Log Templating Using Drain3

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

April 28, 2021

Word count: 4,729

Abstract

1 Introduction

1.1 Background Related Works

Log templates are of much interest to large technology companies and researchers alike. As a result, there is a lot of initial research on the topic. For example, Drain3 is the largest machine-learning templater, which was worked on by IBM as well. Using existing research, it is clear that there is a gap in knowledge in this topic. While log templating is being worked on, there has yet to been a definitive API, the leader being Drain3. By determining Drain's effectiveness, progress can be made in discovering which API works best and should be expanded. Additionally, this will tell if Drain is accurate.

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

2.1 Research Questions

Currently, there are a few existing log templating programs, such as Papertrail and SolarWinds. They focus more on general templating and watching for security concerns. Whereas state-of-the-art templating is focused on this research project goal, finding the best API and how effective it can be. Within logs, current best practice is to keep log format human-readable. By creating logs that are human-readable, it is easier to process and clean logs for Drain and other APIs. Similarly, state-of-the-art is similar to ours. Instead on focusing on pure readability, more complex actions are done to the logs, such as TFID vectorizing.

The last research question revolves around if encoding methods vary based on the downstream machine learning task. Which was discovered to not matter, as the logs from OpenShift have similar structures. Once parsed, they all can be put into Drain3 to cluster the logs into pass and fail buckets. Therefore, by creating a solution for this problem, it can help solve a lot of related log template problems

2.2 Experimental Setup

While there was not much experimental setup, it is good to review. Logs were only pulled from Openshift to allow for some uniformity in the data. From there, the logs are cleaned and stored in an array where each array is a log build, and each value stored is a log. From there, further cleanup and optimization was done. Then, Drain can be used and evaluated for accuracy. The rest of the setup was simply manipulating the data to get it in it's most useful form.

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. The only data we used from the json object 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 and also a shortened version of the log after being parsed. 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 number and the size of that cluster. The usefulness of Drain3 is that we started with n different logs and it helped us narrow down to k unique logs. Reducing to the number of unique logs will increase our accuracy when classifying each log as a pass or fail. Also shortening the logs into a more generalized form will also increase the classifying accuracy as well.

5 Results

5.1 Drain 3 Performance

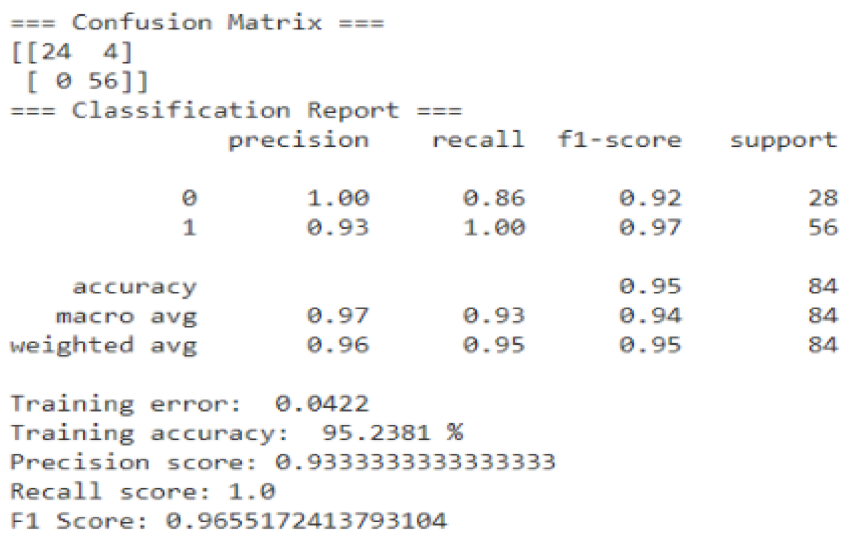
After processing the data logs into Drain3, we vectorized each log line using TF-IDF and then classifying each log using XGB classifier. In order to judge if Drain3 helped classify pass/fail logs, we did the same process on the data logs before parsing and Drain3. The result was that XGB Classifier would perform better on Drain3 processed logs when the data set was larger. When the data set was too small the classifier would either only assign one label to all logs or it would be more accurate for logs that have not been parsed.

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 XGB Classifier to predict if the log passed or failed. What was returned are the confusion matrix, the training error, training accuracy, precision score, recall score and F1 score of how the classifier performed. The accuracy of the unprocessed logs and the accuracy of the processed logs depended on the size of the data set.

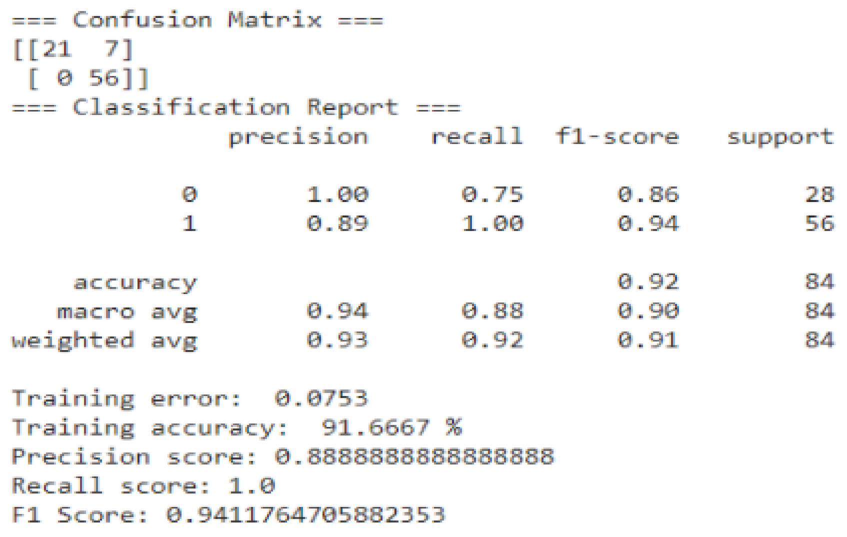
With 416 logs:

model_XGB

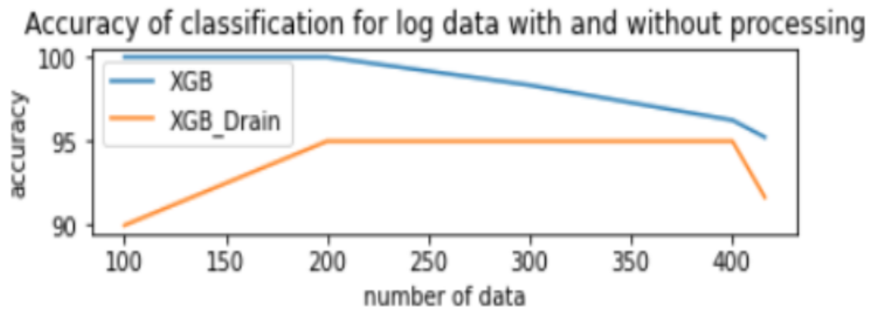


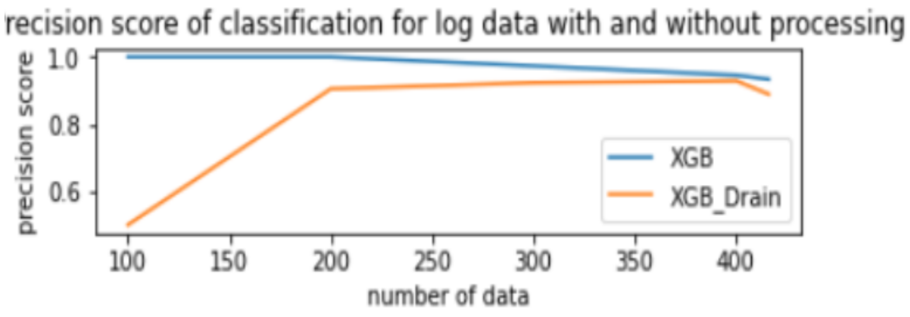
```
=== Confusion Matrix ===
[[24  4]
 [ 0 56]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      0.86      0.92        28
           1       0.93      1.00      0.97        56

    accuracy                           0.95        84
   macro avg       0.97      0.93      0.94        84
weighted avg       0.96      0.95      0.95        84

Training error:  0.0422
Training accuracy:  95.2381 %
Precision score: 0.9333333333333333
Recall score: 1.0
F1 Score: 0.9655172413793104
```

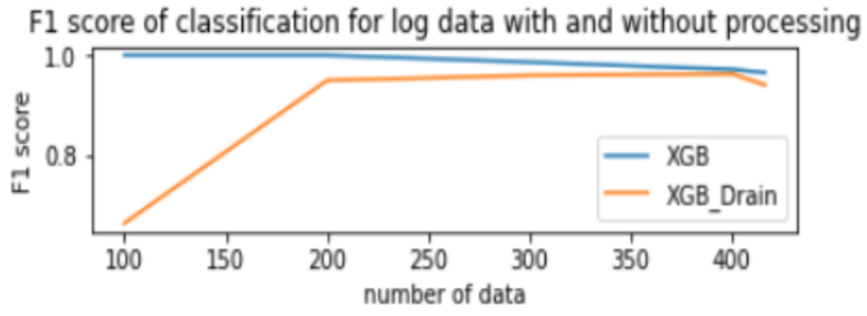
model_XGB_Drain

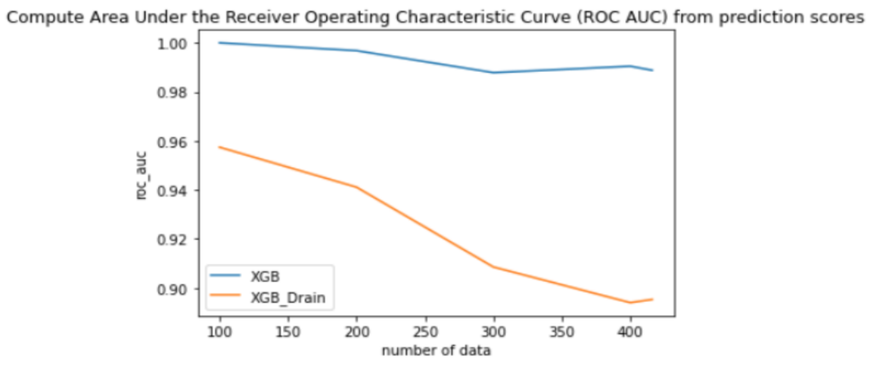


```
=== Confusion Matrix ===
[[21  7]
 [ 0 56]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      0.75      0.86        28
           1       0.89      1.00      0.94        56

    accuracy                           0.92        84
   macro avg       0.94      0.88      0.90        84
weighted avg       0.93      0.92      0.91        84

Training error:  0.0753
Training accuracy:  91.6667 %
Precision score: 0.8888888888888888
Recall score: 1.0
F1 Score: 0.9411764705882353
```



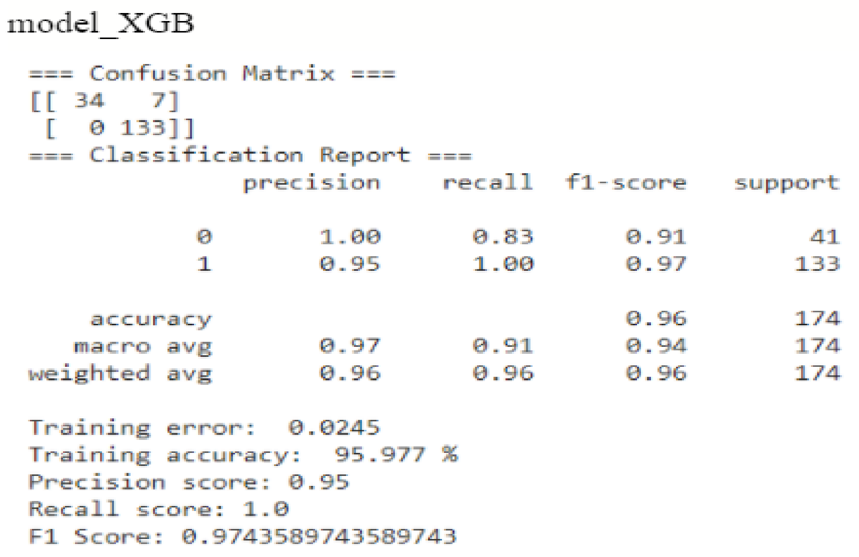






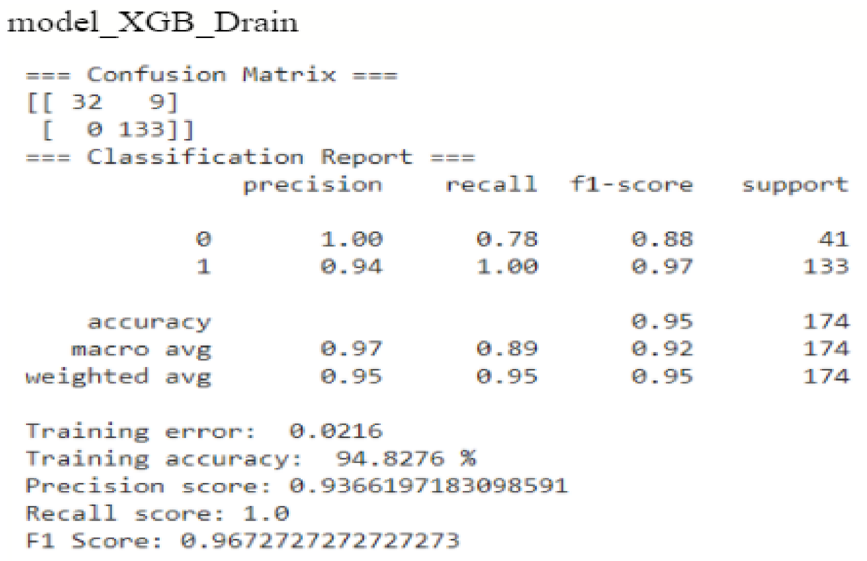
As the images above were showing, at a data size of 416, the performance of $model_{XGB}$ which was XGB without Drain3 processed logs was better than the performance of $model_{XGB_D}$ which was XGB with Drain3 processed logs.

With 868 logs:



model_XGB

```
=== Confusion Matrix ===
[[ 34   7]
 [  0 133]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      0.83      0.91        41
           1       0.95      1.00      0.97       133

    accuracy                           0.96       174
   macro avg       0.97      0.91      0.94       174
weighted avg       0.96      0.96      0.96       174

Training error:  0.0245
Training accuracy:  95.977 %
Precision score: 0.95
Recall score: 1.0
F1 Score: 0.9743589743589743
```

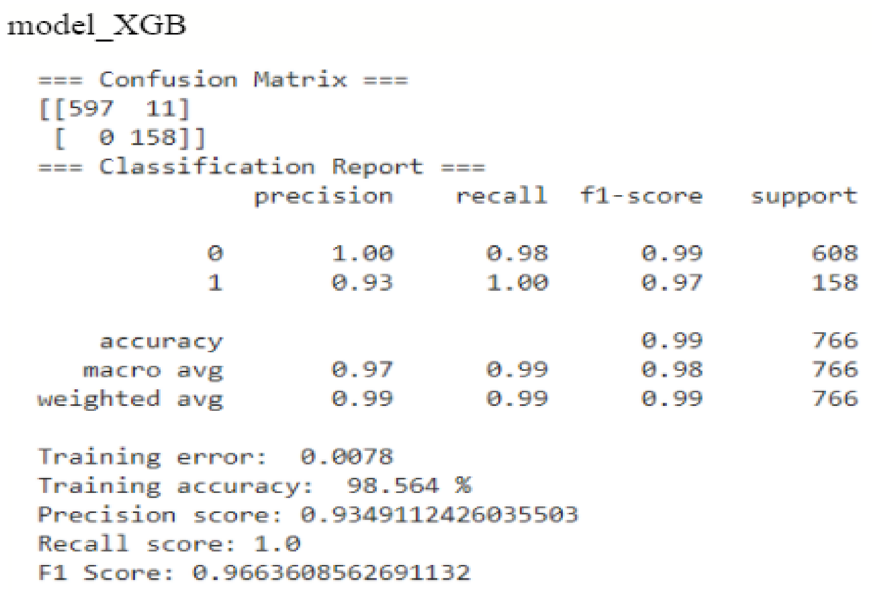


model_XGB_Drain

```
=== Confusion Matrix ===
[[ 32   9]
 [  0 133]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      0.78      0.88        41
           1       0.94      1.00      0.97       133

    accuracy                           0.95       174
   macro avg       0.97      0.89      0.92       174
weighted avg       0.95      0.95      0.95       174

Training error:  0.0216
Training accuracy:  94.8276 %
Precision score: 0.9366197183098591
Recall score: 1.0
F1 Score: 0.9672727272727273
```

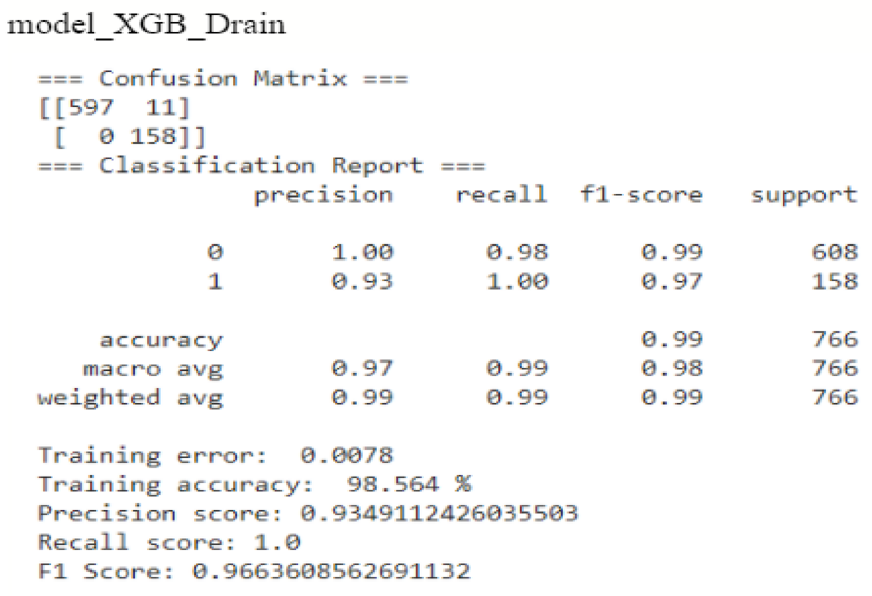
In this image, the performance of the two models was closer, comparing to the previous experiment's result. Therefore, the data size was increased in the next experiment to prove whether the performance of $model_{XGB_D}$ would have the same performance model_XGB

With 3826 logs:



model_XGB

```
=== Confusion Matrix ===
[[597  11]
 [  0 158]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       608
           1       0.93      1.00      0.97       158

    accuracy                           0.99       766
   macro avg       0.97      0.99      0.98       766
weighted avg       0.99      0.99      0.99       766

Training error:  0.0078
Training accuracy:  98.564 %
Precision score: 0.9349112426035503
Recall score: 1.0
F1 Score: 0.9663608562691132
```

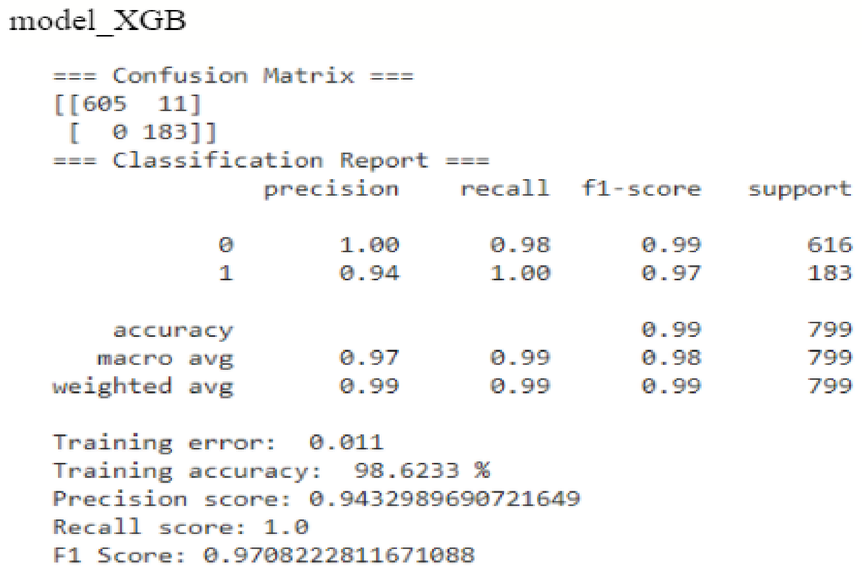


model_XGB_Drain

```
=== Confusion Matrix ===
[[597  11]
 [  0 158]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       608
           1       0.93      1.00      0.97       158

    accuracy                           0.99       766
   macro avg       0.97      0.99      0.98       766
weighted avg       0.99      0.99      0.99       766

Training error:  0.0078
Training accuracy:  98.564 %
Precision score: 0.9349112426035503
Recall score: 1.0
F1 Score: 0.9663608562691132
```

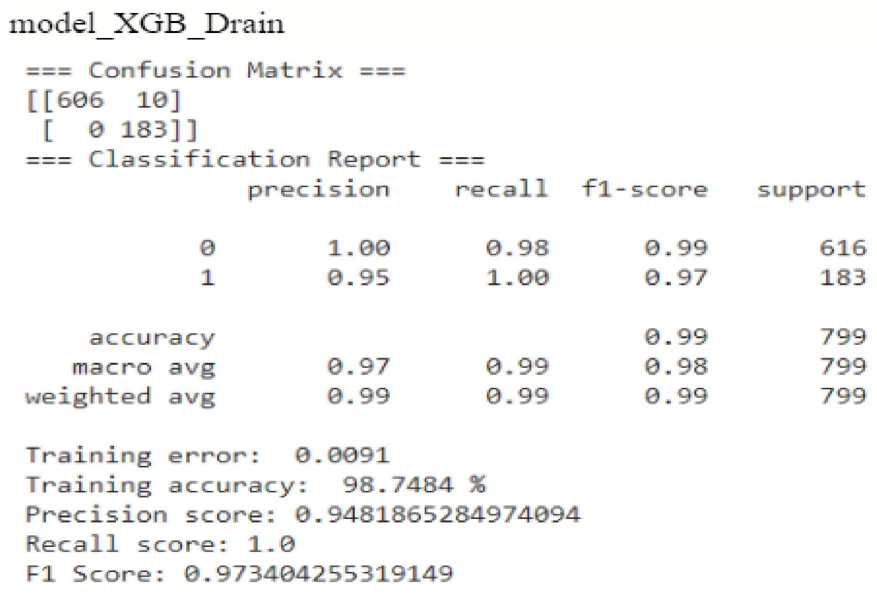
The third experiment proved that the performance $model_{XGB_D}$ was equivalent to the performance of $model_{XGB}$. In the next experiment, the data set was increased again.

With 3994 logs:



model_XGB

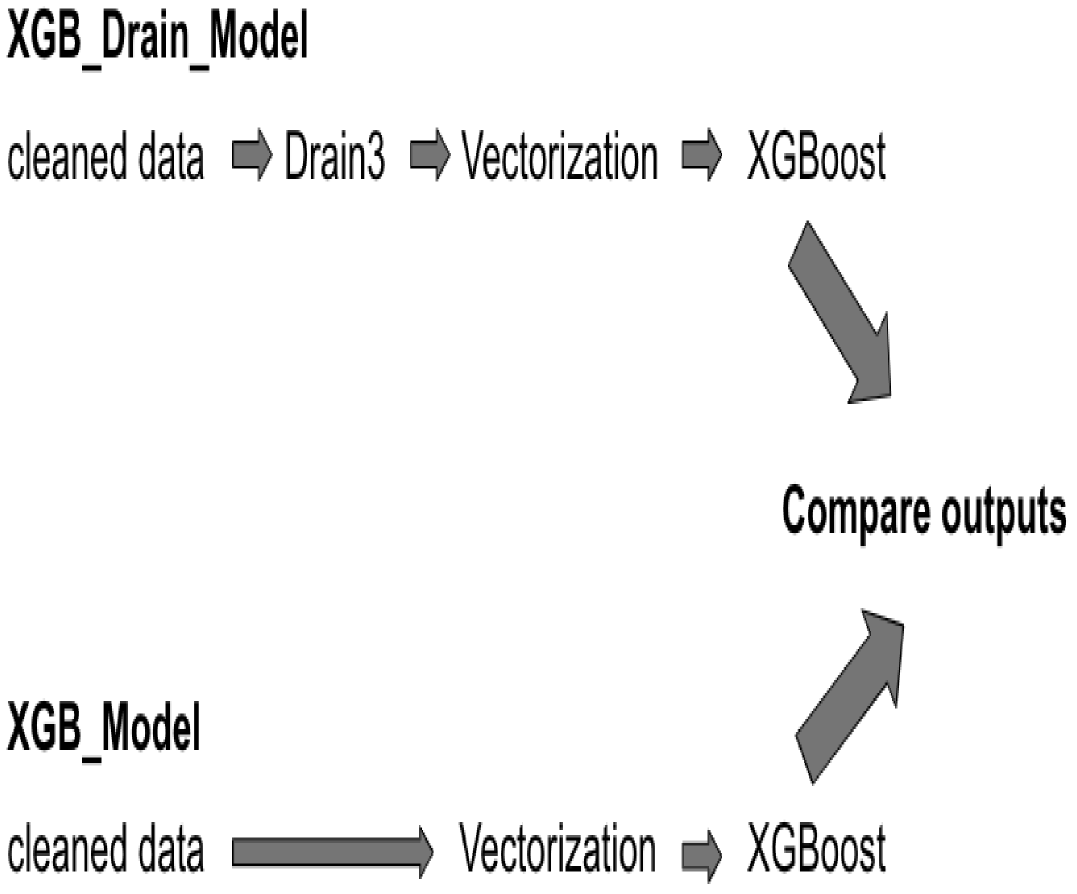
```
=== Confusion Matrix ===
[[605  11]
 [  0 183]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       616
           1       0.94      1.00      0.97       183

    accuracy                           0.99       799
   macro avg       0.97      0.99      0.98       799
weighted avg       0.99      0.99      0.99       799

Training error:  0.011
Training accuracy:  98.6233 %
Precision score: 0.9432989690721649
Recall score: 1.0
F1 Score: 0.9708222811671088
```



model_XGB_Drain

```
=== Confusion Matrix ===
[[606  10]
 [  0 183]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       616
           1       0.95      1.00      0.97       183

    accuracy                           0.99       799
   macro avg       0.97      0.99      0.98       799
weighted avg       0.99      0.99      0.99       799

Training error:  0.0091
Training accuracy:  98.7484 %
Precision score: 0.9481865284974094
Recall score: 1.0
F1 Score: 0.973404255319149
```

In the last experiment, $model_{XGB_D}$'s performance slightly exceeded $model_{XGB}$'s. To conclude, the performance of $model_{XGB_D}$ would be better than the performance of $model_{XGB}$ after a threshold. In this setup, the threshold is a data size of 3826.



The image above visualizes the workflow of our project notebook. It outlines the steps we take for both processed logs and unprocessed logs and then compare the results accordingly.

6 Conclusion

6.1 Limitations

The biggest drawback for our project is the limited amount of data logs we had access to for testing. This was a drawback because we did not anticipate our classifier to predict the same label for all logs with the original amount of logs we had. This result made us believe that our classifier would not work appropriately. Another issue with this is that the manual parsing may not be generalized for other build logs that would be processed in the future which can lead to similar results as the unprocessed logs.

6.2 Future Work and Final Thoughts

From the research done, it is clear the Drain3 is effective when a large data-set is provided. This is as expected as Drain3 and machine-learning performs better with larger data-sets. From here, further testing would be useful to prove that this hypothesis works concretely. After, it would be helpful to compare to other log templating APIs to determine Drain3 is the best API. Lastly, these findings could be used by Red Hat to implement into solutions or for further investigation.

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

Explanation of Drain3: https://jiemingzhu.github.io/pub/pjhe$_{i}cws2017.pdf$

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$_{i}cws2017.pdf$

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