单位代码 10006
学　　号 13061010
分 类 号 TP31

北京航空航天大學
BEIHANG UNIVERSITY

毕业设计(论文)

针对 Github 项目问题的标签推荐方法研究

学 院 名 称　计算机学院
专 业 名 称　计算机科学与技术
学 生 姓 名　王国鸿
指 导 教 师　蒋　竞

2017 年 **6** 月

针对 Github 项目问题的标签推荐方法研究

王国鸿

北京航空航天大学

北京航空航天大学

本科毕业设计（论文）任务书

Ⅰ、毕业设计（论文）题目：

针对 Github 项目问题的标签推荐方法研究

Ⅱ、毕业设计（论文）使用的原始数据（数据）及设计技术要求：

原始资料为 Github 网站上爬取的问题模块数据。

设计技术要求：实现标签推荐到文本分类的转化，并利用爬取的 Github 数据进行文本预处理和关键元素的加权，完成多标签文本分类器的构建，得到推荐方法。

Ⅲ、毕业设计（论文）工作内容：

毕业设计的主要内容是实现针对 Github 项目问题的标签推荐方法。通过对 Github 问题模块的分析，爬取相关数据，提取特征，并对部分特征进行加权，形成特征向量，然后利用 MEKA 构建出多标签分类器，得到 TRBF(Tag Recommendation Based on Features)标签推荐方法。最后进行 TRBF 方法的评估。

Ⅳ、主要参考资料：

[1] Jiang Jing, JiaHuan He, and Xue-Yuan Chen. CoreDevRec: Automatic Core Member Recommendation for Contribution[J]. Journal of Computer Science and Technology,2016.

[2] L. Dabbish C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository[C]. CSCW 12 Computer Supported Cooperative Work, Seattle, WA, USA, February 11-15, 2012.

[3] Casalnuovo C, Vasilescu B, Devanbu P, et al. Developer onboarding in GitHub: the role of prior social links and language experience[C]. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering,2015.

[4] TF Bissyande，D Lo，L Jiang，L Reveillere. Got Issues? Who Cares About It? [C]. IEEE International Conference on Software Analysis, Evolution & Reengineering, 2013.

[5] Aggarwal K, Hindle A, Stroulia E. Co-evolution of projects documentation and popularity within github[C]. Proceedings of the 11th Working Conference on Mining Sortware Repositories,2014.

计算机 学院 计算机科学与技术 专业类 130611 班

学生 王国鸿

毕业设计（论文）时间： 2017 年 3 月 1 日至 2017 年 6 月 11 日

答辩时间： 2017 年 6 月 1 日

成　　绩：

指导教师：蒋竞

兼职教师或答疑教师（并指出所负责部分）：

_____

_____系（教研室） 主任（签字）：_____

注：任务书应该附在已完成的毕业设计（论文）的首页。

本人声明

我声明，本论文及其研究工作是由本人在导师指导下独立完成的，在完成论文时所利用的一切数据均已在参考文献中列出。

作者：王国鸿

签字：

时间：2017 年 6 月

针对 Github 项目问题的标签推荐方法研究

学　　生：王国鸿

指导教师：蒋　竞

摘　要

Github 是世界上最大的计算机项目托管平台，拥有超过 900 万开发者用户。在 Github 上，用户可以十分轻易地找到海量的开源代码，在每个开源项目中，都设置了问题模块，供开发者和浏览者进行问题的提出，讨论和解决。为了方便问题的审查分类，提高问题反馈的效率，Github 为问题提供了标签机制。支持开发人员给问题绑定标签，方便开发者的迅速识别，将问题最快地分到解决该类型问题的专业人员手中。

伴随着互联网的飞速发展，开源项目迅速庞大，Github 上的问题数量增加迅猛。我们设计一种自动推荐标签的方法 TRBF(Tag Recommendation Based on Features)，获取 Github 项目集的问题的属性特征，建立与标签集合的对应关系，将标签推荐的问题转化成文本多标签分类问题，优化 Github 网站上的问题标签分类。首先利用 Github 提供的 API 获取到每个项目所有问题的文本性内容，包括问题标题、问题描述内容、代码描述内容和作者名。TRBF 方法在得到这些文本内容后，提取文本特征、代码特征和作者特征，进行处理后得到优化的特征向量。针对每个项目，得到所有问题对应的特征向量后，利用后百分之九十问题对应的特征向量构造训练集，基于 MEKA 平台，选取分类算法实现基于特征的多标签分类器。将前百分之十问题对应的特征向量作为测试集，利用构建好的多标签分类器得到标签推荐结果，最后进行 TRBF 方法的评估。

本文的 TRBF 方法选取了 Github 上的 adobe、angular、baystation12、cocos2d 和 symfony 五个开源项目的 63666 个问题，在得到标签推荐结果后进行效果的评估，评估的指标有查全率、查准率和 F1-score。TRBF 方法的评估包括：验证提取的文本特征、代码特征和作者特征三者对应的处理方式带来的分类效果优化；验证 TRBF 采取的 J48 分类算法相比朴素贝叶斯和支持向量机的优越性；另外提出了一种基于特征相似度的标签推荐方法，验证本文 TRBF 方法的优越性。

关键词：标签推荐，多标签文本分类，Github

Research on Tag Recommendation Method for Github Project Problems

Author : WANG Guo-hong

Tutor : JIANG Jing

Abstract

Github is the world's largest hosting platform for computer projects, with more than 9 million developers. On Github, users can easily find huge amounts of open source code. In each open source project, problem modules are set up for developers and browsers to ask questions, discuss and solve them.

In order to facilitate the review of questions and to improve the efficiency of problem feedback, Github provides a label mechanism for the problem. Support developers to label problems, facilitate rapid identification of developers, and quickly divide the problem into the hands of professionals who solve this type of problem. The first step is using the API access provided by Github to get the text content of each project for all the problems, including the title, author name and description. After the TRBF method gets the text content, include text feature extraction, feature code and feature of author, then it start text preprocessing and TF-IDF conversion of text features, and then weighted by code characteristic and the feature, feature vector optimization. For each project, we get all the corresponding feature vectors, feature vectors constructed and corresponding problems after using ninety percent training sets, which is based on the MEKA platform, and then select the classification algorithm for multi label classifier based on text. The feature vectors corresponding to the first ten percent questions are used as the test set, and the constructed multi label classifier is used to obtain the tag recommendation results. Finally, the TRBF method is evaluated.

The TRBF method in this paper selected 63666 problems from five projects on Github,

include adobe, angular, baystation12, cocos2d and symphony. The indexes of assessment include recall, precision and F1-score. TRBF evaluation includes a better classification effect when we have weighted text feature extraction and code features and author features ; the TRBF's superiority compared to Naive Bayesian J48 classifier and support vector machine algorithm; we proposed a method based similarity tag recommendation method, TRBF method is also better.

Key words：Tag recommendation, Multi-label text categorization, Github

目录

图目录

表目录

1 绪论

1.1 课题研究背景与意义

全球最大的编程代码托管及开发交流网站 Github[1,2,3,4]，是基于 Git（一种最常见的开源分布式版本控制系统）的开源平台。截止 2015 年，Github 已经拥有了数千万个代码库。Github 网站针对每个开源项目，都提供问题模块，这个模块简而言之就是针对此项目的问题讨论模块，例如缺陷改进、功能完善等等。

在问题模块中，外部人员报告问题并由其提交修改意见，修改信息可能是添加代码也可以是文字描述。每个问题都记录在 Github 网站上，即使已经被关闭的问题，依旧会保存在 Github 网页上，方便人们查看。在每个开发人员新建问题时，往往都会给问题贴上标签，以此区分问题类型，方便不同项目内部人员审核。如图 1.1 所示 Github 问题模块通过标签来为问题建立分类，问题由问题标题和问题的描述内容组成。在描述内容中，作者可能会给出编程代码，因此可以将描述内容分成代码性描述内容和非代码性描述内容[5]。

图 1.1 问题标签示例图

由于项目数量的增长，以及问题数量的日益庞大,手动添加标签显得很麻烦。另外在代码审核过程中，不同的内部人员往往负责不同的审核内容，给问题分类能够最快的让问题分配到该处理此类问题的人员手上，从而大大地提高了审核效率。因此我们研究标签推荐的方法，实现问题的快速分类。

1.2 研究现状

1.2.1 问题和标签的研究

目前，国内外研究人员对 Github 上的问题和标签有了一定的研究[6]。在 Github 上的项目，拥有问题模块的往往存在时间更长，代码量更大，开发人数更多，另外网页开发项目相比其他项目拥有更多的问题。关于标签的研究，在 Github 上使用标签更容易解决问题，对于不同的项目，使用标签的方法不同，例如对于一些标签仅仅是一种处理项目优先级的方式，然而另外一些人在处于开发工作流时用标签来标志当时的进展。

1.2.2 标签推荐的研究

目前关于标签推荐的研究有一定成果，Xin Xia, David Lo 等人对 StackOverflow，SourceForge， FreeCode 三个网站上的标签进行了自动推荐研究，他们也是基于特征来进行分类的。后来 David Lo 和 Shaowei Hua 等人又对自动推荐标签进行了改进，算法更加全面和科学[7,8]。但是这些研究没有涉及到标题和内容的独立影响，即标题内容两个元素单独或组合考虑的模式。

当前在计算机领域中，已经实现了很多文本分类相关的算法，比如 Support Vector Machine（SVM）支持向量机模型，K-Nearest Neighbor（KNN）k 值邻近模型，Naive Bayes（NB）朴素贝叶斯模型等，这些研究对我的研究非常有帮助，参考借鉴的很多。但是这些方法在面对类别规模很大的文本的时候，精确度会急剧下降，这是本文需要重点解决的问题，如何处理大规模文本提高查全查准率和 F1-score。

1.2.3 优化文本分类的研究

传统的文本分类由于只考虑词频，在面对海量数据的时候效率往往很低，精确度也

不高。WeiLiu 等人提出了结合大数据处理平台 Hadoop 和中文文本分类，实现支持向量机 SVM 分类模型。传统方法中，SVM 是单机实现的，而在他们的研究里，基于平台实现的 SVM 并行化算法能够改善面对大数据文本的训练时间和训练效率，而且最后取得的训练器分类效果对比之前也有所优化[9]。

朴素贝叶斯 kNN 也是在文本分类中比较常见的分类算法，但是朴素贝叶斯的分类算法常常效率低下，因为在面对大数据文本分类的时候，维度的增长是指数型的，想要提高运行效率就必须进行降维度的处理。Zhengxuefeng，Yanpeng 等人从文本向量的稀疏性特点出发，对传统的 kNN 算法进行了诸多优化，在欧氏距离分类模型基础上进行了一个简化过程，使分类模型的效率大大提高，分类的准确性也得到了很大提高。

1.2.4 多标签分类的研究

多标签不同于一般的分类问题，每个待分类对象都可能对应一到多个分类属性，常见的分类器都是讲实例分到一个特定的分类属性中去。随着计算机网络资源的迅速增加，多标签分类的分类需求以及十分普遍，人们常常将其转化为单标签对应关系来处理。

在文本的多标签分类中，当文本数据非常大的时候，多标签分类的准确度将会下降明显，LiWen 等人实现了基于维基百科上丰富的语义数据实现的深度多标签分类，将文本的多标签分类不仅仅局限于词频特征上，在特征向量的形成和分类器的构造中考虑了语义，从而提高了大型文本多标签的分类的精确度。因此本文在多标签分类中也不是只考虑词频特征，在构建过程中额外考虑了作者特征，代码特征，优化了多标签分类器的分类效果。

1.3 课题研究目标、内容及实施方案

1.3.1 课题研究目标

由于 Github 问题模块并不能自动推荐标签，本文研究一种自动推荐标签的方法，帮助问题的快速分类。在 Github 的问题模块中，问题由问题的标题，问题的陈述内容，提出问题的作者等等内容组成。本文的课题研究目标就是对问题的组成内容进行分析和处理，将标签推荐转化成分类问题，构建分类器，得到标签自动推荐方法。根据上述研究

目标，本文需要解决的问题具体包括：

（1）如何获取 Github 项目中问题模块的内容。

（2）如何利用已经获取到的问题内容构建多标签分类器，得到标签推荐方法。

（3）如何评估标签推荐方法。

1.3.2 课题研究内容

本文实现的标签推荐方法，通过获取 Github 上部分项目已经存在的问题内容及其标签作为训练数据，提取特征后进行训练，得到分类器，另外一部分作为测试数据，得到标签推荐列表，查看效果并评估。在这个过程中，主要研究方面有：

1、研究影响标签推荐的重要因素

该研究是整个课题研究的基础，只有确定了影响标签推荐的重要因素，才能建立标签推荐算法。本文确定了三方面的属性：问题的标题、问题的陈述内容以及提出该问题的作者。

毫无疑问，问题的标题说明了基本的问题表述，问题的描述内容则是在此基础上进行的更为详尽的叙述，这是标签推荐最重要依据。在问题的描述内容中常常包括代码描述内容，将代码中的类名，方法名或函数名也作为影响标签推荐的重要因素。实际上在 Github 的问题提出中，同一位作者提出相同类型的问题可能性较大，因此将作者关键词也列为影响标签推荐的重要因素。

2、研究数据获取和特征提取

确定了影响标签推荐的重要因素后，我们需要利用程序获取到这些重要因素的信息。Github 提供了强大的 API，让我们可以很方便的获取到问题模块包含的所有内容，包括问题标题，作者名，问题的描述内容，问题对应的标签集合。

获取到这些文本信息后，实际上已经可以进行多标签分类器的构建，但是此时的分类器效果不是很好，需要进行优化。

本文提出的 TRBF 方法，分别提取文本特征，代码特征和作者特征。在特征向量的构建中，对文本特征进行文本预处理和 TF-IDF 转换（详细介绍在第二章），对作者特征和代码特征进行加权。作者特征就是作者名，而代码特征则需要进行额外的提取步骤。代码特征需要从问题描述内容中的代码部分提取，本文的 TRBF 方法利用正则匹配只提

取代码中的类名、方法名和函数名作为代码特征。

3、研究 TRBF 方法的多标签分类器构建

本文将标签推荐问题转化成文本多标签分类(multi-label classifier)问题，分类器构造基于 MEKA。多标签分类的对应关系没有任何限制，标签属性可以对应多个待分类属性，待分类属性也可以对应多个标签属性 [10]，也就是说多标签分类的含义正好可以契合本文中每个问题都可能有一到多个标签的特性。

TRBF 方法在数据获取和特征提取之后，得到特征向量以及分类器中需要的 ARFF 文件，形成训练集。得到训练集之后，选取 MEKA 分类算法 J48（分类算法的种类和选择原因将在后面介绍），最后得到多标签分类器和标签推荐列表。

4、研究 TRBF 方法的效果评估

将查全率，查准率和 F1-score 作为评估分类器准确性的指标。

本文的 TRBF 方法进行了文本特征处理、代码和作者特征加权，因此将 TRBF 分别和未进行文本特征处理、未进行代码特征加权、未进行作者特征加权的分类器进行效果对比，验证 TRBF 方法的优化性。

由于本文的 TRBF 方法选取 J48 分类算法，因此将其和选择 NBM（朴素贝叶斯），SVM（支持向量机）算法得到的多标签分类器进行比较，验证算法选择 J48 的优越性。

另外提出一种基于特征相似度的标签推荐方法，和本文的 TRBF 方法比较，验证 TRBF 方法的优越性。

1.4 论文的组织结构

论文一共分为五章，具体的组织结构如下：

第一章是绪论，主要介绍了本文的研究背景、研究内容与目标、课题实现方案及研究现状。

第二章主要介绍构建基于特征构造多标签分类器的方法，详细说明了问题模块需要关注的一些特征，提取这些特征的方法以及实现标签推荐方法的流程及相关技术要素。

第三章主要介绍基于第二章方法的实现过程，介绍了具体操作，包括如何利用文本来构造分类器需要的 ARFF 文件，选取算法并构建分类器等等，最后实现基于特征的标签推荐方法。

第四章则是利用分类器输出的测试集预测值与实际值的对比，对不同算法的效果进行查全查准率以及 F1-score 的评估，对本文的标签推荐方法的正确性进行验证。

第五章则是本篇论文实现的标签推荐方法的总结和展望，并提到了可以改进的地方和提升空间。

1.5 总结

本章主要介绍了本课题的研究背景与意义，阐述了研究现状，相关研究内容与目标。下一章主要介绍基于特征的标签推荐方法。

2 基于特征的 TRBF 标签推荐方法

2.1 概述

在 Github 的标签系统中，标签是根据提出的问题类型来进行划分的，也就是说标签推荐方法一定建立在问题上，那么在设计标签推荐方法的时候，就必须要弄清楚基于的问题具备哪些特征。

每个文本对应着问题的标题和相应的描述内容，我们将这些内容称为文本特征。提取到文本特征后，在对应的陈述内容中往往会包含代码陈述，本文的 TRBF 方法继续提取代码元素，将代码中的类名，方法名或者函数名提取出来，作为代码特征。另外在 Github 中，同一位作者提出具有相同标签的可能性较大，因此将作者名也作为一种特征加入到多标签分类器构造中，叫做作者特征。

提取到这些特征后，需要构建特征向量得到训练集。在构建特征向量时，TRBF 方法对上述三种特征分别采取了不同的处理方式，对文本特征进行了文本预处理和 TF-IDF 转换，对代码特征进行了×1 频度加权，对作者特征则进行了×2 频度加权处理（加权系数的确定见第四章）。在对这些特征进行处理后，可以得到选取的项目所有的问题对应的特征向量。TRBF 方法针对每个项目，对于该项目所有问题对应的特征向量，将前百分之十问题的特征向量形成测试集，后百分之九十问题的特征向量形成训练集，训练集作为构建多标签分类器的输入，测试集则作为分类器效果试验的输入。

本文的 TRBF 方法得到训练集和测试集后，基于 MEKA，输入训练集，选择多标签分类算法 J48（算法种类确定见第四章），构建基于特征的多标签分类器，实现了基于特征的标签推荐方法。

TRBF 额外评估了问题标题和问题描述内容的独立性影响：在得到的 TRBF 方法基础上，分别只考虑问题标题、只考虑问题描述和同时考虑上述两者，得到了三种不同的分类器效果，这三种考虑方式的 TRBF 方法评估在第四章中详细说明。

如图 2.1 所示，是本课题的基本流程图。

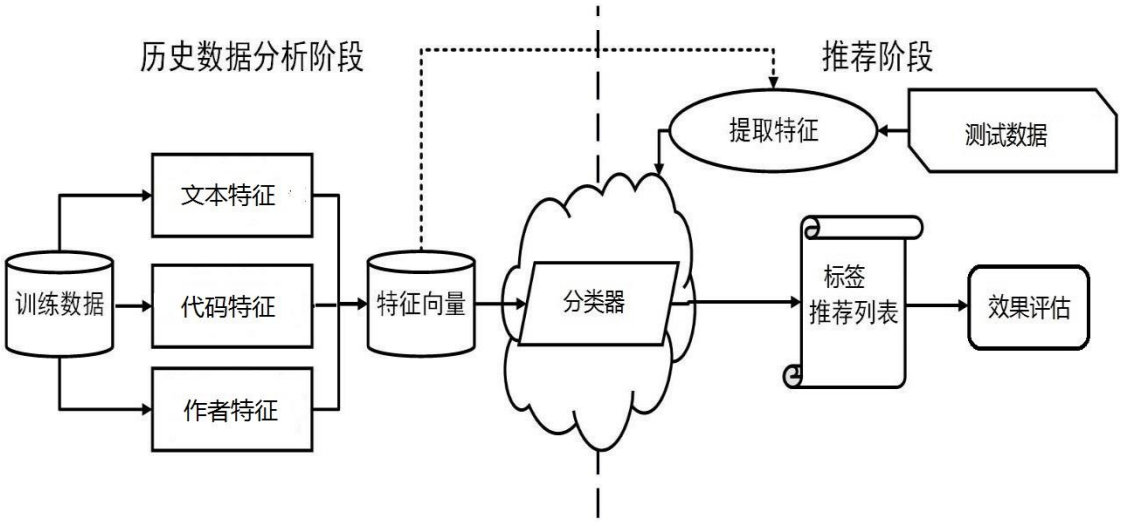


图 2.1 课题流程图

2.2 特征向量转化方法

在机器学习中，特征选取是个非常重要的过程，因为选取的特征有效性越高，分类效果也会更好。本文的 TRBF 方法选取了三个特征，文本特征（问题标题和描述内容）、代码特征（问题代码性描述内容中的类名、方法名或函数名）和作者特征（作者名）

2.2.1 文本特征

文本特征是最重要的特征，是构建训练集的主要依据。TRBF 方法将文本特征分成两个部分，问题的标题和问题的描述性内容。

1、文本预处理

从 Github 上爬取到的问题信息无效信息很多，这就需要我们对其进行预处理。对文本特征做分词、提取词干和过滤停用词的预处理。预处理可以得到问题的词库，基于词库，每个问题的文本数据可以转化为一个特征向量，向量中的元素表示词库中的关键词，n 维度对应对应单词在文本中的频数。

2、TF-IDF 转换

如果简单的将文本特征中的文本数据构造以词频为维度值的特征向量，那么在文本数据非常大的时候，分类的准确度将会下降明显。本文采取 TF-IDF 转化后的值代替仅

仅以频度作为特征向量取值的方法。所谓的 TF-IDF 实际上是 TF 和 IDF 的乘积，那么 TF 和 IDF 是什么呢?

TF(Term Frequency)，词频，简单理解就是词语的频率，频率就是针对当前的文本系统中，盖慈宇出现次数多少的一种评估标准。在某种类别中的词频越高，往往代表这个词越能起到区分的作用，在分类中就可以给予这个词更加多的权重[11]。

IDF(Inverse Document Frequency)，假定一个关键词在 N 个网页中出现过，那么 N 如果比较大，说明在所有的文档中，这个词语都是非常普遍的，不具有分类意义。也就是说当 N 比较大的时候，关键词的权重越小，那么 IDF 较大时，这个关键词具有较好的的类别区分能力，应当在特征向量的值中给予更多权重。

用计算公式计算 TF：

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \quad (2.1)$$

如公式所示，$n_{i,j}$是一种映射关系，表示某个词语 i 在文本 j 中出现的次数总和，分母则对应着所有词语 k 在文本 j 中出现的次数总和。

IDF 的计算公式：

$$idf_i = log\frac{|D|}{|\{j:t_i \in d_j\}|+1} \quad (2.2)$$

其中|D|为文件总数，分母$|\{j:t_i \in d_j\}|$表示文本集合d_j中包含词语t_i的文件数目，因为分母可能为 0，通常使用$|\{j:t_i \in d_j\}|+1$作为分母。然后再计算 TF 与 IDF 的乘积。那么 TF-IDF 则能很好的反应一个词语对一个文档的分类识别度。在本文中，利用 TF-IDF 转换来优化特征向量的取值，达到优化效果。

$$tfidf_{i,j} = tf_{i,j} \times idf_i \quad (2.3)$$

3、标题与描述内容的独立性特征选取

在标签推荐中，通常是将问题的标题和描述内容统一考虑，形成文本特征。在本文中，对这种方法提出了改进，将单独考虑标题，单独考虑问题的称述内容，同时考虑标题和陈述内容作为三种不同的文本特征选取对象，分别进行多标签分类器构造，评估效果后，选取最优。

2.2.2 代码特征

在 Github 的问题模块中，提问者的问题陈述往往都会包括编程代码，这些编程代码不能仅仅当做文本去对待，而是区分开来，尝试去提取代码中的较为关键的部分并进行加权。TRBF 方法将代码部分类名，函数名或方法名作为需要提取的代码特征。

1、编程语言的确定

在代码的特征提取的过程中，首先面对的问题是如何解决编程语言的多样性带来的算法涉及问题。本文的 TRBF 尝试针对常见的编程语言，得到集成提取算法。如表 2.1 是 Github 发布的 2016 年最受欢迎的编程语言排行榜，流行度的数字对应着在 Github 上使用该编程语言的项目数量，引自 https://octoverse.github.com。

表 2.1 编程语言流行度分布表

语言	Java	Python	Ruby	PHP	C++	CSS	C#
流行度	763783	744845	740620	478153	330259	231782	229985
语言	C	Go	Shell	OC	Scala	Swift	TypeScript
流行度	202295	188121	143071	75478	70216	62284	55587

实际的编程语言远远不止这些，本文的 BRTF 只会实现包括上述多种编程语言的一共二十种语言的编程识别。观察之后不难发现，这些编程语言都属于高级语言，由于编程语言之间的语法结构上都存在了一定的相似性，例如当我们想获取一段编程语言的类名作为关键元素时，我们会发现虽然编程语言不同，利用 class 关键词却可以匹配绝大多数的面向对象编程语言。

2、代码特征提取算法的编程思想

代码特征提取算法的编程思想是通过一些关键词，构建正则表达式，进而识别出想要获取的类名，函数名或者方法名。例如在提取 C++代码中的类名时，利用 class 构建正则表达式“class+string（string 要符合命名要求：由数字字母和下横线组成，且首字母不是数字）”则可以实现类名关键词的提取。那么针对每一种语言都可以建立起提取类名、方法名或者函数名的正则表达式，而且由于各种编程语言之间存在的相似性，正则表达式的构造也可能相同。

3、特征提取算法的实现

算法将对输入的任意文本扫描多次，每次扫描都将进入特定编程语言的识别正则表

达式，提取特定编程语言的关键元素。因为不能确定是哪种语言，只能遍历上述二十种编程语言，当属于其中的一种语言时，一定可以提取出对应的关键词，当不属于上述任何一种编程语言的时候，也可能会提取成功，因为这段代码对应的编程语言可能和上述二十种编程语言中的某一个编程语言有着相同的类名、方法名或函数名的定义和调用方式。如表 2.2 为二十种编程语言识别类需要的关键词。

表 2.2 编程语言类识别关键词表

编程语言类型	类识别关键词
C、 Perl、 CSS、 HTML、 VB、 Go、 Shell	无
JS	var
OC	@interface
C++、 Android、 PHP、 Python、 JAVA、 Ruby、 Swift、 C# 、Pascal、 Scala、 TypeScript	class

在方法或者函数的关键词提取上，本文只提取最常见的四种函数或方法定义和调用形式：

（1）string.methodname()

（2）methodname()

（3）string::methodname

（4）string->methodname()

2.2.3 作者特征

在 Github 上，问题模块上的作者往往只会专注其中的一类问题，因此在标签推荐方法中，尝试将作者名作为特征，并在特征向量的构造中，进行不同权重的加权，并对比权重比不同的时候，标签推荐的评估效果，确定是否优化及优化效果。

2.3 多标签分类器构造

2.3.1 多标签分类器构造概述

多标签分类器的构造分为两个步骤，第一步提取需要的特征，构建特征向量，形成

训练集，第二步选取分类算法，应用训练集，在 MEKA 中构造多标签分类器[12]。

2.3.2 将标签推荐问题转化成多标签分类问题

在 Github 的问题模块中，每个项目都会有一个标签集合，我们要实现的方法就是通过问题的属性，例如问题的题目，陈述内容及提出问题的作者等等建立起与标签集合中的元素对应关系。由于每个问题都可以被贴上不同的标签，因此这就完全可以转化成一种多标签分类问题，因为多标签分类中各个类别标签之间不是相互独立的，可以同时分类到同一个待分类属性，也就是说一个实例不仅仅会只属于一个类别，这是完全符合标签推荐的方法要求的。

在解决多标签分类问题的知识领域上，已经有了一些比较常见的解决方法。比较常见的解决方案大致有两种：问题转换和算法扩展。

问题转换常见的是将一个多标签问题拆分开来，针对每个标签都会生成推荐列表，从而分解成多个单标签问题。在实现标签推荐方法的时候，由于每个项目的所有标签集都是可获取的，因此将标签集分解成一个个单标签，然后针对每个问题，那么面对的问题就只有是否分类到这个标签中去，最终通过将所有单标签分类结果的集成则可以得到一个完整的多标签分类器。

算法扩展常见的有标签幂集分解策略，这种策略其实是一种非常简单的思考方式，将所有标签组合方式穷举，则得到所有可能的标签组合，涵盖了所有多标签组合的可能性，将每一种组合都当成一个类进行训练，那么我们就可以将一个多标签问题通过幂集分解转换成一个多类问题。但是在面对大数据文本分类的时候，随着标签数量的增加，标签的组合方式是指数型增长，导致分类的效率和准确度大大下降。由于这种基于算法扩展的多标签分类算法并没有改变数据集的结构，我们并不需要改变正常的分类步骤，虽然方便了很多，但是只能处理那些标签集的元素数目不多的问题。

2.3.3 分类器构造平台 MEKA 与分类数据集格式 ARFF

MEKA 是一款非常常见的机器学习软件，是 WEKA 的拓展版本，不仅拥有友好的 GUI 操作界面，而且提供了很多 Java 包，可以提供各种分类所需要的函数。本文要求的是一种多标签分类，而 MEKA 是一个专门用来解决多标签分类器的构造的软件，可以

利用 MEKA 提供的 GUI 直接进行多标签分类器的构造。

ARFF 文件的源头是 WEKA 默认的储存数据集文件，每个 ARFF 文件对应一个二维表格。整个 ARFF 文件可以分为两个部分，第一部分由关系声明和属性声明组成，第二部分由数据信息组成，即数据集中给出的数据。

这种文件的格式组成在 WEKA 和 MEKA 中有着些许不同。MEKA 的分类属性是写在所有属性的前面的，而且在关系说明中要添加待分类属性标签的个数。 本文的 TRBF 方法利用 WEKA 将特征向量转化成 ARFF 文件格式的测试集和训练集，然后在 MEKA 中进行多标签分类器的构建。

2.3.4 分类器的分类算法

通常的分类类型有二分类，多类分类和多标签分类，本文面向的问题为多标签分类，因此只介绍经常用于多标签分类的分类算法。

1、J48 决策树算法

J48 决策树算法是在机器学习领域非常常见的的算法，基本思想是树与分支的拓展决策生成 。在决策过程中，在树的内部结点进行属性值的比较并且在比较结果中决策向下的分支，从而形成决策树的一系列叶节点，。在生成的决策树中，根到叶结点的一条路径就对应着一条规则，在这条规则上对应着分类对象。那么整个决策树对应着一个分类器，实现了所有决策分支。J48 决策树的最大优点就是不需要学习很多知识，只需要知道分类对象和待分类对象的分类结果就可以训练并得到分类器模型，因此在机器学习中引用非常广泛。

2、SVM 支持向量机

支持向量机模型是一种二分模型，简单的说在某个维度上，找到一条线，可以将其区分开来，实现多标签属性之间的间隔最大化，并且每个被区分的部分都对应着一个类别。这在二维平面上是非常好处理的，但是在多维平面上的支持向量机就需要降低维度来处理，否则效率将会十分低下。将这种算法应用在文本分类中的可行性在文本数据小，标签类别少的情况下表现良好，但是一旦数据量急剧增加后，所带来的维度问题就会严重影响分类的效率。但是 SVM 是一种凸优化解，对于其他获取局部最优解的算法例如基于规则的分类器都较为良好。

3、朴素贝叶斯

朴素贝叶斯分类器(Naive Bayes Classifier,或 NBC)也是一种非常常见的分类器，它将理论建立在古典数学上，因此算法的实现非常具有可靠性，并且有着很稳定的分类效果。这种分类器相较于其他分类器，在实际的运算上很为简便，因为所需估计的参数相对很少，对数据的完备性要求不高。在分析过程中我们发现，如果建立朴素贝叶斯模型，理论上相比其他分类器具有较小的误差值，但是这是建立在分类属性之间互相独立的前提下。在实际的分类中，分类属性多，且每种分类属性往往之间存在着关联，在这种情况下，朴素贝叶斯分类器的准确性是有待商榷的。

上述三种算法是最常见的分类算法，本文分别针对每一种算法都单独构建了一个多标签分类器，分别应用了选取的五个项目，相互比较分类器的分类效果，最后的 TRBF 方法中选取了 J48 作为最优的分类算法（算法选取见第五章）。

2.4 基于特征的 TRBF 标签推荐方法

基于特征的标签推荐方法 TRBF 首先将获取到 Github 上项目问题信息分为三个特征，文本特征、代码特征和作者特征。确立特征之后对文本特征进行了优化处理，对代码特征和作者特征进行了加权处理，形成特征向量，然后构建以 ARFF 文件格式表示的训练集和测试集。最后选取不同的分类算法，在 MEKA 中构建多标签分类器，得到标签推荐结果，最后根据评估分类效果来确定最终选择的算法类型和加权系数等等。

3 标签推荐方法的实现

3.1 数据的获取分析及优化

3.1.1 API 简介

Github 网站上的数据量巨大，涉及全球范围内的代码托管和编程交流，能给很多研究提供一个非常有用的数据信息。我们可以通过调用 Github 提供的 API，方便的获取 Github 上的大部分数据。

Github 的 API 功能很强悍，基本上可以利用它获取所有在 Github 上的信息，这些返回信息通常是以网页字符串的形式返回，而且 API 使用很简单，在代码中基本都是以 GET 方式从 https://api.github.com/ 取得，在 Python 中，我们使用 curl 获取页面内容，但在 github 的 API 中，这种方法必须要提前获取权限，只要是网站的注册用户即可拥有该权限，我们只需要在 Github 网页中生成一个 Token 字符串。Token 字符串信息在 Github 上输入账号密码后，可以在 Github 上生成，是一种必要的权限验证信息。

在 https://github.com 登录帐号后，找到头像右上角的 Settings，点击 Developer settings 下的 Personal access tokens，当完成上述步骤之后，再在网页中点击 https://github.com/settings/tokens/new 网址，进入如图 3.1 所示界面。

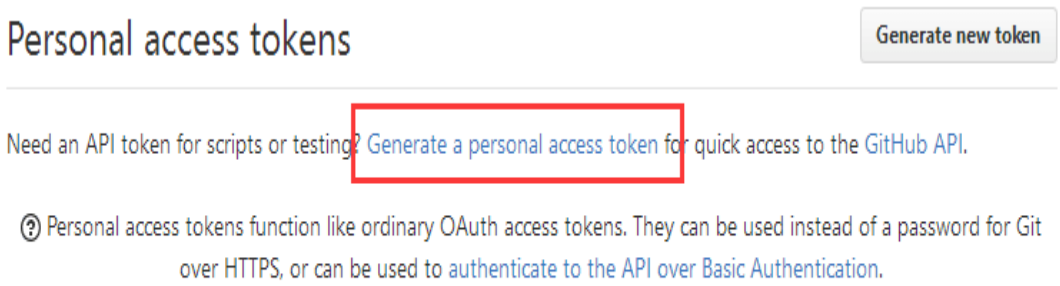


图 3.1 Token 生成界面

如图 3.2 是 Token 要求的权限，全部勾选之后再点击生成 Token，然后就可以得到权限字符串，在代码中直接复制粘贴到 Header 中即可顺利获取 Github API 网页的返回内容。

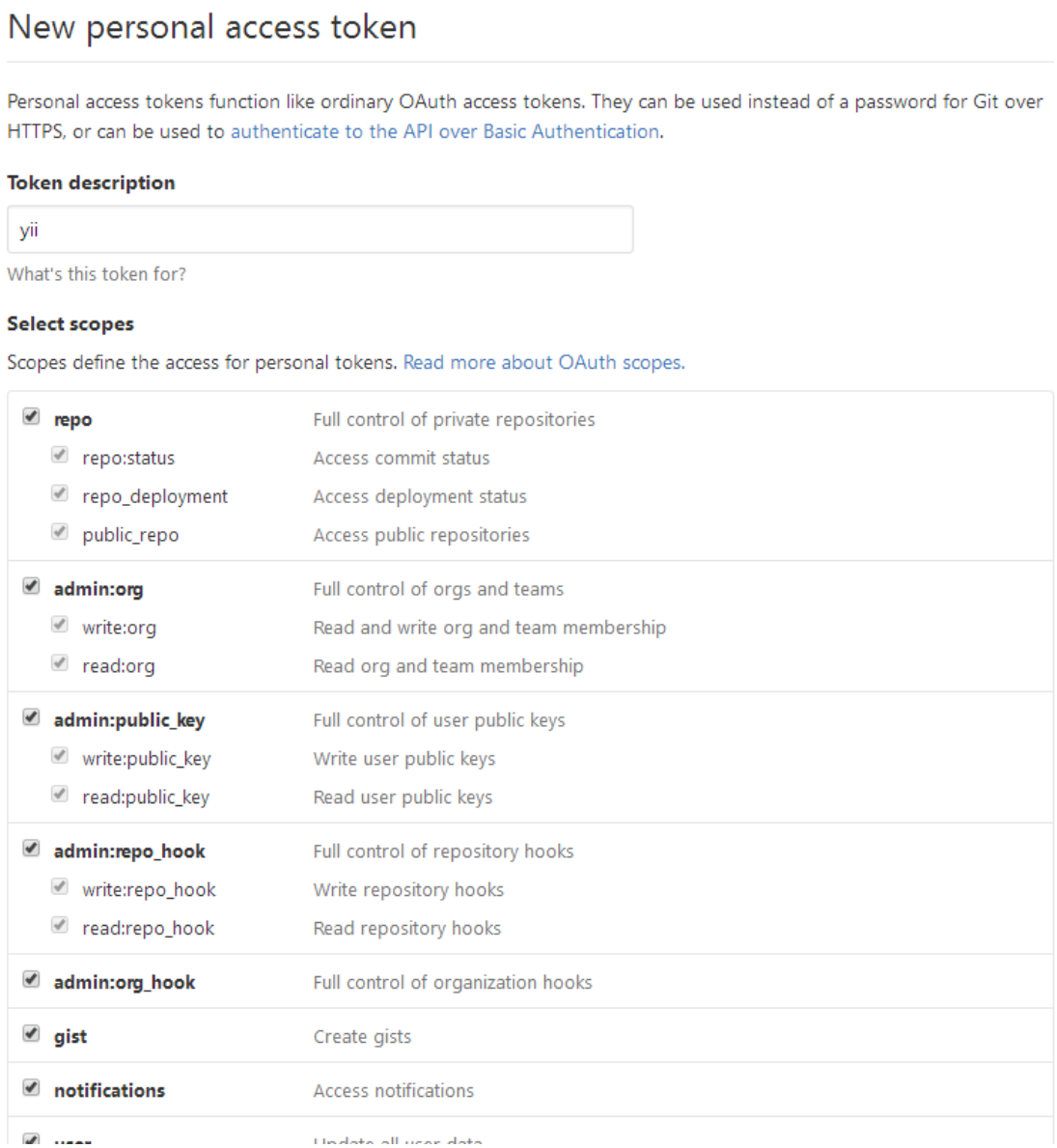


图 3.2 Token 权限图

3.1.2 Github 项目集的选取

选取五个符合问题数量大于 100 并且经常使用标签的项目进行研究，利用 github 提供的 API 获取了这些项目所有的问题名称，内容和所标注的标签以及作者名。在后续的

测试集选取中，针对每个项目按照时间排序的前百分之十问题作为测试集，后百分之九十作为训练集。

表 3.1 项目信息表

项目名	问题总数量	标签总数量	标签类别
adobe_brackets	12808	9001	102
angular_angular	11986	8441	73
Baystation12_Baystation12	14328	3445	32
cocos2d_cocos2d-x	16626	2241	83
symfony_symfony	7918	7379	72

3.1.3 Github 项目集数据爬取程序

本文使用 Python 编程语言爬取所需要的 Github 返回信息。在编写爬虫程序的时候，我们需要 Github API 提供的一些包方法。我们可以在 Python 程序中调用包工具，然后直接调用其中的函数就能实现获取 Github 网站上的大部分内容，这些内容以字符串的形式返回。例如爬取 rails/rails 项目，如下代码：

```
import sys
import requests
import os
import github
import json
import re                                                    //需要的类包
Headers={
    "Authorization": "token 7a98a5981e851c4ad2320f8101c779c4f23d2539"
}                                                    //权限，token 可以在网站上生成
url = "https://api.github.com/repos/rails/rails/issues”      //API 返回信息的网页
js_url = requests.get (url,headers = Headers).text      //以文本形式获取返回信息
```

首选我们需要了解在 Github API 返回的网站格式，在 Github API 中返回的 URL 格式如下（url = https://api.github.com/repos/%s/%s/issues?state=all&page=%d&per_page=%d），其中的缺省字符串%s 为项目名称。要想获取返回信息，需要调用 request.get(url)，则可以收到该项目每个问题的信息，包括问题标题，问题陈述内容，作者等内容。如图 3.3 为截取的部分 API 返回的信息。在返回的 API 信息中，大部分都是我们不需要的，因此我们编写正则表达式只提取问题的标题“title”、问题的陈述内容“body”、标签“labels”和作者“login”。


```
[
  {
    "url": "https://api.github.com/repos/adobe/brackets/issues/13375",
    "repository_url": "https://api.github.com/repos/adobe/brackets",
    "labels_url": "https://api.github.com/repos/adobe/brackets/issues/13375/labels{/name}",
    "comments_url": "https://api.github.com/repos/adobe/brackets/issues/13375/comments",
    "events_url": "https://api.github.com/repos/adobe/brackets/issues/13375/events",
    "html_url": "https://github.com/adobe/brackets/issues/13375",
    "id": 228643134,
    "number": 13375,
    "title": "built in HTML and CSS validation",
    "user": {
      "login": "buimf",
      "id": 16083584,
      "avatar_url": "https://avatars2.githubusercontent.com/u/16083584?v=3",
      "gravatar_id": "",
      "url": "https://api.github.com/users/buimf",
      "html_url": "https://github.com/buimf",
      "followers_url": "https://api.github.com/users/buimf/followers",
      "following_url": "https://api.github.com/users/buimf/following{/other_user}",
      "gists_url": "https://api.github.com/users/buimf/gists{/gist_id}",
      "starred_url": "https://api.github.com/users/buimf/starred{/owner}{/repo}",
      "subscriptions_url": "https://api.github.com/users/buimf/subscriptions",
      "organizations_url": "https://api.github.com/users/buimf/orgs",
      "repos_url": "https://api.github.com/users/buimf/repos",
      "events_url": "https://api.github.com/users/buimf/events{/privacy}",
      "received_events_url": "https://api.github.com/users/buimf/received_events",
      "type": "User",
      "site_admin": false
    },
    "labels": [
      {
        "id": 246039886,
        "url": "https://api.github.com/repos/adobe/brackets/labels/Extension%20Available",
        "name": "Extension Available",
        "color": "009800",
        "default": false
      }
    ]
```


图 3.3 API 返回信息部分截图

3.2 特征的提取和处理

3.2.1 文本特征的提取和处理

在获取到 API 返回信息后，编写提取程序，只提取其中问题的标题和描述内容，我们发现不论是标题还是内容都包含了较多无用的信息，这对多标签分类起着负面作用，因此需要进行优化处理。本文的 TRBF 方法基于 Python 的 nltk 函数库编写程序实现对文本的分词，去除停用词和提取词干预处理[13]。另外 TRBF 方法考虑一个词语的区分性，优化特征向量中向量的大小简单对应着词语频度，将词语频度转化成每个单词对应的 TF-IDF 大小。

1、Python 中的 NLTK 库

NLTK 是个非常强大的自然语言处理库，在自然语言的领域中，引用非常广泛。在已经搭建好 Python 环境和包工具 PIP 的前提下，在 cmd 输入“pip install nltk”安装即可。

2、算法模型和停用词表的下载

编写 Python 程序，调用 nltk.download()下载波特词干提取算法模型和英文停用词表，如 3.4 所示：

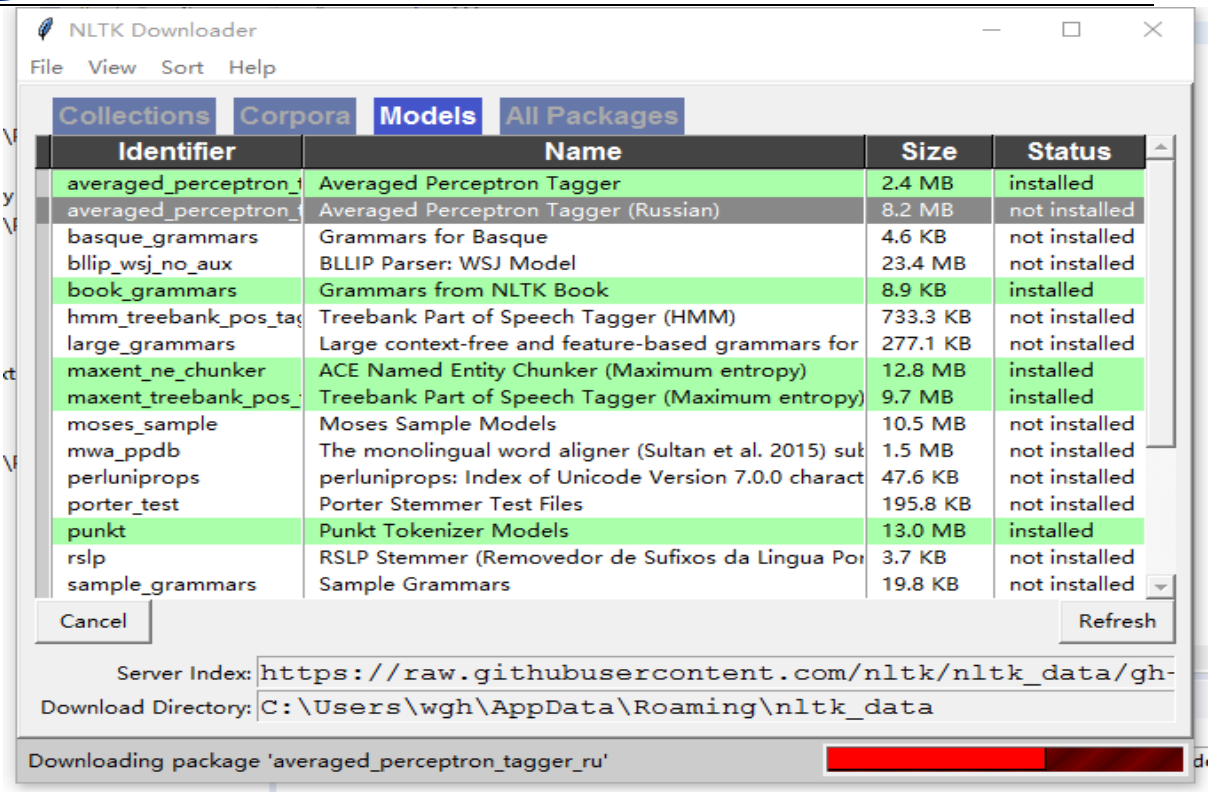


图 3.4 词干模型及各类文本处理包

3、NLTK 库函数的调用

调用下列函数得到分词结果，词干输出和停用词的去除。

表 3.2 调用的 NLTK 库函数

函数名	实现功能
nltk.word_tokenize()	分词算法调用
nltk.PorterStemmer()	波特词干分析算法调用
stopwords.words('english')	调用英文停用词

4、TF-IDF 转换

利用 WEKA 提供的 TF-IDF 转换工具，将上述预处理后的文本添加到对应 ARFF 文件的@attribute document string 位置中，构建 ARFF 文件，通过 WEKA 打开 ARFF 文件，然后在 TF-IDF 处理器选项设置中将 TF-transfer 和 IDF-transfer 设置为 true 就可以直接将频度转换成 TF-IDF 了。

3.2.2 代码特征的提取和加权

代码特征只会存在在问题代码性描述内容中，利用基于正则匹配的集成提取算法实

现代码中类名、方法名或者函数名的提取。

提取到这些关键词后，在文本特征中加权该关键词的频度。TRBF 方法尝试了不同加权系数，选择了从 0 到 4 五个不同的加权系数，最后确定当加权系数为 1 的时候，得到的多标签分类器效果最好（详见第四章）。

3.2.3 作者特征的提取和加权

将 API 返回的作者名作为作者特征，和代码特征相同，在文本特征中加权该关键词的频度。TRBF 方法尝试了不同加权系数，选择了从 0 到 4 五个不同的加权系数，最后确定当加权系数为 2 的时候，得到的多标签分类器效果最好（详见第四章）

3.3 多标签分类器构造

3.3.1 ARFF 文件格式的特征向量构造

在进行完特征提取和特征处理后，接下来需要构造以 ARFF 文件表示的特征向量，形成训练集和测试集，从而在 MEKA 中得到多标签分类器。

在生成标准的 ARFF 文件之前，我们需要先在记事本上构造最基本的以字符串作为基本属性，每个标签作为分类属性的 ARFF，构造的 ARFF 文件格式如下：

```
@relation trainningSet
@attribute label1{yes,no}
@attribute label2{yes,no}
@attribute label3{yes,no}
……
@attribute labeln{yes,no}
@attribute document string
@data
Yes,no,….,’string1 string2 ……’
No,yes,…..’string3 string4….’
……
```

在上述的格式中，前 n 个属性分别对应该项目的标签集合，最后的 document 是一个字符串类型的属性，对应已经实现文本预处理、TF-IDF 转换、代码和作者特征加权的字符串组合成的文本。

TF-IDF 转化后的 ARFF 关系名称由预处理前的关系名加上预处理的过滤器名称及其选项组成，并且字符串型属性转化为数值型的矩阵，产生的文件是压缩格式的 ARFF，也就是只显示不为 0 的属性值。由于 WEKA 中的 TF-IDF 转换得到的 ARFF 文件中的关系声明没有待分类标签的个数，因此需要将@relation 关系名称改为包含待分类属性个数的关系声明，在末尾加入-C: N 即可，N 为待分类属性的个数。

完成上述构造后，需要将每个项目对应的 ARFF 文件分割成训练集和测试集，本文的 TRBF 方法将前百分之十问题对应的特征向量作为测试集，后百分之九十问题对应的特征向量作为训练集。

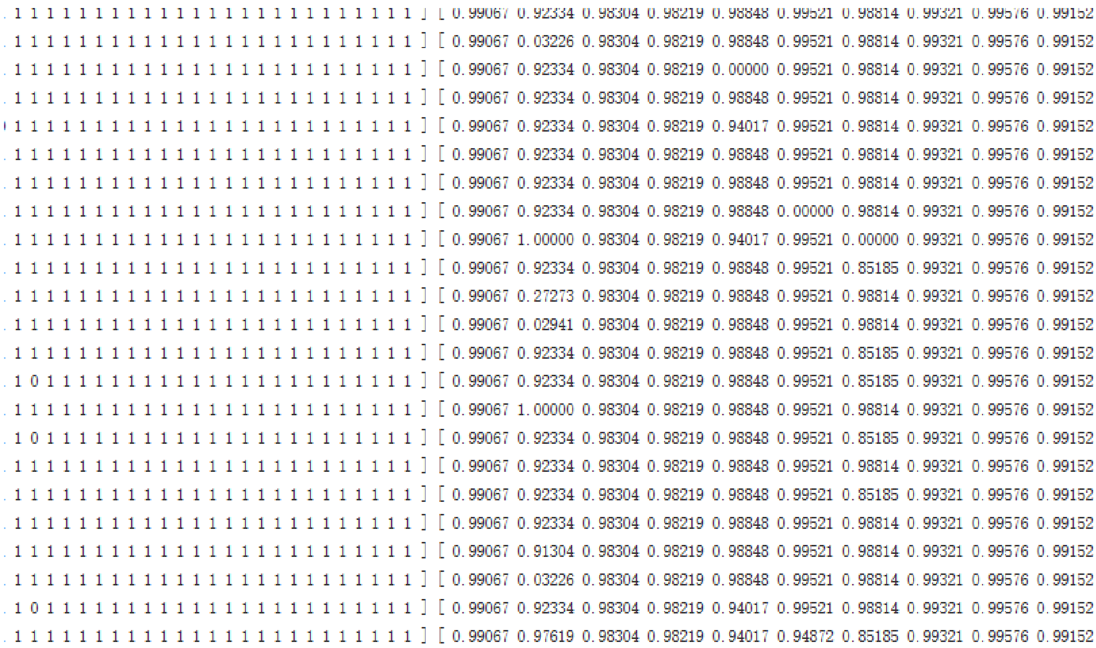
3.3.2 分类器的分类算法

本文的 TRBF 方法在得到了以 ARFF 文件格式表示的训练集和测试集之后，在构建多标签分类器之前需要选取分类算法。TRBF 分别选取了朴素贝叶斯算法，SVM 支持向量机和 J48 分类算法，对于每个项目，分别得到了三种不同的标签推荐结果，评估比较后选定 J48 作为 TRBF 的最终分类算法（算法选取详见第四章）。

3.3.3 分类器的训练和结果输出

打开 MEKA GUI，打开训练集 ARFF 文件，点击 classify，选取好分类算法后应用测试集 ARFF 文件得到标签推荐概率。注意在分类器输出选项第三个参数要改成 10，否则将不会输出标签推荐的每个实例结果。

在最后的标签推荐结果中，输出的不是推荐概率，相反是不推荐概率，如图 3.5 所示。



```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.03226 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.00000 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.94017 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.00000 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 1.00000 0.98304 0.98219 0.94017 0.99521 0.00000 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.85185 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.27273 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.02941 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.85185 0.99321 0.99576 0.99152
1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.85185 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 1.00000 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.85185 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.85185 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.91304 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.03226 0.98304 0.98219 0.98848 0.99521 0.98814 0.99321 0.99576 0.99152
1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.92334 0.98304 0.98219 0.94017 0.99521 0.98814 0.99321 0.99576 0.99152
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ] [ 0.99067 0.97619 0.98304 0.98219 0.94017 0.94872 0.85185 0.99321 0.99576 0.99152
```

图 3.5 结果输出部分截图

3.3.3 分类器结果输出分析与百分比分割方法

每个问题对应一行，每个待分类标签对应一列。前半部分为实际的标签情况，后半部分为预测情况。 0 代表该问题有对应的标签，1 则表示无。后半部分为分类器不推荐该标签的概率。

举个例子来说，对于前半部分第一列的数值，第 n 行为 0 则代表第 n 个问题拥有标签 1，对于后半部分第一列的数值，第 n 行对应着一个概率值 P，则代表分类器推荐第 n 个问题拥有标签 1 的概率为（1-P）。

在认定是否推荐该标签的时候，一般大于 0.5 视作无，反之视作有该标签。本文的 TRBF 方法将将改变 0.5 这个临界值，TRBF 先将每列的非推荐概率排序，这样就可以得到针对每个标签所有问题推荐到这个标签的概率，那么排序越靠前说明越可能推荐。那么是否可以将排序靠前的百分之一或百分之十作为认定推荐的结果呢？

于是 TRBF 方法进行了百分比分割，从百分之一到百分之百共 100 个分割值，认定排序在百分比之前的为拥有该标签，反之则无。这样每个项目都会得到一百个对应的查全查准率和 F1-score， 查全率越高，查准率越低，而 F1-score 存在着一个极大值（详见第四章），这时候的标签推荐方法最准确。

3.4 基于特征的 TRBF 标签推荐方法实现

对于基于特征的 TRBF 标签推荐方法，首先爬取数据得到文本特征、代码特征和作者特征，并在这三个特征基础上，对文本特征进行文本预处理和 TF-IDF 转换，对代码关键词进行了×1 加权，对作者关键词进行了×2 加权，得到了以 ARFF 文件格式表示的特征向量，选取 J48 算法，在 MEKA 中得到多标签分类器并输出推荐非预测的概率表。然后利用百分比分割，可以得到 F1-score 最大的时候对应的百分比，在百分比基础上得到基于特征的 TRBF 标签推荐方法，TRBF 方法中的加权系数和百分比分割值的确定以及算法选取依据将在第四章详细给出。

4 基于特征的 TRBF 标签推荐方法验证

4.1 概述

在评估验证过程中，引用了三个指标，查全率，查准率以及两者的调和平均数 F1-score，一般将 F1-score 作为最好的评估指标，因为在实际的标签推荐分类中，查全率和查准率往往都很重要。主要验证方面包括文本特征处理后优化性的验证，TRBF 方法中关键词加权系数的优越性验证，标题和描述内容的独立性推荐效果验证，算法 J48 优越性的验证以及和一种基于特征相似度的标签推荐方法的比较评估验证本文的 TRBF 更为优越。

4.2 评估指标及其计算过程

TRBF 在获取到 MEKA 输出的预测结果后，进行纵向分析，将预测的数值排序，越大表示越不被推荐，分别取百分之一到百分之百的临界点概率作为判断是否推荐的依据，大于则不推荐，小于则推荐，由此可以得到查全查准率和 F1-score 的值，当 F1-score 最大时，对应的百分比分割值作为选择的最佳临界点概率。

查准率 precision：推荐集合中被正确推荐的比例。T 代表测试集标签集合，P 代表分类器预测标签集合。

$$Precision = \frac{|T \cap P|}{|P|} \quad (4.1)$$

查全率 recall：正确推荐的标签个数占整个数据集中真正正确个数的比例。

$$\text{Recall} = \frac{|T \cap P|}{|T|} \quad (4.2)$$

F1-score：查全率和查准率的调和平均数。

$$\text{F1} - \text{score} = \frac{2 \times Recall \times Precision}{Recall + precision} \quad (4.3)$$

F1 分数是模型准确率和召回率的一种加权平均，众所周知加权平均数是一种同时考虑的效果指标，也就是说在这个模型中，查全率和查准率的重要性是一致的，F1-score

的最大值是 1，最小值是 0，而且值越大越好。

通常在比较中，查全查准率是两个比较重要的指标，但是当两种效果对比的时候，如果其中一个的查全率比另外一个高而查准率却较低，这时候使用 F1-score 则是最正确的评估方法，因为 F1-score 的意义是上述两个指标同时考虑的评估，一旦着重了那个方面而忽略另一个方面都会使得 F1-score 的值变小。

4.3 百分比分割值的确定

针对选取的五个项目，多标签分类器得到五个项目的标签推荐概率，从而得到百分比分割值对应的 F1-score 评估表（省略 20%以后，因为后续的 F1-score 一直降低）：

表 4.1 百分比分割评估表

百分比	adobe F1-score	angular F1-score	baystation12 F1-score	cocos2d F1-score	symfony F1-score
1%	0.0589	0.2736	0.2863	0.3152	0.4202
2%	0.1281	0.2755	0.2833	0.3152	0.4206
3%	0.1481	0.2661	0.2812	0.3852	0.4265
4%	0.1788	0.2514	0.2893	0.3596	0.4274
5%	0.2119	0.2512	0.2471	0.3601	0.4298
6%	0.2195	0.2414	0.2867	0.3584	0.4334
7%	0.2202	0.2431	0.2862	0.3541	0.4351
8%	0.2369	0.2405	0.2857	0.3516	0.4363
9%	0.2372	0.2396	0.2690	0.3363	0.4343
10%	0.2266	0.2284	0.2578	0.3100	0.4348
11%	0.2250	0.2284	0.2576	0.3088	0.4293
12%	0.2219	0.2283	0.2565	0.3046	0.4262
13%	0.2210	0.2261	0.2551	0.3025	0.4253
14%	0.2205	0.2246	0.2508	0.3003	0.4203
15%	0.2189	0.2242	0.2508	0.2955	0.4176
16%	0.2160	0.2132	0.2507	0.2936	0.4086
17%	0.2144	0.2130	0.2498	0.2900	0.4111
18%	0.2128	0.2141	0.2500	0.2866	0.4075
19%	0.2128	0.2131	0.2501	0.2870	0.4073
20%	0.2134	0.2126	0.2499	0.2845	0. 4029

如表格 4.1 所示，adobe、angular、baystation12、cocos2d、symfony 五个项目对应的

效果最佳的百分比分割值分别为 9%、2%、4%、3%、8%，因为此时对应的 F1-score 的值为极大值点。

4.4 文本特征优化效果检验

文本特征的处理主要包括文本预处理（分词，提取词干和去除停用词）和 TF-IDF 转换（将词语频度替换成 TF-IDF 值）。文本预处理可以去除很多无用信息，TF-IDF 转化则不仅仅考虑词语的频度，在频度的基础上考虑词频和逆向文件频率，将词语的重要性和可区分度加入到特征值中。

表 4.2 加权方法效果对比表

项目	加权方式	查全率	查准率	F1-score
adobe	不做任何预处理	9.713%	14.947%	0.1177
	文本特征处理	12.535%	17.110%	0.1447
angular	不做任何预处理	34.390%	23.939%	0.2823
	文本特征处理	28.517%	32.189%	0.3024
baystation12	不做任何预处理	18.776%	39.381%	0.2543
	文本特征处理	33.636%	29.759%	0.3158
cocos2d	不做任何预处理	30.952%	17.931%	0.2271
	文本特征处理	39.157%	22.337%	0.2845
symfony	不做任何预处理	30.145%	49.524%	0.3748
	文本特征处理	35.410%	50.410%	0.4160

如表格 4.2 所示，在 adobe、cocos2d 以及 symfony 项目中，文本特征处理既可以提高查全率也可以提高查准率和 F1-score。在 angular 项目中，在文本特征处理降低了查全率却提高了查准率，但是 F1-score 较大，说明文本特征处理依旧能起到较好的优化效果。在 baystation12 项目中，文本特征处理提高了查全率却降低了查准率，但是 F1-score 依旧在增加，因此认定文本特征处理可以起到优化效果。

4.5 代码特征和作者特征加权检验

代码特征加权：在问题的陈述内容中，代码占了很重要的部分，通常代码中的类名方法名或者函数名都是一种很重要的特征，因此本章检验代码关键词加权对于分类器分类效果的优化效果以及最佳的加权系数。

作者特征加权：在 Github 上，相同的作者往往会提出一个类型的问题，因此将作者关键词加入到特征向量中，检验是否能给多标签分类器带来优化效果以及最佳的加权系数。

在获取了代码关键词和作者特征之后，我们尝试对其进行加权，既然是加权，就得确定加权的系数。本文针对选取的五个项目，分别将作者关键词和代码关键词计算了 0 到 4 加权系数方案的实际效果。

如表格 4.3 是代码关键词在不同加权权重下得到的查全查准和 F1-score，在 adobe 项目中代码的加权系数越高，查全率先增后减，查准率越低，但是 F1-score 在代码关键词×1 的时候存在一个极大值。在 angular 项目中代码的加权系数越高，查全率越低，查准率越高，但是 F1-score 在代码关键词×1 的时候存在一个极大值。在 baystation12 项目中代码的加权系数越高，查全率越高，查准率越低，但是 F1-score 在代码关键词×1 的时候存在一个极大值。在 cocos2d 和 symfony 项目中代码的加权系数越高，查全率先增后减，查准率先增后减，但是 F1-score 在代码关键词×1 的时候存在一个极大值。

表 4.3 代码加权系数对比表

项目	代码加权系数	查全率	查准率	F1-score
adobe	0	9.713%	14.947%	0.1177
	1	14.621%	10.759%	0.1241
	2	13.851%	9.364%	0.1117
	3	12.937%	8.841%	0.1050
	4	11.824%	8.389%	0.0981
angular	0	34.390%	23.939%	0.2823
	1	37.500%	25.206%	0.3015
	2	36.485%	24.813%	0.2953

	3	35.811%	23.872%	0.2867
	4	33.582%	21.044%	0.2587
baystation12	0	18.776%	39.381%	0.2543
	1	23.238%	31.338%	0.2669
	2	22.893%	29.751%	0.2587
	3	21.852%	27.893%	0.2450
	4	19.608%	22.458%	0.2093
cocos2d	0	30.952%	17.931%	0.2271
	1	26.190%	23.404%	0.2472
	2	24.892%	22.789%	0.2379
	3	22.815%	21.552%	0.2216
	4	19.525%	18.715%	0.1911
symfony	0	30.145%	49.524%	0.3748
	1	30.500%	55.882%	0.3946
	2	29.485%	53.747%	0.3807
	3	27.712%	51.058%	0.3592
	4	25.182%	49.817%	0.3345

如表格 4.4 是作者关键词在不同加权权重下得到的查全查准和 F1-score，在 adobe 和 baystation12 项目中作者的加权系数越高查全率越高，查准率越低，但是 F1-score 在作者关键词×2 的时候存在一个极大值。在 angular 项目中，作者的加权系数越高，查全率越低，查准率越高，同样在作者关键词×2 的时候，F1-score 取得一个极大值。在 cocos2d 和 symfony 项目中，F1-score 同样在作者关键词×2 的时候，F1-score 取得一个极大值，因此当作者关键词×2 的时候为效果最好的加权方案。

表 4.4 作者加权系数对比表

项目	作者加权系数	查全率	查准率	F1-score
adobe	0	9.713%	14.947%	0.1177
	1	11.829%	12.684%	0.1224
	2	14.733%	11.233%	0.1275
	3	15.813%	9.664%	0.1199
	4	16.227%	9.018%	0.1159
angular	0	34.390%	23.939%	0.2823
	1	32.179%	25.913%	0.2870
	2	29.724%	29.385%	0.3015
	3	27.334%	31.008%	0.2905
	4	26.853%	31.296%	0.2890
baystation12	0	18.776%	39.381%	0.2543
	1	21.239%	32.658%	0.2573
	2	25.602%	27.311%	0.2644
	3	27.124%	24.338%	0.2565
	4	27.358%	21.957%	0.2436
cocos2d	0	30.952%	17.931%	0.2271
	1	31.094%	19.020%	0.2360
	2	31.928%	20.000%	0.2459
	3	32.086%	19.657%	0.2437
	4	31.025%	18.367%	0.2307
symfony	0	30.145%	49.524%	0.3748
	1	31.356%	50.927%	0.3881
	2	33.822%	53.313%	0.4139
	3	30.854%	51.927%	0.3870
	4	29.287%	49.128%	0.3669

从上述结果可以看出对所有的五个项目，当作者关键词×2，代码关键词×1 加权的时候，能取得的分类效果最好。

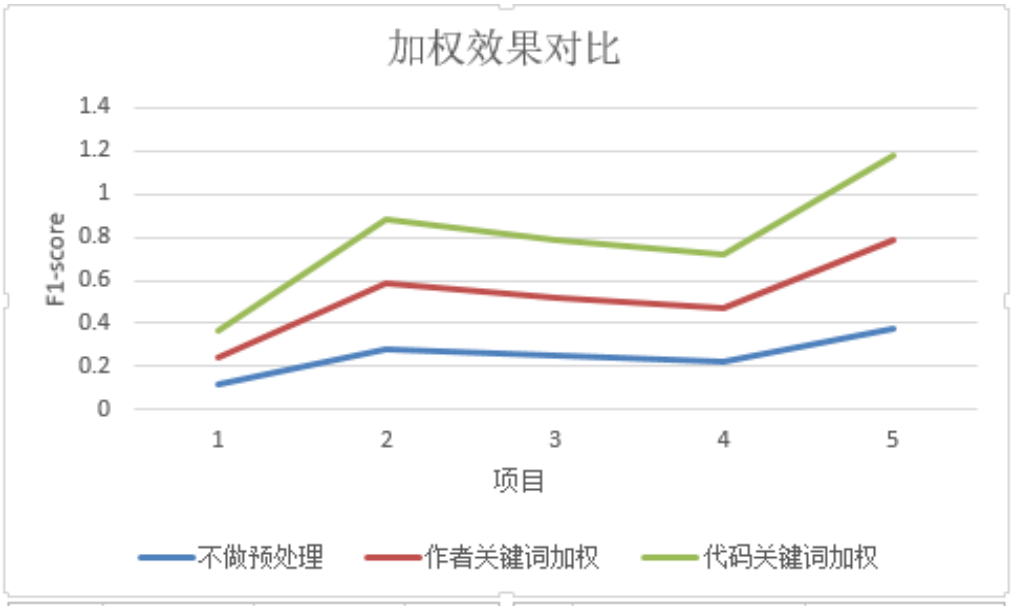


图 4.1 加权效果对比图

如图 4.1 针对选取的五个项目，得到作者关键词加权，代码关键词加权和不做预处理的加权效果对比图。横坐标对应五个项目，纵坐标对应 F1-score 的大小。在五个项目中，作者关键词加权和代码关键词加权都可以提高分类器的分类效果，而且代码关键词加权提升的效果更加明显。

4.6 标题与描述内容的独立性推荐效果检验

在常见的文本分类中，会将问题的标题和陈述内容看成一个整体，本课题尝试将标题和陈述内容分割开来，单独构建多标签分类器，并评估对比标签推荐效果。如下所示，无论是单独考虑标题还是单独考虑问题陈述内容都会降低查全查准率，因此在本文的标签推荐方法中，将标题和问题陈述内容看成一个整体。

表 4.5 独立性检验效果对比表

项目	加权方式	查全率	查准率	F1score
adobe	同时考虑	9.713%	14.947%	0.1177
	只考虑标题	7.512%	13.389%	0.0962

	只考虑陈述内容	13.666%	8.050%	0.1013
angular	同时考虑	34.390%	23.939%	0.2823
	只考虑标题	2.065%	11.765%	0.0351
	只考虑描述内容	32.108%	22.783%	0.2665
baystation12	同时考虑	18.776%	39.381%	0.2543
	只考虑标题	4.393%	21.250%	0.0728
	只考虑描述内容	25.155%	23.276%	0.2418
cocos2d	同时考虑	30.952%	17.931%	0.2271
	只考虑标题	4.000%	12.676%	0.0400
	只考虑描述内容	28.952%	15.931%	0.2055
symfony	同时考虑	30.145%	49.524%	0.3748
	只考虑标题	14.474%	57.143%	0.2310
	只考虑描述内容	28.145%	49.511%	0.3588

通过上述表格，可以得到五个项目在三种情况下分别对应的 F1-score，作出如图 4.2 所示的独立性效果检验图。无论是只考虑标题还是只考虑描述内容，所得到的多标签分类效果都比同时考虑问题标题和描述内容差，而且只考虑标题的效果更差。验证了本文 TRBF 方法同时考虑问题标题和问题描述的优越性。

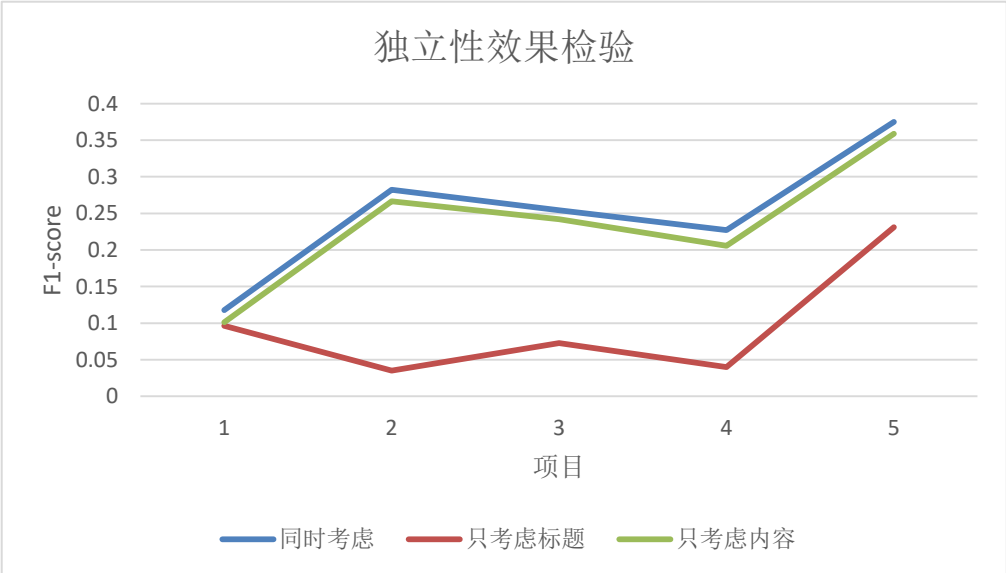


图 4.2 独立性效果检验图

4.7 SVM、NBM 与 J48 算法的预测结果比较

本文得到的 TRBF 方法是基于 J48 分类算法的，在这个基础上，分别于 SVM 支持向量机，NBM 朴素贝叶斯分类器进行标签推荐结果的比较，来证明 J48 算法在标签推荐方法中的优越性。

在 MEKA 的分类结果输出中，SVM 分类器和 NBM 分类器的输出预测结果只有两个值，1 或者 0，那么直接可以计算出查全率，查准率和 F1-score，将其与本文的标签推荐方法比较如下示：

表 4.6 SVM、NBM 和 J48 分类效果对比表

项目	分类算法	查全率	查准率	F1-score
adobe	J48	24.278%	11.347%	0.2428
	NBM	27.154%	5.556%	0.0922
	SVM	12.272%	19.067%	0.1493
angular	J48	47.847%	18.908%	0.2710
	NBM	48.529%	18.966%	0.2710
	SVM	36.275%	36.725%	0.3651
baystation12	J48	49.606%	38.009%	0.4304
	NBM	31.212%	26.823%	0.2885
	SVM	18.182%	24.391%	0.2083
cocos2d	J48	42.012%	20.700%	0.2773
	NBM	36.747%	12.629%	0.188
	SVM	30.121%	22.422%	0.2571
symfony	J48	41.109%	45.472%	0.4318
	NBM	45.183%	25.826%	0.3287
	SVM	31.792%	59.783%	0.4151

根据表格的 F1-score 作出基于不同机器学习分类算法的效果对比图。如图 4.3 所示，在 adobe，baystation12 两个项目中 J48 算法得到的 F1-score 都远远大于其余两种算法，在 angular 项目中，SVM 的表现最佳，而且 J48 的表现效果和 NBM 几乎一致。

在 cocos2d 和 symfony 项目中，J48 和 SVM 的分类效果都远远好于 NBM，而且 J48 的分类效果微弱领先于 SVM，综上所诉，J48 在四个项目中都表现最优，因此认定 J48 的效果最优。

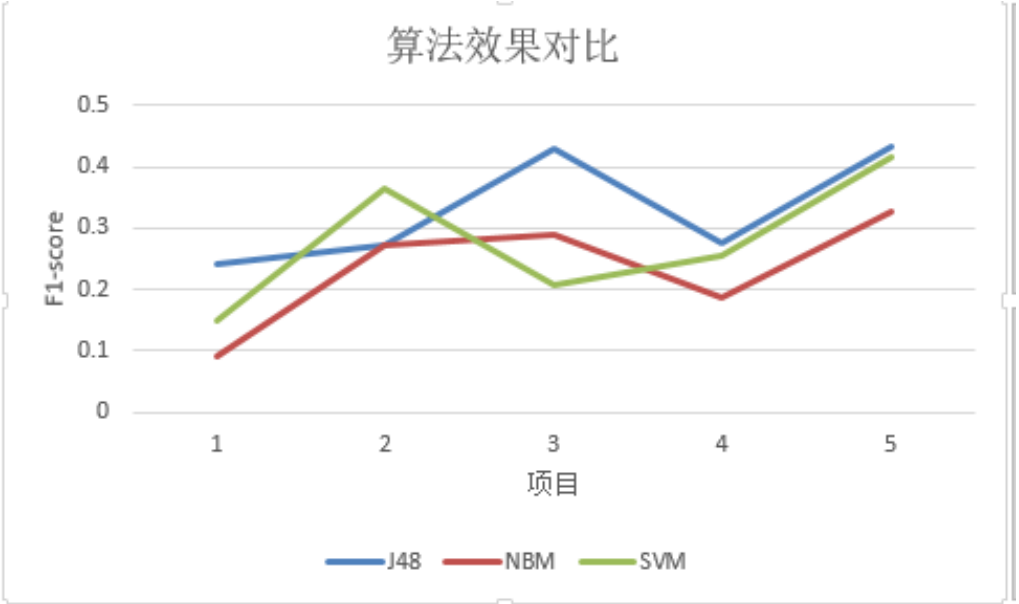


图 4.3 基于不同机器学习算法的效果比较图

4.8 基于特征相似度的预测方法比较

4.8.1 基于特征相似度的标签推荐方法概述

本节提出一种基于特征相似度的标签推荐方法，与本文 TRBF 方法进行推荐效果的对比，并评估验证 TRBF 方法的优越性。对于每个标签，都对应着已经拥有该标签的问题集合，那么，在我们度量一个未被分配标签的问题的时候，相似度度量（Similarity）则可以成为一个评估文本之间关联性的标准，因为两篇文档的相似度越高，代表着他们更可能属于同一种类型的文档，往往越可能会拥有相同的标签。

对于标签 N，首先获取拥有该标签的问题集合，对于待分类的每个问题，分别计算和这个问题集合中每个问题的文本相似度，最后得到平均文本相似度。那么我们可以得到所有的待分类问题针对标签 N 的平均文本相似度。将这些平均文本相似度排序后，值越大，代表拥有该标签的可能性越大，那么对于每个百分比，就可以得到对应的查全率、查准率和 F1-score。同样 F1-score 存在一个极大值，此时的分类器评估效果最好。

本节利用余弦相似度计算文本相似度。对于多个不同的文本要来计算他们之间的相似度，可以将其映射到多维空间，然后形成向量，计算向量之间的余弦值则可以得到相似度。对于 n 维度的两个向量$X(X_1, X_2, \dots X_{n-1}, X_n)$和$Y(Y_1, Y_2, \dots Y_{n-1}, Y_n)$，向量余弦值计算公式如下所示：

$$\cos\theta = \frac{\sum_{i=1}^{n}(X_i \times Y_i)}{\sqrt{\sum_{i=1}^{n}(X_i)^2} \times \sqrt{\sum_{i=1}^{n}(Y_i)^2}} \quad (4.4)$$

在空间中，如果空间中的两个向量越接近，那么他们的余弦值就越接近于 1。相反，当空间中的两个向量越有差异性，那么他们的余弦值就越接近于 0。利用余弦值的特性，就可以计算两个文本之间的相似度了。当我们想要计算两个文本之间的相似度时，首先我们需要找出两篇文章的关键词，通常将所有单词作为关键词，然后生成两个文本的词频向量，计算两个向量的余弦相似度，值越大则表示越相似，文本相似度越高。

4.8.2 文本相似度方法与 TRBF 方法的比较

针对选取的五个项目，得到五个项目百分比分割值对应的查全查准率和 F1-score 评估表（省略 10%以后，因为后续的 F1-score 一直降低）：

表 4.7 TRBF 和文本相似度方法效果对比表

项目	方法	查全率	查准率	F1-score
adobe	TRBF	24.278%	11.347%	0.2428
	文本相似度	13.982%	4.797%	0.0714
angular	TRBF	47.847%	18.908%	0.2710
	文本相似度	10.673%	3.503%	0.0527
baystation12	TRBF	49.606%	38.009%	0.4304
	文本相似度	9.146%	9.317%	0.0923
cocos2d	TRBF	42.012%	20.700%	0.2773
	文本相似度	14.433%	4.396%	0.0674
symfony	TRBF	41.109%	45.472%	0.4318
	文本相似度	23.404%	9.167%	0.1317

如图 4.4 所示，分别对应五个项目集，很显然 TRBF 方法准确率要远远高于基于特征相似度实现的分类方法，在每个项目中对于 F1-score 都是 TRBF 方法远远高于基于文本相似度的标签推荐方法，证明了本文 TRBF 方法的优越性。

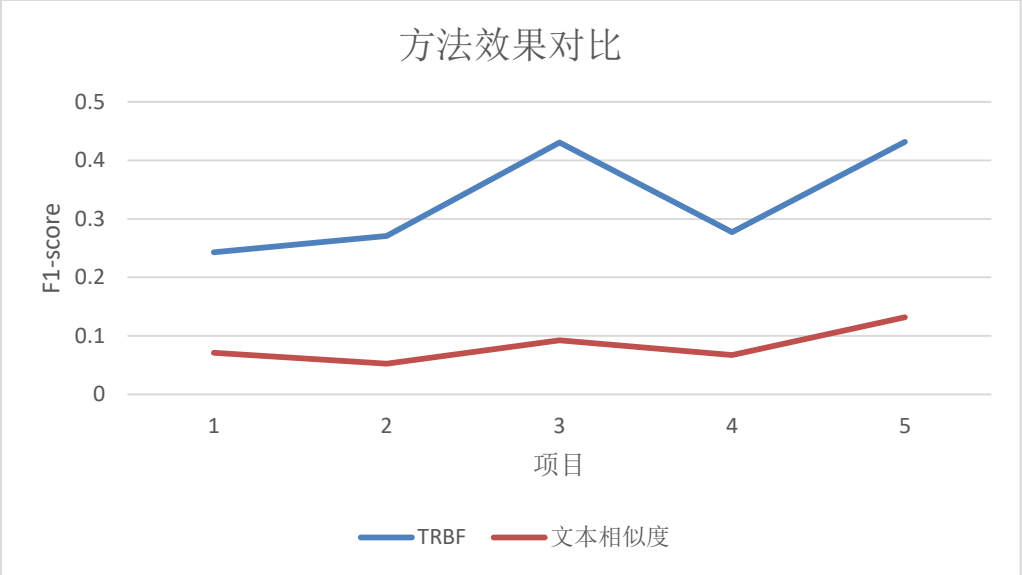


图 4.4 TRBF 与文本相似度方法效果比较图

4.9 TRBF 方法的最终效果

本章验证了文本特征处理的优化效果、得到了最优的代码特征和作者特征加权系数、证明了同时考虑问题标题和描述内容的优越性、验证了 J48 算法在多标签分类器中的效果最好，最后将 TRBF 方法和本节提出的一种基于特征相似度的标签推荐方法进行比较，得出 TRBF 方法的优越性。最后将上述已经验证的优化整合，得到的叠加优化效果如表 4.8 所示，通过百分比临界值的选定得到了每个项目最优的查全查准率和 F1-score，以及当 F1-score 取得最大值的百分比，得到最终的 TRBF 标签推荐方法[14]。

表 4.8 项目最优结果表

项目	adobe	angular	baystation12	cocos2d	symfony
查全率	24.278%	47.847%	49.606%	42.012%	41.109%
查准率	11.347%	18.908%	38.009%	20.700%	45.472%
F1score	0.2428	0.271	0.4304	0.2773	0.4318
百分比	9%	2%	4%	3%	8%

4.10 总结

本章评估了多标签分类器的效果，验证了同时考虑问题标题和描述内容的优越性，验证了文本特征处理和代码和作者特征加权对分类的正面影响，并在此基础上验证了 J48 分类算法相比于 SVM，朴素贝叶斯分类算法的优越性。另外验证了本文的 TRBF 方法相比基于特征相似度的标签推荐方法更准确。最后利用多标签分类器和百分比分割值得到 TRBF 标签推荐方法最终的效果。

结论

本文通过获取 Github 问题模块的问题属性，得到基于特征的标签推荐方法。课题的中心思想是将标签推荐转化成多标签文本分类，基于 MEKA 和 WEKA 构建出多标签分类器。本文猜想了多种可能的优化方法，对应每种方法分别构造对应的多标签分类器，分别输出预测结果，比较后得出结论。另外，在输出的基于 J48 分类器预测结果中，利用百分比分割临界值得到查全率和查准率的最优解，也就是 F1-score 取得极大值时候对应的解。

最后建立评估指标的对比效果，验证了加权之后的特征值和文本分词、提取词干对分类的正面影响，并在此基础上验证了在多标签分类中选取 J48 分类算法相比于 SVM，NBM 的优越性。论文还单独的提出一种通过文本相似度来推荐标签的方法，将每个待分类问题与某个特定标签下的具有该标签的问题集合求解平均文本相似度，同样利用百分比分割值来确定 F1-score 的极值，最后证明这种方法远远没有本文提出的 TRBF 方法效果好。

最终得到的标签推荐方法，查全率均值在 42%左右，查准率均值在 30%左右，在大数据文本分类中，这种效果已经达到较好的水平。

本文设计实现的标签推荐方法虽然分类效果较好，针对的项目集合也有可以拓展，还有比较大的进步空间。

致谢

时间过得飞快，毕业悄然而至。转眼间大学生活就快结束了，在大学里我收获了很多，毕业设计论文也是我大学四年认真学习的见证。

首先感谢蒋竞老师这一年来对我的不倦指导，使我得以明确每一步的工作和目标。在遇到疑难问题时，蒋老师都能及时的给出参考意见，让我最快的解决问题。

感谢陈学渊学长对我的帮助指导，毕设初期是迷茫的，感觉无从下手，是学长您教给我一系列基础知识，让我从爬虫程序入手，慢慢步入正轨。

感谢本科阶段给我授课的各位老师，让我的知识储备得到了进一步的提高，感谢体育老师们教给我的各种体育项目，让我的身心得以全面发展。

感谢 130611 班的所有同学，让本科的生活变得十分充实和有趣，有了你们的陪伴，我感觉十分荣幸。在遇到苦难时，总有同学可以帮助我，在遇到一些非常难懂的课程时，也总能和同学一起找到突破的方法，谢谢有你们！

感谢父母和姐姐在大学期间给予的帮助，让我能够安心的在学校学习科学文化知识，没有后顾之忧。

最后，感谢培养了我的母校北航和计算机学院，让我从四年前稚嫩的高中生蜕变成一个真正能独当一面的男人！

参考文献

[1] L. Dabbish C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository[C]. CSCW 12 Computer Supported Cooperative Work, Seattle, WA, USA, February 11-15, 2012:1277-1286.

[2] G. Gousios, A. Zaidman, M.-A. Storey, and A. Van Deursen. Work practices and challenge in pull-based development: The integrator's perspective[C]. IEEE International Conference on Software Engineering.IEEE, 2015:358-368.

[3] Casalnuovo C, Vasilescu B, Devanbu P, et al. Developer onboarding in GitHub: the role of prior social links and language experience[C]. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering,2015:817-828.

[4] Jason Tsay , Laura Dabbish , James Herbsleb. Influence of social and technical factors for evaluating contribution in Github[C].Proceedings of the 36th International Conference on Software Engineering,2014.

[5] J Cabot，JLC Izquierdo，V Cosentino，B Rolandi. Exploring the Use of Labels to Categorize Issues in Open-Source Software Projects[C].IEEE International Conference on Software Analysis, Evolution & Reengineering,2015.

[6] TF Bissyande，D Lo，L Jiang，L Reveillere. Got Issues? Who Cares About It? [C]. IEEE International Conference on Software Analysis, Evolution & Reengineering, 2013.

[7] Xin Xia, David Lo, Xinyu Wang and Bo Zhou.Tag Recommendation in Software Information Sites[J].College of Computer Science and Technology, 2014.

[8] David Lo, Bogdan Vasilescu, and Alexander Serebrenik. EnTagRec: An Enhanced Tag Recommendation System for Software Information Sites[C]. School of Information Systems, Singapore Management University, 2014.

[9] Aggarwal K, Hindle A, Stroulia E. Co-evolution of projects documentation and popularity within github[C]. Proceedings of the 11th Working Conference on Mining Sortware Repositories,2014:360-363.

[10] Pangsakulyanont, Thai, et al. Assessing MCR Discussion Usefulness Using Semantic

Similarity[C]. International Workshop on Empirical Software Engineering in Practice IEEE, 2014.

[11] Vasilescu, Bogdan, et al. Quality and productivity outcomes relating to continuous integration in GitHub[C]. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, 2015.

[12] Crowston K, Wei K, Howison J, et al. Free/Libre open-source software development:What we know and what we do not know[J]. ACM Coputing Surveys(CSUR),2012.

[13] Xuan J, Jiang H, Ren Z, et al. Developer prioritization in bug repositories[A]. Software Engineering (ICSE), 2012 34th International Conference on[C], 2012.

[14] Jiang Jing, JiaHuan He, and Xue-Yuan Chen. CoreDevRec: Automatic Core Member Recommendation for Contribution[J]. Journal of Computer Science and Technology,2016.