

CS212-Group_13-Project Report

[Hypothesis](#)

[Metrics](#)

[Find Our Data](#)

[The Most Workable Way](#)

[Alternative Way](#)

[Sample Output](#)

[Conclusion](#)

[Code](#)



Bingliang Li
Bolin Cui
Yue Hu
Yuhe Zhang
[Link to this report](#)

Hypothesis

For a certain version of the kernel, the number of patches that submitted daily and the time after the kernel is released is a e^{-t} relation.

For example, for kernel 4.x, the first day it released the developers may submit 100 patches, and on the 5th day, they may only submit 20 patches because the obvious bugs are fixed in the first few days so there tends to be less and less patch after its release.

Metrics

The number of patches that submitted daily: every commit that fixed a bug(or a part of a bug) is a patch. So, from the day that the kernel is released, we could count the daily amount of the "patch" commit.

Find Our Data

The Most Workable Way


There are many ways to identify whether a "git commit" is a patch, we found that the most of them have keywords such as "bug" or "patch" or "fix" in it's commit message, so we can identify a patch by searching those keywords in the commit git log, not very accurate, but easy and workable.

Alternative Way

Considering not all the patches are that obvious, I found some articles that may help us improve accuracy when finding the patch:

LINKSTER | Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering

While many uses of mined software engineering data are automatic in nature, some techniques and studies either require, or can be improved, by manual methods. Unfortunately, manually inspecting, analyzing, and annotating mined data can be difficult and tedious, especially when information from multiple sources must be integrated.

 <https://dl.acm.org/doi/abs/10.1145/1882291.1882352>



Linkster seems to be a useful tool to identify a patch quick and easy, but I can't find it all around the internet, so I asked Dr.Christian Bird(one of the authors) if he can help me, but seems that repo is long lost in the history.

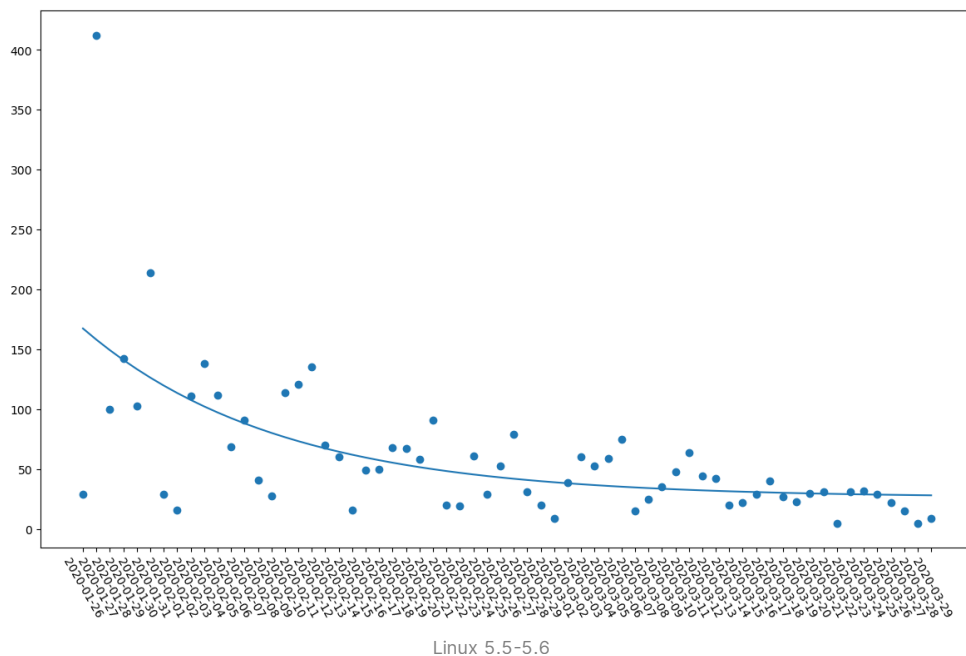
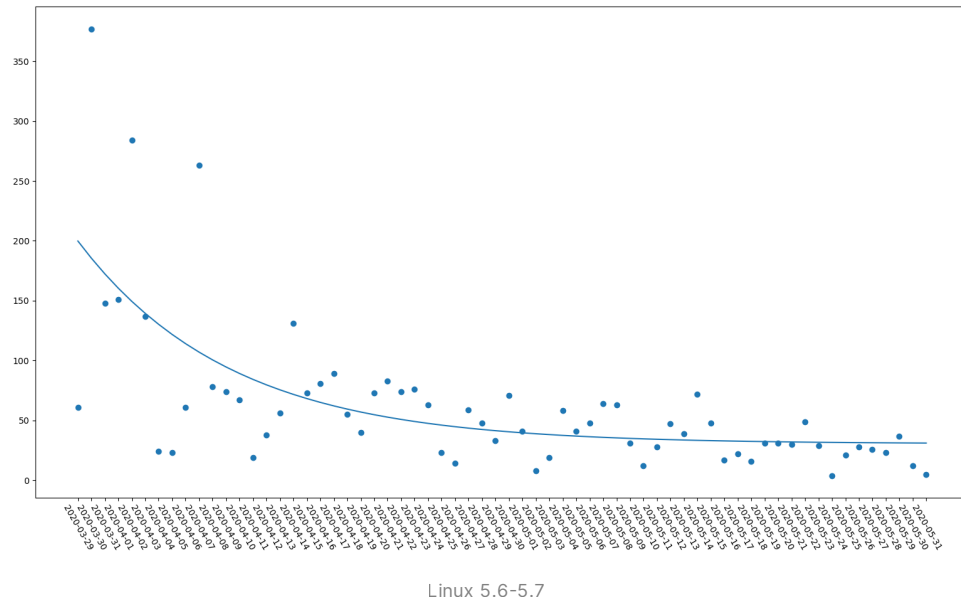
Identifying Linux bug fixing patches - IEEE Conference Publication

IEEE Xplore, delivering full text access to the world's highest quality technical literature in engineering and technology. | IEEE Xplore

 <https://ieeexplore.ieee.org/abstract/document/6227176>

This article is about using machine learning to identify patches, it not only use commit message but also by analyzing the code change in each commit, sounds cool but I can't fully understand some algorithms the article mentioned, and seems to be a overkill for this hypothesis, which doesn't require 100% accuracy.

Sample Output



Conclusion

From the figure we could say basically the pattern is right, it's similar to a e^{-t} relation, but obviously not accurate at all, there are noticeable variations, especially in first few days after a version is released.

I believe there are two possible situations:

1. This is what actually happened, the amount of patches is not highly related to time. (I believe this is the most possible situation, which means our hypothesis is not exactly true, it's generally negative correlation but have some variations)
2. I didn't find the exact amount of patches, searching by keywords is not accurate enough. (of course that will happen but I don't think this is the primary reason)

Code



The log.csv file is organized as "--pretty=format:'%c','%B'".

```
'''
MIT License

Copyright (c) 2020 Bingliang Li, Bolin Cui, Yue Hu, Yuhe Zhang

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
'''
import numpy as np
import matplotlib.pyplot as plt
# import sklearnBO
from scipy.optimize import curve_fit

date_commit=dict()

def readFileRows(fileadd, endver): # File address; when reach the certain version, stop collection

    file = open(fileadd, 'r', encoding='gb18030', errors='ignore')
    context = str(file.read())

    row_date_commits = list(context.split("\n\n")) # Turn each pair of date & commit to an element of list

    for d_c in row_date_commits:
        if d_c[29:] == endver: # When reach tag v5.6, quit the loop(the log file start with v5.7)
            return None
        if len(d_c) <= 10: # Ignore blank and exceptional line
            pass
        else:
            if not d_c[1:11] in date_commit.keys():
                date_commit[d_c[1:11]] = [d_c[29:]] # Set key(date) and value(commit)
            else:
                date_commit[d_c[1:11]].append(d_c[29:]) # If key(date) exists, append the value(commit)

def func(x, a, b, c):
    return a * np.exp(-b * x) + c

readFileRows('log_cs_B.csv', 'Linux 5.6')

date_count = dict()

for k, v in date_commit.items(): # Get dict of {date:patch amount} pair. Actuall this is highly inaccurate
    count = 0
    for commit in v:
        if 'fix' or 'bug' or 'patch' in v:
            count += 1
    date_count[k] = count

# Fit and draw
num_of_date = len(date_count.keys())
x = np.arange(1,num_of_date + 1)
```

```
x_labels = sorted(date_count.keys())

plt.xticks(x,x_labels, rotation=300)
y = list(reversed(date_count.values()))
plt.scatter(x,y)

popt, pcov = curve_fit(func, x, y)
y2 = [func(i, popt[0],popt[1],popt[2]) for i in x]
plt.plot(x, y2)
plt.show()
```