**生成代码片段的自动化标签抽取与应用**

摘 要

随着网络资源日益丰富，开发者获取和掌握编程语言、基础类库、开源类库和更多丰富第三方API的渠道日益增强。随着开源社区的蓬勃发展和开源API的广泛使用，应用软件开发过程中，开发者常常通过网络搜索获得API使用范例和相关开发知识。而目前大多数面向程序设计问答的网站或社区，只支持严格的文本匹配搜索模式。此外问题或范例的分类一般由贡献者手工标注，因此分类标签的效果完全依赖于提问者的领域知识和个人习惯以及经验，在使用者搜索过程中匹配的有效性和可用性较差。

针对于代码本身不同于普通文本的特点，本文以自然语言处理为基本手段，重点结合python[1]的语法、词法以及结构化特点，以程序设计领域的问答网站（以Stack Overflow为例）和python官方文档为辅助，并应用python特定框架Django使用的领域知识，设计并实现了一个初步可应用于程序设计领域问答网站的python代码片段自动标签生成模型。

该模型最终生成的标签分别从代码片段，问答文本内容，问题标题中分析获取。选取Stack Overflow的python问答数据集的数据，过滤内容不全的劣质数据，对问答文本内容进行自然语言处理[4]，包括分词、去除标点和停用词、选取名词、词性还原，训练导出字典，然后构建保存TF-IDF模型[6]和LDA模型[7]，将经过预处理的用户的文本部分输入模型，一方面选取TF-IDF值高的词来实现关键词提取，另一方面根据LDA模型进行相似度计算，从数据库中选取和输入相似度高的文本并获取它们自带的标签进行推荐。通过对程序设计问答网站提问者或者回答者提供的代码片段中库的引用和函数使用进行分析，结合爬取的python和Django官方文档中所有标准库信息，获取该代码片段使用的类和包作为代码关键词进行推荐。针对经过预处理的问题标题，重点提取代码语言和python框架信息。同时，最终系统也支持用户手动删除某些机器标签和增加人工标签。

**关键词：**自然语言处理，代码分析，自动化标签，TF-IDF，LDA潜在狄利克雷分配

**Application of Bjerrum Function in Determination of Stability Constants**

**ABSTRACT**

As network resources become more and more abundant, developers have more channels to acquire and master programming languages, basic libraries, open source libraries, and enriching increasingly third-party APIs. With the flourishing development of the open source community and the wide use of open source APIs, developers often obtain API usage examples and related development knowledge through web search. At present, most websites or communities for programming questions and answers only support strict text matching search mode. In addition, the classification of questions or examples is usually marked manually by contributors. Therefore, the effectiveness of classification tags depends entirely on the domain knowledge of the questioner and personal habits or experience. The effectiveness and usability of code examples matching in the user search process are poor.

For the characteristics of the code itself is different from ordinary text, this article designed and implemented a python code fragment automatic tag generation model that can be initially applied to the Q&A(question and answer) website of the programming field. It uses natural language processing as a basic means, focusing on the grammar, lexical and structural characteristics of python, referring to the Q&A website (in the example of Stack Overflow) and python official documents ,and applying the domain knowledge of python framework Django.

The final generated tags of the model are obtained from the analysis on code fragments, the content of the question and answer text, and the title of the problem. The primary python Q&A dataset are selected from Stack Overflow. The model filtered out incomplete data and performed natural language processing on the Q&A text, including word segmentation, removing punctuation and stop words, selecting nouns, and lemmatization, training to export the dictionary, and then built and saved the TF.- IDF model and LDA model. when pre-processed text part of the user is input into the model, on the one hand, the words with high TF-IDF value will be extracted as the tags. And on the other hand, the model calculates the similarity with other texts in the dataset according to the LDA model and get existed labels of texts in database with high similarity for recommendation. By analyzing the reference and function usage of the library in the code fragments provided by user, the classes and packages used in the code fragment are obtained as Code tags for recommendations according to all the standard library information of python and Django official documents. The tag extracting of pre-processed question titles is focus on code language and python framework. At the same time, the final system also supports users manually deleting certain machine tags and adding manual tags.

**Key words：**stability constants, bjerrum function, potential titration, acid, coordination compound

目 录

[1 引 言 1](#_Toc515324477)

[1.1 课题背景和意义 1](#_Toc515324478)

[1.2 课题目标与研究内容 2](#_Toc515324479)

[1.3 系统参数 2](#_Toc515324480)

[1.4 本文所做的工作 3](#_Toc515324481)

[2 理论部分 4](#_Toc515324482)

[2.1 数据库MongoDB 4](#_Toc515324483)

[2.2 自然语言处理 4](#_Toc515324484)

[2.2.1 NLTK自然语言处理包简介 5](#_Toc515324485)

[2.2.2 预处理 5](#_Toc515324486)

[2.2.3 分词和停用词处理 5](#_Toc515324487)

[2.2.4词性标注、词干化处理和词性还原 6](#_Toc515324488)

[2.2.5 文本数据结构化 7](#_Toc515324489)

[2.3 主题模型 10](#_Toc515324490)

[2.4 文本数据可视化 11](#_Toc515324491)

[2.5 Beautiful Soup解析HTML 12](#_Toc515324492)

[2.6 开发技术 12](#_Toc515324493)

[2.6.1 Django 12](#_Toc515324494)

[2.6.2 React 12](#_Toc515324495)

[2.7 开发环境与参数 12](#_Toc515324496)

[3 系统设计 14](#_Toc515324497)

[3.1 Use Case 14](#_Toc515324498)

[3.2 概要设计 15](#_Toc515324499)

[3.2.1 数据库选择和设计 15](#_Toc515324500)

[3.2.2 系统流程图 16](#_Toc515324501)

[3.3 系统架构设计 20](#_Toc515324502)

[3.4 设计细节 20](#_Toc515324503)

[3.4.1 前端设计 20](#_Toc515324504)

[3.4.2 接口设计 20](#_Toc515324505)

[4 标准化和预处理 22](#_Toc515324506)

[4.1 数据标准化 22](#_Toc515324507)

[4.1.1 合并数据集 22](#_Toc515324508)

[4.1.2 提取代码片段和纯文本 23](#_Toc515324509)

[4.2 预处理 24](#_Toc515324510)

[4.2.1 问题标题预处理 25](#_Toc515324511)

[4.2.2 文本预处理 27](#_Toc515324512)

[4.2.3 代码预处理 30](#_Toc515324513)

[5 文本分析 33](#_Toc515324514)

[5.1 TF-IDF模型 33](#_Toc515324515)

[5.1.1 语料库与向量空间 33](#_Toc515324516)

[5.1.2 TF-IDF转化 33](#_Toc515324517)

[5.1.3 Gensim中可用的转化 35](#_Toc515324518)

[5.2 主题模型 35](#_Toc515324519)

[5.2.1主题模型的创建 35](#_Toc515324520)

[5.2.2 相似度计算 38](#_Toc515324521)

[5.3 模型的储存和加载 39](#_Toc515324522)

[5.4 主题模型详解和实验 42](#_Toc515324523)

[5.4.1 机器学习理论基础 42](#_Toc515324524)

[5.4.2 分类和聚类 42](#_Toc515324525)

[5.4.3 主题模型 43](#_Toc515324526)

[5.4.4 主题模型参数实验 43](#_Toc515324527)

[6 代码分析 50](#_Toc515324528)

[6.1 代码分析的流程探究 50](#_Toc515324529)

[6.2 Python的模块、包、类 50](#_Toc515324530)

[6.3 获取标准库函数的父级 51](#_Toc515324531)

[6.4 分析流程 53](#_Toc515324532)

[7 标签管理和评价 56](#_Toc515324533)

[7.1 标签生成和管理 56](#_Toc515324534)

[7.2 标签种类和等级 57](#_Toc515324535)

[7.3 标签质量评估方法 57](#_Toc515324536)

[7.4 评估结果 59](#_Toc515324537)

[8 系统整体框架和部署 61](#_Toc515324538)

[8.1 前后端框架 61](#_Toc515324539)

[8.2 项目目录结构 61](#_Toc515324540)

[8.3 前端界面展示 61](#_Toc515324541)

[8.4 系统部署 65](#_Toc515324542)

[9 总结与展望 66](#_Toc515324543)

[参考文献 67](#_Toc515324544)

[谢 辞 68](#_Toc515324545)

# 1 引 言

## 1.1 课题背景和意义

随着网络资源日益丰富，开发者获取和掌握编程语言、基础类库、开源类库和更多丰富第三方API的渠道日益增强。随着开源社区的蓬勃发展和开源API的广泛使用，应用软件开发过程中，开发者常常通过网络搜索获得API使用范例和相关开发知识。而目前大多数面向程序设计问答的网站或社区，只支持严格的文本匹配搜索模式。此外问题或范例的分类一般由贡献者手工标注，因此分类标签的效果完全依赖于提问者的领域知识和个人习惯以及经验，在使用者搜索过程中匹配的有效性和可用性较差。如果能针对程序设计问答的网站或社区中代码资源的特点，结合问答的文字描述，自动对代码片段进行分类和标注，配合后期人工修改调整，有利于开发者更好的搜索和学习。同时这一系统也可以进行扩展和改进，用于网络资源知识化管理、源代码搜索和源代码资产自动标注。

然而目前常用的预处理、文本分析的对象都是日常语言的文本，包括新闻、微博等，但是针对结构性强、严格遵守语法的计算机语言，始终缺乏通用有效的分析流程，所以代码分析可以看作是在普通文本分析的基础上，结合和代码相关的文本，利用代码的语法规则和结构特性结合分析目的，有效提取代码信息。本课设选取了python相关的代码片段进行分析，以小见大，如果之后进一步完善可以加入其它语言比如java的分析。

文本处理常用的方法是使用斯坦福大学开发的nltk自然语言包对文档的集合进行预处理，包括分词、去除标点和停用词、词性还原，最后可以得到用若干个单词表示每个文档的集合，并生成一个对应于该文档集合的词典。为了将每一个文档转化成向量，一般使用词袋模型（bag-of-words）来表示每个文档，得到语料库（corpus）。这一步实现了文档的向量化，每个文档可以用一个向量来表示。在这个向量空间的基础上，可以进一步构建TF-IDF和LDA模型。TF-IDF的核心思想是如果某个词或短语在一篇文章中出现的频率很高，并且在文档集合的其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。因此利用TF-IDF模型可以找到文本的关键词。LDA模型假设文档是由一系列主题构成的，然后再从这些主题中依据相应的概率分布生成词语。通过调整参数主题数和组成每个主题的词语个数，给定文档集合可以训练得到一系列由词语和相应概率组成的主题，以及每篇文章和每个主题的相似度。又或者在预处理之后，可以通过句法分析和词法分析得到文本中每句话的语义。这些传统的文本处理方法针对是日常语言的文本，没有考虑代码语言的语法、词法以及结构化特点。单纯套用针对类似新闻一类的文本进行分析的方法，找到的关键词在大部分情况下不够准确，有些提取出来的关键词和代码的关联性较差，不能很好的对代码资源进行标注和分类。由于目前在代码分析领域不存在类似传统文本分析经典的分析方法,所以这也是本课题研究的重点，在文本分析的基础上，需要加入根据特定语言的特点对代码类结构和函数结构的解析，结合官方文档，设计一种提取代码关键词的算法。

## 1.2 课题目标与研究内容

本设计选取了和python有关的问答数据，针对程序设计问答的网站或社区中代码资源片段短小零散的特点，结合python特定框架Django使用的领域知识，配合python和Django的官方文档，进行代码分析处理，实现了一个初步可应用于程序设计领域问答网站的python代码片段自动标签生成模型。

该模型最终生成的标签分别从代码片段，问答文本内容，问题标题中分析获取。选取Stack Overflow的python问答数据集的数据，过滤内容不全的劣质数据，对问答文本内容进行自然语言处理，包括分词、去除标点和停用词、选取名词、词性还原，训练导出字典，然后构建保存TF-IDF模型和LDA模型，将经过预处理的用户的文本部分输入模型，一方面选取TF-IDF值高的词来实现对的关键词提取，另一方面根据LDA模型进行相似度计算，选取和输入相似度高的文本并获取它们自带的标签进行推荐。通过对程序设计问答网站提问者或者回答者提供的代码片段中库的引用和函数使用进行分析，结合爬取的python和Django官方文档中所有标准库信息，获取该代码片段使用的类和包作为代码关键词进行推荐。针对经过预处理的问题标题，重点提取代码语言和python框架信息。同时，最终系统也支持用户手动删除某些机器标签和增加人工标签。

为了实现这个功能，本设计涉及以下几个方面。（1）从软件工程的角度去思考整个系统运行的流程以及功能模块，设计出系统模块架构图，根据功能点将较大的功能模块分成功能点更单一的子模块，特别是要分析提取公用的功能模块，降低耦合性。

（2）了解文本分析的预处理的基本流程和方法，使用斯坦福大学开发的nltk自然语言包对文档的集合进行预处理，并结合问答数据的特点进行修改调整，并建立整个集合的语料库和词典。

（3）学习文本分析的经典模型和方法，理解布尔模型、向量空间模型等相关模型，通过TF-IDF模型从语料库和文本中获取文本内容的关键词。

（4）在文本分析基础上结合python的语法和结构去分析挖掘代码中信息，获取和代码直接相关的关键词，探究效果良好的代码分析方法。

（5）理解机器学习，分类和推荐的关系和理论基础，学习并试验基本的分类算法、聚类算法和主题模型。通过实验结果对比，和与代码关键词提取的契合度，得到最终的方案。

（6）利用非关系型数据库，构建并导出的TF-IDF模型、主题模型，代码分析模型，展示完整的代码片段生成自动化标签的流程，和此过程中的关键代码。

（7）明确前后端设计方案，运用前后端分离的思想，前端使用react，通过RESTful API 实现数据交互，完成问答代码片段自动生成标签的系统构建。

## 1.3 系统参数

根据上述的研究背景和系统目标，最终确定了以下的系统中涉及到的技术参数加以简述与罗列，会在后文中进一步详细介绍。

（1）为了选取高质量和具有代表性的代码范例，选取Stack Overflow 网站的python问答数据作为代码库和文本库的主要数据源，数据量为26万条。在对数据进行标准化处理，选取质量较高的数据大约14万条

（2）使用MongoDB 作为源数据的数据库，MongoDB更适合长篇文档的存储，且表结构相对灵活的场景使用。由于本系统需要存储大量问答的文本和代码，所以选择了MongoDB。

（3）使用正则和Beautiful Soup 等结构化代码提取工具，提取代码、链接及其他有用的数据。

（4）采用NLTK自然语言处理工具，对数据进行预处理，包括分词，去除标点和停用词，词性还原等。通过合适的预处理能从文本中更准确的提取关键词，去除无效信息和干扰。

（5）使用TF-IDF模型和向量空间模型，根据语料库对文本进行关键词提取。

（6）采用主题模型中的潜在狄利克雷分配模型，训练导出问答网站Stack Overflow的数据中的模型和字典。通过向量运算得到最接近的问答，获取并推荐这些问答的tag。

（7）前后端分离，将系统分成前端和数据模型分析两个模块，前端使用react框架。

## 1.4 本文所做的工作

本文将根据上述的技术流程展开，从理论、实验和实际工程相结合的方式，展示并阐述每个模块的实现过程，并解释他们之间的层次调用以及耦合关系。

本章主要阐述了课题背景，以及代码片段自动化标签生成的设计理念和系统范畴的确定。

（1）第二章将简述本系统用到的主要模型和技术以及其原理和优点。

（2）第三章将从软件工程的角度，分析系统的具体需求和对应的设计思路，为理论的选取与深⼊学习提供依据，并展示系统架构图和用例图。

（3）第四章介绍如何对开源数据集进行标准化和预处理，方便后面的分析。

（4）第五章将会逐一展开文本分析具体流程，详细介绍如何对文本部分提取标签，并生成TF-IDF模型和主题模型。

（4）第六章将会介绍代码分析具体流程，在代码片段的基础上参考官方文档进行解析，生成代码标签。

（6）第七章将会阐述对生成标签进行管理以及质量的评判，设立一套较为科学的划分标准，并带入测试数据进行检测，最终可得到量化的结果。根据结果对系统的分析流程将进行一定的调整。

（5）第八章将从整个系统角度，展示模块化系统框架搭建和最终实现。

# 2 理论部分

本章会逐一介绍系统构建和最终实现的完整过程中涉及算法模型和开发技术，也会罗列开发和部署运用到的库和环境等参数。从数据源的选取开始，选取的是Stack Overflow的python的问答数据集，存储到非关系型数据库MongoDB中。使用NLTK自然语言处理工具包对数据进行预处理，并对文本集合生成语料库和词典。同时也使用了主题模型中潜在狄利克雷分配模型对所有的语料进行主题的提取，用于寻找和输⼊相关的问答内容进行标签的推荐。爬取python以及框架Django的官方文档，使用Beautiful Soup进行解析，获取python和Django标准库中的包，类和函数以及它们的从属关系，用于分析代码片段调用的函数从属的类或包。最终的系统构建需要React和Django 等前后端框架进行构建。

以下对上述模型与技术进⾏简要的原理分析并对技术参数和环境做出详细的罗列。

## 2.1 数据库MongoDB

MongoDB是一个面向集合、模式自由的文档型数据库。

面向集合指的是数据被分组到若干集合,这些集合称作聚集(collections)，类似于于关系型数据库中的表。在数据库里每个聚集有一个唯一的名字，可以包含无限个文档。两者的区别是和表相比，聚集不需要进行模式定义。

模式自由是指数据库并不需要将存入到聚集中的文档的任何结构信息，即聚集不需要进行模式定义。因此完全可以在同一个聚集中存储不同结构的文档，而关系型数据库中表中所有数据都必须有相同的字段。

文档型是指我们存储在聚集中的数据是键-值对的集合，键是字符串,值可以是数据类型集合里的任意类型,包括数组和文档，可以表示为{键：值}。这个数据格式被称作BSON，一个轻量级的二进制数据格式。使用BSON可以提高存储效率，且即使在最坏的情况下，BSON格式也比JSON格式在最好的情况下存储效率高。此外，BSON格式的编码和解码都是非常快速的。

MongoDB适合实时的插入，更新与查询，并具备应用程序实时数据存储所需的复制及高度伸缩性。同时，BSON数据格式非常适合文档化格式的存储及查询，在海量⽂本数据的情况下，它的读写速度可以达到MYSQL的十倍以上。聚集不需要进行模式定义，结构松散，可以灵活处理复杂的数据结构。由于本系统选取数据集Stack Overflow的python的问答数据集中有大量的文本和代码片段，因此选用非关系型数据库MongoDB性能更优。

## 2.2 自然语言处理

自然语言文本数据的分析流程与传统数据挖掘是相似之处的，但文本数据表现为非结构性、自由形态的文字，或者是由符合特定计算机语言的语法及语法规则构成的文字和语句（比如说代码语言python,c,c#等），利用现有数据挖掘方法是无法直接进行分析的。抛开复杂的语义结构，挖掘过程首先要考虑将这种非结构化的数据进行结构化处理，常规处理流程就是预处理分词、生成文本-词频矩阵，后续可能会涉及到高维矩阵的处理等问题。

### 2.2.1 NLTK自然语言处理包简介

NLTK是由宾夕法尼亚大学计算机和信息科学使用python语言实现的一种自然语言工具包，其收集的大量公开数据集、模型上提供了全面、易用的接口，涵盖了分词、词性标注(Part-Of-Speech tag, POS-tag)、命名实体识别(Named Entity Recognition, NER)、句法分析(Syntactic Parse)等各项NLP领域的功能。

### 2.2.2 预处理

本设计使用NLTK自然语言工具包分别对问答数据的问答文本，代码片段以及问题标题进行不同的预处理。原始数据中问答内容同时包含了问答文本、代码片段和链接，需要用正则分别将它们提取出再进行预处理。预处理的基本流程为对每篇文本进行[分词token化]----> [字母转换成小写]----> [词性标注]----> [筛选出名词词性的词语]----> [去除停用词和英文标点]----> [词性还原] ----> [得到该篇文本的词语列表]。

对问题标题的预处理考虑到文本只有一句的长度并且代码信息比较多，所以优先提取代码语言信息和框架信息。代码语言指的是问题或者回答涉及了哪几种代码语言，框架信息指在该问题是否是在python的框架Django的情况下提出的。如果获取到了“Django”这一框架信息，在代码片段的处理中将要进行不同的分析操作。之后再进行[分词token化]----> [字母转换成小写]----> [词性标注]----> [过滤动词和数词] ----> [去除停用词和英文标点] ----> [得到该标题的词语列表]。

针对代码片段则要考虑代码结构，首先要提取注释以及代码片段引入标准库包、类以及函数。在python中，注释的一般形式是#.....\n以及””” ..…”””,引入的一般形式是from a import b 或者是import a ,需要用正则将这些内容提取出来。除此之外，代码片段中出现的函数名也需要提取以便进行后面的分析。

### 2.2.3 分词和停用词处理

与中文分词相比，英文分词要简单很多。一般的，英文中的单词之间是以空格作为自然分界符组成语句，语句之间再利用标点分隔组成大篇幅文本，所以我们可以简单的利用标点进行分句处理，利用空格进行分词处理。对于结构比较简单的文本，可以自行编写函数进行处理，但是对于结构、内容非常复杂的英文文本，就很难进行全面、细致的处理了。NLTK为开发者提供了一套更为专业的英文分词工具，相比于调用Python的内置函数，NLTK的英文分词工具模式更加丰富，并且在去除停用词、词干化处理方面更为优秀。

tokenize是NLTK的分词包，其中的函数可以识别英文词汇和标点符号对文本进行分句或分词处理，也允许开发者使用正则表达式自定义区进行分词。tokenize分词包包括分句函数sent\_tokenize，分词函数word\_tokenize以及regexp模块(正则表达式分词)、stanford模块，sexpr模块、util模块。

考虑到模型和最终用户输入的文本是和代码语言python相关的问答，在涉及包，库，类，函数的时候，会出现蛇形命名的词（例如“mod\_python”，用下划线把多个词连起来），也会出用点把多个词连接成一个词的情况（例如“django.contrib.admin”）。类似的词语如果按照英文文本常用的根据标点分词的做法，mod\_python会被分割成mod和python这样2个词。原本的香农信息值很高的专用名词（例如mod\_python 和django.contrib.admin）在进⾏拆分后可能得到了不相关的且香农信息很低的普通英单词，原本香农信息会丢失，会丢失主题构成中重要的关键词，因此选择完整保留这种包，库，类，函数名不进行拆分。因此在进行分词处理的时候选择了regexp模块中的regexp\_tokenize函数，用正则表达式去定义一些常用样式，如果文本中的词语符合定义的样式，即使中间有标点也不会被拆分。为了让提取包，库，类，函数名更加的精确，区分用点连接的词和句号两端的词，例如django.contrib.admin和python.The，需要自行写函数进行更细致的处理。

文本经过简单的而分词处理后，还会包含大量的无实际意义的通用词，需要过滤掉，NLTK提供了一份英文停用词词典供使用者直接使用。需要注意的是，停用词典中未涵盖的大写停用词和标点，这就要求在去除停用词前进行小写化处理，后续再过滤掉多余的标点符号。

### 2.2.4词性标注、词干化处理和词性还原

NLTK的tag包除了定义了一些词性标注的类外，还提供了部分词性标注的接口。它定义的几个词性标注器均以分词结果列表作为输入，对应返回每一个分词结果的词性。多数标注器都是根据训练语料构建的，比如一元语法模型（unigram ）词性标注器，对于给出的词汇，该标注器会在训练语料中查找每个词汇出现最多的词性并对其进行相应的标注，对于训练集中不存在的词汇，其词性会被标注为“None”。需要注意的是，词语的词性并不是一成不变的，英语中很多词往往具有多个词性，例如use，record等，必须根据它所在的句子去判断。

词干化处理(Stemming)就是去除形态词缀得到对应词根的过程，是英文特有的处理过程，比如说同一个英文单词有单数复数的变形（如apple和apples）、ing和ed等时态的变形（doing和did）、人称代词不同谓语的变形等（like和likes），这些词虽然形式上有细微差别，但是都对应着相同的词根，在某些情况下应该当作相同的词处理（比如计算相关性），这就需要进行词干化处理。词干化处理有三大主流算法：Porter Stemming、Lovins stemmer和Lancaster Stemming。

NLTK的stem包提供了几个相关模块进行词干化处理，包括 Lancaster Stemmer, Porter Stemmer，Snowball Stemmer。

词性还原（Lemmatization）是把一个任何形式的语言词汇还原为一般形式，但是能表达完整语义。比如名词有单复数变形，动词有不同时态，通过词性还原apples可以转换为apple，likes转换为like。相对而言，词干提取是简单的轻量级的词形归并方式，最后获得的结果为词干，并不一定具有实际意义。词形还原处理相对复杂，获得结果为词的原形，能够承载一定意义，与词干提取相比，更具有研究和应用价值。

在做预处理的过程中，首先尝试了词干化处理。考虑到最后从用户输入文本提取的关键词需要推荐给用户，如果使用词干化处理，力度太大，最后得到的词往往失去了原本的含义和价值，所以选择使用词性还原，可以使用NLTK的词性还原的函数lemmatizer.lemmatize(word, pos=' ')，传入2个参数，word是进行词性还原的词，pos是word的词性。NLTK里这个词形还原工具的一个问题是需要手动指定词性，比如"working" 这个词，如果不加后面那个pos参数，输出的结果将会是 "working" 本身。

本系统在预处理的时候，在词性还原之前筛选出了名词词性的词，所以pos只需要传入wordnet.NOUN即可。

### 2.2.5 文本数据结构化

（1） 词袋模型简介

“词袋模型”一词源自“Bag of words”，简称BOW，是构建文档-词项矩阵的基本思想。对于给定的文本（该文本可以是一个段落，也可以是一个文档），该模型会直接忽略文本的词汇顺序以及语法、句法，认为文本是由无序、独立的词汇构成的集合。这个集合可以被直观的想象成一个词袋，袋子里面就是构成文本的各种词汇。例如，文本内容为“python code analysis”的文档，用词袋模型可以表示为[python，code，analysis]3个独立的词汇。词袋模型对于词汇的独立性假设，可以简化文本数据结构化处理过程中的计算，因而被广泛采用，但是另一方面，这种假设忽略了词汇之间的顺序和依赖关系，降低了模型对文本的代表性。

（2） 文档-词项矩阵

“文档-词项矩阵”一词源自于“Document-Term Matrix”，简称DTM。如果将DTM矩阵转置后即为TDM。直观来看，DTM矩阵的行代表文档，列代表词汇，矩阵元素即为文档中某一词汇出现的次数。

例如，有以下两个文档：

document1为[python, code, analysis, study]，

document2为[data, secure, privacy, protect]，

基于这两个文档构造一个词典：

{1:“python”,2.“code”,3.“analysis”,4. “study”, 5. “data”, 6.“secure”,7.“privacy”，8.“protect”}，

这个词典一共包含8个不同的词汇，利用词典的索引号，上面两个文档都可以用一个8维的向量表示，具体如下：

document1为（1, 1, 1, 1, 0, 0, 0, 0），

document2为（0, 0, 0, 0, 1, 1, 1, 1），

向量元素表示对应维度的词汇在文档中出现的次数，比如document1的向量（1,1,1,1,0,0,0,0）表示在document1中，词典中序号为1、2、3、4的词都各出现了1次，序号为5、6、7、8的词没有出现。以此类推（0,0,0,0,1,1,1,1），表示在document2中，词典中序号为1、2、3、4的词都没有出现，序号为5、6、7、8的词各出现过1次。

两个向量合并在一起即得到文档-词项矩阵，如下：

[(1, 1, 1, 1, 0, 0, 0,0 )

(0, 0, 0, 0, 1, 1, 1, 1)]

一般地，文档-词项矩阵向量的长度是词典的长度，每篇文档的向量长度都一致，所以是n\*m的一个矩阵。n表示文档的数量，m表示词典的长度，矩阵中的每一个值表示在该文档中该词出现的频数。

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

（3）向量空间模型

向量空间模型把对文本内容的处理简化为向量空间中的向量运算，并且用数学概念上空间上的相似度来表达语义的相似度，非常直观形象。当文档被表示为文档空间的向量，就可以通过计算向量之间的相似性来度量文档间的相似性。文本处理中最常用的相似性度量方式是余弦距离。基本概念如下：

文档（Document）：一般指一篇文章。

项（Term）：文本的内容特征常常用它所含有的基本语言单位(字、词、词组或短语等)来表示,这些基本的语言单位被统称为文本的项。

文本可以用项集(Term List)表示, 其中Tn代表项

项的权重（TermWeight）：对于含有n个项的文本D,项常常被赋予一定的权重表示他们在文本中的重要程度，可以使用TF-IDF来赋予每个项权重。这样，文本就可以表示成如下, 其中Wn代表第n项的权重。TF-IDF将会在后面进行详细介绍。

如果维数很多，后面进一步计算会很困难，这时我们需要文本向量进行降维处理，所谓的降维就是选出有代表性的特征词，剔除一些无用的词语。

一般的，如果有n篇文档，⼀共包括m个特征词，那就可以构成m\*n维的稀疏矩阵。横轴表示第m个特征词在所有n个⽂档中的权值，纵向表示每一个文本向量，若某个词不存在⽤0表示权值。

基于TF-IDF，可以得到权重矩阵类似如下：

表2.1 权重矩阵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | D1 | D2 | D3 | D4 |
| T1 | 5.25 | 3.08 | 0 | 0 |
| T2 | 1.25 | 0.60 | 1.02 | 0 |
| T3 | 0 | 1.78 | 2.50 | 4.38 |
| T4 | 2.15 | 3.16 | 0 | 0 |

相似度(Similarity)：两个文本D1和D2之文本内容相关程度。当文本被表示为向量空间模型时,我们可以利用向量之间的距离来表示文本间的相似度，一般可以用夹角的余弦值表示。

计算特征空间中查询向量q和⽂档向量d两个向量夹角。若夹角越⼩，说明这两个特征向量描述的内容越相似，也就越符合查询标准。如果为1，则表示这两个向量描述的内容相同。如果为0，则表示这两个向量描述的内容完全不同。运算公式如下：

对向量空间模型来说，有个重要的问题就是如何对项的权重进行计算从而建立矩阵，常用的特征加权方法有布尔权重，词频权重，TF\*IDF，熵权重等

布尔权重:矩阵行表示的是项，列表示的是文档。假如第个文档中出现了第个项，就将相应的位置记为1，否则为0。

词频(TF)权重:特征项在文本中出现的次数。

逆文档频率（IDF）： 衡量特征词在整个训练样本集中分布的情况。DF指文档频率，如果包含某个关键字的⽂档数量越多，DF越大表示这个词越不重要，反之越小则权值越高。为了量化这一反比的特征，使⽤IDF来表示。公式如下 ,N表示训练样本总数，nk 表示训练样本集中出现特征项的文本数。为了避免分母为0，所以在分母上加1.

TF\*IDF权重：它⽤词频和逆文档概率的乘积来表示，第个单词在第篇⽂档中的权重可以表示为

（4）词频-逆向文档频率

在大量的文本数据中，通常会存在一些出现频率极高但是并无实际意义的词汇，如部分停用词，，或者是一些与主题关系没有很密切的词语。如果直接用这些所谓的高频词对文档进行进一步的分析处理很可能会忽略某些出现频率没有那么高的重要词汇，所以需要在词频的基础上对各个词汇的频数进一步调整。

“词频-逆向文档频率”就是一种常用的调整方式。“词频-逆向文档频率”一词源自“Term Frequency–Inverse Document Frequency”，简称TF-IDF。TF-IDF算法建立在以下假设之上：

对某个文档最有代表性的词汇或者说对区别文档最有意义的词汇应该是那些在某个文档中出现频率高，而在整个文档集合的其他文档中出现频率少的词汇。

直观来看，词频-逆向文档频率就是对文档词频矩阵的核心信息进行提取的结果，其目的在于使提取的词汇能突出所属文档的个性化主题，而不是文档集合中文档的共性。常用的计算方式就是在文档-词项矩阵（TF）的基础上进行权数（IDF）调整。

TF\*IDF权重：它⽤词频和逆⽂档概率的乘积来表示，第i个单词在第j篇⽂档中的权重可以表示为

## 2.3 主题模型

不论DTM还是TF-IDF矩阵都采用词频作为文档的特征表示，但是一词多义或多词一义的情况并不能通过词频反映出来，这些模型忽略了文本中的语意信息，导致进一步文本分析的精度较低。为了更好的自动生成关键词，本设计引入了主题模型去解决这个问题。主题模型可以将文档映射到指定维度的语义空间，利用各个文档在语意空间的分布可以计算文档之间的相似度。根据模型中已有的数据找出与用户输入相似度最高的一些文本，把这些文本已有的tag推荐给用户。

LDA隐含狄利克雷分布（Latent Dirichlet Allocation），最先由Blei,David M.等人于2003年提出，是一种以词、主题和文档三层贝叶斯概率为核心结构的主题模型。同时，LDA也是一种无监督机器学习算法，在LDA模型训练前不需要进行任何手工标注。LDA在文本挖掘领域包括文本主题识别、文本分类以及文本相似度计算方面都有广泛应用。

主题模型可以在没有任何监督的情况下从大型语料库中学习相关词汇。基于文档中使用的单词，他们通过假设单个文档包含一小组简洁的主题来挖掘主题级的关系。并且这些话题往往与人类概念相关联。例如，一个模型可能会产生涵盖政治，体育和电影等主题的主题。利用这些无监督方法，我们可以在提取有用的语义信息。

在主题模型中，“主题”一词是一个相对抽象的概念，通常表现为一系列相关的词，以及这些词出现的条件概率。LDA的核心目标是展示，在语料集中一共出现了哪些主题，以及这些主题中每个词出现的概率是多少。

在LDA中，模型将文档生成的过程理解为先选定一个主题向量，并确定每个主题被选择的概率，然后在生成文档中每个单词的时候，从主题分布向量中选择一个主题z，按主题z的单词概率分布生成一个单词。对于一篇包含N个词汇的文档，LDA模型的联合概率可表示为：

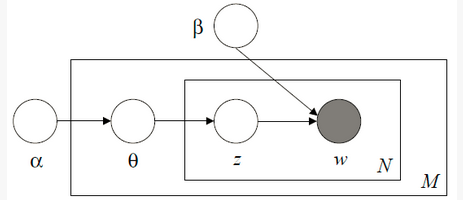


图2.1 LDA模型图

其中，为文档被选定主题向量的概率，服从Dirichlet分布；为从主题向量中选择主题的概率，服从多项式分布；为从主题中选择词汇的概率，服从Dirichlet分布；和则分别表示LDA中两个Dirichlet分布的分布参数，这两个参数属于语料级别，不会由于文档的不同而改变。LDA模型的生成过程也是对Dirichlet分布参数和的估计拟合过程，在实际应用中，计算机程序通常将通过EM算法来完成对与的参数值的估计，即在算法的E-step输入待定参数和的值，从而计算似然函数；之后在M-step最大化似然函数，解出对应的和，不断迭代直至收敛。

LDA 模型在解决文本数据分析问题时的优势表现在其合理的概率层次以及相对直观的结果表现，一般情况下，我们能够直接从主题中包含的词直接观察得主题的含义，从而更容易地找到隐藏在文本之中的语义关联。在本系统中，通过主题模型可以计算模型中已有的文本数据与用户输入文本的相似度

## 2.4 文本数据可视化

文本可视化技术综合了文本数据分析、数据挖掘、数据可视化等学科的理论和方法，是理解复杂的文本内容、结构和内在的规律等信息的有效手段。文本数据分析是文本数据可视化的基础，通过文本数据分析将文本数据结构化处理并提取出关键信息后才能进一步进行灵活、直观的可视化操作。

对于不同的文本数据分析结果，可以有不同的可视化方式，比如对分词的结果进行词频统计后可以绘制词频相关的柱状图或者词云图、对文本聚类或者分类的结果可以绘制矩形树图、对于各种文本数据分析结果还可以结合时间信息判断其变动趋势。

词云图可以将关键词按照一定的顺序和规则排列，以文字的大小代表词语的重要性，直观、快速地展示重要文本信息。为了检验直观检验模型的效果，对于模型自动生成数据集的所有关键词绘制成词云图。通过词语图，可以直观的判断关键词内容是否是合理的，从而修改模型的预处理、参数和流程等。

“wordcloud”是较为常用的Python词云绘制包，利用其定义的“WordCloud”类即可实现词云图的绘制，此外还可以通过“ImageColorGenerator”类使用指定的图片定义词云图形状和颜色。

在主题模型调参的过程中，为了确定参数主题个数，选用50到400个主题，间隔为25个主题，对训练集进行训练，训练完成后获取模型的coherence数值（数值越大，模型效果越好）。以coherence数值为纵轴，主题个数为横轴绘制折线图，观察coherence数值的变动趋势，确定最终的主题个数。

## 2.5 Beautiful Soup解析HTML

Beautiful Soup 是一个可以从HTML或XML文件中提取数据的Python库。为了获取代码运用的python和Django标准库的函数，需要爬取python和Django的官方文档，并使用Beautiful Soup进行解析，得到标准库的包、类、函数以及它们的对应关系。

在对数据集里的数据进行标准化的时候，使用Beautiful Soupk可以去除html标签，提取代码、链接和纯文本。

## 2.6 开发技术

### 2.6.1 Django

Python不仅对科学计算的支持度较好，还有很多开源的机器学习的库，方便模型的建立和训练。python下有许多款不同的Web框架。Django是最有代表性的一个。许多成功的网站和APP都基于Django。Django采用了MVC的软件设计模式，可以用它快速搭建网站。

### 2.6.2 React

前端采用Webpack打包技术，使用React框架和Antd组件库搭建。React是由Facebook开发的开源前端MVVM框架。React开创性的提出了虚拟DOM，以及虚拟DOM的diff算法。使用虚拟DOM的diff算法，可以找到从当前DOM树更新为新的DOM树的最小代价的更新方法。通过虚拟DOM技术，实现了当保存data的State和Props变化时，页面会自动进行更新。通过这种方式，使得JavaScript中保存的data和显示的页面很好的解耦，开发人员只需要专注于JavaScript中有关状态（Props和State）的业务逻辑的编写，无需重复考虑如何更新页面。

## 2.7 开发环境与参数

表2.2 环境详情

|  |  |  |  |
| --- | --- | --- | --- |
| 环境 | 类型 | 版本 | 说明 |
| Pycharm | 开发环境编辑器 | V2018.1.1 | Python编辑器 |
| ROBO 3T | 开发工具 | V1.1 | Mongodb可视化工具 |
| Postman | 接口测试工具 | V5.5.2 | 后端接口测试 |
| MongoDb |  | V3.4.10 |  |

表2.3 主要依赖包参数

|  |  |  |  |
| --- | --- | --- | --- |
| 包/框架 | 实现语言 | 版本 | 说明 |
| nltk | python |  | 自然语言处理工具包 |
| gensim | python |  | 主题模型实现 |
| Django | python | 1.8 | Python框架 |
| React | JavaScript | 1.60 | 前端框架 |
| WordCloud | python |  | 词云 |

# 3 系统设计

本章将具体分析代码标签自动生成系统的用例场景、使用流程和后台自动化分析流程。明确定义系统用例图、系统功能点和用户的参与流程。提供系统概要设计，明确非关系型数据库MongoDb对字段的定义，方便对数据集的操作。架构设计明确各个模块功能，提取公共基础模块，降低系统耦合度。详细设计会阐述分析重点功能模块，以部分规约洞见整个系统的程序设计思路。

## 3.1 Use Case

代码标签自动生成系统主要有用户代码问答网站的提问者和回答者，数据标准化模块，预处理模块，文本分析模块，代码分析模块。数据标准化模块负责将训练数据以及提问者和回答者输入问题或者回答（包括代码和文本）进行标准化处理，方便后续的分析；预处理模块负责分词，去除停用词，词性还原，筛选某种词性的词, 对代码提取注释和引用的库和纯代码；文本分析模块使用训练数据创建模型，对用户的输入经过预处理可以导入模型进行关键词的提取；代码分析模块对纯代码进行解析得到代码引用的包、类、函数。

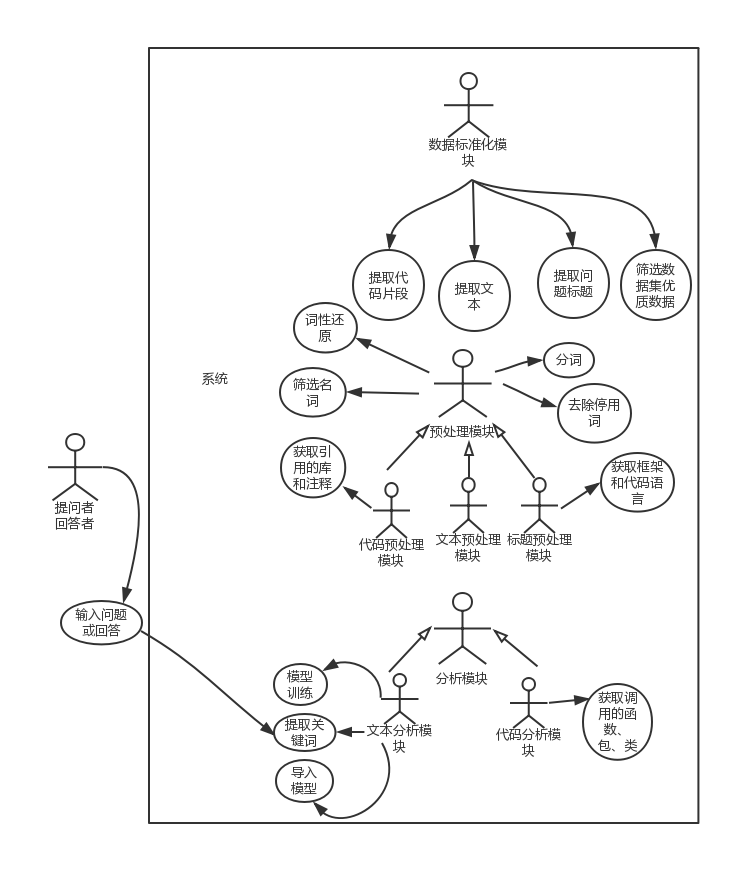


图3.1 系统整体用例图

## 3.2 概要设计

从软件工程的角度，本节介绍数据库的选择和设计以及系统流程图。

### 3.2.1 数据库选择和设计

原始数据集是Stack Overflow网站上和python相关的问题，回答，以及标签，将它们分别存到非关系型数据库MongoDb中。它们原始的数据结构如下

表3.1 Stack Overflow数据库表answers字段

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| \_id | ObjectID | 每一条记录的唯一标识 |
| Id | String | 回答的编号 |
| CreationDate | String | 回答的创建时间 |
| Score | String | 答案评分 |
| Body | String | 回答内容 |
| ParentId | String | 对应问题的编号 |
| OwnerUserId | String | 回答者的编号 |

表3.2 Stack Overflow数据库表questions字段

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| \_id | ObjectID | 每一条记录的唯一标识 |
| Id | String | 问题的编号 |
| Title | String | 问题的标题 |
| CreationDate | String | 问题的创建时间 |
| ClosedDate | String | 问题的关闭时间 |
| Body | String | 问题的内容 |
| OwnerUserId | String | 提问者的编号 |
| Score | String | 问题评分 |

表3.3 Stack Overflow数据库表tag字段

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| \_id | ObjectID | 每一条记录的唯一标识 |
| Id | String | 标签的编号 |
| Tag | String | 标签内容 |

对数据标准化的时候，需要将这三个表整合，方便后面进一步分析。在整合的时候，是以每条回答为中心的，因为每个问题可能有不止一条回答，所以要根据回答的parentId去找到对应的问题以及tag序列。这里需要注意的是，tag在原始数据集中的存储方式并不是以对应每个问题的list的形式存储，而是一个tag对应一个问题id，所以需要根据回答的parentId（相当于问题id）去获取所有对应的tag。

针对回答内容，考虑到代码片段和链接比较多，所以把代码、链接、纯文本提取出来作为新建的字段,所以新建了一个标准化的数据集，每个数据元主要包括问题编号，问题标题，问题内容，问题标签，问题创建者编号，问题创建时间，问题回答时间，问题赞数，回答者编号，回答内容，回答赞数，回答编号，回答代码片段，回答链接，回答纯文本。数据结构如下标所示：

表3.4标准化后表字段

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 说明 |
| \_id | ObjectID | 每一条记录的唯一标识 |
| answer\_Id | String | 回答的编号 |
| answer\_Owner | String | 回答的作者 |
| answer\_CreationDate | String | 回答的创建时间 |
| answer\_Score | String | 回答的分数 |
| answer\_Body | String | 回答的内容 |
| question\_Id | String | 问题的编号 |
| question\_OwnerUserId | String | 问题的作者 |
| question\_CreationDate | String | 问题的创建时间 |
| question\_Score | String | 问题的分数 |
| question\_Title | String | 问题的标题 |
| \_ question\_Body | ObjectID | 问题的内容 |
| Tag | String | 问题自带标签 |
| answer\_Code | String | 回答的代码片段 |
| answer\_Link | String | 回答的链接 |
| answer\_Text | String | 回答的纯文本 |

### 3.2.2 系统工作流程

本系统设计系统流程如下。从数据源采集入手，原始数据集为StackOverFlow的问答数据集，包括问题，回答，标签。对这些数据进行数据处理，整合3份原始数据集之后筛选优质数据并进行标准化后将数据存储到数据库中Mongodb。后端预处理模块会对问题、代码、纯文本分别进行分词、去除停用词、词性还原等不同的预处理，同时生成语料库。分析模块利用语料库生成词典和向量空间，然后创建tf-idf模型和主题模型并保存下来方便加载。

每当用户提交一个请求，即提问者和回答者在浏览器中输入问题或者回答（包括代码和文本）点击提交并自动生成标签，服务器后端解析请求以后，调用数据标准化模块将输入标准之后分成文本和代码，再调用预处理模块对它们进行不同的预处理，其中将文本内容最终生成词典、向量，对代码片段提取注释和引入的库，对问题标题分词、提取框架和代码语言。最后词典和向量导入分析模块建立的模型进行关键词的提取，代码片段则根据官方标准库和标题提取的框架分析它们调用的函数、类、包，并作为关键词。关键词合并之后通过后端程序以json形式传给前端。

下图是系统各个模块运行的整个工作流程的展示。

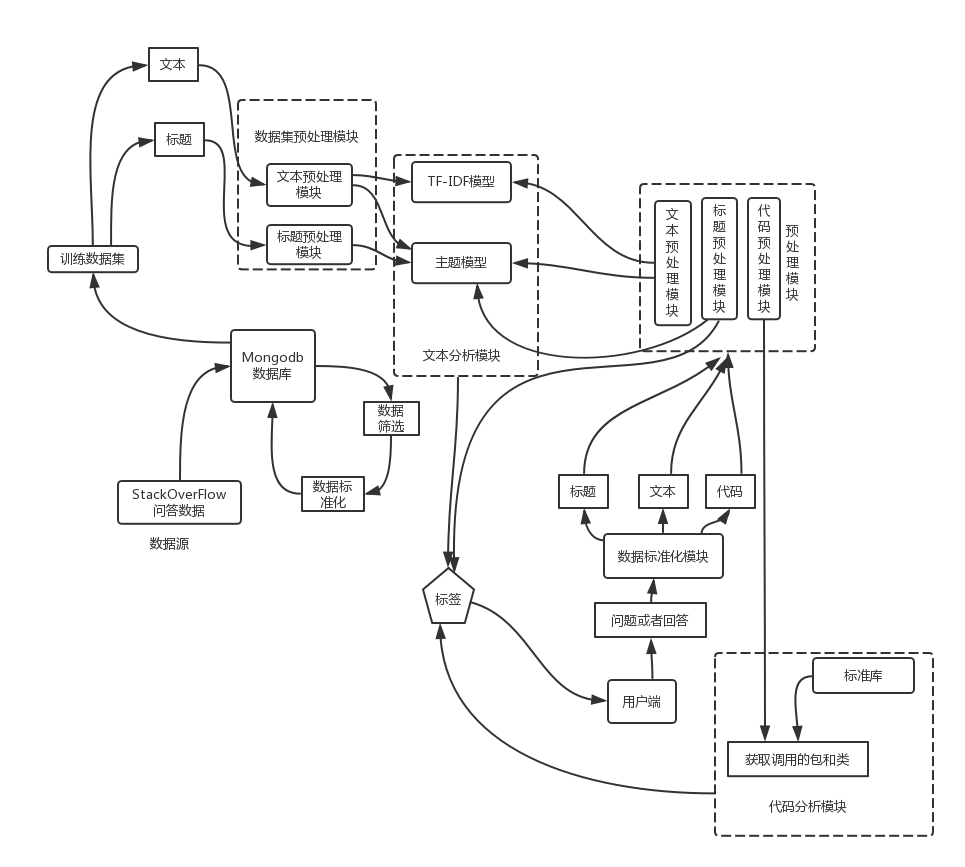


图3.2 系统运行流程

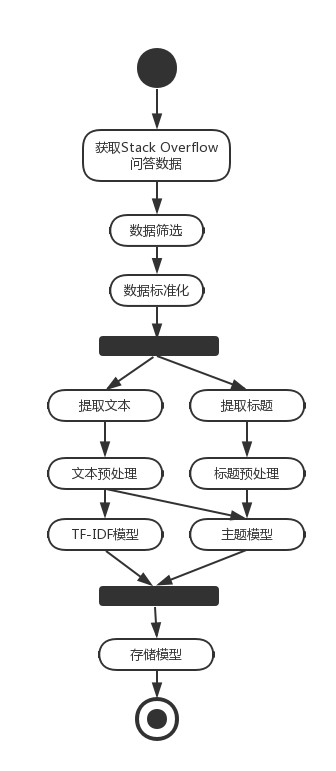


图3.3 系统模型层活动图

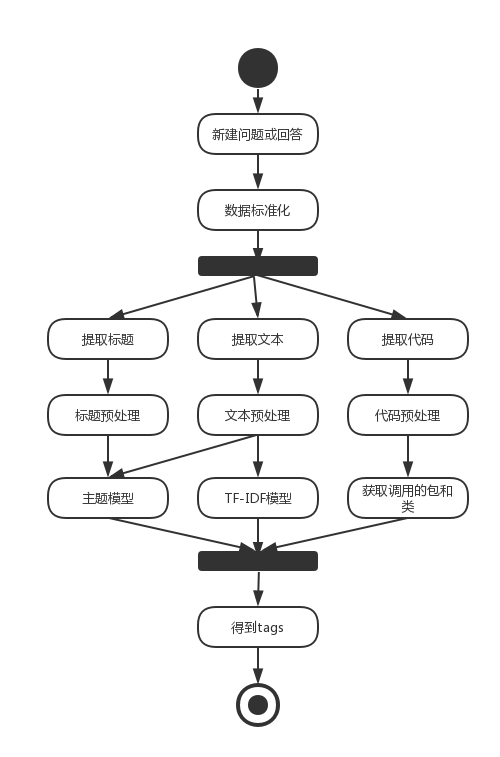


图3.4标签自动生成活动图

## 3.3 系统架构设计

如图，系统分为四层，分别为数据层，服务层，后端，前端。

数据层是系统的最底层，它包括了从用户输入的问题和回答经过数据标准化，再通过预处理模块的分词、停用词处理、词性还原等。

服务层包括TF-IDF模型和主题模型构建，利用数据库中的训练数据集生成模型、词典，导出到服务器文件系统，供后端调用。

后端使用了python的框架Django，可以调用数据层的数据标准化模块、预处理模块，文本分析模块（包括服务层的模型）和代码分析模块，最后生成标签。

前端使用react框架，处理用户输入，展示自动生成的标签，主要包括主页、回答详情页、和标签生成页。

## 3.4 设计细节

### 3.4.1 前端设计

前端设计针对问答网站这个场景，着重展示根据代码片段及相关文本生成标签的流程，所以功能比较简单。包括以下几个界面

1. /questions 问题列表页面：

展示所有存在问题的列表，点击某个问题可以进入该问题的详情页面。

1. /questions/:id 问题详情页面：

问题详情页展示了问题标题和内容，以及一个可以输入回答的文本框，输入回答提交后可以看到生成的标签。

1. /questions/new 创建问题页面：

创建问题页面可以输入问题标题和回答，提交后可以看到生成的标签。

### 3.4.2 接口设计

1. /api/questions

简介：获得所有的问题列表

Method：GET

Response Body

{

"questions": [

{

id: number,

title: string,

body: string

},

{

id: number,

title: string,

body: string

}

]

}

2. /api/questions/:id

简介：根据ID获取对应问题的详情

Method：GET

Response Body

{

id: number,

title: string,

body: string

}

3. /api/tags

简介：根据title和body生成tags

Method：POST

Request Body

{

body: string,

title: string

}

Response Body

{

code: string[],

text: string[],

intersection: string[]

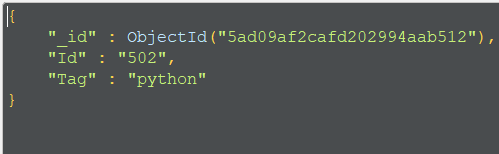
}

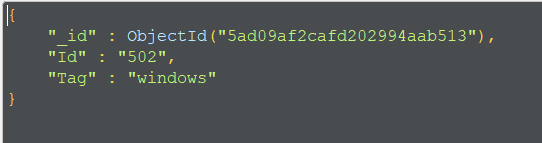
# 4 标准化和预处理

## 4.1 数据标准化

### 4.1.1 合并数据集

原始数据集是StackOverFlow的开源python数据集，包括questions，answers,tags，将它们存储在非关系型数据库MongoDb中。为了方便进一步的分析，需要将回答以及对应的问题和问题对应的标签，进行合并，并存储在新的数据集python\_stadard中,总数据量共有26万条数据。需要注意的是，一个问题可能对应多个标签，但是一个问题的多个标签并不会存储在一条数据中，而是分别存储，但它们对应的问题编号是相同的，需要遍历tag的表进行查询。





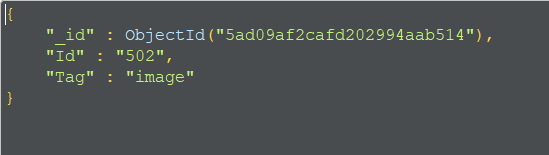


图4.1 问题“Id”为“502”的3个tag

新的数据集python\_stadard囊括了所有基本信息，包括问题编号，问题内容，问题标题，回答内容，标签等，每一条数据看做一个数据元。

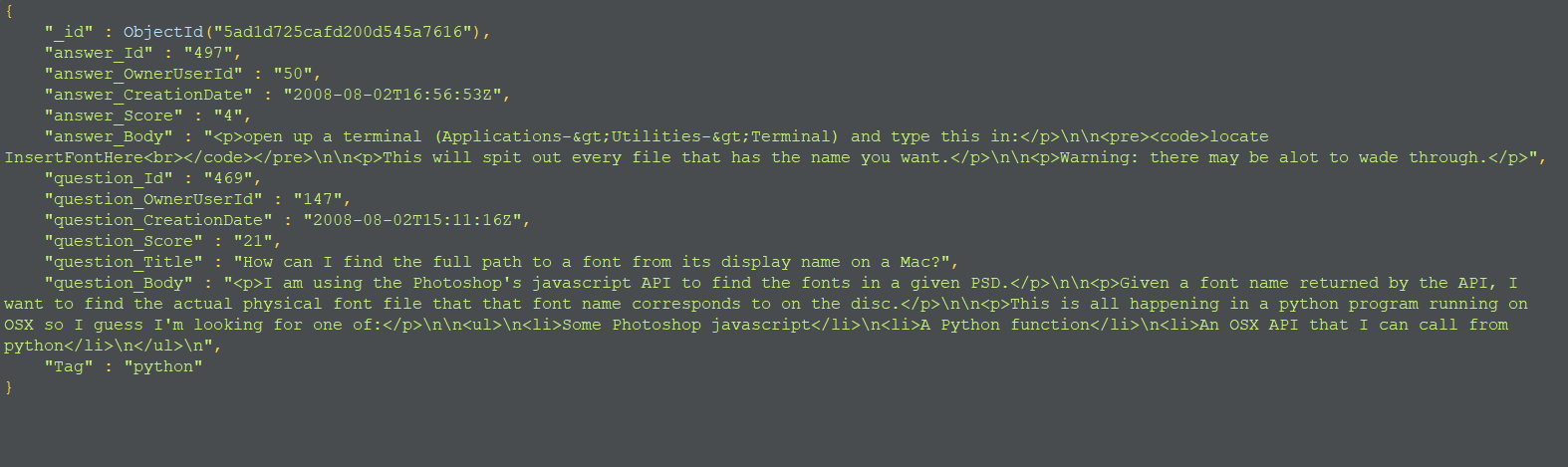


图4.2 “python\_stadard”数据集里的一条数据

合并数据集的部分代码片段如下：



图4.3 合并数据集的代码

### 4.1.2 提取代码片段和纯文本

为了对代码和纯文本进行不同的预处理，需要将它们从文本内容中提取出来。考虑到回答普遍存在代码片段，问题中代码片段较少，所以只提取了回答的代码片段。在文本内容中除去

代码片段和HTML标签即可提取纯文本。

代码使用BeautifulSoup 去根据html标签提取代码和纯文本。将一段文档传入BeautifulSoup 的构造方法,就能得到一个文档的对象，一般使用使用lxml作为解析器,因为效率更高。Beautiful Soup可以将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象,所有对象可以归纳为4种: Tag，NavigableString，BeautifulSoup，Comment。

Beautiful Soup可以通过使用CSS选择器的语法找到tag，以及通过tag标签逐层查找。在源数据中，代码片段的格式为：

<pre><code>[代码片段]< /code > </pre>

代码raw\_codes = soup.select('pre code')可以提取<pre><code>[代码片段]< /code > </pre>中的代码片段。

链接的一般形式为:

<a href="[链接地址] ">[链接内容] </a>

使用Beautiful Soup的 find\_all()方法可以返回文档中符合条件的所有tag构成的列表，用get方法可以提取tag的任意属性。代码raw\_links = soup.find\_all('a')，links = [raw\_link.get('href') for raw\_link in raw\_links]可以提取链接地址

为了获取纯文本，不仅要去除html标签，也要去除代码内容和链接内容，使用Beautiful Soup的extract()方法可以当前tag移除，再调用get\_text()方法获取到tag中包含的所有文本内容,其中包括子孙tag中的内容

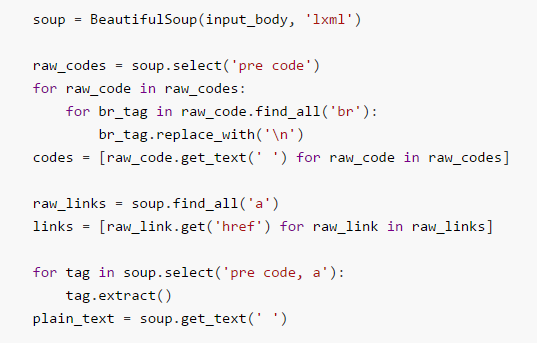


图4.4 提取代码片段和纯文本的代码

## 4.2 预处理

本系统的预处理分为2部分,分别是源数据的预处理和处理用户输入的预处理。源数据的预处理分成了问题标题的预处理和问题、回答纯文本的预处理。用户输入内容包括了代码和纯文本（如果是提出问题，则还包括问题标题），将它们按照上文的步骤提取出来；处理用户输入的预处理包括问题标题的预处理（如果用户输入是回答，则会获取对应问题的标题）和问题、回答纯文本的预处理和代码部分预处理。其中源数据的预处理和处理用户输入的预处理在问题标题、文本部分是一致的。

文本预处理的基本流程为对每篇文本进行[分词token化] ----> [字母转换成小写] ----> [词性标注] ---> [去除停用词和英文标点] ----> [词性还原] ----> [得到该篇文本的词语列表]。根据处理对象的不同和处理效果适当调整预处理的步骤。

### 4.2.1 问题标题预处理

问题标题虽然一般情况下比较简短，长度大概是一句话，可以看做是对问题内容的高度浓缩。由于Stack OverFfow的提问机制非常严格，所以它的包含的代码信息很有价值。问题标题预处理流程为2个部分：

（1）提取框架信息和代码语言

Django是Python的一个常用框架，它在python的基础上，又添加了很多自带的标准库方便开发者。一方面，在代码分析的时候为了更好的提取代码片段调用了标准库中哪些包和类中的函数，需要区分这些代码是否使用Django框架。另一方面，作为python中最重要的框架，在配置Django，使用Django的类、包开发，以及最后项目发布，开发者会遇到很多和Django框架直接相关的问题。所以Django可以算是Python中的重要分支，也是一个很重要的标签。一般的，如果是涉及Django有关的python问题，提问者会在问题中注明,只要判断一下问题中是否包含“Django”（“Django”）一词即可。

从词云可以看出，根据问题自带的标签，Python的问题还常常涉及和其他的代码语言的类比，或者是和前端语言，数据库的结合，这些和代码片段直接相关，也应当作为重要的标签予以推荐。所以定义了一个包含常用代码语言的list：

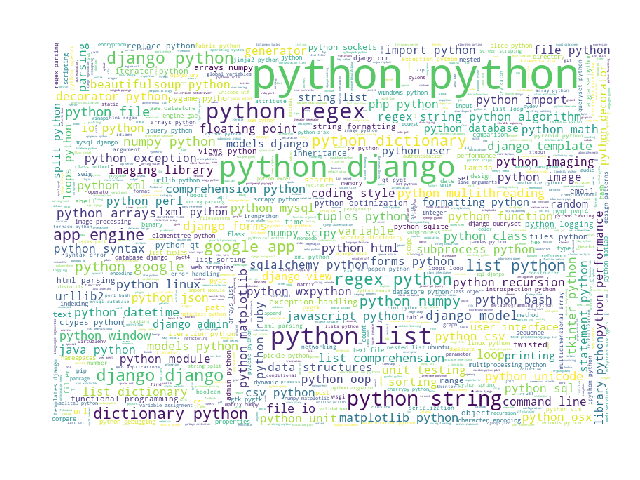


图4.5 词云



图4.6 包含常用代码语言的list

在问题标题分析的时候，判断一下标题是否包含以上这些词，如果有，则作为代码语言关键词存储。

（2）提取关键词

在完成（1）的提取过程之后，对于标题进行常规预处理如下：

[分词token化] ----> [字母转换成小写] ----> [去除动词和数词] ----> [去除停用词和英文标点] ----> [得到词语列表]

其中分词token化调用了nltk中tokenize分词包的regexp模块中的函数regexp\_tokenize(title, pattern)，对于分词的规则采用了用正则表达式自定义。普通的分词函数会按照空格、标点分词，但是python代码或者相关的包、类、函数中有很多词语是类似mod\_python，django.core这样用.或者下划线链接的词，如果按照标点分割会把类似mod\_python这样的词分成mod和python，这显然是不合适的。在参考nltk分词regexp模块的官方教程有关编写正则条件的基础上（一般是英文文本常用的连接词），添加了在python代码中常用的连接词，符合正则表达的词语中间即使出现标点也是不能分割的。

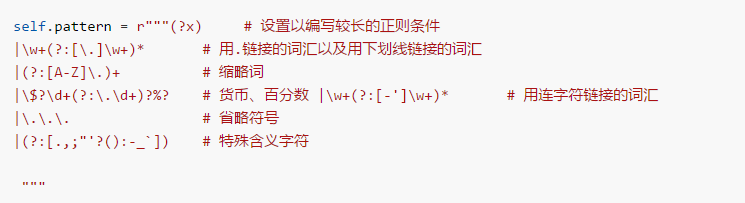


图4.7 用于分词规则的正则表达式

同时，需要注意的是对于用.链接出来的词，需要编写函数判断这个.是不是英文中的句号，例如django.core.The中，前一个.不需要分割，但是后一个.需要分割，可以根据英文书写的一般规则句子开头首字母是大写来判断。在django.core.The中，T是大写，c是小写，所以第一个点不需要分割，而第二个点需要分割。

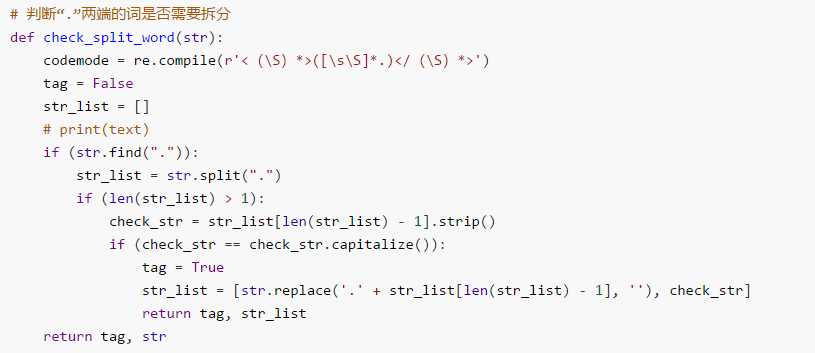
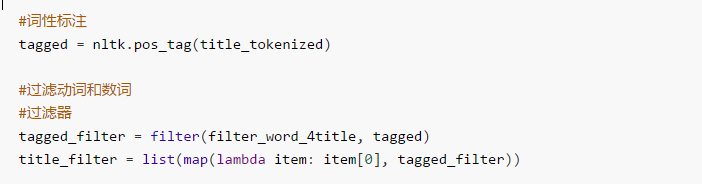


图4.8 判断.两端的词是否需要拆分

去除动词和数词首先需要使用nltk自然语言包的词性标注去标注语句中所有词语的词性，然后自己编写函数过滤动词和数词。这样做的原因是动词和数词一般都和问题核心关联较小。例如有标题“Python Regular Expressions to implement string unescaping”，我们希望获取的标签是类似“python”,“expression”,“string unescaping”这样名词词性的词语，而不是“implement”这样的动词。



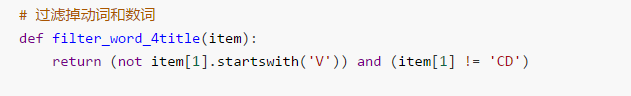


图4.9 词性标注和过滤

### 4.2.2 文本预处理

文本预处理指的是对纯文本的预处理，包括了问题纯文本和回答纯文本，基本流程如下：

[分词token化]----> [字母转换成小写]----> [去除停用词和英文标点]----> [词干提取] ----> [得到该篇文本的词语列表]。

分词token化和上文阐述的一致，在这里就不赘述了。

词干提取（stemming）是预处理常用的步骤，抽取词的词干或词根形式，但是提取之后一般不能够表达完整语义。例如 “leaves”进行词干提取之后得到词干“leav”，实际上它应该是名词“leaf”的复数形式。



图4.10 使用nltk进行词干提取

由于提取的关键词将会呈现给用户，如果使用词干提取，最后推荐的关键词也是词语的词干形式，不能很好表达完整语义，所以词干提取并不是很适合这一场景。英文单词普遍存在单复数或者是各个时态，如果不进行适当处理，类似于“do”和“done”会被当做不同的单词，在这种情况，选择了词性还原去解决这个问题。

词形还原（lemmatization），是把一个词汇还原为一般形式，一般能表达完整语义，例如 “leaves”进行词性还原之后可以得到单数形式“leaf”。nltk中使用词性还原要求手动注明词性，由于一个单词可能有多个词性，否则可能会有问题。因此一般先要分词、词性标注，再词性还原。

除了词干提取力度太大影响预处理效果，在测试了3万条数据自动生成标签，将这些标签汇聚在词云上后发现，仍旧有很多关键词并不是很理想。类似于many，this，except，exclude

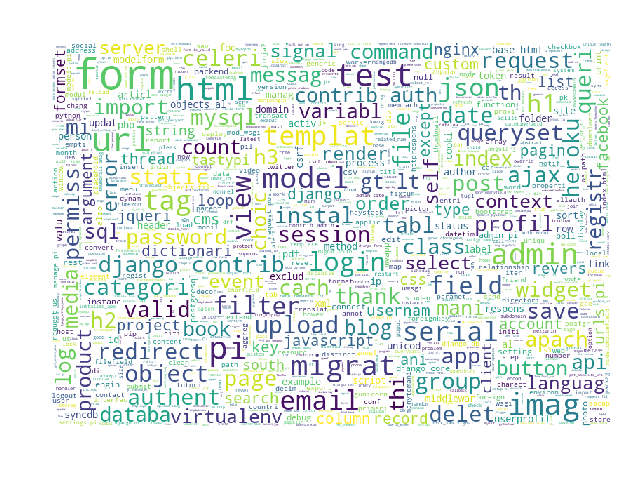


图4.11 改进前词云

这样的词缺乏实际意义，select, test, login有一定代码意义,但是作为关键词仍缺乏一定的概括性。同时，类似json，jquery，django, image, server,list,template,session可以看做是理想的关键词。可以发现，比较符合我们思维习惯，并且具有一定概括性的，代码相关程度高的关键词基本是名词。为了过滤到劣质的关键词，所以改进了处理流程，在除去停用词之前增加了词性标注和筛选出名词词性的词语。如下图，是改进后的词云，可以发现关键词的质量有了一定程度的提高

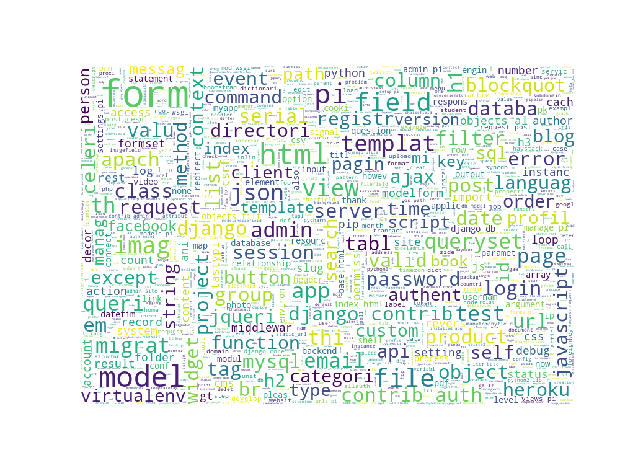


图4.12 改进后词云

最终确定文本预处理流程为[分词token化]----> [字母转换成小写]----> [词性标注]----> [筛选出名词词性的词语]----> [去除停用词和英文标点]----> [词性还原] ----> [得到该篇文本的词语列表]。处理后得到的是一个list,每个元素为该篇文章的词语列表。对语料库中所有文本完成预处理后，将所有文章的list存储在一个list中，可以进一步生成词典或者转化成其他类型的矩阵，再导入到TF-IDF和主题模型中

最终的大list类似如下：

[

[app, model, mysql…],

[tag, email, type…],

[json, python-3,ajax…],

[django, pip, type…]

[pip, session, admin…],

…

]



图4.13 文本预处理代码

### 4.2.3 代码预处理

代码预处理和文本采用了不同的处理流程，由于代码语句的内容是不能作为关键词推荐给用户的，所以直接分词是不可取的，需要尽量挖掘代码的结构化信息。问答中出现的代码都是以片段的形式呈现的，在结构和功能上都不够完整，甚至有些代码是不完全符合语法规则的。

如下图，是一个典型的例子。

问题是:“Adding a Method to an Existing Object Instance”(对一个已经存在的实例如何添加新的方法)

图4.13 一个典型的python问答例子

代码片段如下：

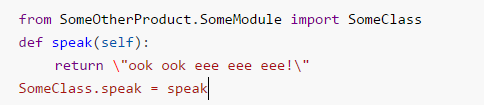


图4.14 回答的代码片段

这段代码在StackOverFlow回答的代码片段中非常具有典型意义，篇幅上短小精悍，主要针对问题的痛点进行解析示范，并且没有实际工程意义。从图4,12可以看到，它使用SomeOtherProduct.SomeModule 指代某个项目某个模块, SomeClass 指代模块中的某个类。这样不完整的代码片段大部分很难使用语法树去解析，即使部分问题对应的回答里的代码片段可以被成功解析，但受制于不是完整的代码文件，仍旧不能起到良好的分析效果。

一般的，开发者在问答网站上搜寻、浏览、回答某个问题，往往需要明确问题涉及代码语言里面哪一个的模块才能深入，比如python中有I/O读取，正则表达式，装饰器，迭代器，切片，词典，集合，框架Django等。如果可以更精确地给出问题或者回答中代码语言涉及的标准库API中哪些包、模块、函数，有利于提高开发者的学习效率。这些工作将在代码分析阶段完成。

承接这一思路，代码预处理首先提取import信息和注释信息，python中引入库、函数、包的一般形式有from a import b ,import c,注释信息一般形式有#[注释内容]，”””[注释内容]”””,均采用正则表达式提取。

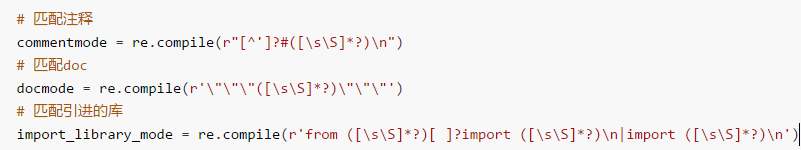


图4.15 正则表达式



图4.15 代码预处理代码截图

# 5 文本分析

## 5.1 TF-IDF模型

### 5.1.1 语料库与向量空间

语料（corpus）是一组原始文本的集合，用于无监督地训练文本主题的隐层结构,并且语料中不需要人工标注的附加信息。在python的Gensim包中，corpora是文档集的表现形式，也是后续进一步处理的基础。从本质上来说，corpora就是一个二维矩阵。向量（vector）是由一组文本特征构成的列表。词典（dictionary）是所有文档中所有单词的集合，而且记录了各词的出现次数等信息。

在预处理过程中将文档分割成词语之后生成词语列表，并生成词典：

dictionary = corpora.Dictionary(wordlist)

词典有很多功能，比如diction.token2id 存放的是（word，id）这样的键值对，diction.dfs 存放的是单词的出现频率。利用keys方法可以查看所有词汇id，利用values方法可以查看所有词汇。

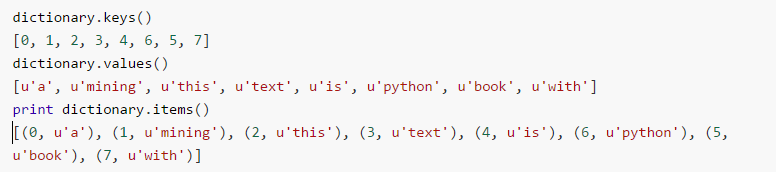


图5.1 词典的功能

利用doc2bow方法可以在构建的词典基础上得到文档词频矩阵，调用方式如下：

corpus = [dictionary.doc2bow(text) for text in texts]

函数doc2bow()对每个不同单词的出现次数进行了计数，并将单词转换为其编号，然后以稀疏向量的形式返回结果。因此，稀疏向量[(0, 1), (1, 1)]表示：在词典中，id为 0 和 id 为1的单词各出现一次，隐含意义为其他词典中的单词均没有出现过。

### 5.1.2 TF-IDF转化

我们已经创建了一个用向量流表示文档的语料库，接下来需要进行TF-IDF的转化。转换是标准的Python类，通常通过训练语料库的方式初始化：

tfidf = models.TfidfModel(corpus)

不同的转化需要不同的参数，在TF-IDF转化中，训练的过程就是简单的遍历训练语料库，然后计算文档中每个特征,也就是每个词的频率（在多少文档中出现过）。需要注意的是，转换常常是在两个特定的向量空间之间进行。训练与后续的转换必须使用相同的向量空间。例如使用不同的预处理方法处理字符串、使用不同的属性编号、需要TF-IDF向量的时候却输入了词袋向量，将会导致转换过程中属性匹配错误，进而使输出结果无意义并可能引发异常。

创建的tfidf将被视为只读的对象，可以用它来转换将任何采用旧表示方法的向量（比如词袋整数计数）转换为新的表示方法（TF-IDF权重），并且可以对整个语料库进行转换。

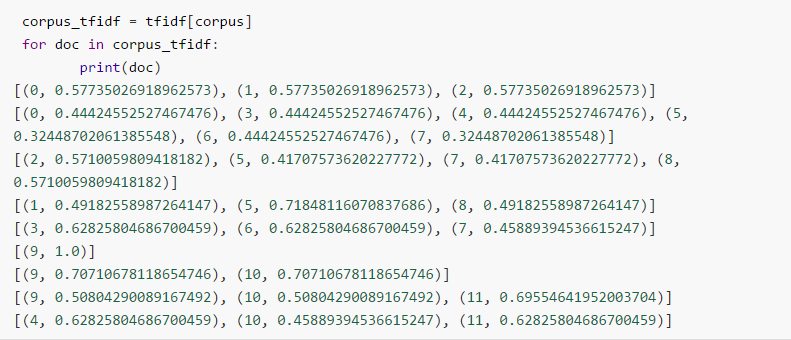


图5.2 使用TF-IDF对整个语料库进行转化

如上图，转换之后打印可以看到文档集合中每个文档中每个特征，也就是词语的TF-IDF权重。

在这个特殊的情况中，被转换的语料库与用来训练的语料库是相同的，但是这不是必须的。一旦转换模型被初始化了，它可以用来转换任何向量，当然最好是使用与训练语料库相同的向量空间。

TF-IDF数值与一个词在文档中的出现次数成正比，与该词在整个语言中的出现次数成反比。所以，自动提取关键词的算法就是计算出问答中所有文本集合中，每份文本的每个词的TF-IDF值，然后按降序排列，每份文本取排在前3的词作为关键词。

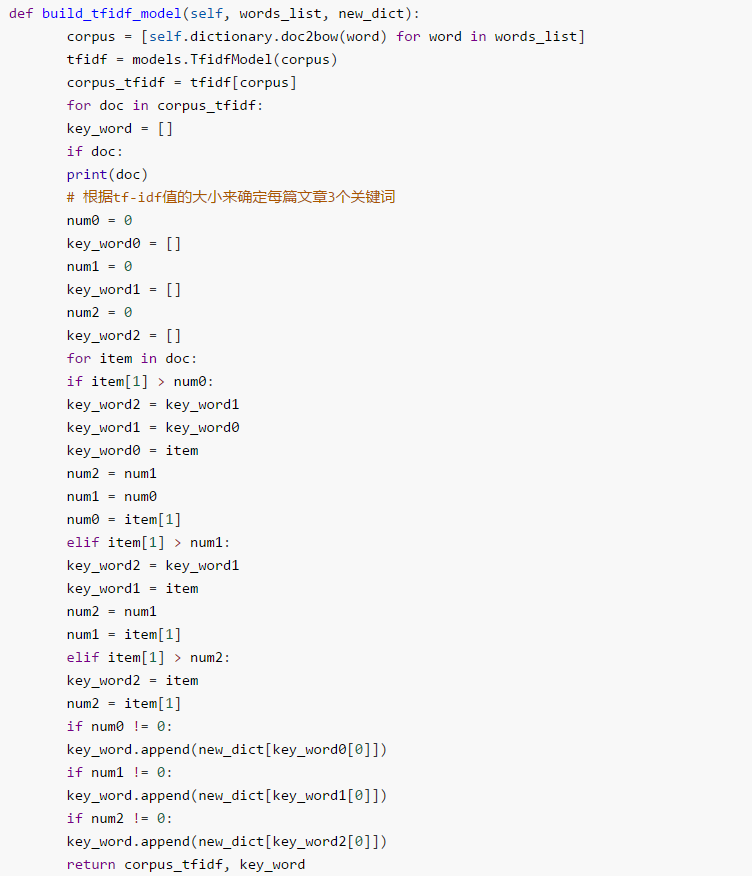


图5.3 创建一个TF-IDF模型并提取关键词

### 5.1.3 Gensim中可用的转化

Gensim实现了几种常见的向量空间模型算法，有词频-逆文档频（TF-IDF），潜在语义索引（LSI），隐含狄利克雷分配（LDA）以及分层狄利克雷过程（HDP）：

a. 词频-逆文档频（Term Frequency \* Inverse Document Frequency，TF-IDF）

需要一个词袋形式的训练语料库来实现初始化。转换过程中，它将会接收一个向量同时返回一个相同维度的向量，在语料库中非常稀有的属性的权重将会提高。因此，模型会将整数型的向量转化为实数型的向量，同时让维度不变。创建模型的函数和参数如下：

model = tfidfmodel.TfidfModel(bow\_corpus, normalize=True)

b. 潜在语义索引（Latent Semantic Indexing，LSI，or sometimes LSA）

将文档从词袋或TF-IDF权重空间更好地转化为一个低维的潜在空间。LSI训练的独特之处是我们仅需提供更多的训练文本，能在任何时候继续“训练”。这是通过对底层模型进行增量更新，这个过程被称为“在线训练”。正因为它的这个特性，输入文档流可以是无限大。创建模型的函数和参数如下：

model = lsimodel.LsiModel(tfidf\_corpus, id2word=dictionary, num\_topics=300)

c. 隐含狄利克雷分配（Latent Dirichlet Allocation, LDA）

将词袋计数转化为一个低维主题空间的转换。LDA是LSA的概率扩展，因此LDA的主题可以被解释为词语的概率分布。就像LSA一样，这些分布是从训练语料库中自动推断的。文档可以被解释为这些主题以及对应概率的一个混合。创建模型的函数和参数如下：

model = ldamodel.LdaModel(bow\_corpus, id2word=dictionary, num\_topics=100)

d. 分层狄利克雷过程（Hierarchical Dirichlet Process，HDP）

是一个无参数贝叶斯方法（注意：这里没有num\_topics参数），效果比较粗糙。创建模型的函数和参数如下：

model = hdpmodel.HdpModel(bow\_corpus, id2word=dictionary)

## 5.2 主题模型

### 5.2.1主题模型的创建

隐含狄利克雷分配（Latent Dirichlet Allocation，LDA），即主题模型的创建也和TF-IDF模型的创建是类似的，可以简单的归纳为以下的步骤：

[获取文档集合] ----> [清洗文档] ----> [建立文档术语矩阵] ----> [转换为LDA模型] ----> [检验]。

获取文档集合是从Mongodb数据库取得pythony有关问题和回答的纯文本，清洗文档是对纯文本进行一系列预处理，在这里就不再赘述了。

为了创建一个LDA model,我们首先需要知道每篇文本每个单词出现的次数，和TF-IDF模型类似，利用doc2bow方法可以在构建的词典基础上得到文档词频矩阵，调用方式如下：

corpus = [dictionary.doc2bow(text) for text in texts]

创建一个LDA model代码如下：

图5.4 主题模型构造函数和参数

部分参数简介

* corpus：用该参数传入的文档语料将会被用来训练模型；
* num\_topics：需要提取的潜在主题数；
* id2word：用于设置构建模型的词典，决定了词汇数量，id2word = dictionary.id2token；
* distributed：是否开启分布式计算；
* chunksize：文件块大小，等同深度学习训练的batch，一次性给入2000篇文章，一次性给入越多，性能越好，该指标会略微影响最终结果；
* passes：等同epoch，利用整个语料库训练模型的迭代数；
* iterations：控制每篇文章迭代数；
* alpha：决定文档主题狄利克雷先验分布的超参数，默认取值为对称 1.0/num\_topics 先验，可以自行设置，也支持以下两种取值：

（1） ‘asymmetric’ ：固定的非对称 1.0/topicno 先验

（2） ‘auto’：根据实际数据学习得到的非对称先验

* eta：决定主题词汇狄利克雷先验分布的超参数，可以自行设置为对称的先验分布常量或者长度为词汇总数的向量作为非对称先验，此外也支持以下两种取值：

（1） ‘auto’：根据实际数据学习得到的非对称先验

（2） 形如 num\_topics x num\_word的矩阵：为每一个主题都引入一个词汇非对称先验分布

* minimum\_probability：用于限制返回一个文档主题的概率，利用random\_state进行随机化设置。

除了iterations和num\_topics需要根据语料库的具体情况调整和修改，其他基本取默认值就可以了。在下一节会具体阐述如何调参和参数选择。

创建一个简单的主题模型代码如下：



图5.5 创建主题模型

选用了100份问答文本作为语料库，设置了num\_topics为3，num\_words为3，即总共用3个主题，每个主题由3个词语构成,输出结果如下

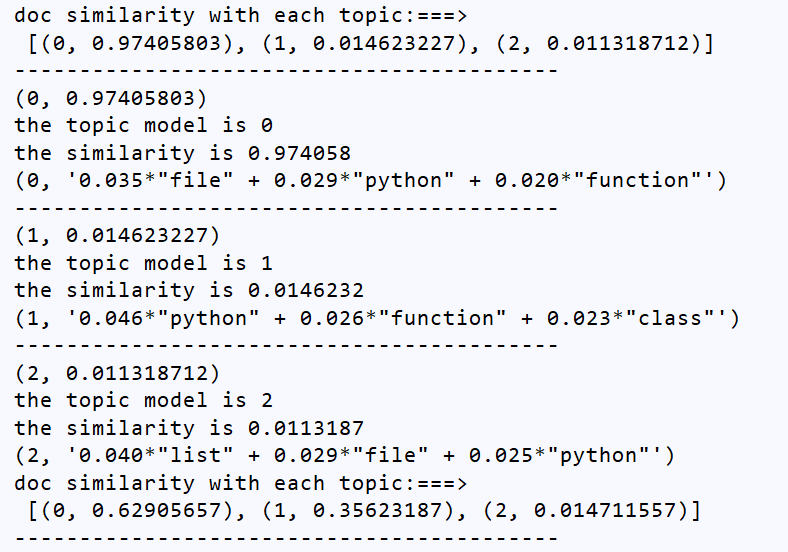


图5.6 文本和主题的相似度

输出结果展示了3个主题向量的构成以及每份文本和每篇主题的相似度。

每份文本和每篇主题的相似度：[(0, 0.97405803), (1, 0.014623227), (2, 0.011318712)]

每个元组的含义为（主题编号，文本和该主题的相似度）

整个列表表示该文本和第1个主题相似度为0.97405803，和第2个主题相似度为0.014623227, 和第3个主题相似度为0.011318712。

主题向量的构成：(0, '0.035\*"file" + 0.029\*"python" + 0.020\*"function"')

元组第一个元素为主题编号，第二个元素为词语的概率\*词语。

### 5.2.2 相似度计算

由于主题模型的结果是给出了每篇文章对应的主题，这些主题是由一系列特征，也就是一些词语和对应的概率构成的元组组成的。作为一个代码标签自动生成系统，这样的结果推荐给用户显然是不合适的。因为每个主题向量中都存在一些无关紧要的词，所以我们只能把它看做是一种中间的结果，根据文献[10][12]考虑利用主题模型的去计算文本之间的相似度，然后选取相似度最高的3篇问答，获取它们的人工标签推荐给用户。

相似度计算可以先调用gensim库中的similarities.MatrixSimilarity用整个语料库构造一个相似度模型。对输入的文本进行预处理后得到词语列表，然后调用doc2bow函数转换成corpus,利用这个corpus创建一个主题模型,导入相似度模型可以得到它和语料库中每份文本的相似度，筛选出相似度最高的文本即可。

筛选出前3个相似度最高的文本之后，根据文本编号去数据库中获取文本对应的人工标签返回给用户



图5.7 相似度计算

## 5.3 模型的储存和加载

上面的4.2.1和4.2.2只是简单的实现了主题模型和TF-IDF模型的创建，但是没有考虑corpus以及两种模型的保存和加载。在系统正常的推荐流程中，为了提高速率，不应该每次重新创建corpus以及模型，而应该首次创建就保存在文件中，以后只需从文件中读取加载即可。因此，模型层的结构如下：

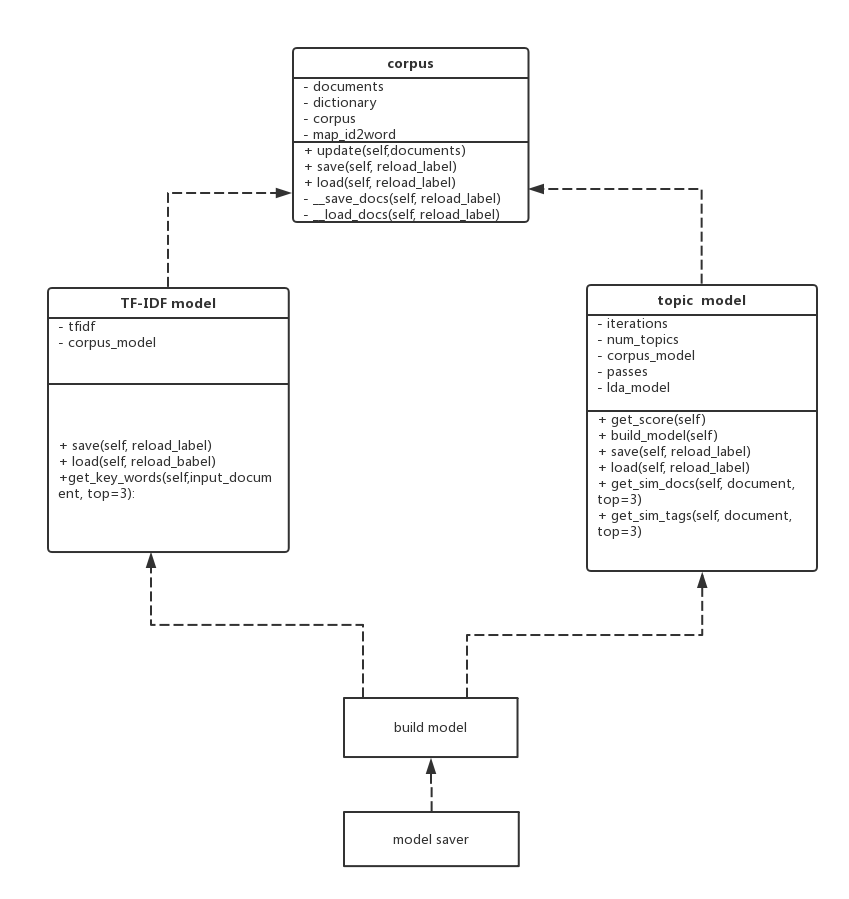


图5.8 模型层结构图

corpus类主要负责语料库的生成、存储和加载。corpus类里存储了训练数据中所有的词语，并且它们是以文本向量的形式存储的，词典，corpus文档词项矩阵,以及将(word，id)转换为(id，word)的词典。

在语料库中有几十万的文档，将这些文档都以list的形式存入内存中是不现实的。Gensim通过streaming corpus接口实现了序列化向量空间语料（向量序列）到硬盘，这样文件会被惰性地从硬盘中读取，每次读取一个文件，而不是一次将所有语料读入主内存。储存中的时候会将语料存储为Matrix Market format的格式，反过来，加载的时候也可以从Matrix Market文件读取语料迭代器，需要注意的是因为是惰性读取，所以需要list化一下。

除了corpus，词典和词语列表也需要存储。词语列表是一个二维数组，存储在文件中，不同文档用换行分割，每份文档的词语用“,”分割。词典使用gensim自带的save函数进行存储，需要的时候可以重新加载。



图5.9 corpos 类代码

在corpus的基础上，主题模型和TF-IDF模型类中也包含创建、存储和加载的操作。

主题模型首先需要创建模型，由于本系统中创建了2个主题模型，语料分别是基于问题标题和问答纯文本的，需要额外用一个参数reload\_label来区分。reload\_label为“body”时，代表语料是基于问答纯文本的；为“title”时，代表语料是基于问题标题的。

首次创建时传入主题模型参数即可。特别的，reload参数默认为false,即根据语料生成模型；再次创建只需要将reload参数设为true,构造函数将会自动调用load函数。代码如下图：

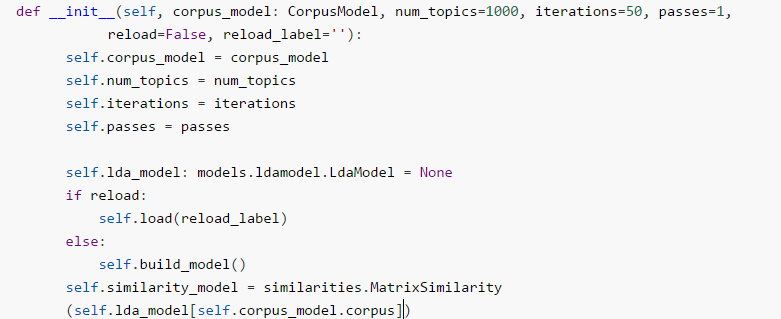


图5.10 TopicModel类的创建

TopicModel类其他的save和load函数和corpus类类似，调用了gensim里topic自带的save和load方法，存储格式也和corpus类中的存储格式一致。

TF-IDF类的创建、存储和加载在这里就不再赘述了，和TopicModel类基本相同。

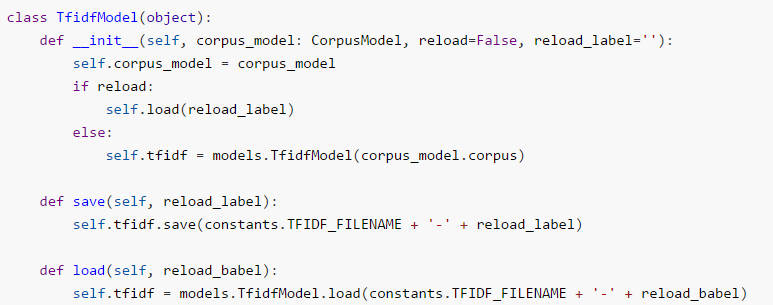


图5.11 TfidfModel类的创建

## 5.4 主题模型详解和实验

### 5.4.1 机器学习理论基础

目前机器学习领域主要分为4种，分别是监督学习，无监督学习，半监督学习和强化学习。本代码标签自动生成的系统就属于监督学习这一场景。这系统中，给包括问答文本在内的代码片段打标签就可以看做⼀个人工训练的过程，通过对部分优质数据集作为训练集训练出模型后，然后将剩余的内容作为测试集通过模型进⾏运算，可以得到剩余所有内容的标签无监督学习指的是，不需要⼈为的去探寻数据中的标签与规律，从数据本身找到⼀个模式。不需要人工分类的过程，将特定内容的或相似的内容归类到⼀起，当⽤户查找内容的时候，就可以通过算法得到同类的内容。聚类算法是典型的无监督算法

a. 有监督学习

有监督学习算法是机器学习任务的一种。它从有标记的训练数据中推导出预测函数。这里有标记的训练数据是指每个训练实例都包括输入和期望的输出。可以用简单的一句话进行概括就是：给定数据，预测标签。

b. 无监督学习

无监督学习是机器学习任务的一种。它从无标记的训练数据中推断结论。最典型的无监督学习就是聚类分析，它可以在探索性数据分析阶段用于发现隐藏的模式或者对数据进行分组。可以用简单的一句话进行概括就是：给定数据，寻找隐藏的结构。

c. 半监督学习

半监督学习在训练阶段结合了大量未标记的数据和少量标签数据。与使用所有标签数据的模型相比，使用训练集的训练模型在训练时可以更为准确，而且训练成本更低。

d. 强化学习

是机器学习的另一个领域。它关注的是软件代理如何在一个环境中采取行动以便最大化某种累积的回报。主要的用途是给定数据，学习如何选择一系列行动，以最大化长期收益。比如学习下围棋就是典型的强化学习。

### 5.4.2 分类和聚类

分类和聚类都是数据挖掘领域中常用的方法，在文本数据分析过程中也有重要的应用。文本分类可以预测文本的类别，用于垃圾邮件的过滤、网页分类、个性化新闻提供等；文本聚类对结构内容相近的文本进行归类，用于实现文档集合的自动整理、搜索信息定位、用户兴趣模式识别等。

a. 聚类算法

分类作为一种监督学习方法，要求必须事先明确知道各个类别的信息，并且断言所有待分类项都有一个类别与之对应。但是很多时候上述条件得不到满足，尤其是在处理海量数据的时候，如果通过预处理使得数据满足分类算法的要求，则代价非常大，这时候可以考虑使用聚类算法。

聚类是一种无监督学习。也就是说，聚类是在预先不知道欲划分类的情况下，根据信息相似度原则进行信息聚类的一种方法。聚类的思想是使得属于同类别的对象之间的差别尽可能的小，而不同类别上的对象的差别尽可能的大。与分类规则不同，进行聚类前并不知道将要划分成几个组和什么样的组，也不知道根据哪些空 间区分规则来定义组。常见的聚类算法包括：K-均值聚类算法、K-中心点聚类算法、CLARANS、BIRCH、CLIQUE、DBSCAN等

d. 分类算法

分类和聚类最重要的区别在于训练样本是否有类别标注。分类模型的构建基于有类别标注的训练样本，属于有监督学习，即每个训练样本的数据对象已经有对应的类标识，一般形式可以表示为：(v1,v2,...,vn; c)，其中，vi表示样本属性或特征，c表示对应的类别。通过分类学习可以形成一个分类函数或分类模型，也就是常说的分类器，该分类器能把数据项映射到给定类别中的某一个类中，进而预测测试数据的类别。 文本分类，就是根据提取到的文本属性或特征，以及人工分标注的文本类别，构建分类器，将文本划分到已有的类别中。

一般的分类方法都可用于文本分类，常用的文本分类算法包括：决策树、神经网络、朴素贝叶斯（naïve bayes）方法、支持向量机（SVM）等等。

### 5.4.3 主题模型

主题模型和聚类不同的地方在于不是对对象进行聚类，而是把它们放入几个分组中，也就是对应的主题，它可以看做是特征的概率分布。通过主题模型对⽂本的生成过程进⾏模拟，通过矫正参数得到不同的主题。将大量的文本通过主题模型的训练得到最终的主题可以看做是⼀种降维的过程，因为降维后得到的主题个数远小于特征数。

这个特性更符合问答网站中类似主题的选取，主题模型不光可以找到代码与代码间的关联也可以找到代码和其上下文（可以是回答、问题也可以是文档）的关联关系，生成的标签效果更佳。目前，主题模型也是自然语言这一领域常用的方法[14]。

主题是由⼀系列相关的单词以及这些单词的条件概率构成的，通常的理解是一篇文章的每个词以⼀定的概率选取了某个主题，并从这个主题中以⼀定的概率选择了某个词。出现的概率可以表示为：

### 5.4.4 主题模型参数实验

根据第二章对主题模型的介绍，有α和β参数的参与，其中α代表每篇⽂档所可能

包含的主题个数，α的值越高也就代表⽂档可以由更多的主题所构成。β指的是组成某⼀个主题的关键字个数，α值越高则代表每个主题中的关键字越多。

⼀般情况下根据文献[]的实验经验将α和β分别取值为 50/K，β= 0.01可以起到平滑数据的作⽤。其他除了主题个数num\_topics和迭代次数iterations，参数皆取默认值即可。

主题模型的参数实验实质是针对训练集中所有问题标题以及所有问答纯文本这两个语料寻找模型效果最好的主题个数和迭代次数。

关于主题模型中如何确定topic数量一直都没有公认的好方法，主要原因在于不同业务对于生成topic的要求是存在差异的。一般的观点是，如果主题模型只是一个中间的步骤主题个数需要尽量多一些，如果主题个数过少会影响最终效果，过多不会对效果有负面影响，但是会造成主题冗余的情况，同时也会加长训练模型的时间。业界最常用的指标包括Perplexity，MPI-score等，但是这些指标只能作为参考，而不能作为标准，甚至gensim包的作者Radim认为 perplexity不可以作为一个好的评价topic质量的指标。

为了评判主题模型的优劣，也可以使用可视化的方法。比如为文档里的词，根据它所属于的主题进行上色或者使用PyLDAvis可视化，观察每个主题包含单词数的多少，以及从各个圆的距离看出主题之间的距离。本训练数据共有12万条左右，如果使用可视化调参依靠肉眼观察比较困难。

模型训练前期如果不是很确定设定多少个topic 最合适，gensim 提供层次狄利克雷过程模型（HdpModel），不需要设定topic 的个数，可以自动生成模型。但是缺点是目前HdpModel并不成熟，所以后者的模型训练时间远⾼于LDA，且主题的区分度和可控性不如LDA，效果不够理想。

根据文献[8][9]，可以使用主题一致性来评判主题模型的好坏，它通过测量主题中高分词汇之间的语义相似程度来评分单个主题，这些测量有助于区分语义上可解释的主题和统计推断的主题。主题一致性将主题的连贯性计算为主题词集V上的成对分布相似度分数的总和，公式如下：

V是描述主题的一组词，而表示保证分数返回实数的平滑因子。

gensim 0.13.1版提供了一个“主题相干性”（topic coherence）数值定量评估的方法。人们对于主题模型的理解更倾向于属于同一主题的单词在语料库中共同出现的频率。“C\_v topic coherence”做的就是这样的工作。gensim提供了几种不同的主题相干性测量方法，其主要的不同在于“共现”的定义不同。Palmetto Online Demo这里定义了几种不同的共现定义，其中C\_v，C\_UCI，C\_NPMI为gensim所采取的可选方法。

所以我们可以直接根据函数得到的topic coherence数值去判断主题模型的优劣。

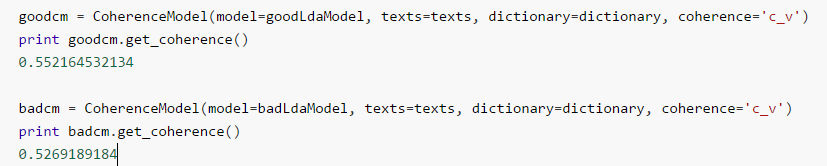


图5.12 topic coherence评估模型

从上图可以清楚的看到goodLdaModel的coherence比badLdaModel的要高，因此孰优孰劣一目了然。所以在在试验中利用topic coherence的数值判断主题模型的好坏。

在测试过程中，首先需要对标准化的数据进行预处理，生成Corpus并储存，然后不断调整参数去创建LDA模型，并调用gensim的函数得到该模型的topic coherence的数值。

由于训练集的数据量比较大，大约有12万条数据，为了减少训练时间，使用了gensim包的多核心的LdaMulticore( )去创建主题模型。

目前系统有2个语料库，一个是基于所有问题标题的，一个是基于所有问答纯文本的，数据量都为12万条。第一个语料明显生成的词典的词语个数会比第二个小，最佳主题个数也会相应少一些，所以会对它们分别进行调参。

对于问题标题的语料库生成主题模型，主题个数选择在50到400之间，间隔为25个主题。由于文本量比较少，所以迭代取25次，最后通过Echart作出曲线，如下图4.25所示，横轴是主题数目，纵轴是topic coherence的数值。从图可以清楚的看出在主题个数在150取到最大值，所以主题数就选择150个，迭代数选择25次。

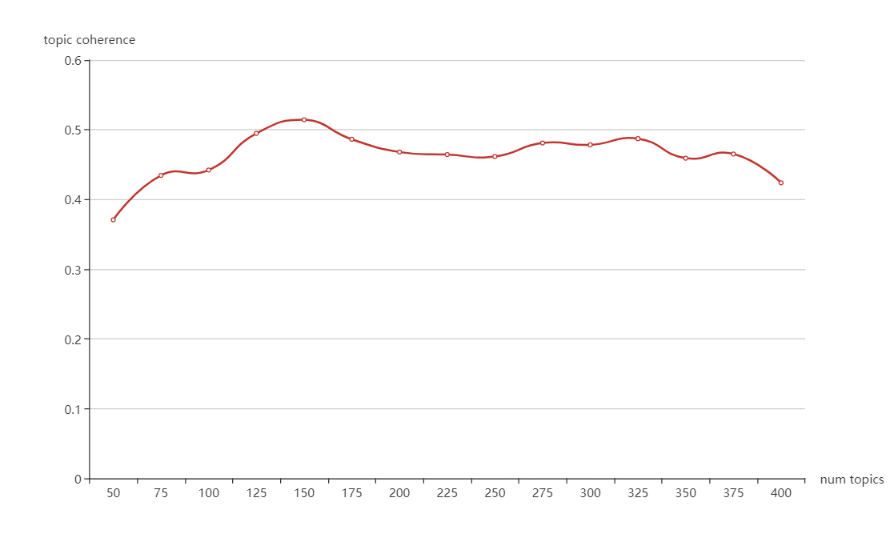


图5.13 基于问题标题语料库的主题模型实验



图5.14 曲线图代码

同样的，对于问答纯文本的语料库生成主题模型，主题个数选择在50到400之间，间隔为25个主题。由于文本量比较大，所以迭代次数取得大一些，取了50,100,250,500,1000。

通过Echart作出曲线, 如下图4.27所示，横轴是主题数目，纵轴是topic coherence的数值，3条曲线分别代表iterations是50,100,250。iterations取50,100的时候，topic coherence数值的最高点在主题个数在50的时候取到，违背一般的经验。iterations取250的时候，主题个数取300的时候效果最佳，比较符合预期的估计。猜想是数据量大的时候，iterations数值过小会影响topic coherence的数值。

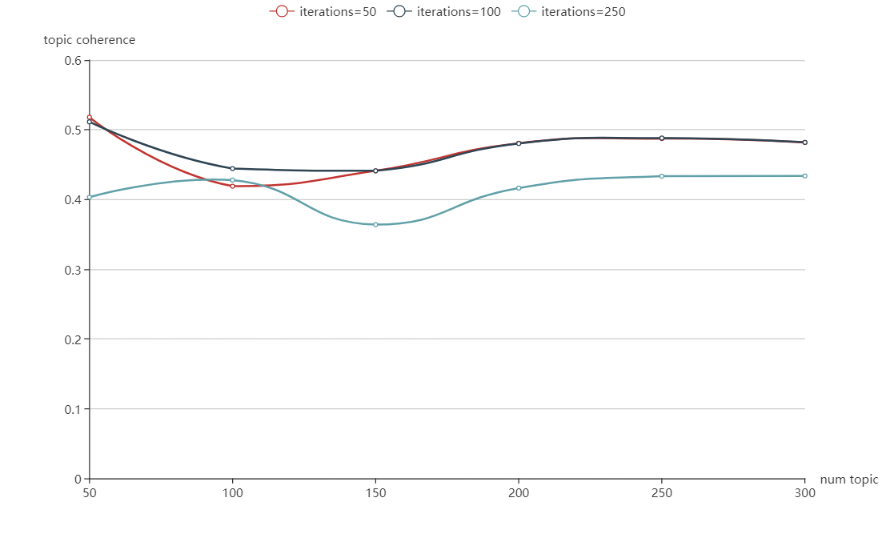


图5.15 在不同iteration下的topic coherence折线图



图5.16 代码截图

为了探究iterations取250的时候，如果主题个数大于300时，topic coherence的数值是否会持续增长，所以num\_topic数值设为350，400，450，500继续试验，结果如下图

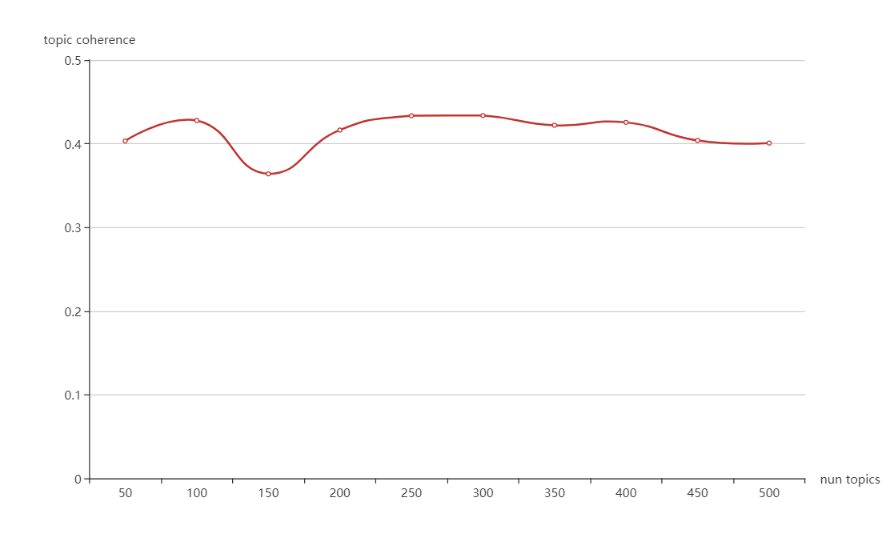


图5.17 iterations为250的topic coherence折线图

从上图可以清晰的观察到主题个数取300时，topic coherence的数值最大，效果最好。为了确认是否300个主题是最佳的，取iterations为500时，主题个数取200到350,，间隔为50，试验效果如下图，可以清楚的观察到仍是主题个数取300时效果最好：

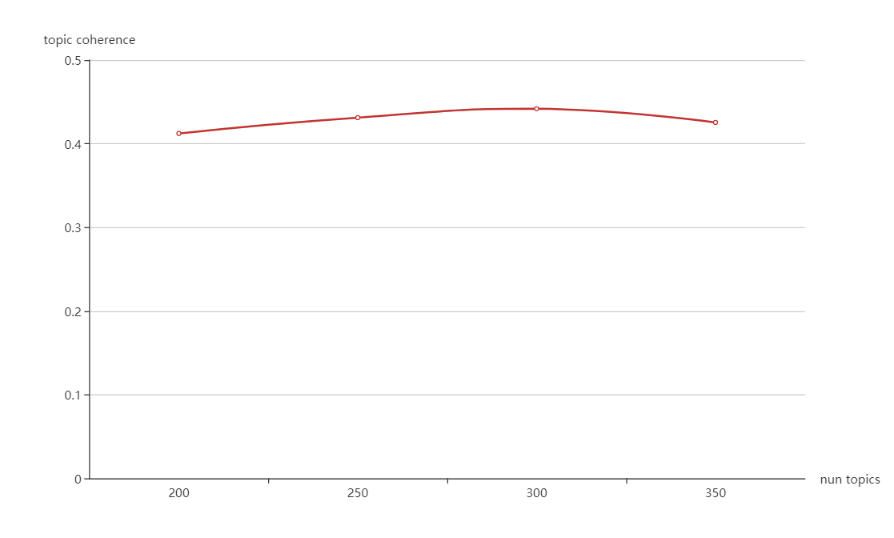


图5.18 iterations为500的topic coherence折线图

在以上试验的基础上，进一步探究合适的iterations的数值，取iterations=1000，num topics=300创建主题模型，获取topic coherence的数值，将结果绘制成折线图，这次x轴代表iterations，y轴代表topic coherence：

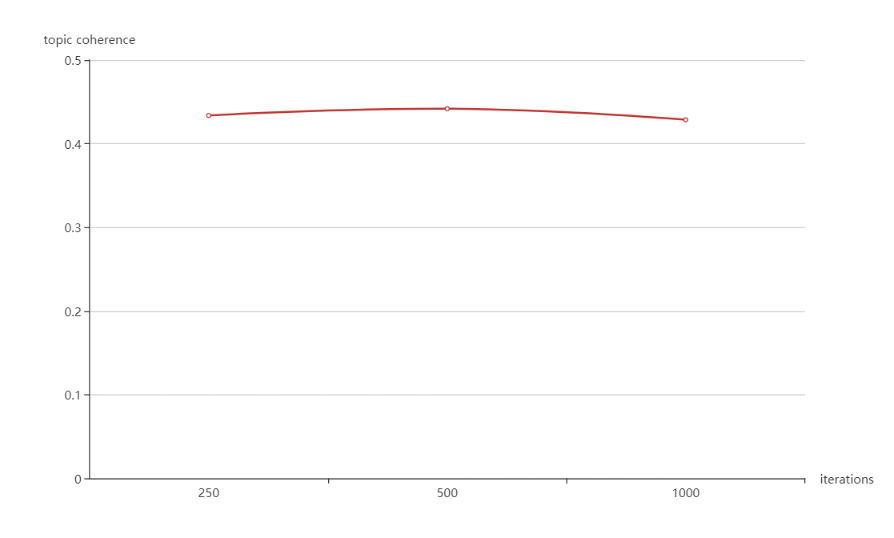


图5.19 探究iterations对topic coherence的影响

从上图可以观察到iterations取500，num topics取300时，topic coherence的数值最大，效果最好。

因此最终主题模型的参数确定为iterations取500，num topics取300，α和β分别取值为 50/K，β= 0.01，其他参数均取默认值

# 6 代码分析

本课设实现了一个针对问答网站和python有关的问题，对代码片段实现自动化标签生成的系统。系统主要的算法为文本分析和代码分析。文本分析的常规做法就是分词、生成文本-词频矩阵，后续可能涉及词频-逆文档率（TF-IDF）、主题模型等。目前常用的预处理、文本分析的对象都是日常语言的文本，包括新闻、微博等，但是针对结构性强、严格遵守语法的计算机语言文本，始终缺乏进行处理的通用方法，所以代码分析可以看作是在普通文本分析的基础上，根据文献[11]，利用代码的语法规则和结构特性结合分析目的，有效提取代码信息。

进行代码分析优先考虑的是代码的长度和完整性。如果针对完整项目的代码以及一份文件中的完整代码，可以考虑用语法树进行解析，甚至可以结合注释文本对代码块的功能点进行解析。本系统的数据源是问答网站中的python代码片段，它往往是根据提问者的不能正常运行的代码给出的解决方案，所以在长度上普遍只有几行，不仅没有明确的功能点，在语法上甚至有些也不是完整的函数、类。为了更好地挖掘代码信息,生成代码标签[15]，需要结合第三方的资源，比如python官方文档, 从问答的分类和帮助开发者快速搜索、学习的角度出发进行代码标签的提取。

## 6.1 代码分析的流程探究

对于开发者来说，面对一段不熟悉的代码，如果能获取代码API使用情况，就可以明确代码是和哪些模块相关的以及它主要的功能，同时也能提高开发者的学习效率。

由于Stack Overflow上的问答里大部分代码片段比较短小，代码信息量少，在语法上也存在和标准语法有细微出入的可能（可能不止有python一种语言）。由于上述种种原因，无法使用语法树工具对代码进行解析，这让代码分析变得有些困难。

为了弥补代码信息量少的不足，考虑通过参考python标准文档，根据代码中对标准库中函数的调用，获取代码片段涉及了哪些类或者模块、包，并可以作为代码的标签返回给用户。

考虑到Django是python中常用的web框架，它提供了全套的解决方案，有强大的URL路由配置和自助管理后台，因而和Django有关的问题是python问题的一个重要分支。对于从标题分析中发现和Django有关的问题，将和一般的python问题区分，需要参考Django标准文档进行分析。

## 6.2 Python的模块、包、类

Python在处理功能复用和功能颗粒度划分时采用了类、模块、包的结构。这种处理跟C++中的类和名字空间类似，但更接近于Java所采用的概念。

类：将数据（变量）和操作（函数）进行封装，以便将来的复用。

模块：在Python中可理解为对应于一个文件，并且在文件中定义了某些函数和变量。在其他需要这些功能的文件中，只要导入这模块，就可重用这些函数和变量。一般调用模块中函数或者变量的时候会使用module\_name.fun\_name，和module\_name.var\_name。这样的语义用法使模块看起来很像类或者名字空间，可将module\_name 理解为名字限定符。

包：在python中是一个有层次的文件目录结构，它定义了由n个模块或n个子包组成的python应用程序执行环境。可以使用import语句导入包，或者from a import b语句来导入包中的部分模块包是一个包含\_\_init\_\_.py文件的目录，该目录下一定得有这个\_\_init\_\_.py文件和其它模块或子包。

在分析代码的时候，首先要获取有哪些函数调用，然后根据标准库去搜索这些函数属于哪些类然后返回。需要注意的是，函数的父级不一定是类，也有可能是包和模块。

## 6.3 获取标准库函数的父级

首先爬取了python和django官方文档中和类、包、模块、函数有关的html, 使用Beautiful Soap对html文档进行解析，主要目的是获取每一个函数的父级（可能是类、模块、函数）。

html文档大致结构如下

<td>

<li> <a>函数（父级）</a></li>

<li><a>函数（父级）</a></li>

</td>

提取a标签内的文本函数及其父级即可。



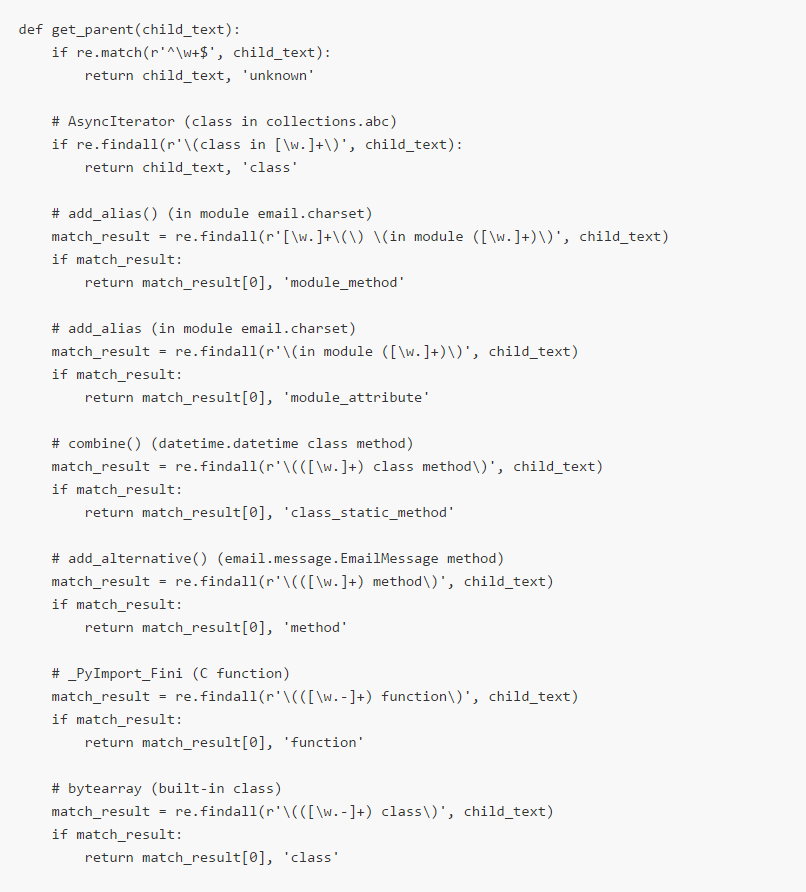
图6.1 HTML文件

对于python的官方文档，解析的语句有以下几种类型：

* AsyncIterator (class in collections.abc)
* add\_alias() (in module email.charset)
* add\_alias (in module email.charset)
* combine() (datetime.datetime class method)
* add\_alternative() (email.message.EmailMessage method)
* \_PyImport\_Fini (C function)
* bytearray (built-in class)
* license (built-in variable)
* inquiry (C type)
* ALWAYS\_TYPED\_ACTIONS (optparse.Option attribute)
* asyncio (module)

Django中需要解析的语句也是类似的，一般形式为[函数或者变量(类、模块或者包)]

通过解析，可以获取每个函数或者变量对应的父级，以及自身的属性，然后以[函数本身, 父级, 自身类型]的形式存储在列表中。



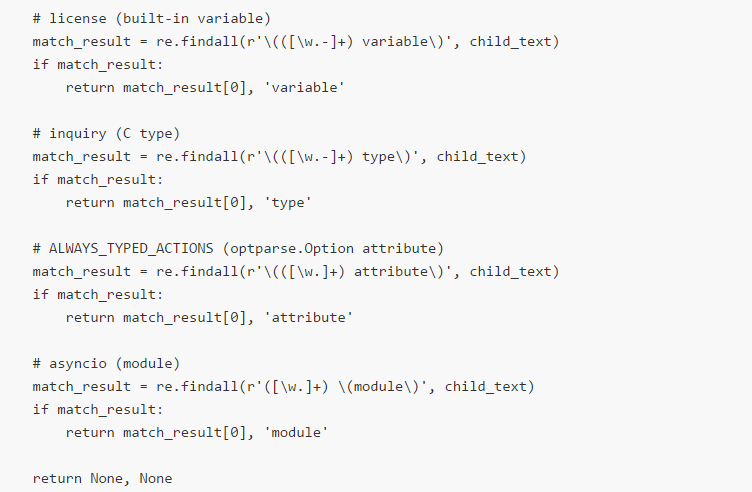


图6.2 获取标准库中函数对应的父级代码

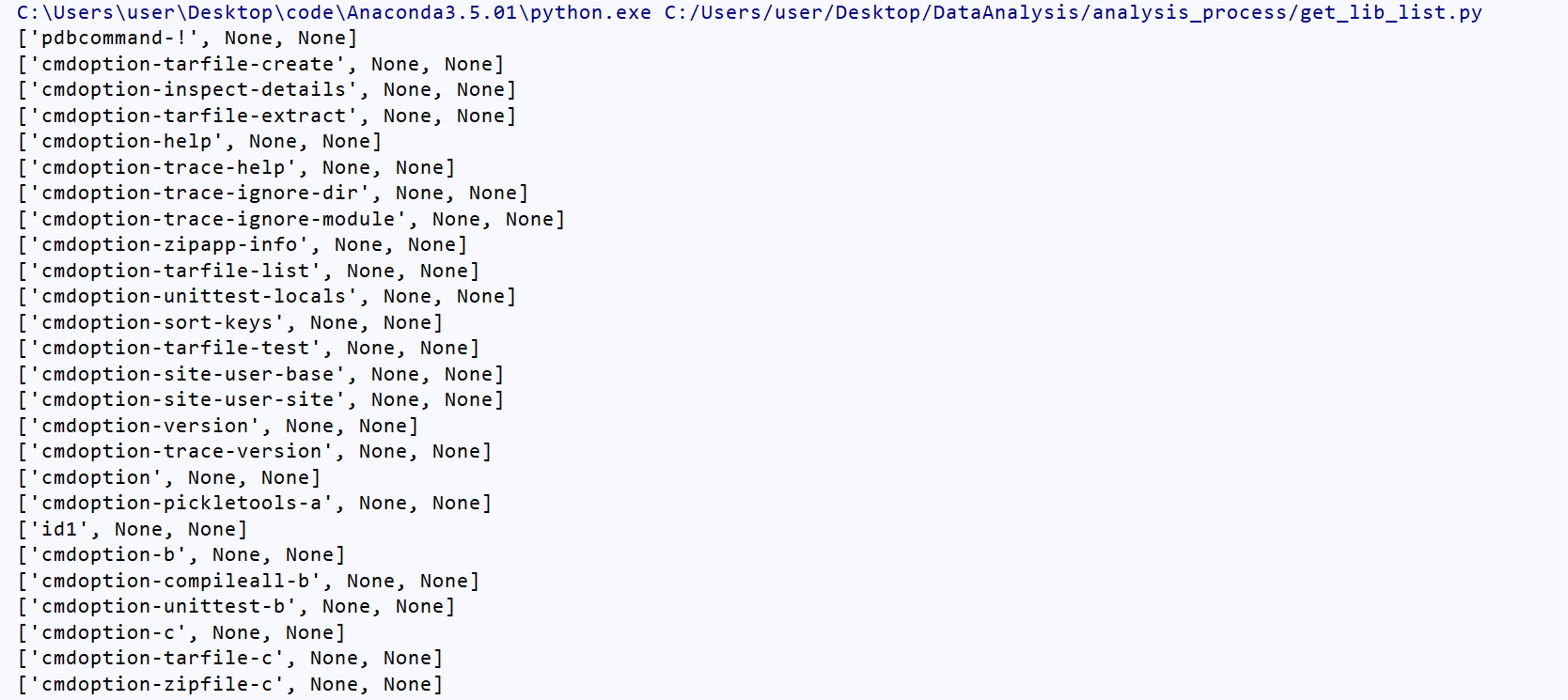


图6.3 输出标准库中函数对应的父级

## 6.4 分析流程

对用户输入的代码片段，如果有import语句，首先根据正则表达去匹配获取引用的模块、库、包、函数作为代码标签的一部分。

import语句的一般格式如下:

from a import b, c …

import a

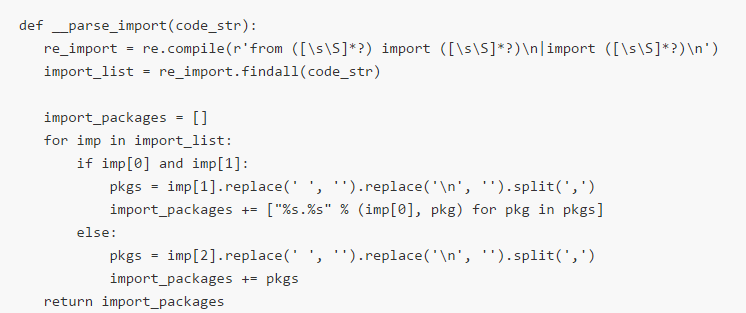


图6.4 解析import语句

然后要正则提取代码里每一个调用的函数名，根据（）来判断，（）前面的是函数名，允许函数名和（）之间有空格。一般形式是function（），也有可能是a.funtion（a 是一个对象）。

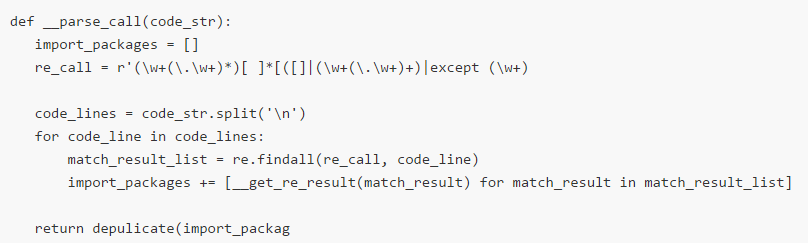


图6.5 提取所有函数名

在已经获取标准库函数列表（元素为[函数本身, 父级, 自身类型]）的情况下，对每一个获取的函数名，查询是否在这个列表里。

需要注意的是，代码里调用的函数有很多复杂的情况，例如有a.b.c（），可能部分函数是属于标准库的，部分是自己实现的，所以查询的时候a.b, b.c, c,a. b.c都需要查询。例如

“a.join”可以匹配到'queue.Queue.join'、'bytes.join'、'str.join'、'os.path.join',但是无法匹配到'pathlib.PurePath.joinpath', 'multiprocessing.JoinableQueue'。

考虑到built-in模块中类似“print”,“range”,“str”等常用函数，和代码片段的核心功能往往没有关联，所以父级是built-in模块的一律预以去除。

一般的，返回的父级不止一个，也就是说很多类、模块、包里有重名的函数，为了提高准确度,如果返回的父级超过最大值，将返回本身。

最后得到代码片段中涉及的类、模块、和库的列表，去除重复之后，作为代码标签推荐给用户。

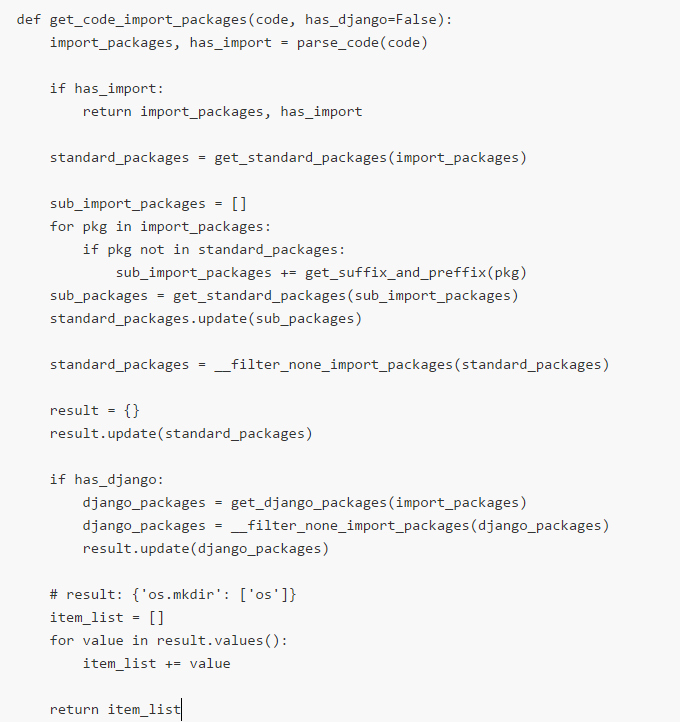


图6.6 获取代码中所有涉及的类、模块、包

# 7 标签管理和评价

本课设对开源问答网站StackOverFlow数据集的14万条数据 按照2：8的比例随机分成测试集和训练集。对训练集的数据进行标准化，预处理，生成语料之后创建TF-IDF模型和主题模型并保存下来。测试集中数据进行标准化，预处理后分别进行文本分析和代码分析，其中文本分析需要将测试集每条数据生成的语料导入训练集生成的模型中，最后生成标签存储在数据库中。

为了检验分析效果和模型效果，需要对测试集生成的标签质量进行评估，并针对不足之处对系统进行改进。

## 7.1 标签生成和管理

创建TagsGenerator类来管理模型，调用了数据标准化模块，预处理模块，文本分析和代码分析模块，实现了为用户输入生成标签，标签一共有5种：

* 问题标题的标签
* TF-IDF的标签
* 问答纯文本主题模型标签
* 问题标题主题模型标签
* 代码中和引用包、模块、库有关的标签

在这5种标签的基础上，设置重复标签列表和全部标签列表，代码如下图：



图7.1 标签生成和管理代码

## 7.2 标签种类和等级

根据文献[17],不难看出针对某个代码片段（包括它对应问答的文本）的标签大致可以分成3类，一类是标签的文字出现在了文本或代码中，另一类标签是依靠有经验的开发者使用自己的语言概括总结问题的核心，标签本身并不能文本或者代码中直接找到，最后一类是自动标签和人工标签的结合。

显然，人工标签的价值是难以被自动标签完全取代的，因为本系统的重点是从文本和代码提取标签，无法直接创造标签，但是不可否认的是，后者有很高的价值，这也在某种程度上体现了人工标签和自动的标签相结合才是最好的方式。

如果对第一类自动标签从分析方法来分类：

（1）从文本中提取的标签：根据文献[]利用相关的文档、注释、问题回答这些文本，提取代码的功能点或者涉及的技术。文本质量的越高，数量越多，效果越好。

（2）从代码中提取的标签：一般根据自身的需要，利用代码结构化的特点去提取信息，可以使代码调用的API等。

## 7.3 标签质量评估方法

在自然语言处理, 信息检索等领域, 评估是一个必要的工作, 而其评价指标往往有如下几种：

（1） 汉明损失(Hamming loss)

表示所有标签中错误样本的比例，该值越小则表明分类能力越强。计算公式如下，其中|D|表示样本总数，|L|表示标签总数，xi和yi分别表示预测结果和真实情况，xor表示异或运算：

（2） 精确率(precision)、召回率(recall)和F1-Measure[16]

以检索为例，可以把搜索情况用下图表示：

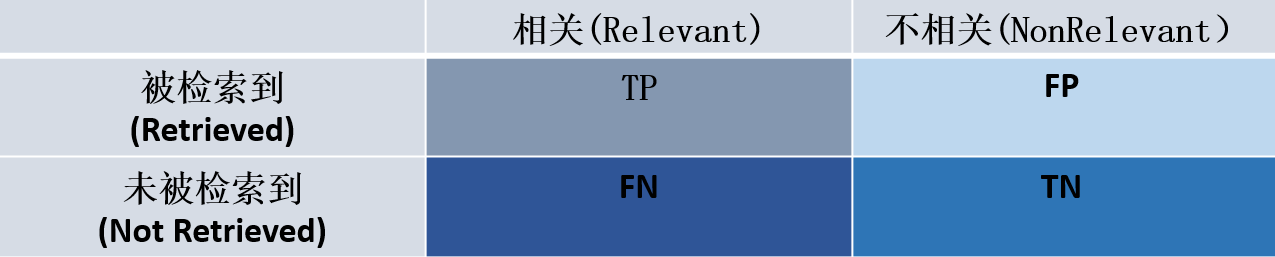


图7.2召回率和准确率示意图

* TP：检索到并且内容相关的
* FP：检索到但是内容不相关的
* FN：未检索到的，但内容却是相关的
* TN：未检索到的，内容也不相关的

准确率（precision）计算的是所有"正确被检索的结果(TP)"占所有"实际被检索到的(TP+FP)的比例，计算公式为：

召回率（recall）计算的是所有"正确被检索的结果(TP)"占所有"应该检索到的结果(TP+FN)"的比例,计算公式为：

F1值是精度和召回率的调和平均数,计算公式为：

（3） 准确匹配

这个是最严格的标准，指的是是预测结果和真实情况完全一致时的样本数与总的样本数的比值。

（4）可视化

基于词频的文本可视化可以直观的反映所有自动生成标签的总体质量。

（5）实际

针对测试集自动生成的标签，可以参考数据集中对应问题的人工标签去进行质量评估，主要目的是评测系统自动生成提取的标签是否覆盖了人工标签列表中的每一个标签，类似上文中介绍召回率的概念。

例如有人工标签如下：

"Tag" : [

"python",

"string",

"escaping"

]

自动标签如下：

"all" : [

"django-q",

"escaping",

"re",

"operator",

"python",

"re.regex",

"string",

"iteration",

"hell",

"version",

"addslashes",

"django",

"htmlspecialchars",

"stripslashes",

"nested-lists",

"php",

"html-entities"

]

自动标签下划线的部分和人工标签一致，并且人工标签列表中的所有项都被覆盖了，我们可以认为它的召回率是：

覆盖的标准并不是字符串完全一致，考虑到表述方式的不同，只要人工标签和自动标签存在包含关系即可，可以是人工标签是自动标签的子串，也可以是自动标签是人工标签的子串，只要出现上述任一情况，均认为覆盖。

对于标签质量的评估，很难从汉明损失或者精度的角度去进行，即需要判断自动生成提取标签和问答是完全不相关的。由于系统最后生成的所有标签会返回给用户开发者，开发者可以进行筛选和增删，所以原则上我们会推荐尽可能多的标签，并且对于用不同分析方法得到的共同标签予以颜色上的标注，认为它们更有可能被用户采纳。

需要注意的是人工标签一般不会含有精确的代码信息，即不可能出现代码中所使用的python模块、函数、库等信息，所以对本系统的代码标签质量的评估比较困难，需要人力去判断。由于资源的限制，所以本课设并没有做这方面的评估工作。原则上仍旧会推荐尽可能多的标签

## 7.4 评估结果

（1）对测试集中自动生成的标签基于词频生成词云

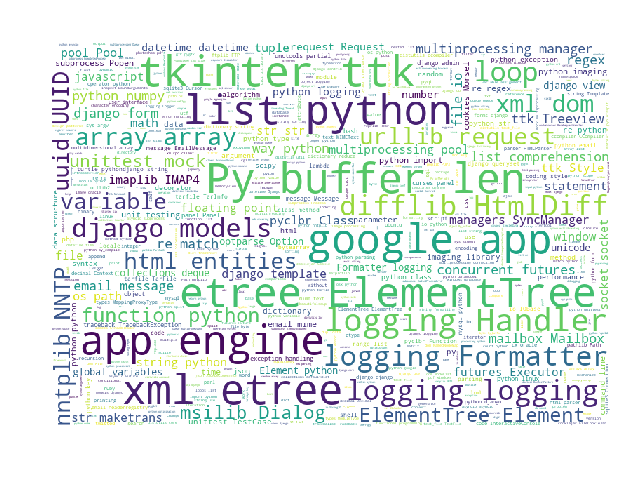


图7.3 测试集生成标签的词云

（2）针对本系统自动生成标签的特点，需要从文本代码中提取，但是人工标签中有些词语是开发者自己总结加工的，并不能从文本代码中直接找到，所以会以2种方式计算召回率的时候，一种是人工标签中会剔除无法在文本代码中找到的词，一种则不做处理，结果如下表：

图5.2召回率

|  |  |
| --- | --- |
|  | 召回率 |
| 剔除无法直接找到的词 | 0.9657 |
| 不做处理 | 0.9165 |

# 8 系统整体框架和部署

所有的主要功能都已经开发完成，可以独立的在本地环境中运行。为了实现最终的搜索引擎，需要进行前后端的开发，完成服务器部署，完成浏览器端的应用。本章将介绍部分前后端实现的框架和核心代码，并展示前端页面。

## 8.1 前后端框架

后端使用Django框架实现，Django为Python的常用Web框架。前端使用React实现，用于构建用户界面。React具有高效，灵活的优点；采用声明范式，可以轻松描述应用；通过 React 构建组件能够使得代码更加容易得到复用。最后用Node来转发代理，请求都先到node这里，然后一部分请求返回前端页面；另外一部分，转发给Python的Django

## 8.2 项目目录结构

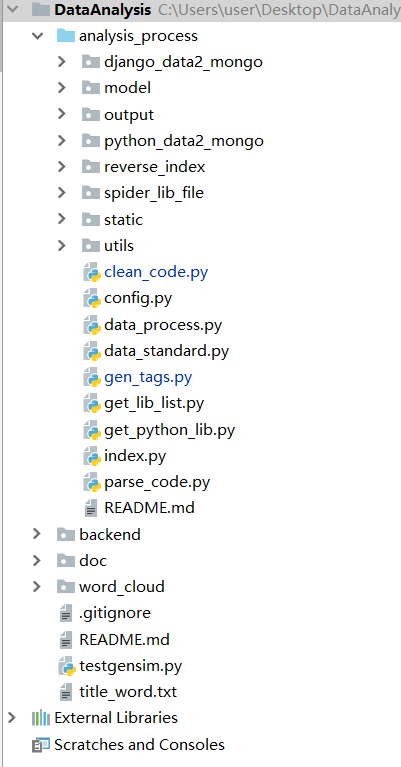


图8.2 项目目录结构

## 8.3 前端界面展示

这是用户输入问题的界面，有上下两个输入框。上面的输入框用于输入标题，在下面的输入框中用户可以输入文字和代码，如果选中代码部分点击输入框上面的“”，会将输入转化成代码格式。输入完毕后点击提交，系统自动生成标签，返回给前端，如下图：

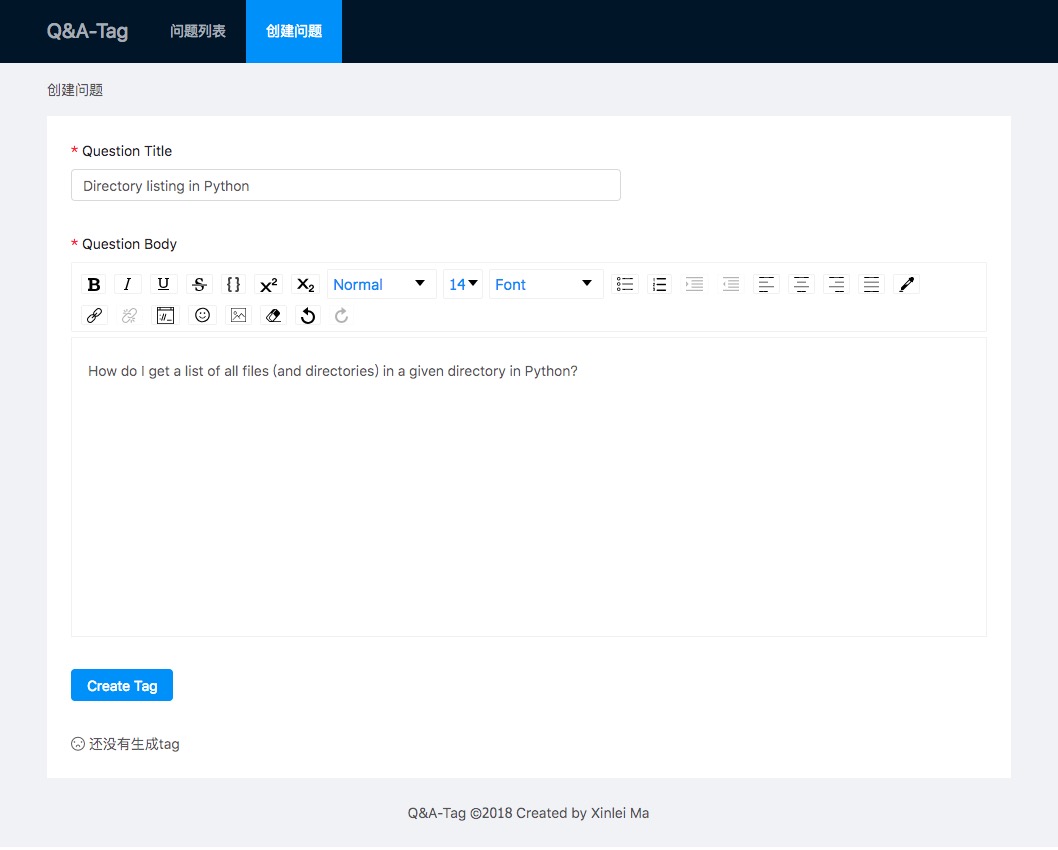


图8.2创建问题

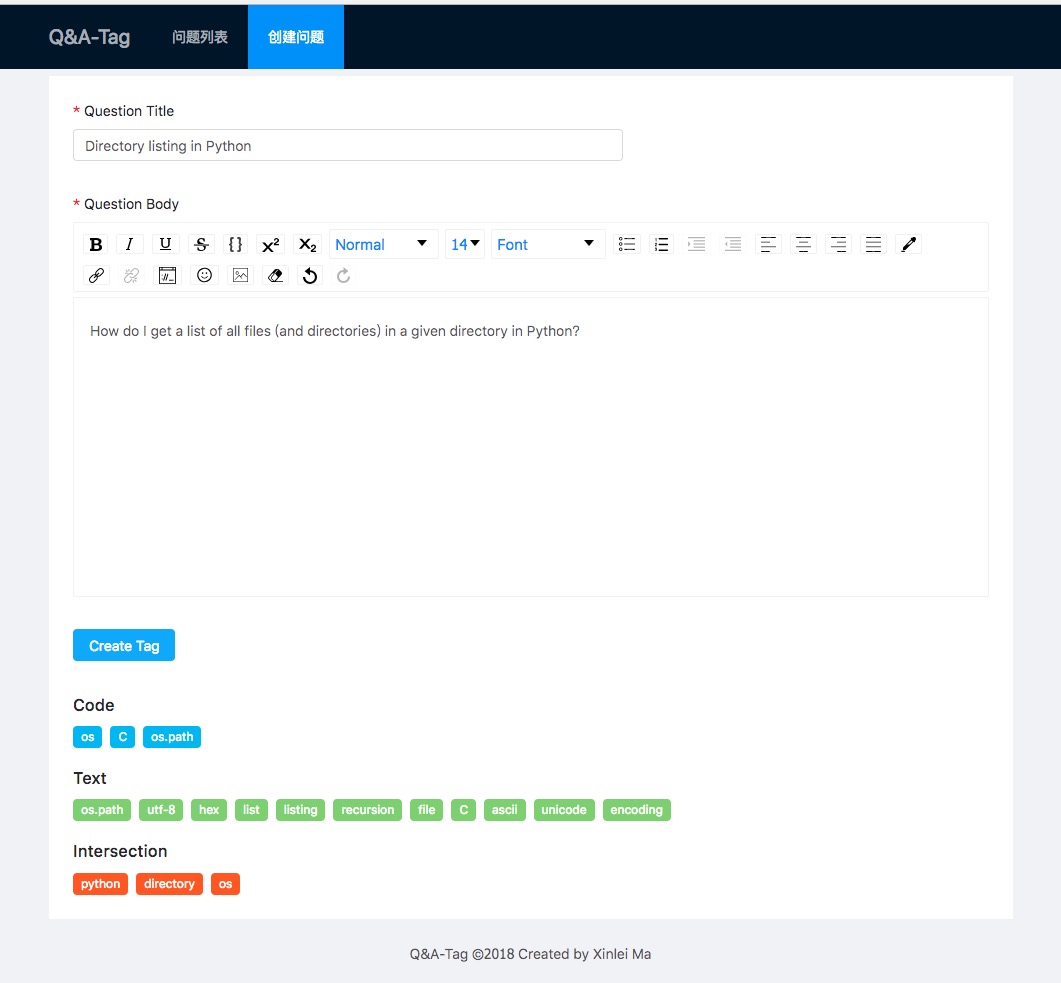


图8.3 提交创建的问题后生成标签

这是问题列表界面，用户可以点击问题进入详细页面进行回答。

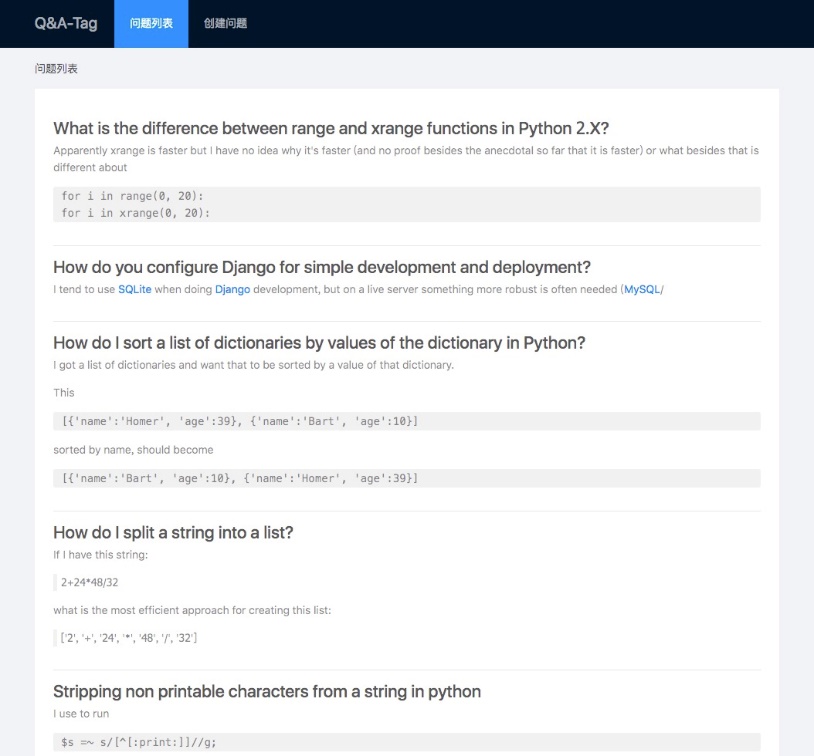


图8.4 问题列表

这是用户进入问题的详情，然后回答问题的界面，操作和提问题基本一致，但是不需邀输入问题标题，直接输入回答即可。点击提交以后系统会自动生成标签返回给前端

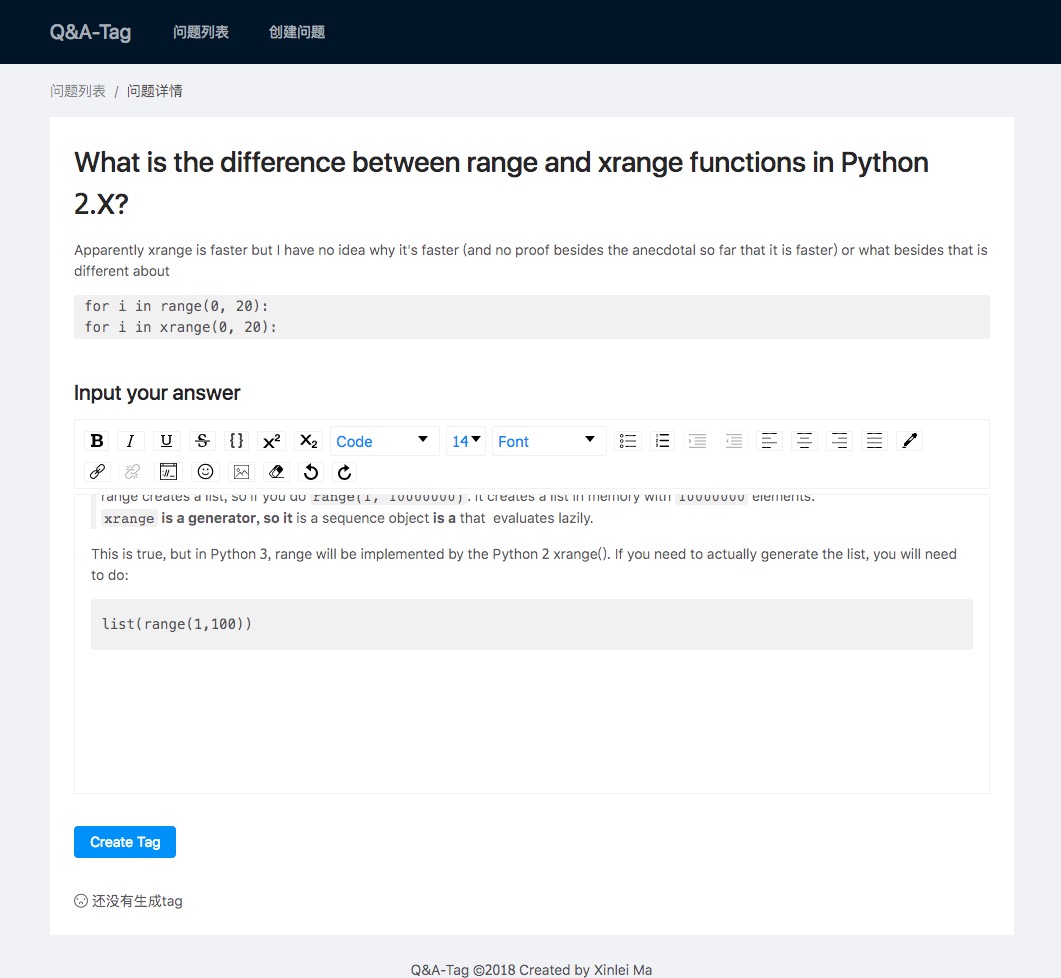


图8.5 回答问题

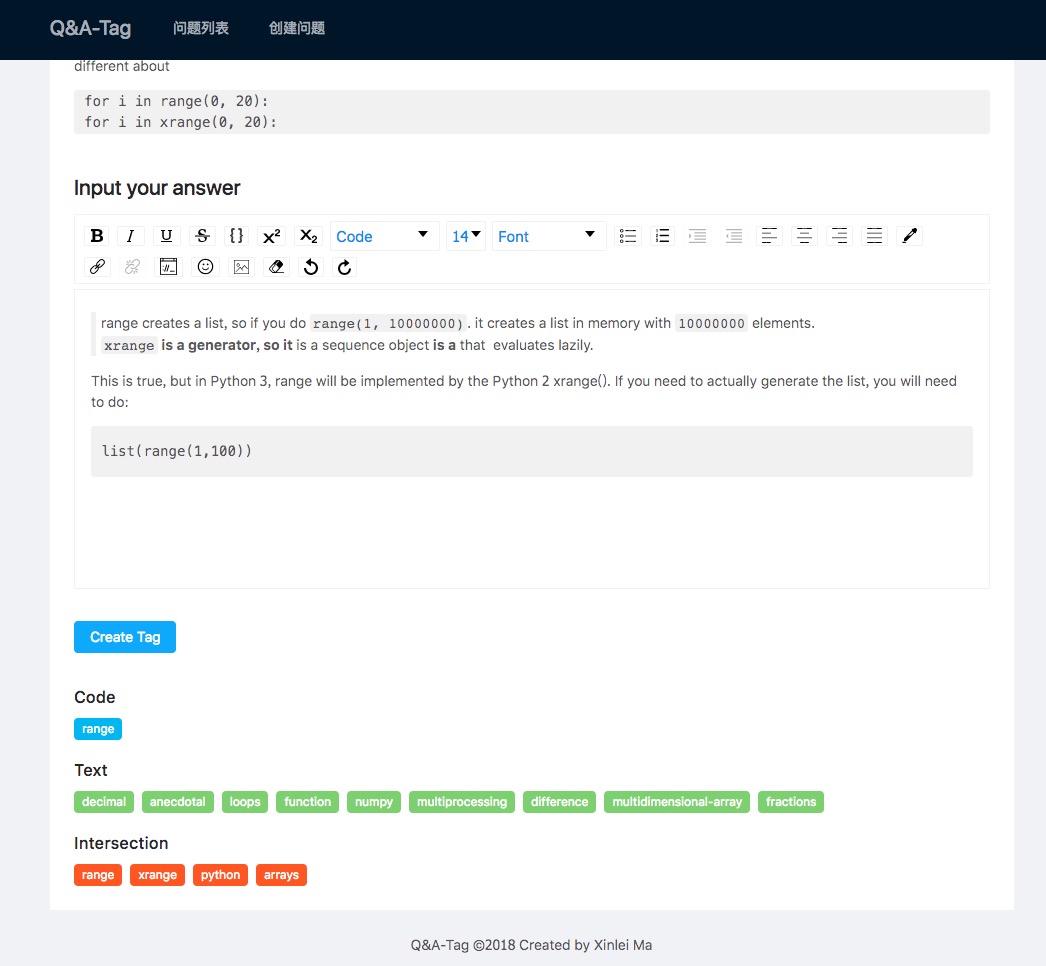


图8.6 提交回答的问题后生成标签

## 8.4 系统部署

最后将前后端分别放到服务器上运行，在服务器上，使用pm2工具启动node 集群和python服务器，支持动态重载。

# 9 总结与展望

以上，所有的功能都已经实现，性能上基本符合系统的预期。虽然在实现过程中也遇到了很多问题，比如在探究为代码片段打标签的算法时遇到了瓶颈，因为目前基本没有通用的算法流程，网上也很难找到相关的资料，幸而在和导师多次沟通，并且阅读相关论文之后明确了思路，找到了合适的代码分析方法。开始的时候自动标签质量不高，为了提高自动标签质量，反复研究数据特点，改进预处理方法，最终也取得了较好的效果。

由于本系统针对的是问答网站的python问答的代码片段，所以在代码分析的时候考虑的是python的语法规则和标准文档，未来可以进一步扩展到其他常用的代码语言，比如java ,c++等。结合这些语言的语法规则和结构特点，调整分析的方法和策略。除此之外，代码片段标签的自动化生成可以有更多的应用场景，而不仅仅局限于为问答网站的提问者和回答者的代码片段进行标注，例如可以为软件公司的代码资源进行分类，或者可以将自动标签作为中间步骤为开发者做代码范例推荐。当然针对不同的场景，不同类型的代码，预处理、文本分析和代码分析的算法需要调整。

针对本系统内部，如何完善预处理、文本分析和代码分析算法，提高标签质量，也是一个值得思考的问题。目前文本分析得到的标签仍旧有一些基本没有代码含义的关键词，为了杜绝这种情况，考虑可以扩大数据量，建立一个相对完整的python标签库，加入一些有经验的开发者的人工标签，推荐给用户之前系统需要判断分析得到的关键词是否在标签库中的标签列表中（不需要严格匹配，只需要存在共同的子串即可），这样可以在一定程度上提高标签的规范性和合理性。

除此之外，由于TF-IDF模型和主题模型的建立需要训练数据，提高训练数据的质量也是重要的一环。目前只是简单的选取了一些有代码片段的问答方便后续的分析。在之后的改进中，可以选取支持数、文本篇幅、代码篇幅、甚至是用户历史信誉作为评判的标准提高训练数据的质量。

标签的质量评估也不够完善，目前采用自动评测算出召回率的方式来判断系统的性能。如果资源允许的话，可以邀请开发者作为志愿者人工判断标签的合理性，比如标签是否和问答文本和代码完全无关，这样可以进一步计算精确度和F1值，完善评测机制。在目前的系统中，推荐标签是以尽可能覆盖所有内容的方式，精确度有所缺失。

# 参考文献

1. Lutz, Mark.  Learning python[M]. " O'Reilly Media, Inc.", 2013.
2. D.M. Blei, A.Y. Ng, and M.I. Jordan,“Latent Dirichlet Allocation,” J.Machine Learning Research, vol. 3, 2003, pp. 993–1022.
3. Han J, Haihong E, Le G, et al. Survey on NoSQL database[C].
4. Bird, Steven, Ewan Klein, and Edward Loper. Natural language processing with Python[M]. " O'Reilly Media, Inc.", 2009.
5. Ramos, Juan. "Using tf-idf to determine word relevance in document queries." Proceedings of the first instructional conference on machine learning. 2003.
6. 施聪莺.TF-IDF算法研究综述[J].计算机应用，2009-06-30.
7. Xi Qiu, Christopher Stewart. Topic words analysis based on LDA model [J]. CoRR abs/1405.3726 (2014).
8. Stevens, Keith, Kegelmeyer, Philip, Andrzejewski, David, and Buttler, David. Exploring topic coherence over many models and many topics. In EMNLP, 2012.
9. Mimno, David, Wallach, Hanna, Talley, Edmund, Leenders, Miriam, and McCallum, Andrew. Optimizing semantic coherence in topic models. In EMNLP, 2011.
10. 王振振.基于LDA主题模型的文本相似度计算.计算机科学[J]，2013-12-15.
11. Sultan S. Alqahtani, Juergen Rilling. “An Ontology-Based Approach to Automate Tagging of Software Artifacts” in 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2017, pp.169-174.
12. 许冶冰，刘超.基于主题的文档与代码间关联关系的提取研究[J].计算机工程与应用，2009.
13. S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor,P. Baldi, and C. Lopes. A search engine for open source code supporting structure-based search. OOPSLA'06, pages 681{682, New York, NY, USA, 2006.
14. 徐戈,王厚峰.自然语言处理中主题模型的发展[J].计算机学报,2011,34(8):1423-1436.DOI:10.3724/SP.J.1016.2011.01423.
15. X. Xia et al., “Tag Recommendation in Software Information Sites,” Proc.10th Working Conf. Mining Software Repositories, 2013, pp. 287–296.
16. D.M. Powers, “Evaluation: From Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation,” J. Machine Learning Technologies, vol. 2, no. 1, 2011, pp.37–63.
17. Maleknaz Nayebi, Shaikh Jeeshan Kabeer, Guenther Ruhe, Chris Carlson, Francis Chew, “Hybrid Labels Are the New Measure,” IEEE Computer Society, 2017, pp. 54-57.

# 谢 辞

系统的实现构建和论文的撰写在在经历了一系列阻碍之后终于完成了，我要特别感谢导师刘岩⽼师。从最开始对如何写论文的一无所知，到一次又一次由浅入深的探讨，再到最后系统成型，刘岩老师指导我理清课题的脉络，并和我⼀起完善细节，让我有了明确的思路和方向

由于我对代码分析领域不甚了解 ，刘岩老师也在和我交流的过程中，引导我了解文本分析并以此为基础，完成对代码分析算法的探索。正是刘岩老师耐心的指导，让我最终完成了代码标签自动推荐系统，

尽管系统还有不是最完善的功能，但我非常希望以后能和刘岩老师保持学术上的沟通。在

这段时间⾥我也渐渐的对源代码分析产生了兴趣，希望刘岩⽼师能给我提供指点，我也会在日后完善学习算法，将系统进一步优化。