主观性与客观性是否总是一致？Stack Overflow 问题案例研究

Saikat Mondal 萨斯喀彻温大学，加拿大 saikat.mondal@usask.ca

Mohammad Masudur Rahman 达尔豪西大学，加拿大 masud.rahman@dal.ca

Chanchal K. Roy 萨斯喀彻温大学，加拿大 chanchal.roy@usask.ca

arXiv:2304.03563v1 [cs.SE] 7 Apr 2023

摘要—**在 Stack Overflow（SO）中，帖子（即问题和答案）的质量由用户通过投票机制进行主观评价。帖子获得的净票数（赞成票 – 反对票）通常被视为其质量的近似指标。然而，获得可行解决方案的问题中，约有一半获得的反对票多于赞成票。此外，约 18% 的已接受答案（即经验证的解决方案）也未获得最高得分。所有这些违反直觉的发现都使人们对 SO 所采用的评价机制的可靠性产生了怀疑。此外，许多用户对这种评价，尤其是对其帖子的反对票，表示担忧。因此，亟需对主观评价进行严格验证，以确保质量评估机制不存在偏见且值得信赖。在本文中，我们使用250 万个问题和十项文本分析指标，将问题的主观评价与其客观评估进行比较。根据我们的研究，四项客观指标与主观评价一致，两项不一致，一项可能一致也可能不一致，剩余三项既不一致也不不一致于主观评价。随后，我们开发机器学习模型，对获得促进和受到抑制 的问题进行分类。我们的模型优于最先进的模型，准确率最高约为 76%–87%。**

索引词—**Stack Overflow、问题质量、客观评价、主观评价、质量指标、分类模型**

I. 引言

SO 已成为最大的、最受欢迎的编程问题和解决方案问答网站之一。截至2022 年12月，它已成为一个宝贵的知识库，其中包含约 2,300 万个问题和3,500 万个答案，可用于软件开发过程中的各种问题解决 [52]。SO中任何问题或答案的质量，都是由用户通过投票机制进行主观评价的。高质量帖子通常会获得赞成票，而含糊、不清晰或模棱两可的帖子则很可能会获得反对票。SO 中针对给定帖子的净票数（赞成票 – 反对票）构成了一项称为 得分 的评价指标，同时也近似反映了帖子的主观质量。然而，根据我们的调查，获得可行解决方案的问题中有48%的得分低于零 [1]。也就是说，它们获得的反对票多于赞成票。换句话说，尽管这些问题是有效的，并且已经成功获得答案，但在其他人看来，它们可能不被视为高质量问题。约 18% 的已接受答案（即经验证的解决方案）也未能在问答主题的所有答案中获得最多的投票 [1]。也就是说，最佳答案可能并未被选为解决方案。SO 的用户也

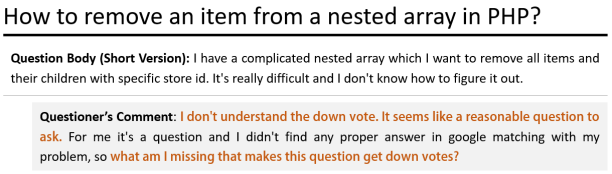


图 1：问题提交者在评论中抱怨，称对其问题的主观评价不合理（https://stackoverflow.com/questions/47183288）。

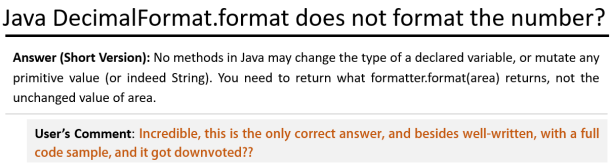


图 2：用户在评论中抱怨答案不得被投反对票的示例（https://stackoverflow.com/questions/ 11765616）。

通常会对其问题（例如图1）和答案（例如图 2）所获得的评价感到失望。上述所有发现都表明，Stack Overflow 用户所投出的投票具有高度主观性。

此外，先前的多项研究还表明，问答网站的激励系统未必总能以积极的方式引导某些用户。例如，机会主义用户可能会发现漏洞，并为自身利益而恶意操纵系统 [25，27，54]。上述所有发现都对 Stack Overflow 所采用的评价机制的可靠性提出了严重质疑。为了确保评价机制公平、可靠且有效，有必要开展一项调查，通过相应的客观评价来验证某篇帖子的主观评价（由 StackOverflow 用户给出）。客观评价涉及通过最小化主观偏差来估计质量的数值指标。在 StackOverflow中，对主观评价和客观评价进行此类一致性分析，或许有助于我们更好地理解其所采用的投票系统的潜在益处或危险。

多项研究分析了 Stack Overflow 上的问题和答案，并尝试评估其质量。Nasehi 等人分析了 Stack Overflow 帖子中所包含代码片段的质量，并研究构成优质代码示例的要素。Treude 和 Robillard 调查了开发者在多大程度上认为这类代码示例能够自解释。

Ercan 等人进一步研究了代码解释的恰当性，采用摘要导向的召回率评估方法（ROUGE）。Calefato 等人关注四种轻量级、以用户为导向的特征（例如用户声誉），以预测技术问答网站上的高质量答案。Novielli 等人认为，技术问题的情感风格可能是一项质量因素，并可能影响该问题的回答时间。因此，上述研究仅依赖轻量级、临时选取的属性来估计帖子的质量。另有少数研究尝试结合主观指标（例如用户声誉）和客观指标（例如文本可读性），区分获赞问题和被踩问题。虽然我们在方法上与这些研究有所重叠，但研究目标不同。具体而言，我们尝试验证主观评价是否合理，以及在不同客观质量指标方面，Stack Overflow 上得分较高（即受到推崇的）的问题是否确实优于得分较低（即不受青睐的）的问题。

在本研究中，我们以 250 万个 Stack Overflow 问题为对象开展了一项案例研究，并通过客观评价验证 Stack Overflow 问题的主观质量。具体而言，我们考察用户感知的问题质量是否与客观质量估计相一致。通过一致性分析，我们得出了若干结论。我们发现，四项 指标与帖子质量的主观评价完全一致，一项 指标基本一致，而两项 指标则与帖子质量的主观评价完全不一致。我们还基于客观度量开发了分类模型，其性能优于当前最先进的分类模型。Stack Overflow 可以利用我们的指标，引入一种估计帖子客观质量的机制。这样，用户既可以评估 Stack Overflow 上帖子的质量，也可以在提交时评估自己帖子的质量。或者，也可以开发一个采用我们模型的客户端浏览器插件。安装该插件后，用户不仅可以估计已有帖子的客观质量，还可以估计新帖子的客观质量。本文回答了两个研究问题，并作出以下两项贡献：

(研究问题$_1$)基于人群的主观评价与对 SO 问题质量的客观评估是否一致? 我们使用文献中的十项客观指标，对推荐问题和不鼓励的问题进行了比较分析。分析得出了若干发现。首先，主题熵、指标熵、文本-代码比和文本-代码相关性支持主观评价。也就是说，推荐问题主题明确、歧义较少、使用不常见的术语，而且其中的代码片段都有配套文字作出充分解释。其次，代码片段的可复用性和文本可读性与主观评价并不一致。也就是说，推荐问题中包含的代码示例难以复用，其可解析性低于不推荐问题中的代码示例。不推荐问题中的代码示例则完整且更易解析。不过，它们往往包含冗余语句，可能会增加代码分析所需的时间。根据我们的调查，推荐问题通常会在文本中包含程序元素（例如 API 类）

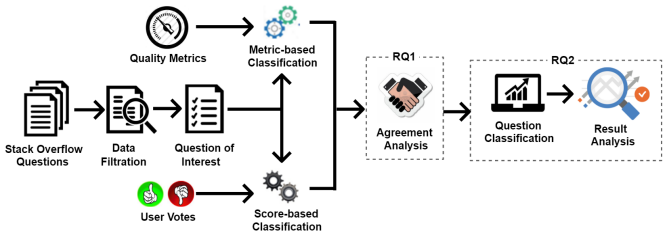


图 3：本研究的示意图。

。现有的可读性指标会将这类元素视为影响可读性的复杂词汇。第三，根据编程语言的不同，推荐问题的代码可理解性可能更高，也可能更低。第四，两类问题的标题质量、代码可读性和情感极性几乎相同。

(RQ$_2$)我们能否对 Stack Overflow 问答网站上的推荐问题和不推荐问题进行分类? 我们开发了五种机器学习模型，用于评估问题质量，并在问题创建时识别推荐和不推荐的问题。我们分析了各项特征并对其进行排名。分析结果表明，主题熵、文本可读性、指标熵和文本-代码比是四个最强的预测指标。我们的模型对问题进行分类的准确率最高约为 76%–87%，优于文献中最先进方法的准确率 [36] 。提前预测可能会被投反对票的问题，或许能极大地帮助用户在提交问题时改进问题。

复现包可在我们的在线附录 [31] 中找到。

II. 研究方法

图 3 展示了本研究的示意图。我们首先从 Stack Overflow 收集约 250 万个问题。我们从四种热门编程语言中收集这些问题，并采用了三项限制条件。然后，我们使用十项客观指标从不同方面评估问题质量。接下来，我们分析主观和客观质量度量之间的一致性。最后，我们使用五种机器学习模型，根据问题的客观质量对推荐问题和不推荐问题进行分类，并评估模型的性能。以下各节将讨论我们方法的不同步骤。

表 I 研究数据集概述

主题	推广问题	不推荐问题	总计
C#	550,173	76,181	626,354
Java	585,410	116,268	701,678
JavaScript	598,670	114,645	713,315
Python	418,700	67,035	485,735
总计	2,152,953	374,129	**2,527,082**

III. 数据收集

为通过客观评价验证对 Stack Overflow 问题的主观评估，我们共收集了 2,527,082

个来自 Stack Overflow、通过 StackExchange 数据 API [1] 获取的问题。具体而言，我们收集了与四种主要编程语言相关的问题：C#、Java、JavaScript 和 Python。我们按照以下限制筛选这些问题：（**i**）每个问题必须至少有一个答案，以确保该问题已引起社区关注；（**ii**）问题得分不得为零，这表明该问题至少获得过一次赞成票或反对票；以及（**iii**）问题须于 2017 年或更早发布在 Stack Overflow 上，这意味着它们已接受 Stack Overflow 用户相当长一段时间的评价。表 I 列出了本研究所用的数据集。我们根据问题得分将其分为两个子类别，如下所示。

- **推广问题**：得分大于 0 的问题（即获得更多赞成票的问题）。
- **不推荐问题**：得分低于 0 的问题（即反对票更多的问题）。

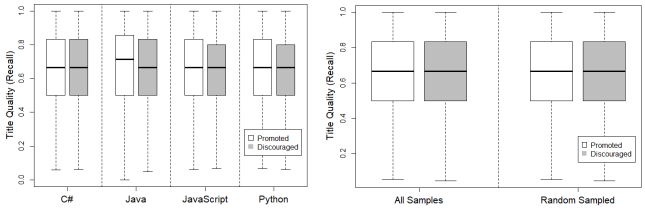
若干客观指标（例如代码片段的可复用性）可用于评估问题中包含的代码片段。根据我们的调查，2,527,082 个问题中有 1,909,539 个（即 75.56%）至少包含一行代码。我们识别出这类问题，并使用专用 HTML 标签（例如 <code> 位于 <pre> 下）提取代码片段，将其用于本研究。我们还通过 StackExchange 数据 API [1] 从标签 表中收集了总计 54,463 个标签。我们使用这些数据计算问题的主题熵。

IV. 使用十项客观指标对推荐问题和不鼓励的问题进行比较研究（RQ_1）

一个 SO 问题由三个部分组成：标题、正文和标签。正文通常包含代码片段及其文本描述。为回答 RQ_1，，我们使用文献中的十 项指标，对推荐问题和不鼓励的问题进行分析，以量化这三个部分的客观质量。这些指标包括：(a) 标题质量，(b) 文本可读性，(c) 代码可读性，(d) 文本-代码比，(e) 文本-代码相关性，(f) 代码可复用性，(g) 代码可理解性，(h) 主题熵，(i) 指标熵，以及 (j) 情感极性。所有报告的指标均可根据问题提交时可用的数据计算。标题质量用于估计标题与问题其他部分之间的一致性。解释质量指标（例如，文本-代码比、文本-代码相关性）用于估计代码解释的全面性，以及代码片段与其解释之间的连贯性。可读性指标（例如，文本和代码的可读性）用于估计文本和代码片段的可读性。主题熵和指标熵用于估计问题主题的模糊性，以及问题文本中所用术语的随机性。代码可用性指标（例如，代码可复用性和可理解性）用于估计代码片段的可解析性和可理解性。最后，情感极性指标用于估计问题文本所表达的正面、负面或中性情感。具体而言，我们比较推荐问题与不鼓励问题的客观质量指标。我们的目标是确定：(1) 两者之间是否存在任何明显差异。这两组指标之间；因此，（2）它们是否支持或反驳 Stack Overflow用户所作出的主观评价。最后，我们尝试根据这些发现为 RQ_1 得出结论。我们的比较研究分为以下几项不同的分析：

A. 标题质量（*TQ*）

标题是一个重要部分，应当恰当地概括问题，而正文则会详细解释编程问题。然而，撰写不佳的标题可能无法吸引那些本可以回答问题的潜在用户。因此，我们使用一种广受欢迎且轻量级的指标 ROUGE 来衡量标题质量。ROUGE 会将摘要与人工撰写的摘要进行比较，从而自动评估摘要质量。ROUGE 与人工评分具有很高的相关性 [19]。我们在分析中使用 ROUGE 2.0 [21] 。我们研究了不同类型的 ROUGE 指标，例如 ROUGE-N（N 元语法共现统计）、ROUGE-L（最长公共子序列）、ROUGE-W（加权最长公共子序列）、ROUGE-S（跳跃二元组共现统计）和 ROUGE-SU（在跳跃二元组中加入一元语法作为计数单位）。经过仔细分析，我们发现 ROUGE-1（一元语法）在我们的研究情境中表现良好。因此，我们使用 ROUGE-1 评估标题质量。在这里，我们将问题的标题视为参考摘要（即人工生成的摘要）。ROUGE 将包含更多问题详细信息的正文作为系统摘要。接下来，我们按如下方式计算召回率，以确定标题质量。

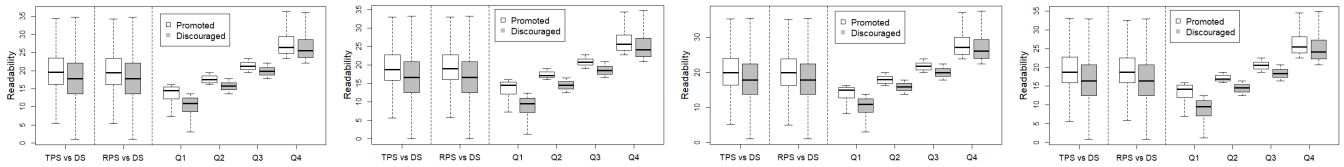


(a) 对于每一种编程方式语言　(b) 对于所有编程语言-仪表

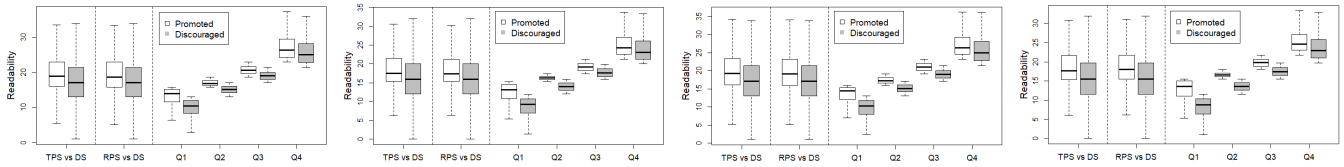
图 4：标题质量

$$Recall = \frac{COUNT(\text{参考摘要} \cap \text{系统摘要})}{COUNT(\text{参考摘要中的词})}$$

RQ_1（a）：标题质量是否与主观评价一致？ 如图 4 所示，我们没有发现推广问题和不推荐问题之间的标题质量存在任何显著差异。首先，我们尝试确定推广问题和不推荐问题的标题质量。我们发现，与不推荐问题相比，与 Java 相关的推广问题的标题具有略高的召回率。其他三种语言的问题类别在召回率方面几乎完全相同。随后，我们合并了所有语言的样本，同样没有发现任何显著差异。最后，我们对推广问题进行了欠采样，以平衡数据集，并观察到了类似的结果。



（a）没有代码片段的 C# 问题 (b) 带有代码片段的 C# 问题 (c) 不包含代码片段的 Java 问题-pet （d）包含代码片段的 Java 问题



(e) 没有代码的 JavaScript 问题片段 （f）包含代码的 JavaScript 问题片段 (g) 没有代码的 Python 问题片段 （h）含代码片段的 Python 问题-pet

图5：文本可读性（**TPS** = 总推广样本，**RPS** = 随机推广样本，**DS** = 不鼓励样本，**Q1** = 第一四分位数，**Q2** = 第二四分位数，**Q3** = 第三四分位数，以及 **Q4** = 第四四分位数）。

B. 可读性

可读性是读者理解书面文本的容易程度 [2]。SO 问题通常包含两类内容——普通文本和代码片段。因此，我们尝试使用两个独立的指标分别捕捉文本和代码的可读性：文本可读性 和 代码可读性。我们通过解析每个问题的 HTML 内容，提取代码片段和文本（正文 + 标题）。然后，我们按如下方式对它们应用可读性指标。

*1)*文本可读性（*TR*）：文本的可读性取决于其词汇、句法和呈现风格的复杂程度，例如字体大小、行高和行长 [2]。我们分别计算以下两类问题的文本可读性：（1）仅包含文本描述的问题；（2）包含文本描述和代码片段的问题。为评估文本可读性，我们计算一种常用的可读性指数，即 可读性指数（*RIX*） [4]。该可读性指数根据美国年级水平表示英文段落的理解难度。首先，我们计算每段文本的可读性，然后将得分归一化到 0 到100之间。得分越低表示可读性越高，而得分越高表示问题文本的可读性越差。

研究问题$_1$（b）：推荐问题的文本可读性是否高于不推荐问题? 首先，我们仅选取正文中不包含任何代码片段的问题中的 25%。如图5所示，我们发现推荐问题与不推荐问题之间的文本可读性存在显著差异，其中推荐问题的文本可读性分数高于不推荐问题。也就是说，推荐问题的可读性低于不推荐问题。在我们的数据集中，推荐问题的数量多于不推荐问题。因此，我们对推荐问题进行欠采样，以平衡数据集。随后，我们比较不推荐样本与推荐样本的可读性。对于所有编程语言，我们通过p 值从 *Mann-Whitney-Wilcoxon* 统计检验和效应量从*Cliff's delta*检验中获得了显著结果。对于 Java，我们得到p 值 $\approx 0 < 0.05$和*Cliff's d* $= 0.20$（小）。对于其他三种语言——C#、JavaScript 和 Python，我们也得到了类似的结果。接下来，我们将可读性分数划分为四个四分位数，并使用箱线图进行比较。我们发现，推荐问题与不推荐问题在所有四分位数之间的文本可读性均存在统计学显著差异（即p 值 < 0.05）。然而，效应量从小到大不等。其次，我们测量其余 75% 包含代码片段和文本描述的问题的可读性。不过，我们仅考虑其中的文本来测量可读性。在这种情况下，我们同样发现推荐问题与不推荐问题之间的可读性存在显著差异。

我们进一步探究推荐问题可读性较低的原因。我们发现，推荐问题通常包含程序元素（例如 API 类、函数名），这些元素被视为复杂词汇。遗憾的是，现有的可读性指标并不适合处理文本中的代码或程序元素。我们曾尝试从文本描述中移除程序元素，但这面临着若干挑战。首先，我们发现许多用户没有使用恰当的 HTML 标签（例如，<code>....</code>）来格式化代码元素，因此难以通过标签识别这些元素。其次，正则表达式也不足以检测这些元素，因为不同编程语言中代码元素（例如变量名）的表示方式各不相同。因此，我们没有从文本中移除代码元素。

我们还尝试了其他可读性指标（例如，弗莱施-金凯德得分、冈宁雾指数）。然而，这些指标无法区分推荐的问题和不推荐的问题。有趣的是，RIX 的可读性分数能够区分这两类问题。也就是说，组合使用多个可读性指标并不会影响我们的研究发现，反而会增加计算复杂度。因此，我们仅使用 RIX 来确定文本的可读性。

2）代码可读性（代码可读性）：代码可读性是一种属性，用于衡量未编写特定代码片段的开发者能否轻松阅读和理解该代码 [38]。代码可读性与软件可维护性和质量密切相关。因此，其基本理念是：代码越易读，长期来看就越容易复用和维护。由于代码可读性差会耗费开发时间和精力，我们分析了问题中所添加代码片段的可读性。具体而言，我们试图找出推荐的问题和不推荐的问题在代码可读性方面是否存在明显差异。

为了计算代码可读性，我们使用 Buse 和 Weimer [11] 开发的工具。他们的可读性模型基于人类对可读性或可理解性的感知进行训练。该模型使用代码中的若干文本特征，这些特征可能会影响人类对可读性的感知，并预测一个介于零和一之间的可读性分数。一表示源代码的可读性很高，而零表示可读性很差。

RQ_1(c)：推荐问题的代码示例是否比不推荐问题的代码示例具有更高的可读性分数？ 根据我们的分析，在 Stack Overflow 的推荐问题或不推荐问题中找到的代码片段的可读性都非常差。例如，相当大一部分（34%）代码片段的可读性分数为零。只有约 14% 的代码片段具有中等或以上（即，得分为 ≥ 0.5）的可读性分数。尽管 p 值来自 *Mann-Whitney-Wilcoxon* 统计检验的结果具有显著性（即，p 值 <0.05），但 d 值来自 *Cliff'sDelta* 检验表明，对于全部四种编程语言，效应量均可忽略不计（即，d *<0.1*）。

为了进一步深入了解，我们从四种编程语言中随机选取了 200 个代码片段（100 个被提升， + 100100 个被抑制），并花了约十小时进行分析。其中一位作者拥有十余年的软件开发经验，他手动检查了这些代码片段。我们没有安排多位标注人员参与，因为我们只关注代码片段的结构、完整性和噪声（例如堆栈跟踪）。不过，我们首先仔细分析了该工具的规范， [11] 以计算可读性分数。随后，我们根据这些规范检查了影响可读性的违规情况。进一步的人工调查得出了以下观察结果。(1) 大多数代码片段都不完整，也没有保持恰当的结构、缩进和标识符命名约定。(2) 问题提交者常常在一个问题中包含多个代码片段。将这些片段合并到单个文件后，文件就会变得庞大而混乱。我们采用的工具可能不太适用于这类合成代码片段。(3) 许多代码片段都包含噪声。程序员常常会在代码片段中加入非代码元素（例如堆栈跟踪）。这类噪声可能会降低代码片段的可读性。

C. 代码片段的解释质量

在 Stack Overflow 问题中，用户通常会用常规文本写成的说明文字来解释相关代码片段。Ercan 等人

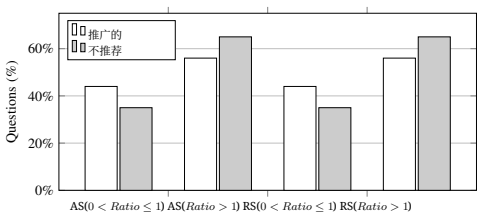


图 6：文本-代码比（**AS**=所有样本，**RS**=随机抽样）。

表 II 文本-代码比 (TCR)汇总（**PM**：推荐，**DC**：不推荐）

比例	C#		Java		JavaScript		Python	
	PM	DC	PM	DC	PM	DC	PM	DC
比例 <= 1 (百分比)	46	36	39	28	44	35	51	44
比例 >1 (百分比)	54	64	61	72	56	65	49	56

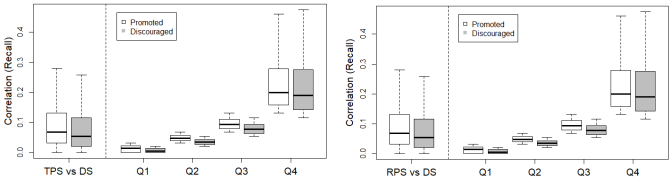
[19]指出，篇幅较长且缺乏解释的代码片段会使问题难以理解，这可能会影响获得合适答案所需的时间。恰当解释代码的说明可能有助于用户回答问题。因此，代码解释质量可以视为衡量问题质量的一项指标。我们通过以下两种方式衡量代码解释质量。

1) 文本-代码比*(TCR)*：该指标用于确定问题中相关正文对代码的解释程度。为了考察代码解释的全面性，我们从问题的正文中提取代码片段和文本。然后，我们根据所使用的 ASCII字符数量计算它们的长度。最后，我们按照如下方式测量文本-代码比。

$$TCR = \frac{LENGTH(code)}{LENGTH(texts)}$$

RQ_1(d)：推广问题是否比不推荐问题包含更多针对其代码的代码解释？ 我们将 TCR 分为两类。当文本长度大于或等于代码长度时，这些问题属于第一类。第二类包括代码长度大于文本长度的问题。从图 6 中可以看出，推广问题比不推荐问题包含更多代码解释。Squire 和 Funkhouser [48]也指出，高质量问题相比低质量问题，会对代码片段进行更详细的解释。如表 II所示，在四种编程语言中，第一类的所有推广问题都比不推荐问题具有更高的代码解释比率。例如，C#、Java、JavaScript和 Python的推广问题，其文本-代码比分别比不推荐问题高 10%、11%、9%和 7%。我们还确定了推广问题与不推荐问题之间的文本-代码比差异是否具有统计显著性。我们使用 *Mann-Whitney-Wilcoxon*统计检验，并发现显著的 p 值 （p 值 ≈ 0），且具有较小的效应量。

2) 文本-代码相关性（*TCC*）：上一节衡量了代码解释的完整性。本节中，我们确定代码解释的适当性。



(a) 所有编程语言 (b) 所有编程语言（之后随机抽样)

图 7：文本-代码相关性测量总结。

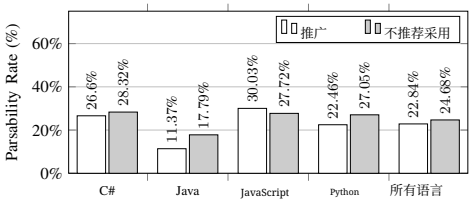


图 8：编程语言的可解析性比率。

通过测量一个问题中代码片段与代码解释之间的相关性来进行代码解释。Ercan 等人 [19] 评估代码片段解释的质量，评估标准是人类可读性，并指出，代码片段与代码解释之间较高的文本重叠度与问题质量高度相关。因此，我们假设，如果存在词元重叠，那么文本描述可能是在谈论代码片段。由此，我们使用 ROUGE研究代码解释与代码片段之间的内容重叠。具体而言，我们将代码片段用作参考摘要，将代码解释用作系统摘要。然后，我们计算召回率。所得的 ROUGE 得分（即召回率）表明，代码解释（系统摘要）对目标代码（参考摘要）的解释恰当程度。

研究问题$_1$（e）：推荐问题的代码片段与代码解释之间的相关性是否高于不鼓励的问题？根据我们的分析，推荐问题的代码片段与代码解释之间的相关性强于不鼓励的问题。图 7 以箱线图的形式总结了结果。我们发现，所有推荐问题的解释与代码之间的相关性都更高，且 p 值显著，效应量从小到大不等。

我们使用 曼-惠特尼-威尔科克森统计检验和 克利夫的 *delta* 检验，分别计算统计差异和效应量。检验结果表明，第一、第二和第三四分位数存在统计学上的显著差异（即，p值 < 0.05），且效应量较大。相比之下，第四四分位数显示出较小或可忽略的效应量。

D. 代码可复用性（*CRUSE*）

源代码的可用性基于源代码的可解析性、可编译性和可执行性来确定 [59]。在本研究中，我们仅考察 Stack Overflow 问题中发现的代码片段的可复用性的一部分，即 可解析性 。我们的目标是测量四种编程语言（C#、Java、JavaScript 和 Python）代码片段的可解析性比率。随后，我们对比推荐问题与不鼓励的问题的可解析性比率。对于 C# 代码片段的解析，我们使用微软提供的一款名为 *Roslyn* 的工具。Roslyn 提供了丰富的 API（例如 CSharpSyntaxTree.ParseText ），用于从 C# 代码中解析抽象语法树。对于 Java 代码片段的解析，我们使用了 *JavaParser*[1]。我们使用 *Esprima*[2] 解析 JavaScript 代码示例。我们使用 Python 内置的 AST 模块解析 Python代码片段。

研究问题$_1$(f)：推荐问题中的代码片段是否比不鼓励的问题中的代码片段更容易解析？如图 8 所示，不推荐问题中的代码示例解析率高于推荐问题中的代码示例。四种编程语言中，有三种（C#、Java 和 Python）呈现出这一现象。我们发现，推荐问题中代码示例的总体解析率约为 23%，而不推荐问题中代码片段的解析率为 25%。我们还通过人工分析，试图找出不推荐问题中的代码示例解析率较高的原因。根据我们的调查，Stack Overflow 上的用户（尤其是新手用户）经常提交包含冗长代码的问题，却没有进行适当分析。我们认为，这样冗长的代码会给其他用户回答问题增加额外负担。因此，即使这些问题中的代码可解析，它们也常常会收到反对票，且得不到合适的答案。

E. 代码可理解性（*CUA*）

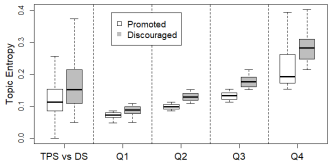
我们使用现有工具 [11] 估计了代码片段的可读性。然而，感知到的可读性与代码的实际可理解性有所不同。例如，开发者可能认为一段代码具有可读性，但仍然难以理解该代码的功能[46]。Scalabrino 等人 [46]报告称，代码可理解性通常取决于程序中所使用的 API 数量。使用的 API 越多，代码就越难理解。因此，我们将代码片段中的 API 数量作为代码可理解性的代理指标。我们使用正则表达式查找代码片段中使用的 API。

RQ$_1$(g)：推荐问题中的代码片段是否比不推荐问题中的代码片段更易于理解？根据我们的分析，推荐问题与不推荐问题在 API 使用数量上的差异没有统计学意义。

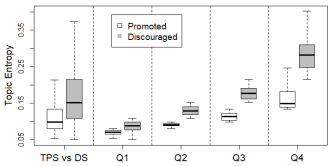
F. 主题熵（主题熵）

只讨论常见主题（例如 Java）的编程问题可能不够精确，或存在歧义。精确的主题有助于用户找到自己感兴趣的问题。我们研究问题主题的不确定性或歧义，并尝试确定不推荐问题是否比推荐问题使用了更常见的主题。在 SO 中，标签

[1]http://javaparser.org [2] http://esprima.org

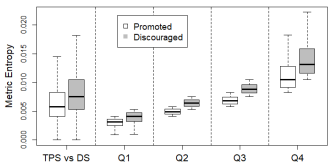


(a) 所有编程语言

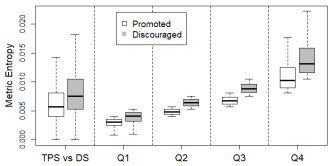


(b) 所有编程语言（经过随机抽样）

图 9：主题熵汇总。



（a）所有编程语言



（b）所有编程语言（经过随机抽样）

图 10：指标熵汇总。

捕捉与问题相关联的主题 [60]。每个问题最多可以有五个标签，并且至少必须有一个标签。在信息论中，熵被视为对取不同值的随机变量的不确定性的度量 [40]。为了度量与每个问题相关联的主题熵，我们首先计算每个主题（即标签）的概率。假设某个特定主题的主题概率 i 为 P_i。

$$P_i = \frac{F_i}{\sum_{i=1}^{N} F_i} \tag{1}$$

其中， F_i 是 i^{th} 主题的频次， N 表示主题总数。然后，我们按如下方式确定每个问题的主题熵。

$$TE = -\frac{1}{n}\left[\sum_{k=1}^{n} P_k \times log(P_k)\right] \tag{2}$$

n 表示一个问题的主题总数（即标签数）。由于不同问题的标签数量不同，这里我们计算平均熵（将熵除以 n）。随后，我们对其进行归一化。如果 Stack Overflow 上的一个问题使用了一组相当常见的主题，其熵就会更高。根据信息论，较高的熵值表示问题主题具有较高的模糊性或不确定性，反之亦然。

研究问题$_1$（h）：推广问题的主题是否较少歧义？ 图 9 展示了主题熵的箱线图。我们可以看到，对于所有编程语言，推广问题的主题熵都低于不推荐问题。换言之，推广问题具有明确的主题，而不推荐问题的主题则存在歧义。我们进一步检验了对于每种语言，主题熵的差异是否具有统计显著性。我们采用 *Mann-Whitney-Wilcoxon* 检验，发现所有检验案例的 p 值均具有显著性（p 值 $\approx 0 < 0.05$）。我们使用 *Cliff's delta* 检验进行的四分位数分析表明，对于全部四个四分位数，这些差异均具有统计显著性，且效应量较大。当我们采用整个数据集时，差异同样具有统计显著性，但效应量为中等或较小。

G. 指标熵（*ME*）

我们使用指标熵来对比推荐的问题和不推荐问题。指标熵是香农熵[49] 除以文本长度。它表示问题文本中所包含信息的随机性。[36]。更高的指标熵表明，问题文本中更频繁地使用常见词，反之亦然。

研究问题$_1$（i）：推荐问题是否使用了更多不寻常的术语？ 图 10 展示了指标熵的箱线图。我们可以看到，对于所有编程语言，推荐问题的指标熵都低于不推荐问题。也就是说，推荐问题使用了更多不寻常的术语，而不推荐问题中的术语则较为常见。随后，我们使用 *Mann-Whitney-Wilcoxon* 检验，发现整个数据集的 p 值均具有显著性（即 $p \approx 0 < 0.05$）。不过，效应量较小。我们使用*Cliff'sdelta* 检验进行的四分位数分析表明，对于全部四个四分位数，差异均具有显著性，且效应量为中等或较大。

表 III 情感极性详情

语言	类别	情感极性（百分比）			
		正面	负面	混合	中性
Java	已推广	23.60	17.26	15.46	43.68
	不鼓励使用	20.95	16.25	11.05	51.74
C#	已推广	24.03	16.78	15.53	43.66
	不鼓励	22.00	15.05	10.00	52.94
JavaScript	已推广	26.53	14.71	14.75	44.00
	气馁的	24.52	12.71	9.94	52.81
Python	已推广	23.48	17.47	15.77	43.29
	不鼓励	21.43	16.70	11.33	50.53

H. 情感极性（*SP*）

软件开发者经常在各种社区问答网站（例如 SO）和开发环境（例如 GitHub）中彼此互动。Calefato 等人 [12] 指出，开发者的情绪状态等社会因素可能会影响问答网站上的响应时间。也就是说，问题文本中表达的情感可能决定问题的回答时间。因此，我们将 SO 问题文本的情感极性视为一种客观的质量因素。我们使用 *SentiStrength-SE*这一情感分析工具来衡量情感极性。SentiStrength-SE 由 Islam 和 Zibran [26]针对软件工程领域开发。

研究问题$_1$（j）：SO 问题中表达的情感是否会影响其主观评价？ 我们测量了从问题的标题和正文中提取的文本所表现出的积极、消极、混合和中性情感极性。表 III 总结了 SO 问题的情感测量结果。我们发现，推荐问题和不鼓励的问题中，约有一半内容既不积极也不消极。我们还发现，与不鼓励的问题相比，推荐问题文本中积极情感的比例更高，而中性情感极性的比例更低。不过，这些差异在统计学上均不显著。

表 IV
特征排名（信息增益）

Rank	特征	得分
1	TE	0.6931
2	ME	0.5328
3	TCR	0.3156
4	CR	0.2803
5	TR	0.2486
6	TCC	0.0167
7	TQ	0.0044
8	SP	0.0040
9	CUA	0.0021
10	CRUSE	0.0017

表 V
使用特征分析决策树

Rank	特征	准确率
1	TE	66.36%
2	TR	63.44%
3	ME	60.49%
4	TCR	56.25%
5	TCC	54.11%
6	SP	53.95%
7	CUA	53.01%
8	TQ	52.86%
9	CR	52.18%
10	CRUSE	52.17%

积极情感的比例更高，而中性情感极性的比例低于不鼓励的问题。不过，这些差异在统计学上均不显著。

V. 基于问题质量对问题进行分类的模型构建（RQ_2）

在本节中，我们首先讨论如何对第四节中发现的特征进行排名。然后，我们介绍所选的机器学习模型及其设置。接下来，我们评估模型的性能。最后，我们将模型的性能与基线模型进行比较。

A. 特征排名

在第 IV 节中，我们展示了若干属性（例如主题熵），这些属性的值在获推荐的问题和不推荐问题之间存在差异。这些发现表明，这些特征可能是预测某个问题会获得赞成票还是反对票的关键估计指标。然而，我们还不知道，在区分获赞（即获推荐）和获反对票（即不推荐）的问题方面，哪些属性比其他属性更为重要。因此，对这些属性进行排名可能有助于选择排名靠前的 N 特征来预测问题。我们使用以下两种常用的度量方法对这些特征进行排名。

信息增益。在信息论中，随机变量的信息增益是先验状态与获取某些信息[44]后的状态之间信息熵的变化。因此，预测问题是否会被推广或劝阻的特征的信息增益如下。

$$InfoGain(C, a_i) = H(C) - H(C|a_i) \quad (3)$$

其中，C 表示特定类别（即被推荐或不被推荐），a_i 表示属性，H 表示信息熵。如表 IV 所示，主题熵的信息增益最高。也就是说，主题熵可能最准确地区分问题。指标熵、文本-代码比、代码可读性和文本可读性的信息增益分别位居第二、第三、第四和第五位。然而，其余属性的增益率要么较小，要么可以忽略不计。

使用决策树进行分析。 在本研究中，我们不仅尝试预测问题在提问时的质量提交时间，但也要了解哪些属性会影响问题的质量。因此，我们尝试使用决策树，观察每个属性如何对优质问题和劣质问题进行分类。表 V 展示了各个特征的分类准确率。我们发现，主题熵能够以 66.36% 的总体准确率对问题进行分类。文本可读性、指标熵和文本-代码比的区分能力分别位居第二、第三和第四。与信息增益的结果相反，我们发现文本-代码相关性的分类准确率高于代码可读性。不过，在两种情况下，排名前五的属性中有四个是相同的。因此，我们选取这四个特征（即主题熵、文本可读性、指标熵、代码-文本比）作为最优特征集，并分别使用全部特征和排名前四的特征来分析模型的表现。

B. 分类模型与设置

模型选择。 根据我们的比较研究，问题类别与其对应特征值之间的关系可能较为复杂。因此，我们选择了五种采用不同学习策略的常用监督式机器学习技术，分别是：i) 决策树 [39, 9] (DT) ii) 随机森林 (RF) [8], iii) 人工神经网络 (ANN) [57],iv) K近邻 (KNN) [3], 和 v) 高斯朴素贝叶斯 (GNB) [56, 42]。我们选择这些机器学习算法，特别是因为它们在相关研究中得到了广泛应用 [44, 36, 40, 6]。因此，我们相信它们能够构建可靠的模型，对推广类问题和讨论类问题进行分类。我们使用 *Scikit-learn* [35],这一广受欢迎且应用广泛的工具来实现这些技术。

参数调优。 对分类器进行参数调优非常重要，因为参数会改变决定模型如何学习的启发式规则 [15]。例如，它可以控制随机森林中使用的决策树数量，或 K近邻 中的簇数量。由于参数设置取决于数据集 [23]，使用次优参数设置训练的模型可能表现不佳。为选择最佳模型配置，我们使用 *GridSearchCV*，即 Scikit-learn [35] 的交叉验证网格搜索算法。GridSearchCV 算法通过运行并评估所有可能的参数组合所对应的模型性能来完成搜索。我们为机器学习模型的重要参数选择若干合理的调优选项。GridSearchCV 并行执行两项任务：(1) 交叉验证和 (2) 参数调优。最后，它会返回使模型准确率最高的参数值。这里，我们保留可选参数的默认值。我们还考察训练集和测试集划分的效率，并据此选择参数，以避免模型过拟合。

C. 成功指标

我们使用四个常用的性能指标来评估我们的分类模型的性能——(1) 精确率、(2)召回率、(3) *F1*分数和(4) 分类准确率。

表 VI 分类器的性能（**H** 表示最高值）

分类器	特征	数据集	推荐问题			不鼓励的问题			总体
			精确率	召回率	F1分数	精确率	召回率	F1分数	准确率
DT	**All**	**不平衡**	90.90	89.98	90.44	59.27	61.80	60.51 **H**	84.60
		平衡	74.56	74.17	74.37	74.30	74.70	74.50	74.30
	前4名	**不平衡**	91.85	91.10	91.48 **H**	63.51	65.73	64.60 **H**	86.26 **H**
		平衡	75.37	74.90	75.14 **H**	75.06	75.53	75.29 **H**	75.21 **H**
RF	**All**	**不平衡**	88.39	94.03	91.12 **H**	65.29	47.65	55.09	85.18 **H**
		平衡	76.98	72.09	74.46	73.76	78.44	76.03 **H**	75.27
	前四名	**不平衡**	89.01	94.45	91.45	68.22	50.54	58.06	86.07
		平衡的	75.17	71.90	73.50	73.07	76.25	74.62	74.07
ANN	**All**	**不平衡**	83.01	97.45	89.65	58.79	15.40	24.41	81.80
		均衡的	73.54	79.41	76.36 **H**	77.62	71.42	74.39	75.42 **H**
	前 4 名	**不平衡**	76.42	89.80	82.57	68.61	44.60	54.06	74.73
		平衡	70.37	75.79	72.98	73.77	68.08	70.81	71.94
KNN	**All**	**不平衡**	76.11	88.77	81.95	66.33	44.27	53.10	73.93
		平衡	66.87	74.71	70.57	71.35	62.99	66.91	68.85
	前四名	**不平衡**	76.14	88.74	81.96	66.34	44.37	53.17	73.95
		平衡	66.43	73.61	69.74	70.41	62.80	66.39	68.21
GNB	**All**	**不平衡**	82.52	95.19	88.40	41.51	14.49	21.48	79.79
		平衡	55.73	90.60	69.01	74.88	28.01	40.77	59.31
	前 4 名	**不平衡**	82.59	95.97	88.78	45.36	14.20	21.63	80.37
		平衡	56.62	89.59	69.39	75.07	31.35	44.23	60.47

D. 性能分析

我们采用 10 折交叉验证来训练和测试每个机器学习模型。我们使用 (1) 所有特征和 (2) 排名前四的特征来训练和测试这些模型。随后，我们分别分析结果。表 VI 总结了分类结果。我们用**H**标记了最高性能。使用所有特征和不平衡数据集时，决策树对推荐问题的分类表现最佳，精确率最高约为 91%，召回率最高约为 90%。在不平衡数据集中，不推荐问题的精确率约为 60%。不平衡数据集可能会生成次优的分类模型。在平衡数据集上训练模型时，精确率从约 60% 提高到 75%。平衡数据集和不平衡数据集的总体分类准确率分别约为 75% 和 85%。选择排名前四的特征时，评估指标显示的结果与上述结果基本相似。这表明其余特征对分类模型的影响较小。使用随机森林和不平衡数据集训练模型时，我们发现不推荐问题的精确率从约 60% 提高到 66%，但召回率下降了约 14%。使用排名前四的特征训练模型时，也得到了类似的结果。

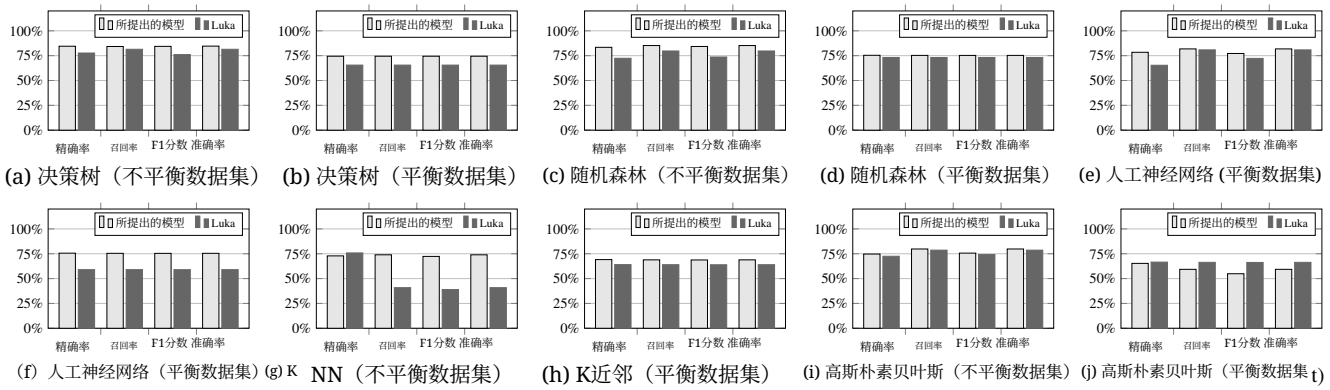
在其余三种机器学习模型中，使用平衡数据集时，人工神经网络的表现尚可；使用前 4 个特征时，K近邻的表现良好。K近邻通常在特征较少时表现更好，而且不太受训练阶段影响。使用不平衡数据集进行训练时，人工神经网络和高斯朴素贝叶斯在预测不鼓励的问题方面表现不佳。高斯朴素贝叶斯分类器对数据分布的形状作出了非常强的假设。也就是说，在给定输出类别的情况下，它假设任意两个特征要么相互独立，要么不独立。这一假设可能会影响模型的性能。高斯朴素贝叶斯还会受到类别不平衡（导致概率偏斜）以及连续特征 [41]。

分析结果后可以得出结论：基于树的分类器（例如决策树）能够根据我们的数据集和特征生成更好的模型来对问题进行分类。另一方面，概率模型（如高斯朴素贝叶斯）无法生成最优模型，尤其是在使用不平衡数据集时。

E. 与基线模型的性能比较

据我们所知，Stack Overflow上主观性与客观性之间的一致性分析仍是一项尚未完成的尝试。然而，Ponzanelli 等人 [36]尝试根据质量对问题进行分类，并构建机器学习模型，以区分质量较差（即，得分 <0）和质量较好（即，得分 >0）的问题。他们确定了三组指标，其中包括SO问题的文本特征和非文本特征。这些指标分别是 SO（M_{SO}）、可读性（M_R）和用户受欢迎度（M_P）。然而，用户受欢迎度指标的属性（例如，赞成票、反对票）直接或间接地与主观评价相关。在本文中，我们研究客观质量指标。因此，我们选择 M_{SO} 和 M_R指标来准备一个等价属性集。

图 11 展示了我们模型与 Ponzanelli 等人 [36] 的基线模型的对比结果。我们采用相同的机器学习模型设置和相同的数据集来比较模型的性能。首先，我们计算精确率、召回率、F1分数和总体准确率。然后比较我们模型与基线模型在各项评估指标上的数值。当我们使用决策树训练模型并采用不平衡数据集时，F1分数和准确率均约为 85%（图 11a）。而当我们使用基线研究 [36] 中采用的属性训练模型时，这两项指标分别低至约 76% 和 82%。使用平衡数据集时，模型性能也显著更高。F1分数和总体准确率比基线模型高约 10%（图 11b）。我们发现，使用随机森林训练模型时，基线模型的总体性能有所提升。



(a) 决策树（不平衡数据集） (b) 决策树（平衡数据集） (c) 随机森林（不平衡数据集） (d) 随机森林（平衡数据集） (e) 人工神经网络 (平衡数据集)

(f) 人工神经网络（平衡数据集） (g) K NN（不平衡数据集） (h) K近邻（平衡数据集） (i) 高斯朴素贝叶斯（不平衡数据集） (j) 高斯朴素贝叶斯（平衡数据集 t)

图 11：我们的模型与基线模型的性能比较。

不过，我们的模型优于基线模型。使用不平衡数据集时，F1分数高出 10%（图 11c）。使用人工神经网络（图 11e 和图 11f）及 K近邻（图 11g 和图 11h）训练模型时，我们的模型也优于基线模型。然而，如图 11j 所示，只有在使用高斯朴素贝叶斯训练模型并采用平衡数据集时，Ponzanelli 等人 [36] 的模型才优于我们的模型。其数据分布以及部分特征的独立性，或许可以解释为何他们的模型在使用高斯朴素贝叶斯时表现更佳。不过，我们使用五种工作原理各异的机器学习分类器进行分析后发现，与基线模型的特征相比，我们的特征在区分受欢迎问题和不鼓励的问题方面具有更强的判别能力。

VI. 研究结果总结

我们的研究揭示了若干可能避免问题获得反对票的见解。

请注意主题的选择。仅讨论常见主题（例如 Python、Java）的问题，可能会妨碍用户找到自己感兴趣的问题。用户通常会通过投反对票来阻止使用不恰当的主题（例如标签）。根据我们的调查，不鼓励的问题比获推荐的问题使用了更多常见主题。

代码可复用性并不总是保证高质量。只有应添加那些理解所报告问题所必需的代码语句，并应避免冗余代码。冗长且冗余的代码会浪费开发者的时间，这也可能增加获得反对票的几率。

缺少代码解释会造成不利影响。对代码示例进行清晰解释可能会影响问题质量。我们还发现，使用恰当的专业术语很重要。简短且不专业的解释可能会误导用户，使其无法理解实际问题。

情感极性并不是一个大问题。最近的一些研究报告称，情感极性（例如，积极或消极）可能会影响帖子的质量。Calefato 等人 [12]发现，回答者评论中表达的情感可能有助于估计答案的质量。根据我们的分析，情感极性对主观 SO 问题质量评估。事实上，大约 50% 的问题既没有正面情感，也没有负面情感。

VII. 相关工作

由于该问答网站日益增长的受欢迎程度和重要性，近期研究对Stack Overflow的多个方面进行了探讨。以往关于问答网站的大多数研究主要集中于评估帖子 [36, 37, 58, 43, 5, 15],理解开发者在问答网站上的互动方式 [52],为撰写高质量帖子提供实证证据 [14, 7],答案中附带的代码片段的可用性 [59, 24],以及情感极性对获得答案的影响 [12, 33]。据我们所知，这是首次尝试使用客观质量指标来考察 SO 的主观评价。

Ponzanelli 等人 [36, 37] 提出了一种根据问题质量对问题进行分类的方法。具体而言，他们识别低质量问题，以提升 SO 人工审核流程的能力。他们定义了三组指标来衡量问题质量，分别是 SO（M_{SO}）、可读性（M_R）和用户受欢迎度（M_P）。M_P 指标的属性源自对用户的直接或间接评价。然而，我们尝试使用客观质量指标评估问题质量。Ponzanelli 等人 [36] 试图改进审核流程，而我们的目标是考察对 SO 问题/答案的主观评价是否与客观评价一致。我们衡量了十个客观质量方面（例如，文本可读性、主题熵），并据此构建了五种机器学习模型，用于预测被推荐和不推荐的问题。我们的模型显著优于 Ponzanelli 等人[36]提出的模型。Calefato 等人 [15]开展了一项实证评估，以预测技术问答网站中的最佳答案。他们试图评估答案的质量，而我们则将 SO 的主观质量控制机制与客观质量指标进行对比。

多项研究 [47, 20, 16, 30, 22, 20] 将文本可读性作为客观质量评估的一部分。在本研究中，我们也计算了问题的文本可读性。

描述文本，以评估描述文本的易读性。除了文本可读性之外，正如许多研究所建议的那样，我们还测量 SO 问题中所包含代码片段的可读性 [11, 17, 38, 45, 10, 51]。SO中包含的代码片段通常是不完整且嘈杂的，因此对于两类问题，其可读性都被发现较低。正如现有研究所建议的那样，我们还测量代码片段的可理解性 [53, 29, 46]。Zhang等人 [60]报告称，大量 API 会损害代码可理解性。因此，我们统计代码片段中所使用的 API 数量，以估计代码片段的可理解性。

近期有多项研究 [59, 24] 着重评估问答网站代码片段的可用性（例如，可解析性）。Yang等人 [59]分析了来自 Stack Overflow 的 914,974 个 Java 代码片段的可用性，并报告称其中只有 3.89% 可解析。他们仅分析了 Stack Overflow 已接受答案中的代码片段。然而，我们考察了问题正文中所含代码片段的可解析性，以估计其可复用性。尽管我们发现的解析率高于 Yang等人 [59], 的结果，但来自推荐问题和不鼓励问题的代码片段，其解析率相近。

多项研究 [55, 48, 12, 14] 试图探究快速回答、高质量帖子或成功回答背后的影响因素（例如，信息呈现、问题发布时间）。我们也发现了若干影响问题质量的指标。例如，当问题主题不够具体或含糊不清时，用户往往会投反对票。此外，近期有多项研究 [28, 13, 33, 34, 26] 着重识别协作型问答网站（如 Stack Overflow）交流痕迹中蕴含的开发者情绪（例如，积极或消极情绪）。Novielli等人 [33] 指出，技术问题的情感风格确实会影响其能否及时获得令人满意的答案。然而，我们考察了情感极性是否会影响 Stack Overflow 问题的质量。结果表明，情感极性对问题质量没有明显影响。

VIII. 效度威胁

外部效度威胁 关系到结果的普遍适用性。我们分析了约 250 万个 SO 问题。然而，将我们尝试的方法应用于不同类型的编程问题时，结果可能会有所不同。为缓解这一问题，我们选择在数据集中纳入与四种不同编程语言相关的问题：Java、C#、JavaScript 和 Python。其中两种是动态类型语言，另外两种是静态类型语言。通过评估我们的方法，并使用来自四种不同语言的问题，可以衡量这一威胁的影响。问题质量分类模型可能会受到数据集的影响。因此，我们构建了五种不同的分类模型，并分别分析其性能。

对内部效度构成威胁的因素与实验误差和偏差 [50]有关。评估系统威胁到了 SO问题的质量评估。我们使用现有的工具和方法，评估问题所有组成部分（例如标题、正文、标签）的客观质量。然后，我们研究客观评价与 Stack Overflow 上的主观评价如何保持一致。然而，当结果似乎不一致时，我们会人工分析相关问题。这种混合方法缓解了这一威胁。尽管如此，我们承认所用指标可能并非最佳，改进各项指标可能会改变结果。因此，我们的研究可以通过选取更合适的指标或进行后续人工分析来进一步拓展。

统计结论威胁涉及数据是否足以支持相关论断这一问题。在所有结果分析中，我们都考虑了具有统计显著性的样本。我们依靠群体评估收集数据，而非人工检查问题，从而能够收集足够的数据。

另一个问题是，推荐问题和不受欢迎问题的样本分布不平衡。对于所有编程问题，推荐样本的数量都多于不受欢迎样本。为了解决这种不平衡情况，我们对推荐问题进行欠采样，以平衡数据集。随后，我们分析平衡数据集和不平衡数据集的结果，从而缓解这一问题。

IX. 结论

像 Stack Overflow 这样的问答网站为程序员参与社会化学习提供了一种新途径。Stack Overflow 帖子的质量由用户通过投票进行评估。为确保评估过程严谨可靠，我们尝试考察用户对问题的主观评价与客观质量评估之间的一致性。我们使用十项客观指标来涵盖 Stack Overflow 问题客观质量的所有关键方面。我们发现，主题熵、指标熵、文本-代码比和文本-代码相关性与主观评价完全一致。代码片段的可复用性（即可解析性）和文本可读性则与主观评价完全不一致。然而，代码可理解性是否一致取决于编程语言；而标题质量、代码可读性和情感极性在推荐问题和不受欢迎问题中的结果几乎相同。我们的研究可为进一步开展主观性与客观性之间的一致性分析提供初步基准。

理解和分类问题质量对于维持问答服务良好的用户体验至关重要。低质量问题被认为对在线问答网站的用处较小。我们开发了五种不同的机器学习模型，用于分类被推荐和被劝阻的问题。我们的机器学习模型能够以约 76%—87% 的最高准确率对问题进行分类，这显著高于最先进模型。

致谢：本研究部分得到加拿大自然科学与工程研究委员会（NSERC）探索项目基金、NSERC 协作研究与培训体验（CREATE）项目基金，以及由全球粮食安全研究所（GIFS）和全球水安全研究所（GIWS）协调的两项加拿大首个研究卓越基金（CFREF）项目资助。

参考文献

[1] StackExchage API，访问日期：2018年4月。网址：http://data. stackexchange.com/stackoverflow。 [2] 可读性，访问日期：2018年4月。网址：https://en. wikipedia.org/wiki/Readability。 [3] N. S. Altman. 核方法与最近邻非参数回归简介。《美国统计学家》，46(3)：175–185，1992。 [4] J. Anderson. Lix 和 rix：鲜为人知的可读性指数的变体。Journal of Reading，26(6)：490–496，1983。 [5] A. Baltadzhieva 和 G. Chrupała. 预测 Stack Overflow 上问题的质量。载于 RANLP 论文集，第 32–40 页，2015 年。[6] Stefanie Beyer、Christian Macho、Massimiliano Di Penta 和 Martin Pinzger. 自动将帖子分类到 Stack Overflow 的问题类别中。载于 2018 IEEE/ACM 第 26 届国际程序理解会议（ICPC），第 211–21110 页。IEEE，2018。 [7] A. Bosu、C. S. Corley、D. Heaton、D. Chatterji、J. C. Carver 和 N. A. Kraft. 在 Stack Overflow 上建立声誉：一项实证调查。载于 MSR 论文集，第 89–92 页，2013。 [8] L. Breiman. 随机森林。机器学习，45(1)：32–5，2001。[9] L. Breiman. 分类与回归树。Routledge，2017。[10] R. P. L. Buse 和 W. R. Weimer. 软件可读性度量。载于 ISSTA 论文集，第 121–130 页，2008。 [11] R. PL Buse 和 W. R. Weimer. 学习代码可读性度量。TSE，36(4)：546–558，2010。 [12] F. Calefato、F. Lanubile、M. C. Marasciulo 和 N. Novielli. 挖掘 Stack Overflow 上的成功回答。载于 MSR 论文集，第 430–433 页，2015。 [13] F. Calefato、F. Lanubile、F. Maiorano 和 N. Novielli. 软件开发中的情感极性检测。ESE，23(3)：1352–1382，2018。 [14] F. Calefato、F. Lanubile 和 N. Novielli. 如何寻求技术帮助？在 Stack Overflow 上撰写问题的循证指南。IST，94：186–207，2018。 [15] F. Calefato、F. Lanubile 和 N. Novielli. 对技术问答网站最佳答案预测模型的实证评估。ESE，第 1–48 页，2019。[16] M. Coleman 和 T. L. Liau. 一种专为机器评分设计的计算机可读性公式。《应用心理学杂志》，60(2)：283，1975。 [17] E. Daka、J. Campos、G. Fraser、J. Dorn 和 W. Weimer. 建模可读性以改进单元测试。载于 FSE 论文集，第 107–118 页，2015。 [18] M. Duijn、A. Kucera 和 A. Bacchelli. 高质量问题需要高质量代码：对 Stack Overflow 上的代码片段进行分类。载于 MSR 论文集，第 410–413 页，2015。 [19] S. Ercan、Q. Stokkink 和 A. Bacchelli. 自动评估代码解释：预测 Stack Overflow 上的回答时间。载于 MSR 论文集，第 442–445 页，2015。 [20] Rudolph Flesch. 一种新的可读性衡量标准。《应用心理学杂志》，32(3)：221，1948。 [21] K. Ganesan. Rouge 2.0：用于评估摘要任务的更新和改进指标。arXiv 预印本 arXiv:1803.01937，2018。 [22] R. Gunning. 清晰写作技巧。1952。 [23] T. Hall、S. Beecham、D. Bowes、D. Gray 和 S. Counsell. 关于软件工程中故障预测性能的系统性文献综述。TSE，38(6)：1276–1304，2012。 [24] E. Horton 和 C. Parnin. Gistable：评估 GitHub 上 Python 代码片段的可执行性。载于 ICSME 论文集，第 217–227 页，2018。 [25] G. Hsieh、R. E. Kraut 和 S. E. Hudson. 为什么要付费？探索如何在问答中使用金钱激励。载于 CHI 会议论文集，第 305–314 页，2010 年。[26] M. R. Islam 和 M. F. Zibran。将自动情感分析应用于软件工程。载于 MSR 会议论文集，第 203–214 页，2017 年。[27] S. TK Jan、C. Wang、Q. Zhang 和 G. Wang。探索社交问答服务中的金钱激励。arXiv 预印本 arXiv:1703.01333，2017 年。[28] B. Lin、F. Zampetti、G. Bavota、M. Di Penta、M. Lanza 和 R. Oliveto。软件工程的情感分析：我们能走多远？载于 ICSE 会议论文集，2018 年。[29] J. C. Lin 和 K. C. Wu。基于模糊矩阵的软件可理解性评估。载于 WCCI 会议论文集，第 887–892 页，2008 年。[30] G. Harry M. Laughlin。烟雾分级——一种新的可读性公式。《Journal of Reading》，12(8):639–646，1969 年。[31] S. Mondal、M. M. Rahman 和 C. K. Roy。复现包，2023 年。网址：https://bit.ly/3YKRczm。[32] S. M. Nasehi、J. Sillito、F. Maurer 和 C. Burns。什么造就了优秀的代码示例——Stack Overflow 编程问答研究。载于 ICSM 会议论文集，第 25–35 页，2012 年。[33] N. Novielli、F. Calefato 和 F. Lanubile。探索情绪在 Stack Overflow 中的作用。载于 SSE 会议论文集，第 33–36 页，2014 年。[34] N. Novielli、D. Girardi 和 F. Lanubile。软件工程研究中的情感分析基准研究。arXiv 预印本 arXiv:1803.06525，2018 年。[35] F. Pedregosa、G. Varoquaux、A. Gramfort、V. Michel、B. Thirion、O. Grisel、M. Blondel、P. Prettenhofer、R. Weiss、V. Dubourg、J. Vanderplas、A. Passos、D. Cournapeau、M. Brucher、M. Perrot 和 E. Duchesnay。Scikit-learn：Python 中的机器学习。《Journal of Machine Learning Research》，12:2825–2830，2011 年。[36] L. Ponzanelli、A. Mocci、A. Bacchelli 和 M. Lanza。理解并分类技术论坛问题的质量。载于 QSIC 会议论文集，第 343–352 页，2014 年。[37] L. Ponzanelli、A. Mocci、A. Bacchelli、M. Lanza 和 D. Fullerton。改进低质量 Stack Overflow 帖子检测。载于 ICSME 会议论文集，第 541–544 页，2014 年。[38] D. Posnett、A. Hindle 和 P. Devanbu。一种更简单的软件可读性模型。载于 MSR 会议论文集，第 73–82 页，2011 年。[39] J. Ross Quinlan。决策树归纳。《Machine Learning》，1(1):81–106，1986 年。[40] M. M. Rahman 和 C. K. Roy。深入探究 Stack Overflow 上未解决的问题。载于 MSR 会议论文集，第 426–429 页，2015 年。[41] J. D. Rennie、L. Shih、J. Teevan 和 D. R. Karger。解决朴素贝叶斯文本分类器中的不合理假设。载于 ICML 会议论文集，第 616–623 页，2003 年。[42] Irina Rish 等。朴素贝叶斯分类器的实证研究。载于 IJCAI 2001 人工智能实证方法研讨会，第 3 卷，第 41–46 页，2001 年。[43] P. K. Roy、Z. Ahmad、J. P. Singh、M. A. A. Alryalat、N. P. Rana 和 Y. K. Dwivedi。在社区问答网站中查找并排列高质量答案。《Global Journal of Flexible Systems Management》，19(1):53–68，2018 年。[44] R. K. Saha、A. K. Saha 和 D. E. Perry。探究软件信息网站上未回答问题的成因：Stack Overflow 案例研究。载于 FSE 会议论文集，第 663–666 页，2013 年。[45] S. Scalabrino、M. Linares-Vásquez、D. Poshyvanyk 和 R. Oliveto。利用文本特征改进代码可读性模型。载于 ICPC 会议论文集，第 1–10 页，2016 年。[46] S. Scalabrino、G. Bavota、C. Vendome、M. Linares-Vásquez、D. Poshyvanyk 和 R. Oliveto。自动评估代码可理解性：我们进展到什么程度了？载于 ASE 会议论文集，第 417–427 页，2017 年。[47] E. A Smith 和 RJ Senter。自动可读性指数。AMRL-

TR. 航空航天医学研究实验室（美国），第 1–14 页，1967 年。[48] M. Squire 和 C. Funkhouser。“一段代码”：Stack Overflow 社区如何创造高质量帖子。载于 *HICSS* 会议论文集，第 1425–1434 页，2014 年。[49] J. A. Thomas 和 T.M. Cover。《信息论基础》。*Wiley- Interscience*，第 187–202 页，1991 年。[50] Y. Tian、D. Lo 和 J. Lawall。自动构建软件专用词语相似度数据库。载于 *CSMR- WCRE* 会议论文集，第 44–53 页，2014 年。[51] C. Treude 和 M. P. Robillard。理解 Stack Overflow 代码片段。载于 *ICSME* 会议论文集，第 509–513 页，2017 年。[52] C. Treude、O. Barzilay 和 M.A. Storey。程序员如何在网络上提问和回答问题？（NIER 专题）。载于 *ICSE* 会议论文集，第 804–807 页，2011 年。[53] A. Trockman、K. Cates、M. Mozina、T. Nguyen、C. Kästner 和 B. Vasilescu。重新分析“自动评估代码可理解性”：组合指标至关重要。第 314–318 页，2018 年。[54] S. Wang、T. H. P. Chen 和 A. E. Hassan。用户如何修改技术问答网站上的答案？以 Stack overflow。*TSE*，2018。[55] S. Wang、T.H. Chen 和 A. E. Hassan。理解技术问答网站中快速回答的影响因素。*ESE*，23(3)：1552–1593，2018。[56] Geoffrey I Webb、Ea monn Keogh 和 Risto Miikkulainen。朴素贝叶斯。机器学习百科全书，15 714，2010 713：–。[57] P. J. Werbos。时间反向传播：其作用及实现方法。*IEEE* 会议录，78(10)：1550–1560，1990。[58] X. Xia、D. Lo、D. Correa、A. Sureka 和 E. Shihab。双人共舞：基于文本和元特征的 Stack Overflow 已删除问题预测。发表于*COMPSAC*，第 1 卷，82，2016 73页–。[59] D. Yang、A. Hussain 和 C. V. Lopes。从查询到可用代码：Stack Overflow 代码片段分析。发表于*Proc. MSR*，第 391–402 页，2016。[60] H. Zhang、S. Wang、TH P. Chen、Y. Zou 和 A. E. Hassan。Stack Overflow 上过时答案的实证研究。*IEEE* 软件工程汇刊，2019。