INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 5, Issue 2)
Available online at: www.ijariit.com

Anomaly detection in code base

Sudipto Nandan
sudipto.nandan@gmail.com
Oracle India Private Limited, Bengaluru, Karnataka

ABSTRACT

Any product when under development, goes through numerous changes before finally being released to the customer. While these changes are being done, it adds new features, modifies existing ones. How do we know if a product is in good shape to be released? Yes, we test the product, run the existing unit, functional, performance tests etc. What if the number of tests is in 10000s. How do we analyse each test result? Is there an automated way to detect the overall health of the product using the results of regression tests? Anomaly Detection using machine learning algorithms gives us a way to find out the overall health of the product. Using Anomaly Detection, we can quickly find out about the code base and if new changes should be allowed in before the existing code base is stabilized. It helps to determine, how far the existing code base is away from being released to the customer. It can help the code base to be almost always stable irrespective of the number of code changes that are merged into it.

Keywords— *Anomaly detection, Regression results, Isolation forest, Principal Component Analysis*

1. DEVELOPMENT AND TESTING

A software development process typically consists of coding, testing, release cycles. The cycles – the duration spent on each of this component decreased as days passed. Time has now come, with agile development methodology, the sprint cycle is reduced to 2-3 weeks. So, with this 2-3 week of time, new features are added to the existing product, they are tested, and they are certified for release. This may sound okay for smaller sized application but turns out to be a real problem for bigger, enterprise level products.

Automated testing has come a long way. Almost every product developed is tested using automated test suites. New testing framework helps the QA team to generate new automated tests very quickly. With the advent of cheap and reliable hardware, testing a product is no more a huge constraint. Thus, for any enterprise product, automated tests are a very crucial component. With time, the number of automated test suites starts increasing. They sometimes reach to humungous quantity. For each code base, the automated test suites run for 1000s of hours for verification. The test failures are analyzed manually and then we try to get a feel of the code base health.

If there is an automated way to get an overall feel of the code base, it helps the release management team to determine if the product is in a good shape for release or needs more efforts on fixing the code. We achieve this using machine learning algorithm– Isolation Forest.

2. ISOLATION FOREST

This algorithm tries to find out the outliers from a given set of data. These outliers are also called an anomaly. The data can be of higher dimensions as well, i.e. it can have multiple features too. These features are nothing but characteristics of each codebase, what has gone into the code base, what is the outcome of the code base. Isolation Forest analyzes this data to find the outliers.

From the given set of data, the algorithm randomly selects a set of data and features, finds out its minimum and maximum value and randomly chooses the split value. Based on the split value, it partitions the data and forms a tree structure. For a given tree, we determine the "normality" of the data based on its path length in the tree from the root. The process is repeated multiple times based on multiple split values. The average of the path length for each of such trees are calculated. For a data value (observation), if it is anomalous, the path length will most likely to be very short. Thus, it detects which value is "outside" the cluster of good values and which falls "inside" the cluster.

Example: For a given set of data [2,3,4,5,6,103], the path length of 103 is just 1, while for others it is 1 or more.

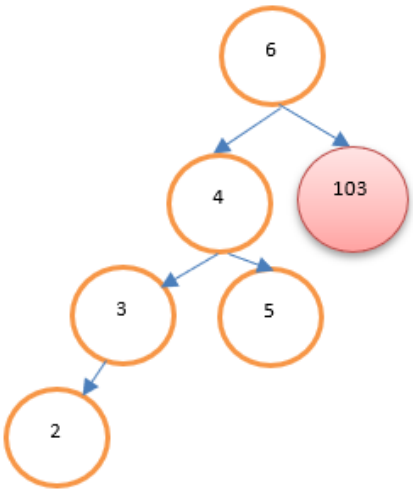

Fig. 1: Isolation Forest algorithm

2. CODE BASE HEALTH

2.1 Data Set

As a first step, we need to collect all the available data for the given code base. The data includes

(a) Total Number of tests
(b) Number of success and failures
(c) Total Execution time
(d) Any other error reported – e.g. any additional log file generated for analysis
(e) Count of additional files e.g. images of error messages etc.
(f) The amount of disk space the results is consuming
(g) Amount of code changes went in – it can be a number of code blocks, a number of code files or any other representation of the number of code changes happened. Etc.

The above data collected for a previous significant amount of time, say last one year. Each observation is denoted using a codebase id for identification. The data should be of the numeric datatype.

2.2 Data Preprocessing

The data collected needs to be cleaned and filtered. All the null value data is replaced appropriately, e.g. either with 0 or 0.0 etc. The data which is not supposed to be included in the input dataset should be filtered out too.
The filtered data is put into a data frame: X_train

The dataset is then normalized and scaled.

```
X_train_normalized = preprocessing.normalize(X_train)
X_train_normalized = preprocessing.MinMaxScaler(\
                     feature_range=(0, 1)).fit_transform(\
                                          X_train_normalized)
```

2.3 Principal Component Analysis

We determine the principal components of all the features using principal component analysis. This helps to reduce the redundant data and reduce the complexity of the model to be generated in the next step. The Principal Component Analysis will generate the co-variance ration of each of the components. We select only those components which result in maximum covariance and ignore those data/features which have less covariance.

2.4 Generating Model

Using *sklearn.ensemble.IsolationForest* we generate the model. Among the various inputs, we mention the number of observations to be considered for each tree, the maximum number of features to be considered. Once the model is generated, we use a *fit()* function to predict whether the given code base is anomalous or not.

decision_function() is a function which also gives us a number determining how far the given codebase is from the boundary of a cluster of good codebases. Higher the score, more away it is from the boundary and thus more anomalous it is.

Figure 2 shows the anomaly score histogram on the given set of 150 code bases. The positive values determine the good code base and the ones with negative values shows the anomalous code bases and thus corrective actions to be taken on those code bases.

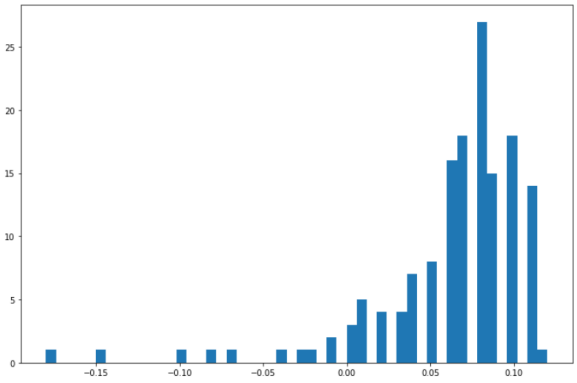

Fig. 2: Anomalous Score's Histogram

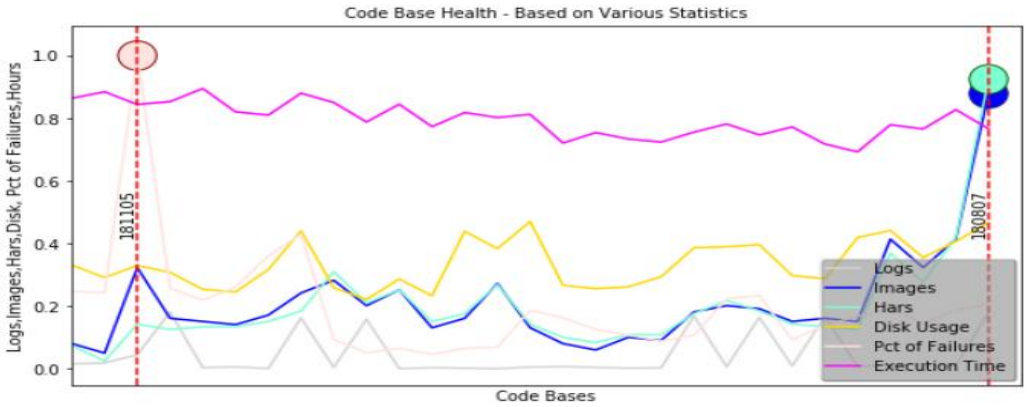

Fig. 3: Various Statistics with predicted values

Code Base Id	# of Tests	# of Failures	% of Failures	Total Execution Time (In Hours)	# of Error Images	# of HARs	Disk Space Usage (In GB)	Anomaly Score*
190312	9326	19621	2.62	21422	61	5768	3048.9	8
190307	10012	13311	1.58	21623	43	829569	3067.12	11
190305	9300	12725	1.67	20674	40	26827	2857.14	
190304	9908	11690	1.47	22096	46	5630	3072.81	
190228	10903	13701	1.48	24577	49	655288	3469.76	2
190226	9915	13945	1.75	21912	60	9650	3235.74	

Fig. 4: Various statistics for the anomalous code base

3. CONCLUSION

The anomaly detection algorithm can be applied on various features and can help the development team, release management team or test team to detect the anomalous code base. Once detected, corrective measures can be taken before the next codebase is released. In case all the test suites results are not available in a short period of time, partial regression results can be taken into consideration and extrapolated. The extrapolated data also works quite well in determining if something is going wrong in the code base.

5. REFERENCES

[1] Pattern Recognition and Machine Learning - Christopher M. Bishop
[2] Real World Machine Learning – Henrik Brink, Joseph W Richards, Mark Fetherwolf
[3] Foundations of Machine Learning – Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar
[4] Hands-On Machine Learning with Scikit-Learn & TensorFlow – Aurelien Geron
[5] Wiki: Principal Component Analysis Explanation - https://en.wikipedia.org/wiki/Principal_component_analysis
[6] Scikit Learn Documentation for Outlier Detection https://scikit-learn.org/stable/modules/outlier_detection.html#isolation-forest

BIOGRAPHY

Sudipto Nandan
Consulting Member Technical Staff,
Oracle India Private Limited, Bengaluru, Karnataka, India

