第 5 章

自然语言处理技术实例

自然语言是以语音为物质外壳，由词汇和语法两部分组成的符号系统。《新华词典》中对语言的定义是人类交际的工具，是人类思维的载体，是约定俗成的，有别于人工语言（程序设计语言）。自然语言处理（Natural Language Processing，NLP）是用机器处理人类语言的理论和技术，它主要研究在人与人交际中以及人与计算机交互中的语言问题的一门学科。它主要研究表示语言能力和语言应用的模型，建立计算框架进行功能实现，提出完善和评测模型的方法，并依此设计各种实用系统。

本章主要介绍自然语言处理方面的核心技术，其中主要是文本处理相关技术，例如分词、词性标记、情感分析、语言模型、语义角色标记等。

5.1 业务背景分析

某电商网站需要实现机器自动生成商品的推荐标题和推荐语。具体要求是根据一个商品的标题及商品的详细描述信息，自动生成推荐语的标题和推荐内容，推荐语的内容字数要求在 50～90 字之间，要求生成的文字语言通顺，语义准确。推荐标题不可照抄产品原标题，标题字数应控制在 6～10 字，将商品主体及特点描述清楚；不能是关键词堆砌；标题内不允许含有除逗号、书名号（中英格式）之外的其他符号。必须和实物完全对应，元素可包含以下方面：品牌（中英文品牌最多选其一，选择较为熟知的）、产地、形容词、品类名称，必须清晰展示商品的品类名词；不能使用品类通用格式作为标题，如车载蓝牙耳机，要突出商品的其他特性。标题中不得使用营销类和主观类短句，例如：旅行好伙伴、超好用的充电宝。对于推荐内容的要求也比较多，其中主要是字数应在 50～90 字之间，符合广告法文案规范；不可出现违禁、敏感字眼；不可出现负面情感指向；不可直接抄袭商品详情；不可堆砌百搭关键词（如：高端、必备、大牌），而无实际导购价值；不可出现带有时效类或促销类的信息，如：情人节必备、新品上市、包邮、买一送一、满 100 减 10 等。

5.2 分析框架

自然语言处理属于文本挖掘的范畴，包括文本分类、自动摘要、机器翻译、自动问答、阅读理解等，本案例涉及自然语言生成方面的相关技术，其基础则是自然语言理解。自然语言处理涉及的具体内容说明如下。

1. 分词

分词(Word Segmentation)主要是基于词典对词语识别，最基本的方法是最大匹配法(MM)，效果取决于词典的覆盖度。此外，常用基于统计的分词方法，基于语料库中的词频和共现概率等统计信息对文本进行分词。对切分歧义的消解的方法包括句法统计和基于记忆的模型，前者将自动分词和基于马尔可夫链词性自动标注结合起来，利用从人工标注语料库中提取出的词性二元统计规律来消解切分歧义，而基于记忆的模型，对机器认为有歧义的常见交集型歧义切分，如“辛勤劳动”切分为“辛勤”“勤劳”“劳动”，并把它们的唯一正确切分形式预先记录在一张表中，其歧义消解通过直接查表实现。

2. 词性标注

词性标注(Part-of-speech Tagging)是对句子中的词标记词性，如动词、名词等。词性标注本质上是对序列中各词的词性进行分类判断，所以早期用隐马尔科夫模型进行标注，以及后来出现的最大熵、条件随机场、支持向量机等模型。随着深度学习技术的发展，出现了很多基于深层神经网络的词性标注方法。

3. 句法分析

在句法分析时，人工定义规则费时费力，并且维护成本较高，在近几年，自动学习规则的方法是句法分析的主流方法，目前主要是应用数据驱动的方法进行分析。通过在文法规则中加入概率值等统计信息(如词共现概率)，从而实现对原有的上下文无关文法分析方法进行扩展，最终实现概率上下文无关文法(Probabilistic Context Free Grammar，PCFG)分析方法，在实践中取得较好效果。句法分析主要分为依存句法分析、短语结构句法分析、深层文法句法分析和基于深度学习的句法分析等。

4. 自然语言生成

自然语言生成(Natural Language Generation，NLG)主要难点在于从知识库或逻辑形式等方面需要进行大量基础工作，人类语言系统中又存在较多的背景知识，而机器表述系统中一方面较难将背景知识集成(信息量太大)。另一方面，语言在机器中难以合理表示，所以目前自然语言生成的相关成果较少。

现在的自然语言生成方法大多是用模板，模板来源于人工定义、知识库，或从语料库中进行抽取，这种方式生成的文章容易出现僵硬的问题。目前也可以用神经网络生成序列，如Seq2Seq、GAN 等深度学习模型等，但由于训练语料的质量各异，容易出现结果随机且不可控。

自然语言生成的步骤包括内容规划、结构规划、聚集语句、选择字词、指涉语生成、文本生成等几步，目前比较成熟的应用主要还是一些从数据库或资料集中通过摘录来生成文章的系统，例如一些天气预报生成、财经新闻或体育新闻的写作、百科写作、诗歌写作等，这些文章本身具有一定的范式，类似八股文一样具有某些固定的文章结构，语言的风格变化较少。此外，此类文章重点在于其中的内容，读者对文章风格和措辞等要求较低。综合来看，目前人工智能领域中，以自然语言生成的难题还未真正解决，可谓“得语言者得天下”，毕竟语言也代表着较高级的人类智能。

5. 文本分类

文本分类(Text Categorization)是将文本内容归为某一类别的过程，目前对其研究层出不穷，特别是随着深度学习的发展，深度学习模型在文本分类任务上取得了巨大进展。文本分类的算法可以划分为以下几类：基于规则的分类模型、基于机器学习的分类模型、基于神经网络的方法、卷积神经网络(CNN)、循环神经网络(RNN)。文本分类技术有着广泛的应用。例如，社交网站每天都会产生大量资讯内容，如果由人工对这些文本进行整理将会费时费力，且分类结果的稳定性较差，而应用自动化分类技术可以避免上述问题，从而实现文本内容的自动化标记，为后续用户兴趣建模和特征提取提供基础支持。除此之外，文本分类还作为基础组件用于信息检索、情感分析、机器翻译、自动文摘和垃圾邮件检测等。

6. 信息检索

信息检索(Information Retrieval)是从信息资源集合中提取需求信息的行为，可以基于全文或内容的索引，目前在自然语言处理方面，信息检索用到的技术包括向量空间模型、主题提取、TF-IDF(词频-逆向文档频率)词项权重计算、文本相似度计算、文本聚类等。具体的应用于搜索引擎、推荐系统、信息过滤等方面。

7. 信息抽取

在信息抽取(Information Extraction)方面，从非结构化文本中提取指定的信息，并通过信息归并、冗余消除和冲突消解等手段将非结构化文本转换为结构化信息。应用方向很多，例如从相关新闻报道中抽取出事件信息：时间、地点、施事人、受事人、主要目标、后果等；从体育新闻中抽取体育赛事信息：主队、客队、赛场、比分等；从论文和医疗文献中抽取疾病信息：病因、病原、症状、药物等。还广泛应用于舆情监控、网络搜索、智能问答等领域。与此同时，信息抽取技术是中文信息处理和人工智能的基础核心技术。

8. 文字校对

文本校对(Text-proofing)应用的领域主要是对自然语言生成的内容进行修复或对OCR识别的结果进行检测和修复，采用的技术包括应用词典的方式和语言模型等，其中词典是将常用词以词典的方式对词频进行记录。如果某些词在词典中不存在，则需要对其进行修改，选择最相近的词语进行替换，这种方式对词典要求高，并且在实际操作中，由于语言的变化且存在较多组词方式，导致很多误判，在实际应用中准确性不佳。而语言模型是基于词汇之间搭配的可能性(概率)来对词汇进行正确性判断，一般是以句子为单位对整个句子

进行检测，目前常见的语言模型有 SRILM 和 RNNLM 等几种。

9. 问答系统

自动问答(Question Answering)系统在回答用户问题之前，第一步需要能正确理解用户用自然语言提出的问题，这涉及分词、命名实体识别、句法分析、语义分析等自然语言理解相关技术。然后针对提问类、事实类、交互类等不同形式的提问分别应答，例如用户提问类问题，可从知识库或问答库中检索、匹配获得答案，除此之外还涉及对话上下文处理、逻辑推理、知识工程和语言生成等多项关键技术。因此，问答系统代表了自然语言处理的智能处理水平。

10. 机器翻译

机器翻译(Machine Translation)是由机器实现不同自然语言之间的翻译，涉及语言学、机器学习、认知语言学等多个学科。目前基于规则的机器翻译方法需要人工设计和编纂翻译规则，而基于统计的机器翻译方法能够自动获取翻译规则，最近几年流行的端到端的神经网络机器翻译方法可以直接通过编码网络和解码网络自动学习语言之间的转换算法。

11. 自动摘要

自动摘要(Automatic Summarization)主要是为了解决信息过载的问题，用户阅读文摘即可了解文章大意。目前常用抽取式和生成式两种摘要方法，其中抽取式方法是通过对句子或段落等进行权重评价，按照重要性进行选择并组成摘要。而生成式方法除了利用自然语言理解技术对文本内容分析外，还利用句子规划和模板等自然语言生成技术产生新句子。传统的自然语言生成技术在不同领域中的泛化能力较差，随着深度学习的发展，生成式摘要应用逐渐增多。目前主流还是采用基于抽取式的方法，因为这一方法易于实现，能保证摘要中的每个句子具有良好的可读性，并且不需要大量的训练语料，可跨领域应用。

自然语言的处理过程如图 5.1 所示。在一般情况下，原始文本需要经过分词、去停用词、词形归一化等先期处理，其中词形归一化也称为词干化，一般用于英语等语言的处理中。经过分词之后，则以词为单位对其进一步分析处理，例如词性标注、句法分析、语义分析等，这部分工作是自然语言处理的基础，也是容易被忽视的工作，属于处理工作量较多，但是没有太多成效可供展示的工作，但是它又是后期模型分析的基础，例如，提高分词的质量可能比改进模型超参对于提高模型性能更加显著。

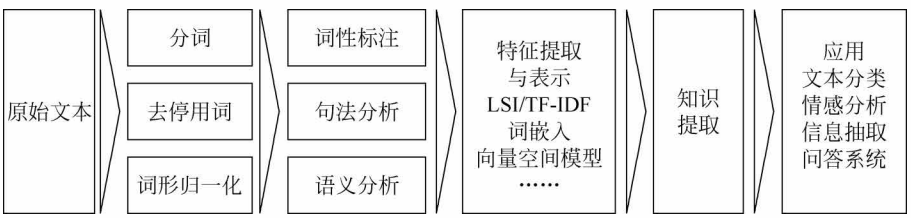

图 5.1　自然语言处理的一般流程

经过前述处理之后，可进行特征提取与表示、主题词提取、词嵌入、向量空间模型，这属于自然语言处理中的模式创新领域，可以有较多成果和应用创新，也是目前自然语言处理方

面的主要研究方向。而知识提取及文本分类、情感分析、信息抽取、问答系统等具体应用则处于成果收获阶段，随着深度学习的不断深入，在这一方面的算法越来越多，性能也越来越强。

本节主要阐述分词、语义角色标记、文本特征提取与表示、文本分类应用等基本技术，并说明其在项目中的应用。

5.3　数据收集

目前该电商网络推荐标题和推荐语由人工生成。写手数量几百人，每天发布文章量几千篇，每天的阅读量在70万以上，已经审核通过超过30万条，覆盖类别50%以上。机器生成推荐语需要的数据如下：商品标题、三级分类类别、广告语、商品规格属性、商品描述(OCR识别结果和发布系统中商家录入的描述信息)、评价内容、评价标签、推荐好货内容、查看量、商品销量等。

5.4　建立模型

为了实现自然语言生成，需要先收集较多的语料素材，对其进行加工处理，然后建立生成模型，并对生成的句子进行评价，不仅可以反向优化生成模型，也可进行预审，提高审核通过率。

5.4.1　文本分词

词法分析目的是找出词汇的各个词素，从中获得语言学信息。而分词主要应用于中文处理中，因为中文词与词之间没有明显的分隔符，使得计算机对于词的准确识别非常困难。因此，分词就成了中文处理中所要解决的最基本的问题，分词的性能对后续的语言处理，如机器翻译、信息检索等有着至关重要的影响。

主流分词方法：有基于词典、基于统计和基于规则的方法，其中基于词典的方法又包括最大正向匹配法(Maximum Matching Method，MM)和逆向最大匹配法(Reverse Maximum Matching Method，RMM)。基于统计的方法主要是从统计概率的角度全切分，它与词典的方式相比，需要语料进行训练，对于较长的句子，分词速度较慢，但是效果较好，主要从概率角度进行统计分析，例如隐马可夫链、N-Ggram等，并基于语言模型进行结果修正。基于规则的方法主要是抽取句法、语法、语义等规则，这一方法比较生硬，对于源语句的质量要求高，难以应付目前的网络语言，所以其应用较少。

常见的中文分词工具或软件有以下几种：

(1) SCWS简单中文分词系统。

(2) IK Analyzer开源轻量级中文分词工具包。

(3) ICTCLAS中文词法分词系统。

(4) jieba分词系统。

(5) 斯坦福分词系统。

(6) HanLP分词系统。

(7) 哈尔滨工业大学分词器。

(8) THULAC 分词系统。

首先对物品标题进行分词以提取其中的关键特征,本节采用 jieba 分词组件,代码如下:

```
import jieba
import jieba.analyse
import csv
import re
in_debug = True
```

其中采用 csv 组件将预先提取的物品标题信息进行加载,re 组件是正则表达式在 Python 实现,in_debug 是自定义的调试信息输出标志,将其置为 True 可将中间结果输出,方便进行代码调试。

由于标题和商品详情均由电商平台上的各个商家自行编辑和上传,所以标题及商品详情中均可能包含一些特殊符号,采用如下正则化的方式将此类特殊符号移除。

```
def remove_punc(line_sentence):
    multi_version = re.compile("-\{.*?(zh-hans|zh-cn):([^;]*?)(;.*?)?\}-")
    punctuation = re.compile("[-~!@#$%^&*()_+`=\[\]\\\{\}\"|;':,./<>?·!@#¥%
……&*()——+【】、;':"",.、«»?「『」』]")
    line = multi_version.sub(r"\2", line_sentence)
    line = punctuation.sub('', line_sentence)
    return line
```

在自然语言处理过程中,一般要先过滤掉某些字或词,也就是停用词(Stop Words),一般是原始文本集不需要的词汇、字符。虽然有通用的停用词表,但是如果想提高后续的分词效果,还要结合实际业务进行自建,下面的代码是加载自定义的词表。

```
def get_stop_words_set(file_name):
    with open(file_name,'r') as file:
        return set([line.strip() for line in file])

def load_words_list(stop_word_file):
    global stop_words_set
    stop_words_set = get_stop_words_set(stop_word_file)
    if in_debug:
        print("共计导入 %d 个停用词" % len(stop_words_set))
```

其中,stop_words_set 是全局变量,这些自定义的停用词以文本形式存在记事本中,每一行是一个词语,读取此文件并加载到内存中,方便后续使用。

定义句子分词的通用函数 cut_words,代码如下,函数的输入参数是已清理过特殊符号的完整句子,输出的是这一句话经过分词(jieba.cut)之后的词语列表。

```
def cut_words(sentence):
    word_list =[]
    words = jieba.cut(sentence)
    for word in words:
        if word in stop_words_set:
            if in_debug:
```

```
                print("ignore words:" + word)
            continue
        if len(word.strip())> 0:
            word_list.append(word)
    return word_list
```

其中，依次遍历各个分词的结果词，如果它属于停用词列表，则将其排除，并输出到日志中。通过使用 word.strip()方法将空格进行过滤，避免在结果列表中包含空格字符。

现在开始处理物品的标题，代码如下：

```
jieba.load_userdict("dict.txt.big.txt")
stop_word_file = "stopwords.txt"
jieba.analyse.set_stop_words(stop_word_file)
load_words_list(stop_word_file)
```

其中，通过 jieba.load_userdict 方法加载自定词的词表，此词表的格式是：词语 词频 词性，例如“A 股 3 n”表示 A 股属于一个独立的词的可能性是 3，其词性为名词，词频越大，在分词时越可能将其分为独立的一个词。调用前面定义的 load_words_list 方法加载停用词词表。

通过如下代码从 csv 中读取标题列表，将其存入 item_titles 列表对象中。

```
with open('item_titles.csv', 'r') as f:
    reader = csv.reader(f)
    item_titles = list(reader)

result_titles = []
for item in item_titles:
    result_titles.append(" ".join(cut_words(item[1])))
result_titles
```

然后，依次遍历各个标题，对其进行停用词过滤，输出的调试信息如下：

```
ignore words:9
ignore words:2
ignore words:4
ignore words:2
…
ignore words:8
ignore words:6
```

可以看到大部分都是数量词，这部分在文本分析时需要进行过滤。将原始标题输出：

```
[['3151888', '金立 S6 Pro 玫瑰金 移动联通电信 4G 手机 双卡双待'],
['5181386', '荣耀 9 全网通 尊享版 6GB + 128GB 魅海蓝 移动联通电信 4G 手机 双卡双待'],
['5239538', 'OPPO R11 Plus 6GB + 64GB 内存版 全网通 4G 手机 双卡双待 金色'],
['2910505', '金立 S6 Pro 耀金 移动联通电信 4G 手机 双卡双待'],
['4723112', 'vivo X9Plus 全网通 6GB + 64GB 星空灰 移动联通电信 4G 手机 双卡双待'],
['10080895816', '小米 Max 手机双卡双待 金色 全网通 4G(3G RAM + 32G ROM)标配'],
['1413846', '努比亚(nubia)【2 + 16GB】小牛 4 Z9mini 黑色 移动联通电信 4G 手机 双卡双待'],
['2876320', '三星 Galaxy S7 edge(G9350)4GB + 64GB 铂光金 移动联通电信 4G 手机 双卡双待'],
['10182955056', '保千里打令 D8 至尊版 64G 香槟金 全网通 4G 空间拍摄 VR 智能手机 双卡双待 香槟金'],
['10399054644', 'OPPO R11plus 手机 4G 全网通 6G RAM + 64 GROM 双卡双待 玫瑰金色']]
```

其中，第一列是物品的编号，第二列是标题，调用 cut_words(item[1])之后，输出分词后的标题，前 10 条结果如下：

```
['金立 S6 Pro 玫瑰 移动 联通 电信 4G 手机 双卡 双待',
'荣耀 网通 尊享 6GB 128GB 魅海 移动 联通 电信 4G 手机 双卡 双待',
'OPPO R11 Plus 6GB 64GB 内存 网通 4G 手机 双卡 双待 金色',
'金立 S6 Pro 耀金 移动 联通 电信 4G 手机 双卡 双待',
'vivo X9Plus 网通 6GB 64GB 星空 移动 联通 电信 4G 手机 双卡 双待',
'小米 Max 手机 双卡 双待 金色 网通 4G 3G RAM 32G ROM 标配',
'努比亚 nubia 16GB 小牛 Z9mini 黑色 移动 联通 电信 4G 手机 双卡 双待',
'三星 Galaxy S7 edge G9350 4GB 64GB 光金 移动 联通 电信 4G 手机 双卡 双待',
'千里 打令 D8 至尊版 64G 香槟金 网通 4G 空间 拍摄 VR 智能手机 双卡 双待 香槟金',
'OPPO R11plus 手机 4G 网通 6G RAM 64 GROM 双卡 双待 玫瑰 金色']
```

可以看到大部分词语已经分离成功，如果发现某一些词汇分词不准确，可将正确的分词结果加入自定义词典中进行修正。

5.4.2 主题词提取

商品的标题一般会比较着重展示商品的重要特点，为了提取待写商品的关键词，需要从标题中提取关键词，在提取主题词的方法中主要有 TextRank、文档主题生成模型（Latent Dirichlet Allocation，LDA）、词频-逆文件频率（Term Frequency-Inverse Document Frequency，TF-IDF）等几种方法。

首先采用 jieba.analyse.textrank 方法提取标题中的主题词，代码如下：

```
def textrank_words(line):
    line = remove_punc(line)
    line = line.strip()
    if len(line) < 1: return ""
    line_words = ""
    for word,x in jieba.analyse.textrank(line.strip(), withWeight = True, allowPOS = ('n', 'vn')):
        if word.strip() == "": continue
        line_words = line_words + (word + " ")
    return line_words
```

其中，jieba.analyse.textrank 方法的参数 withWeight＝True 是表示在提取关键词时输出词的权重值，allowPOS 是对输出结果进行词性过滤，本例中是只提取词性为名词和动名词的词语，通过如下代码进行调用。

```
result_titles = []
for item in item_titles:
    rank_result = textrank_words(item[1])
    if len(rank_result)> 0: result_titles.append(rank_result)
result_titles[:10]
```

执行之后的结果如下：

```
['移动 电信 玫瑰 手机 ',
'移动 电信 魅海 手机 ',
'网通 内存 手机 ',
```

```
'电信 耀金 移动 手机 ',
'移动 电信 星空 手机 ',
'金色 手机 网通 小米 ',
'电信 黑色 移动 手机 ',
'电信 移动 光金 手机 ',
'网通 空间 香槟金 智能手机 ',
'金色 玫瑰 ']
```

可以看到，其中大部分商品的输出词雷同率较高，采用上述关键词很难将商品之间的差异性区分出来，所以此提取主题词的方法并不适用于这种场景。

下面尝试使用 LDA 方法，代码如下，首先引入自然语言处理工具包 gensim，LdaModel 位于 gensim. models 模块中。

```
from gensim.models import LdaModel
from gensim.corpora import Dictionary
from gensim import corpora, models
```

然后，遍历前述处理过的 item_titles 中的每一个标题，组成一个标题数组 titles，代码如下：

```
titles = [item[1] for item in item_titles]
for title in titles:
    title_words_list = [cut_words(title)]
    dictionary = corpora.Dictionary(title_words_list)
    corpus = [ dictionary.doc2bow(title) for title in title_words_list ]
    lda = LdaModel(corpus = corpus, id2word = dictionary, num_topics = 2)
    print(lda.print_topics(num_topics = 2, num_words = 2))
```

对每个标题都进行分词，并将分词的结构作为词的数组，传入 corpora. Dictionary 构造函数中，建立词典，通过 dictionary. doc2bow 方法将此词典建立语料素材 corpus，将语料素材和词典作为 LdaModel 的参数传入，并设置主题数量为 2 个，即构造了一个 LDA 主题模型，通过 lda. print_topics 方法将每个商品的 2 个主题，每个主题中限制主题词的数量为 2 个进行输出，前 10 个标题的主题词分别如下：

```
[(0, '0.097 * "电信" + 0.095 * "S6"'), (1, '0.098 * "联通" + 0.097 * "玫瑰"')]
[(0, '0.082 * "电信" + 0.081 * "尊享"'), (1, '0.086 * "双卡" + 0.082 * "4G"')]
[(0, '0.090 * "网通" + 0.089 * "OPPO"'), (1, '0.097 * "金色" + 0.093 * "手机"')]
[(0, '0.102 * "S6" + 0.099 * "电信"'), (1, '0.098 * "移动" + 0.097 * "双卡"')]
[(0, '0.082 * "电信" + 0.082 * "双卡"'), (1, '0.085 * "4G" + 0.081 * "移动"')]
[(0, '0.088 * "4G" + 0.082 * "Max"'), (1, '0.083 * "双待" + 0.082 * "小米"')]
[(0, '0.079 * "努比亚" + 0.079 * "双待"'), (1, '0.086 * "双卡" + 0.080 * "联通"')]
[(0, '0.071 * "三星" + 0.068 * "S7"'), (1, '0.073 * "64GB" + 0.071 * "4G"')]
[(0, '0.103 * "香槟金" + 0.075 * "智能手机"'), (1, '0.104 * "香槟金" + 0.082 * "VR"')]
[(0, '0.083 * "金色" + 0.082 * "GROM"'), (1, '0.083 * "手机" + 0.083 * "OPPO"')]
[(0, '0.089 * "双待" + 0.083 * "Plus"'), (1, '0.082 * "标准版" + 0.080 * "手机"')]
```

可以看到，其中排名靠前的主题词也没有将商品中明显的主题特征提取出来。这主要是因为输入到 LDA 模型中的内容太短，LDA 属于概率统计模型，从 10 几个字的标题中很难将其主题提取出来，一般情况下，只有在字数超过 500 字以上的文本中采用 LDA 算法实

现主题词提取。

下面接着尝试 TF-IDF 算法，首先引入 scikit-learn 库中的 CountVectorizer 和 TfidfTransformer 类，代码如下：

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import re

result_titles = []
for item in item_titles:
    result_titles.append(" ".join(cut_words(item[1])))
```

将所有的商品标题进行分词后，构造成一个标题列表，作为语料集合。初始化 CountVectorizer 对象，并将语料集进行训练转换，代码如下：

```
cv = CountVectorizer(max_df = 0.8,stop_words = stop_words_set, max_features = 100, ngram_range
 = (1,3))
X = cv.fit_transform(result_titles)
```

其中，max_df 是一个防止某些高频词对模型产生影响的阈值，max_features 决定了特征列的数量，其值越大，计算量也就越大。ngram_range 是 n 元组的数量范围，一般不要超过 3 元组。经过 fit_transform 之后，得到整体样本集合的稀疏矩阵，它由 n 行和 100 列(max_features)组成。构造一个 TfidfTransformer 对象，并指定其使用平滑 IDF(smooth_idf)，经过 fit 之后即可获得相应特征，代码如下：

```
tfidf_transformer = TfidfTransformer(smooth_idf = True,use_idf = True)
tfidf_transformer.fit(X)
feature_names = cv.get_feature_names()
for title in result_titles[:10]:
    tf_idf_vector = tfidf_transformer.transform(cv.transform([title]))
    sorted_items = sort_coo(tf_idf_vector.tocoo())
    keywords = extract_topn_from_vector(feature_names,sorted_items,3)
    str_keywords = [k + " " + str(keywords[k]) for k in keywords]
    print(str_keywords)
```

针对前 10 个商品的标题，通过 tfidf_transformer 进行转换，获得其 tf_idf_vector 对象，经过 sort_coo 排序之后提取其中前 n 个主题词(目前为 3 个)，并将主题词与对应的权重值输出，结果如下：

```
['金立 0.474', '玫瑰 0.418', 'pro 0.404']
['荣耀 0.426', '6gb 128gb 0.394', '128gb 0.358']
['网通 4g 手机 0.386', '6gb 64gb 0.368', '内存 0.354']
['金立 0.522', 'pro 0.445', '电信 4g 手机 0.224']
['网通 6gb 0.41', 'vivo 0.41', '6gb 64gb 0.392']
['金色 网通 0.357', '3g ram 0.357', '3g 0.34']
['小牛 0.382', '16gb 0.382', '黑色 0.326']
['三星 galaxy 0.401', 'galaxy 0.401', '三星 0.383']
['香槟金 0.706', '智能手机 双卡 双待 0.353', '智能手机 双卡 0.353']
['6g 0.514', '玫瑰 0.454', 'oppo 0.454']
```

可以看到通过 TF-IDF 方法获得标题主题词的方式与前面两种方式相比，已经具有较强的差异化和区分度，以这种方式进行推荐标题和推荐语的生成，将具有良好的基础。

5.4.3　情感分析

在推荐语的审核要求中，明确说明了对于生成的推荐内容不可以有负面情绪，所以需要实现一个文本分类模型，用于对素材句子的情感分析，将负面情感滤除。目前有两个技术方案，使用基于贝叶斯网络的 SnowNLP 算法和卷积神经网络(CNN)的文本分类算法。

SnowNLP 是一个自然语言处理的套件，它除了用于文本分类的情感分析之外，还具备分词、文本摘要、词性标记、文字转拼音、繁体转简体、主题词提取及句子相似度计算等功能模块，使用 pip3 install snownlp 即可实现安装。

SnowNLP 中情感分析类的核心定义如下：

```
class Sentiment(object):
    def __init__(self):
        self.classifier = Bayes()
    def save(self, fname, iszip=True):
        self.classifier.save(fname, iszip)
    def load(self, fname=data_path, iszip=True):
        self.classifier.load(fname, iszip)
    def handle(self, doc):
        words = seg.seg(doc)
        words = normal.filter_stop(words)
        return words
    def train(self, neg_docs, pos_docs):
        data = []
        for sent in neg_docs:
            data.append([self.handle(sent), 'neg'])
        for sent in pos_docs:
            data.append([self.handle(sent), 'pos'])
        self.classifier.train(data)
```

其中，self.classifier 为 Bayes()，训练时只需要准备两份素材文件，其中一份是正面的商品评论，另一份为负面的情感评论内容，每一行是一条评论，评论内容不需要分词，在 SnowNLP 中会调用 handle 方法进而调用 seg.seg 进行分词，且去除停用词(filter_stop)，最后调用 Bayes 类的 train 方法即可完成训练。

在使用 SnowNLP 时，只需要将 SnowNLP 中的 sentiment 引入调用程序，代码如下：

```
from snownlp import SnowNLP
from snownlp import sentiment
if __name__ == "__main__":
    is_train = False
    if is_train:
        #训练模型
        sentiment.train("neg.txt","pos.txt")
        sentiment.save('phone_sentiment_model.marshal')
    else:
        # 测试模型
```

```
        sentiment.load('./phone_sentiment_model.marshal')
        s = SnowNLP('这款手机速度快')
    print("情感分析结果值:",s.sentiments)
```

首先,设置 is_train 变量为 True,将准备好的正、负情感文件命名为 neg.txt 和 pos.txt,放置在当前目录下,运行之后将在当前目录下生成一个 phone_sentiment_model.marshal.3 的文件,这一文件即为手机类目下的情感分类模型。将 is_train 置为 False 时,则可以加载模型,并使用待分析的句子构建 SnowNLP 对象,之后就可以输出其情感值(s.sentiments),重新运行之后,其输出结果如下:

```
情感分析结果值: 0.8190812500980639
```

5.4.4 语义角色标记

语义分析处理的是词之间语义关联关系,获取句子的深层意思,即语言单元之间的语义关系,这里涉及论元(argument)与谓词的概念,以"小猫钓鱼"这一句话为例,其中的"小猫"和"鱼"是谓词"钓"的两个论元,而"小猫"是施事,"鱼"是受事。动词都有自己的论元结构,及物动词有两个论元,而非及物动词只有一个论元,如"孩子吵闹"。像"张三把李四打了"和"李四被张三打了"两句话,在句法分析中,主语分别是张三和李四,但是这两句话描述的意思是相同的,而在语义分析中,张三在两句话中都是施事,而李四都是受事。所以,从语义角色标记的角度,可以提取出语句所描述的真实主体和含义。

在分析过程中,需要提取某一句子中描述的主体信息,用于实现语义实体相似度比较,减少使用全部词语带来的噪声干扰。语义角色标记采用哈尔滨工业大学的自然语言处理套件 LTP,通过 pip3 install pyltp 先安装套件,然后下载离线模型,其下载地址为 https://ltp.ai/download.html,将其存到当前目录的子目录 ltp_data_v3.4.0 中,包括 pos.model、cws.model、parser.model、pisrl.model、ner.model。其使用方法如下:

```
class RoleParser:
    def __init__(self):
        MODELDIR = "ltp_data_v3.4.0/"
        self.segmentor = Segmentor()
        self.segmentor.load_with_lexicon(os.path.join(MODELDIR, "cws.model"),str(os.path.
join(MODELDIR, "dict.txt")))
        self.postagger = Postagger()
        self.postagger.load(os.path.join(MODELDIR, "pos.model"))
        self.parser = Parser()
        self.parser.load(os.path.join(MODELDIR, "parser.model"))
        #语义角色标注
        self.labeller = SementicRoleLabeller()
        self.labeller.load(os.path.join(MODELDIR, "pisrl.model"))

    def get_role(self,input_sentence):
        input_sentence = input_sentence.replace(',',',')
        input_sentence = input_sentence.replace('!',',')
        input_sentence = input_sentence.replace('?',',')
        input_sentence = input_sentence.replace('!',',')
```

```
input_sentence = input_sentence.replace('?',',')
slist = SentenceSplitter.split(input_sentence)     # 分句

result = ""
for sentence in slist:
    if len(sentence)< 1: continue
    words = self.segmentor.segment(sentence)
    wordlist = list(words)
    postags = self.postagger.postag(words)
    arcs = self.parser.parse(words, postags)
    #语义角色标注
    roles = self.labeller.label(words, postags,arcs)
    #输出标注结果
    for role in roles:
        for arg in role.arguments:
            if arg.range.start != arg.range.end:
                ws = ''.join(wordlist[arg.range.start:arg.range.end])
                if arg.name == "A0":
                    if ws not in result:
                        result += ws
return result
```

其中，在 RoleParser 类初始化时，先依次加载自定义词典、分词模型、词性标记模型、句法依存模型和语义角色标记模型，上述模型加起来大约占用 1.2GB 硬盘存储，对内存也有一定要求。加载完成之后，将句子按照逗号、句号、分号、问号、感叹号等分割成短句，依次提取各个短句的角色内容，并只选择主体角色（A0）作为描述主体。直接通过如下方式调用其提取方法（get_role）。

```
if __name__ == '__main__':
    role = RoleParser()
```

sentence = "瓶身设计十分大方优雅，白色偏透明的面膜液水润易推。瓶装设计取用方便，不易造成浪费。主打的补水保湿效果优秀，也能较好地持久锁水。但清爽性方面有待改善。"

```
role_result = role.get_role(sentence)
print(role_result)
```

输出如下结果。

描述主体提取结果：瓶身白色偏透明的面膜主打的补水保湿清爽性

其输出结果在正常的句式结构下正确性较高，但是在网络用语方面主体提取存在较多问题，所以此模型主要用于识别句子是否为正常和合法的句子结构，通过判断其输出的 result 内容决定其句子的质量高低和合法性，也是语义角色标记的另外一种副产品应用。

5.4.5　语言模型

为了对推荐语生成模型生成的句子进行质量检测，使用语言模型对其通顺程度进行检测。语言模型的核心思想是通过概率分布的方式来计算句子完整性的模型，通过分析构成

句子的词之间的共现概率，实现句子合理性的判定，但是由于其计算公式中的参数过多，计算复杂度过高，需要近似的计算方法。最常用 n-gram 模型方法，此外还有决策树、最大熵、马尔科夫模型和条件随机场等方法。n-gram 模型也称为 $n-1$ 阶马尔科夫模型，它是一个有限历史假设，即当前词的出现概率仅仅与前面 $n-1$ 个词相关。n 越大，模型越准确，也越复杂，需要的计算量就越大。最常用的是 bigram，其次是 unigram 和 trigram，$n \geqslant 4$ 的情况较少。

语言模型常用的训练工具是 SRILM 和 RNNLM 等，其中 RNNLM 是基于循环神经网络的语言模型。本例中使用 SRILM 作为语言模型的生成工具，其核心代码如下：

```
ngram - count - vocab dict.txt.big.txt - text input.txt - order 3 - write out.count - unk
ngram - count - vocab dict.txt.big.txt - read out.count - order 3 - lm output.lm - interpolate
- kndiscount
ngram - ppl test.txt - order 3 - lm output.lm - debug 2 > test_result.ppl.txt
```

其中，ngram-count 是 n 元组的统计工具，通过-vocab 指定自定义词典，训练语料存于 input.txt 中，每一行是一句话经过分词和去停用词之后的结果，-lm 后的 output.lm 是生成的语言模型结果。通过-ppl 指定输出困惑度指标值，将结果保存在 test_result.ppl.txt 中。

以下面这句话为例。

虽然 说 现在 已经 进入 冬天 了 但是 补水 保湿 的 工作 的 不能 停下 的 上班族 的 小美女 们 每天 面对 着 电脑 和 空调 的 侵害 肌肤 整天 都 是 紧 紧绷绷 的 仔细 看看 全都 是 干皮 护肤品 用 了 一堆 也 都 不太有 效果 要 知道 你 肌肤 缺失 的 水分 是 需要 用 面膜 才能 补 回来 的 那 你 就 一定 需要 这些 面膜 咯

其困惑度的输出结果为 162，完整的调试信息如下：

```
p( 虽然 | <s> ) = [2gram] 0.002524794 [ - 2.597774 ]
    p( 说 | 虽然 ...)   =   [3gram] 0.0647906 [ - 1.188488 ]
    p( 现在 | 说 ...)   =   [2gram] 0.001056698 [ - 2.976049 ]
    p( 已经 | 现在 ...) =   [2gram] 0.01859457 [ - 1.730614 ]
    p( 进入 | 已经 ...) =   [2gram] 0.02077514 [ - 1.682456 ]
    ...
    p( 咯 | 面膜 ...)   =   [2gram] 0.000294772 [ - 3.530514 ]
    p( </s> | 咯 ...)   =   [2gram] 0.03777696 [ - 1.422773 ]
1 sentences, 71 words, 4 OOVs
0 zeroprobs, logprob = - 150.3414 ppl = 162.5185 ppl1 = 175.3482
```

5.4.6 词向量模型 Word2vec

神经网络在处理自然语言时，需要将其数值化，可用的编码方式有独热(One-hot)编码、Word2vec 和 GloVe 等。其中，独热编码方式也称为一位有效编码，这是因为经过它编码之后，所有位数中只有一位是 1，其他全部为 0，例如“爱”字在词典中的顺序为 32，词典的总字数为 3500，则在总位数 3500 位的数值序列中，只有第 32 位是 1，即[0,0,0,…,0,0,1,0,0,..0,0,0]。如果是一句话，则通过独热编码之后就得到了一个稀疏矩阵，以下是以独热编码的调用示例。

```
from sklearn import preprocessing
```

```
enc = preprocessing.OneHotEncoder()
enc.fit([['我'],['爱'],['北'],['京'],['天'],['安'],['门']])
enc.transform([['爱']]).toarray()
```

其中，OneHotEncoder 是在 scikit-learn 库的预处理模块（preprocessing）中，enc.fit 实现了对“我爱北京天安门”的编码，其中的每个字都是以独热的形式表示了，可将“爱”字用 transform 输出，其结果为 array([[0.,0.,0.,0.,0.,1.,0.]])。这一编码方式的不足之处是存在词义鸿沟问题，即词与词之间在语义关系这种词向量中并没有体现出来，一般应用此方法来编码标签列等离散型数据。

Word2vec 是利用一个 3 层的简单神经网络进行无监督式学习词向量，这一词向量是其语义的表示，其输入是语料，输出是词向量，在 Word2vec 的训练过程中，利用 skip-gram 或连续词袋（CBOW）来建立输入与输出变量。其中，skip-gram 是在给定的一句话中，它指定一个中心词，让神经网络来预测此词前后 n 个词（字），其中 n 为窗口的大小，以“我爱北京天安门”为例，从“我”字开始，依次遍历到“门”字，当遍历至“京”字为中心词时，窗口大小为 2，则神经网络的输入是“京”的独热编码，而输出则分别是“爱”“北”“天”“安”的独热编码，即构造了（“京”“爱”）、（“京”“北”）、（“京”“天”）、（“京”“安”）4 个训练样本。训练过程中，如果神经网络的输出与目标输出不一致，则不断调整网络参数，那么在训练完成之后，意味着隐层学习的特征为词与词之间的关系信息，此时将最后的输出层去掉，其中隐层的输出就是词的向量值，即实现了从词到向量的编码，而这些向量值内含了词间关系，不仅解决了词义鸿沟问题，实现了词语的语义表示。前面的示例中是以字为单位，在实际工作中一般采用词作为单位，即神经网络的输入是经过分词之后的句子，窗口移动是针对词语进行的。下面是利用 gensim 实现的词向量训练代码，可通过 pip3 install gensim 安装。

```
import os, sys
import multiprocessing
import gensim

def word2vec_train(input_file, output_file):
    sentences = gensim.models.word2vec.LineSentence(input_file)
    num_lines = sum(1 for line in open(input_file))
    if os.path.exists(output_file):
        print("model exist and loading...")
        model = gensim.models.Word2Vec.load(output_file)
        model.min_count = 6
        model.build_vocab(sentences , update = True)
        model.train(sentences,total_examples = num_lines,epochs = 5)
    else:
        print("training new model...")
        model = gensim.models.Word2Vec(sentences, size = 600, min_count = 6, workers =
multiprocessing.cpu_count())
    model.save(output_file)
    model.wv.save_word2vec_format(output_file + '.vector', binary = True)

if __name__ == '__main__':
    if len(sys.argv) < 3:
        print("Usage: python word2vec_train.py infile outfile")
```

```
        sys.exit()
    input_file, output_file = sys.argv[1], sys.argv[2]
    word2vec_train(input_file, output_file)
```

其中，首先引入 gensim 库，并定义 word2vec_train 方法，其输入语料是经过分词和去停用词的句子，每一行作为一句，输入是模型文件的路径，首次运行时模型并不存在，所以直接调用 gensim. models. Word2Vec 方法进行训练即可，其中 size 表示的是神经网络中间隐单元的数量，其值越大表示学习的特征越丰富，也意味着训练时间更长，一般情况下，语料较少时，其值也就相应调小。min_count 表示词最少出现次数，workers 是指定训练过程的进程数量，这里指当前机器的 CPU 核数，训练完成之后使用 save 方法保存，并使用模型的 save_word2vec_format 方法保存词向量。

如果 Word2vec 词向量已经存在，需要进行增量学习时，使用 gensim. models. Word2Vec. load 方法加载原模型，并指定最小词频数，基于新的语料使用 build_vocab 方法更新字典，然后调用 train 方法实现训练，最后，与初次训练一样，在训练完成之后将结果保存。

Word2vec 训练完成之后，这些固定下来的词向量既可用于其他模型的输入向量化工具，也可用于句子(词)之间相似度的计算。基于不同的训练语料，相同的词其输出的词向量值会存在差异，所以不同企业依据其内部数据所生成的词向量也可以认为是其数字资产。下面是 Word2Vec 的使用示例，代码如下：

```
import os, sys
import multiprocessing
import gensim
from gensim.models import Word2Vec
import jieba

model = None

def load_model(model_path):
    global model
    model = Word2Vec.load(model_path)

def word2vec_eval(model, word):
    if word in model.wv.vocab:
        return True
    else:
        return False

def get_vector(word):
return model.wv[str(word)]

def get_most_similar(words_list, neg_list=[]):
    return model.wv.most_similar_cosmul(positive=words_list, negative=neg_list)


if __name__ == '__main__':
    load_model("./new_model/zhwiki")
```

```
for word in jieba.cut("我爱北京天安门"):
    is_exist = word2vec_eval(model, word)
    print(word + " " + str(is_exist))

word = "北京"
print(word," vector:",get_vector(word))

words_list = ["面料","气质","性感","亲肤","裙子","印花","时尚","面料","版型","蕾丝"]
sim_words = get_most_similar(words_list)

f = next(iter(sim_words or []), None)
print("特点:", " ".join(words_list), " 最相近的词为:", f[0])
```

首先，通过 load_model 加载 Word2vec 模型，由于词向量的训练语料可能不会覆盖全部词汇，所以在使用模型生成词向量时，需要先使用 word2vec_eval 方法检测该词是否存在于 Word2vec 的词典中，如果存在则可以使用 get_vector 方法获得其向量值，这些值可以作为其他神经网络的输入值，上述代码运行之后的输出结果如下：

```
我 True
爱 True
北京 True
天安门 True
北京  vector: [-1.29923499e+00 -5.83571732e-01  1.08659372e-01 -1.96763605e-01
  2.09212732e+00 -2.57835007e+00 -6.30052507e-01  1.84415162e-01
  …
  2.83506632e-01 -2.91695380e+00 -9.04185653e-01 -1.14341331e+00
 -2.90749818e-01  4.46737498e-01 -8.56931508e-01  1.81815311e-01]
```

特点："面料 气质 性感 亲肤 裙子 印花 时尚 面料 版型 蕾丝"最相近的词为："连衣裙"。

从中可以看到，"我爱北京天安门"已经在训练语料中大量出现，也就在其词典中存在，通过 get_vector 方法得到的向量为 600 维。而输入多个词之后，可以从中检索与之都相近的词是哪些词，从中选择最近的一个(连衣裙)，可以看到其相似度计算的结果中已经内含了基本的语义信息。

本章主要说明了与自然语言处理相关的核心技术应用方法，在自然语言处理过程中，通过预处理分词、去停用词、词性标记、句法分析、语义分析等，然后进行文本处理应用，例如：文本分类、主题提取、文章摘要、文本检索等。通过掌握基础的自然语言处理技能，有助于在高层应用中进行创新，同时，基础工作的质量高低也决定了高层应用算法的性能。