油价 成品油价或迎四年内最大涨幅'),
                  ('体育', '裁判界C罗！英超名哨动作引网友群嘲：他要罚球吗')]
print grocery.test(test_src)
Out[25]:0.5
```

6.3.3 使用 NLTK 进行文本分类

NLTK 库中 `classify` 包给出了文本分类的相关接口和模块，包括决策树、朴素贝叶斯、支持向量机等。在 NLTK 文本分类过程中，有以下几个常用的概念：

feature set：字面意思是特征集合，实际上是一个样本所包含的特征名称（`feature names`）和特征值（`feature values`）一一对应构成的字典，形如 `{feature name:feature value}`，`feature sets` 即为所有样本的 `feature set` 的集合

labeled feature sets：带有类别标记的 `feature sets`，形如 `[{feature name:feature value},label]`，是通用的训练集表示方式，也可以用于分类器分类效果的评价

比如，现有 8 个文档，包含“x,y,z”三种字符串，分为“1,2”两类，以字符串为特征项，字符串出现次数为特征值，构建可用于训练 NLTK 分类器的训练集和测试集如下：

```

train = [(dict(x=5,y=8,z=1), '1'),
          (dict(x=3,y=5,z=0), '1'),
          (dict(x=6,y=4,z=0), '1'),
          (dict(x=1,y=9,z=2), '1'),
          (dict(x=0,y=1,z=7), '2'),
          (dict(x=1,y=1,z=7), '2'),
          (dict(x=1,y=1,z=8), '2'),
          (dict(x=2,y=0,z=7), '2')]

test = [(dict(x=3,y=8,z=1), '1'),
         (dict(x=7,y=1,z=2), '1'),
         (dict(x=2,y=1,z=9), '2'),
         (dict(x=0,y=3,z=6), '2')]

```

6.3.3.1 词袋特征提取

在基于文本数据构建分类器之前，有必要先进行相应的结构化处理，而 NLTK 分类器指定传入上述字典类型的 feature set，词袋模型是进行这一转换的最简单的方法，即对每一个文本样本中的词汇构建一个是否存在的特征集合，featx.py (<https://github.com/japerk/nltk3-cookbook/blob/master/featx.py>) 中的 bag_of_words 函数可以快速实现这一转换，使用方式也颇为简单：

```

In [26]:from featx import bag_of_words
        bag_of_words(['bag', 'of', 'words'])
Out[26]:{'bag': True, 'of': True, 'words': True}

```

除函数 bag_of_words 外，featx.py 中还定义了其他稍微复杂的文本结构化处理函数，如指定词汇筛选 (bag_of_words_not_in_set)、去除停用词 (bag_of_non_stopwords)、输出 bigram 词组 (bag_of_bigrams_words) 等等。

6.3.3.2 朴素贝叶斯分类器

得到 feature sets 形式的文本数据后，就可以训练分类器了，首先对构建朴素贝叶斯分类器的类和方法进行说明。NLTK 库 naivebayes 模块下定义了 NaiveBayesClassifier 类及相应的方法训练朴素贝叶斯分类器，NaiveBayesClassifier 类初始化方式为：

NaiveBayesClassifier(label_probdist, feature_probdist)，利用 train 方法计算相关参数值并传入进而得到训练模型。

参数说明：

label_probdist：基于训练集计算的各个类别的先验概率

feature_probdist：基于训练集计算的给定类别时各个特征的条件概率

相关类方法及调用方式见下表：

@todo 补充方法列表

另外，利用 `classify` 包 `util` 模块的 `accuracy` 函数可以计算分类器的准确性，调用方式为：
`accuracy(classifier, gold)`，其中参数 `classifier` 即为需要评价的分类器，`gold` 为用来评价的测试集数据。

例 9

```
In [26]:from nltk import NaiveBayesClassifier
        from nltk.classify.util import accuracy
        nb_classifier = NaiveBayesClassifier.train(train)
        nb_classifier.classify_many([dict(x=3,y=8,z=1),dict(x=7,y=1,z=2)])
Out[26]:['1', '2']
In [27]:probs = nb_classifier.prob_classify(test[0][0])
        probs.samples()
Out[27]:['1', '2']
In [28]:probs.prob('1')
Out[28]:0.9642857142857137
In [29]:probs.prob('2')
Out[29]:0.03571428571428566
In [30]:nb_classifier.most_informative_features(5)
Out[30]:[('x', 1), ('z', 0), ('y', 9), ('z', 1), ('y', 1)]
In [31]:accuracy(nb_classifier, test)
Out[31]:0.75
```

6.3.3.3 决策树分类器

`classify` 包中的 `decisiontree` 模块中定义了决策树类 `DecisionTreeClassifier` 和相关方法来训练决策树分类器，该类同样通过调用 `train` 方法进行模型训练，调用方式为：

`train(labeled_featuresets, entropy_cutoff=0.05, depth_cutoff=100, support_cutoff=10, binary=False, feature_values=None, verbose=False)`

参数说明：

`labeled_featuresets`：训练集，形如`[({feature name:feature value},label)]`

`entropy_cutoff`：该参数用于决策树优化过程，即决策树是否需要扩展新的决策枝，如果新增决策枝的类别概率分布熵值大于该参数值，说明在当前节点或者特征项下，类别分布差异较小，应该继续优化，否则停止优化，也就是说当某一类别出现的次数显著高于其他类别时，决策树不需要进一步扩展

`depth_cutoff`：最大决策树深度，默认取值为 100，降低该参数值可以加快训练速度，但是也很有可能降低分类器的准确性

support_cutoff：该参数用于控制优化决策树所需的样本数目，在决策树优化过程中，一旦样本对训练过程不提供新的信息则不考虑该样本，当样本数小于或等于该参数值时，则停止优化

binary：当所有的特征项取值均为二元类型时，该参数取值为“True”，注意不是训练二元分类器

其他相关类方法及调用方式见下表：

@todo 补充方法列表

例 10

```
In [32]:from nltk import DecisionTreeClassifier
        dt_classifier = DecisionTreeClassifier.train(train, entropy_cutoff=0.8, depth_c
utoff=5, support_cutoff=2)
        dt_classifier.classify(test[0][0])
Out[32]:'1'
In [33]:dt_classifier.error(test)
Out[33]:0.25
In [34]:accuracy(dt_classifier, test)
Out[34]:0.75
```

6.3.3.4 最大熵分类器（maximum entropy classifier）

classify 包中的 maxent 模块中定义了类 MaxentClassifier 和相关方法来训练最大熵分类器，该类同样通过调用 train 方法进行模型训练，调用方式为：train(cls, train_toks, algorithm=None, trace=3, encoding=None, labels=None, gaussian_prior_sigma=0, **cutoffs)

参数说明：

train_toks：训练集，形如[({feature name:feature value},label)]

algorithm：用于设置训练算法，可选算法包括：'GIS'（Generalized Iterative Scaling）、'IIS'（Improved Iterative Scaling）、「megam」（LM-BFGS algorithm with training performed by Megam），默认取值是'IIS'

trace：用于设置分类器训练结束后输出的信息详细水平，值越高输出越详细，默认取值为 3

labels：所有可能的类别标签集合，若不指定，则默认使用训练集中出现的所有类别

gaussian_prior_sigma：在算法为'megam'时，该参数用于设置高斯先验模型权数

cutoffs：其他用于终止训练过程的参数，包括：

(1) **max_iter**：最大迭代次数

(2) **min_lldelta**：如果两次迭代之间对数似然函数的变动值小于该参数，则停止迭代

(3) min_ll：如果负平均似然函数值小于该参数，停止迭代

其他相关类方法及调用方式见下表：

@todo 补充方法列表

例 11

```
In [35]:from nltk.classify import MaxentClassifier
me_classifier = MaxentClassifier.train(train)
Out[35]: ==> Training (100 iterations)

      Iteration      Log Likelihood      Accuracy
-----+-----+-----+-----+-----+-----+
          1            -0.69315        0.500
          2            -0.42087        1.000
          3            -0.30768        1.000
          ....
          97           -0.01176        1.000
          98           -0.01164        1.000
          99           -0.01152        1.000
          Final         -0.01141        1.000

In [36]:me_classifier.weights()
Out[36]:array([ 2.21461873,  2.21461873,  2.21461873,  2.21461873,  2.21461873,
               2.21461873,  2.21461873,  2.21461873,  3.75522551, -1.21475776,
               3.75522551,  2.30467634,  2.18639661,  2.25638061,  0.5696256 ,
               2.35114273,  2.20626101,  2.20626101])

In [37]:me_classifier.classify(test[0][0])
Out[37]:'1'
In [38]:accuracy(me_classifier, test)
Out[38]:1.0
```

6.3.4 使用 TextBlob 进行文本分类

TextBlob 库也封装了 NLTK 中定义的分类工具，但是 TextBlob 对训练集或测试集的形式要求较 NLTK 更为宽松，在利用 TextBlob 进行文本分类之前，毋需将文本数据转换为形如 `[({feature name:feature value},label)]` 的 labeled feature sets，直接传入 `[(text,label)]` 即可，其中 `text` 即为原始的待分类文本字符串，当然也可以直接读入文本文件。下面以一个简单的例子快速说明 TextBlob 库的文本分类工具的使用方法。

例 12

首先安装最新版本的 TextBlob

```
In [39]:!pip install -U textblob
Collecting textblob
  Downloading textblob-0.11.1-py2.py3-none-any.whl (634kB)
    100% |██████████| 634kB 33kB/s
Requirement already up-to-date: .....
Installing collected packages: textblob
Successfully installed textblob-0.11.1
In [40]:import textblob
         from textblob.classifiers import NaiveBayesClassifier
```

以情感分类文本数据为例

```
In [41]:train = [('I love this film.', 'pos'),
              ('he is an amazing actor!', 'pos'),
              ("what an awesome story", 'pos'),
              ('I do not like this cinema', 'neg'),
              ('a horrible story line.', 'neg')]
test = [('the film was good.', 'pos'),
        ("I feel amazing!", 'pos'),
        ("It sucks.", 'neg')]
In [42]:cl = NaiveBayesClassifier(train) # 训练分类器
In [43]:cl.classify(test[0][0]) # 预测文本情感分类
Out[43]:'pos'
In [44]:cl.accuracy(test) # 检验分类器准确率
Out[44]:0.6666666666666666
```

6.3.5 使用 Pattern 进行文本分类

Pattern 库 vector 包下定义了 IGTree、KNN、NB、SLP、SVM 类实现分类算法，相关类方法的调用方式一致，下面以 KNN 为例进行说明。KNN 类实例化方式为：实例=KNN(train=[], baseline='majority', k=10, distance='cosine')

参数说明：

train：训练集数据列表，列表元素取值类型为 Document，或者(document, type)元组，其中 document 可以是 Document、向量、字典、字符串。这里面提到的 Document 是 vector 模块中定义的类，在进行分类之前需要调用相关方法将文本数据转换为 Document，转换方式为：document = Document(string, punctuation = '.,;:!?"[]{}\"@#\$*+-|=~_', top = None, threshold = 0, exclude = [], stemmer = None, stopwords = False, type = None, language = None)

相关参数说明：

string：待处理文本，可以是字符串，也可以是包含字符串数值元组的列表、字典等
punctuation：需要过滤掉的标点符号 **top**：过滤掉出现次数不在此最频繁范围之内的词汇
threshold：删除出现次数小于该值的词汇 **exclude**：直接过滤掉该列表中的词汇 **stemmer**：词干化处理算法，有以下四种取值：**STEMMER**、**LEMMA**、自定义函数、**None** **stopwords**：是否包含停用词 **type**：类别标注 **language**：语言种类，有以下几种取值：**en**（英语）、**es**（西班牙语）、**de**（德语）、**fr**（法语）、**it**（意大利语）、**nl**（荷兰语）

Document 类支持词频统计等方法，具体可以参考官方文档
(<http://www.clips.ua.ac.be/pages/pattern-vector#document>)。

例 13

```
In [45]:import pattern
        from pattern.vector import Document
        s="text mining with python"
        d = Document(s, threshold=0, stopwords = True)
        d.words
Out[45]:{u'mining': 1, u'python': 1, u'text': 1, u'with': 1}
```

baseline：默认的类别预测规则，多数投票法或者自行定义

k：分类数

distance：距离衡量方式，默认取值为 **COSINE**，即夹角余弦，此外还支持 **EUCLIDEAN**、**MANHATTAN** 和 **HAMMING**

相关类方法及调用方式见下表：

@todo 补充方法列表

例 14 以例 12 中的情感分类文本为例

```
In [46]:import pattern
        from pattern.vector import KNN,Document
        train = [Document('I love this film.', type='pos'),
                  Document('he is an amazing actor!', type='pos'),
                  Document("what an awesome story", type='pos'),
                  Document('I do not like this cinema',type='neg'),
                  Document('a horrible story line.', type='neg')]
        test = [Document('the film was good.', type='pos'),
                  Document("I feel amazing!", type='pos'),
                  Document("It sucks.", type='neg')]
        cl=KNN(train=train,k=2)
        cl.classify(test[0], discrete=True)
Out[46]:'pos'
In [47]:cl.classify(test[0], discrete=False)
Out[47]:defaultdict(float, {'pos': 1.0})
In [48]:cl.test(test)
Out[48]:(0.6666666666666666, 0.3333333333333333, 0.5, 0.4)
```

6.4 常用文本聚类工具

6.4.1 使用 scikit-learn 进行文本聚类

scikit-learn 库提供了丰富的聚类算法，本部分选取几个常用聚类方法进行说明。

6.4.1.1 k-means 聚类

sklearn 库 cluster 包中的 `kmeans` 模块下定义了实现 k-means 聚类的类 `KMeans` 及相关方法，`KMeans` 类实例化方式为：实例=`KMeans(n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=1, algorithm='auto')`

参数说明：

`n_clusters`：聚类数，也是需要初始化的类中心的个数，默认取值为 8

`max_iter`：一次聚类算法所执行的最大迭代次数，默认取值为 300

`n_init`：使用不同的初始化类中心进行聚类的次数，最终输出结果为几次聚类中效果最好的，以组内距离衡量

`init`：初始化方法，有以下几种取值方式：

- (1) ‘k-means++’：使用 k-means++ 算法选取初始聚类中心以加速收敛过程
- (2) ‘random’：随机选取样本点最为初始聚类中心
- (3) 由使用者自行制定初始聚类中心，形如 (`n_clusters, n_features`) 的对象
默认取值为‘k-means++’

`algorithm`：选择聚类算法，有以下几种取值方式：

- (1) “auto”：根据数据类型自动选择合适的算法，稀疏数据选择“full”算法，否则选择“elkan”算法
- (2) “full”：经典的 EM 算法
- (3) “elkan”：使用三角不等式进行“elkan”变换，该算法目前不支持稀疏数据
默认取值为“auto”

`precompute_distances`：是否进行距离的预先计算，有以下几种取值方式：

- (1) ‘auto’：若样本总数乘以聚类数 `n_samples * n_clusters` 大于 1200 万，就不进行预计算
- (2) `True`：进行预计算
- (3) `False`：不进行预计算
默认取值为‘auto’

`tol`：当两次迭代的组内距离之差小于该值时，停止迭代

n_jobs：设置并行计算的处理器个数。取值为“-1”时，使用所有的处理器；取值为“1”时，不进行并行计算；小于“-1”时，使用($n_cpus + 1 + n_jobs$)个处理器进行计算

random_state：随机初始化聚类中心的随机规则。取值为整数时，随机规则固定，默认使用 numpy 随机数生成器

相关方法及属性见下表：

@todo 补充方法、属性列表

例 1

```
In [1]:from sklearn.cluster import KMeans
import numpy as np
X = np.array([[1, 2, 3], [1, 3, 4], [1, 2, 0],
              [4, 5, 2], [4, 4, 6], [4, 5, 0]])
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)

Out[1]:KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=2, n_init=10, n_j
obs=1, precompute_distances='auto', random_state=None, tol=0.0001, verbose=0)
In [2]:kmeans.fit_predict(X)
Out[2]:array([0, 0, 0, 1, 1, 1], dtype=int32)
In [3]:kmeans.cluster_centers_
Out[3]:array([[ 1.          ,  2.33333333,  2.33333333],
              [ 4.          ,  4.66666667,  2.66666667]])
In [4]:kmeans.inertia_
Out[4]:28.666666666666664
```

6.4.1.2 Birch 聚类

sklearn 库 cluster 包中的 birch 模块下定义了实现 Birch 聚类的类 Birch 及相关方法，Birch 类实例化方式为：实例=Birch(threshold=0.5, branching_factor=50, n_clusters=3, compute_labels=True, copy=True)

参数说明：

threshold：叶节点簇直径阈值，如果新的样本点加入后，簇的直径大于该值，则需要新建簇，默认取值为 0.5

branching_factor：每个节点所能包含的最大聚类特征（CF）数，如果超过该值，则需要分割当前节点，默认取值是 50

n_clusters：用于设置最后的聚类数，有以下几种取值方式：

(1) 整数：当原始聚类结果类别数不等于该值时，对聚类结果进行聚合 (2) **None**：不设置时直接返回原始聚类结果

(3) cluster 包的其他聚类模型：将叶节点的子类作为新样本继续按照指定聚类模型进行聚类，默认取值为 3

`compute_labels`：是否输出类别标签，默认取值为 `True`

`copy`：是否备份原始数据，默认取值为 `True`，取值为 `False` 时，原始数据会被覆盖

相关方法及属性见下表：

@todo 补充方法、属性列表

例 2

```
In [5]:from sklearn.cluster import Birch
      x = np.array([[0, 0, 1], [0.5, 0.3, 1], [-0.2, -1, 1], [0, -0.4, -1], [0.3, 0.2
      , -1]])
      birch = Birch(n_clusters=None)
      birch.fit(X)
Out[5]:Birch(branching_factor=50, compute_labels=True, copy=True, n_clusters=None, threshold=0.5)
In [6]:birch.predict(X)
Out[6]:array([0, 0, 1, 2, 2])
```

6.4.2 使用 NLTK 进行文本聚类

NLTK 工具库中的 `cluster` 包中有几种基本的聚类算法，包括 k-means 聚类、E-M 聚类、GAAC 聚类，这些聚类方法对于多维空间向量聚类均有较好的效果。

6.4.2.1 k-means 聚类

`kmeans` 模块下定义了 `KMeansClusterer` 类及相关方法实现 k-means 聚类，`KMeansClusterer` 类实例化方式为：`实例=KMeansClusterer(num_means, distance, repeats=1, conv_test=1e-06, initial_means=None, normalise=False, svd_dimensions=None, rng=None, avoid_empty_clusters=False)`

参数说明：

`num_means`：初始聚类中心的个数

`distance`：距离定义函数，用于衡量向量之间的距离，可以自行定义，也可以直接调用 `util` 模块下定义的距离函数，包括：

- (1) `cosine_distance`：夹角余弦距离
- (2) `euclidean_distance`：欧氏距离

`repeats`：随机初始化聚类次数

`conv_test`：如果类中心变动小于该参数，则停止迭代

`initial_means`：用于指定初始聚类中心向量

`normalise`：是否对聚类向量进行标准化处理

`svd_dimensions`：使用 SVD 进行降维的维度

`rng`：随机数生成器

`avoid_empty_clusters`：在下一次迭代计算过程中考虑当前的聚类中心，以免出现类别为空的情况

相关方法及属性见下表：

@todo 补充方法、属性列表

例 3

```
In [7]:from nltk.cluster import KMeansClusterer
      X = np.array([[0, 0, 1], [0.5, 0.3, 1], [-0.2, -1, 1], [0, -0.4, -1], [0.3, 0.2,
      , -1]])
      km=KMeansClusterer(num_means=3,distance=nltk.cluster.util.cosine_distance)
      km.cluster(X)
      for i in X:
          print i,km.classify(i)
Out[7]:[ 0.  0.  1.] 1
       [ 0.5  0.3  1.] 1
       [-0.2 -1.  1.] 0
       [ 0.  -0.4 -1.] 2
       [ 0.3  0.2 -1.] 2
In [8]:km.means()
Out[8]:[array([-0.2, -1. ,  1. ]),
       array([ 0.25,  0.15,  1. ]),
       array([ 0.15, -0.1 , -1. ])]
In [9]:km.num_clusters()
Out[9]:3
```

6.4.2.2 GMM

em 模块下定义了 `EMClusterer` 类及相关方法实现混合高斯聚类，采用 EM 算法求解，`EMClusterer` 类实例化方式为：`实例=EMClusterer(initial_means, priors=None, covariance_matrices=None, conv_threshold=1e-06, bias=0.1, normalise=False, svd_dimensions=None)`

参数说明：

`initial_means`：每一类的初始化聚类中心

`priors`：每一类的先验概率

`covariance_matrices`：每一类的协方差矩阵

`conv_threshold`：当似然函数值变动小于该值时，停止迭代

`bias`：确保非奇异协方差矩阵的偏差

`normalise`：是否对向量进行标准化处理

`svd_dimensions`：使用 SVD 进行降维的维度

相关方法及属性见下表：

@todo 补充方法、属性列表

例 4

```
In [10]:from nltk.cluster import EMClusterer
        X = np.array([[0, 0, 1], [0.5, 0.3, 1], [-0.2, -1, 1], [0, -0.4, -1], [0.3, 0
.2, -1]])
        means=[[-0.2, -1, 1],[ 0.25, 0.15, 1],[ 0.15, -0.1, -1]]
        em=EMClusterer(initial_means=means)
        em.cluster(X)
Out[10]:[[ 0.2  1.  0. ] [[ 1.  0.  0. ]
 [ 0.  1.  0. ]
 [ 0.  0.  1. ]]
 ...
 [ 1.5e-05  3e-05  -4.64910288e-09] [[ 8.94117646e+00  -2.1
1.764706e+00  -3.73064999e-10]
 [ -2.11764706e+00   5.76470588e+00  -2.69806806e-08]
 [ -3.73064999e-10  -2.69806806e-08   9.99999907e+00]]
In [11]:em.cluster_names()
Out[11]:[0, 1, 2]
In [12]:for i in X:
        print i,em.classify(i)
Out[12]:[ 0.  0.  1.] .... 1
 [ 0.5  0.3  1.] .... 1
 [-0.2 -1.  1.] .... 1
 [ 0.  -0.4 -1.] .... 2
 [ 0.3  0.2 -1.] .... 2
```

6.4.2.3 GAAC 聚类

gaac 模块下定义了 `GAACClusterer` 类及相关方法实现聚合分层聚类，采用夹角余弦度量向量相似度，`GAACClusterer` 类实例化方式为：实例=`GAACClusterer(num_clusters=1, normalise=True, svd_dimensions=None)`

参数说明：

num clusters : 聚类数

normalise：是否对向量进行标准化处理

`svd_dimensions`: 使用 SVD 进行降维的维度

相关方法及属性见下表：

@todo 补充方法、属性列表

例 5

```
In [13]:from nltk.cluster.gaac import GAAClusterer
        X = np.array([[0, 0, 1], [0.5, 0.3, 1], [-0.2, -1, 1], [0, -0.4, -1], [0.3, 0.2, -1]])
        gaac=GAAClusterer(num_clusters=3)
        gaac.cluster(X)
        gaac.dendrogram().show()
Out[13]:  
+-----+-----+-----+  
| | | |  
| | | +-----+  
| | | |  
+-----+ | | |  
| | | |  
| | | |  
| | | |  
[ 0.  0.  1.]  [ 0.5  0.3  1. ] [-0.2 -1.   1. ] [ 0.  -0.4 -1. ] [ 0.3  0.2  
-1. ]  
In [14]:for i in X:  
        print i,gaac.classify(i)
Out[14]:[ 0.  0.  1.] 1
        [ 0.5  0.3  1. ] 1
        [-0.2 -1.   1. ] 0
        [ 0.  -0.4 -1. ] 2
        [ 0.3  0.2 -1. ] 2
```

6.4.3 使用 Pattern 进行文本聚类

Pattern 工具库中 `vector` 包内定义了 `Model` 类及相关的方法可以实现文本聚类，支持 k-means 聚类和分层聚类，`Model` 类实例化方式为：实例 = `Model(documents= [...], weight='tf-idf')`

参数说明：

documents：需要聚类的文本列表，其元素类型为 Document。

weight：计算文本相似性的依据，包括：TF、TFIDF（默认取值）、IG、BINARY、None（使用初始文本权重）

对 Model 类实例调用 cluster 方法即可实现聚类，调用方式为：实例.cluster(method, k, iterations, distance)

参数说明：

method：聚类方法，支持 KMEANS 和 HIERARCHICAL 两种聚类算法

k：聚类数

iterations：迭代次数

distance：距离度量方式，默认取值为 COSINE，即夹角余弦，此外还支持 EUCLIDEAN、MANHATTAN 和 HAMMING

例 7

```
In [16]:from pattern.vector import HIERARCHICAL,KMEANS
        from pattern.vector import Model
        text=['Cats are independent pets.', 'Dogs are trustworthy pets.', 'Boxes are made of cardboard.']
        document=[Document(i) for i in text]
        document
Out[16]:[Document(id='Q6TVPey-14'),
         Document(id='Q6TVPey-15'),
         Document(id='Q6TVPey-16')]
In [17]:model=Model(document)
        model.cluster(method=KMEANS, k=2)
Out[17]:[[Document(id='Q6TVPey-14'), Document(id='Q6TVPey-16')],
          [Document(id='Q6TVPey-15')]]
```

6.4.4 聚类效果评价指标

sklearn 库 metrics 模块下的 cluster 包给出了多种聚类效果评价指标，按照聚类对象是否有真实类别标记，可分为两类，即有监督和无监督，使用者可以根据实际数据情况考量各个指标的优缺点自行选择适合的评价指标，本部分对部分指标的计算原理及调用方式做简要说明：

6.4.4.1 有监督

(1) 调整兰德指数 (Adjusted Rand index)

如果已知聚类对象的实际类别标签，结合预测的分类结果，就可以计算调整兰德指数对比二者的相似度，对于一个样本量为 n 的样本空间 S ，我们将空间中原本的样本标签划分与我们要评价聚类结果划分分别用 $X=\{X_1, X_2, \dots, X_n\}$ 与 $Y=\{Y_1, Y_2, \dots, Y_n\}$ 表示，此时调整兰得指数 R 可以由如下公式表示：

$$R = \frac{a + b}{a + b + c + d} = \frac{a + b}{\binom{n}{2}}$$

其中，上式中的指标 a, b, c, d 含义如下：

- a : 在样本空间 S 中，在 X 划分中属于同一类且在 Y 划分中属于同一类的样本对数
- b : 在样本空间 S 中，在 X 划分中属于不同类且在 Y 划分中属于不同类的样本对数
- c : 在样本空间 S 中，在 X 划分中属于同一类且在 Y 划分中属于不同类的样本对数
- d : 在样本空间 S 中，在 X 划分中属于不同类且在 Y 划分中属于同一类的样本对数

可以看到，上式中的指标 $a+b$ 的值越大，反映了划分 X 与 Y 之间的一致程度越强；而指标 $c+d$ 的值越大，则反映了划分 X 与 Y 之间的差异程度越高。调整兰德指数正是以这样一种直观的比值形式来反映聚类结果与样本标签的相似程度。Scikit-Learn 中的 cluster 包提供了调整兰德指数计算函数 `adjusted_rand_score`，调用方式如下：

```
sklearn.metrics.adjusted_rand_score(labels_true, labels_pred)
```

参数说明：

`labels_true`：样本真实类别标签列表

`labels_pred`：预测的类别标签列表

该函数直接返回取值范围在-1到1之间的聚类结果相似度得分，如果结果位于0-1之间表明聚类效果较好。

例 8

```
In [18]:from sklearn import metrics
        labels_true = [0, 0, 1, 1, 1]
        labels_pred = [0, 0, 1, 1, 2]
        metrics.adjusted_rand_score(labels_true, labels_pred)
Out[18]:0.5454545454545454
In [19]:labels_pred = [0, 0, 1, 1, 1]
        metrics.adjusted_rand_score(labels_true, labels_pred)
Out[19]:1.0
In [20]:labels_pred = [1, 1, 0, 0, 0]
        metrics.adjusted_rand_score(labels_true, labels_pred)
Out[20]:1.0
```

(2) 互信息 (Mutual Information)

作为信息论里一种常用的信息度量，互信息也是对两组分类集合相似度的衡量。互信息 I 的定义如下式所示：

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

其中 $H(X, Y)$ 为联合熵， $H(X|Y) = \sum p(x,y) \log p(x,y)$; $H(X|Y)$ 为条件熵， $H(X|Y) = \sum_x \sum_y p(xy) \log_2 p(x|y)$

为了消除聚类中心和样本数对聚类效果评价产生的影响，有时我们也采用 AMI (Adjusted Mutual Information) 对聚类结果的互信息水平进行评价，其定义如下：

$$AMI(U, V) = [MI(U, V) - E(MI(U, V))] / [\max(H(U), H(V)) - E(MI(U, V))]$$

cluster 包提供了互信息计算函数 `normalized_mutual_info_score` 和 `adjusted_mutual_info_score` 分别计算 Normalized Mutual Information(NMI) 和 Adjusted Mutual Information(AMI) 两种互信息，调用方式为：

`sklearn.metrics.normalized_mutual_info_score/adjusted_mutual_info_score(labels_true, labels_pred)`

例 9

```
In [21]:labels_true = [0, 0, 1, 1, 1]
        labels_pred = [0, 0, 1, 1, 2]
        metrics.normalized_mutual_info_score(labels_true, labels_pred)
Out[21]:0.79873276276445015
In [22]:metrics.adjusted_mutual_info_score(labels_true, labels_pred)
Out[22]:0.46557755827004066
In [23]:labels_pred = [1, 1, 0, 0, 0]
        metrics.adjusted_mutual_info_score(labels_true, labels_pred)
Out[23]:1.0
```

(3) 同质性 (Homogeneity) 和完备性 (completeness)

同质性（Homogeneity）用于衡量聚类结果中类内样本之间的平均相似度，相似度越高，代表该聚类结果中的簇越“纯”；而完备性（Completeness）则用于反映属于相同类别的样本对象是否都被聚类入同一个簇。

对于含有 n 个样本的样本空间 S ，划分 $X=\{x_1, x_2, \dots, x_n\}$ 为原本的样本标记，而划分 $Y=\{y_1, y_2, \dots, y_n\}$ 则为我们的聚类结果，我们以 a_{ij} 来表示属于类别 x_i 且属于聚类簇 y_j 的样本序号。此时，聚类结果的同质性 h 以及完备性 c 可由下式表示：

$$h = \begin{cases} 1 & \text{if } H(X, Y) = 0 \\ 1 - \frac{H(X|Y)}{H(X)} & \text{else} \end{cases}$$

$$c = \begin{cases} 1 & \text{if } H(Y, X) = 0 \\ 1 - \frac{H(Y|X)}{H(Y)} & \text{else} \end{cases}$$

其中，

$$H(X|Y) = -\sum y_j \frac{\sum a_{ij}}{n} \log \frac{\sum a_{ij}}{\sum x_i a_{ij}}$$

$$H(X) = -\sum y_j \frac{\sum a_{ij}}{n} \log \frac{\sum a_{ij}}{n}$$

不难看出，当一个聚类结果同时具有较高的同质性和完备性水平时，聚类的效果也较好，适合我们在分析中采用。

`cluster` 包提供了互信息计算函数 `homogeneity_score` 和 `completeness_score` 分别计算同质性和完备性，调用方式为：

```
sklearn.metrics.homogeneity_score/completeness_score(labels_true, labels_pred)
```

例 10

```
In [24]: labels_true = [0, 0, 1, 1, 1]
                  labels_pred = [0, 0, 1, 1, 2]
                  metrics.homogeneity_score(labels_true, labels_pred)
Out[24]: 1.0
In [25]: metrics.completeness_score(labels_true, labels_pred)
Out[25]: 0.63797402631333133
```

6.4.4.2 无监督

在实际聚类过程中，聚类对象的真实类别往往是未知的，此时只能借助聚类模型本身得到聚类效果评价指标，轮廓系数（Silhouette Coefficient）就是较为常用的一种。

轮廓系数是在没有基准可用的情况下考察聚类结果中类的分离与紧凑情况的度量。对于样本量为 n 的样本数据集 D ，假设我们的聚类结果将其划分为 k 个簇 C_1, C_2, \dots, C_n 。而对于每个样本 $o \in C_i$ ，我们可以计算出它与所属簇中其他样本之间的平均距离 $a(o)$ ，以及它与其他簇之间的最小平均距离 $b(o)$ ，即

$$s(o) = \frac{\sum_{i \neq o} dist(o, o_i)}{|C_o| - 1}$$

$$b(o) = \min \{ \frac{\sum_{j \neq o} dist(o, o_j)}{|C_j|} \}$$

此时，样本 o 的轮廓系数 $s(o)$ 定义为：

$$s(o) = \frac{b(o) - a(o)}{\max\{a(o), b(o)\}}$$

可以看到，轮廓系数的值在 -1 与 1 之间，当样本 o 的轮廓系数越接近于 1 时，表明聚类的效果越好；而当轮廓系数的值为负时，则说明我们的这次聚类相当失败，可以完全舍弃了。

cluster 包提供了轮廓系数计算函数 `silhouette_score`，计算所有样本轮廓系数的均值，调用方式如下：`silhouette_score(X, labels, metric='euclidean', sample_size=None, random_state=None)`

参数说明：

`X`：待聚类样本特征数组或者已经计算好的样本两两之间的距离数组

`labels`：聚类得到的每一个样本的类别标签数组

`metric`：计算轮廓系数所使用的距离度量方式，有以下两种取值方式：

(1) 字符串：必须是函数 `metrics.pairwise.pairwise_distances` 定义下的距离，包括 '`cityblock`'、'`cosine`'、'`euclidean`'、'`l1`'、'`l2`'、'`manhattan`' 等，更加具体的信息可以参见 `pairwise_distances` 函数说明 (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances.html#sklearn.metrics.pairwise.pairwise_distances)

(2) "precomputed"：当 `X` 为已经计算好的样本两两之间的距离数组时，`metric` 只能取该值

`sample_size`：用于计算轮廓系数的样本个数，取值为整数或 `None`（不抽样）

`random_state`：当 `sample_size` 取值非 `None` 时，使用该参数设置用于生成指定样本数的随机数生成器

例 11

```
In [26]:import numpy as np
X = np.array(([ 5,  3,  1,  0],
              [ 4,  3,  1,  1],
              [ -4,  3,  1,  0],
              [ -4,  3,  1,  2],
              [ -5,  3,  1,  2]))
y = np.array((0,0,1,1))
metrics.silhouette_score(X, y, metric='euclidean')
Out[26]:0.81845375003407805
```


6.5 文本分类、聚类分析实例

本部分以 NLTK 提供的文本数据为例，构建了电影评论文本情感分类的分类器，对 20 newsgroups 的新闻文本进行了聚类，并对比了不同算法的效果。

6.5.1 电影评论文本情感分类

NLTK 提供的 movie_reviews 电影评价文本数据被分为两类：pos 和 neg，即正向评价和负向评价，属于文本情感分类。在训练过程中我们以每一条电影评论为一个样本，构建一个二元分类器即可。

在调用文本分类算法之前，需要先加载数据，根据不同的工具库分类器要求将文本数据转换为指定形式，并合理划分训练集和测试集。

首先导入文本数据并查看数据基本信息

```
In [1]:from nltk.corpus import movie_reviews

# 可以通过调用以下方法查看相关文本信息

In [2]:movie_reviews.fileids() # 查看语料文件名称
Out[2]:[u'neg/cv000_29416.txt',
       u'neg/cv001_19502.txt',
       ....]

In [3]:movie_reviews.raw() # 查看具体文本内容
Out[3]:u'plot : two teen couples go to a church party , drink and then drive . ..... \nne
vertheless , i look forward to niccol\'s next film , whatever it may be . \n'
In [4]:movie_reviews.sents() # 分句查看文本内容
Out[4]:[[u'plot', u':', u'two', u'teen', u'couples', u'go', u'to', u'a', u'church', u'
party', u',', u'drink', u'and', u'then', u'drive', u'.'], [u'they', u'get', u'into', u
'an', u'accident', u'.'], ...]
In [5]:movie_reviews.categories() # 查看文本具体分类
Out[5]:[u'neg', u'pos']
```

下面选取不同的工具库、不同的分类算法对以上数据训练分类器，对比分类效果：

6.5.1.1 NLTK 多项朴素贝叶斯分类器 MultinomialNB

利用 NLTK 进行分类首先要将文本数据结构化为 [(featureset, label)] 的形式，可以使用 featx.py 中的 bag_of_words 函数直接标注词汇是否出现，该函数定义如下：

```
def bag_of_words(words):
    ...
    >>> bag_of_words(['the', 'quick', 'brown', 'fox'])
    {'quick': True, 'brown': True, 'the': True, 'fox': True}
    ...
    return dict([(word, True) for word in words])
```

利用上述函数对文本数据进行处理，并打乱原有文本顺序，划分训练集和测试集：

```
In [6]:from featch import bag_of_words
import random
labeled_fsets = [(bag_of_words(movie_reviews.words(fileid)), category)
                  for category in movie_reviews.categories()
                  for fileid in movie_reviews.fileids(category)]
random.shuffle(labeled_fsets)
train_feats = labeled_fsets[:1500]
test_feats = labeled_fsets[1500:]
len(train_feats)
Out[6]:1500
In [7]:len(test_feats)
Out[7]:500
```

电影评论文本由 1000 个正向评论和 1000 个负向评论组成，我们最终使用其中的 1500 条评论作为训练集，其余的 500 条作为测试集。

下面直接利用 NLTK 工具库 NaiveBayesClassifier 类的 train 方法训练朴素贝叶斯分类器。

```
In [8]:from nltk.classify import NaiveBayesClassifier
nb_classifier = NaiveBayesClassifier.train(train_feats)

# 使用 classify 进行分类预测：

In [9]:from featch import bag_of_words
negfeat = bag_of_words(['the', 'story', 'was', 'boring'])
nb_classifier.classify(negfeat)
Out[9]:'neg'

# 利用测试集测试分类器精度

In [10]:from nltk.classify.util import accuracy
accuracy(nb_classifier, test_feats)
Out[10]:0.714
```

下面采用词频作为特征值进行分类，首先使用 word_tokenize 分词、Counter 统计词频，并将数据转换为 labeled feature sets 形式：

```
In [11]:from nltk.tokenize import word_tokenize
      from collections import Counter

      word_counter = Counter()

      labeled_fsets = [(dict(Counter(word_tokenize(movie_reviews.raw(fileid)))), cat
egory)
                      for category in movie_reviews.categories()
                      for fileid in movie_reviews.fileids(category)]
```

将转换结果划分为训练集和测试集，并构建分类器，查看分类效果：

```
In [12]:random.shuffle(labeled_fsets)
      train_feats =labeled_fsets[:1500]
      test_feats=labeled_fsets[1500:]
      nb_classifier = NaiveBayesClassifier.train(train_feats)
      accuracy(nb_classifier, test_feats)
Out[12]:0.774
```

6.5.1.2 TextBlob 朴素贝叶斯分类器 NaiveBayesClassifier

使用 TextBlob 进行文本分类不需要进行上述复杂的文本分词、计数等预处理，直接传入 [(text,label)] 型数据即可：

```
In [13]:import textblob
      from textblob.classifiers import NaiveBayesClassifier

      # 构建符合训练格式的文本数据集合 labeled_fsets

      labeled_fsets = [(movie_reviews.raw(fileid), category)
                      for category in movie_reviews.categories()
                      for fileid in movie_reviews.fileids(category)]

      # 划分训练集和测试集
      random.shuffle(labeled_fsets)
      train_feats =labeled_fsets[:1500]
      test_feats=labeled_fsets[1500:]

      # 训练分类模型并利用测试集计算分类准确率

      cl = NaiveBayesClassifier(train_feats)
      cl.accuracy(test_feats)
Out[13]:
```

6.5.1.3 Pattern KNN

分别以欧式距离和夹角余弦为距离衡量方式，构建 KNN 分类器：

```
In [14]:import pattern
        from pattern.vector import KNN, Document, EUCLIDEAN, COSINE

        # 构建符合训练格式的文本数据集合 labeled_fsets

        labeled_fsets = [Document(movie_reviews.raw(fileid), type=category)
                         for category in movie_reviews.categories()
                         for fileid in movie_reviews.fileids(category)]

        labeled_fsets=[]

        for f in movie_reviews.fileids():
            fset= movie_reviews.raw(fileids=[f])
            label=movie_reviews.categories(fileids=[f])[0]
            labeled_fsets.append(Document(fset, type=label))

        # 划分训练集和测试集

        train_feats =labeled_fsets[:1500]
        test_feats=labeled_fsets[1500:]

        # 使用欧式距离训练分类模型并利用测试集计算分类准确率

        cl=KNN(train=train_feats,k=2,distance=EUCLIDEAN)
        cl.test(test_feats)
Out[14]:(0.352, 1.0, 0.352, 0.5207100591715976)

        # 使用夹角余弦训练分类模型并利用测试集计算分类准确率

In [15]:cl=KNN(train=train_feats,k=2,distance=COSINE)
        cl.test(test_feats)
Out[15]:(0.44, 1.0, 0.44, 0.6111111111111112)
```

虽然两种情况下分类器的准确率均较低，但是夹角余弦的分类效果还是要优于欧式距离。

以上示例在聚类之前都未做严格的停用词过滤、词干化等预处理，读者可以尝试进行适当地预处理，对比分类效果是否有改进。

6.5.2 新闻文本聚类

本部分使用 `sklearn` 内建函数 `fetch_20newsgroups` 加载 `20 newsgroups` 新闻数据集中三类新闻文本，计算 TF-IDF 矩阵作为特征值，使用不同的聚类算法进行聚类。关于 `fetch_20newsgroups` 的调用方法 6.1 节有详细介绍，此处不做多余说明。

同样，在调用文本聚类算法之前，需要先导入文本数据并查看数据基本信息：

```
In [1]:import sklearn
       from sklearn.datasets import fetch_20newsgroups
       dataset = fetch_20newsgroups(subset="train", shuffle=True, random_state=1, remove
=('headers', 'footers', 'quotes'), categories=['alt.atheism', 'talk.politics.mideast', 't
alk.religion.misc'])
       dataset
Out[1]:{'DESCR': None,
        'data': [u'\n\nAnas, of course ! ..... trying to discredit Israel).",.....]
        'description': 'the 20 newsgroups by date dataset',
        'filenames': .....
        'target': array([1, 2, 0, ..., 0, 2, 1]),
        'target_names': ['alt.atheism', 'talk.politics.mideast', 'talk.religion.misc']}
In [2]:len(dataset["data"])
Out[2]:1421
In [3]:text=dataset["data"]
       labels_true = dataset["target"]
```

'alt.atheism'、'talk.politics.mideast'、'talk.religion.misc'三类新闻共 1421 条文本。接下来利用 NLTK cluster 包中的 KMeansClusterer 类对以上文本进行简单的聚类处理：

6.5.2.1 文档词频矩阵聚类

使用 sklearn 库 CountVectorizer 类构建文档词频矩阵 dtm：

```
In [4]:from sklearn.feature_extraction.text import CountVectorizer
       dtm_vectorizer = CountVectorizer()
       dtm = dtm_vectorizer.fit_transform(text).toarray()
```

利用上步得到的 dtm 矩阵直接将样本聚合成三类

```
In [5]:from nltk.cluster import KMeansClusterer
      from sklearn import metrics

# 使用欧式距离进行聚类

In [6]:km=KMeansClusterer(num_means=3,distance=nltk.cluster.util.euclidean_distance)
      km.cluster(dtm)
      labels_pred = [km.classify(i) for i in dtm]
      metrics.homogeneity_score(labels_true, labels_pred)
Out[6]:0.012559370220130626

In [7]:metrics.completeness_score(labels_true, labels_pred)
Out[7]:0.057515794427500502

In [8]:metrics.adjusted_rand_score(labels_true, labels_pred)
Out[8]:-0.003911994181202576

# 使用夹角余弦距离进行聚类

In [9]:km=KMeansClusterer(num_means=3,distance=nltk.cluster.util.cosine_distance)
      km.cluster(dtm)
      labels_pred = [km.classify(i) for i in dtm]
      metrics.homogeneity_score(labels_true, labels_pred)
Out[9]:0.02218039873071102

In [10]:metrics.completeness_score(labels_true, labels_pred)
Out[10]:0.026715706061670171

In [11]:metrics.adjusted_rand_score(labels_true, labels_pred)
Out[11]:0.02496855873580896
```

6.5.2.2 TF-IDF 矩阵聚类

使用 `sklearn` 库 `TfidfVectorizer` 类构建 TF-IDF 矩阵：

```
In [12]:from sklearn.feature_extraction.text import TfidfVectorizer
      vectorizer = TfidfVectorizer(stop_words="english")
      tfidf=vectorizer.fit_transform(text).toarray()
```

利用上步得到的 `tfidf` 矩阵直接将样本聚合成三类

```
# 使用夹角余弦距离进行聚类

In [13]: km=KMeansClusterer(num_means=3,distance=nltk.cluster.util.cosine_distance)
          km.cluster(tfidf)
          labels_pred = [km.classify(i) for i in tfidf]
          metrics.homogeneity_score(labels_true, labels_pred)

Out[13]: 0.23932874020793218
In [14]: metrics.completeness_score(labels_true, labels_pred)
Out[14]: 0.25465755776723092
In [15]: metrics.adjusted_rand_score(labels_true, labels_pred)
Out[15]: 0.30077069169403853
```

6.6 小结

本章内容可以划分为理论、工具和案例三部分，理论部分希望读者可以理解和掌握以下算法：

@todo 补充算法列表

工具库 scikit-learn 和 NLTK 的分类、聚类工具可以参见以下汇总表，具体类方法的调用方式在本章前面的小节中均有介绍，在此未一一列出。在具体分析时读者可以针对实际情况选择合适的算法工具：

@todo 补充算法工具列表

6.7 深度阅读材料

6.7.1 朴素贝叶斯

H. Zhang (2004). The optimality of Naive Bayes. Proc. FLAIRS.

<http://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf>

J. Rennie et al. (2003), Tackling the poor assumptions of naive Bayes text classifiers, ICML.

Naive Bayes text classification <http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html>

A. McCallum and K. Nigam (1998). A comparison of event models for Naive Bayes text classification. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.1529>

V. Metsis, I. Androutsopoulos and G. Paliouras (2006). Spam filtering with Naive Bayes – Which Naive Bayes? <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.5542>

6.7.2 K-最近邻

算法

“Multidimensional binary search trees used for associative searching”, Bentley, J.L., Communications of the ACM (1975) <http://dl.acm.org/citation.cfm?doid=361002.361007>

“Five balltree construction algorithms”, Omohundro, S.M., International Computer Science Institute Technical Report (1989) <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.91.8209>

6.7.3 支持向量机

《支持向量机导论》，[英] Nello Cristianini / John Shawe-Taylor 著

A Tutorial on Support Vector Regression <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.114.4288>

Support-vector networks <http://link.springer.com/article/10.1007%2FBF00994018>

LIBSVM: A Library for Support Vector Machines

<http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>

LIBLINEAR -- A Library for Large Linear Classification

<http://www.csie.ntu.edu.tw/~cjlin/liblinear/>

Probability Estimates for Multi-class Classification by Pairwise Coupling
<http://www.csie.ntu.edu.tw/~cjlin/papers/svmprob/svmprob.pdf>

6.7.4 决策树

L. Breiman, J. Friedman, R. Olshen, and C. Stone, “Classification and Regression Trees”, Wadsworth, Belmont, CA, 1984. <http://www.stat.cmu.edu/~cshalizi/350/lectures/22/lecture-22.pdf>

L. Breiman, and A. Cutler, “Random Forests”
http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

6.7.5 神经网络

“Learning representations by back-propagating errors.” Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams.

http://www.iro.umontreal.ca/~pift6266/A06/refs/backprop_old.pdf

“Backpropagation” Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen - Website, 2011. http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm

“Stochastic Gradient Descent” L. Bottou - Website, 2010. <http://leon.bottou.org/projects/sgd>

Adam: A Method for Stochastic Optimization <https://arxiv.org/abs/1412.6980>

6.7.6 K-means

“k-means++: The advantages of careful seeding” David Arthur and Sergei Vassilvitskii
<http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf>

6.7.7 BIRCH

Tian Zhang, Raghu Ramakrishnan, Maron Livny BIRCH: An efficient data clustering method for large databases. <http://www.cs.sfu.ca/CourseCentral/459/han/papers/zhang96.pdf>

6.8 练习

6.8.1 简述分类和聚类的思想及文本分类和文本聚类的一般步骤。

6.8.2 列举常用文本分类算法并对比不同算法之间的差异。

6.8.3 列举常用文本聚类算法并对比不同算法之间的差异。

6.8.4 列举常用聚类效果评价指标对比不同指标间的优缺点。

6.8.5 了解 **scikit-learn**、**NLTK**、**Pattern**、**TextGrocery** 等工具库支持的分类、聚类算法，掌握常用类方法的调用方式，对路透社新闻

(<http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html>) 文本进行分类、聚类处理，对比不同算法对分类或聚类效果的影响。

第 7 章 主题模型

不论 DTM 还是 TF-IDF 矩阵都以词频为基础作为文档的特征表示，而一词多义或多词一义的情况并不能通过词频反映出来，也就是说忽略了文本中的语意信息，导致进一步文本分析的精度较低，本章介绍的主题模型就是为了解决这一问题而提出的，主题模型可以将文档映射到指定维度的语义空间，利用各个文档在语义空间的分布可以计算文档的相似度，也可以进一步进行文档分类或聚类处理。本章内容安排如下：7.1 节和 7.2 节分别介绍 LSI 模型和 LDA 模型的原理及构建方式，7.3 节则利用主题模型得到的主题分布结果，对已经标注的文本数据进行相似度计算。

7.1 LSI

7.1.1 LSI 简介

LSI 是 Latent Semantic Indexing 的缩写，可以译为潜在语意索引，也常被称为 Latent Semantic Analysis (LSA)。Scott Deerwester 等在 1988 年提出了 LSI，其核心思路为将文档或词矩阵进行奇异值分解 (Singular Value Decomposition, SVD)，使得文档和文档或者文档和查询之间的相似性在简化的潜在语义空间中得以表达。由于奇异值分解的方法本身是对文档特征的排序，可以通过限制奇异值的个数对数据进行降噪和降维，可以较好的挖掘文档的语义信息。

奇异值分解：对于 $m \times n$ 阶矩阵 M ，存在矩阵分解使得 $M = U \Sigma V^T$ ，其中，矩阵 V 和 U 的列分别为 $M^T M$ 与 $M M^T$ 的特征向量，矩阵 Σ 对角线上的元素为 $M^T M$ 及 $M M^T$ 的特征值的非零平方根，这些元素也被称为奇异值。

在文本数据分析中，LSI 的处理对象通常为文档数据集在空间向量模型中的矩阵映射，或者是经过计算得到的 TF-IDF 矩阵，根据奇异值分解结果，LSI 会将文档样本投射到对应维数的潜在语义空间中，是我们能够在较低的维度下判断文档样本的类型特征，对于文本的分类与聚类有明显的帮助。

7.1.2 利用 gensim 构建 LSI 模型

gensim 库 models 包中的 LsiModel 模块中定义了 LsiModel 类及相关方法，实现快速的 SVD 分解，将文档映射到潜在语义空间，LsiModel 类的实例化方式为：实例

```
=LsiModel(corpus=None, num_topics=200, id2word=None, chunksize=20000, decay=1.0,
distributed=False, onepass=True, power_iters=2, extra_samples=100)
```

参数说明：

corpus：用该参数传入的文档语料将会被用来训练模型

id2word：用于设置构建矩阵的词典

`num_topics`：潜在语义空间维度

`chunksize`：一次训练过程中使用的文件块大小，该值较大有利于提高训练速度，但是会占用较大的内存

`distributed`：是否开启分布式计算

`onepass`：是否使用多次随机算法，取值为 `True` 时，使用多次随机算法，否则使用前端算法

`power_iters` 和 `extra_samples`：这两个参数都会影响算法的准确性，迭代参数 `ower_iters` 的增加会增强算法的准确性，更详细的参数影响关系请参见 6.6.4 提供的深度阅读材料。

相关方法及属性如下：

(1) `show_topic` 方法

`show_topic` 方法用于返回指定的主题，调用方式为：实例`.show_topic(topicno, topn=10)`

参数说明：

`topicno`：待返回的主题序号

`topn`：返回的词汇数，默认取值为 10，按对主题贡献度（正值、负值均有）从大到小排序

(2) `show_topics` 方法

`show_topics` 方法可以同时返回多个主题，调用方式为：实例`.show_topics(num_topics=-1, num_words=10, log=False, formatted=True)`

参数说明：

`num_topics`：控制返回的主题数，默认取值为 -1，表示返回全部主题

`num_words`：每个主题包含的词汇数，默认取值为 10

`formatted`：取值为 `True` 时以字符串形式返回词汇及相应概率，取值为 `False` 时返回词汇和相应概率组成的元组

`log`：取值为 `True` 时，同时将结果输出到日志中

(3) “`[]`” 方法

“`[]`” 方法可以返回文档在各个主题的分布，调用方式为：实例`[document]`

(4) `add_documents` 方法

`add_documents` 方法可以使用新的文本语料更新已有的 SDV 分解结果，调用方式为：实例`.add_documents(corpus, chunksize=None, decay=None)`，其中参数 `corpus` 即为新增文本，当参数 `decay` 取值小于 1 时，侧重于使用新的语料进行语意定向，而不是原有语料

(5) `print_debug` 方法

`print_debug` 方法用于输出不同主题的重要词汇，与 `show_topics()` 方法不同，该方法输出的词汇更能突出不同主题之间的差异，结果更易于解释，调用方式为：实例`.print_debug(num_topics=5, num_words=10)`

例 1 本例中 `corpus` 包含的文本是两类新闻标题：环境和教育，利用 LSI 模型将语料映射到二维主题空间

```
In [1]:from gensim import corpora,models
        from gensim.models.lsimodel import LsiModel
        title = ['Beijing lifts red alert with smog set to disperse',
                  'Source of major pollutant in China smog revealed',
                  'Xi stresses clean energy use to reduce smoggy days',
                  'New Zealand university to open innovation center in China',
                  'School combines soccer with kung fu to train players']
        # 简单分词处理
        texts = [[word for word in document.lower().split()] for document in corpus]
        # 构建词典
        dictionary = corpora.Dictionary(texts)
        # 得到词汇词频
        corpus = [dictionary.doc2bow(text) for text in texts]
        # 构建 TF-IDF 矩阵
        tfidf_model = models.TfidfModel(corpus)
        corpus_tfidf = tfidf_model[corpus]
        # 以 TF-IDF 矩阵构建 2 个主题的 LSI
        lsi = LsiModel(corpus=corpus_tfidf,id2word=dictionary,num_topics=2)
        # 查看主题关键词
        lsi.show_topics (num_topics=-1, num_words=10)
Out[1]:[(0,
          u'0.275*"in" + 0.275*"china" + 0.264*"of" + 0.264*"major" + 0.264*"source" +
          0.264*"revealed" + 0.264*"pollutant" + 0.229*"smog" + 0.220*"new" + 0.220*"zealand"),
         (1,
          u'-0.270*"with" + -0.251*"school" + -0.251*"soccer" + -0.251*"kung" + -0.251*
          "train" + -0.251*"players" + -0.251*"combines" + -0.251*"fu" + -0.224*"beijing" + -0.2
          24*"lifts")]
In [2]:# 返回各个文档在两个主题上的分布
        for i in corpus_tfidf:
            print lsi[i]
Out[2]:[(0, 0.40112933717769061), (1, -0.60176390850417838)]
[(0, 0.72129310654139789), (1, 0.18444288880409218)]
[(0, 0.026674205908750175), (1, -0.058666737715942399)]
[(0, 0.63620409768707431), (1, 0.37352685238560701)]
[(0, 0.18038797936342355), (1, -0.70806631562713807)]
```

利用各个文档在不同主题上的分布可以计算文档间的相似度，也可以采用不同的聚类方法进行文档聚类。

7.1.3 利用 `sklearn` 构建 LSI 模型

sklearn 库 decomposition 包也提供了 truncated_svd（截断奇异值分解）模块构建 LSI 模型，主要利用其下定义的类 TruncatedSVD 对文档矩阵进行处理，支持随机规划求解以及使用 ARPACK (<http://www.caam.rice.edu/software/ARPACK/>) 的朴素算法，更多细节可以阅读官方文档 (<http://scikit-learn.org/stable/modules/decomposition.html#lsa>)。该类的初始化方式为：实例=TruncatedSVD(n_components=2, algorithm='randomized', n_iter=5, random_state=None, tol=0.0)

参数说明：

n_components：待输出的语义空间的维度，一定小于原始数据的特征项数目，默认取值为 2，便于可视化展示，在构建 LSI 模型时，建议取值 100

algorithm：用于求解的算法，支持“`arpack`”和“`randomized`”两种，默认取值为“`randomized`”。

n_iter：当算法为“`randomized`”时，用该参数设置算法迭代次数，默认取值为 5

random_state：伪随机数生成器

tol：当算法为“`arpack`”时，用该参数设置残差容忍下限

相关方法及属性如下：

@补充方法属性列表

例 2 利用上例 title 构建 LSI 模型

```
In [3]:import sklearn
        from sklearn.decomposition import TruncatedSVD
        from sklearn.feature_extraction.text import TfidfVectorizer
        # 利用 TfidfVectorizer 计算 TF-IDF 矩阵
        vectorizer = TfidfVectorizer()
        corpus_tfidf=vectorizer.fit_transform(title).toarray()
        # 构建二维语义空间
        svd = TruncatedSVD(n_components=2)
        svd.fit(corpus_tfidf)

Out[3]:TruncatedSVD(algorithm='randomized', n_components=2, n_iter=5,
                     random_state=None, tol=0.0)

In [4]:svd.fit_transform(corpus_tfidf)
Out[4]:array([[ 0.53998204,  0.41174044],
               [ 0.614138 , -0.48422793],
               [ 0.23842791,  0.32792508],
               [ 0.61358726, -0.40925992],
               [ 0.39402097,  0.62935865]])
```


7.2 LDA

7.2.1 LDA 简介

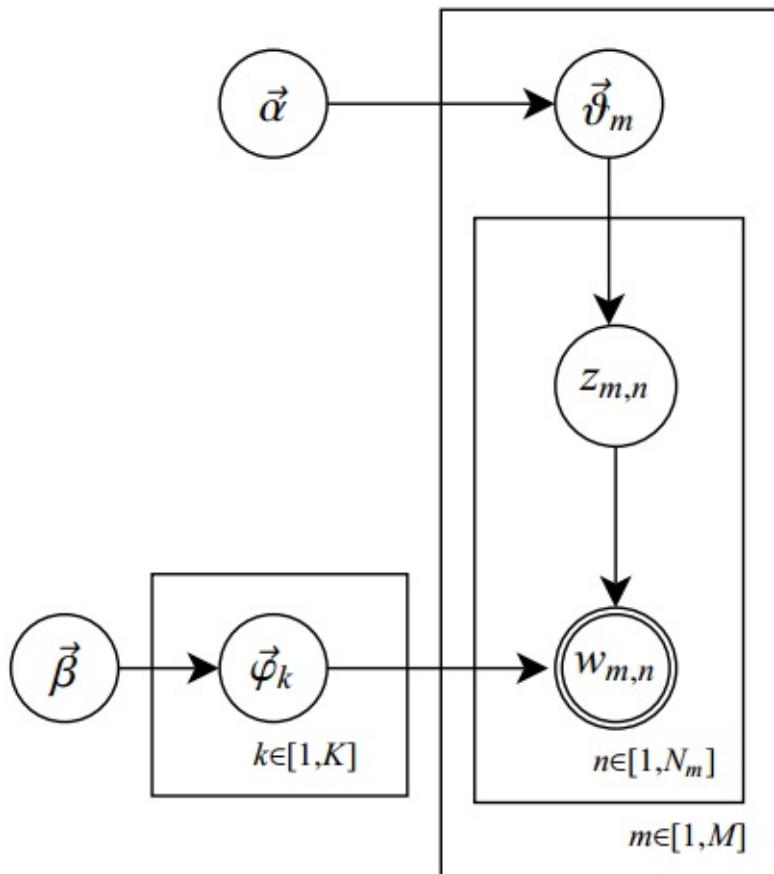
LDA (Latent Dirichlet Allocation) , 中文全称为隐含狄利克雷分布，最先由 Blei, David M. 等人于2003年提出，是一种以词、主题和文档三层贝叶斯概率为核心结构的主题模型。同时，LDA 也是一种无监督机器学习算法，我们在 LDA 模型训练前不需要进行任何手工标注。目前，LDA 在文本挖掘领域包括文本主题识别、文本分类以及文本相似度计算方面都有广泛应用。

在主题模型中，“主题”一词是一个相对抽象的概念，通常表现为一系列相关的词，以及这些词出现的条件概率。一个主题可以被理解为一个袋子，这个袋子有概率抖出一些词汇用于组成文档，我们可以通过这些词汇得知主题所要传达的语义。LDA 的核心目标便是告诉我们，在我们的语料集中一共出现了哪些主题，以及这些主题中每个词出现的概率是多少。

在 LDA 中，模型将文档生成的过程理解为先选定一个主题向量 θ ，并确定每个主题被选择的概率，然后在生成文档中每个单词的时候，从主题分布向量 θ 中选择一个主题 z ，按主题 z 的单词概率分布生成一个单词。对于一篇包含 N 个词汇的文档，LDA 模型的联合概率可表示为：

$$p(\theta, z, w | \alpha, \beta) = p(\theta | \alpha) \prod_{n=1}^N p(z_n | \theta) p(w_n | z_n, \beta)$$

其中， $p(\theta | \alpha)$ 为文档被选定主题向量 θ 的概率，服从 Dirichlet 分布； $p(z_n | \theta)$ 为从主题向量 θ 中选择主题 z_n 的概率，服从多项式分布； $p(w_n | z_n)$ 为从主题 z_n 中选择词汇 w_n 的概率，服从 Dirichlet 分布； α 和 β 则分别表示 LDA 中两个 Dirichlet 分布的分布参数，这两个参数属于语料级别，不会由于文档的不同而改变。



LDA 模型的生成过程也是对 Dirichlet 分布参数 α 和 β 的估计拟合过程，在实际应用中，计算机程序通常将通过 EM 算法来完成对 α 与 β 的参数值的估计，即在算法的 E-step 输入待定参数 α 和 β 的值，从而计算似然函数；之后在 M-step 最大化似然函数，解出对应的 α 和 β ，不断迭代直至收敛。

LDA 模型在解决文本数据分析问题时的优势表现在其合理的概率层次以及相对直观的结果表现，一般情况下，我们能够直接从主题中包含的词直接观察得主题的含义，从而更容易地找到隐藏在文本之中的语义关联。在下文中，我们将介绍一系列工具库，帮助大家实现 LDA 模型的构建与应用。

7.2.2 利用 gensim 库训练 LDA 主题模型

gensim 库 models 包中的 `Ldamodel` 模块中定义了 `LdaModel` 类及相关方法实现 LDA 主题模型，`LdaModel` 类的实例化方式为：`实例 = LdaModel(corpus=None, num_topics=100, id2word=None, distributed=False, chunksize=2000, passes=1, update_every=1, alpha='symmetric', eta=None, decay=0.5, offset=1.0, eval_every=10, iterations=50, gamma_threshold=0.001, minimum_probability=0.01, random_state=None, ns_conf={})`

部分参数说明：

`corpus`：用该参数传入的文档语料将会被用来训练模型，如果不指定该参数，则不进行任何训练，默认后续会调用 `update()` 方法对模型语料进行更新

`num_topics`：需要提取的潜在主题数

`id2word`：用于设置构建模型的词典，决定了词汇数量

`distributed`：是否开启分布式计算

`chunksize`：文件块大小

`alpha`：决定文档主题狄利克雷先验分布的超参数，默认取值为对称 $1.0/\text{num_topics}$ 先验，可以自行设置，也支持以下两种取值：（1）‘asymmetric’：固定的非对称 $1.0/\text{topicno}$ 先验

（2）‘auto’：根据实际数据学习得到的非对称先验 `eta`：决定主题词汇狄利克雷先验分布的超参数，可以自行设置为对称的先验分布常量或者长度为词汇总数的向量作为非对称先验，此外也支持以下两种取值：（1）‘auto’：根据实际数据学习得到的非对称先验

（2）形如 `num_topics x num_words` 的矩阵：为每一个主题都引入一个词汇非对称先验分布

`minimum_probability`：用于限制返回一个文档主题的概率

相关方法及属性如下：

（1）`get_document_topics` 方法

`get_document_topics` 方法以（主题 `id`，主题概率）的形式返回指定文档的主题分布，调用方式为：`get_document_topics(bow, minimum_probability=None, minimum_phi_value=None, per_word_topics=False)`

参数说明：

`bow`：指定分析的文档

`minimum_probability`：忽略概率小于该参数值的主题

`per_word_topics`：取值为 `True` 时，同时按照可能性的降序返回词汇对应的主题

（2）`get_term_topics` 方法

`get_term_topics` 方法用于返回词典中指定词汇最有可能对应的主题，调用方式为：实例`.get_term_topics(word_id, minimum_probability=None)`，其中参数 `word_id` 即为指定词汇 `id`，`minimum_probability` 为返回主题的最小概率限定

（3）`get_topic_terms` 方法

`get_topic_terms` 方法以（词汇 `id`，概率）的形式返回指定主题的重要词汇，调用方式为：`get_topic_terms(topicid, topn=10)`，参数 `topicid` 即为主题 `id`，`topn` 为返回的词汇数。

（4）`show_topic` 方法

`show_topic` 方法直接以（词汇，概率）的形式返回主题的重要词汇，调用方式为：`show_topic(topicid, topn=10)`，其中参数 `topicid` 即为主题 `id`，`topn` 为返回的词汇数。

(5) show_topics 方法

`show_topics` 方法可以同时返回多个主题的重要词汇，调用方式为：

`show_topics(num_topics=10, num_words=10, log=False, formatted=True)`，需要注意的是，和上一节介绍的 LSI 不同，该方法返回的主题并未按照概率自动排序，而是随机抽取的，所以多次调用该方法可能会得到不同的结果

参数说明：

`num_topics`：返回的主题数，默认取值为 10

`num_words`：每个主题返回的词汇数，默认取值为 10

`log`：是否将得到的结果输出到日志中

`formatted`：取值为 `True` 时以字符串形式返回词汇及相应概率，取值为 `False` 时返回词汇和相应概率组成的元组

(6) “[]” 方法

“[]” 方法可以返回文档在各个主题的分布，调用方式为：实例[`document`]

例 3 仍然以两类新闻标题为例，利用 LDA 模型将语料映射到二维主题空间

```
In [5]:from gensim.models.ldamodel import LdaModel
        lda = LdaModel(corpus=corpus_tfidf, id2word=dictionary, num_topics=2)
        lda.show_topics(2)
Out[5]:[(0,
          u'0.032*"new" + 0.032*"open" + 0.031*"disperse" + 0.031*"alert" + 0.031*"innovation" + 0.031*"set" + 0.031*"red" + 0.031*"beijing" + 0.031*"university" + 0.030*"lifets"),
        (1,
          u'0.034*"of" + 0.034*"source" + 0.034*"pollutant" + 0.033*"revealed" + 0.033*"major" + 0.032*"school" + 0.032*"players" + 0.031*"kung" + 0.030*"fu" + 0.030*"combin es")]
In [6]:lda.get_term_topics(1)
Out[6]:[(0, 0.014964645383614911)]
In [7]:for i in corpus_tfidf:
        print lda[i]
Out[7]:[(0, 0.82505037243004165), (1, 0.17494962756995844)]
[(0, 0.16708784121775896), (1, 0.83291215878224101)]
[(0, 0.80727070347450824), (1, 0.19272929652549178)]
[(0, 0.82151361100587073), (1, 0.1784863889941293)]
[(0, 0.17230199161117327), (1, 0.82769800838882668)]
```

7.2.3 利用 `sklearn` 构建 LDA 模型

sklearn 库 decomposition 包也提供了 `online_lda` 模块构建 LDA 模型，主要利用其下定义的类 `LatentDirichletAllocation` 通过变分贝叶斯算法实现这一过程（官方文档 <http://scikit-learn.org/dev/modules/decomposition.html#latentdirichletallocation>）。该类的初始化方式为：实例 = `LatentDirichletAllocation(n_topics=10, doc_topic_prior=None, topic_word_prior=None, learning_method=None, learning_decay=0.7, learning_offset=10.0, max_iter=10, batch_size=128, evaluate_every=-1, total_samples=1000000.0, perp_tol=0.1, mean_change_tol=0.001, max_doc_update_iter=100, n_jobs=1, verbose=0, random_state=None)`

部分参数说明：

`n_topics`：主题空间维度

`doc_topic_prior`：文档主题先验分布，对应文档中的参数 `alpha`，默认 `None` 值表示取值 $1 / n_{topics}$

`topic_word_prior`：主题词汇先验分布，对应文档中的参数 `eta`，默认 `None` 值表示取值 $1 / n_{topics}$

`learning_method`：模型的学习方法，取值为 ‘batch’ 或 ‘online’。一般情况下，当数据量比较大时，‘online’ 方法学习速度比较快。

`learning_decay`：该参数用于控制 ‘online’ 学习方法的学习率，取值范围应在 $(0.5, 1.0]$ 以保证渐进收敛。当该参数取值为 0 并且参数“`batch_size`”取值为 `n_samples` 时，等同于‘batch’学习方法，即文档中的 `kappa`。

`learning_offset`：该参数也会影响 ‘online’ 学习方法的学习率，会降低学习过程的迭代次数。

`max_iter`：最大迭代次数

`total_samples`：文档总数，仅当调用 `partial_fit` 方法时才需要设置

`batch_size`：‘online’ 学习方法每一次 EM 迭代的文档数

`evaluate_every`：在调用 `fit` 方法时设置困惑度（perplexity）评估的频率，困惑度一般在自然语言处理中用来衡量训练出的语言模型的好坏。困惑度评估一方面可以检查训练过程的收敛情况，但是也会增加训练时长，如果每次迭代都评估一次的话，训练时长可能会增加为原来的两倍。当该参数取值为 0 或负时，表示在训练过程中不评估困惑度，

`mean_change_tol`：EM 算法的 E 步中停止更新文档主题分布的临界值

`max_doc_update_iter`：EM 算法的 E 步中更新文档主题分布的最大迭代次数

相关方法及属性如下：

@补充方法属性列表

例 4 仍然以两类新闻标题为例，利用 LDA 模型将语料映射到二维主题空间

```
In [8]:from sklearn.decomposition import LatentDirichletAllocation  
       lda = LatentDirichletAllocation(n_topics=2)  
       # 利用 TfidfVectorizer 计算 TF-IDF 矩阵  
       vectorizer = TfidfVectorizer()  
       corpus_tfidf=vectorizer.fit_transform(title).toarray()  
       # 构建二维语义空间  
       lda = LatentDirichletAllocation(n_topics=2)  
       lda.fit_transform(corpus_tfidf)  
Out[8]:array([[ 3.29983659,  0.66251637],  
              [ 0.66252396,  3.15163069],  
              [ 0.62946368,  3.33981907],  
              [ 3.30959033,  0.65276262],  
              [ 3.26581911,  0.69936272]])
```

7.3 主题模型应用实例

本章的前两节介绍了利用 `gensim` 以及 `sklearn` 库构建主题模型的方法，并且通过简单的例子演示了各个类及方法的调用方式，本节将使用 `20 newsgroups` 新闻数据集中的实际数据，利用 `gensim` 库相关类和方法，测试主题模型对于计算文档相似度以及文本聚类的效果。

7.3.1 加载数据

依然使用 `sklearn` 内建函数 `fetch_20newsgroups` 加载 `20 newsgroups` 新闻数据

```
In [1]:import sklearn
       from sklearn.datasets import fetch_20newsgroups
       dataset = fetch_20newsgroups(shuffle=True, random_state=1, remove=('headers', 'footers', 'quotes'))
       dataset
Out[1]:{'DESCR':None,
        'data': [u"Well i'm not sure ..... It is unfortunate.\n",
                  u"\n\n\n\n\n\n\nYeah, ..... III",
                  ...],
        'description':'the 20 newsgroups by date dataset',
        'filenames':array(['.....'], dtype='|S97'),
        'target':array([17, 0, 17, ..., 9, 4, 9]),
        'target_names':['alt.atheism', 'comp.graphics', 'talk.politics.misc', 'talk.religion.misc']} # 新闻文本保存在键 "data" 对应的列表内，“target”对应新闻类型，“target_names”为不同新闻类型名称
In [2]:corpus=dataset["data"] # 抽取新闻文本
       len(corpus) # 查看新闻文本数
Out[2]:11314
```

7.3.2 文本数据预处理

(1) 分词、小写

使用 `NLTK` 分词函数 `word_tokenize` 对新闻文本进行分词处理

```
In [3]:import nltk
       from nltk import word_tokenize
       corpus_token=[[word.lower() for word in word_tokenize(sent)] for sent in corpus]
```

(2) 过滤停用词

直接使用 NLTK 提供的英文停用词词典

```
In [4]:from nltk.corpus import stopwords
english_stopwords = stopwords.words("english")
corpus_filter_stop=[[word for word in words if not word in english_stopwords] for words in corpus_token]
```

(3) 过滤标点符号

使用自定义英文表单符号列表进行标点过滤

```
In [5]:english_punctuations = [',', '.', ':', ';', '?', '(', ')', '[', ']', '!', '@', '#', '%', '$', '*']
corpus_filter_pun=[[word for word in words if not word in english_punctuations] for words in corpus_filter_stop]
```

(4) 词干化处理

使用 nltk lancaster 模块下定义的 LancasterStemmer 类进行词干化处理

```
In [6]:from nltk.stem.lancaster import LancasterStemmer
st = LancasterStemmer()
corpus_stem=[[st.stem(word) for word in words] for words in corpus_filter_pun]
# 查看部分预处理结果
corpus_stem[:2]
Out[6]:[[u'wel', u'm', u'sur', ..., u'got', u'pow', u'unfortun'],
[u'yeah', u'expect', u'peopl', ..., u'bak', u'timmon', u'ii']]
```

7.3.3 文本数据结构化处理

利用 gensim 库 TfidfModel 类计算 TF-IDF 矩阵

```
In [7]:from gensim import corpora
dictionary = corpora.Dictionary(corpus_stem)
text = [dictionary.doc2bow(words) for words in corpus_stem]
tfidf_model = models.TfidfModel(text)
text_tfidf = tfidf_model[text]
```

7.3.4 构建 LSI 模型，计算文本相似度

gensim 库 similarities 包的 docsim 模块定义了计算向量空间中文档集合相似度的类和函数，其中最重要的类就是 **Similarity**，该类可以为文档集合构建一个索引，只要指定一个文档，我们可以轻松的得知其余各个文档和该文档的相似程度。如果需要计算相似度的文档已经转化为 LSI 空间向量，需要用到的类是 **MatrixSimilarity**，该类初始化方式为：实例
=MatrixSimilarity(corpus, num_best=None, dtype=, num_features=None, chunksize=256, corpus_len=None)

部分参数说明：

corpus：需要计算相似度为文档特征数据

num_best：输出的最相似的文档个数，若不指定，则返回全部文档的相似度

num_features：为文档特征数，如果不指定的话默认为传入文档的特征数，一般为字典长度或者主题模型主题数

get_similarities 方法用于返回指定文档的与全部文档的相似度，调用方式为：实例.get_similarities(query)，参数 query 即为指定文档，也可同时传入多个文档，返回结果即为一个反映多个指定文档与全部文档相似度的二维表。但是 gensim 官方文档建议不要直接调用该方法，而是使用“实例[query]”的方式计算指定文档的相似度。

将上步计算得到的 TF-IDF 矩阵传入初始化的 MatrixSimilarity 类，构建文档相似度索引

```
In [8]:from gensim.similarities import MatrixSimilarity
        sim_index = MatrixSimilarity(text_lsi)
        # 输出第一篇文档与其他文档之间的相似度
        sim_index[text_lsi[0]]
Out[8]:array([ 1.          ,  0.1223305 ,  0.73435599, ...,  0.05217935,
              0.01776145,  0.16315971], dtype=float32)
In [9]:print list(enumerate(sim_index[text_lsi[0]])) # 利用 enumerate 返回以上结果的枚举形式，直观上看是返回了相似度和对应的文档序号
Out[9]:[(0, 1.0), (1, 0.1223305), (2, 0.73435599), ..., 0.052179348), (11312, 0.017761454), (11313, 0.16315971)]
In [10]:sort_sims = sorted(enumerate(sim_index[text_lsi[0]]), key=lambda item: item[1], reverse=True) # 利用 sorted 函数按照相似度从大到小排序
        print sort_sims[0:10] # 输出最为相似的前 10 个文档编号及相似度
Out[10]:[(0, 1.0), (5519, 0.82798922), (3580, 0.82728314), (1405, 0.82274032), (6148, 0.82230043), (3668, 0.81276625), (8016, 0.80825418), (7875, 0.80753094), (3706, 0.80649906), (2632, 0.80520225)]
In [11]:for j in [i[0] for i in sort_sims[0:10]]:
        print j, "\n", corpus[j] #输出原始文档内容，查看检索效果
Out[11]:0
        Well i'm not sure about the story nad it did seem biased. What I disagree with
is your statement that the U.S. Media is out to.....It is unfortunate.

        5519
        You are conveniently ommitting the fact that the Arab governments told the Ara
b citizens of Israel to leave Israel, join with the Arab armies so that.....you claim to
know so much about.

        Steve
        --
        ....
        ppear to be. When you realize that you can't care for other people while you
hate yourself you might actually begin to do some good.
```

查看以上输出的较为相似的 10 个文档内容，发现均有提及 Israels、Jews、Nazi 等相关政治、人权话题。

7.4 小结

本章介绍了 LSI 和 LDA 两个主题模型的基本原理和构建方式，涉及的 Python 库有 gensim 和 sklearn，对应的类和方法汇总如下：

@todo 补充汇总表

7.5 深度阅读材料

7.5.1 LSI

Fast and Faster: A Comparison of Two Streamed Matrix Decomposition Algorithms

https://nlp.fi.muni.cz/~xrehurek/nips/rehurek_nips.pdf

7.5.2 LDA

“Latent Dirichlet Allocation” D. Blei, A. Ng, M. Jordan, 2003

<https://www.cs.princeton.edu/~blei/papers/BleiNgJordan2003.pdf>

Online Learning for Latent Dirichlet Allocation

<https://www.cs.princeton.edu/~blei/papers/HoffmanBleiBach2010b.pdf>

“Stochastic Variational Inference” M. Hoffman, D. Blei, C. Wang, J. Paisley, 2013

<http://www.columbia.edu/~jwp2128/Papers/HoffmanBleiWangPaisley2013.pdf>

7.6 练习

7.6.1 在 7.3 节实例分析的基础上，进一步构建 **LDA** 模型及文档相似度索引，并利用新的文档向量训练分类器，与使用 **TF-IDF** 矩阵分类进行效果对比。

7.6.2 从公开数据源下载或者自行抓取待分析中文文本数据，根据数据实际情况进行分词、去除停用词、结构化等处理，进一步运用主题模型进行降维处理，以此为基础计算各个文档之间的相似程度。

第 8 章 文本数据可视化

大数据环境下，人们处理和理解海量信息的难度日益增大，传统的文本分析技术提取的信息已经无法满足人们的分析需求，将文本中复杂的或者难以通过文字表达的内容和规律以图表的形式表达出来，即文本数据的可视化处理，使人们能够快速获取大数据中所蕴含的关键信息。

文本可视化技术综合了文本数据分析、数据挖掘、数据可视化等学科的理论和方法，是人们理解复杂的文本内容、结构和内在的规律等信息的有效手段。其中，文本数据分析是文本数据可视化的基础，通过文本数据分析将文本数据结构化处理并提取出关键信息后才能进一步进行灵活、直观的可视化操作。截止到目前为止的章节，已经介绍了文本数据结构化处理及分析的基本方法，本章将在此基础上对这些整合过的信息进行可视化展示。

对于不同的文本数据分析结果，可以有不同的可视化方式，比如对分词的结果进行词频统计后可以绘制词频相关的柱状图或者词云图、对文本聚类或者分类的结果可以绘制矩形树图、对于各种文本数据分析结果还可以结合时间信息判断其变动趋势，本章根据文本数据可视化对象的不同将文本数据可视化任务分为文本数据内容可视化、文本数据关系可视化及基于时间信息的文本数据可视化三类，具体内容安排如下：8.1 节主要介绍文本数据内容的可视化，包括词频柱状图和词云图；8.2 节讨论文本数据主题模型可视化；8.3 节结合时间信息对文本数据进行可视化处理，反映词频及话题随时间的变动情况，

8.1 文本数据内容可视化

文本内容可视化旨在快速展示文本的大致内容，快速直观的为文本阅读者提取文本内容的重点，对进一步进行复杂文本数据分析具有指导意义。最常用的是基于词频的可视化，其基本原理是：将文本看成词汇的集合，统计词汇出现的次数用以表现文本特征，进而用相关统计图表对词频进行展示，柱状图和词云（标签云）是两种常见的文本数据内容可视化形式。

8.1.1 柱状图

8.1.1.1 使用 **matplotlib** 绘制柱状图

matplotlib 是基于 Python 的二维绘图库，其中的 **pyplot** 模块提供了可以直接调用的编程接口支持 MATLAB 形式的绘图框架，该模块下定义的 **bar** 函数可以实现柱状图的绘制，调用方式为：**bar(left, height, width=0.8, bottom=None, **kwargs)**

参数说明：

left：柱状图中每个矩形左起点横坐标，取值为数值序列或者标量

height：柱状图中每个矩形的高度，取值为数值序列或者标量

其他部分参数：

width：柱状图中每个矩形的宽度，默认取值为 0.8

bottom：柱状图中每个矩形底边纵坐标

color：柱状图填充颜色，取值为标量或者数组

edgecolor：柱状图框线颜色

linewidth：柱状图框线宽度，默认取值为 **None**，使用默认宽度，取值为 0 时，不绘制框线

align：设置横坐标对齐方式，有以下两种取值：

(1) "center"：横坐标位于柱状图每个矩形底边的中间位置 (2) "edge"：横坐标位于柱状图每个矩形底边的左侧

orientation：设置柱状图的方向，有以下两种取值：

(1) "vertical"：垂直方向 (2) "horizontal"：水平方向 也可以通过函数 **barh** 直接绘制水平方向柱状图，相关参数说明见官方文

档：http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.barh

label：图例标签

注意：**color**、**edgecolor**、**linewidth** 这些参数取值既可以常数也可以是长度为柱状图矩形个数的序列，取值为常数时，表示所有矩形参数取值一致，取值为序列时，可以为每个矩形设置不同的参数取值。

此外，在绘图过程中，为了增强图像的可读性、美观性，经常需要进行一些细节调整，可以利用 **pyplot** 模块下定义的这些函数：

@todo 补充表格

例 1

```
In [1]:import matplotlib.pyplot as plt
%matplotlib inline # 执行该命令后，绘图时，可将图片嵌入 notebook 交互框，而不是单独弹出一个图片窗口
import numpy as np

# 绘制柱状图

In [2]:fre_1 = (120, 95, 230, 35, 27)
        fre_2 = (213, 130, 324, 100, 59)
        index = np.arange(5)
        plt.bar(index, fre_1, width=0.3, color="g", align="center", label="fre_1")
        plt.bar(index+0.3, fre_2, width=0.3, color="r", align="center", label="fre_2")
        plt.xlabel("category")
        plt.ylabel("fre")
        plt.text(2, 330, "highest")
        plt.legend()
        plt.title("barplot")
Out[2]:@todo 补充图片 9.1.1.1
```

8.1.1.2 使用 Pandas 绘制柱状图

对于 Pandas 库的基本数据结构 Series 和 DataFrame，可以直接调用其对应的 plot 方法绘制所需图像，本部分对数据结构并未做详细说明，而是重点讲解 Series 和 DataFrame 两种数据结构的柱状图的绘制，关于数据结构介绍读者可以阅读官方文档

(<http://pandas.pydata.org/pandas-docs/stable/dsintro.html>)。

(1) 绘制 Series 的柱状图

Series 可以用于存储任何类型的一维数据，其中的每一个数据都有自己的索引，构建 Series 使用 series 模块下的 Series 类即可实现，构建方式为：Series(data, index=index)

参数说明：

data：用于构建 Series 的数据，可以是列表、元组、数组、字典、标量

index：索引列表/数组或者 Index 对象，长度与参数 **data** 相同，取值必须唯一，如果不指定该参数，则默认赋予从 0 开始的索引

例 2

```
In [3]: data=(0,1,2)
       index=("a","b","c")
       pd.Series(data,index)
Out[3]: a    0
        b    1
        c    2
       dtype: int64
In [4]: pd.Series(data)
Out[4]: 0    0
        1    1
        2    2
       dtype: int64
```

如果 `data` 为字典类型，不指定 `index` 参数时，则直接以字典中的键为索引构建 `Series`，如果指定 `index` 参数，则以 `index` 参数为基准对应字典键抽取相应的字典值，没有对应的参数则赋值为 `NaN`，具体可参见下例：

例 3

```
In [5]: pd.Series({"x":1,"y":2,"z":3})
Out[5]: x    1
        y    2
        z    3
       dtype: int64
In [6]: pd.Series({"x":1,"y":2,"z":3},index=["a","b","x","z"])
Out[6]: a    NaN
        b    NaN
        x    1.0
        z    3.0
       dtype: float64
```

`data` 参数取值为常数会出现什么情况呢？读者可以尝试生成，查看结果。

`Series` 中数据的索引方式与数组和字典类似，具体可以参见相关文档（<http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing>），本书在此不做过多说明。

在文本数据分析过程中，可以将分词、词频统计等结果统一保存为 `Series` 结构，对需要绘图的数据直接调用 `plot` 方法调用 `matplotlib` API 进行绘图，`plot` 方法的调用方式为：

```
Series.plot(kind='line', ax=None, figsize=None, use_index=True, title=None, grid=None,
legend=False, style=None, logx=False, logy=False, loglog=False, xticks=None,
yticks=None, xlim=None, ylim=None, rot=None, fontsize=None, colormap=None,
table=False, yerr=None, xerr=None, label=None, secondary_y=False, **kwds)
```

部分参数说明：

kind：绘制的图像类型，所支持的图像类型及对应的字符串取值如下：
-'line'：折线图（默认取值）

-'bar'：垂直方向柱状图

-'barh'：水平方向柱状图

-'hist'：直方图

-'box'：箱线图

-'kde'：概率密度估计图

-'density'：与'kde'相同

-'area'：堆叠面积图

-'pie'：饼图

figsize: 图片大小，取值为形如 (width, height) 的数组，分别表示图片的宽和高，单位为英寸

use_index：是否使用 index 取值作为图像横坐标，默认取值为 True

title：图片的名称

grid：是否在图片背景添加坐标网格线，默认取值为 False

例 4

```
In [7]: s = pd.Series(data=[13, 10, 14, 9], index=["a", "b", "c", "d"])
s.plot(kind="bar", title="bar plot", grid=True)
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6efaec73d0>
@todo 补充截图 9.1.1.2-1
```

(2) 绘制 DataFrame 的柱状图

DataFrame 是包含多列数据的二维数据结构，类似 excel 表格。构建 DataFrame 使用 frame 模块下的 DataFrame 类即可实现，构建方式为：DataFrame(data, index, columns)

参数说明：

data：用于构建 DataFrame 的数据，支持多种数据结构，可以是一维数组、列表、字典、Series 的字典（即命名了列名的一维数据，数据类型可以是数组、列表、字典、Series），或者二维数组，甚至是 DataFrame

index：行标签

columns：列标签

与 Series 类似，如果 data 为字典类型，不指定 index 和 columns 参数时，则直接以原有列标签和字典中的键为行列标签构建 DataFrame，如果指定 index 和 columns 参数，则对应抽取相应的值构成 DataFrame，没有对应的参数则赋值为 NaN，若不专门设定行标签，则默认赋予从 0 开始，具体可参见下例：

例 5

```
In [8]: s_1 = pd.Series(data=[1, 2, 3], index=["a", "b", "c"])
      s_2 = pd.Series(data=[4, 5, 6, 7], index=["a", "b", "c", "d"])
      pd.DataFrame({"x":s_1, "y":s_2})
Out[8]:@todo 补充图片 9.1.1.2-2
In [9]:pd.DataFrame({"x":s_1, "y":s_2}, index=["b", "c", "d"])
Out[9]:@todo 补充图片 9.1.1.2-3
In [10]:pd.DataFrame({"x":s_1, "y":s_2}, index=["b", "c", "d"], columns=["x", "z"])
Out[10]:@todo 补充图片 9.1.1.2-4
```

除了将文本数据分析的结果统一保存为 Series，也可以将多个结果汇总为 DataFrame 结构，筛选需要绘图的数据并调用 plot 方法绘图即可，plot 方法的调用方式为：

`DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False, sharex=None, sharey=False, layout=None, figsize=None, use_index=True, title=None, grid=None, legend=True, style=None, logx=False, logy=False, loglog=False, xticks=None, yticks=None, xlim=None, ylim=None, rot=None, fontsize=None, colormap=None, table=False, yerr=None, xerr=None, secondary_y=False, sort_columns=False, **kwds)`

大部分参数与 Series 相同，部分参数区别如下：

在使用数据框中某两列绘制图像时，使用 `x`、`y` 参数指定图形横纵坐标，其中：`x` 为横坐标数据的列标签，`y` 为纵坐标数据的列标签

`kind`：除了上面提到几种图像类型，DataFrame 还支持：

- `'scatter'`：散点图
- `'hexbin'`：高密度散点图

`subplots`：是否为数据框中的每一列单独绘图，默认取值为 `False`

`sharex/sharey`：每一列单独绘图时，各个子图是否使用相同的横坐标/纵坐标

`layout`：输出子图的排列格式，取值类型为 `(rows, columns)` 的数组，其中 `rows` 为子图行数，`columns` 为子图列数

例 6

```
In [11]:s_1 = pd.Series(data=[1,2,3,4],index=["a","b","c","d"])
    s_2 = pd.Series(data=[4,5,6,7],index=["a","b","c","d"])
    df=pd.DataFrame({"x":s_1,"y":s_2})
    df.plot(kind="bar")
Out[11]:@todo 补充图片 9.1.1.2-5
In [12]:df.plot(kind="bar", subplots=True)
Out[12]:todo 补充图片 9.1.1.2-6
In [13]:df.plot(kind="bar", subplots=True, sharex=False, layout=(1,2))
Out[13]:todo 补充图片 9.1.1.2-7
In [14]:df["x"].plot(kind="bar")
Out[14]:todo 补充图片 9.1.1.2-8
```

(3) Series 文本数据处理方法

Series 支持一系列的方法，其中包括字符串处理方法 **str**，类似 Python 字符串处理方法，下面介绍几种常用方法：

(3.1) cat 方法

cat 方法可以使用指定的分隔符号将 **Series** 中字符串进行连接，调用方式为：

```
Series.str.cat(others=None, sep=None, na_rep=None)
```

参数说明：

others：有两种取值类型：

(1) **None**：默认取值，直接返回 **Series** 中字符串连接后形成的字符串 (2) 字符串列表：
将列表中的字符串与 **Series** 元素按照顺序对应连接，返回一个字符串 **Series**

sep：用于连接字符串的分隔符，取值类型为字符串，默认取值为 **None**

na_rep：有两种取值类型：

(1) **None**：默认取值，表示忽略 NA 值 (2) 字符串：使用指定的字符串替换 NA 值

例 7

```
In [15]:pd.Series(['text','mining',np.nan,'python']).str.cat(sep=' ')
Out[15]:'text mining python'
In [16]:pd.Series(['text','mining',np.nan,'python']).str.cat(sep=' ',na_rep="?")
Out[16]:'text mining ? python'
In [17]:pd.Series(['text','with']).str.cat(others=['mining','python'],sep=',')
Out[17]:0      text,mining
            1      with,python
            dtype: object
```

(3.2) count 方法

count 方法用于统计某个字符串在 Series 各个字符串中出现的次数，调用方式为：

Series.str.count(pat, flags=0, **kwargs)

参数说明：

pat：需要统计出现次数的字符串或者正则表达式

flags：当参数 pat 为正则表达式时，该参数用于设置正则表达式的匹配模式

例 8

```
In [18]:pd.Series(['text','mining','with','python']).str.count('t')
Out[18]:0    2
           1    0
           2    1
           3    1
dtype: int64
```

(3.3) endswith 方法

当需要判断 Series 中字符串是否以某个字符串结尾时，可以调用 endswith 方法，调用方式为：Series.str.endswith(pat, na=nan)

参数说明：

pat：需要判断的末尾字符串

na：Series 中缺失值的返回形式，默认返回 nan

例 9

```
In [19]:pd.Series(['text','mining',np.nan,'python']).str.endswith ("ing")
Out[19]:0    False
           1    True
           2    NaN
           3    False
dtype: object
```

(3.4) find 方法

find 方法可以返回指定字符串在 Series 各个字符串的起始位置，若不包含指定字符串则返回 -1，调用方式为：Series.str.find(sub, start=0, end=None)

参数说明：

sub：需要搜索位置的指定字符串

`start`：从字符串左侧开始索引

`end`：从字符串右侧开始索引

例 10

```
In [20]:pd.Series(['text','mining',np.nan,'python']).str.find ("t")
Out[20]:0      0.0
           1    -1.0
           2      NaN
           3     2.0
dtype: float64
```

其他类似 Python 字符串处理方法可参加以下表格：

@补充表格

8.1.2 词云图

词云图将关键词按照一定的顺序和规则排列，以文字的大小代表词语的重要性，直观、快速地展示重要文本信息。

`wordcloud` 是较为常用的 Python 词云绘制包，利用其定义的 `WordCloud` 类即可实现词云图的绘制，此外还可以通过 `ImageColorGenerator` 类使用指定的图片定义词云图形形状和颜色，下面将对这两个类的初始化方式及相关方法作重点介绍。

8.1.2.1 WordCloud 类

`WordCloud` 类的初始化方式为：`实例 = WordCloud(font_path=None, width=400, height=200, margin=5, ranks_only=False, prefer_horizontal=0.9, mask=None, max_words=200, stopwords=None, random_state=None, background_color='black', max_font_size=None)`

参数说明：

`font_path`：词云图使用的字体路径

`width`：画布宽度

`height`：画布高度

`margin`：画布边缘空白宽度

`ranks_only`：是否按照词汇的排序进行绘图，而不是词频，默认取值为 `False`

`prefer_horizontal`：词云图中词汇横向呈现与纵向呈现的比率

mask：用于绘制词云图的背景图片，取值类型为数组（`scipy` 的 `imread` 函数可以将图片转换为数组形式，后续会详细说明），若指定该参数，将忽略词云图原有宽度和高度设置，而直接使用图片形状，并且不会在白的背景位置绘制词汇

max_words：词云图中显示的最大词汇数

stopwords：不在词云图中显示的词汇集合

background_color：词云图背景颜色，默认取值为“black”

max_font_size：最大的字体大小，默认取值为 `None`，使用图片高度作为限制

`WordCloud` 类方法：

(1) `fit_words`

调用 `fit_words` 方法（或者 `generate_from_frequencies` 方法）可以根据词汇词频绘制词云图，调用方式为：实例.`fit_words(frequencies)`，其中参数 `frequencies` 即为词频，其取值类型为列表，列表元素为包含词汇和相应词频的元组

例 11

```
In [21]:!pip install wordcloud # 安装 wordcloud 包
In [22]:import wordcloud
        cloud = wordcloud.WordCloud()
        frequencies = [("text",10),("mining",25),("with",2),("python",11)]
        cloud.fit_words(frequencies)
Out[22]:<wordcloud.wordcloud.WordCloud at 0x7f0ba2bea990>
In [23]:plt.imshow(cloud) # 在坐标轴上显示图像
        plt.axis("off") # 去除图像坐标轴
        plt.show() # 显示词云图
@todo 补充截图 9.1.2.1-1
```

(2) `generate`

`generate` 方法（或者 `generate_from_text` 方法）可以直接将文本数据展示为词云图，不必事先计算词频，调用方式为：实例.`generate(text)`，参数 `text` 即为需要可视化展示的文本内容

例 12

```
In [24]:text="text mining with python , text visualization "
        cloud.generate(text)
Out[24]:<wordcloud.wordcloud.WordCloud at 0x7f0ba2bea990>
In [25]:plt.imshow(cloud)
        plt.axis("off")
        plt.show()
@todo 补充截图 9.1.2.1-2
```

(3) process_text

process_text 方法可以对文本进行分词处理，并自动过滤停用词，调用方式为：实例.
process_text(text)，其中参数 text 即为需要分词的文本，该方法可以直接返回分词结果以及
对应的词频

例 13

```
In [26]:cloud.process_text("Good muffins cost $3.88\nin New York. Please buy me two o
f them.\nThanks.")
Out[26]:[('Good', 1),
 ('Please', 1),
 ('two', 1),
 ('cost', 1),
 ('York', 1),
 ('muffins', 1),
 ('New', 1),
 ('buy', 1),
 ('Thanks', 1)]
```

(4) recolor

recolor 方法用于改变当前词云图的颜色，直接调用该方法比重新绘制词云图更快，调用方式
为：实例.recolor(random_state=None, color_func=None)

参数说明：

random_state：随机配色方案种子，取值类型为整数，默认取值为 None

color_func：根据词频、字体等生成新配色的函数，若不设置，则默认使用 self.color_func

例 14

```
In [27]:cloud.recolor()
plt.imshow(cloud)
plt.axis("off")
plt.show()
@todo 补充截图 9.1.2.1-3

In [28]:cloud.recolor(random_state=2)
plt.imshow(cloud)
plt.axis("off")
plt.show()
@todo 补充截图 9.1.2.1-4
```

8.1.2.2 ImageColorGenerator 类

除了在绘图过程中选择不同的配色方案种子，也可以通过 `ImageColorGenerator` 类根据图片生成配色方案，词云图中的文字颜色由其所在图片位置的颜色决定，调用方式为：实例 = `ImageColorGenerator(image)`，其中参数 `image` 为图片的数组形式。该实例可以当做配色函数传入 `color_func` 参数。

例 15

将配色图片（如下所示）上传到 `notebook` 文件夹，命名为“`mask.jpg`”，利用该图片对词云图进行配色调整

@补充图片 9.1.2.1-5

```
In [29]:from scipy.misc import imread  
  
# 使用 imread 函数将图片转换为数组形式  
mask = imread("mask.jpg")  
  
# 将图片设置为词云图背景图片  
cloud = wordcloud.WordCloud(background_color="white", mask=mask)  
  
# 使用 wikipedia "Text mining" 检索结果作为语料，绘制词云图  
text="Text mining, ..... the information extracted."  
cloud.generate(text)  
plt.imshow(cloud)  
plt.axis("off")  
plt.show()
```

@补充图片 9.1.2.1-6

```
In [30]:image_colors = wordcloud.ImageColorGenerator(mask)  
  
# 使用图片颜色对词云图进行调色  
plt.imshow(cloud.recolor(color_func=image_colors))  
plt.axis("off")  
plt.show()
```

@补充图片 9.1.2.1-7

8.2 文本数据主题模型可视化

pyLDAvis 是主题模型交互可视化的 Python 包，移植自 Carson Sievert 和 Kenny Shirley 写的 LDAvis R 包（<https://github.com/cpsievert/LDAvis>），利用该包可以在 Jupyter Notebook 中轻松实现主题模型的可视化。

安装方式如下：

```
!pip install pyldavis
```

8.2.1 gensim 模块

pyLDAvis 包下定义了 gensim 模块，通过 gensim 库构建的主题模型利用该模块下定义的 `prepare` 函数即可实现可视化，调用方式如下：`prepare(topic_model, corpus, dictionary, doc_topic_dist=None)`

参数说明：

`topic_model`：训练得到的 gensim LdaModel 对象，不支持其他 gensim 模型类型

`corpus`：以词袋形式表示的语料（用于训练主题模型的语料）

`dictionary`：用于构建词袋模型的字典，即 gensim Dictionary 对象

`doc_topic_dist`：可选参数，用于传入 LDA 模型的文档主题分布，默认取值为 `None`，当需要多次调用 `prepare` 函数时，可以传入该参数

第八章已经介绍过利用 gensim 构建 LDA 模型的方式，本节不做过多说明，下例构建一个简单的 LDA 模型，利用该模型演示可视化函数调用

例 1

```
In [1]:import pyLDAvis
    import pyLDAvis.gensim
    # 导入构建 LDA 模型所需的相关工具
    from gensim import corpora,models
    from gensim.models.ldamodel import LdaModel
    # 读取文本数据
    title = ['Beijing lifts red alert with smog set to disperse',
              'Source of major pollutant in China smog revealed',
              'Xi stresses clean energy use to reduce smoggy days',
              'New Zealand university to open innovation center in China',
              'School combines soccer with kung fu to train players',
              'NASA bets big on private sector to put humans on Mars',
              'Why A Tornado-Damaged Facility In New Orleans Is Critical To NASA',
              'NASA Plans to Send This Robot Lander to Look for Alien Life on Europa'
    ]
    # 简单分词处理
    texts = [[word for word in document.lower().split()] for document in title]
    # 构建词典
    dictionary = corpora.Dictionary(texts)
    # 得到词汇词频
    corpus = [dictionary.doc2bow(text) for text in texts]
    # 构建 TF-IDF 矩阵
    tfidf_model = models.TfidfModel(corpus)
    corpus_tfidf = tfidf_model[corpus]
    # 构建 LDA 模型
    lda = LdaModel(corpus=corpus_tfidf,id2word=dictionary,num_topics=3)
    # 输出各个主题的词汇分布
    lda.show_topics(3)
Out[1]:[(0,
          u'0.024*"innovation" + 0.024*"center" + 0.023*"university" + 0.023*"zealand"
+ 0.023*"open" + 0.023*"beijing" + 0.023*"disperse" + 0.022*"lifts" + 0.022*"red" + 0.
022*"alert"),
         (1,
          u'0.028*"on" + 0.025*"smoggy" + 0.025*"days" + 0.025*"reduce" + 0.025*"stress
es" + 0.025*"clean" + 0.024*"use" + 0.024*"energy" + 0.024*"xi" + 0.024*"bets"),
         (2,
          u'0.024*"of" + 0.024*"pollutant" + 0.024*"major" + 0.024*"revealed" + 0.024*"
source" + 0.023*"in" + 0.023*"orleans" + 0.023*"why" + 0.023*"is" + 0.023*"a")]

In [2]: # 可视化主题模型
    vis_data = pyLDAvis.gensim.prepare(lda, corpus, dictionary)
    # 在 notebook 中显示可视化结果，需要调用 display 方法，或者执行 "pyLDAvis.enable_notebook()"，即可在 notebook 中自动展示可视化结果，无需再调用 display
    pyLDAvis.display(vis_data)
Out[2]:@todo 补充截图 9.2.1-1
```

主题模型的可视化结果给出了包括所有主题在内的全局视图，可以看到，输出结果分为左右两部分，左侧为“主题距离地图”，展示各个主题之间的差异，图中带有数字编号的圆形即代表各个主题，圆形的面积与该主题出现的可能性成正比，并且按照面积大小自动进行编号，右

侧为各个主题前 30 个最为相关的词汇，对各个主题进行解释说明，以水平柱状图的形式展示，蓝色表示整体词频，红色表示主题词频，当将鼠标光标移至某个主题圆形上方时，右侧将会显示该主题对应的词汇，也可以在左上角“Selected Topic”输入框中输入主题编号得到同样的效果。

@todo 补充截图 9.2.1-2 此外，某个词语与主题的相关性，可以通过右上方的 λ 参数来调节， λ 越接近 1，那么在该主题下出现越频繁的词语，跟主题越相关。想更多的了解算法及可视化系统等相关内容，可以参考深度阅读材料 8.5.1。

8.2.2 sklearn

和 gensim 模块一样，pyLDAvis 包下也定义了 sklearn 模块，来可视化通过 sklearn 库构建的 LDA 模型，同样利用该模块下定义的 prepare 函数实现可视化，调用方式如下：

```
prepare(lda_model, dtm, vectorizer, **kwargs)
```

参数说明：

lda_model：利用参数“dtm”构建的 LDA 模型

dtm：用于构建 LDA 模型的矩阵，可以是 dtm 矩阵，也可以是 tf-idf 矩阵

vectorizer：将原始语料转化为参数“dtm”的对象，即 sklearn.feature_extraction.text 下定义的 CountVectorizer 或 TfidfVectorizer 实例

例 2

```
In [3]:import pyLDAvis.sklearn
# 在 notebook 中自动展示可视化结果
pyLDAvis.enable_notebook()
# 导入构建 LDA 模型所需的相关工具
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
# 利用 CountVectorizer 计算 dtm 矩阵
vectorizer = CountVectorizer()
corpus_dtm=vectorizer.fit_transform(title)
# 构建三维语义空间
lda = LatentDirichletAllocation(n_topics=3)
lda.fit_transform(corpus_dtm)

Out[3]:array([[ 0.34782339,  0.35310333,  9.29907328],
   [ 0.3410852 ,  8.30516941,  0.35374539],
   [ 9.31370592,  0.3427256 ,  0.34356848],
   [ 0.34707267,  9.29574632,  0.35718102],
   [ 0.34862886,  9.29841902,  0.35295212],
   [ 0.35934982,  11.29026163,  0.35038855],
   [ 0.35032286,  0.35805016,  11.29162699],
   [ 14.29856804,  0.35289431,  0.34853765]]))

In [4]: # 可视化主题模型
pyLDAvis.sklearn.prepare(lda, corpus_dtm, vectorizer)

Out[4]:@todo 补充截图 9.2.1-3
```

8.3 基于时间信息的文本数据可视化

除了单纯的文本信息外，结合其他方面的信息对文本数据进行考量可以更加全面深入的理解文本数据，时间信息就是不容忽视的一种，在邮件、新闻、电子商务平台顾客评论等文本信息中几乎都会涉及到相关的时间信息，结合时间信息可以了解文本内容的变化规律，包含时间信息的文本数据可视化近年来也受到越来越多的关注。

本节仍以 4.1 节使用的亚马逊中国站上热门商品“Kindle”的评论语料作为分析实例，语料已经被保存为 CSV 文件“kindle_corpus.csv”，并上传到 Jupyter Notebook 文件列表。新建一个 notebook，读取文本数据，本节使用 pandas 库定义的 `read_csv` 函数直接将相关数据读取为 `dataframe` 格式。

```
In [1]:import pandas as pd
       df_reviews = pd.read_csv('kindle_corpus.csv')
       df_reviews.head()
Out[1]:@todo 补充截图 9.3.1-1
```

可以看到函数 `read_csv` 将 CSV 文件直接读取为 `dataframe` 格式，该函数定义在 `pandas.io.parsers` 模块内，调用时有诸多参数可供调节，最简单的调用方式为：
`read_csv(filepath_or_buffer, names=None, usecols=None)`

参数说明：

`filepath_or_buffer`：待读取的文件路径或者支持 `read` 方法的对象

`names`：列名称列表，如果文件不包含列名，使用该参数进行命名

`usecols`：需要读取的列名称列表，当只需读取部分列时，使用该参数进行选择，列表中包含的列名称（或者列编号）将会被读取

使用内置函数 `min` 和 `max` 返回时间列的最小值和最大值，查看评论文本的时间跨度

```
In [2]:[min(df_reviews.pubdate),max(df_reviews.pubdate)]
Out[2]:['2014-10-03', '2015-08-24']
```

8.3.1 热图

热图可以用颜色变化来反映二维矩阵或表格中的数据信息，直观地将数据值的大小以定义的颜色深浅表示出来，时间是常用的维度信息。本节我们使用数据可视化库 `seaborn` 中 `matrix` 模块下的函数 `heatmap` 进行热图的绘制，该函数可以将二维数据矩阵对应转化为热图，调用

方式为：`heatmap(data, vmin=None, vmax=None, annot=None, fmt='%.2g', linewidths=0, linecolor='white', xticklabels=True, yticklabels=True)`

部分参数说明：

data：用于绘制热图的二维数据矩阵，取值类型应为可以转为 n 维数组的二维数据集，如果传入的是 `dataframe` 结构的数据，则直接使用 `dataframe` 的行列信息作为热图的坐标信息

vmin/vmax：用于锚定热图的颜色，若不进行设置，则根据实际数据和其他参数自行选择颜色

annot：是否给热图添加数值注释，有以下两种取值方式：

- (1) 布尔型：若取值为 `True`，则用参数 `data` 中的数据对热图进行数值注释
- (2) 二维数据矩阵：和参数 `data` 相同维度的矩阵，用该矩阵中的数值对热图进行数值注释

fmt：传入 `annot` 参数进行数值注释时，利用该参数设置注释的字符串格式，可选值及含义可参考 <https://docs.python.org/2/library/string.html#format-specification-mini-language>

linewidths：划分各个单元格的线条的宽度

linecolor：划分各个单元格的线条的颜色

xticklabels：取值为 `True` 时，热图上将对应显示数据框列名，或者自行指定名称列表

yticklabels：取值为 `True` 时，热图上将对应显示数据框行名，或者自行指定名称列表

9.3.1.1 评论数热图

在使用 `heatmap` 绘图之前，需要将数据转换为相应的二维矩阵，如果需要绘制不同年度各个月份的评论数热图，就需要在原始数据的基础上计算不同年度各个月份的累计评论数，为了便于计算，先将原时间数据由“年-月-日”拆分为“年”、“月”、“日”三列，并添加评论数计数列：

```
In [3]:# 创建空列表用于存放拆分得到的年、月、日数据
year=[]
month=[]
day=[]
for i in df_reviews.pubdate:
    year.append(i.split("-")[0]) # 拆分出年度数据
    month.append(i.split("-")[1]) # 拆分出月度数据
    day.append(i.split("-")[2]) # 拆分出日数据
# 将拆分得到的数据保存到原数据框中
df_reviews["year"]=year
df_reviews["month"]=month
df_reviews["day"]=day
# 创建评论计数列，每条评论对应的都是"1"
df_reviews["count"]=len(df_reviews)*[1]
查看新的数据框
df_reviews.head()
Out[3]:@todo 补充截图 9.3.1-2
```

得到以上新数据框后，可以通过创建数据透视表的方式得到所需的二维矩阵，pandas 库的函数 `pivot_table` 即可实现这一需求，在调用该函数之前，先做一个简单的介绍：

`pivot_table` 函数位于 `pandas.tools.pivot` 模块，可以为数据框创建数据透视表，调用方式为：
`pivot_table(data, values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All')`

部分参数说明：

`data`：用于创建数据透视表的数据框

`values`：可选参数，指定用于聚合的数据列

`index`：数据透视表的行

`columns`：数据透视表的列

`aggfunc`：数据聚合函数，默认取值为`'mean'`，即求均值

`fill_value`：缺失值的替换值

下面以数据框中“`year`”和“`month`”为数据透视表的两个维度，评论计数“`count`”为汇总对象，构造数据透视表 `dataset_ym`：

```
In [4]:dataset_ym=pd.pivot_table(df_reviews, values = 'count', index = 'year', columns = 'month',aggfunc='sum')
dataset_ym
Out[4]:@todo 补充截图 9.3.1-3
```

可以看到，得到的数据透视表清晰的展示了不同年份的各个月份的累计评论数，利用该数据透视表就可以直接绘制相应的热图了：

```
In [5]:%matplotlib inline
    import seaborn as sea
    sea.heatmap(dataset_ym,linewidths=.5)
Out[5]:<matplotlib.axes._subplots.AxesSubplot at 0x7fd04a2ed990>
@todo 补充截图 9.3.1-4
In [6]:sea.heatmap(dataset_ym,annot=True,fmt="n",linewidths=.5)
Out[6]:<matplotlib.axes._subplots.AxesSubplot at 0x7fd049580b90>
@todo 补充截图 9.3.1-5
```

下面以数据框中“month”和“day”为数据透视表的两个维度，评论计数“count”为汇总对象，构造数据透视表 `dataset_md`，并绘制热图：

```
In [7]:dataset_md=pd.pivot_table(df_reviews, values = 'count', index = 'month', columns = 'day',aggfunc='sum')
    dataset_md
Out[7]:@todo 补充截图 9.3.1-6
In [8]:sea.heatmap(dataset_md,linewidths=.5)
Out[8]:<matplotlib.axes._subplots.AxesSubplot at 0x7fd0480fe110>
@todo 补充截图 9.3.1-7
```

从热图可以看出，7月初和12月中旬累计评论量显著高于其他时间段。

9.3.1.2 热门词汇出现次数热图

除了统计不同时间的累计评论数，还可以计算不同时间热门词汇出现的次数，下面对评论文本进行简单的分词处理，并统计热门词汇“闪屏”出现的次数：

```
In [9]:# 简单分词处理，将分词结果保存在数据框 "words" 列
    import jieba
    df_reviews["words"]=[list(jieba.cut(i)) for i in df_reviews["content"]]
    # 统计每条评论热门词汇"闪屏"出现次数，并将统计结果保存在数据框 "word_count" 列
    df_reviews["word_count"]=[i.count(u"闪屏") for i in df_reviews["words"]]
    # 生成数据透视表 dataset_md_words
    dataset_md_words=pd.pivot_table(df_reviews, values = 'word_count', index = 'month', columns = 'day',aggfunc='sum')
    dataset_md_words
Out[9]:@todo 补充截图 9.3.1-8
In [10]:sea.heatmap(dataset_md_words,linewidths=.5)
Out[10]:<matplotlib.axes._subplots.AxesSubplot at 0x7fd042bd0890>
@todo 补充截图 9.3.1-9
```

8.3.2 折线图

8.3.2.1 累计计数趋势图

在绘制趋势图之前需要对数据框中的表示时间的列“pubdate”进行类型转换，可以查看一下，原始的时间数据被存储为字符串：

```
In [11]: type(df_reviews["pubdate"][0])
Out[11]: str
```

为了绘制时间趋势图，我们需要将其转换为时间类型，这里用到的是 `datetime` 时间处理模块中的 `strptime` 函数，调用方式如下：

```
In [12]: from datetime import datetime
        print(datetime.strptime(df_reviews["pubdate"][0], "%Y-%m-%d"))

2015-07-11 00:00:00
```

可以看到，函数 `strptime` 有两个需要传入的参数，一是待格式转换的字符串，一是需要转换成的时间格式，下面就利用该函数对“pubdate”列所有的时间字符串进行格式转换，并保存到新列 `time` 中：

```
In [13]: df_reviews["time"]=[datetime.strptime(i, "%Y-%m-%d") for i in df_reviews["pubdate"]]
        df_reviews.time.head()

Out[13]: 0    2015-07-11
          1    2015-07-08
          2    2015-07-01
          3    2015-07-12
          4    2015-08-05
         Name: time, dtype: datetime64[ns]
```

想要绘制不同时点累计评论数，有一种方式就是先对不同时点的评论数进行汇总，再利用汇总结果绘制折线图，这里我们使用的汇总方法是序列方法 `value_counts`，该方法可以直接返回序列中各个唯一值的出现次数，调用方式为：`Series.value_counts(normalize=False, sort=True, ascending=False, bins=None, dropna=True)`

参数说明：

`normalize`：取值类型为布尔型，默认取值是 `False`，取值为 `True` 时将返回出现的频率

`sort`：取值类型为布尔型，默认取值是 `True`，即按照取值大小排序

`ascending`：取值类型为布尔型，表示是否按照升序排列，默认取值是 `False`，即降序

`bins`：只适用于取值类型为数值型的序列，用于将序列值划分到指定跨度的半开区间中，而不是进行计数

`dropna`：取值类型为布尔型，表示是否忽略缺失值，默认取值为 `True`

下面对序列"time"直接调用方法 `value_counts`，统计不同时点的评论数量并绘制折线图：

```
In [14]:df_reviews["time"].value_counts()
Out[14]:
2014-12-11    99
2015-07-01    93
2015-07-02    58
...
2014-11-06     2
2014-12-05     2
2015-02-18     1
Name: time, dtype: int64
In [15]:df_reviews["time"].value_counts().plot(kind='line', rot=0, figsize=(14, 8))
Out[15]:<matplotlib.axes._subplots.AxesSubplot at 0x7fd042724310>
@todo 补充截图 9.3.2.1
```

从趋势图可以看到，在 2015 年 1 月和 7 月均有明显的评论量增长。

8.3.2.2 分组趋势图

除了直接对时间数据进行统计汇总外，还可以以时间数据作为分组依据，对其他变量进行汇总，利用数据框的 `groupby` 方法即可轻松实现分组，最简单的调用方式如下：

`DataFrame.groupby(by=None, sort=True)`

参数说明：

`by`：分组依据，取值类型可以为函数列表、字典、序列，或者数据框列名构成的列表或元组，将按照传入的参数进行分组。如果传入的是字典或序列，将以字典或序列的值作为分组依据。

`sort`：对分组的键值进行排序，但不影响组内取值的排序，默认取值为 `True`

该方法返回一个 `GroupBy` 对象，该对象本身并不包含实际的数据分组结果，只有对该对象调用相关方法时，`pandas` 才会根据 `GroupBy` 对象记录的信息进行分块运算，并返回计算结果。

常用的 `GroupBy` 对象方法有：

`@todo 插入方法列表`

下面以 kindle 评论文本的发布时间为分组依据，计算不同时间点的平均星级：

```
In [16]:group_time = df_reviews.groupby(["time"])
group_time.mean().head()
Out[16]:@todo 补充截图 9.3.2.2-1
```

若只要看某列，可以通过索引实现：

```
In [17]:group_time["rating"].mean().head()
Out[17]:time
2014-10-03    4.645161
2014-10-04    4.909091
2014-10-05    4.666667
2014-10-06    4.800000
2014-10-07    4.421053
Name: rating, dtype: float64
```

或者直接对"rating"列进行分组：

```
In [18]:df_reviews["rating"].groupby(df_reviews["time"]).mean()
Out[18]:time
2014-10-03    4.645161
2014-10-04    4.909091
2014-10-05    4.666667
...
2015-08-22    4.230769
2015-08-23    4.645161
2015-08-24    4.400000
Name: rating, dtype: float64
```

对得到的分组统计结果直接绘制折线图，即可得到评论评分序列的平均趋势图：

```
In [19]:df_reviews["rating"].groupby(df_reviews["time"]).mean().plot(kind='line', rot=0
, figsize=(14, 8))
Out[19]:<matplotlib.axes._subplots.AxesSubplot at 0x7fd042c1aa10>
@todo 补充截图 9.3.2.2-2
```

从平均星级随时间的变动趋势图可以看出，整体星级保持在 4.3 分上下波动，在 2015 年 6 月到 7 月之间出现了历史最低值 2.5 分。

统计不同时间热门词汇“闪屏”出现的次数，并绘制折线图：

```
In [20]:group_word = df_reviews.groupby(["time"])["word_count"].sum()  
group_word.plot(kind='line',figsize=(14, 8))  
Out[20]:<matplotlib.axes._subplots.AxesSubplot at 0x7fd0423068d0>  
@todo 补充截图 9.3.2.2-3
```

可以看出，2015年3月以后，“闪屏”一次出现的次数明显增多。

8.4 小结

本章从文本数据内容可视化、文本关系可视化以及基于时间维度的文本数据可视化等三个方面介绍了文本数据可视化的常用方法及工具，具体包括：

- (1) 基于词频的文本数据可视化，包括使用 `matplotlib` 和 `Pandas` 绘制词频柱状图，使用 `wordcloud` 绘制词云图以及词云图的美化
- (2) `Series` 文本数据处理方法
- (3) 使用 `pyLDAvis` 包实现主题模型交互可视化
- (4) 基于时间信息的文本数据可视化，包括使用数据可视化库 `seaborn` 中 `matrix` 模块绘制热图、利用数据框 `groupby` 分组方法按照时间分组进行统计汇总并绘制趋势图

8.5 深度阅读材料

8.5.1 LDavis

LDavis: A method for visualizing and interpreting topics

<http://nlp.stanford.edu/events/illvi2014/papers/sievert-illvi2014.pdf>

8.5.2 pyLDavis

pyLDavis's documentation <https://pyldavis.readthedocs.io/en/latest/#>

8.6 练习

8.6.1 下载路透社新闻

(<http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html>) 文本，选取部分文本，完成以下操作：

- (1) 绘制高频名词和形容词的柱状图
- (2) 分别绘制高频名词和形容词的词云图，尝试进行词云图美化
- (3) 构建主题模型，使用 pyLDAvis 中的相应工具实现主题模型可视化

8.6.2 下载或者自行抓取电子商务网站商品页面评论文本，结合评论发布时间，绘制评论量分布热图、趋势图，以及所关注的热门词汇出现次数热图、趋势图