Evaluating Machine Learning Model Stability for Software Bug Prediction

Joud El-Shawa[1,2], Marios Grigoriou[1,2], Konstantinos Kontogiannis[1,2]

Western University[1] IBM Centre for Advanced Studies[2]

Introduction

Large software systems are implemented using many different programming languages and scripts, and consequently the dependencies between their components are very complex. It is therefore difficult to extract and understand these dependencies by solely analyzing the source code, so that failure risks can be detected accurately. On the other hand, it is a common practice for software engineers to keep track of process related metrics such as the number of times a component was maintained, with which other components it has been co-committed, whether the maintenance activity was a bug-fixing activity, and how many lines of source code have been altered. These data provide valuable information to be used for training a machine learning model and for devising metrics which can predict the risk associated with a future failure of a component due to maintenance activities in this or in another component related to it.

Methods

Data Acquisition:
A custom GitHub extractor was used to acquire detailed dumps of project repositories containing the data used

Model Training:
For each system, the first 40% of commits were used to train the model based on whether a file is faulty or not

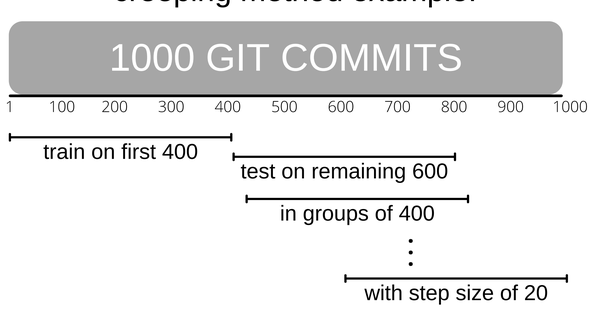
Model Testing:
The remaining 60% of commits were used to test the model. This was done with a step size of 20, same length as the 40% trained

Feature Gathering:
While training and testing, stats such as num of faulty/clean files and mean time between commits were extracted for each "creep"

Data Analysis:
Average values were acquired from the 10 runs, stored in excel file, identified points of interest, calculated percent changes to analyze

Repeat 10x

creeping method example:



Results

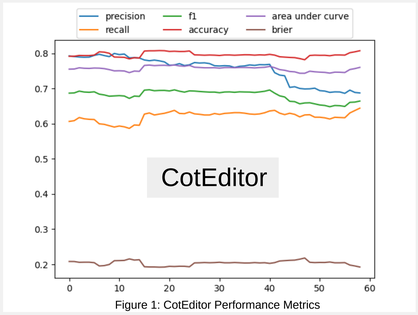


Figure 1: CotEditor Performance Metrics

Table 1: CotEditor Performance Metrics and Statistics

1	precision	recall	f1	accuracy	auc	brier	num of faulty files	num of clean files
16	0.787	0.6	0.68	0.787	0.75	0.21	340	563
17	0.781	0.6	0.7	0.807	0.77	0.19	318	585
18	0.779	0.6	0.7	0.807	0.77	0.19	317	586

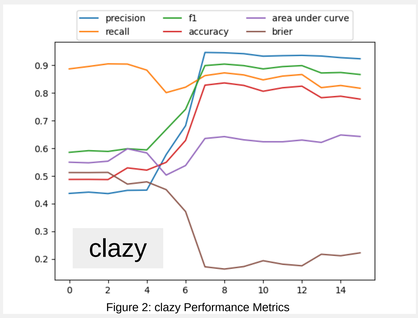


Figure 2: clazy Performance Metrics

Table 2: clazy Performance Metrics and Statistics

1	precision	recall	f1	accuracy	auc	brier	num of faulty files	num of clean files
7	0.577	0.801	0.67	0.549	0.503	0.45	179	131
8	0.682	0.821	0.74	0.629	0.538	0.37	206	106
9	0.946	0.863	0.9	0.828	0.636	0.17	292	24
10	0.945	0.873	0.9	0.836	0.642	0.16	292	25

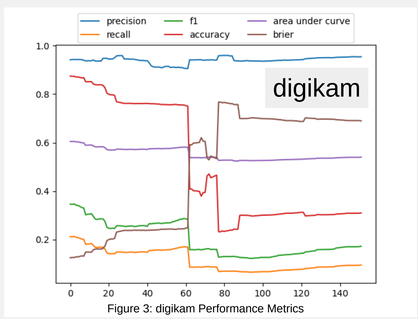


Figure 3: digikam Performance Metrics

Table 3: digikam Performance Metrics and Statistics

1	precision	recall	f1	accuracy	auc	brier	num of faulty files	num of clean files	Avg CommitsFeature	Percent Difference CommitsFeature
63	0.906	0.169	0.285	0.751	0.58	0.25	1153	2779	5.768056968	0.00881912
64	0.942	0.086	0.158	0.409	0.54	0.59	2529	1403	6.313835198	9.462081129
65	0.941	0.087	0.159	0.409	0.54	0.59	2530	1403	6.318586321	0.07524939
77	0.94	0.087	0.159	0.463	0.54	0.54	2760	1958	5.863925392	0.760102118
78	0.941	0.088	0.16	0.466	0.54	0.53	2761	1986	5.852538445	0.19418642
79	0.959	0.069	0.128	0.232	0.53	0.77	3900	848	6.564448189	12.16411904
80	0.959	0.069	0.128	0.233	0.53	0.77	3900	849	6.579490419	0.229146912

main metrics used for analysis:

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

$$precision = \frac{TP}{TP + FP}$$

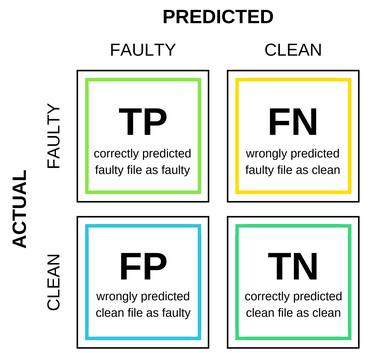


Figure 4: Confusion Matrix

Findings & Next Steps

The **points of interest** were calculated by taking the absolute difference between the previous value and the current value and dividing by the previous value. Then, if the result acquired is greater than two times the standard deviation plus the average, it is considered a point of interest.

Generally, models, like the one for **CotEditor** (Figure 1, Table 1), do not lose predictive capability. They have overall stable trend lines with few bumps or increases/decreases. This is the case for many models, and thus it can be confidently claimed that once a model is trained, it will most likely not need to be retrained again.

For some systems, such as **clazy** (Figure 2, Table 2), there seems to be a correlation between the number of clean and faulty files in each creep and the precision of the machine learning model. Observe when the number of clean files decreases drastically (lines 8-9), precision (along with some other metrics) increases. This is likely due to the fact that the machine learning models are geared towards detecting faulty files, so when the ratio of clean files to faulty files changes from 0.51 to 0.08, the likelihood of the machine learning model wrongly predicting a clean file as faulty (i.e., FP) decreases since there are less clean files to be predicted, and thus the precision has a lower denominator (i.e., its value goes up).

Finally, a correlation was noted in some systems, such as **digikam** (Figure 3, Table 3), between accuracy and the commits feature (one of the features used to test and train the model on). It can be observed that both times (65 and 79) points of interest were present, the percent differences for the commits feature went up drastically, from 0.009 and 0.19 to 9% and 12%. Also, it can be observed once again that in both lines 64 and 79, precision went up, and simultaneously the number of clean files decreased, and the number of faulty files increased.

For future research, why we have these observations must be investigated and attempts to find a cause-and-effect relationship should be made.

Acknowledgements

Alberto Giammaria (IBM US, Austin TX. Lab),
Chris Brealey (IBM Canada, Toronto Canada Lab)

Joud El-Shawa
jelshawa@uwo.ca