Just-In-Time Software Defect Prediction

Ali M, Yifei Gong, Yunhua Zhao

April 7, 2021

1 Introduction

Software plays a more and more important role in our real life, a small problem in a software may cause a huge loss,like one of the biggest American market makers for stocks Knight struggled to stay afloat after a software bug triggered a $440 million loss in just 30 minutes. The primary objective of software quality assurance activities is to reduce the number of defects in software products. Software defect prediction has been an active research area.
The defect is found earlier, the cost is less, also now it is a challenging problem considering the limitation of budget and time allocation for such activities [1]. Just-in-time software defect prediction(JIT-SDP) extracts features from code changes(eg.codes from a git commit), feed these features to a model then to predict if new code changes are buggy or clean. JIT-SDP predicts defects during the code change state, when the knowledge is still fresh to the developers, which help to save the debug time. Figure 1 is an example of buggy commit in OPENSTACK 1, it includes commit id, commit author, commit date, commit message and commit diffs, which describe a defective code submit.

```
commit d60f6efd7f70efba1ccd007d55b1fa740fb98c76
Author: Dan Prince <email address hidden>
Date: Mon Jan 14 12:26:36 2013 -0500
     Name the securitygrouprules.direction enum.
     Updates to the SecurityGroupRule model and migration so that we
     explicitly name the securitygrouprules.direction enum. This fixes
     'Postgresql ENUM type requires a name.' errors.

     Fixes LP Bug #1099267.
     Change-Id: Ia46fe8d4b0793caaabbfc71b7fa5f0cbb8c6d24b
diff --git a/quantum/db/migration/alembic_migrations/versions/3cb5d900c5de
_security_groups.py
index ff39de84a..cf565af0f 100644
--- a/quantum/db/migration/alembic_migrations/versions/3cb5d900c5de_
security_groups.py
+++ b/quantum/db/migration/alembic_migrations/versions/3cb5d900c5de_
security_groups.py
@@ -62,7 +62,10 @@ def upgrade(active_plugin=None, options=None):
-        sa.Column('direction', sa.Enum('ingress', 'egress'), nullable=True),
+        sa.Column('direction',
+                  sa.Enum('ingress', 'egress',
+                          name='securitygrouprules_direction'),
+                  nullable=True),
diff --git a/quantum/db/securitygroups_db.py b/quantum/db/securitygroups_db.py
index 9903a6493..5bd890bbe 100644
--- a/quantum/db/securitygroups_db.py
+++ b/quantum/db/securitygroups_db.py
@@ -62,7 +62,8 @@ class SecurityGroupRule(model_base.BASEV2, models_v2.HasId,
-    direction = sa.Column(sa.Enum('ingress', 'egress'))
+    direction = sa.Column(sa.Enum('ingress', 'egress',
+                                  name='securitygrouprules_direction'))
```

Fig. 1: An example of a buggy commit change in OPENSTACK.

Figure 1: example

2 Each team member's role and contribution

Our team wants to do experiments and compare to Hoang's paper's result [2], we will try some simple models and dnn.

- Dataset exploration: Yifei
- Class balance: Ali
- Feature pre-preprocessing: Yunhua
- Model build, logistic regression plus dnn: Ali
- Model build, random forest tree plus dnn: Yifei
- Model build, Bayesian Network plus dnn: Yunhua
- Evaluation: all of the team will report F1, accuracy auc, recall and precision
- Mode comparison: all of the team
- Report: All of us need to write their parts report.

3 Enumerate the CSCI 740 related topics (questions and solution approaches) each individual will contribute to the project

- Dataset exploration: Yifei
  Will plot figures to show the data sets, also calculate the mean, max, min and check if there is na or empty values, and how the correlation of the data set features.
- Class balance: Ali
  Try different methods, oversampling(SMOTE) and under-sampling methods.
- Feature pre-preprocessing: Yunhua
  Check the relationships among features, remove some correlated features, rank them and select features
- Model build, tune parameters and visualization, logistic regression plus dnn: Ali
- Model build, tune parameters and visualization, random forest tree plus dnn: Yifei
- Model build, tune parameters and visualization, Bayesian Network plus dnn: Yunhua

- Evaluation: all of us will report G-mean, F1, accuracy auc, recall and precision
  Will print the results, also plot ROC curve and confusion metrics

- Mode comparison: together
  Compare our results with Hoang's result [2], if our results do not win, may try ensemble learning.

- Report: All of us need to write their parts of the report.

4 Describe the dataset

There are totally 2 data sets: openstack and qt; openstack is for cloud computing and QT is widget toolkit for creating graphical user interfaces. Each of them has 35 features, like lines of modified, developer experience.

data set	row number	column number
qt	25150	35
openstack	26854	35

5 Provide a timeline for the project

week	work
04/09/2021-04/16/2021	data exploration, class balance, pre-processing
04/16/2021-04/23/2021	model build and tune
04/23/2021-04/30/2021	evaluation and model comparison
04/30/2021-05/06/2021	team members result sharing and discussion
05/06/2021-05/13/2021	write report
05/13/2021-05/20/2021	modify report and presentation preparation

6 Describe what you plan to demo on the final exam date and other deliverables

- our data visualization
- our data pre-processing results(store to a file and visualize)
- model tuned parameters
- evaluation results: F1, accuracy, auc, G-mean
- Comparison result with Hoang's paper [2]

7 Describe your plan to evaluate the project.

We will use the methods studied from this class to these popular datasets in JIT-SDP, then compare to the well-known papers' results, maybe we can make an improvement and find some new angle in this area.

References

[1] Zhiyuan Wan, Xin Xia, Ahmed E Hassan, David Lo, Jianwei Yin, and Xiaohu Yang. Perceptions, expectations, and challenges in defect prediction. *IEEE Transactions on Software Engineering*, 46(11):1241–1266, 2018.

[2] Thong Hoang, Hoa Khanh Dam, Yasutaka Kamei, David Lo, and Naoyasu Ubayashi. Deepjit: an end-to-end deep learning framework for just-in-time defect prediction. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 34–45. IEEE, 2019.