Introduction

The Rails Project:

Ruby on Rails, commonly referred to as Rails, is an open-source web application framework written in Ruby. Rails is a model–view–controller (MVC) framework, which provides default structures for databases, web services, and web pages. Since its initial release in 2004, Rails has amassed a large and dedicated community of developers, leading to its adoption in many popular web applications and services. Its emphasis on convention over configuration (CoC) and the "Don't Repeat Yourself" (DRY) principle has greatly influenced web app development, advocating for simplicity and maintainability.

The Rails GitHub repository serves as the central hub for development, allowing developers from around the world to contribute, report issues, and collaborate on enhancements. With an active community, a multitude of issues and pull requests are frequently submitted, highlighting the software's dynamic evolution.

Objective of the Analysis:

The main aim of this analysis is to delve deep into the last 500 issues reported in the Rails GitHub repository. By studying these issues, we intend to:

1. Understand the temporal trends of issue reports, potentially identifying any patterns or recurrent periods of heightened activity.

2. Identify if there are any contributors who report issues more frequently than others.

3. Ascertain the most popular categories or labels under which issues are classified.

4. create a machine learning model to predict issue labels based on their descriptions, automating a critical aspect of issue triage.

Through this investigation, we hope to gain insights into the dynamics of the Rails community, the challenges they frequently encounter, and the potential areas of focus for the Rails core team.

Got it. Let's revise the Data Collection section to reflect the two methods you've utilized: the GitHub API and the Scrapy project.

Methodology

Data Collection:

1. Scrapy Framework:

Scrapy is an open-source web-crawling framework for Python. It allows users to write spiders to crawl websites and extract structured data from them.

- Data Extraction via Scrapy:

- We initialized a Scrapy spider tailored to navigate the Rails GitHub issues page.

- The spider begins its journey from the primary issues page, extracting key details like the title, associated labels, and direct links to individual issues.

- Once an issue's link is identified, the spider navigates to that page to gather more detailed information, such as the creation date, the individual who submitted the issue, and the comments associated with the issue.

- Using BeautifulSoup, we parsed the data, extracting relevant details from the HTML content.

- The process continues until 500 issues are scraped or until no more pages are available for scraping.

2. GitHub API:

The GitHub API is a powerful tool that provides programmatic access to GitHub's vast array of functionalities, from user profiles to repository data.

- Data Extraction via GitHub API:

- We accessed the '/rails/rails /issues' endpoint to fetch details of the Rails project's issues.

- With parameters set to capture both open and closed issues and to fetch the maximum number of issues per page, we iterated through pages until 500 issues were accumulated.

- The data obtained this way was saved into a JSON file for further analysis.

Data Analysis:

For the analysis phase, the collected data was loaded into a Pandas DataFrame, a popular data manipulation tool in Python. The steps taken for each of the questions were as follows:

Issue Evolution Over Time:

- Issues were grouped based on their creation dates.
- A time-series plot was used to visualize the number of issues reported daily.

Periods with More Issues:

- Issues were further grouped by month to get a monthly perspective.
- A bar chart was employed to identify months with exceptionally high issue counts.

Frequent Issue Reporters:

- The data was grouped by the reporter's username.
- A histogram showcased the distribution of issues among different reporters.

Most Popular Category (Label):

- Labels were extracted from the list associated with each issue.
- A count was maintained for each unique label to identify the most popular ones.
- A pie chart provided a visual breakdown of label popularity.

Issue Classification Using HuggingFace Model:

- For this task, descriptions (body) were used as input, and the associated labels served as the output target.

- A DistilBERT model, a lightweight version of BERT from the HuggingFace Transformers library, was employed.
- The dataset was split into training and validation sets. The model was trained on the training data and evaluated on the validation set.
- Model performance was assessed using metrics such as accuracy, precision, recall, and F1-score.

Results & Analysis

For a detailed walkthrough of the data extraction process, data cleaning, and pre-processing, please refer to the repository https://github.com/ElabdiAchraf/rails-issue-analysis.git . Both datasets (using Scrapy Framework and GitHub API) were separately cleaned, pre-processed, and analyzed to draw the following insights.

Evolution of Issues Over Time (Detailed Analysis)

- **Cumulative Issues Over Time:**

Using a cumulative plot helps visualize the overall growth of issues in the Rails project over time.

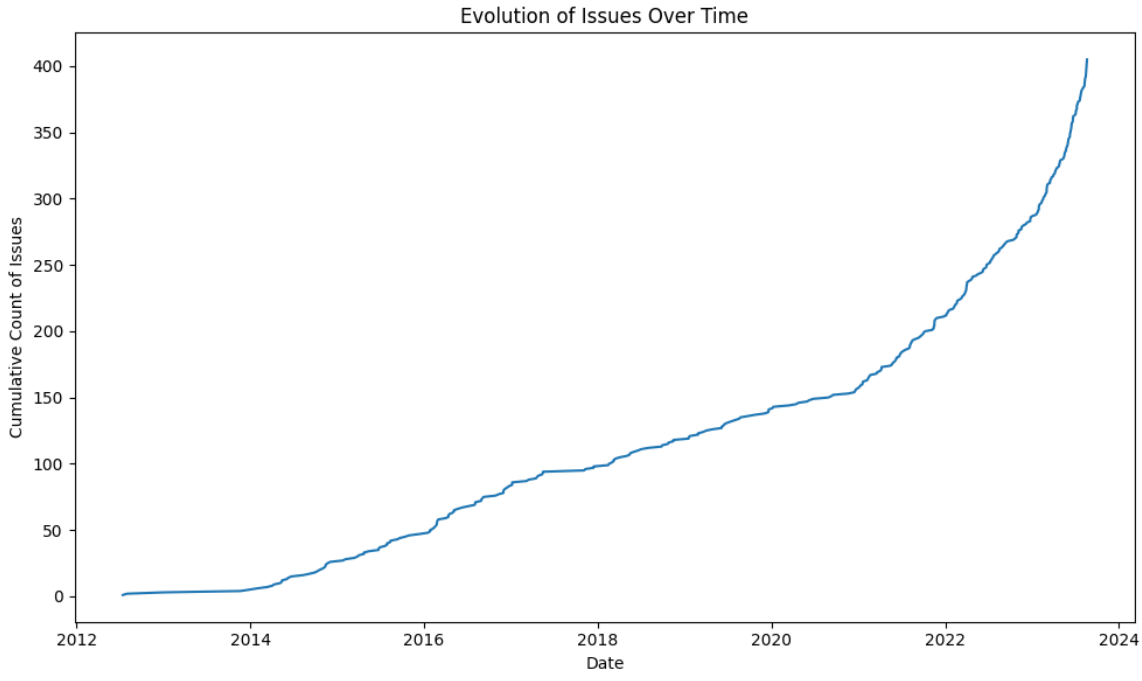


Figure 1 Cumulative Issues Plot

In the plot, it's evident that the growth rate of issues was steady until 2021. After this year, there was a noticeable acceleration in the issue reporting rate.

Interpretations:

- **Steady Phase:** Until 2021, the project saw a stable increase in issues. This phase indicates regular development, maintenance, and engagement by the community.
- **Acceleration Post-2021:** The sharper incline after 2021 signifies a potential change in the project dynamics. It might be due to various reasons like more users adopting Rails, significant updates/releases leading to more reported issues, or increased community participation

- **Monthly & Daily Issues Over Time:**

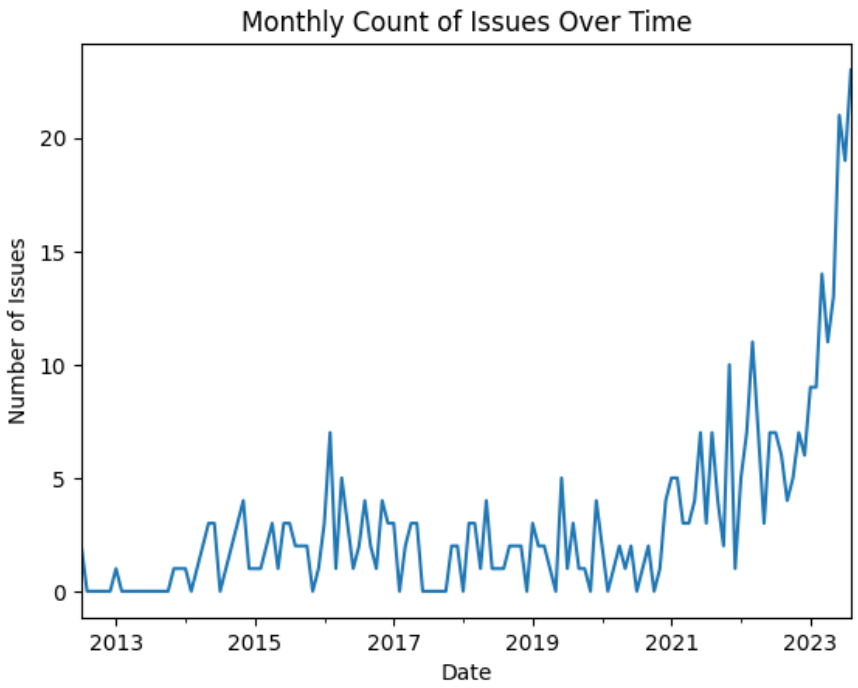


Figure 2 Monthly Issues Plot

Breaking down the issue count into smaller time intervals (monthly and daily) provides more granular insights into the project's health and community engagement.

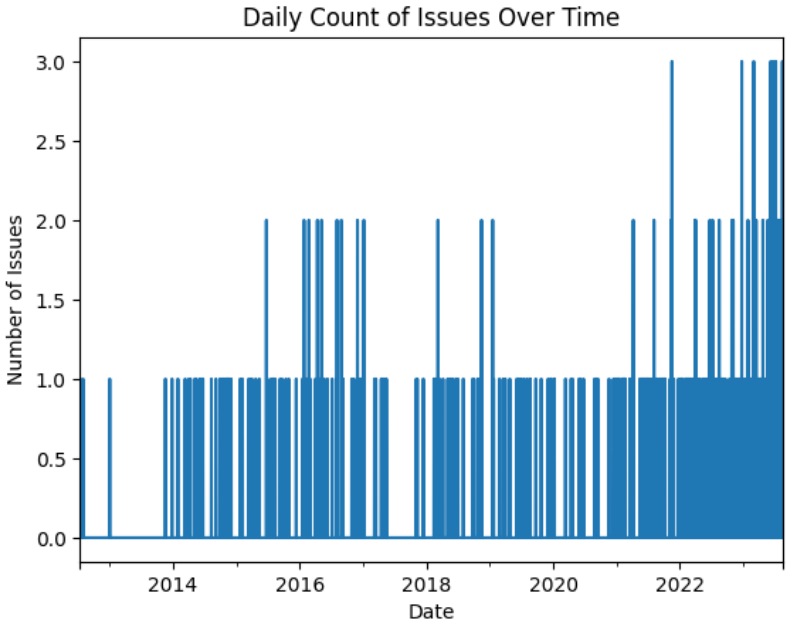


Figure 3 Daily Issues Plot

On a day-to-day breakdown, there's an average of about 3 issues reported per week after 2021. This rate is relatively high, indicating consistent engagement or persistent challenges.

Interpretations:

- **Monthly Increase:** After 2021, some months saw a higher number of issue reports, possibly due to major updates or widespread adoption.
- **Daily Reporting:** Roughly 3 issues are reported weekly, suggesting a consistently active community but also highlighting the need for regular monitoring.

Periods with High Issue Reporting:

Using monthly data, you've generated insights about the frequency of issues being reported over time. The plotted graph illustrates both the direct count of monthly issues and a 3-month rolling mean for smoother trend analysis.

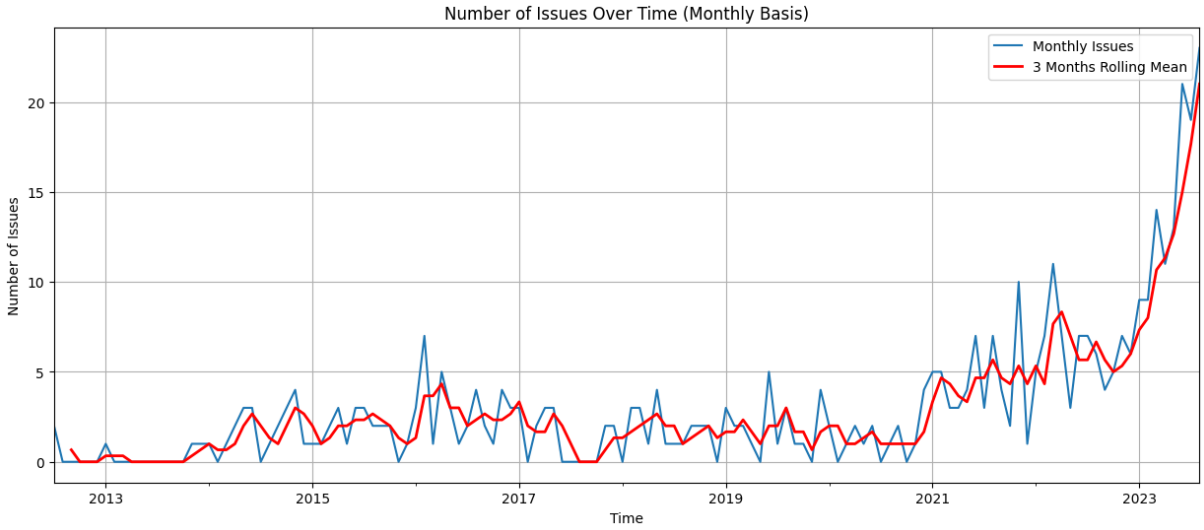


Figure 4 Number of Issues Over Time (Monthly Basis)

Upon analyzing the results:

- **Graphical Analysis**: The plotted graph indicates certain months where the number of issues reported spiked. The red line, which represents the 3-month rolling mean, provides a clearer view of this upward or downward trend over time. It allows us to understand the moving average, minimizing the effects of any outliers or extreme months.

- **Month Analysis**: When you breakdown issue reports by month, we notice that June (with 50 issues) has the highest number of issues reported, followed by August (with 48 issues). On the other hand, October has the lowest count (with only 16 issues).

This pattern can be interpreted in a few ways:

- There might be significant software releases or changes in June and August that might lead to more bugs or concerns.

- Alternatively, it could indicate a seasonal pattern where more users interact with the Rails project during these months, leading to increased issue reporting.

- October being the month with the lowest reports might suggest a relatively stable phase for the Rails project, or it could be a period of lesser user interaction.

Frequent Issue Reporters Analysis:

From the data collected and processed, we're able to identify frequent reporters of issues in the Rails project. Understanding who these individuals are can be pivotal for several reasons. They might be core contributors, stakeholders...

The analysis presents the following insights:

Key Issue Reporters:

- Topping the list with 5 issues, 'dhh' (David Heinemeier Hansson) is notably the creator of Ruby on Rails. His active involvement in issue reporting underscores his dedication to the project's quality and its continuous improvement.

- Following closely is 'georgeclaghorn' with 4 issues. Given the volume of their reports, they might be a significant contributor or a seasoned user keen on enhancing Rails.

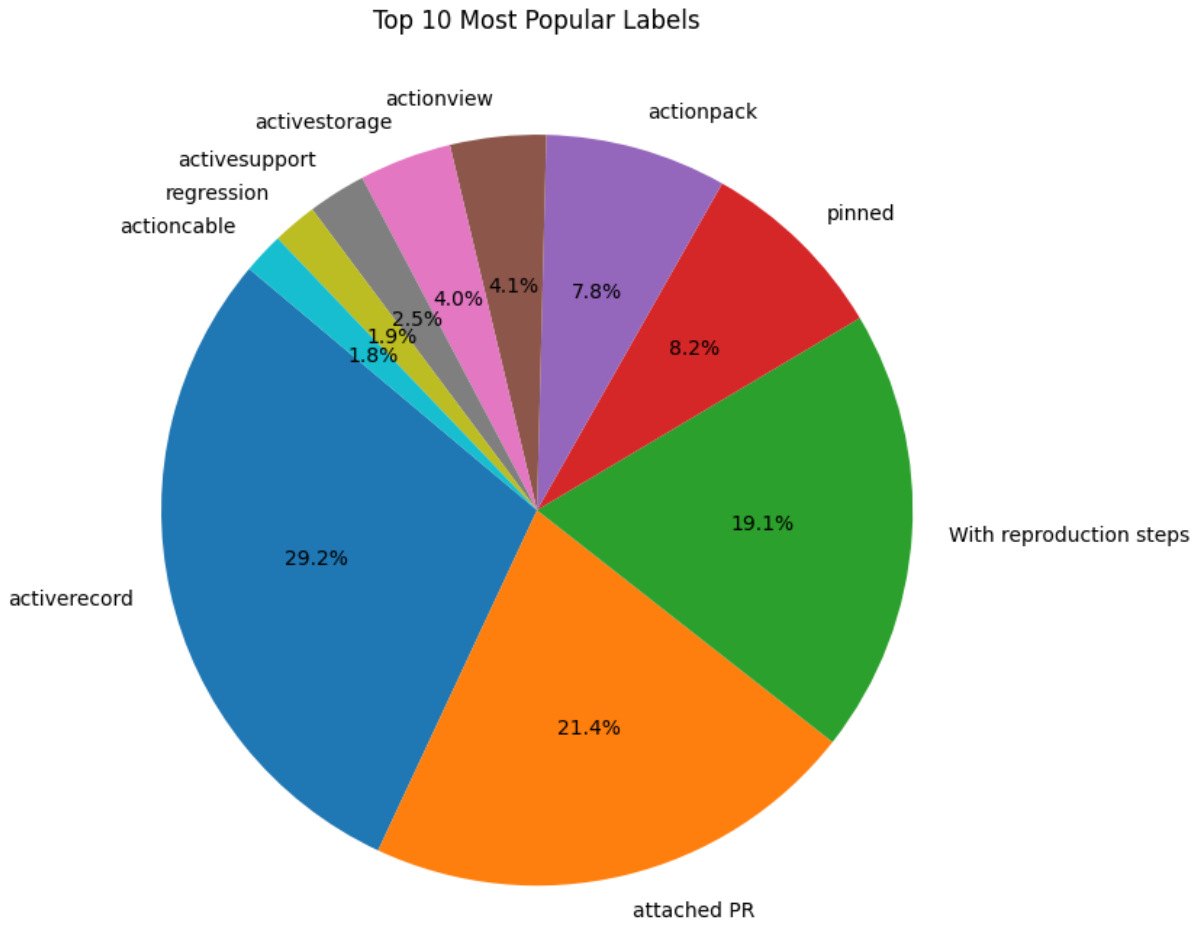
- Multiple contributors including khiav223577, shouichi, matthewd, maximerety, and Jeremy have reported 3 issues each. Their frequent interaction suggests a consistent engagement with the Rails project.

Implications:

The fact that prominent names like "dhh" and "jeremy" appear in the list is a positive sign indicating that leading members are actively engaged in the project's refinement.

The distribution also suggests a balanced community where not only key individuals but various other members contribute to identifying and addressing challenges.

Most Popular Category (Label):



This pie chart visualizes the distribution of the top labels associated with Rails issues, making it evident which categories dominate the discussions or challenges.

➔ The most popular label is 'activerecord' with 198 issues.

Interpretation:

The label ‘activerecord’ stands out as the most frequently associated label with 198 issues. This indicates that a significant number of issues or discussions in the Rails repository revolve around ‘Active Record’, which is Rails' Object-Relational Mapping (ORM) system. Either there are many features or enhancements being requested for ‘Active Record’, or it might have had several bugs or concerns raised by the community.

Issue Classification Using HuggingFace Model Analysis

The goal was to classify Rails project issues based on their descriptions using a deep learning model. The output should correspond to the correct label associated with each issue.

Methodology:

- A distilled version of the BERT model (DistilBERT) from the HuggingFace Transformers library was used.
- The dataset utilized the issue descriptions as input and the associated labels as the output.
- Data was split into training (80%) and validation (20%) sets.

Result:

```
100%|██████████| 11/11 [02:44<00:00, 14.98s/it]
{'eval_loss': 1.8101749420166016, 'eval_accuracy': 0.4772727272727273,
'eval_f1': 0.34655248133509003, 'eval_runtime': 180.6292,
'eval_samples_per_second': 0.487, 'eval_steps_per_second': 0.061}
```

Interpretations:

- Loss Value: The evaluation loss value is 1.8101. Generally, a lower loss value indicates a better model,
- Accuracy: The model achieved an accuracy of approximately 47.73% on the validation set. This means that the model correctly predicted the label nearly half of the time.
- F1-Score: The weighted F1-Score was calculated as 0.3465, which is lower than desired. F1-score considers both precision and recall, and a higher score indicates a better balance between these two metrics. The lower score suggests there may be some classes where the model struggles more than others.

The DistilBERT model's performance on classifying Rails project issues based on their descriptions is moderate. While the model does achieve nearly 50% accuracy, there is significant room for improvement. Factors to consider for enhancing performance include:

  - Incorporating more training data.
  - Experimenting with different architectures or training strategies.

Conclusion

We found a noticeable increase in issue reporting after 2021 in our thorough analysis of the Rails project's problems, with the activerecord component appearing as a key area of concern. It was possible to identify active contributors, highlighting the project's active community. Additionally, we attempted to categorize problems based on descriptions using the DistilBERT model, with an accuracy of about 47.73%. While this is a starting point, tweaking is necessary for better triage. Overall, by recognizing these tendencies and utilizing machine learning, the Rails community may be directed toward more effective development and problem-solving.