# Classification of GitHub Contributors as Active or Inactive Using Machine Learning

## Abstract

This project aims to classify GitHub contributors into “active” and “inactive” categories based on their contribution histories. By collecting data from GitHub’s API and applying supervised machine learning techniques, we developed models to predict contributor activity levels. The report outlines the problem addressed, data collection methodology, machine learning approach, validation results, and suggestions for future improvements.

## 1. Introduction

### 1.1 Overview of the Problem

Open-source software projects rely heavily on the contributions of individual developers. Identifying active contributors is crucial for project maintainers to engage the community effectively, allocate resources, and plan future development. This project seeks to automate the classification of contributors into “active” and “inactive” categories based on quantitative metrics derived from their contribution history.

## 2. Data Collection Methodology

The data was collected from 4 GitHub open source repositories osu, tensorflow, blender, mindustry using GitHub REST API v3. A personal access token was used to authenticate requests to the GitHub API to increase rate limits. I fetched a list of the first 10 pages of contributors to the repository. I have collected the most recent 100 commits per contributor to manage API usage and data size. I extracted commit statistics including additions and deletions. I collected data on pull requests and issues authored by each contributor.

## 3. Data Cleaning and Preprocessing

I dropped email, bio, and is\_bot columns as they were either empty or irrelevant. I removed contributors with zero activity across all key metrics (total\_commits, total\_additions, total\_deletions, total\_pull\_requests, total\_issues). I converted first\_contribution and last\_contribution strings to datetime objects. I have created a couple of new column using the extracted attributes like contribution\_span\_days as the difference between last\_contribution and first\_contribution, avg\_commits\_per\_day, avg\_additions\_per\_commit, additions\_deletions\_ratio, and code\_changes\_per\_commit.

## 4. Machine Learning Approach and Implementation Details

### 4.1 Labeling Strategy

I have created two labels:

**Active**: Contributors with an average of more than 1 commit per day **Inactive**: Contributors with an average of 1 or fewer commits per day.

### 4.2 Feature Selection

* **Selected Features**:
  + total\_commits
  + total\_additions
  + total\_deletions
  + total\_pull\_requests
  + total\_issues
  + avg\_additions\_per\_commit
  + avg\_deletions\_per\_commit
  + additions\_deletions\_ratio
  + code\_changes\_per\_commit
* **Excluded Features**: Omitted username, first\_contribution, and last\_contribution from modeling to focus on quantitative metrics. An removing avg\_commits\_per\_day as this was used to create the labels.

### 4.3 Data Splitting

* **Training and Testing Sets**:
  + **Training Set**: 80% of the data.
  + **Testing Set**: 20% of the data.
  + **Stratification**: Ensured that both sets have a representative distribution of active and inactive contributors.

The dataset had an obvious problem: there way too many users labeled as inactive than active. Below you can see the histogram by label. There are 761 users labeled as inactive and 152 labeled as active.

A graph with a blue rectangle and a red rectangle

Description automatically generated

### 4.4 Model Selection

I chose logistic regression because it is specifically designed for binary classification problems where the target variable has two classes—in my case, “active” (1) and “inactive” (0) contributors.

### 4.5 Model Implementation

#### **Logistic Regression**

I used the scikit-learn library to implement logistic regression for my classification task. Before training the model, I standardized the features using StandardScaler from the sklearn.preprocessing module. This step was crucial because it ensured that all features were on the same scale, which can improve the performance and convergence speed of the logistic regression algorithm.

To address the class imbalance in my dataset, I configured the model with class\_weight='balanced'. This parameter adjusts the weights inversely proportional to class frequencies, allowing the model to pay more attention to the minority class. I also set max\_iter=1000 to provide the algorithm with sufficient iterations to converge to an optimal solution.

## 5. Validation Results and User Feedback

### Logistic Regression Results

After training the logistic regression model on the standardized features, I evaluated its performance on the testing dataset. The results provided insights into the model’s effectiveness in classifying contributors as active or inactive.

#### Classification Report

I generated a classification report to assess precision, recall, F1-score, and support for each class:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1-score | Support |
| 0 | 0.99 | 0.69 | 0.82 | 153 |
| 1 | 0.38 | 0.97 | 0.55 | 30 |
| Accuracy |  |  | 0.74 | 183 |

* **Precision for Class 0 (Inactive)**: 99% - indicating that when the model predicts a contributor is inactive, it is correct 99% of the time.
* **Recall for Class 0**: 69% - the model correctly identified 69% of the actual inactive contributors.
* **Precision for Class 1 (Active)**: 38% - when the model predicts a contributor is active, it is correct 38% of the time.
* **Recall for Class 1**: 97% - the model correctly identified 97% of the actual active contributors.

The overall **accuracy** of the model is **74%**.

#### Confusion matrix

|  |  |
| --- | --- |
| True negatives: 106 | False positives: 47 |
| False negatives: 1 | True positives: 29 |

### 5.4 Limitations

One limitation of my project was that I only considered the last 100 commits per user. This approach may have omitted long-term activity patterns, potentially impacting the completeness and accuracy of the analysis. Additionally, despite making adjustments to address class imbalance such as applying class weighting in the models the imbalance may still have affected the model's performance, leadingto biased predictions or reduced effectiveness in classifying contributors accurately.

## 6. Suggestions for Future Improvements or Further Research

I've got some ideas for future improvements. For data enrichment, I could add features like the number of followers or total public repositories to get a better picture of each contributor's overall GitHub activity. Analyzing the programming languages they use might also help identify their areas of expertise.

To make my models more accurate, I'm considering advanced techniques like Random Forests or Gradient Boosting Machines (like XGBoost or LightGBM). These can handle complex patterns better than basic models.

Hyperparameter tuning is another step I could take. By using grid search or random search, I can systematically find the best settings to optimize model performance.

Addressing class imbalance is important too. I might apply resampling techniques like SMOTE to increase the number of active contributors in the training set. Experimenting with different thresholds for labeling contributors as active or inactive could also help balance the classes.

I'm also thinking about using unsupervised learning methods. Clustering algorithms like K-Means or DBSCAN could help me discover natural groupings in the data without relying on predefined labels.