华 东 师 范 大 学 软 件 学 院

2015 年软件工程学士学位论文

基于 NLP 的海量中文文本观点词挖掘方法研究

The Research of Chinese Opinion Words Mining from Massive Text based on NLP

姓　　名：李克西

学　　号：10112510348

班　　级：2011 级 3 班

指导教师姓名：郭建

指导教师职称：副教授

2015 年 5 月

目录

摘 要

随着互联网信息量的逐年膨胀，越来越多的人开始注意到大数据所蕴含的价值，大数据挖掘成为一项备受瞩目的新兴技术。文本类数据占据当前数据总量的大部分比例，中文文本挖掘以其形式多样、语法复杂等特点，一直是研究工作的难点。作为文本挖掘中的重要组成部分，观点挖掘在评论处理、舆情监控等多方面具有重要的研究价值。

本文以研究中文观点词挖掘方法为中心，利用 FNLP 和 Stanford Parser 两款优秀的 NLP 工具对中文句子进行处理和解析，在 RNN 算法构造的语法树基础上设计了攀升式的观点词挖掘算法，就算法改进过程进行详细的论述，并对算法性能进行了评估。为了适应文本挖掘的多样需求，基于该算法实现了三种针对不同输入的挖掘模式，并用 Hadoop MapReduce 框架对其进行扩展，以对海量中文文本数据进行处理。

从评测结果中看到，本文提出的观点词挖掘算法在句子结构完整、语义明确的评论句中有着出色的表现。其中，<句子，关键词>模式的 F 值趋近于 90%，关键词表模式的 F 值趋近于 86%。实验同时也说明，Stanford NLP 的 Deep Learning 方法对中文文本的处理同样适用。随着更深入的研究和 NLP 技术的进一步发展，该方法的效果将会得到进一步提高。

关键词：自然语言处理，文本挖掘，大数据，观点词挖掘

Abstract

With the expansion of Internet information, more and more people begin to notice the inherent value of big data, and big data mining becomes a newly-developing technology in the public eye. Text data occupies most proportion of the total data in current years. Chinese text mining has been a difficult research due to its various forms, grammatical complexity, and etc. As an important part of text mining, opinion mining has significant research value in review processing, public opinion monitoring and so on.

This paper centers the research of Chinese opinion words mining method, and uses two excellent NLP tools, FNLP and Stanford Parser, to analyze the Chinese sentences. Based on the syntax tree constructed by RNN algorithm, I design a rising opinion words mining algorithm which discussed in detail on algorithmic processing, and evaluate the algorithm performance. In order to meet the diverse needs of text mining, the algorithm is realized to three different input mining modes, and with Hadoop MapReduce framework expanded to deal with the massive Chinese text data.

According to the testing result, opinion words mining algorithm in this paper has a good performance in the comments sentences which have intact structures and explicit semantics. The value of F approaches 90% in <sentence, keyword> pattern, while it approaches 86% in keyword table pattern. Experiments also show that Stanford NLP's Deep Learning method is suitable for Chinese text mining. With the further development and more in-depth study of NLP technique, the effect of this method will be improved in the future.

Keywords: Natural Language Processing, Text Mining, big data, opinion words mining

一、 绪论

（一） 研究背景

随着大数据时代到来，基于大数据挖掘的应用越来越多地渗透到人们的生活中来，例如电商、中介、社交网站的产品推荐系统，金融机构的投资决策系统，以及邮件服务商的垃圾邮件过滤系统等。从海量数据中挖掘有价值的信息已经成为各公司展开攻坚的热点。

通常而言，计算机中的数据分为两种，一种是存储在数据库或结构化文件（如 XML）中的数据，这类数据格式规范，使用方便，利用率高。而另一种数据则是非结构化数据，其主要代表便是文本数据。据估计，这类数据约占目前数据总量的 80%，形式复杂，难以直接利用，但蕴含十分丰富的有效信息，因此，文本挖掘越来越成为当前研究的重点内容。

文本挖掘也称文字探勘或文本数据挖掘，一般指通过文本处理得到高质量信息的过程。典型的文本挖掘方法包括文本分类，文本聚类，概念/实体挖掘，生产精确分类，观点分析，文档摘要和实体关系模型。文本挖掘与信息检索、数据挖掘、机器学习、自然语言处理等重要领域息息相关。[1]

观点提取则是文本挖掘研究的一个重要组成部分，旨在对特定对象提取描述、修饰的词汇或短语，从而了解对该对象看法。这在评论反馈、舆情监控等诸多方面有非常重大的价值。尤其对于互联网企业、电商而言，及时了解用户评论内容，对制定改进措施和进一步拓展市场有着重要意义。因此，本文将围绕从中文文本中提取观点词的方法进行研究和论述。

目前英文文本挖掘技术已经有了相当大的进展，而包括中文在内的其他语言的文本挖掘技术仍处于攻坚阶段。中文文本具有较高的复杂性，从分句、分词、标注等基本工作，到语法剖析、关系挖掘、情感分析等高级工作，其困难度都远远高于英文。

传统的中文文本挖掘方法以向量空间模型为基础，分词后采用 TF-IDF 方法计算文本词频向量，进而可使用各种基于词频向量的方法来完成文本挖掘工作。这种方法被广泛应用于信息检索和机器学习领域，但由于词频向量并没有关注句子本身结构和语法特点，因此在深度挖掘文本信息时准确性和适应性较差。

近年来，美国 Stanford 大学自然语言处理工作小组提出一套基于深度学习

的自然语言处理方法，利用人工神经网络使计算机学习语言处理规则，建立语言处理模型，很大程度上提高了自然语言处理的准确性。如何利用有效的自然语言处理方法进行海量文本数据中的观点词挖掘，是本文作者研究的重点内容。

（二） 国内外研究现状

中文观点挖掘技术目前还处于起步阶段，已有的各种方法的准确率和召回率还有待提高。

最早的中文观点挖掘的研究见于2005年，Benjamin K.Y. Tsou 和 Raymond W.M. Yuen等[2]对600篇新闻进行极性分析，从整个文档的层次对布什、克里、小泉纯一郎、陈水扁四位政界人物的声誉褒贬情况进行分析。

娄德成（2007）[3]利用 NLP 方法对中文句子的语义极性倾向进行了研究，通过计算词语的上下文极性与句法分析寻找主题词与极性成分的匹配关系。从而判断句子中每个主题的极性倾向。

章剑锋，张奇，吴立德，黄萱菁（2008）[4]应用最大熵模型，并结合词、词性、语义和位置等特征进行关系抽取，一定程度上解决了指代消解以及评价对象遗漏的问题，F 值比已有的方法有了很大的提高。

王辉，王晖昱，左万利（2009）[5]实现了一种针对英文的基于特征的观点挖掘方法，从语句层次提取观点的具体细节，能够识别评论者评论中的产品特征，并对其观点进行情感分析。

丁晟春，孟美任，李霄（2014）[6]根据微博的语体特征分析结果选取情感特征和观点句特征，结合CRFs模型进行观点句识别研究，使得观点挖掘的召回率得到了显著提高。

（三） 研究内容和相关工作

笔者在上海惠普有限公司全球 IT 技术部门实习期间，曾参与多个与企业产品评论分析的任务。决策者在评论处理方面比较注重针对产品某些特定方面进行评估，而传统的机器学习算法无法满足这一细粒度的需求。因此，笔者开始对基于自然语言处理的文本挖掘方法进行了一系列研究，本文则专门针对中文文本的观点词挖掘方法展开论述。

为了进行本课题的研究，笔者对当前主要的文本挖掘技术和自然语言处理工具进行了广泛的了解和比较，最终使用复旦大学的 FNLP 和 Stanford 大学的

Stanford NLP 工具作为文本挖掘的核心工具，并在 Stanford Parser RNN 算法构造的语法树基础上研究观点词挖掘算法，并逐步优化，评估效果。最后，为适应海量文本的挖掘的需求，利用 Hadoop MapReduce 框架对算法进行扩展。

（四） 论文组织

本文共分为六个部分，第一章节介绍了问题的研究背景，然后介绍笔者在撰写之前所做的一系列工作，包括相关文献的学习和程序开发过程。

第二章节是对相关技术的介绍，包括自然语言处理技术、美国 Stanford 大学在 NLP 技术方面取得的突破性进展以及当前十分火热的大数据处理技术的概述；

第三章节是本课题程序的总体流程和架构，以及程序开发的思路，详细介绍了每个部分所使用的工具以及该部分的实现方法；

第四章节是本文重点内容，详细讨论了对于中文文本，观点词挖掘算法的原理、描述、改进点以及改进后的算法评价，然后给出了在实际运用场景中的三种模式及实现方法。

第五章节是采用 Apache Hadoop 的 MapReduce 框架对算法进行的一个扩展，经过环境配置、程序的实际开发和调试，使之能够运用到海量数据的挖掘上；

最后，在第六章节对全文进行了总结，并对程序可进一步改进和运用的方面作出展望。

二、 相关技术

（一） 自然语言处理技术

自然语言处理（Natural Language Processing，简称 NLP）是人工智能和计算机语言学领域的分支学科，是一种让计算机理解人类的语言的技术。

Bill Manaris 早在 1999 年《从人-机交互的角度看自然语言处理》一文中说“自然语言处理可以定义为研究在人与人交际中以及在人与计算机交际中的语言问题的一门学科。自然语言处理要研制表示语言能力和语言应用的模型，建立计算框架来实现这样的语言模型，提出相应的方法来不断完善这样的模型，根据这样的语言模型设计各种实用系统，并探讨这些实用系统的评测技术”。[7]

目前主要的自然语言处理模型是基于统计语言模型，宗成庆在其著作《统计自然语言处理》一书中介绍了包括概率论、信息论、支持向量机（Suport Vector Machine，简称 SVM）、形式语言和自动机在内的许多自然语言处理方法，涵盖分词、词性标注、句法分析、语义消歧、机器翻译、文本分类、自动摘要、信息抽取等主要 NLP 方法。[8]世界上主要的自然语言研究机构包括斯坦福大学、卡内基梅隆大学、哥伦比亚大学等，Apache 基金会也专门设立的 Open NLP 项目进行该方面的工作。在中国，北京大学、复旦大学、台湾大学、哈尔滨工业大学等知名学府对中文自然语言处理作出了较大贡献。

（二） 自然语言处理与机器学习方法的区别

机器学习（Machen Learning）是 20 世纪人工智能领域兴起的一门交叉领域学科，研究计算机怎么模拟人类进行学习和经验积累，并不断完善自身结构和性能，在分类、聚类、关联规则挖掘方面有突出的表现。其曾经一度成为文本挖掘的主流，其中 Bayes 分类、LDA 话题提炼、文本聚类、SVM 都是非常优秀的文本挖掘方法。

但是，使用机器学习算法有一个共同的特点，就是在词频的基础上对文本进行向量化或概率估计，而没有关注自然语言本身的语法规则，因而很多关注文本细节的功能难以实现，如信息抽取、观点提取、情感分析等。

自然语言处理方法的出发点是对语言本身，关注自然语言结构内在的联系，采用统计学的方法建立语料库，设计相应的算法进行自然语言的解析，从而能更好的分析文本中所蕴含的信息。

（三） Stanford NLP 深度学习方法

1. 概述

Stanford NLP 工作组是目前世界上最知名的自然语言处理方法科研机构之一，项目领导人 Christopher Manning，从 1995 年起就在 Stanford 大学从事自然语言处理的研究。其团队发布的开源自然语言处理工具 Stanford NLP 是目前主流的自然语言处理包之一，涵盖英语、中文、德语、法语、阿拉伯语，实现了包括分词（tokenize）、分句(Sent. split)、词性标注(POS)、词素规约(Lemma，即将单词在不同语境下的不同呈现形式规约为同一个词素)、命名实体识别(NER)、语法分析(Parse)、依存句法分析(Dep. Parse，即在句子中标注出每个部分与有直接依赖关系的其他部分)、情感分析(Sentiment)等在内的十余种功能，支持 Java、C#/F#/.Net、Python、Ruby、Perl、Scala、Clojure、JavaSript 等多种编程语言或平台。其覆盖功能如图 1 所示。

Annotator	Ara-bic	Chi-nese	Eng-lish	Fre-nch	Ger-man
Tokenize	✓	✓	✓	✓	✓
Sent. split	✓	✓	✓	✓	✓
Truecase			✓		
POS	✓	✓	✓	✓	✓
Lemma			✓		
Gender			✓		
NER		✓	✓		✓
RegexNER	✓	✓	✓	✓	✓
Parse	✓	✓	✓	✓	✓
Dep. Parse		✓	✓		
Sentiment			✓		
Coref.			✓		

图 1 Stanford Core NLP 覆盖功能[1]

近年来，Stanford NLP 项目组率先提出使用 Deep Learning 方法对自然语言处理算法进行改进，在准确性上取得了突破性的进展。

[1] 图片出自 *The Stanford CoreNLP Natural Language Processing Toolkit* ，见引文[10]

2. 向量空间模型与基于单词的向量空间模型

1) 向量空间模型

向量空间模型（Vector Space Model，简称 VSM）于 20 世纪 70 年代由 Salton 等人提出，最早用于信息检索领域，是目前较为成熟的理论之一。[11]其思想是将一篇文档 d_j 抽象 n 维向量，每个维度为文档中一个单词（token）$w_{i,j}$ 相对该文档的一个权重。

$$\boldsymbol{d_j = (w_{1,j}, w_{2,j}, \dots, w_{n,j})}$$

目前使用最广泛的计算向量维度方法是 Tf-Idf，该方法有效避免了公共常用词成为高权重词的情况。其中，Tf（词项频率）代表某个单词出现在该文档中的频数，Idf（逆文档频率）是指在一个文档集合中，某个单词出现过的文档数目的倒数，其出现篇数越多，Idf 值越小。计算公式如下：

$$\boldsymbol{w_{t,d} = tf_{t,d} \times \frac{log|D|}{|\{d' \in D | t \in d'\}|}}$$

其中|D|是文档集合中的文档总数，|{d’ ∈D|t∈d’}|是含有单词 t 的文件数。采用 Tf-Idf 方法计算文档向量可以有效避免常用词占据文档权重过高的现象。

当需要计算两篇文档(d1,d2)相似程度时，通常采用余弦相似度，公式如下：

$$\boldsymbol{sim(d1, d2) = cos\,\theta = \frac{d1 \times d2}{||d1| \times |d2||}}$$

因此，在向量空间模型中，两篇文档的相似程度主要取决于文档中相同的单词规模。

2) 基于单词的向量空间模型

如果两篇文档讲述同一件事情，但是使用的是完全不同的单词（譬如用别称或近义词代替），那么使用向量空间模型的效果就会大打折扣甚至完全失效，这是因为 VSM 没有在语义层面上对文档进行任何处理。

为了解决这一问题，Stanford NLP 工作组提出了基于单词的向量空间模型（Build on Word Vector Space Model，简称 BWVSM）。设 D 为向量空间（可能出现的向量集合），R 代表相关标签集合（可能与该单词有关联的其他单词集合），

那么 BWVSM 将每一个单词 w 用一个三元式（v, R, R^{-1}）来表示。V 描述该单词的词义, R 代表 R→D 映射中对于每个关系，单词 w 的选择偏好，R^{-1} 代表 R→D 映射中对于每个关系，单词 w 的逆选择偏好。[13]

$$\boldsymbol{w = (v, R, R^{-1})}$$

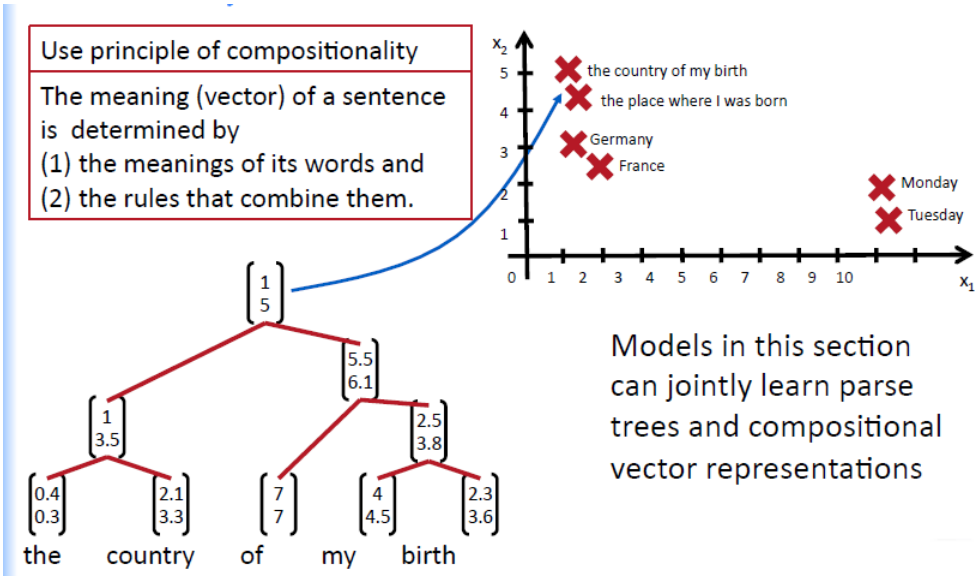

图 2 基于单词的向量空间模型[2]

这种方法从每一个单词的词义和结合评分出发，逐步将单词向量结合成短语向量，进而结合成句子向量。在这种方法下，两个句子即使出现的单词是完全不同的，如果描述的是同一件事情，那么它们的相似度也会很高。

3. 基于人工神经网络的语法树建立方法

1) 人工神经网络概述

人工神经网络是受生物大脑启发，并模仿其结构和功能，采用数学和物理方法进行研究，从而构成的一种新型信息处理系统。它由许多简单的并行工作的处理单元（人工神经元）按照某种方式相互连接，并依靠其状态对外部输入信息进行动态响应。具有并行的结构和处理能力，有良好的容错性、高度非线性和计算的非精确性，以及自学习、自组织和自适应等特点，并具有非线性映射、模式识

[2] 图片出自 Richard Socher 和 Christopher Manning.讲座 Deep Learning for Natural Language Processing，http://nlp.stanford.edu/courses/NAACL2013/，图 4 同。

别、分类与聚类、联想记忆、优化计算以及知识获取与表示等功能。

人工神经元是人工神经网络的基本处理单元，其功能是对每个输入信号进行处理以确定其强度（加权）；确定所有输入信号的组合效果（求和）；确定其输出（转移特性），其结构图 3 所示。

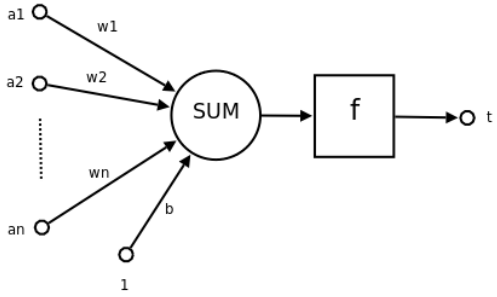

图 3 人工神经元

设输入向量为

$$\mathrm{A}_j = (a_1, a_2, \ldots, a_n)^T$$

其中 a_i 表示第 i 个神经元的输入，是神经元 j 的多个输入之一。输入神经元到神经元 j 的加权向量为

$$\mathrm{W}_j = (w_{1j}, w_{2j}, \ldots, w_{nj})^T$$

式中 w_{ij} 表示从第 i 个输入神经元到神经元 j 的加权值。神经元 j 的阀值为 bj，则神经元 j 的输入加权和为

$$\mathrm{s}_j = \sum_{i=1}^{n} a_i w_{ij} - \mathrm{b}_j$$

因此，神经元 j 的输出状态为

$$\mathrm{t} = \mathrm{f}(\mathrm{s}_j) = f(\sum_{i=1}^{n} a_i w_{ij} - \mathrm{b}_j)$$

其中，f(.)是转移函数。常用的转移函数主要有 4 类：线性函数、阈值型函数、非线性函数和概率型函数。多个人工神经元经某种方式相互连接便形成了人工神经网络。[15]

2) 基于神经网络的向量合并算法

Stanford NLP Deep Learning 方法充分利用人工神经网络的优点，设计了递

归神经网络（RNN）算法来实现语法树的建立。每一个单词有一个对应的向量值，两个相邻单词作为人工神经网络的输入，输出值为合并后的短语语义向量值和该合并的评分。

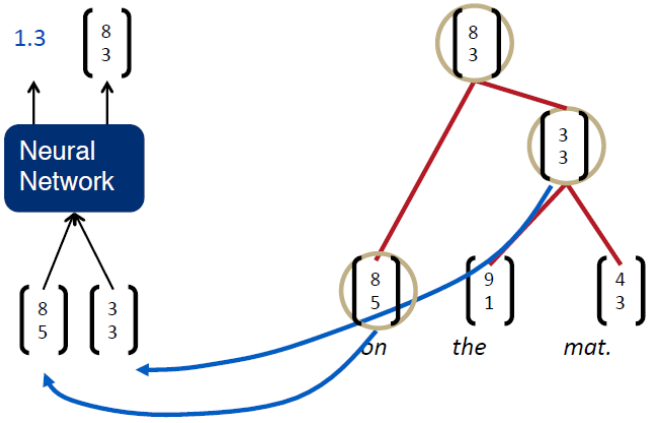

图 4 基于自然神经网络的向量合并算法

使用人工神经网络合并向量的计算方法如下：

$$p = \tanh\left(W\begin{bmatrix}c1\\c2\end{bmatrix} + b\right)$$

转移函数 tanh 是一个双曲函数，W 是经过训练建立的模型矩阵，b 是偏置。归约评分计算方法如下：

$$score = U^T p$$

其中 U 也是经过训练建立的评分模型矩阵，p 是合并后的语义向量值。

3) 递归神经网络与语法树建立方法

整个语法树的建立需要做很多次单词向量的合并，直到整个句子只有一个语义向量。这是一个递归类型算法，我们把语法树建立算法描述如下：

1	分词
2	对每个单词进行向量标注
3	重复以下步骤，直到该句子只有一个语义向量：
4	从第一个语义向量开始，依次将相邻的两个语义向量进行合并（如果记录表中 5 已经有合并运算结果，则跳过），将合并后的向量值和评分填入记录表中；
5	找出本轮合并中评分最高的一个合并，作为本轮的合并，语法树向上生长；

（四） 大数据处理

近年来，互联网上的信息增长速率越来越快，大数据处理技术被提上了一个新的高度。据国际数据公司（IDC）的研究结果称，2008 年全球产生的数据量为 0.49ZB，2009 年为 0.8ZB，2010 年增长为 1.2ZB，2011 年更是高达 1.82ZB；而 IBM 的研究报告称，截止到 2012 年，人类文明所获得的全部数据中，90%是在过去的两年中产生的。而到了 2020 年，全世界所产生的数据规模将达到今天的 44 倍。[16]面对如此大规模的数据量，传统的单机环境进行数据挖掘的效率已经远远不能满足需求，而大数据处理技术正是应这种需求而诞生的。

自从 Google 公司发表关于 MapReduce 和 Google 档案系统的论文后，大数据处理成为计算机界一个举足轻重的话题。Apache 基金会为此专门成立 Hadoop 项目，用于大数据处理系统框架的开发，该项目已成为目前最主要的大数据处理技术的应用框架。

（五） 本章小结

本章节对文本挖掘中几个相关的技术进行了阐述。首先，解释了使用 NLP 方法而非机器学习方法进行高精度的文本挖掘的原因；接着，对 Stanford NLP 提出的 Deep Learning 方法进行了概述，介绍了单词向量空间模型与传统的向量空间模型的区别、人工神经网络的构造原理以及基于 RNN 算法建立语法树的算法过程。最后，对大数据处理进行了简单的介绍，明确了本文将提出的方法的技术背景和基础知识。

三、 总体架构

本文对应的程序分为两步完成，第一步实现在单机环境（即普通的 Java 开发环境）下的观点词挖掘程序，主要有三个过程：文本预处理、语法解析和观点词挖掘。第二步是利用 MapReduce 程序框架对已实现的程序进行扩展。总体流程如图 5 所示。

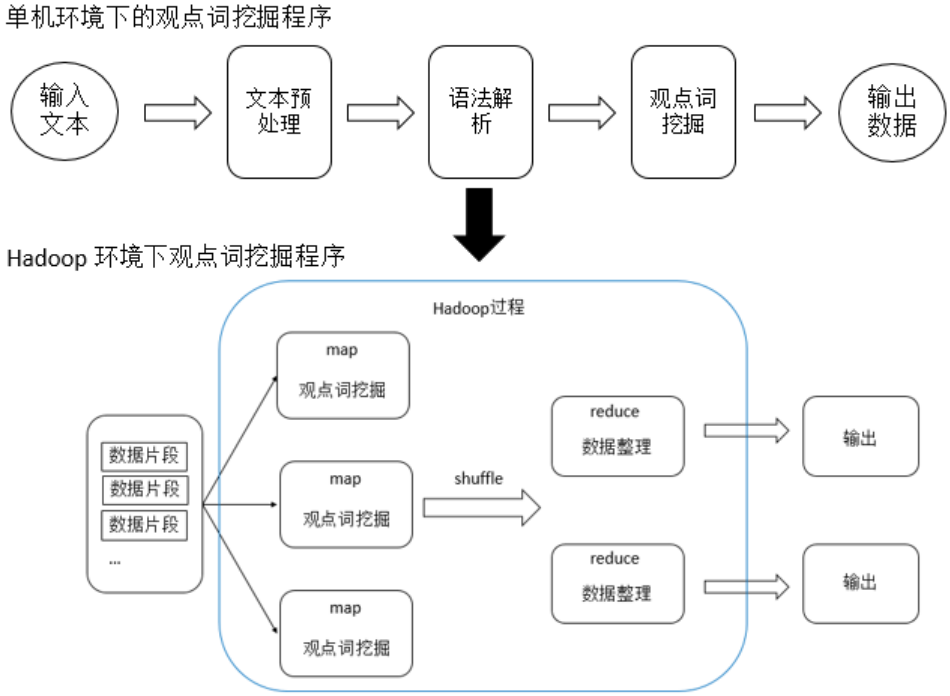

图 5 总体流程

对于单机版的观点词挖掘程序，第一部分是文本的预处理，即从源文本到 token stream 的转换，该过程涉及数据读取、分句、分词三个步骤；第二部分是语法解析，即从 token stream 到语法树的建立；第三部分是观点词挖掘过程，该部分需要设计观点词挖掘算法和合适的挖掘模式以提取出我们的预期结果，然后将结果进行整理和输出。

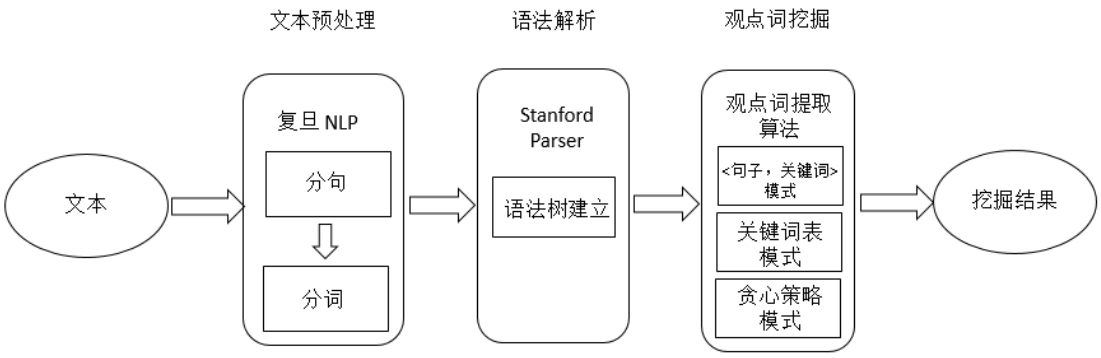

图 6 单机环境下程序执行详细过程

对于 Hadoop 环境下的程序，则是将单机版本中的三大核心部分融入到 Map 过程中，而对数据的整理部分融入到 Reduce 过程中。

出于代码兼容性和考虑，本程序采用 java 语言编写，JDK 配置为 1.7，使用到的工具有 FNLP，Stanford Parser，Apache hadoop 2.6.0 以及 Apache Maven 3.3.1。

（一） 文本预处理

文本预处理主要包含输入文本的读取、分句和分词三部分。输入文件的读取在不同环境下有所不同，单机环境下可使用 JDK 提供的 IO 库，而在 Hadoop 环境下，HDFS 有专门的输入文件读取方法。

分句的方法比较简单，通常使用中文句子终结符“。”、“！”，“？”、“……”等符号对段落进行分句，这里不再赘述。

在开发过程中，使用 FNLP 作为分词工具，其分词效率和中文分词准确性要高于 Stanford NLP。该部分输出为一个由词组成的 String，即 token stream，词与词之间用空白符隔开。

FNLP 进行分词操作需要构造一个标注器 CWSTagger，可调用文件分词函数 tagFile 或对 String 进行分词的函数 tag。CWSTagger 在构造时需要加载一个模型文件，也可以通过添加自己构造的分词词典来提高对新词的适应性。使用复旦 NLP 对句子分析效果如图 7 所示，该结果为使用 FNLP 内置词典进行分词的效果。

1. 在 股市 低迷不振 的 日子 里 ， 新股 依旧 涨 得 非常 好 。
2. 惠普 笔记本 的 销量 一直 都 不 错 。
3. 没有 什么 比 这个 东西 更 好 了 ！
4. 在 股市 低迷不振 的 日子 里 ， 新股 依旧 涨 得 非常 好 。
5. 最近 ， 众筹 的 火爆 已经 超乎 所有 人 的 想象 ， 就 像 目前 的 股市 一样 疯狂 得 不 可理喻 。
6. 今天 的 新闻 真正 好 ， 一 个 铜板 就 买 两 份 报 。
7. 苹果 电脑 的 显示器 效果 非常 棒 ， 我 特别 喜欢 ！
8. 头 几 天 淘宝网 上 出现 了 很多 假 货 。
9. 穿 着 很 舒服 ， 发货 也 快 ， 服务 态度 也好 。 一 个 字 “ 棒 ” 。

图 7 分词结果示例

（二） 语法解析

使用 Stanford Parser 将上一步得到的 token stream 转换成语法树时，首先需要构造一个语法解析器 LexicalizedParser，并加载模型文件。本文使用了 Stanford Parser 内置的模型文件 chinesePCFG.ser.gz。

解析好的语法树存放在 Stanford Parser 实现的一个数据结构 Tree 当中，并作为这一步的输出。利用 Stanford Parser 提供的图形化工具我们可以直观地看到 Tree 的组织形态。

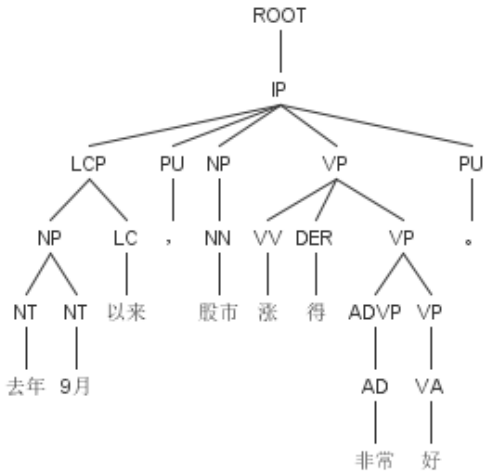

图 8 语法树示例

（三） 挖掘过程

挖掘过程是整个程序的重点部分，主要研究观点词挖掘算法的设计、实现、评价及改进方法，以及在实际应用中挖掘模式的实现。

观点词挖掘算法是这一步的核心内容，该算法以上一步得到的 Stanford 语法树作为输入，以句子中的一个关键词作为起始点，按照观点词的提取规则采用一定的语法树访问策略来挖掘其所对应的观点词。目前已有一些企业团队在进行文本挖掘时使用到了语法树，但就目前公开的资料看，这些已有的成果并没有进行理论验证，在方法上也存在一定的错误。本文从理论出发，进一步总结了基于语法树结构的观点挖掘过程中遇到的问题和改进措施，最后给出了自己的算法。对于算法的评估，采用文本挖掘中常用的度量手段，计算算法对测试集合的准确率、召回率、F 值以及处理每句用时。

在观点词提取算法的基础上，针对不同的应用场景和文本输入类型，设计了三种观点词挖掘模式：<句子，关键词>模式、关键词表模式和贪心策略模式。这些模式基本能够涵盖算法的主要应用场景。图 9 给出了三种挖掘模式流程的对比。

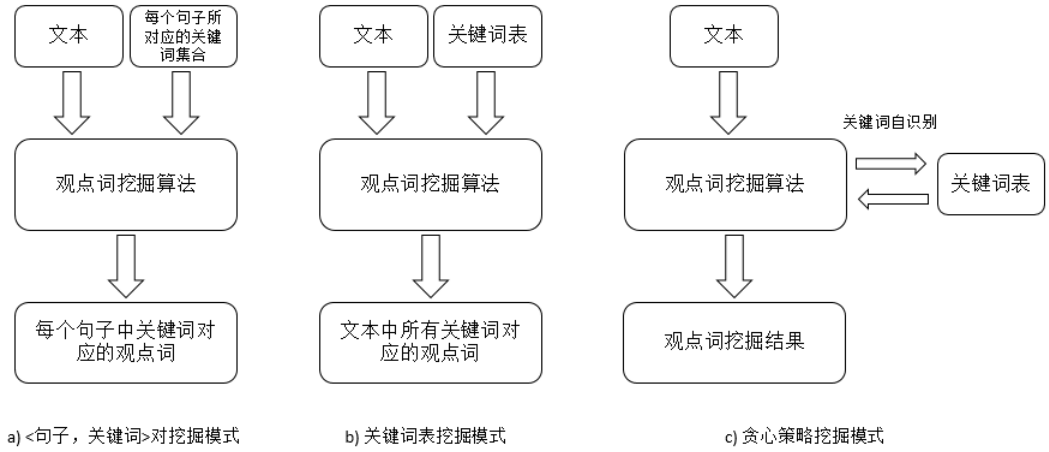

图 9 三种挖掘模式的流程

不同的挖掘模式需要的输入数据中关键词存在形式各不相同，但最终会将输入数据整理成观点词挖掘算法的参数的两个参数——句子和对应的关键词两个参数。<句子,关键词>模式需要手工或用其他方式将每个句子的关键词标注出来；关键词表模式则是对于一个文本集合，只需给出想要了解的一些关键词，作为一个整体输入；贪心策略模式不需要输入关键词，而是在处理过程中采用一定的方法自动识别句子的关键词。

对于挖掘过程的细节，将在第四部分内容当中进行详细介绍。

（四） 本章小结

本章节对观点词挖掘程序的总体框架和流程进行了介绍。本文中的程序分两步实现，第一步是实现在单机环境下的观点词挖掘程序，第二步将其扩展为 Hadoop 程序。

程序的核心部分共 3 个流程,首先是文本的预处理,将输入文本转换成 token stream；其次是语法解析，将 token stream 转换成语法树的形式；最后是观点词挖掘算法，根据关键词的类型和在语法树的位置，通过对应的提取规则找到所对应的观点词。

本章节中还提到了不同场景下应用三种的挖掘模式，介绍了挖掘模式与算法之间的调用关系。根据实际输入的不同，设计合适的挖掘模式，拓展了程序的使用面。

四、 关键算法设计

作为本文核心部分，本章节将对观点词提取算法的原理、过程、评价及改进等内容作出详细的论述，并对三种挖掘模式的设计实现方法进行论述。

在利用 NLP 工具对输入文本进行预处理和语法解析后，我们得到了按照 RNN 算法建立的中文语法树。为了利用该语法树实现观点词提取算法，首先应当对一系列数据结构进行分析。

(一) 相关概念和数据结构

1. 关键词和观点词的定义

在一个句子中，我们将想要了解的对象称为对于该句子的关键词（Key Word），将描述该对象的修饰词成为观点词（Opinion Word）。

例如，在句子“去年 9 月以来，股市涨得非常好。”一句中，如果“股市”是关键词，那么“非常”和“好”就是描述关键词的观点词。

在电子商务、社交网站上，往往有大量的评论或状态，其中存在大量极性句子。通过对这些句子进行观点词挖掘，我们可以了解一个关键词对应的观点词的分布，从而作出对应的决策。

2. 语法树的结构

利用 Stanford Parser 对已经分好词的句子进行语法剖析将得到句子语法树。从图 8 直观地看到，语法树的叶子节点为一个词语或标点符号的词素（lexeme），每个叶子节点的父节点是该词语或符号的类型助记符；从叶子节点的父节点开始，向上归约成短语，较短的短语再往上与单词或短语归约成新的短语或句子，一直到根节点成为一个完整的句子。

各助记符的含义请参见附录（二）。一般情况下，一个父节点下只有两棵子树，但也存在多棵子树的情况，因此整个语法树呈多路的形态。

语法树实现在 Stanford Parser 包中的 edu.stanford.nlp.trees 下，是一个抽象类。出于封装性考虑，该类的成员变量声明为私有，提供了数十个公共访问的方法对 Tree 的实例进行操作。这里介绍一些主要的方法：

```
public boolean isLeaf()
```

功能：判断当前 Tree 是否是叶子结点。

```
public int numChildren()
```

功能：获取当前 Tree 的子树数目。

```
public List<Tree> getChildrenAsList()
```

功能：以 List 的格式获取当前 Tree 的子树集合。

```
public void setChildren(Tree[] children)
```

功能：将一个 Tree 数组中的所有 Tree 设置为当前 Tree 的子树。

```
public Tree firstChild()
```

功能：得到当前 Tree 的第一棵子树，对应的，还有获取最后一棵子树和第 i 棵子树的方法。

```
public int depth()
```

功能：获取当前 Tree 的深度。

```
public <T extends Tree> List<T> getLeaves()
```

功能：按从左到右的顺序获取当前 Tree 的所有叶子结点，以 List 形式返回。

```
public Tree parent()
```

功能：获取当前 Tree 的父树。

```
public Iterator<Tree> iterator()
```

功能：获取用于遍历当前 Tree 的迭代器。

```
public List<Tree> postOrderNodeList()
```

功能：后续遍历当前 Tree 的结点。对应的还有前序、中序遍历方法。

使用这些公共访问的方法已经能够满足算法实现的所有需求，因此不需要对抽象类 Tree 进行进一步的实现。

（二） 观点词挖掘算法

句子中的被描述对象一般为名词、动词或形容词类型，其中名词包括 NN（普通名词）、NT（时间名词）、NR（命名实体）三种，对应的需要提取的观点词为形容词 JJ、VA 及相应动作 VV；修饰形容词和动词的观点词则是副词 AD。

根据 RNN 算法的思想，采用动态规划的思想，对一个 token stream，每次会计算相邻单词组合的评分和合成向量值，每一步将评分最高的两个向量归约为一个新的向量，直到没有可进一步归约的向量存在。

那么当分词和语法解析正确时，相关程度越高的部分会越早归并为一个向量。也就是说，被描述对象所在子树与修饰词所在子树会比与其他部分更早地归约到一棵子树上，反映在语法树上可以有以下结论：

> **结论 1**：与被描述对象所在子树成为兄弟子树的对象，在整棵语法树的层次越低，所含有的观点词与被描述对象越相关。

例如在句子“勇敢的李明有一颗善良的心”中，若关键词为“李明”，则其修饰词“勇敢”会早于“善良”与其归约到一棵子树中；同理，关键词为“心”时，其修饰词“善良”也会早于“勇敢”归约到同一棵子树中。

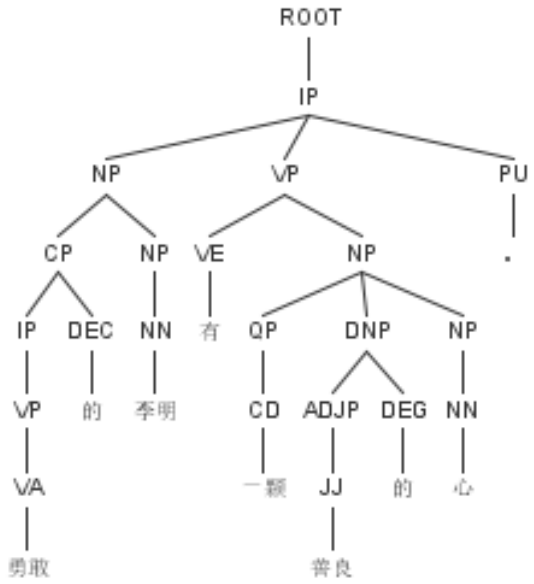

图 10 结论 1 实例

基于这个结论，我们可以在语法树已经建立的前提上，设计访问策略，从而实现观点词挖掘算法。

1. 算法描述

针对基于语法树结构的观点词挖掘算法，一般描述如下：

```
strExtra ( token _stream , keyword )
1   遍历 token_stream，寻找关键词
2   If  keyword 不存在与 token_stream 中
3       return null
4   根据 keyword 所在位置找到其所在叶子节点
5   找到该叶子节点的父节点
6   将起始节点设置为叶子节点的父节点
7   从起始节点作为当前节点开始，重复以下规则，直到根据挖掘规则找到观点词
    opinion word 或当前节点为根节点 root：
8       列出当前节点父节点的兄弟子树
9       确定兄弟子树的访问顺序
10      for 子树 subtree : 兄弟子树列表
11          遍历 subtree，按照抽取规则搜索观点词 opinion word
12          If 找到了观点词
13              return opinion word
14      将当前节点的父节点设置为当前节点
15  return null
```

该算法的输入有两个，一是已经分好词的句子 token stream，是一个 String，token 与 token 之间使用空格符隔开；二是关键词 keyword，类型是一个 String。记语法树层数为 m，token 总数为 n。

1-3 行程序首先对 keyword 是否存在于 token stream 中作出判断，不存在就直接返回 null。这部分的时间开销为 O(n)。

接着，4-5 行里是一些准备工作，需要记录 keyword 在 token stream 中所在位置，Stanford Parser 将 token stream 解析为语法树之后，有一个 List 指向各个 token，因此只要知道 token 的位置，就能在 O(1)的时间开销里找到 token 所在叶子结点。然后标记叶子结点的父节点为起始结点，准备进行挖掘。

7-14 行是算法的主体部分，这是一个两层嵌套循环结构。外层循环中，列出当前节点的兄弟子树，确定兄弟子树的访问顺序，并进行访问，如果找到 opinion word 则返回，否则继续向树的上一层进行相同的操作；内层循环则是对每一个兄弟子树进行访问，搜索 opinion word。这部分的时间开销为 O(m . n)。

2. 访问顺序

在算法第 9 行我们提到了“确定兄弟子树的访问顺序”，不同的访问策略对于算法的准确率有很大的影响。名为“淘宝商城数据挖掘团队”的博主在其新浪博客中提到了相似的挖掘算法，其采用的访问顺序为顺序访问。[3]如图 11 所示，若 J 为关键词父节点，L 为观点词父节点，首先对 K 进行访问，若 K 中不存在 opinion word，则对 B、C、E 三个子树进行访问。按照顺序访问策略，则访问顺序为 B→C→E，如果 B 或者 C 中存在挖掘规则对应的词性，那么将得到错误的结果。

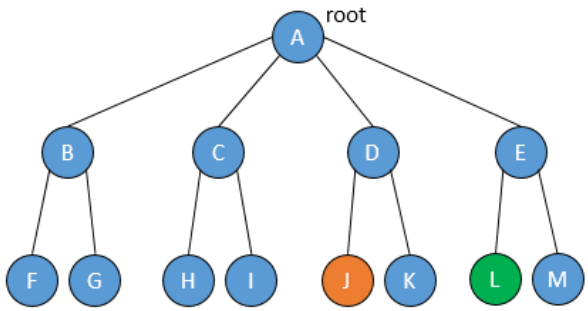

图 11 算法过程示例

这种顺序访问的方法对于较短的句子是适用的，因为短句中往往只有一个观点词修饰关键词。但是句子长度一旦增加，这种策略的缺陷就变得显而易见了。例如，对句子“在股市低迷不振的日子里，新股依旧涨得很好。”，如果以“新股”作为关键词，采用顺序访问兄弟子树的方法，第一次访问的子树是第 2 层中第一个子树 PP，那么最终提取的结果是“低迷不振”，这与这句话原本的含义大相径庭。

为了修正该错误，我采取的策略一般是“最近子树优先”，对应于图 11 则先访问 C 或者 E，如果没有找到观点词，再访问 B。为了实现该策略，每一次列出兄弟子树后，需要增加一个变量记录当前子树在兄弟子树列表中的位置，然后根据该值和访问次数决定下一个访问的子树。回到图 12 的例子，第一次访问的是第 2 层第 4 棵子树 VP，最终结果为“涨”“很”“好”。

[3] 博客地址：http://blog.sina.com.cn/s/blog_a16534260100z7hw.html，见参考文献[17]

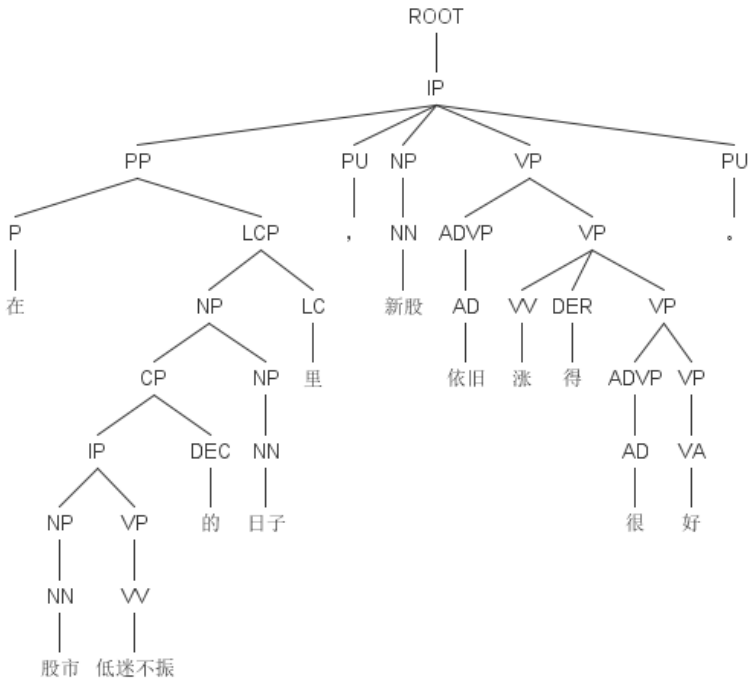

图 12 访问顺序出错示例

3. 挖掘规模

最初的算法还有一大缺陷是对于一个关键词只能提取一个对应的观点词，这会使算法的召回率大打折扣。对于短句而言，只提取一个观点词可能是适用的，但是对于图 12 中那样的句子，只返回“涨”或“好”都不全面。因此算法不应该在找到一个观点词后就立即返回，而是将正在遍历的兄弟子树集合遍历完成之后再返回观点词 list。

4. 改进后的算法

将访问顺序和挖掘规模对应的代码进行修改后，算法描述如下，修改部分在第 9-20 行：

```
strExtra ( token_stream , keyword )
1   遍历 token_stream，寻找关键词
2   If  keyword 不存在与 token_stream 中
3       return null
4   根据 keyword 所在位置找到其所在叶子节点
5   找到该叶子节点的父节点
6   将起始节点设置为叶子节点的父节点
7   从起始节点作为当前节点开始，重复以下规则，直到根据挖掘规则找到观点
    词 opinion word 或当前节点为根节点 root：
8       列出当前节点父节点的兄弟子树
9       找到当前子树在兄弟子树列表中的位置 index
10      incre = 1，flag = false
11      while 已访问子树数目<兄弟子树数目
12          访问第 index + incre 棵子树，如果找到 opinion word 则放入 list
13          访问第 index – incre 棵子树，如果找到 opinion word 则放入 list
14          If 这两次访问中找到到 opinion word
15              flag = true;
16          incre++
17      if flag == true
18          return list
19      将当前节点的父节点设置为当前节点
20  return null
```

算法的第 9-10 行寻找当前子树在兄弟列表中所在位置，并初始化访问增量 incre 为 1，搜索标志 flag 为 false，目的在于以当前子树作为起始点逐步访问左右兄弟子树。

11-16 行中，每一次循环会访问两棵兄弟子树，分别是第 index+incre 和 index-incre 棵，如果在每次访问过程中找到了 opinion word，那么在把 opinion word 放入 list 之后，会将 flag 设置为 true。最后增加一次访问增量 incre，以访问更远的两棵兄弟子树。

17-19 行会检查本轮循环中是否找到了 opinion word，如果找到了，就停止往上查找；若没有，则向上攀爬一层继续查找 opinion word。

算法修改后，对应的部分的时间开销略有所增加，原因是第 9 行对当前父节点子树列表进行了一次遍历以找到当前子树所在位置。但这只对系数产生影响，因此算法的大 O 复杂度不会发生变化。

5. 算法分析

1) 效果评估和时间开销

为了了解算法的特性，我整理了一批中文评论作为测试集，并用其对改进后的观点词挖掘算法进行测试。测试数据为手工整理得到，材料多来源于淘宝、京东上的产品评论，选取的句子均结构完整，语义明确，对于句中的关键词存在对应的观点词。

对于文本挖掘算法，对其效果描述的基本指标有 3 个，分别是准确率、召回率和 F 值。准确率指程序返回的正确结果占返回结果的比例；召回率指程序返回的正确结果占应返回结果的比例；F 值则是对准确率和召回率的一个调和平均。

$$\text{准确率} = \frac{\text{返回结果中提取正确的关键词数目}}{\text{有返回结果的关键词数目}}$$

$$\text{召回率} = \frac{\text{返回结果中提取正确的关键词数目}}{\text{测试集应返回结果的关键词数目}}$$

$$\text{F值} = \frac{\text{准确率}\times\text{召回率}\times 2}{\text{准确率} + \text{召回率}}$$

使用这三个指标度量文本挖掘算法，可以对其效果一目了然。另外，对算法的性能评估指标，是每秒处理的句子数目。下表是测试集合运行结果的统计情况。

表 1. 算法性能测试统计结果

序号	句子数量	召回率	准确率	F 值	运行时间	每句用时
1	50	90%	88.9%	89.45%	19556ms	391.12ms
2	100	90%	92.2%	91.09%	45674ms	456.74ms
3	150	92%	90.4%	91.19%	97113ms	647.42ms
4	200	90.5%	90.05%	90.27%	120250ms	601.25ms

从表 1.数据可以看到，当文本句子结构完整、语义明确时，观点词挖掘算法的 F 值均在 90%左右，挖掘效果优秀，但仍然存在一些算法失效的情况。

2) 错误分析

对于未能返回结果的测试样本中，错误类型多数为分词错误，如将“中科曙光”、“鞋底”拆分成“中科 曙光”、“鞋 底”。另一种错误形式是词性标注错误，如将名词形式的“做工”标注成了动词，以至于对应的提取规则找不到相应的观点词。分析结果说明，NLP 分词功能的准确率直接影响到了算法的召回率。

另外，对于提取错误的情况，错误类型主要是词性标注错误，将本应成为观点词的词性标注为名词、动词等，造成提取规则失效。结果说明，NLP 词性标注的准确率直接影响算法的准确率。

3) 性能分析

从统计的程序运行时间上看，平均每个句子处理时间为 601.25ms。在测试集 1 中，平均每个句子的处理时间最短，为 391.12ms；测试集 3 中最长，为 1028.78ms。导致程序运行速率快慢不一的主要原因是测试集中句子长短。测试集 1 中短句所占比例较大，长度在 25 字以内；而测试集 3 中的句子几乎都是超过 25 字的长句。由分析结果知，句子长度对算法性能影响较大。

（三） 挖掘模式

在文本挖掘的应用当中，会遇到各式各样的需求，因此单一用法的算法在实际使用当中有很大的局限性。本小节将着重讨论对几种文本挖掘运用场景设计的挖掘模式，以及观点词挖掘算法在该模式下的实现方法。

<句子，关键词>模式适用于给定输入句子和该句的观点词的情况；关键词表模式则对于一个文本集合，不需要为每个句子设置关键词，只需要设置一张统一的关键词表；而贪心策略模式将自动在文本中提取关键词进行挖掘。

1. <句子，关键词>模式

至此，本文讨论的程序的输入都是句子集合和对应个句子对应的关键词，然后调用挖掘算法为每个句字找到关键词对应观点词。该挖掘模式在本文统称为<句子，关键词>模式。采用这种模式对文本进行挖掘，需要对输入文本进行手工标注，找到每一个句子中想要了解的观点的关键词。这种模式下程序运行速度较快，准确率

较高。例句中，股市为关键词，低迷不振为得到的观点词。

在 **股市** **低迷不振** 的 日子 里 ， 新股 依旧 火热 。

但是，在实际运用中，显然不可能人工地对每个句子进行关键词标注，这种模式仅用于挖掘算法的性能测试。

2. 关键词表模式

一种应用更广的方法是对于输入文本集，设置一个关键词表，表中加入所有我们想要了解的关键词，调用算法前首先加载这个关键词表，然后对每一个句子的 token stream，检测每一个 token 是否在表中，若在，则调用算法挖掘其对应的观点词。最后，我们将得到表中的关键词的观点分布情况。

例如，有如下一张有关股市的词汇的关键词表，用于对“在股市低迷不振的日子里，新股依旧火热”进行观点提取，那么句中的“股市”和“新股”两词将会命中，与 token stream 一起作为观点词提取算法的参数进行挖掘。

关键词表：股市 股票 新股 行情 板块 领域 大盘 创业板 沪港通 A 股 H 股 市场

在 **股市** 低迷不振 的 日子 里 ， **新股** 依旧 火热 。

该模式比较符合人们的使用习惯。实现该模式时，为了以最快速度检测每个 token 是否出现在了关键词表中，可使用 HashMap 来存放关键词表，这样查找的时间开销为 O(1)，但相应的空间开销较大。

一个牺牲时间开销换取较小的空间开销的方法是使用字典树（Trie），在关键词较多的情况下比较适用。

对于该模式，进行效果评估测试，得到的数据如下表所示。

表 2. 关键词表模式测试统计结果

序号	句子数量	召回率	准确率	F 值	运行时间	每句用时
1	50	98.28%	77.19%	86.47%	28564ms	571.28ms
2	100	98.62%	76.28%	86.02%	48372ms	483.72ms
3	150	98.37%	75.83%	85.64%	99450ms	663.00ms
4	200	97.66%	76.80%	85.98%	127148ms	635.74ms

从表 2.中的数据可以看到，关键词表模式表现出了更高的召回率，但准确率有

所降低，这是因为对于一个句子，可能包含好几个关键词，但有的关键词本身并没有观点词修饰，这增加了算法出错的可能性；但同时由于召回的关键词数量增多，导致召回率上升。该模式 F 值在 86%左右，比起<句子，关键词>模式略有降低，。

3. 贪心策略模式

很多时候我们不知道输入文本究竟讨论了哪些对象，希望程序能够自动识别文本中的关键词，并对这些自动识别的关键词进行观点挖掘。例如对“在股市低迷不振的日子里，新股依旧火热”这句话，希望程序能够自动识别“股市”、“日子”、“新股”、“火热”等名词或形容词作为关键词，然后在该句中对这些关键词进行观点词挖掘。

在 **股市** 低迷不振 的 **日子** 里 ， **新股** 依旧 火热 。

这种策略由于将句子中每一个符合成为关键词条件的 token 都写入到了关键词表，并进行了挖掘，因此称之为“贪心策略”。这种策略下不需要进行任何额外的人工操作，但由于挖掘对象多，整个程序运行速度较慢，并且自动识别的关键词不一定存在观点词，增加了算法出错概率。在该模式下，关键词自动识别的正确率成为影响最终准确率的重要瓶颈。

该策略是在关键词表挖掘模式上做了进一步的修改。程序初始化时，关键词表为空；，没输入一个句子时，采用一定的方法识别句中的关键词，并判断该关键词是否符合词性条件：如果符合，则查询关键词表，其是否已经存在；如果不存在就将其加入关键词表；然后调用观点词挖掘算法，将返回的观点词放入 HashMap 中。

对于该模式的效果评估，主要由两个部分组成，一是对关键词的识别效果，另一个是算法的挖掘效果。目前已有的中文关键词自动识别方法较多，FNLP 也提供了关键词识别功能，本文不对此进行详述；通过关键词自动识别后，算法挖掘效果与关键词表方法效果类似。

（四） 本章小结

本章节对本文的核心内容——观点词挖掘算法进行了详细的论述，从算法原理、描述、评价等方面论证了算法的可行性。

算法的基础是 Stanford Deep Learning 方法中基于递归神经网络（RNN）建立的语法树，并由该 RNN 算法得出了结论 1。基于结论 1，设计了向上攀爬式的观点词

提取算法。

在算法设计过程中，考虑中文句子的结构特点，提出最近子树优先原则和多观点词挖掘两大改进点，经测试证明这样的改进确实对算法起到了优化作用。

最后，为了使算法能够在多种场景下得到运用，提出了三种挖掘模式的设计方法，并予以实现。

五、 基于 Hadoop 的海量文本挖掘方法实现

本文至此所讨论的都是在一台计算机中进行中文文本观点词挖掘方法，对于文本规模较小的应用来说，实现到这一步已经足够。但是，假设要将整个互联网上所有的中文文本用该方法进行挖掘，那么所耗费的时间将十分漫长。这便是海量数据挖掘方法面临的问题。

在第一章中谈到一个用于大数据挖掘的平台框架 Apache Hadoop，如果采用 Hadoop MapReduce 程序框架改写算法，便可以将程序运用到海量中文文本的挖掘中。对此，将在本章进行论述。

（一） Hadoop 概述

Hadoop 是 Apache 基金会旗下的一个开源项目，最早是 Lucene 的子项目，由于大数据处理变得越来越重要，后来独立出来成为一个单独的项目，并成为当前普遍应用的大数据处理框架和平台。

Apache Hadoop 之所以能够在大数据处理中得以广泛使用，原因在于其对数据提取、变形以及加载等方面上有着天然优势。分布式架构的设计将数据处理尽可能的靠近存储，MapReduce 过程将单个任务切分，并将这些切分后的小块任务分配到多个节点上进行处理，之后再进行集中，最终存储到数据仓库里。[18]

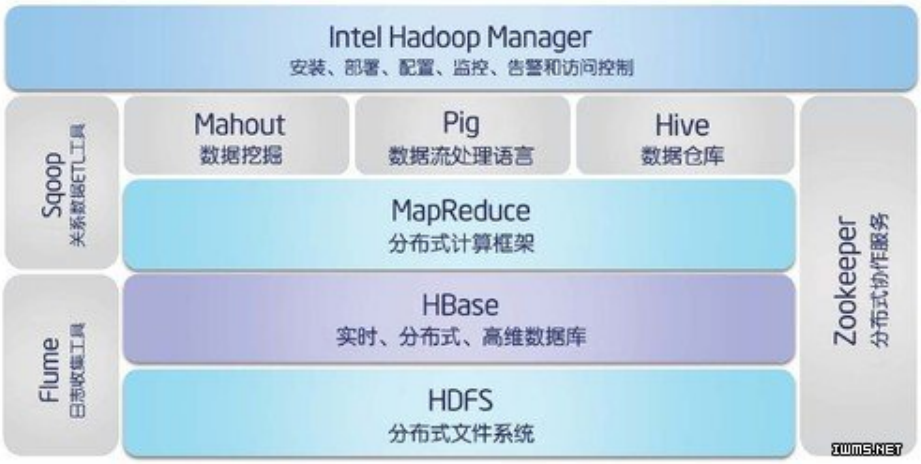

图 13. Hadoop 的架构[4]

Hadoop 平台中，最主要的组件是分布式文件系统 HDFS、MapReduce 过程、数据仓库工具 Hive 和分布式数据库 Hbase，它们共同实现了数据的读写、处理和存储功能，其中 HDFS 和 MapReduce 又是 Hadoop 分布式系统体系的核心。HDFS 在 MapReduce 任务处理过程中提供了文件操作和存储等功能支持，而 MapReduce 在 HDFS 的基础上进行了任务的分发、跟踪、执行等工作，并收集结果，二者相互作用，共同完成分布式集群的主要任务。[19]

（二） 开发环境

Apache Hadoop 早期的版本只能在 Unix/Linux 系统上运行，后来扩展到支持 Windows 环境，但是 Windows 任仅作为 Hadoop 程序的一个基本开发环境，真正使用 Hadoop 还是应该在 Unix/Linux 系统下进行开发和运行。本文中，用于运行 Hadoop 程序的操作系统为 64 位 MAC OS X 10.10.2，Hadoop 版本为 2.6.0，JDK 配置为 1.7，采用伪分布式单节点配置，在进行环境配置前，确保计算机安装好了 maven，并开启了 ssh 服务。

Hadoop 环境配置并不复杂。首先，在 Apache Hadoop 官方主页上下载 Hadoop 稳定的安装包，本文中使用到的是 hadoop-2.6.0，如果使用的操作系统是 64bit 的，那么需要下载 hadoop-2.6.0-src，用 maven 在本地进行编译；接着，将压缩包解压到用户目录下，在系统环境配置文件中加入 HADOOP_HOME，并将 HADOOP_HOME 加入到 PATH 中；然后，按官网说明修改 Hadoop 环境目录下的 4 个 xml 文件；修改完成后，格式化 hdfs namenode，并在 sbin 目录下执行 start-all.sh，并输入

[4] 图片出自《详解 Hadoop 核心架构》. http://www.thebigdata.cn/Hadoop/10973.html.

命令 jps 查看各个节点运行情况，如果都启动起来了，说明 Hadoop 配置完成。完成环境配置后，打开浏览器可以查看本地 Hadoop 集群的工作状态。

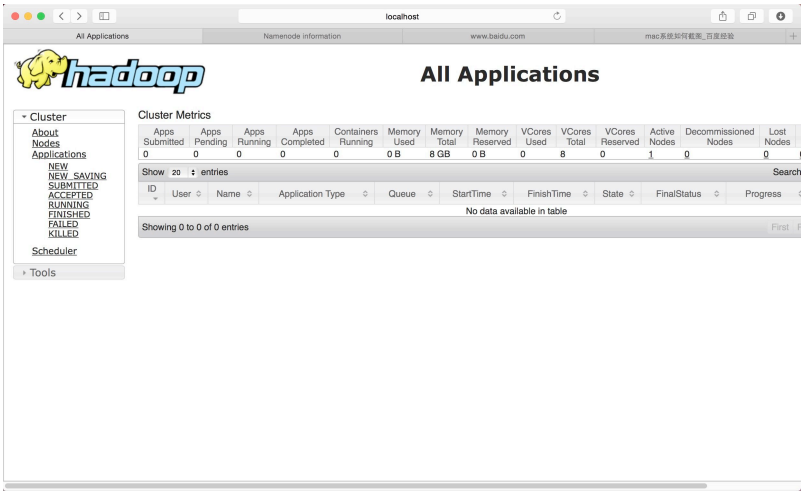

图 14. 本地 Hadoop 集群工作状态

为了方便在本地开发和调试 Hadoop 程序，还需要在 IDE 中安装 Hadoop MapReduce 插件。本文是以 eclipse 作为 IDE，方法是将编译好的插件 jar 包放在 plugin 目录下，重启 eclipse 即可。

（三） Map 和 Reduce

为了充分发挥 Apache Hadoop 所提供的并行处理优势，需要将之前整个程序改写成 MapReduce 作业的方式，经过本地小规模测试，最终在集群设备上运行。

实现一个 Hadoop 程序需要完成三个部分：map 方法，reduce 方法以及 main 方法中用来运行 job 的代码。其中 map 和 reduce 是 MapReduce 过程中两个主要阶段所对应的函数，具体实现决定了 Hadoop 的数据处理过程。每个阶段都要以<键，值>的形式作为输入和输出，并由我们决定其数据类型，map 的输出类型必须与 reduce 的输入类型保持一致，否则程序运行会出现数据不匹配问题。

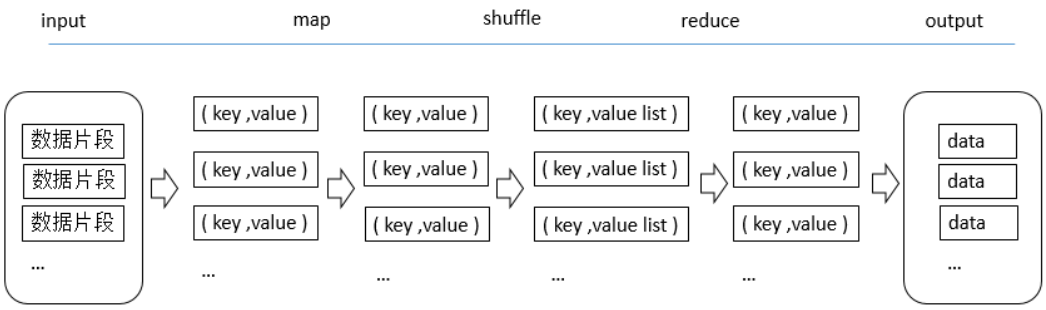

图 15. MapReduce 工作过程

map 和 reduce 两个函数需要我们进行改写以完成需要的功能。map 函数主要是对输入数据进行处理，并将结果转换成<键，值>的形式输出，reduce 函数则是将 map 的输出进行进一步处理，得到我们想要的格式并输出到结果文件中。

map 函数在 Mapper 接口声明，该接口是一个泛型，有四个形参，分别指定 map 函数的输入键、输入值、输出键和输出值的数据类型，这些数据类型均是 Hadoop 自身提供一套为了优化序列传输的基本类型，而非 Java 语言中原有的类型，但可以与 Java 原有基本数据类型进行转换。在 map 方法的最后，使用 OutputCollector 的实例用于输出内容的写入。[20]

reduce 函数在 Reducer 接口中进行定义，定义方法与 map 函数类似。整个过程的结构如下所示。

```
1   public static class MapClass
2   extends Mapper<LongWritable,Text,Text,Text>{
3       public void map(LongWritable key,Text value,
4           OutputCollector<Text,Text> output, Reporter reporter)
5           throws IOException, InterruptedException{
6       //Map 过程执行代码
7       output.cellect(key,value));
8       }
9   }

10  public static class Reduce
11  extends Reducer<Text,Text,Text,Text>{
12      public void reduce(Text key,Iterable<Text> values,
13          OutputCollector<Text,Text> output, Reporter reporter)
14          throws IOException, InterruptedException{
15          //Reduce 过程执行代码
16           output.cellect(key,value));
17      }
18  }
```

图 16 MapReduce 程序框架

（四） 采用 Hadoop 扩展观点词挖掘算法

根据 MapReduce 过程中对数据片段进行整合的原理，观点词挖掘算法采用下图所示过程进行扩展。

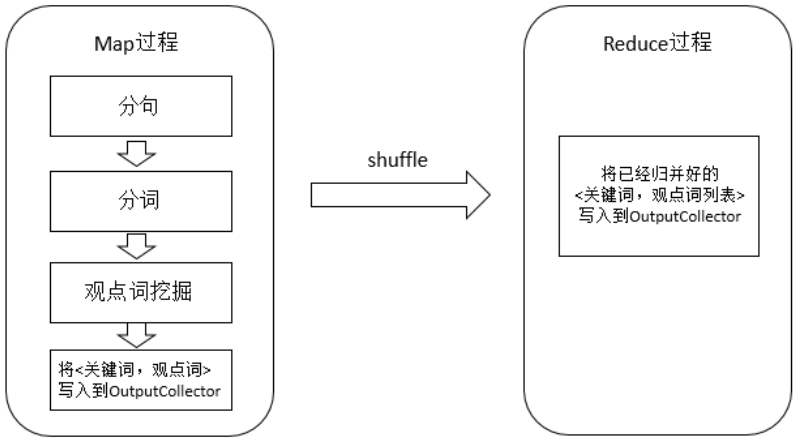

图 17． MapReduce 框架下的观点词挖掘程序流程

在输入文件中，每一行应是一段文字或一句文字，作为调用一次 map 方法的输入。Map 过程进行程序的主体功能，其输入是一段文字或一句话，执行分句、分词、观点词挖掘等任务，然后以<关键词，观点词>的格式写入到 OutputCollector 中去。接着是隐藏得 shuffle 过程，该过程将关键词相同的观点词写入到一个 list 中，作为 Reduce 过程的输入。Reduce 过程相对简单，因为经过 shuffle 过程后，相同关键词对应的观点词被集中了，因此只需要将结果写入到输出文件中即可。

main 函数的实现相对简单，实例化一个 JobConf，并设置 Job 的名称、调用的 Map 和 Reduce 类、输出形式、输出的<key,value>对应的类，以及输入输出路径，最后调用 runJob。

整个程序的实现代码见附录文档。

（五） 本章小结

本章在观点词挖掘算法实现的基础上，引入了 Apache Hadoop 对观点挖掘方法进行进一步的扩展，使其能够运用在海量中文文本的挖掘上。

Apache Hadoop 是一款开源的大数据处理框架和运行平台，在 Unitx/Linux 系统下部署应用。本文在 Mac OS X 系统下配置伪分布式环境，并进行程序的编写和调试。

Hadoop 程序最重要的是 Map 和 Reduce 两个过程对应的方法实现，map 方法将输入的数据片段进行处理，并以键值对形式进行输出；reduce 则将 map 输出后经过 shuffle 过程整理得到的<键，值列表>进一步的整合以及进行最终结果输出。利用 Hadoop 程序这一特性，将主要的挖掘方法放在 map 方法中，再通过 reduce 进行结果的输出，就可以得到每个观点词对应的观点词列表。

六、 总结和展望

（一） 总结

中文文本挖掘是当前数据挖掘的一大热点和难点，采用自然语言处理方法结合语言特征进行挖掘有助于提高挖掘效果，完成通过传统机器学习方法难以实现的任务。

本文论述了一种基于 NLP 的中文文本观点词的挖掘方法，并使用 Apache Hadoop 对其进行扩展，使之能够很好地运用到海量文本的挖掘中去。本文论述的方法的基础是 Stanford NLP 深度学习方法中基于递归神经网络算法构造的语法树，采用 FNLP 进行分词，Stanford Parser 进行语法树的建立，最后使用观点词挖掘算法进行观点挖掘。

RNN 方法建立的语法树具有相关短语优先合并的特点，因此本文的观点词挖掘算法是向上攀升式的，并采用最近子树优先和多观点词挖掘的策略。经测试，算法具有较高的准确率和召回率，适合用于中文网络评论的观点挖掘。

由于文本挖掘一般数据规模较大，传统的单机运行挖掘算法不能完全满足使用者的需求，因此本文采用 Apache Hadoop 对算法进行进一步的扩展。扩展后的算法，可以在 Hadoop 集群上良好地运行。

（二） 展望

本文算法的准确率和召回率很大程度上依赖于使用的 NLP 工具的分词和语法剖析效果，随着 NLP 技术的不断发展，今后会有更好的挖掘效果。另外，也应该注意到本文目前实现的算法对句子的容错率不高，句子结构的不完整很大程度上会影响算法的性能。在贪心策略中，句子关键词的识别率也直接影响算法的效果。

在今后的研究中，首先进行残缺句子过滤或结构自动修正的工作；其次应寻找一种准确率更高的句子关键词识别方法；最后，可以将该方法与其他文本挖掘方法结合使用，发现更大的价值。

随着中文文本挖掘技术和大数据处理技术的进一步发展，未来文本数据的利用价值更加重大。观点挖掘作为信息反馈、战略决策、舆情监控等领域的重要手段，定能发挥出更为重大的价值。

参考文献

[1] 文本挖掘 [E] 维基百科. 2015.2.21

[2] Benjamin K.Y. Tsou, Raymond W.M. Yuen, Oi Yee Kwong, Tom B.Y. Lai, Wei Lung Wong. Polarity Classification of Celebrity Coverage in the Chinese Press[C]. *In Proc. of the International Conference on Intelligence Analysis.* McLean, USA. 2005

[3] 娄德成. 基于 NLP 技术的中文网络评论观点抽取方法的研究[D].上海：上海交通大学，2007

[4] 章剑锋，张奇，吴立德，黄萱菁. 中文观点挖掘中的主观性关系抽取[J].中文信息学报. 2008，第 22 卷第 2 期

[5] 王辉，王晖昱，左万利. 观点挖掘综述[J].计算机应用研究. 2009 年，第 26 卷第一期

[6] 丁晟春，孟美任，李霄. 面向中文微博的观点句识别研究[J]. 情报学报 2014，第 33 卷第 2 期，P175-182

[7] Bill Manaris, Natural language processing: A human-computer interaction perspective [J]. Advances in Computers，1999，Volume 47

[8] 宗成庆. 统计自然语言处理[M]. 北京：清华大学出版社，2008，序 2

[9] Stanford Natural Language Processing Group . Stanford CoreNLP [E]. http://nlp.stanford.edu/software/corenlp.shtml. 2015.3

[10] Manning, Christopher D., Surdeanu, Mihai, Bauer, John, Finkel, Jenny, Bethard, Steven J., and McClosky, David. The Stanford CoreNLP Natural Language Processing Toolkit[C]. *In Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60. 2014.

[11] 向量空间模型[E]. 维基百科. 2015.1.5

[12] Christopher D. Manning， Prabhakar Raghavan， Hinrich Schtüze. 王斌（译）.信息检索导论[M]. 北京：人民邮电出版社，2010，P81-94

[13] Katrin Erk and Sebastian Padó. A Structured Vector Space Model for Word Meaning in Context[C]. *In Proceedings of EMNLP*. 2008.

[14] Richard Socher , Christopher Manning. Deep Learning for Natural Language Processing[E]. http://nlp.stanford.edu/courses/NAACL2013/ 2015.3

[15] 马锐.人工神经网络原理[M]. 北京：机械工业出版社. 2010. P17-24

[16] 大数据时代[E]. 百度百科. 2015.4

[17] 淘宝商城数据挖掘团队. 利用 Stanford Parser 进行中文观点抽取[E]. http://blog.sina.com.cn/s/blog_a16534260100z7hw.html. 2012.2

[18] 为什么 hadoop 对大数据处理的意义重大[E]. 中国大数据 http://www.thebigdata.cn/Hadoop/9064.html. 2014.3.25

[19] 详解 Hadoop 核心架构[E]. 中国大数据. http://www.thebigdata.cn/Hadoop/10973.html. 2014.7.4

[20] Tom White. 周敏奇，王晓玲，金澈清，钱卫宁（译）. Hadoop 权威指南（中文第二版）[M]. 北京：清华大学出版社. 2011. P18 - P23

[21] Danqi Chen and Christopher D Manning. A Fast and Accurate Dependency Parser using Neural Networks [C]. *Proceedings of EMNLP* 2014

[22] Richard Socher, John Bauer, Christopher D. Manning and Andrew Y. Ng. Parsing With Compositional Vector Grammars [C]. *Proceedings of ACL* 2013

[23] Pi-Chuan Chang, Huihsin Tseng, Dan Jurafsky, and Christopher D. Manning. Discriminative Reordering with Chinese Grammatical Relations Features [C]. *In Proceedings of the Third Workshop on Syntax and Structure in Statistical Translation.* 2009.

[24] Roger Levy and Christopher D. Manning. 2003. Is it harder to parse Chinese, or the Chinese Treebank? [J] ACL 2003, pp. 439-446.

附录

(一) 部分函数代码

1. 挖掘算法核心函数

```
/** <句子，关键词>挖掘模式
 * 输入：已分词句子、关键词
 * 输出：观点词
 * */
public void extraOpinion(String sentence, String keyword) {
    // TODO Auto-generated method stub
    Tree strTree = buildTree(sentence);          //建立句子的语法树
    List<Tree> leaves = strTree.getLeaves();    //获取叶子列表
    //遍历叶子列表，寻找关键词位置
    Iterator<Tree> treeIter = leaves.iterator();
    assertNewWordIn(keyword);
    while (treeIter.hasNext()) {
        Tree leaf = treeIter.next();
        //如果找到关键词
        if (leaf.nodeString().trim().equals(keyword)) {
            getOpinions(leaf,strTree,keyword);//对该关键词进行观点抽取
        }
    }
}

/**关键词表模式
 * 输入：已分词句子
 * 输出：观点词
 * */
public void extraOpinionByGroup(String sentence) {
    // TODO Auto-generated method stub
     Tree strTree = buildTree(sentence);          //建立句子的语法树
     List<Tree> leaves = strTree.getLeaves();  //获取叶子列表
    //遍历语法树，寻找关键词位置
    Iterator<Tree> treeIter = leaves.iterator();
    while (treeIter.hasNext()) {
        Tree leaf = treeIter.next();
        //该 token 在关键词表中
        if (keyWordsMap.get(leaf.nodeString().trim())!=null) {
```

```
                String keyword = leaf.nodeString().trim();
                getOpinions(leaf,strTree,keyword);  //对该关键词进行观点抽取
            }
        }
    }

    /**贪心策略模式
     * 输入：分好词的句子
     * 输出：观点词
     * */
    public void extraOpinionBySelf(String sentence) {
        // TODO Auto-generated method stub
        Tree strTree = buildTree(sentence);          //建立句子的语法树
        List<Tree> leaves = strTree.getLeaves();  //获取叶子列表
        Map<String,Integer> keys = fudan.getKeyWord(sentence, KEYNUM);
        if(keys == null) return;          //如果没有关键词就直接返回
        //遍历语法树，寻找关键词位置
        Iterator<Tree> treeIter = leaves.iterator();
        while (treeIter.hasNext()) {
            Tree leaf = treeIter.next();
            String lex = leaf.value().toString();
            if(keys.get(lex)!=null){
                //找到关键词
                String label = leaf.parent(strTree).value().toString().trim();

                //判断是否符合 4 中关键词词性
                if (label.equals("NN") || label.equals("NT")
                        || label.equals("NR") || label.equals("VA")) {
                  //是否是已经存在的关键词
                  assertNewWordIn(leaf.value());

                    getOpinions(leaf,strTree,leaf.value());
                }
            }
        }
    }

    /**语法树构造函数
     * 输入：已分好词的句子
     * 输出：句子对应的语法树
     * */
    public Tree buildTree(String sentence){
        //构造语法树
        TreebankLanguagePack pack = new ChineseTreebankLanguagePack();
```

```
        Tokenizer<? extends HasWord> tokenizer = pack.getTokenizerFactory()
                .getTokenizer(new StringReader(sentence));
        List<? extends HasWord> strList = tokenizer.tokenize();
      return lp.apply(strList);
}

/**最近子树优先策略
 * */
public boolean nearestTreeFirst(Tree strTree, Tree cur, Tree start,
          boolean extraedflg,String tag,
          List<String> points)
{
    //获得父节点的兄弟节点
      List<Tree> bros = start.siblings(strTree);
      List<Tree> all = start.parent(strTree).getChildrenAsList();
      int index = 0 ;
      Iterator<Tree> treeIter = all.iterator();
      //第一次遍历所有树，找到当前树的位置
      if (all != null) {
            while (treeIter.hasNext()) {
                Tree bro = treeIter.next();
                if(bro.equals(cur)){
                    break;
                }
                index++;
              }
      }
      //从当前树所在位置开始，每一步向两边遍历兄弟子树
      if(bros != null){
          int incre = 0;
          int total = bros.size();
          int i=0;
          //问题原因：不应该只以 flg 为标志，应该是以兄弟树为标记
          while(i<total){
              Tree follow , front;
              boolean find = false;
              //防止进入死循环
              if(incre > MAX_LIMIT) break;
              //后面的兄弟树
              if((index+incre)<bros.size()){
                  follow = bros.get(index+incre);
                  find = findOpinionWordsInTree(follow, tag, points);
                  if(find){
                      extraedflg = true;
```

```
                    }
                    i++;
                }
                //前面的兄弟树
                if(!extraedflg && (index-incre-1)>=0 && incre>0){
                    front = bros.get(index-incre-1);
                    find = findOpinionWordsInTree(front, tag, points);
                    if(find){
                        extraedflg = true;
                    }
                    i++;
                }
                incre++;
            }
            return extraedflg;
        }
        return true;
    }

    /**观点词挖掘函数
    * 由各个挖掘模式函数进行调用
    */
     public void getOpinions(Tree leaf,Tree strTree,String keyword){
         Tree start = leaf;
          start = start.parent(strTree);
          String tag = start.value().toString().trim();
          boolean extraedflg = false;
          System.out.println("关键词：\t"+keyword+"\t"+tag);
          // 如果当前节点的父节点是 NN，则遍历该父节点的父节点的兄弟节
点
          if (tag.equals("NN") || tag.equals("VA") || tag.equals("NT") ||
             tag.equals("NR") ) {
               LinkedList<String> points = new LinkedList<String>();
            for (int i = 0; i < strTree.depth(); i++) {
                Tree cur = start;
                    start = start.parent(strTree);  //向上爬一层
                    if (start.value().toString().trim().equals("ROOT")
                            || extraedflg == true) {
                        break;
                    } else {
                        boolean find =
                        nearestTreeFirst(strTree,cur,start,extraedflg,tag,points);
                       if(find)     break; //如果这一层里找到的观点词，就停止搜索
                    }
```

```
            }
            System.out.print("特征词：\t");
            for(String s:points){
                System.out.print(s+" ");
                addPointWord(leaf.value(),s);
            }
            System.out.println();
        }else if(tag.equals("NN")){
            System.out.println(tag);
        }
    }

    /**在指定子树中搜索观点词
     * 递归函数
     * 输入：一棵当前子树的兄弟子树、关键词词性标记、观点词列表
     * 输出：是否在本棵兄弟子树中找到观点词标记
     * */
    private boolean findOpinionWordsInTree(Tree bro, String tagKey, List<String>
                                                    points) {
        List<Tree> subTree = bro.getChildrenAsList();
        Iterator<Tree> treeIter = subTree.iterator();
        boolean find = false;
        while (treeIter.hasNext()) {
            Tree end = treeIter.next();
            String opTag = end.value().toString().trim();
            //抽取规则：  key 与 opinion 间的关系
            if ( ( tagKey.equals("NN") || tagKey.equals("NT") ||
                    tagKey.equals("NR") ) && ( opTag.equals("VA")||
                    opTag.equals("VV") || opTag.equals("JJ"))
                      || ( tagKey.equals("VA") && opTag.equals("AD"))    ) {
                Tree opTree = end.getChild(0);
                points.add(opTree.value().toString());
                find = true;
            } else if (findOpinionWordsInTree(end, tagKey, points)) {
                return true;
            }
        }
        return find;
    }
```

2. Hadoop 程序代码

```
public class HadoopMain {

    public static class Map extends MapReduceBase implements
            Mapper<LongWritable, Text, Text, Text> {
        private Text opinion = new Text();
        private Text word = new Text();
        private FudanTokennizer tokennizer = new FudanTokennizer();
        private GetChineseOpinion mgco    = new GetChineseOpinion();

        public void map(LongWritable key, Text value,
                OutputCollector<Text, Text> output, Reporter reporter)
                throws IOException {
            String line = value.toString();
            String ret = tokennizer.processSentence(line);
            System.out.println(ret);
            mgco.extraDepWordBySelf(ret);
            List<String> keys = mgco.getKeywordsList();
            HashMap<String,List<String>> map = mgco.getKeywordsMap();

            for(String keyword:keys){
                word.set(keyword);
                List<String> opinions = map.get(keyword);
                for(String s:opinions){
                    if(s==null)
                        opinion.set(" ");
                    else
                        opinion.set(s);
                    output.collect(word,opinion);
                }
            }
            mgco.clear();
        }
    }

    public static class Reduce extends MapReduceBase implements
    Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterator<Text> values,
                OutputCollector<Text, Text> output, Reporter reporter)
                throws IOException {
          StringBuilder builder = new StringBuilder();
          while (values.hasNext()) {
```

```
                builder.append(values.next().toString()+",");
            }
            Text result = new Text();
            result.set(builder.toString());
            output.collect(key,result);
        }
    }

    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(HadoopMain.class);
        conf.setJobName("hadoop_main");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(Text.class);
        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);
    }
}
```

(二) Stanford Parser 助记符含义对应表

表格 2 Stanford Parser 助记符含义对应表

单词		短语	
助记符	对应单词	助记符	对应短语
NN	普通名词	NP	名词短语
NR	命名实体	VP	动词短语
PN	代词	PU	断句符，通常是句号、问号、感叹号等标点符号
VV	动词	LCP	方位词短语
VC	是	PP	介词短语
CC	表示连词	CP	由‘的’构成的表示修饰性关系的短语
VE	有	DNP	由‘的’构成的表示所属关系的短语

VA	表语形容词	ADVP	副词短语
NT	时间名词	ADJP	形容词短语
AS	内容标记（如：了）	DP	限定词短语
VRD	动补复合词	QP	量词短语
CD	表示基数词		
DT	表示限定词		
EX	存在句		
FW	外来词		

致谢

时光如白驹过隙，转眼我的大学生活已接近尾声。在华东师范大学软件学院学习的四年时光里，许多人给予了我重要帮助，在这里，我要向他们表示诚挚的谢意。

首先，我要感谢我的论文指导老师。不论是在数学馆参与实验的两年还是进行毕业设计的半年，郭建老师总是细致、耐心地予以指导，并给与我高度的肯定。郭老师利用自己不多的空余时间悉心指导写作此文，对每一处细节都认真把关，最终得以完成此文。

其次，我要感谢软件学院里给予我帮助的老师，在老师们的悉心教授下才有了我的成长。特别是陈良育老师和钱卫宁老师，陈老师的教学风格激发了我实践的潜能，而钱老师开阔的视野吸引我接触了很多新鲜的知识。

然后，要感谢我在上海惠普有限公司全球 IT 部门实习期间的项目经理刘晔和导师史杰，以及其他同事。在 HP 的半年工作中，他们给了我充足的时间和空间学习机器学习、大数据处理和自然语言处理方面的知识，并给了我大量的指导和建议，也高度肯定了我在实习工作中的表现，最终我才能将顺利将本课题作为毕业设计。

另外，还要感谢周天佑同学为我提供的一些关于 Hadoop 开发环境方面的支持，感谢我的室友崔昕、沈崮和赵思远四年间的关心和照顾，同时也感谢华东师范大学软件学院其他给予我帮助和关心的同学。

最后，我要感谢我的父母，在远离家乡的上海读书的四年中，我的父亲和母亲含辛茹苦地供我完成了本科学业，时时刻刻牵挂着在外的游子，让我即使离家千里，也能感受到家的温馨。值此论文完成之际，特向他们表示由衷的感谢。