Log Templating Using Drain3

Kyle Bryant, Tianze Shan,Parker Stone, Ningxiao Tang,Hong Xin

April 13, 2021

Word count: 4,729

Abstract

1 Introduction

1.1 Background Related Works

kyle

1.2 Motivation for Research

As technology advances and becomes more heavily used, more machine generated data logs will be produced. These data logs carry a lot of information and the need to automatize log processing is at an all time high and continuing to rise. In our case, we are processing build logs that record all of the output generated by compiling the job's source code and executing tests. The desired result is to parse these logs, try to cluster them based on similarity and then classify if they are a pass log or a fail log.

2 Preliminaries

kyle

2.1 Research Questions

2.2 Experimental Setup

3 Data

3.1 Log Sources

The logs we used as our source of data comes from Red Hat's open-shift build logs from Test-Grid. These build logs would contain the log itself and a json object containing specific data regarding that log, but the only data we used was the log itself and whether or not it passed.

3.2 Cleaning the Data

In order to maximize the accuracy of our classifier, cleaning the data logs is a must. The components that we wanted to remove were anything that we would consider to be unique. Things such as dates, timestamps, machine generated hashes, URLs and version numbers would all be considered unique are would be removed from the data log. In our project there are two different processes that parse the data logs. The first is manual parsing and the second is parsing from Drain3. As a result Drain3 is able to cluster each log more accurately.

4 Drain 3

4.1 Understanding the API

Drain3 is a powerful open source log processing tool that was created by IBM. This tool uses clever ways of classifying each log which maintains efficiency while still being accurate at the same time. The first thing Drain does is parse the logs by removing unique classifiers such as dates/timestamps and hashes. Drain3 creates a tree-like structure where each leaf is different based on the length of a log. When there are multiple logs of the same length it clusters the logs by using longest common subsequence to identify any similarity. The result returns a dictionary object for each log classifying its cluster ID, its cluster size and a few other data points.

4.2 Working with the API

Using Drain3 is very straightforward. Once all of the required dependencies are installed, we would simply pass in our array of logs into the imported function and then Drain3 would cluster

our logs accordingly. The only drawback with using Drain3 was that it does not consistently parse certain components in every log. In order to fix this issue we had to do some manual parsing before processing the logs into Drain3. As a result all of the desired content remains in the logs without the noise that would negatively influence the clustering.

4.3 Results of the API

After processing the logs into Drain3, the API returns a dictionary for each cluster with a few points of data. We used this result to produce two different dictionaries. One of the dictionaries contains the cluster ID number and all of the logs that contain that ID number. The other dictionary contains the cluster ID numer and the size of that cluster. The usefulness of Drain3 is that we started with 266 different logs and it helped us narrow down to 155 unique logs. Reducing to the number of unique logs will increase our accuracy when classifying each log as a pass or fail.

5 Results

5.1 Drain 3 Performance

After processing the data logs into Drain3, we vectorized each log line using TF-IDF and then classifying each log using Random Forest Classifier. In order to judge if Drain3 helped classify pass/fail logs, we did the same process on the data logs before parsing and Drain3. The result was that Random Forest Classifier performed slightly better using Drain3.

5.2 Log Templating Accuracy

For how to determine if a log is failed or passed, we used TF-IDF to vectorize each log line and then used Random Forest Classifier to predict if the log passed or failed. What was returned is the RMSE and the accuracy of how the classifier performed. the accuracy on the unprocessed logs was 0.838 and the accuracy for the processed logs was 0.944.

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None, min_samples_split=12,
                               min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None,
                               max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,
                               bootstrap=True, oob_score=False, n_jobs=None, random_state=1, verbose=0,
                               warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None).fit(X_trai

from sklearn.metrics import mean_squared_error
y_test_predictions = model.predict(X_test)
print("RMSE on testing set = ", mean_squared_error(y_test, y_test_predictions))

from sklearn.metrics import accuracy_score
print("Accuracy on testing set = ",accuracy_score(y_test, y_test_predictions))
```

```
RMSE on testing set =  0.16129032258064516
Accuracy on testing set =  0.8387096774193549
```

5.3

```
[42] from sklearn.ensemble import RandomForestClassifier
     model = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=None, min_samples_split=12,
                                    min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None,
                                    max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,
                                    bootstrap=True, oob_score=False, n_jobs=None, random_state=1, verbose=0,
                                    warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None).fit(X_train,

     from sklearn.metrics import mean_squared_error
     y_test_predictions = model.predict(X_test)
     print("RMSE on testing set = ", mean_squared_error(y_test, y_test_predictions))

     from sklearn.metrics import accuracy_score
     print("Accuracy on testing set = ",accuracy_score(y_test, y_test_predictions))

     RMSE on testing set =  0.05555555555555555
     Accuracy on testing set =  0.9444444444444444
```

5.4

6 Conclusion

6.1 Limitations

The biggest drawback for our project is the limited amount of data logs we had access to for testing. The issue with this is that the manual parsing may not be generalized for other build logs that would be processed in the future. Another limitation from our project is the pass log to fail log ratio. There are far more pass logs than there are fail logs. The skewed ratio will lead the classifier to be more biased towards pass logs.

6.2 Future Work Final Thoughts

kyle

Data Source: https://testgrid.k8s.io/

Explanation of Drain3: https://jiemingzhu.github.io/pub/pjhe$_icws2017.pdf$

GitHub repository for Drain3: https://github.com/IBM/Drain3]Data Source: https://gcsweb-ci.apps.ci.l2s4.p1.openshiftapps.com/gcs/origin-ci-test/logs/canary-release-openshift-origin-installer-e2e-aws-4.5-cnv/

Data Source: https://testgrid.k8s.io/

Explanation of Drain3: https://jiemingzhu.github.io/pub/pjhe$_icws2017.pdf$

GitHub repository for Drain3: https://github.com/IBM/Drain3