|  |  |
| --- | --- |
|  | 学校代码： 10246 |
|  | 学 号： 16212010030 |
|  |  |



|  |
| --- |
| 硕 士 学 位 论 文 |

（学术学位）

**基于API文档的约束性语句分类及应用**

**Directive Classification and Application based on API Documents**

|  |  |
| --- | --- |
| 院 系： | 软件学院 |
| 专 业： | 计算机软件与理论 |
| 姓 名： | 张凯 |
| 指 导 教 师： | 赵文耘 教授 |
| 完 成 日 期： | 2019年 2月 25日 |

**复旦大学**

**学位论文独创性声明**

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。论文中除特别标注的内容外，不包含任何其他个人或机构已经发表或撰写过的研究成果。对本研究做出重要贡献的个人和集体，均已在论文中作了明确的声明并表示了谢意。本声明的法律结果由本人承担。

作者签名： 日期：

**复旦大学**

**学位论文使用授权声明**

本人完全了解复旦大学有关收藏和利用博士、硕士学位论文的规定，即：学校有权收藏、使用并向国家有关部门或机构送交论文的印刷本和电子版本；允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。涉密学位论文在解密后遵守此规定。

作者签名： 　 导师签名： 日期：

指导小组名单

赵文耘，彭鑫

目录

摘要

API在程序员进行编程时得到了广泛的使用，而程序员对API的查询和理解依赖于API文档。API文档不仅对API进行了功能及使用流程的说明，还提供了一些使用时的规定和限制，我们把这些语句成为约束性语句。约束性语句的是否正确解读决定了程序员能否准确无误的使用这个API，保证程序的正确性。

借助fastText工具，通过对已有API文档的@todo XXX句文本进行解析、训练、识别，我们将约束性语句分为@todo XXXXX几大类，平均准确率为@todo、查全率为@todo。接下来，通过对几大类中的句子进行分析总结，我们针对@todo XX等不同小类定义出了不同的启发式规则。最后，通过通用的匹配模式，我们获取约束性语句中的信息，对代码进行扫描检测，找到其中与约束性语句不符的片段，提示程序员注意与修复。

Abstract

第一章 引言

应用程序编程接口（Application Programming Interface，简称API）是一组定义好的函数，供编程人员调用后实现一系列特定的功能[1]，而实现的细节对编程人员来说是透明的。API通常实现的是一些经常得到复用、通用性比较强的功能，单独把这些功能抽取出来进行包装，能够使得不同程序员在实现该相同功能时能够节省编码的时间，提高构建软件时的效率。API在程序员进行编程时得到了广泛的使用，适用于各种编程场景，实现不同的程序功能[2-4]。

API文档就是对这些API的说明解释，是指导程序员如何使用API的指南。特别是对于第三方组织实现的API接口，由于发布时间较新、接口过于复杂、相关资料过少等原因，程序员必须严重依赖于其API文档，才能获得足够的知识去正确的使用它[5, 6]。

API文档是关于某个API的整体知识库，包含了如何正确使用API的各种信息，包括了API实现的功能、API相关概念解释、示例代码、API使用约束等。API约束性语句是我们关注的重点，它是描述了在使用API编程时应该遵守的限制和规范的语句。它必须被严格遵守，否则编写出来的程序就有可能不能运行或出现错误，降低程序的正确性，甚至可能造成损失。为了对API约束性语句有更直观的印象，我们在API文档中找了几个例子来说明，图1中展示了我们作为示例的3个片段。

第一段的描述片段来源于JDK文档中javax.xml.crypto.dom. DOMCryptoContext类中的getNamespacePrefix方法，该方法将传入的uri进行解析，返回此uri的前缀。其中的约束性语句“Throws:NullPointerException - if namespaceURI is null”表示参数namespaceURI如果为空，则抛出异常，这是对入参非空值的规定，应该被重视和遵守。

第二段的描述片段取自Apache Commons Collections library文档中org.apache.commons.collections4.set.ListOrderedSet的listOrderedSet方法，它的功能是返回一个排好序的集合对象，其中约束性语句“The list and set must both be empty.”规定了方法的入参必须都是空集，不能含有任何元素。

第三段展示了JDK文档中java.sql.Time类中的valueOf方法的描述片段，valueOf方法将一个JDBC格式的字符串转为Time对象，而其中的约束性语句“Parameters:s - time in format "hh:mm:ss"”的意思是s这个参数必须是

图 1 三段示例描述片段，黑色方框中的语句是其中作为代表的约束性语句

**valueOf**

**public static Time valueOf(String s)**

Converts a string in JDBC time escape format to a Time value.

**Parameters:** s - time in format "hh:mm:ss"

**Returns:** a corresponding Time object

**listOrderedSet**

**public static <E> ListOrderedSet<E> listOrderedSet(Set<E> set,**

**List<E> list)**

Factory method to create an ordered set specifying the list and set to use.

The list and set must both be empty.

**Type Parameters:** E - the element type

**Parameters:** set - the set to decorate, must be empty and not null

list - the list to decorate, must be empty and not null

**Returns:** a new ordered set

**Throws:** NullPointerException - if set or list is null

IllegalArgumentException - if either the set or list is not empty

**Since:** 4.0

**getNamespacePrefix**

**public String getNamespacePrefix(String namespaceURI, String defaultPrefix)**

This implementation uses an internal HashMap to get the prefix that the specified URI maps to. It returns the defaultPrefix if it maps to null.

**Specified by:** getNamespacePrefix in interface XMLCryptoContext

**Parameters:** namespaceURI - a namespace URI

defaultPrefix - the prefix to be returned in the event that the the specified namespace URI has not been bound to a prefix.

**Returns:** the prefix that is associated with the specified namespace URI, or defaultPrefix if the URI is not registered. If the namespace URI is registered but has no prefix, an empty string ("") is returned.

**Throws:** NullPointerException - if namespaceURI is null

**See Also:** XMLCryptoContext.putNamespacePrefix(String, String)

“小时：分钟：秒钟”这样格式的字符串，才能转换出正确的Time对象，否则输出的Time对象可能出现不可预测的错误。

从上面三个示例片段可以看出，首先，API约束性语句是融合在其他类型的描述语句之中的几种类型的语句可能相互连接，互为支撑，形成了一篇完整的API描述文档。其次，API约束性语句在整个文档中占比是比较小的，甚至有些描述文档是不含有API约束性语句的，因此分离工作就如同沙滩上寻找金子一般，虽然明显但也需要付出很大的努力。同时API约束性语句其实还有这各种不同的类别，上面三个片段中每个片段的约束性语句都属于这不同的类别，分别是“不允许空值”“允许空值”和“字符串类型限定”。有些类别自然语言特征结构是比较明显的，但是有些类别的自然语言形态又是多种多样的，甚至有些类别的语言形态是十分相似的，这对自动化分类提出了挑战。总而言之，对API约束性语句的研究存在着以下几个问题：第一个问题是怎么能够将API约束性语句从文档中这么多种类的知识中分离出来；第二个问题是这些多种多样的API约束性语句又是怎么具体分类的。

在现有的研究中，对API约束性语句的识别主要是依靠人工的标注和模式的识别，包括但不限于一些显著的标识，如@param、@exception、@throws、must等等[7]。而对于API约束性语句的具体分类，则只有人工进行阅读、标注来保证准确性。但是对于API文档中成千上万的约束性语句，仅仅依靠人工去标注显然是效率十分低下的一种方法，同时也对大规模识别提出了挑战。在这个背景下，我们对这个问题进行了良好的解答，为大规模自动化识别约束性语句提供了新方法和新思路。

首先，我们利用已有的人工对API约束性语句分类的数据库作为我们的训练集和验证集，利用fastText来进行较为粗糙的大类API约束性语句分类。因为我们的主要研究是针对API约束性语句，因此API文档中除了约束性语句的其他类别全部归为非约束性语句。其他的大类类别还包括@todo 等。然后，针对fastText分好的每一个大类数据，我们人工的阅读其中的每一个句子，观察句子的形态，结合自然语言处理技术中对句子结构的分析，定义解析跟API相关名词的识别方法，得到每个细分小类的一组启发式规则。根据这些启发式规则，我们就能匹配出高准确率的属于某个细分小类的约束性语句。细分小类属于某个大类，如@todo属于@todo，而针对这个细分小类的启发式规则有@todo等等。

整个工作的贡献如下@todo填充一些具体数据进行支撑：

1. 通过fastText的帮助，我们结合机器学习的方法训练出了约束性语句大类分类模型，将原本种类多样繁杂的约束性语句先进行了预分类，将大类间相似启发式规则的相互影响降低，提高了后续分类的准确性。
2. 针对所有细分小类都定义了与其匹配的一组启发式规则，保证了分类的准确性，也为大规模分类提供了方法和思路。
3. 给出了一个应用，将分好类后的约束性语句用于代码正确性的检查，提供了一种保证程序员编程正确性检查的新方法，丰富了代码检查的方式，给代码正确性提供了更多保证。

@todo 文章章节结构简述

第二章 背景知识

2.1 API相关知识

2.1.1 API概念及种类

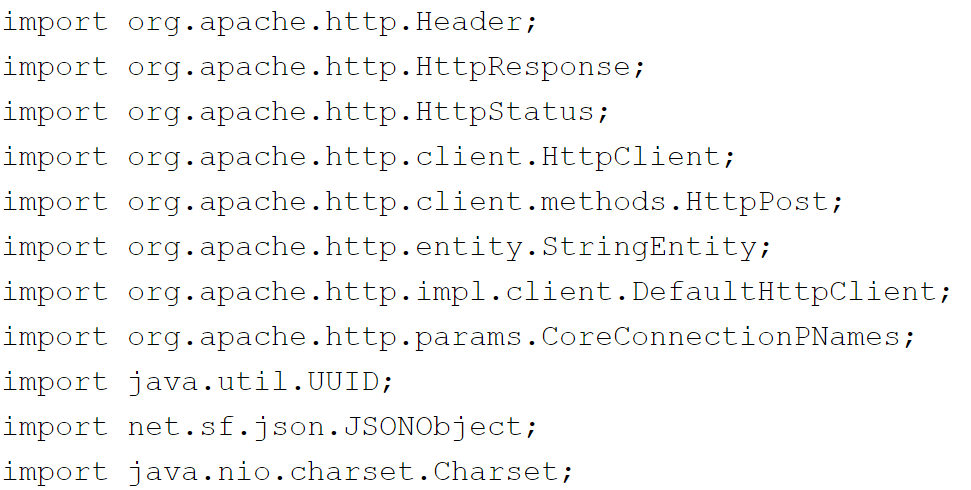
API，又名应用程序接口，是一组程序功能的定义、通信协议或用于软件编程的工具，定义了各个组件之间是如何交互的。应用程序接口通过抽象底层实现，只将需要用到的对象或操作暴露给程序员，底层的源代码或者内部工作机制的细

图 2 JAVA编程中利用HttpClient实现网络信息发送

节对程序员来说是透明的，这使得程序员在编写软件时调用这些功能时更加简单方便。

举例说明，在JAVA编程时，发送HTTP请求是我们在网络中进行交互必不可少的功能。一般来说，程序员通过调用HttpClient这个API进行网络请求的发送。图2展示了API调用的示例代码，首先我们引进（import）相关的API包，这使得程序能够获知需要调用API的接口和对象，保证程序编译的正确性。然后，我们调用API中的方法，填充相关对象作为参数，在这个例子中就是填充HTTP相关的连接信息、组建HTTP包。最后，通过“HttpResponse response = httpClient.execute(post);”这句代码进行网络信息的发送，并检验发送结果。

通过API进行HTTP调用，如果不计算填充调用参数的代码行数，其实只需要短短的三行代码就可以实现。然而，真实的HTTP调用是十分复杂的，API帮调用者屏蔽了底层复杂的实现，无疑大大提高了编程效率。例如，在进行HTTP调用时会传入目标域名，但是域名是便于人方便记忆的一个地址，在整个网络传输过程中是不能通过域名进行定位的。因此，需要对传入的域名（URL）进行域名解析，这涉及到通过缓存、域名服务器DNS解析进行域名到IP地址转换的整个流程。HTTP请求是一个应用层协议，需要依赖于运输层的协议进行数据的传输，因此Http是基于TCP/IP协议的。TCP的三次握手连接、数据传输、四次挥手断开连接、IP层的传输等等底层的协议调用都是API为调用者完成的，不需要调用者去关心具体实现，否则，HTTP调用就不是简单的几行代码能够完成的任务了。

API也有不同的类别，如库和框架的API，操作系统API，远程API，网络API等等[[1]](#endnote-1)：

1. 库API：API通常与软件库有关，API是一个功能的描述，而库则是对这个功能的具体实现。因此，单个API可以拥有相同程序接口不同的库形式的多种实现，这有一个好处就是一种语言可能可以调用另一种语言实现的库，例如SCALA开发时就可以调用JAVA的API。API的使用可能因为语言的不同而不同，比如面向对象语言的API提供了类与方法的规范，我们的研究主要针对的就是面向对象语言的API。
2. 操作系统API：API定义了操作系统和应用程序之间的接口，如windows操作系统提供了Windows API库供开发者使用。
3. 远程API：这种类型的API的作用是让程序员通过某种协议来操作远程资源，而特定的协议标准使得不同语言、不同平台的程序能够协同工作。JAVA远程方法调用就是一个很典型的例子，它通过JAVA远程方法协议来远程操作函数，但使用感受就像调用本地方法一样。因为JAVA远程方法API会在本地执行一个远程方法的代理对象，而通过代理对象经过协议去调用远程的方法，并返回远程方法执行的结果作为本地代理对象方法的返回值。因此，本地代理对象和远程对象之间存在一定映射关系，如果要修改其中一个，两者都要同时被修改。
4. 网络API：在进行网络开发时，API通常被定义为一组规范，比如HTTP的消息结构、HTTP传输协议等等。网络API在历史上几乎已经成为网络服务的同义词。但是随着网络2.0的发展，网络API也逐渐发展成为不同领域、不同类型的消费者的不同应用程序提供某种服务的程序接口。

2.1.2 API文档

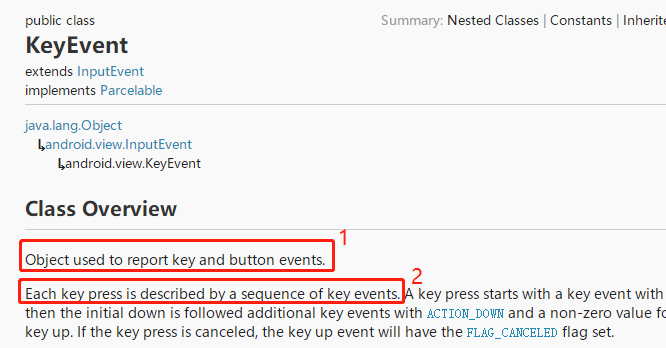
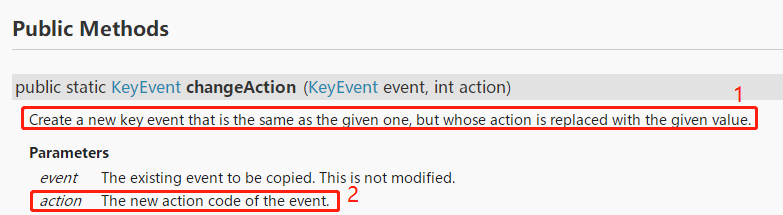
一个程序员在尝试使用API解决问题的时候，会想要知道如何准确有效的使用，这时API文档就起到了很大的作用。API文档是一种技术内容可交付成果，包含了如何有效使用和集成API的说明，也是一个简明的参考手册[[2]](#endnote-2)。

文档一般能在API的配套文档文件中能够找到，比如Javadoc、Pydoc等，他们具有相同的呈现方式和文档结构，但是其中的内容会因API的而不同[8]。API文档主要包括了类和方法的描述、设计原理、典型的使用场景、性能表现、示例代码片段等等，而API本身的内部实现细节一般是隐藏起来的。这类信息主要蕴含在功能性语句、概念性语句和值定义语句三种句子中。功能性语句主要表达了API是在什么场景下为了实现什么功能而编写的，在何种情况下性能如何，使用的流程是什么样的。如图3中类描述和方法描述的第一个方框中的那个语句，前一句表达了KeyEvent这个类的功能就是表示按键的事件，后一句说明了changeAction这个方法是把event 的action改成参数值传入的值。概念性语句描述了跟这个API相关的一些改变，具有一定的解释性、通用性。比如图3中类描述第二个方框中的那个语句，解释了每个按键的按压动作由一系列事件组成这个原理。但是如果开发者对这个原理很了解的话，这句话除去不看也不影响整篇文档的理解，这句话使得文档更加通俗易懂，这就是概念性语句的作用。值定义语句描述了一个方法参数或者返回值的属性，比如方法描述第二个方框中入参action其实就是需要新传入的事件的动作码。

除此之外，文档还包含了API的限制和约束，具体例子如图1所示。比如，此函数可能已经被废弃了，推荐使用别的新版本的方法；此函数的入参不能为空等等。这些语句称为约束性语句，是我们研究的重点。因为API文档对于使用者来说需要仔细阅读和理解，所以也对作者提出了更高的要求，需要随着API的更新而更新，还要保证文档的正确性。然而，即便是大型机构的API官方文档，API文档还是会存在一些错误，这是无可避免的，需要使用者在阅读和使用的时候格外注意[7]。

图 3 安卓API官网文档中示例片段，取自KeyEvent类

2.2 机器学习相关方法及工具

2.2.1 有监督学习

在机器学习领域，主要有两种类型的任务：监督和无监督，两者之间的区别在于是否需要根据基础事实来完成，也就是说我们是否事先知道样本的输出应该是什么样的。无监督学习是没有示例样本，根据给定数据推断这组数据中存在的自然结构；而有监督学习则是根据给定的样本数据和期望的输出来推测最近似于样本数据的输入和输出之间的一种关系。两者之间各有优劣，选择时需要根据具体的场景需求来判断哪种学习方式更合适。

有监督学习一般遵循以下几个基本步骤：首先，确定训练数据的类型，收集训练集。比如，在进行句子分类的时候，需要确定要分类的句子是哪些，然后将这些句子的示例从网站上、书籍中等地方收集起来。其次，确定学习函数的输入、结构和相应的学习算法。函数的输入一般来说取决于输入对象的表达方式，通常会被转换成特征向量。特征向量包含了许多能够描述对象的特征，这些特征不应该太多，但又需要蕴含足以准确预测输出的信息，所以要控制好特征向量的维度。现有的学习函数也有许多，如支持向量机、线性回归、决策树等等，需要根据使用场景和数据特征来选择。最后一步就是在收集的训练集上运行学习算法，有时可以通过验证集去优化学习算法的参数，使模型更加准确。

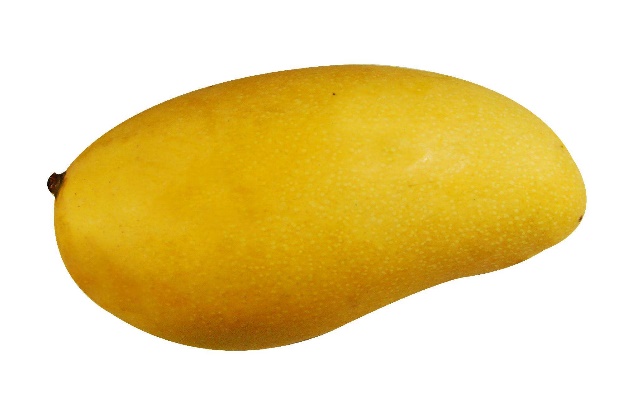
可以用一个例子来说明一下有监督学习的过程，假设现在有一个装满了不同水果盘，如图4上半张图所示，那么这个水果盘里面的水果就是我们的训练集，我们会做如下训练：如果水果的形状是椭圆形的且表面光滑颜色为黄色，那么它会被标记为芒果；如果水果形状是不规则的、表面有一个个小点、表皮为红色且顶端带有绿色叶片的，那么它会被标记为草莓。假设在我们训练了这个数据之后，给这个模型一个单独的水果，如图4下半张所示，要求识别这个水果的类型。由于模型已经从原来的果盘中学到了水果的各种特征，它将识别新给的这个水果是黄色的、表皮光滑且为椭圆形，因此归类为芒果。这就是有监督学习将训练数据（给定的果盘中的水果）学习到的特征结构等知识用于辨别测试数据（用于被识别的芒果）的过程。分类问题是有监督学习典型的应用场景，它的训练数据是一些已经分好类的数据，输出变量是类别，例如“红色”或“蓝色”或“绿色”。

图 4 上半张图是作为训练集装有各种水果的水果盘，下半张图是需要通过模型判别类型的给定水果

2.2.2 n元语法

n元语法（N-Gram）是一种概率语言模型算法，主要用于根据前n-1个词推断文本中下一个词是什么。Gram就是一个文本片段，按照事先设定的阈值进行所有出现的Gram的频度的统计，我们可以得到这个完整文本的特征向量。

n元语法基于一种假设，那就是一段文本中每个词的出现只与它前面的n-1个词有关，因此每个词出现的概率就是该词与前面n-1个词同时出现的概率，而整段文本出现的概率就是每个词出现概率的乘积。

图 5 Google搜索中的N-Gram应用

n元语法中的n对整个模型有着重要的影响。当n越大时，对每个词的约束性就更大每个Gram里面含有的信息就越多，Gram的维度就越大，能够更好的分辨每一个词，但是会使得形成的每个词的特征向量更加稀疏。当n越小时，每个词的特征向量更丰富，但是约束性条件更少。举例说明，假设目前有这么一句话：“你好吗”，那么词表中所有可能的2-Gram的总数就是32=9个，分别是“你你”，“你好”，“你吗”，“好好”，“好你”，“好吗”，“吗你”，“吗好”，“吗吗”。而如果是3-Gram的话，词表中的词向量数量就有可能为33=27.

n元语法有一个非常常见的应用就是搜索引擎或者输入法中的备选词联想，如图5所示，我们在进行Google搜索时，只输入前两个单词“白天”，他就会自动联想出我们可能输入的下一些词是什么，且越靠前的说明其查询频率越高，这是根据词库中储存的N-Gram词向量实现的。假设我们使用的是2-Gram语言模型来预测下一个词，那么在这个例子中，Google的词库储存的词向量存在以下信息：p(“不懂夜的黑”|“白天”)>p(“黑夜交错”|“白天”)>……>p(“天羽”|“白天”)。这些概率值就是根据用户输入的查询数据用上面介绍的N-Gram方法求出来的

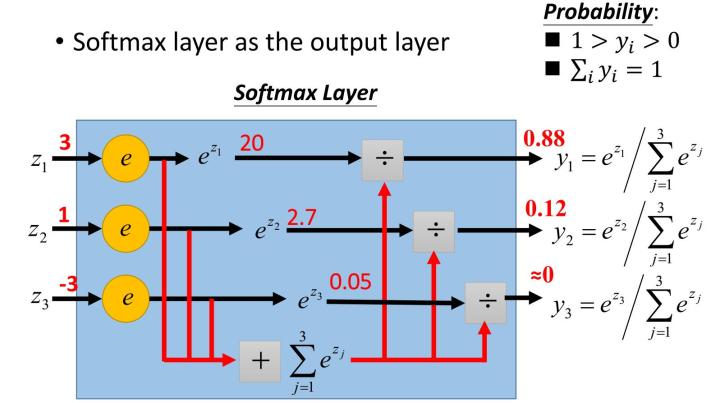
2.2.3 SoftMax

图 6 SoftMax得到最后的分类概率

SoftMax函数[[3]](#endnote-3)，又称归一化指数函数，它将一个非标准化的实数向量归一化到一个概率分布里面。在应用SoftMax之前，实数向量的元素可能出现负值或大于1，应用之后，使得每个元素的范围都在[0,1]之中，并且所以元素的和为1。假如我们有一个数组，数组中的元素个数为n，那么，第i个元素的SoftMax值通过公式(1)计算：

(1)

通过一个例子来说明SoftMax的工作原理，如图6所示[[4]](#endnote-4)，当我们获得了前面三个动作{a,b,c}经过一系列处理后的输出{3,1,-3}之后，通过上面的公式算出每个值对应的SoftMax值是{0.88,0.12,0},因此我们会选取概率最大的也就是0.88对应的那个动作a来作为最后的结果。那么如果对应的多分类问题，想选取两个动作，就找到概率最大的两个{0.88,0.12}对应的动作{a,b}。

2.3 自然语言处理

自然语言处理[[5]](#endnote-5)是计算机科学、信息工程和人工智能交互的一个子领域，主要涉及计算机和人类（自然）语言之间的交互，特别是如何编写处理和分析大规模自然语言数据的计算机程序。自然语言处理又可以分成很多个子任务，如词性标注、句法分析、文本分类、命名实体识别、信息检索等等。我们主要涉及的是词性标注、句法分析和命名实体识别。

词性标注是为了给句子中的每个单词确定正确的词性，比如动词、名词、形容词、代词等等。而句法分析则假设每句话里面只有一个单词是核心词而存在，其他词都直接或间接依赖于核心词，即修饰词。每个词能且只能依赖于另一个词而存在，不能同时依赖于多个次。句法分析就是将句子中的依赖关系（如表1所示）标记出来。命名实体识别则是将句子中具有特定意义或指代性的实体，比如公司名、地名、人名、时间、专有名词等等，按照业务要求从非结构化的句子中识别出来。

表格 1 句法分析中的依赖关系，只列出部分作为参考

|  |  |  |  |
| --- | --- | --- | --- |
| 关系 | 标签 | 描述 | 示例 |
| 主谓关系 | SBV | 主语-谓语 | 你给我一张纸（你🡨给） |
| 动宾关系 | VOB | 直接宾语， verb-object | 你给我一张纸（给🡪纸） |
| 间宾关系 | IOB | 间接宾语，indirect-object | 你给我一张纸（给🡪纸） |
| 定中关系 | ATT | attribute | 蓝精灵（蓝🡨精灵） |
| 介宾关系 | POB | preposition-object | 在大海中（在🡪中） |

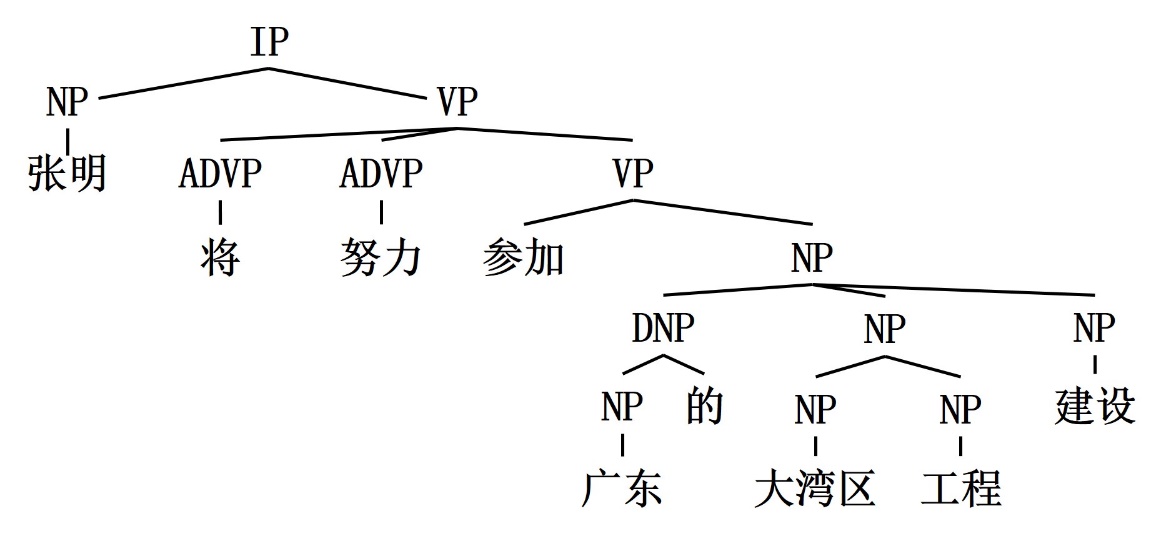


图 7 词性标注示例

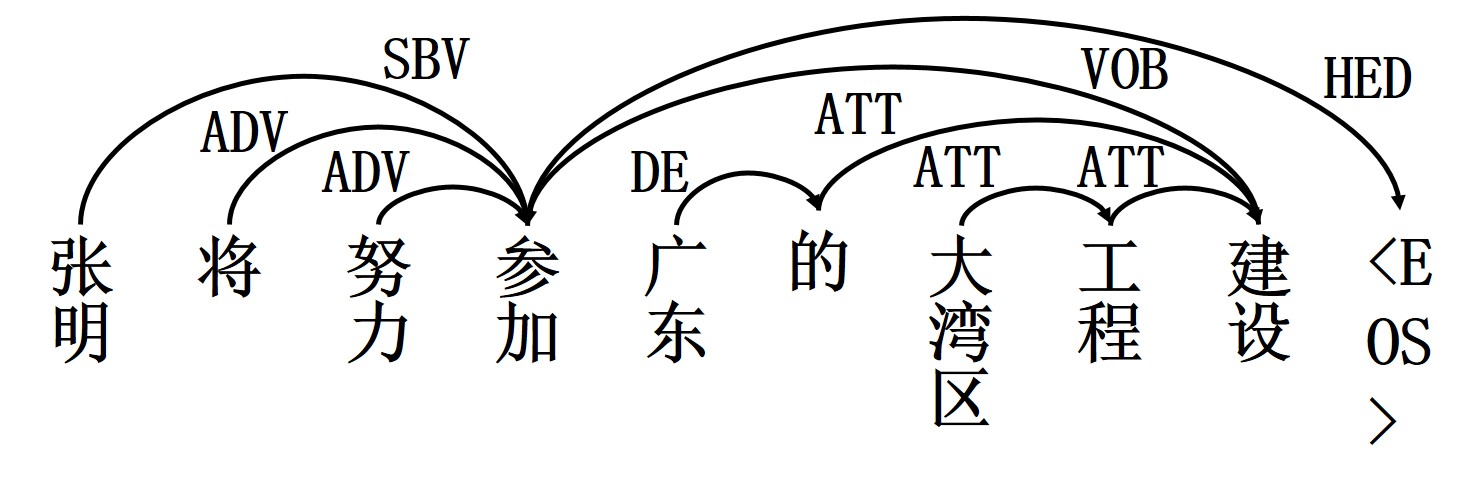
假设现在我们要分析这样一句话：“张明将努力参加广东的大湾区工程建设”，那么我们可以先用命名实体识别找出其中的潜在实体：“张明”、“广东”、“大湾区”，这些实体可以为我们提供重要的信息。其次，我们可以进行词性标注，得到每个单词的词性，如图7所示。词性标注可以展示为树的形式，由终结点，非终结点和短语标记组成，非终结点需要进行进一部分解，一系列终结点构成了一个短语。句法分析结果则如图8所示，依赖关系用有向弧来表示，从修饰词指向核心词。

图 8 句法分析示例

2.4 静态程序分析

在计算机科学中，程序分析[[6]](#endnote-6)是为了自动化的分析计算机程序正确性、稳健性、安全性等指标的过程，主要分为两种方式：静态程序分析和动态程序分析。静态程序分析指的是在程序不运行的状态下分析程序的源码或机器码来完成对代码的检查，而动态分析则恰恰相反，需要在程序执行的时候进行代码的分析，可能会对程序的运行效率造成影响。两种分析方法并不是完全互不相容的，他们都有各自的优缺点，在一定程度上能起到相辅相成的作用。静态程序分析会考虑到程序所有要执行的路径，而动态分析只检查得到其中某一条被执行的路径；但动态分析能够得到运行的真实值，相对来说更加精确。

我们主要涉及到的是静态程序分析技术，它包含了很多可以单独使用的子方法，如数据流分析、控制流分析、语法树分析等等。语法树分析[[7]](#endnote-7)是源代码语法结构的一种抽象表示，它的每个节点表示的是源代码的其中一个语义结构，但它是抽象的，不是所有源代码的细节都会在树状结构中作为一个节点被体现出来，比如if-else 判断条件，在抽象语法树中是作为一个分叉去表示的。数据流分析则是通过收集程序运行到不同地方时各个值的变化信息，构成一个数据的流向图。

2.5 本章小结

我们介绍了一系列在研究中运用到的软件技术，这些技术来自各个领域，包括代码分析、自然语言处理、机器学习、深度学习等，我们对这些技术进行了应用，巧妙地穿插在研究的各个阶段。各个领域的知识并无边界，都可以相辅相成，完成跨领域的联合研究，这不仅体现了我们对新技术新方法的学习，还能够使我们的研究跟上最新的研究水平，不会老旧落后，也能得到更好更精确的结果。

第三章 整体框架

3.1 框架概览

整体框架如图所示，分为两个阶段：图9的大类分类阶段和图10的细分小类分类阶段，两个阶段相互连接、共同工作。

在大类分类训练阶段，我们首先得到训练数据，训练数据已经是每句话对应其大类类别的形式，然后需要将训练数据进行平衡，这里主要平衡的是非约束性语句的数据，使其能够和其他类别差别不是这么大。接着就进入了FastText[9]的整个训练流程，过程是先进行N-Gram的提取，将语料的词序列和提取出来的N-Gram词向量一起作为隐层的输入，输出经过SoftMax得到最后训练模型结果。

在大类分类预测阶段，我们将还未分类的原生句子作为训练好的FastText模型输入，模型会自动为我们预测出句子所属的类别。而我们可以通过FastText得到SoftMax分成各类别的概率，当预测类别的概率高于某个阈值时，我们将此结果重新作为训练数据进行模型的训练。如此迭代的将新数据作为训练数据更新模型，可以使模型接收到更多种多样的语句样式而提高其准确性和完备性。

在细分小类分类阶段，我们先将前面大类分类过的结果进行自然语言的处理，得到其中蕴含的命名实体、词性、词与词之间的依赖关系，这些信息是为之后的模式识别作辅助。然后，我们人工的观察所有带有细分小类标注的约束性语句，总结所有细分小类中可能存在的匹配模式。通过这些匹配模式，我们就能自动化的得到属于这个细分小类的约束性语句。

3.2 约束性语句分类

在参考了Martin Monperrus等[10]人对于约束性语句的分类之后，我们结合自己对数据的观察和工具的特性，得到了一个有少许不同的约束性语句分类模型，如图11所示。

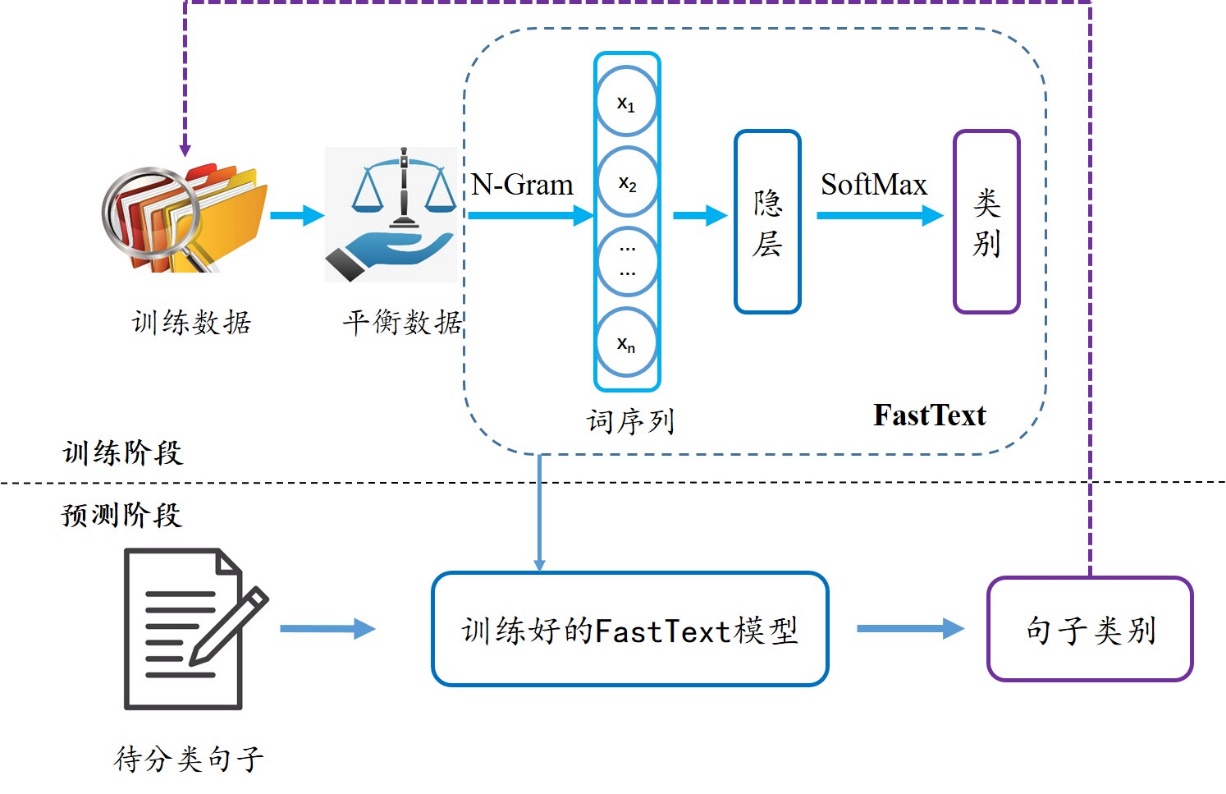
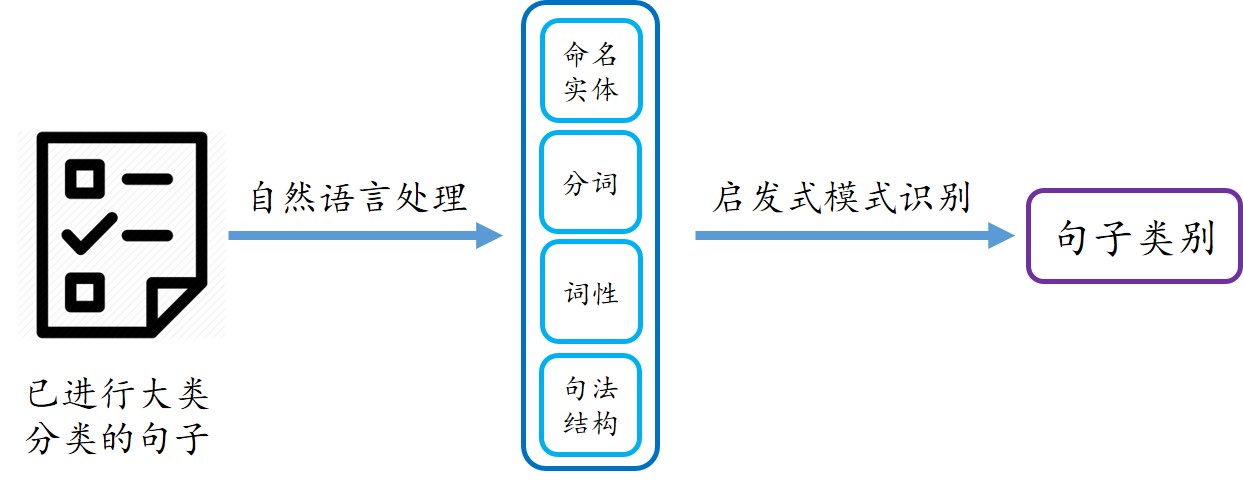
所有API文档中抽取出来的语句分为四个大类：方法调用相关约束性语句、继承相关约束性语句、其它约束性语句和非约束性语句，这对应着框架中第一阶段大类分类中的类别。方法调用相关约束性语句指的是在我们调用API某个具体方法的时候需要遵循的规则；继承相关约束性语句则是在我们继承或实现某个API接口或者类的时候，我们需要注意的与继承过程有关的一些约束；剩余的一些约束性语句全部归为其它约束性语句；最后的非约束性语句则是指的不属于约束性语句的句子，比如描述功能的、解释概念的、描述参数的等等。

图 9 细分小类分类框架图

图 10 大类分类框架图

在四个大类下面还有这一系列细分小类，这是对应着第二阶段的分类体系，非约束性语句因为不是本文研究重点，在第二阶段中不会考虑进行细分。每个大类里都含有一个细分小类“其它”，这是表示大类中未被细分的所有语句都会归到相应大类的这个细分小类，它们形态各异、不易区分，需要后续进行更深入的研究才能再进行细分，此次工作中就先不进行考虑。

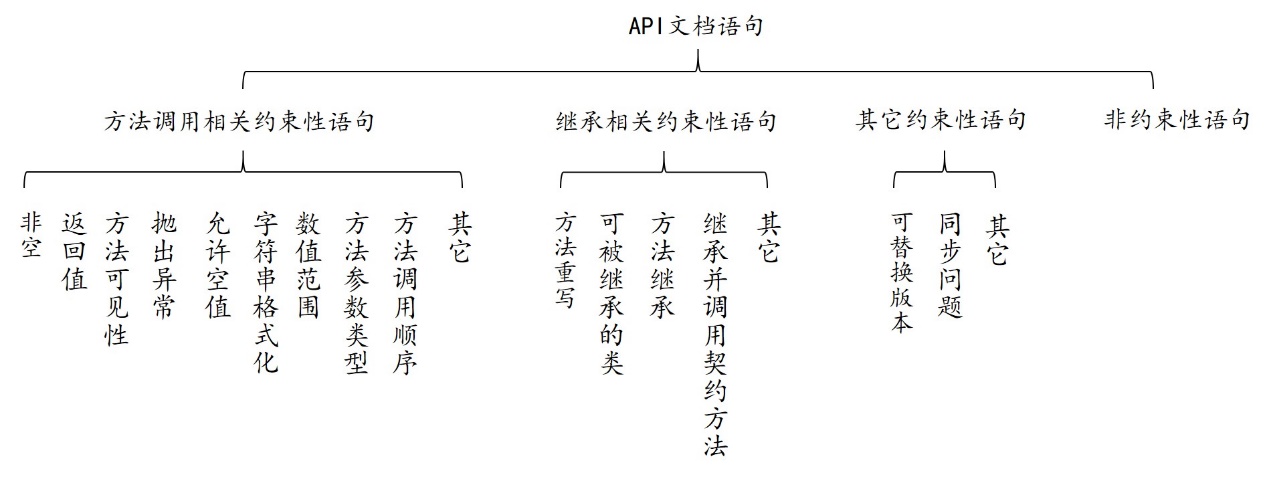
方法调用相关约束性语句可细分为非空、返回值、方法可见性、抛出异常、允许空值、字符串格式化、数值范围、方法参数类型、方法调用顺序：

图 11 语句分类图

1. 在许多情况下，面向对象语言的方法参数未明确禁止使用空值（null），但是空值是与非空值有很大区别的，甚至影响着后面是否需要非空判断。因此，细分小类“非空”就是为了规定某些参数不允许传入空值，遵守这个约束就能很好地避免后续可能因为空值而产生的缺陷。例如，“@param set the set to decorate, must not be null”这句非空约束性语句就表达了set这个参数传入时不能为空，保证了传参的正确性。
2. “返回值”规定了在调用或实现方法时返回的类或值必须遵守的约束，在方法被调用时，清晰的了解方法返回时需要注意的事项对调用者来说十分重要，而在方法被实现时，返回值约束是必须要遵循的。例如“The returned control's layout data must be an instance of GridData.”就规定了返回的layout必须是GridData的一个实例，如果错误的返回了别的类型的实例就有可能导致后续对返回值的处理出错。
3. “方法可见性”约束的是方法对调用者来说是否可见。它既可以规定方法能够被调用，甚至可以规定被调用的范围，如“Constructor only used in deserialization, do not use otherwise.”；它也可以规定方法不能被使用，如“ListUtils should not normally be instantiated.”。
4. “抛出异常”约束是为了告诉调用者可能或者已经抛出的异常，主要关注可能抛出的异常的类型、异常抛出的条件和情境、不能抛出的异常类型等等。比如，“If the input object is null, a PredicateException is thrown.”讲的就是如果传入空对象，则要抛出PredicateException异常。
5. “允许空值”是一个和“非空”相反的约束，告诉调用者方法参数可以为空、在什么情况下为空、为空时参数的含义是什么。例如“When called the first time, priorVariables should be null.”讲的就是当这个方法第一次被调用时，priorVariables这个参数必须为空值。
6. “字符串格式化”明确了传入方法的参数应该是什么形式的，字符串作为最常用的入参类型，具有简洁易读的优点，可能如果不规范其格式，可能使得开发人员会不知道里面包含的内容是什么，造成调用传参不规范。比如“The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' (\u002D') to indicate a negative value.”就规定了作为入参的字符串的每个字符都应该是一个十进制的数，除了第一个字符可以是一个减号“-”。
7. “数值范围”限定了入参里面作为数字类型（byte,int,long,double等等）的取值范围。例如“Hours must be between 0 and 23, and Minutes must be between 00 and 59”规定了传入参数中小时必须在0~23之间，分钟必须在0~59之间。在Java中，还有一种是限定了数值在enum这种类型里面取值，比如“The mode argument must be either OPEN\_READ or OPEN\_READ | OPEN\_DELETE.”。
8. “方法参数类型”规定的是方法的参数必须为什么类型，常见的类型有基本数据类型，如int，char等，也有在面向对象语言中的某个类。例如，“All elements inserted into the set must implement the {@link Comparable} interface.”表明了所有插入到集合中的元素都必须实现了Comparable这个接口，也就是说它们都是能相互比较的。
9. “方法调用顺序”规定了方法或者类的调用顺序，只有遵循这个顺序才能保证对API的调用不会出错。这里的顺序不仅仅是方法与方法之间的顺序，还包括了方法与类、方法入参的预处理、方法返回值后续处理等之间的顺序。例如“All of the columns in a result set must be given a value each time this method is called before calling insertRow.”就规定了在调用insertRow这个方法之前，必须将前面结果集合里面的每个元素都赋一个值。

除了一些方法相关的约束性以外，在面向对象语言中，继承是一个常用且重要的特性，理所当然地，API文档也会对类似功能做出一些约束，保证其正确性。继承相关约束性语句包括方法重写、可被继承的类、方法继承、继承并调用契约方法：

1. 方法重写在这里有广泛的含义，包括实现抽象方法、重写替换掉父类中的实现等。“方法重写”定义了一个方法是否期望于被子类重写，例如“Subclasses must override this method”就指定了子类必须重写这个方法。当然，也有可能不推荐重写该方法，并明确如果一定要重写需要注意的事项，例如“Overriding this method is generally not required; however, if overriding in a subclass, super.setLabelProvider must be invoked.”不要求重写这个方法，但是如果重写就要调用父类的setLabelProvider方法。
2. “可被继承的类”规定了哪些类和接口可以被继承，有些可以或者一定要被继承，而有些则不能被继承。例如“Clients are not permitted to extend this class.”就表达了调用者是不能继承这个类的；而“Clients can also extend it in their own subclass.”则告诉调用者可以在自己的子类中继承该类。
3. “方法继承”特指在重写某个父类的方法时是否需要通过super调用父类的相应方法。例如“Any subclass of RequiredModelMBean overloading or overriding this method should call super.preDeregister()”告诉调用者在继承重载或者重写这个方法时都需要通过super去调用父类的相应方法。当然，这种约束性语句也可能是告诉调用者不一定需要调用父类方法，如“Applications that override this method need not call super.paint(g)”。
4. “继承并调用契约方法”跟“方法继承”有点相似，但是它表达的是在重写父类方法时是否需要调用非父类的别的方法。例如“The default implementation of this method should call getContent() and screen the return type for a match of the suggested types.”说的就是在实现这个继承的方法时要调用getContent方法，这个方法并不属于父类，不需要利用super进行调用。

最后一种与约束性语句有关的分类就是其它约束性语句，这里面的句子形式种类多种多样，经过仔细对数据的阅读，我们总结了其中两种数量较多，形式相对通用的细分小类：可替换版本、同步问题：

1. API是一个不断更新换代的产品，每隔一段时间作者就会提供更加优化、更与时俱进的服务，那么新版本的接口就会替换掉老版本的接口为调用者提供服务。“可替换版本”就是提醒开发人员版本的更替信息，让开发人员能够及时的用上最新版的接口和功能。例如“@deprecated use StyledText.invokeAction instead”就是一个提醒使用新接口的标识。
2. “同步问题”是一种描述了当在多线程程序中使用了该接口或方法可能导致同步问题，因此给出一些约束性描述帮助开发者注意到这些问题。例如“If multiple threads access a linked hash set concurrently, and at least one of the threads modifies the set, it must be synchronized externally.”就提示开发者如果有多个线程同时读写这种集合类的时候，任何一个线程在使用前都要使用外部同步机制进行同步。

以上这些细分小类的分类并不是互斥的，也就是说某个约束性语句可能同属于多个细分小类，一句话中可能提出了很多种约束，甚至这些细分小类可能同属于不同的大类。例如“Standard customers should not overwrite this method but {@link #getValue(Object)}]]>”说的是标准用户不应该重写这个方法，而应该重写getValue方法。首先这是一个继承相关约束性语句中的方法重写约束，因为它明确了在继承时应该重写的方法。然后，这也是一个其它约束性语句中的可替换版本约束，因为它告诉调用者不要重写这个老的方法，而应该用已经替换的新版本方法来重写。其实这种一个约束性语句可以分成很多类是普遍存在的，毕竟自然语言的描述十分精妙，在短短的一句话中就可能包含了两三个关键信息点，蕴含了不同种类的约束信息。

3.3 数据平衡

在API文档的预料中，普遍存在着数据量不平衡的现象，比如在Java、JFace和Commons.collections的API官网文档里面，约有5%、7%、12%的约束性语句，这么大的比例差距会使得在训练非约束性语句和约束性语句的不同特征时造成比较大的影响。在机器学习分类模型的训练时，尽可能使得所有分类数据比例平衡是构建更加准确的分类模型的好办法，因此我们就要进行数据平衡[11, 12]工作。数据平衡就是将数据量差距过大的类别进行调整，可以标注更多数量较少的类别，也可以从数量较多的类别中挑选适量的数据进行训练，使得它和其它类别的数量相近。

在我们的工作中，数据量差距比较大的是非约束性语句，因为约束性语句的数量较少，所以我们在非约束性语句中随机挑选一定数量的句子。随机挑选是为了保证句子的多样性和全面性，数量尽量保证和其它三种大类的差距不大，使得我们训练出来的大类分类模型数据分布尽量平均，准确性尽量高。

3.4 FastText

在我们选择分类模型的时候，在深度学习模型和FastText中进行了一番思量，最后将FastText作为我们的训练的目标。FastText是Facebook公司研发并开源的一个提供简单高效的词向量表征和文本分类的模型，典型的应用场景就是有监督文本分类。它既能够得到和深度学习相近的精度，又可以在比深度学习更快的速度下运行，有更好的性能。

FastText核心主要有三个部分：N-Gram，模型架构和SoftMax，字符级N-Gram和分层SoftMax是它的特色。N-Gram主要是对句子的特征进行提取，将语料按顺序不断截取大小为N的窗口，最终形成长度为N的一系列片段。它还可以分为不同的粒度：词粒度、字符粒度。例如句子：

“This method is not intended to be overridden by subclasses.”

1. 如果按照词粒度来解析
   1. 2-Gram序列为{(This,method),(method,is),(is,not),(not,intended) … }
   2. 3-Gram序列为{(This,method,is),(method,is,not),(is,not,intended) … }
2. 而如果按照字符粒度来解析
   1. 3-Gram序列为{Thi,his,is ,s m, me,met … }

通过N-Gram得到的这些特征序列是作为文本特征的候选集，在后续模型中会对重要的语料特征进行筛选。字符级的N-Gram能够保留词向量的上下文信息，不会忽略掉单词的内部形态特征，例如“banana”和“bananas”其内部有大量相似的结构，几乎由此可以判定是十分相近或者关系密切的词。在传统的Word2vec中就会给它们每个词都生成单独的向量，忽略了这些内部形态。

随后，将这句话的所有单词和N-Gram特征输入到模型架构里面进行词向量的训练，训练过程对开发人员来说是透明的。经过隐层的训练之后，输出会经过分层SoftMax，得到句子的大类类别。分层SoftMax实际上是相对原来的扁平化SoftMax来说的，扁平化的Softmax在计算Softmax概率时需要对所有的N维特征做归一化处理，这在需要计算的类别数K非常大时是十分耗时的，时间复杂度为O(KN)。而分层SoftMax则是将分类通过出现的频度分布在一棵霍夫曼树上，于是出现越频繁的类别越容易获得，时间复杂度降低到O(N;log2K)，加快了搜索速度。根据SoftMax的各类别概率，我们选取概率较大的那个类别来作为我们分类的结果，输出出来，整个FastText的训练模型和预测模型的流程到此结束。

针对FastText分类模型，我们还做出了一些优化。根据赵雪娇等[13]的研究可知，在分类模型中不断迭代的加入新的可靠的训练集能不断提高其分类准确性。我们的做法是将每次经过FastText预测的结果提取出来，得到其SoftMax分类概率。如果最后分到的类别的概率大于0.9，我们认为这是一个属于这种类别的句子，将这些句子重新加入训练集，然后再训练一次模型。下次再有数据需要预测时，就使用最新的分类模型。

3.5 自然语言处理

经过大类分类模型训练后的约束性语句，还需要经过更进一步的加工，才能分到具体的细分小类。但是在加工之前，我们要先对句子进行自然语言处理，得到一些可以辅助我们加工的基本信息，主要是句子中蕴含的实体、句子中每个词的词性、句子的句法结构等等。我们进行自然语言处理使用的工具是Stanford CoreNLP[14]，这是斯坦福开发的自然语言处理工具包，包含了一系列自然语言处理中的功能，例如词性标注、语义分析等等。

首先要进行的是实体识别，Stanford CoreNLP本身是有实体识别功能的，主要可以识别一些常见的概念实体，例如人名、机构名、时间等等。但是我们要求的不止这些，比如类名、方法名等就是我们想识别的重点，然而Stanford CoreNLP的命名实体识别是具有通用性的，专业性并不够强，这种专业的名词需要我们自己去识别出来。因此我们在工具的基础上补充了一些简单的命名实体识别的方法。第一个补充点是识别那些符合驼峰式命名规范的词语，包括一些带“.”、“()”等标志性符号的，因为一般这种都是类名或方法名，例如“getValue”，“Math.random()”等；第二个补充点就是将API涉及的所有包名、类名、方法名形成一个实体库，每次都将所含名词在其中匹配看有没有符合的实体，但库中数据量大了以后，可能需要结合倒排索引、前缀树等手段提高识别效率。

下一步要做的是词性分析，这个是工具自带的功能，词性分析的目的是获得句子中每一个单词的词性，我们主要关注的一般是名词和动词，通过词性分析可以有效地将一部分不重要的词语过滤掉，方便后续处理。但是光有词性还不足以体现整个句子的自然语言结构，因为词与词之间是有一定关系的，这时就需要句法分析发挥作用了。句法分析可以得到句子中词语的一个依赖关系，使得中心词一目了然。如果前面分析出来的词性是两个名词挨在一起，那句法分析之后就可以知道一定有一个名词是修饰另一个名词的，还能清晰的知道句子中主谓宾的存在。句子结构信息是我们在后续模式匹配的时必须的重要参考依据，关系着生成模式的简洁性，模式匹配的准确性。

3.6 启发式模式识别

在细分小类分类框架中，启发式模式识别是一个核心步骤。因为细分小类中语句形态各异，即使是不同类别也很容易出现相似的结构，且标注数据量过少，因此使用传统的模式识别而非深度学习是一个很好的选择。人力是这个阶段需要的主要资源，因为每一个模式都是通过工人看数据总结出来的。

针对已经标注好的细分小类类型的数据，我们一行行的阅读句子，从中找到可能存在模式，然后记录下来，相同的模式只把数量加一而不会新增加一条。当阅读完整个小类的数据以后，我们选取数量大于5的模式进行记录，因为数量过少的模式可能是一种特例，也可能是因为某个特殊的API而出现的相似约束，这种就应该被去除。例如，“Subclasses may extend.”这句话里，我们得到的句子结构信息有主语“Subclasses”、情态动词又是谓语“may”和行为动词“extend”，而第一印象观察下来的模式可定为“(Subclasses) (may) (extend)”。

而在后续句子的阅读中，我们会发现很多相似的句子，例如“Subclasses can override or extend.”，“Subclasses may reimplement or extend.”，“Subclass must implement this method to create the field editors.”。我们可以将这些句子总结出来的模式进行合并重组，最后得到这样一个模式：“Subclass(es)? 情态动词 (override| extend| implement)”。这个模式中Subclass单复数都可以，情态动词代表了may、can、must等词，而后面的行为动词是标识这种模式的几个词语，跟继承重写方法有关。当然，这个模式匹配时中间可能会加入一些别的修饰词，或者后面多一些从句，但是在自然语言处理后通过句子结构信息就可以不受干扰的识别出句子。

最后，表2展示了我们总结的所有匹配模式，由表中数据我们可以看出，每个细分小类还是存在一些特点十分鲜明的模式，为了保证分类的准确性，我们尽可能使匹配模式更具体，但是这可能会导致查全率较低，因为每个小类中还存在着一些没法用模式来匹配的形态各异的语句。

表格 2 细分小类的匹配模式

|  |  |  |
| --- | --- | --- |
| 大类 | 细分小类 | 匹配模式 |
| 方法调用相关约束性语句 | 非空 | 1. @exception|@throws (be|is|are) (equal|equivalent to) null 2. non-null 3. (never|not) (be) null 4. (neither|none of) (be|is|are) (equal|equivalent to)\* null |
| 返回值 | 1. return|result（从开头匹配） 2. Return|Result|@return 3. (return|returns|returned|returning) (a|the|this) 实体 4. (value|parameter|object|实体)s? (is|be|are)? (return|returns|returned|returning) 5. (方法名) (must|should|can|could|may)? (return|returns|returned|returning) (a|the|this)? (实体|true|false|element|object|null) 6. (result|results|returned 实体|实体returned) (must|should|can|could|may)? (is|be|are) 7. (must|should|can|could|may)? return |
| 方法可见性 | 1. be necessary to use method 2. (this|it) (be|is|are) called (from|outside) 3. call (method|方法名) directly 4. ((should not)|(discouraged from)) call (method|方法名) 5. Method (should)\* not be (relied|used|referenced) 6. method is for … only 7. 方法名 should (no|not)\* be (relied|used|referenced|called|instantiated) 8. may (call|instantiate) (method|class) 9. (method|class) (may|not) be instantiated |
| 抛出异常 | 1. Exception (be|is|are) thrown 2. @exception|@throws 3. throw Exception 4. raise Exception |
| 允许空值 | 1. (@param)\* (can|could|may) ((be|is|are) (equivalent|equal) to) null 2. null (be|is|are)\* ignored 3. null (results|use|be passed in) 4. or null|null means|this parameter is null|null returns|null otherwise|null indicates|If null|Specify null to |
| 字符串格式化 | 1. (must|should)\* (be|is|are|start with|starting|ends with) (".\*"|'.\*'|string) 2. string must (contain|be) 3. the format (follow|(be taken from)) 字符串编码类型 4. must conform to字符串编码类型 5. string must be digit 6. sequence of characters must represent |
| 数值范围 | 1. (equals|>|<|=|<=|>=|<>) 数字或某特定值 2. (be|is|are) (less|greater|larger|equal|equivalent|below) (than|to) 数字或某特定值 3. (be|is|are) (less|greater|larger|below) 4. (be|is|are) (at least|at most) 数字或某特定值 5. (be|is|are) (range|between) 数字 (and|to|-|~|,) 数字 6. 实体and实体(be|is|are) the same 7. (be|is|are) (in|out of|outside) range 8. (be|is|are) (in|out) of bounds 9. (be|is|are) 数字或某特定值 10. one of … (constant|value) … (such as|:) 数字或某特定值 11. (either of|one of|(be|is|are)) ({(数字,XXX)+}| 数字或特定值各种集合形式) 12. (be|is|are) (negative|positive|non-negative|non-positive) |
| 方法参数类型 | 1. (be|is|are) of type 类型实体 2. subclass of 类型实体 3. (@throws|@exception|parameter|property|value|@param|key|element) (must|should)\* (be|is|are|implement) 类型实体 4. (be|is|are) (equivalent|come from|equal to|a class derived from|an instance of) 类型实体 5. (key|value|element|property|argument)s? (must|should) implement 类型实体 |
| 方法调用顺序 | 1. (after|before) .\*((being used)|calling|using|invoking|called) 2. (should|must) be called (whenever|when|associated with) 3. (invoked|called|call|calls) .\*(before|after) |
| 继承相关约束性语句 | 方法重写 | 1. (subclass)(es)? (reimplement|override|extend|override|extend) 2. (extend|override) method 3. method (override|overridden|re-implemented|extended|implemented) |
| 可被继承的类 | 1. (instantiate|extend) (class|interface) 2. (implemented|implement) (interface) 3. (may|should|can) subclass 4. (it|class|interface) (be|is|are) (subclassed|extended|implemented) |
| 方法继承 | 1. (call|invoke) (super.|(.\*super)) 2. (super.) (should|must) be (called|invoked) |
| 继承并调用方法契约 | 1. (subclasses|subclass) .\*(call) .\*(method|constructor) 2. If you override this method, then you should … |
| 其它约束性语句 | 可替换版本 | 1. @deprecated 2. be (preferable|more efficient) to (use)? 新版本 3. recommend to 新版本Instead 4. 旧版本be replace 5. 新版本be a better choice |
| 同步问题 | 1. thread-safe 2. synchronized |

3.7 本章小结

本章介绍了约束性语句分类的整个框架，分为两个阶段：大类分类阶段和细分小类分类阶段。大类分类阶段是先平衡各个大类的数据，然后基于FastText训练分类模型，最后用来预测语句的大类，这个阶段还可以将非约束性语句分离出去。细分小类分类阶段则利用自然语言处理和启发式模式识别技术将句子的形态结构根据每个细分小类进行总结，通过这些模式来对约束性语句句子进行分类。下面的章节将通过设计实验来验证分类框架两个阶段各自的表现，通过实验结果来验证这样分两个阶段来进行分类的合理性。

第四章 实验设计

4.1 数据来源

在我们的研究中，训练FastText的标注数据是必不可少的资源，这部分数据来自Martin Monperrus等[10]人的对API文档中约束性语句的经验研究。他们的数据主要来自Java，JFace和commons.collections三种API官方文档，这些官方文档都是以网页的形式供开发人员查阅。它们都以包分类，每个类用一个单独的网页来展示，包含类、方法、域变量等信息。Martin Monperrus等先将网页中提取的分成很多个单独的句子，然后总结了一系列可能得到约束性语句的语法模式，通过这些语法模式先对句子进行初筛。筛选出来的可能是约束性语句的句子会经过人工阅读，由此来判定这个句子是不是约束性语句，且这个句子在约束性语句中分类是什么。因此，我们可以认为这个标注数据的准确度是很高的，能够为我们训练出很好的分类模型。

表3展示了标注数据里面各个类别的数量，数量具体到每种API文档、每个大类和细分小类。由表中数据可知，JDK文档不论是约束性语句和非约束性语句的数量都是最多的，约束性语句1995句，基本上等于JFace和commons.collections的总和；非约束性语句54366句，是其它两种总和的两倍左右。这与JDK文档中API的丰富程度有很大关系，因为JDK是我们开发中最基础、特别常用的一种API。非约束性语句比约束性语句多得多，总和的比例大约是19:1，而每个大类之间的数量也不是完全对等的，方法调用相关的约束性语句数量最多，有2389句，其它约束性语句最少，只有735句，其他约束性语句数量较少也说明了大部分约束性语句能够准确的被分到某个别类里面，有利于后续的分类工作。其实这些数据量的不均衡与API文档的特性是有关系的。因为在一个API的描述网页中，很大一段话都应该是对这个API功能的介绍、概念的解释。而在阐释约束规则的时候一般都是针对某个方法，因此方法调用相关的约束性语句比较多。是否为空值是API使用时最需要考虑的地方，文档也会对这方面做出详细规定。数量较多的约束语句类别代表着它是经常被提及，总是被人们关注的，也是开发人员在使用时需要小心的潜在出错点。

* 1. 大类分类实验

大类分类是基于FastText训练分类模型而得到的一个分类器，细分小类分

表格 3 各个数据源中各类别语句的数量

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **分类** | **JDK** | **JFace** | **Col** | **总数** |
| **方法调用相关** | **1189** | **368** | **832** | **2389** |
| 非空 | 28 | 34 | 531 | 593 |
| 返回值 | 148 | 52 | 32 | 232 |
| 方法可见性 | 78 | 93 | 22 | 193 |
| 抛出异常 | 156 | 13 | 16 | 185 |
| 允许空值 | 61 | 19 | 75 | 155 |
| 字符串格式化 | 122 | 1 | 0 | 123 |
| 数值范围 | 80 | 13 | 29 | 122 |
| 方法参数类型 | 59 | 10 | 19 | 88 |
| 方法调用顺序 | 171 | 97 | 55 | 323 |
| 其它 | 286 | 36 | 53 | 375 |
| **继承相关** | **376** | **683** | **70** | **1129** |
| 方法重写 | 95 | 364 | 46 | 505 |
| 可被继承的类 | 10 | 186 | 5 | 201 |
| 方法继承 | 46 | 45 | 1 | 92 |
| 继承并调用契约方法 | 22 | 32 | 4 | 58 |
| 其它 | 203 | 56 | 14 | 273 |
| **其它** | **430** | **142** | **163** | **735** |
| 可替换版本 | 199 | 119 | 75 | 393 |
| 同步问题 | 105 | 10 | 61 | 176 |
| 其它 | 126 | 13 | 27 | 166 |
| **约束性语句** | **1995** | **1193** | **1065** | **4253** |
| **非约束性语句** | **54366** | **17632** | **8615** | **80613** |

类是基于启发式模式识别来匹配类别的，像这样用自动化的方式来完成相似的分类工作在以前的研究中是未曾出现过的。在此之前基本上是靠人工阅读然后标注的方式来完成，即时偶尔有基于模板的方式也只是涉及到了其中的某几个特别常见且形式简单易于总结的分类[7]，像我们研究这样全面且自动的方式还是第一次。因此，我们的分类指标对比没有可参照的基线，只能自己针对数据指标去进行分析解读。

为了验证大类分类框架的有效性，我们使用了机器学习在建模过程中常用的一种验证方式：十折交叉验证。在机器学习建模时，通常将数据分为训练集和验证集，训练集负责训练模型，验证集负责模型的评估，不参与训练过程。而如果需要验证集能够评估模型，那么验证集就要和训练集是一样的，都是已经分好类的数据，只不过不用于训练。这样，我们使验证集取自训练集，可以相对客观的验证分类模型对于训练集外数据的匹配程度。

我们构建训练集和验证集的方法很简单，就是将已有标注数据随机分为10份，只分这一次，后续过程全部基于这10份数据。每次选取9份数据作为训练集，剩下那1份自动归为验证集，然后用这个验证集去评估其它9份数据训练出来的分类模型。10份数据，每份数据都作为一次验证集，剩下9份作为训练集进行一次评估，因此会有10个评估结果，然后将这10次结果取平均就能得到我们对整个大类分类框架的评估结果了。

* 1. 细分小类分类实验

在细分小类分类阶段，输入的待分类语句都是前一阶段已经分好大类的数据，即他们的大类类别是已知的。因此我们要验证细分小类分类框架时，我们把每一个大类的数据单独输入，然后用那个大类的匹配模式去识别句子的细分小类的类别，跟句子标注的类别作对比，看看正确与否。

例如，在标注数据中方法调用相关的约束性语句有2389条，而我们在前面定义过在方法调用相关大类下的每种细分小类的匹配模式，我们将这2389条数据用这些模式去分类，不需要去理会别的大类下的匹配模式，然后看分类的正确性。如果一个句子被某个细分小类的匹配模式识别出来，那么这个句子都定义为相应的细分小类，而如果所有的匹配模式都没有识别，这个句子就会被分为这个大类下的“其它”类别。

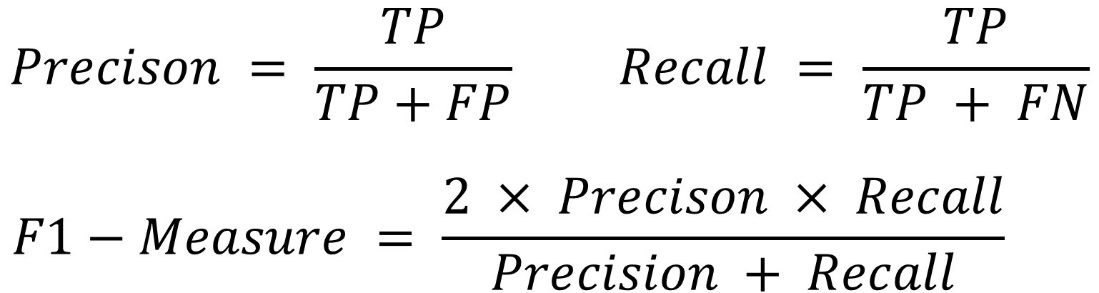
* 1. 评价指标

图 12 精准率、召回率、F1-度量值的计算公式

针对框架的评估，我们使用的是三个非常常见的评价指标：精准率（Precision）、召回率（Recall）和F1-度量值（F1-Measure），计算公式如图所示。精准率是正确被检索到的数量占所有被检索到的数量的比例，而召回率是正确被检索到的数量占所有应该被检索到的数量的比例，F1-度量值是精准率和召回率的调和均值，可以调整精准率和召回率在其中的权重，是一个综合性的度量指标。

我们的工作主要是评估分类框架，不论大类分类或者细分小类分类，评估的原理都是一样的。假设现在针对类别“非空”，我们判断一个句子是不是这个类别的，如表4所示，可以得到以下四种结果：本来是“非空”类别的句子，如果能够正确的被分到“非空”这个类别，这就是TP - 正确且被检索到，如果被分到别的类别，就是TN – 正确但未被检索到；而对于一个本来就不是“非空”类别的句子，如果被分到“非空”这个类别，就是FP – 错误但被检索到，如果被分成别的类别，就是FN – 错误且未被检索到。

针对这四个值，我们可以得到精确率就是被分成“非空”这个类别的句子中，分类正确的比例；召回率就是本来是“非空”类别的句子，被正确分到这个类别的比例。如果想要提高精确率，就要减少错误被分到此类别的语句；如果想要提高召回率，就要把此类别的语句尽可能的找出来，但是这个两种情况恰恰是存在

表格 4 分类相关精确率和召回率指标解释

|  |  |  |
| --- | --- | --- |
|  | 相关，正确 | 无关，错误 |
| 被分成此类 | True Positives（TP，本来是此类，且被分成此类） | False Positives（FP，本来不是此类，但被分成此类） |
| 未被分成此类 | True Negatives（TN，本来应是此类，但未被分到此类） | False Negatives（FN，本来不是此类，也未分到此类） |

一定矛盾的。因为如果我们想要减少误分类提高精确率，可以提高准入门槛严格把控，只有确定性很高的句子才分入此类，但是这会造成很多跟分类标准不是那么接近但是其实也是此类的句子被拒之门外，此时精确率提高了但是召回率降低了。而如果想尽可能多的把这个类别的所有句子都识别出来，就放低准入门槛，所以看起来有点像这个类别的句子都归入进来，这样保证尽可能把所以这个类别的句子都找出来，但是会带入很多别的类别的句子，这时精确率就被降低了。所以在两种指标的需求中寻找一个适合自己的平衡点是很重要的。

因此，在具体实践中，如果人们需要对其中某个指标有所偏重，F-度量值就是用来综合考量精确率和召回率的指标，通过这个指标可以观察这个系统的综合能力。F1-度量是比较常用的一种F-度量值，认为精确率和召回率的权重是一样的，而我们可以调整权重α来加重其中一种指标的影响力，得到Fα-度量值。

* 1. 本章小结

本章介绍了整个研究的数据来源、大类和细分小类的分类框架验证试验、整个实验的评价指标。数据主要来源于已有的标注数据，数据准确性高，可作为训练集和验证集。大类分类框架的验证使用了十折交叉验证法，证明了模型对训练集以外的数据同样有效。细分小类非分类框架验证直接将所有数据作为输入，但是根据大类类别来分别匹配，保证了前一阶段对后一阶段的隔离性。实验的评价指标使用了常见的精确率和召回率，对模型的各方面都进行了体现。下一章将对实验结果进行展示，并对产生这样结果的原因进行剖析，找到模型的优缺点与合理性。

第五章 实验结果

为了更好地对实验结果进行阅读解析，我们提出了三个研究问题，研究问题结合实验结果能让我们有条理的了解到整个研究工作的效果如何：

1. 大类分类框架表现怎么样？
2. 细分小类分类框架表现怎么样
3. 我们将整个分类过程分成两个阶段的合理性？
   1. 大类分类结果

针对第一个研究问题，我们可以先看一下大类分类实验的结果，如表5所示，并观察其中存在的规律。表中展示了大类分类框架验证实验中十折交叉验证法得到的4个大类10次结果的3个指标，还将10次结果做了平均，得到了就是整个框架不同类别的性能数据，所有指标均保留一位小数。

非约束语句的平均精确率和平均召回率都是最高的，分别为99.5%和99.8%，从而使得其F1-度量值也是最高的99.6%，甚至有几次精确率和召回率都达到了惊人的100%。这说明了约束性语句和非约束性语句之间的区别还是很大的，主要体现在句子形态上。非约束性语句一般为功能描述、概念解释、数值定义等，句式为陈述句较多，不带有任何命令的意思，例如“The BigDecimal class provides operations for arithmetic, scale manipulation, rounding, comparison, hashing, and format conversion.”只是表达了BigDecimal这个类的功能作用，没有要求开发人员做任何别的操作。另一方面，约束性语句一般规定了某些注意事项，多为祈使句和假设句，有需要开发人员注意或者采取某些行动的意思，经常伴随有情态动词的出现。例如“Clients should not call this method directly”里面不仅包含情态动词“should”，句子还是祈使句，告诉开发人员不要干什么。或者又如“Subclasses should override configureShell if the shell needs to be customized.”同样是约束性语句，但是它的句式中还带有一种假设的场景，意思是在某种情况下应该怎么做。在语句形态上的区别，使得在经过大量样本训练情况下区分约束性语句和非约束性语句是相对准确的，因此这样的高精确率高召回率的结果并不是意料之外。

方法调用相关约束性语句的分类指标是比较优秀的，精确率85.3%，召回率93.3%，使得F1-度量也有89.2%，这在分类指标中算是很好的了。

表格 5 大类分类框架实验结果

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 类别 | 指标 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | **平均** |
| 方法调用相关 | 精确率 | 82.2% | 86.5% | 86.4% | 84.9% | 86.0% | 81.9% | 88.8% | 83.0% | 87.5% | 86.2% | **85.3%** |
| 召回率 | 95.5% | 94.5% | 90.8% | 94.5% | 87.8% | 95.4% | 93.7% | 92.0% | 94.5% | 94.5% | **93.3%** |
| F-M | 88.4% | 90.4% | 88.6% | 89.5% | 86.9% | 88.2% | 91.2% | 87.3% | 90.9% | 90.2% | **89.2%** |
| 继承相关 | 精确率 | 89.3% | 84.9% | 81.6% | 84.1% | 78.6% | 89.4% | 90.9% | 86.6% | 83.9% | 89.0% | **85.8%** |
| 召回率 | 76.1% | 80.4% | 71.4% | 70.6% | 82.1% | 67.9% | 79.5% | 81.3% | 83.9% | 79.5% | **77.3%** |
| F1-M | 82.2% | 82.6% | 76.2% | 76.7% | 80.3% | 77.2% | 84.8% | 83.9% | 83.9% | 84.0% | **81.2%** |
| 其它 | 精确率 | 79.0% | 83.6% | 70.5% | 77.8% | 69.8% | 72.9% | 73.6% | 71.7% | 82.4% | 73.3% | **75.5%** |
| 召回率 | 60.0% | 63.9% | 69.5% | 68.1% | 61.1% | 59.8% | 73.6% | 52.8% | 58.3% | 61.1% | **62.8%** |
| F1-M | 68.0% | 72.5% | 69.9% | 72.6% | 65.2% | 65.6% | 73.6% | 60.8% | 68.3% | 66.7% | **68.3%** |
| 非约束性语句 | 精确率 | 100% | 99.0% | 98.5% | 100% | 100% | 99.0% | 99.5% | 100% | 99.0% | 99.5% | **99.5%** |
| 召回率 | 99.5% | 99.5% | 100% | 100% | 99.5% | 99.5% | 100% | 100% | 100% | 100% | **99.8%** |
| F1-M | 99.7% | 99.3% | 99.3% | 100% | 99.7% | 99.3% | 99.8% | 100% | 99.5% | 99.7% | **99.6%** |

5.2 细分小类分类结果

5.3 分类两阶段合理性

5.4 本章小结

第六章 应用

第七章 相关工作

第八章 总结与未来工作

参考文献

**参考文献**

[1] PICCIONI M, FURIA C A, MEYER B. An Empirical Study of API Usability: Acm, 2013[C].

[2] MAALEJ W, ROBILLARD M P. Patterns of Knowledge in API Reference Documentation[J]. IEEE Transactions on Software Engineering, 2013,39(9):1264-1282.

[3] HE J, ZHANG J, REN Z, et al. An Unsupervised Approach for Discovering Relevant Tutorial Fragments for APIs: IEEE/ACM International Conference on Software Engineering, 2017[C].

[4] PETROSYAN G, ROBILLARD M P, MORI R D. Discovering Information Explaining API Types Using Text Classification: IEEE/ACM IEEE International Conference on Software Engineering, 2015[C].

[5] EARLE R H, ROSSO M A, ALEXANDER K E. User preferences of software documentation genres: International Conference on the Design of Communication, 2015[C].

[6] LI H, XING Z, XIN P, et al. What help do developers seek, when and how? Working Conference on Reverse Engineering, 2013[C].

[7] YU Z, GU R, CHEN T, et al. Analyzing APIs Documentation and Code to Detect Directive Defects: IEEE/ACM International Conference on Software Engineering, 2017[C].

[8] MAALEJ W, ROBILLARD M P. Patterns of Knowledge in API Reference Documentation[J]. IEEE Transactions on Software Engineering, 2013,39(9):1264-1282.

[9] JOULIN A, GRAVE E, BOJANOWSKI P, et al. Bag of Tricks for Efficient Text Classification[J]. 2016.

[10] MONPERRUS M, TEKES E, MEZINI M. What should developers be aware of? An empirical study on the directives of API documentation[J]. Empirical Software Engineering, 2012,17(6):703-737.

[11] KAMEI Y, MONDEN A, MATSUMOTO S, et al. The Effects of Over and Under Sampling on Fault-prone Module Detection[J]. First International Symposium on Empirical Software Engineering & Measurement, 2007,11(1):196-204.

[12] HAN H, WANG W Y, MAO B H. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning: International Conference on Advances in Intelligent Computing, 2005[C].

[13] ZHAO X, XING Z, KABIR M A, et al. HDSKG: Harvesting Domain Specific Knowledge Graph from Content of Webpages: IEEE International Conference on Software Analysis, 2017[C].

[14] MANNING C D, SURDEANU M, BAUER J, et al. The Stanford CoreNLP Natural Language Processing Toolkit, 2014[C].

**校对报告**

当前使用的样式是 [中华人民共和国国家标准\_GBT\_7714-2015]

当前文档包含的题录共17条

有0条题录存在必填字段内容缺失的问题

所有题录的数据正常

致谢

1. https://en.wikipedia.org/wiki/Application\_programming\_interface [↑](#endnote-ref-1)
2. https://swagger.io/blog/api-documentation/what-is-api-documentation-and-why-it-matters/ [↑](#endnote-ref-2)
3. https://zh.wikipedia.org/wiki/Softmax%E5%87%BD%E6%95%B0 [↑](#endnote-ref-3)
4. https://www.slideshare.net/tw\_dsconf/ss-62245351?qid=108adce3-2c3d-4758-a830-95d0a57e46bc&v=&b=&from\_search=3 [↑](#endnote-ref-4)
5. https://en.wikipedia.org/wiki/Natural\_language\_processing [↑](#endnote-ref-5)
6. https://en.wikipedia.org/wiki/Program\_analysis [↑](#endnote-ref-6)
7. https://en.wikipedia.org/wiki/Abstract\_syntax\_tree [↑](#endnote-ref-7)