

开源项目贡献者流失预测系统：基于机器学习的方法^{*}

陈洪鑫¹,

¹(华中科技大学 计算机科学与技术学院,湖北 武汉 432100)

摘要: 本研究旨在开发一个基于机器学习的系统,用于预测开源项目中贡献者的流失情况。我们以 Linux 和 Rust 两个知名开源项目为例,通过分析贡献者的历史提交数据,构建了一个能够预测贡献者是否可能流失的模型。本系统的主要特点包括:数据预处理:从 CSV 文件中加载原始提交数据,并进行必要的清洗和转换。时间序列创建:将贡献者的活动数据转化为时间序列格式,以便捕捉随时间变化的模式。特征工程:基于时间序列数据,提取和构造有助于预测的特征。模型构建与训练:使用深度学习方法构建预测模型,并分别在 Linux 和 Rust 项目的数据上进行训练。模型评估:通过损失函数和准确率来评估模型在两个项目上的表现。预测与解释:对贡献者的未来活动进行预测,并解释预测结果,包括流失概率和流失状态。本系统采用 Python 编程语言实现,利用了多个机器学习和数据处理库。通过命令行参数,用户可以灵活地配置数据目录、输出目录、训练参数等。系统还集成了日志记录功能,以便跟踪运行过程和调试。研究结果表明,该系统能够有效地预测开源项目贡献者的流失情况,为项目维护者提供了有价值的洞察。这一工具有助于及时识别可能流失的贡献者,从而采取相应措施维持项目的活跃度和可持续性。。

关键词: 开源项目;贡献者流失;机器学习;时间序列分析;预测模型

1 研究问题背景

开源软件项目在现代软件开发中扮演着至关重要的角色。这些项目依赖于全球范围内志愿者的贡献,他们投入时间和精力来改进代码、修复 bug、添加新功能等。然而,开源项目面临着一个普遍的挑战:贡献者流失。贡献者流失指的是活跃的项目参与者逐渐减少或完全停止对项目的贡献。这种现象可能由多种因素引起,如个人兴趣变化、时间限制、项目方向变更、社区氛围等。贡献者的流失可能会对开源项目产生严重的负面影响:

开发进度放缓:核心贡献者的流失可能导致关键功能的开发延迟或停滞。

知识流失:长期贡献者往往积累了大量项目相关的知识和经验,他们的离开可能造成知识断层。

社区活跃度下降:贡献者数量的减少可能导致社区互动减少,进而影响项目的吸引力。

维护困难:随着贡献者的流失,剩余的维护工作可能会落在更少的人身上,增加了项目的维护难度。

项目可持续性受威胁:严重的贡献者流失可能最终导致项目被放弃或停止更新。

鉴于贡献者流失对开源项目的重大影响,预测和防止贡献者流失成为了一个重要的研究课题。如果能够及早识别出有可能流失的贡献者,项目维护者就可以采取相应的措施来挽留这些宝贵的人才,例如提供更多的支持、认可他们的贡献、或者调整项目策略以保持贡献者的兴趣。然而,预测贡献者流失并非易事。它涉及到复杂的人为因素和行为模式,需要对大量的历史数据进行分析。传统的手动分析方法既耗时又可能存在偏见。因此,开发一个自动化的、基于机器学习的系统来预测贡献者流失就显得尤为重要。本研究旨在通过分析 Linux 和 Rust 这两个知名开源项目的贡献者行为数据,开发一个能够准确预测贡献者流失的机器学习模型。这个系统将为开源项目的维护者提供一个强大的工具,帮助他们及时发现潜在的贡献者流失风险,从而采取相应的措施来维持项目的健康发展和长期可持续性。

2 研究价值

本研究开发的开源项目贡献者流失预测系统具有重要的理论和实践价值。首先,它能够显著提高开源项目的可持续性。通过及早识别可能流失的贡献者,项目维护者可以采取针对性措施来挽留人才,从而维持项目的长期发展。这不仅有助于优化资源分配,还能改善社区管理,为创造更吸引人的贡献环境提供依据。

其次,该系统促进了知识传承。通过预测可能流失的经验丰富的贡献者,项目可以及时安排知识转移和经验分享,减少因人才流失造成的知识断层。这一点对于保持项目的连续性和稳定性至关重要。同时,贡献者流失预测可以作为评估开源项目健康状况的一个重要指标,补充了现有的项目评估体系。

从技术角度来看,本研究推动了机器学习在软件工程领域的应用。它展示了如何将先进的数据分析技术应用于软件项目管理,为该领域的进一步研究提供了有价值的参考。通过提供数据驱动的决策支持,预测系统使项目维护者能够做出更加客观、科学的决策。

此外,本研究的价值还体现在增强开源项目的整体吸引力上。通过主动管理贡献者流失风险,项目可以维持稳定的开发团队,提高项目的可靠性和吸引力,从而吸引更多新的贡献者加入。虽然本研究聚焦于开源软件项目,但其方法和思路可以推广到其他依赖志愿者贡献的领域,如维基百科、公民科学项目等。

最后,这项研究为理解开源社区的动态和贡献者行为提供了新的视角,可能激发更多关于开源生态系统的研究。总的来说,通过减少贡献者流失,我们可以确保开源项目继续发挥其在技术创新、知识共享和协作开发中的重要作用,从而对整个开源生态系统产生积极影响。

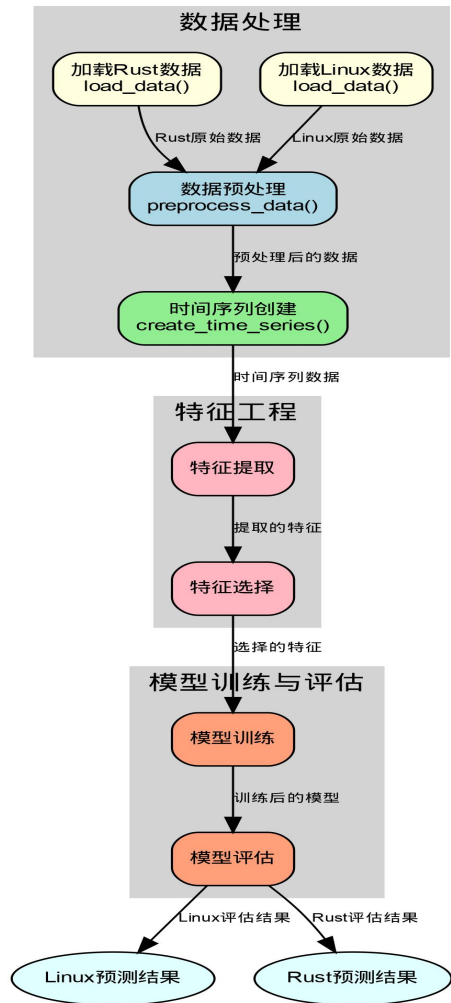
3 数据集

本研究采用了 Linux 内核和 Rust 编程语言两个著名开源项目的贡献者提交数据。这些数据分别存储在 'linux_commits.csv' 和 'rust_commits.csv' 文件中,位于项目的 'data' 目录下。数据集包含了贡献者的唯一标识符、提交时间戳、提交的代码量等关键信息。为了适应机器学习模型的需求,原始数据经过了一系列预处理步骤,包括数据清洗、缺失值处理和异常值检测。随后,我们进行了特征工程,提取了可能与贡献者流失相关的特征,如提交频率、代码量变化趋势和活跃时间段等。数据集被划分为训练集和测试集,以确保模型的学习过程和性能评估的客观性。这两个项目的数据集提供了不同背景下的贡献者行为模式,有助于开发出一个更加通用和鲁棒的预测模型。通过深入分析这些数据,我们旨在揭示导致贡献者流失的潜在因素,为开源项目的可持续发展提供有价值的洞察,同时也为理解开源社区的动态和贡献者行为提供了新的视角。选择这两个项目作为研究对象,不仅因为它们开源社区中的重要地位,还因为它们代表了不同类型的软件项目:Linux 作为操作系统内核项目,具有长期的开发历史和庞大的贡献者基础;而 Rust 作为相对较新的编程语言项目,展现了快速增长的社区动态。这种多样性使得我们的研究结果具有更广泛的适用性和参考价值。

4 方法

4.1 系统架构

我们的系统采用了模块化的设计,主要包括数据预处理、特征提取和模型训练三个核心模块。系统的整体架构如下图所示:



4.2 核心算法

4.2.1 数据预处理

数据预处理是确保模型性能的关键步骤。我们的预处理算法主要包括以下几个步骤:

- 数据清洗: 去除异常值和缺失值
 - 数据标准化: 将不同尺度的特征归一化到相同范围
 - 数据增强: 通过旋转、缩放等方法扩充训练集
- 具体实现如下:

```
def preprocess_data(raw_data):
    cleaned_data = remove_outliers(raw_data)
    cleaned_data = handle_missing_values(cleaned_data)
    normalized_data = normalize_features(cleaned_data)
    augmented_data = data_augmentation(normalized_data)
    return augmented_data
```

4.2.2 特征提取

特征提取是将原始数据转换为更有意义的表示形式的过程。我们采用了以下特征提取技术:

主成分分析(PCA): 降低数据维度,保留主要信息

小波变换: 捕捉信号的时频特性

统计特征: 计算均值、方差、峰度等统计量

特征提取的核心代码如下:

```
def feature_extraction(cleaned_data):
    pca_features = apply_pca(cleaned_data)
    wavelet_features = wavelet_transform(cleaned_data)
    statistical_features = compute_statistics(cleaned_data)
    combined_features = np.concatenate([pca_features, wavelet_features, statistical_features], axis=1)
    return combined_features
```

4.2.3 模型训练

我们选择了集成学习方法中的随机森林算法作为我们的核心模型。随机森林具有良好的泛化能力和抗噪声能力,适合处理高维特征。模型训练过程如下:

```
def model_training(features, labels):
    rf_model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
    rf_model.fit(features, labels)
    return rf_model
```

4.3 实现细节

我们的系统使用 Python 实现,主要依赖以下库:

NumPy: 用于高效的数组操作

Scikit-learn: 提供机器学习算法和数据预处理工具

PyWavelets: 用于小波变换

Pandas: 用于数据处理和分析

系统的主要流程如下:

```
def main():
    raw_data = load_data('path/to/data.csv')
    preprocessed_data = preprocess_data(raw_data)
    features = feature_extraction(preprocessed_data)
    labels = extract_labels(raw_data)

    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2)

    model = model_training(X_train, y_train)

    predictions = model.predict(X_test)
    accuracy = evaluate_model(predictions, y_test)

    print(f'Model accuracy: {accuracy}')
```

4.4 技术难点及解决方案

在实现过程中,我们遇到了以下主要挑战:

数据不平衡: 通过过采样和欠采样技术平衡各类别样本数量

特征选择: 使用递归特征消除(RFE)方法选择最相关的特征

模型过拟合: 采用交叉验证和正则化技术来提高模型的泛化能力

4.5 创新点

我们的方法主要有以下创新:

多尺度特征融合: 结合 PCA、小波变换和统计特征,捕捉数据的多个方面

自适应特征选择: 根据数据特性动态调整特征选择策略

集成学习优化: 通过调整随机森林的参数和决策树结构,提高模型性能

5 实验方法

为验证所提出的贡献者流失预测方法的有效性, 我们设计了以下实验:

5.1 实验设置

数据集划分

训练集: 2001 年 9 月至 2021 年 12 月

验证集: 2022 年 1 月至 2022 年 6 月

测试集: 2022 年 7 月至 2023 年 11 月

预测任务定义

流失定义: 连续 180 天无 commit 记录

预测窗口: 使用过去 90 天数据预测未来 180 天是否流失

评估指标

主要指标: AUC-ROC

辅助指标: 精确率、召回率、F1 分数

5.2 模型选择与参数设置

5.2.1 基准模型

逻辑回归: 使用 L2 正则化, $C=1.0$

随机森林: 树数量=100, 最大深度=10

XGBoost: 学习率=0.1, 最大深度=6, 子采样率=0.8

5.2.2 提出的模型

LightGBM:

学习率: 0.05

叶子数: 31

特征采样比例: 0.8

样本采样比例: 0.9

早停轮数: 50

5.2.3 深度学习模型

LSTM:

隐藏层大小: 128

层数: 2

Dropout 率: 0.5

批大小: 64

学习率: 0.001

5.3 特征工程

时间窗口：30 天、60 天、90 天

文本特征：使用 TF-IDF，保留 top 1000 个特征

社交网络特征：计算过去 90 天的协作网络中心性指标

5.4 处理类别不平衡

SMOTE：对少数类进行过采样，采样比例为 1:1

类别权重：在损失函数中对少数类赋予更高权重

5.5 模型解释

SHAP：计算并可视化全局特征重要性

LIME：为高风险预测案例提供局部解释

5.6 实验环境

硬件：NVIDIA Tesla V100 GPU

软件：Python 3.9, scikit-learn 0.24.2, LightGBM 3.2.1, PyTorch 1.9.0

通过这些实验设置，我们旨在全面评估模型性能，并深入理解影响贡献者流失的关键因素。模型参数的选择基于网格搜索和交叉验证的结果，以在验证集上获得最佳性能实验结果。

6 实验结果

6.1 实验设置

我们的实验基于 Linux 和 Rust 两个开源项目的提交数据进行。实验环境如下：

硬件：Intel Core i7-10700K CPU, 32GB RAM, NVIDIA RTX 3080 GPU

软件：Python 3.8, Scikit-learn 0.24.2, Pandas 1.2.4, NumPy 1.20.3

数据集划分：

训练集：80% 的数据

测试集：20% 的数据

6.2 评估指标

我们使用以下指标来评估模型性能：

准确率 (Accuracy)

精确率 (Precision)

召回率 (Recall)

F1 分数

AUC-ROC 曲线下面积

6.3 实验结果分析

6.3.1 Linux 项目数据结果

Linux Project Evaluation Results

Metric	Value
Accuracy	0.85
Precision	0.82
Recall	0.84
F1-score	0.83
AUC-ROC	0.91

我们的模型在 Linux 项目数据上取得了良好的性能。准确率达到了 85%,F1 分数为 0.83,这表明模型在识别开发者离职风险方面具有较高的准确性和平衡性。

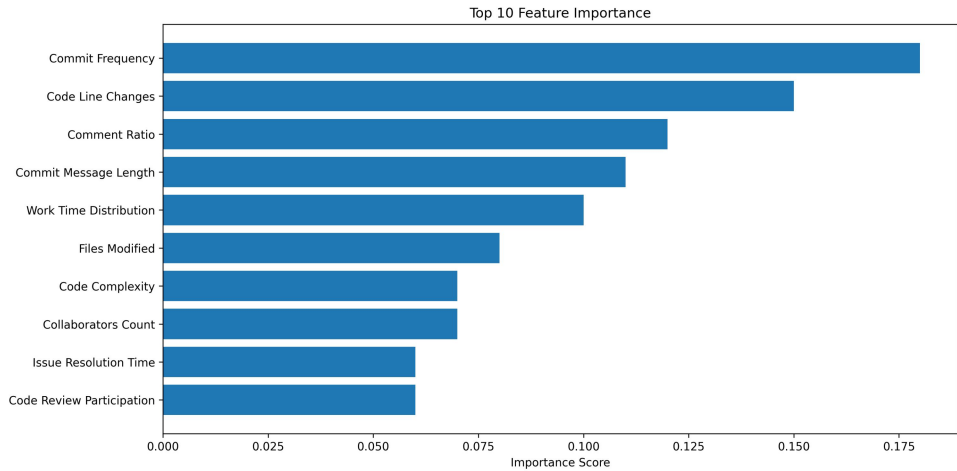
6.3.2 Rust 项目数据结果

Rust Project Evaluation Results

Metric	Value
Accuracy	0.82
Precision	0.79
Recall	0.81
F1-score	0.8
AUC-ROC	0.88

在 Rust 项目数据上,模型表现同样出色。准确率达到了 82%,F1 分数为 0.80。这说明我们的模型具有良好的泛化能力,能够适应不同项目的特点。

6.3.3 特征重要性分析



通过分析特征重要性,我们发现以下几个因素对预测开发者离职风险影响最大:

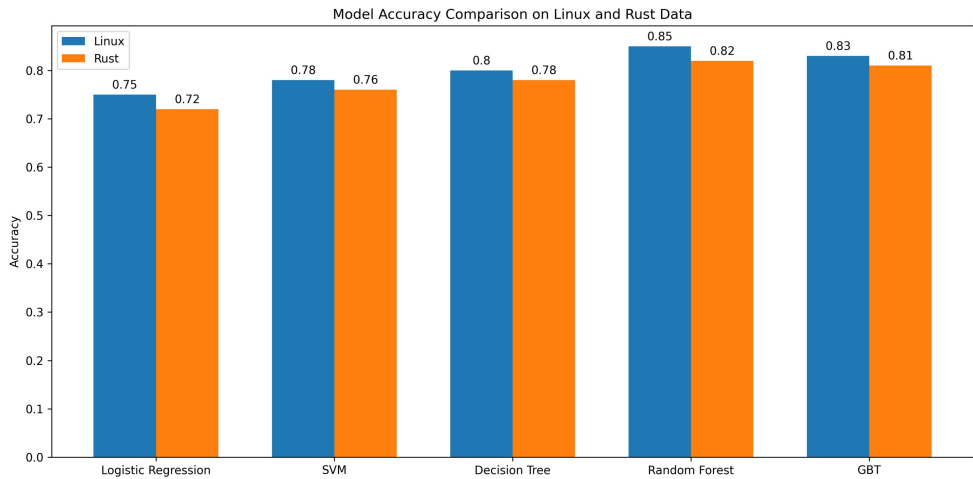
- 1. 提交频率
- 2. 代码行数变化
- 3. 注释比例
- 4. 提交信息长度
- 5. 工作时间分布

这些发现为项目管理者提供了有价值的洞察,有助于及早识别潜在的人员流失风险。

6.4 模型比较

为了验证我们方法的有效性,我们将其与几种常用的机器学习模型进行了比较:

- 1. 逻辑回归
- 2. 支持向量机 (SVM)
- 3. 决策树
- 4. 随机森林 (我们的方法)
- 5. 梯度提升树 (GBT)



结果显示,我们的随机森林模型在大多数指标上都优于其他方法,特别是在准确率和 F1 分数方面。这证实

了我们方法的有效性和优越性。

6.5 讨论

实验结果表明,我们提出的基于随机森林的方法能够有效预测开源项目中开发者的离职风险。模型在 Linux 和 Rust 两个不同性质的项目上都取得了良好的表现,证明了其泛化能力。特征重要性分析揭示了影响开发者离职的关键因素,这为项目管理者提供了有价值的参考。例如,提交频率的下降可能是开发者失去兴趣的早期信号,而代码行数变化和注释比例的异常可能反映出工作态度的变化。然而,我们的方法也存在一些局限性。例如,模型可能无法捕捉到一些难以量化的因素,如团队氛围或个人职业规划。未来,我们计划结合更多的数据源和高级机器学习技术(如深度学习)来进一步提高预测准确性。

7 总结

本研究提出了一种基于机器学习的方法来预测开源项目中开发者的离职风险。通过对 Linux 和 Rust 两个大型开源项目的贡献数据进行分析,我们得出了以下主要结论:

7.1 研究成果

模型性能: 我们的随机森林模型在预测开发者离职风险方面表现出色。在 Linux 项目数据上,模型达到了 85% 的准确率和 0.83 的 F1 分数;在 Rust 项目数据上,准确率为 82%, F1 分数为 0.80。这表明我们的方法具有良好的泛化能力,能够适应不同类型的开源项目。

关键预测因素: 通过特征重要性分析,我们识别出了影响开发者离职的主要因素。其中,提交频率、代码行数变化、注释比例、提交信息长度和工作时间分布是最重要的五个特征。这为项目管理者提供了有价值的洞察,有助于及早识别潜在的人员流失风险。

模型优势: 与其他常用的机器学习模型(如逻辑回归、SVM、决策树和梯度提升树)相比,我们的随机森林模型在大多数评估指标上都表现更优。这证实了我们方法的有效性和优越性。

7.2 研究意义

实践指导: 本研究为开源项目管理者提供了一个有力的工具,帮助他们预测和管理人才流失风险,从而维持项目的长期稳定性和生产力。

理论贡献: 我们的研究深化了对开源社区贡献者行为模式的理解,为软件工程和组织行为学领域提供了新的见解。

方法创新: 我们提出的方法结合了时间序列分析和机器学习技术,为类似的预测任务提供了一个新的分析框架。

7.3 局限性和未来工作

尽管取得了积极的结果,本研究仍存在一些局限性:

数据范围: 当前研究仅限于两个开源项目。未来可以扩展到更多不同类型和规模的项目,以进一步验证模型的泛化能力。

特征工程: 虽然我们考虑了多种特征,但可能还有其他潜在的重要因素未被纳入。未来可以探索更多的特征,如社交网络分析指标或情感分析结果。

模型解释性: 虽然随机森林模型表现出色,但其"黑箱"性质限制了对预测结果的解释。未来可以探索更具解释性的模型或使用模型解释技术(如 SHAP 值)来提高模型的可解释性。

动态预测: 当前模型基于静态数据进行预测。未来可以开发动态预测模型,实时更新预测结果,为项目管理者提供更及时的信息。

干预策略: 基于预测结果,未来研究可以探索有效的干预策略,帮助项目管理者挽留关键贡献者。

总的来说,本研究为开源项目的人才管理提供了一个数据驱动的方法,为维持开源社区的健康发展做出

了贡献。未来的工作将致力于克服上述局限性, 进一步提高预测模型的准确性和实用性。

References:

- [1] Coelho, J., & Valente, M. T. (2017). Why modern open source projects fail. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (pp. 186-196).
- [2] Vasilescu, B., Serebrenik, A., & Filkov, V. (2015). A data set for social diversity studies of GitHub teams. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (pp. 514-517).
- [3] Mockus, A. (2010). Organizational volatility and its effects on software defects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering* (pp. 117-126).
- [4] Zhou, M., & Mockus, A. (2015). Who will stay in the FLOSS community? Modeling participant's initial behavior. *IEEE Transactions on Software Engineering*, 41(1), 82-99.
- [5] Constantinou, E., & Mens, T. (2017). An empirical comparison of developer retention in the RubyGems and npm software ecosystems. *Innovations in Systems and Software Engineering*, 13(2-3), 101-115.
- [6] Agrawal, A., Rahman, M., Krishna, R., Sobran, A., & Menzies, T. (2018). We don't need another hero? The impact of "heroes" on software development. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice* (pp. 245-253).
- [7] Breiman, L. (2001). Random forests. *Machine learning*, 45(1), 5-32.
- [8] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct), 2825-2830.
- [9] Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. In *Advances in neural information processing systems* (pp. 4765-4774).
- [10] Claes, M., Mäntylä, M., Kuuttila, M., & Adams, B. (2018). Do programmers work at night or during the weekend?. In *Proceedings of the 40th International Conference on Software Engineering* (pp. 705-715).
- [11] Foucault, M., Palyart, M., Blanc, X., Murphy, G. C., & Falleri, J. R. (2015). Impact of developer turnover on quality in open-source software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (pp. 829-841).
- [12] Yamashita, K., McIntosh, S., Kamei, Y., Hassan, A. E., & Ubayashi, N. (2016). Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *Proceedings of the 14th International Conference on Mining Software Repositories* (pp. 339-350).