INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 6.078
(Volume 6, Issue 1)
Available online at: www.ijariit.com

Performance Benchmark using Machine Learning

Sudipto Nandan
sudipto.nandan@gmail.com
Oracle India Pvt. Ltd., Bengaluru, Karnataka

ABSTRACT

Software Performance testing is one of the important aspects of testing a product. It is more important when we have a cloud application, i.e. not a standalone application but a web application. Many of us use various tools like JMeter to test the performance of different web requests. But how do we analyze the results of JMeter? A failed Request is easy to catch. What about performance degradation? How do we know if a request performed worse than the previous runs or not? How do we take into consideration other environmental parameters – memory, processing power, operating system limits, etc? Performance benchmark using machine learning is one of the ways using which we can take care of some of these manual study and detection of performance regressions which may otherwise go unnoticed.

Keywords— *Anomaly Detection, Benchmarking, Isolation Forest, Performance Testing, Principal Component Analysis, Regression Results, Web Application*

1. PERFORMANCE TESTING

Most of the web applications are tested not only for its functionalities but also for how well it performs. Performance testing determines the response time of any application; the speed with which any request is processed given a network bandwidth and load. Apart from speed or response time, it also helps us to determine the scalability factor of the application, i.e. how well it can scale, how many users can use the application simultaneously. It can also help us to determine the stability of the application. When the load is huge, the application may behave or perform differently. It is acceptable? The aim of performance testing is to remove any kind of performance bottlenecks and come up with a set of metrics which acts a guideline for future scaling or releases of the product.

Performance testing can happen in a testing environment as well as in production environment. In the testing environment, the load is created by the tester artificially; while in a production environment the load is not artificially generated but the actual load at that point of time is taken into consideration. In all different environments, the parameter values are different, and the results are different too. Comparing and analyzing these metrics can be a challenging job.

Finding an automated way to process these data is a bigger challenge. One easier way to is to gather each of these sets of data, find a mean and variance of one such feature (say for the data “throughput”), from the whole dataset) and then compare out new “throughput” data with the mean “throughput” value. This sounds great when the number of parameters we are discussing is lessexample, 4 or 5 parameters. When the number of variables increases to 25 or 30 or even more, calculating and comparing each of these parameters become more difficult. Comparing independently also excludes their inter-dependencies.

A better way is to process them using a machine learning algorithm – Isolation Forest to find an anomaly. This is best suitable when we have a very large number of a dependent as well as independent parameters.

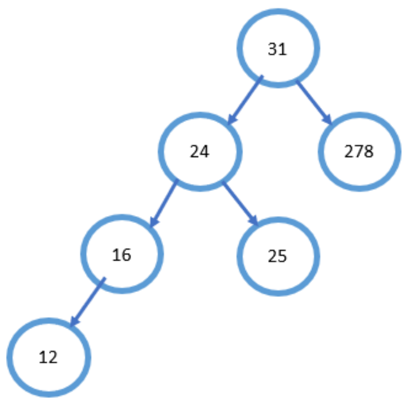
2. ISOLATION FOREST

This machine learning algorithm analyzes a given set of metrics and filters out the outlier. We term these set of outliers as anomalous data – or simply anomaly. The input set of data contains multiple parameters. They are not just few but many of them. Each of these parameters or features are characteristics of that set of data. They determine various aspect of data. E.g. consider the following output of JMeter.

```
Summariser: summary = 500 in 00:03:27 =
2.4/s Avg: 4047 Min: 1039 Max: 4686 Err: 0
(0.00%)
```

It contains seven different parameters of a single run. And some of the above parameters are dependent on another as well. i.e. if the value of one parameter increases, the value of other increases too. The ISOLATION Forest algorithm analyzes such a set of many data and tries to determine the anomaly.

From the given set of data, the algorithm randomly selects a feature. It then finds out its minimum and maximum value of that feature. Subsequently, it randomly chooses the split value. Based on the split value, it partitions the data. This partitioning goes on and ultimately it forms a tree-like structure.
Example:- For a given set of data [12,31,24,25,16,278]
For one split value 31, a tree will look like below



The algorithm determines the length of the path of each node from its root. The same process of tree generation and calculating every node's path length is repeated for multiple split (root) value. The average of all such path lengths is considered. For any data to be anomalous, the path average length will be smaller than others. This determines the "normality" or "abnormality" of the data. Thus, outlier is detected. An outlier's value is "outside" the cluster of good values.

3. ANALYZING PERFORMANCE DATA

3.1 Data Set

This is the very first step. We need to determine the metrics that we need to analyze. The metrics should include various numerical as well as non-numeric data. We need to collect all the available data for multiple performance tests. The data includes

a. Total Threads
b. Total number of loops
c. Total Execution time
d. Throughput
e. Average Request process time
f. Minimum Request process time
g. Maximum Request process time
h. Number of failures
i. Memory size
j. Processing power
etc.

Each of the above data needs to be collected and converted into a numeric type. The more the amount of data the better it is. We must keep in mind that each such data should belong to the same version of the application. Multiple versions of application behave in different ways. So, the data should not be mixed.

A sample set of data looks like below

```
total,time(sec),throughput,average,min,max,err
5000,2043,2.4,4077,1123,4765,0
1000,427,2.4,4184,1053,4673,0
2000,898,2.3,4080,967,4562,0
5000,2041,2.4,4262,410,7748,0
2500,1043,2.4,4076,1351,5015,0
3000,1264,2.5,4078,1063,4739,0
5000,2044,2.4,4577,1057,4665,0
5000,2211,3.4,4765,2051,9721,0
4000,2041,2.4,4072,1274,4969,0
1000,492,2.3,4078,1277,4926,0
```

Some of the non-numerical values are also taken into consideration when the performance benchmarks are executed across various environments. e.g. CPU Information, Operating System Limits etc. A sample set of such information gathered when JMeter test is executed, are shown below.

```
2020-01-01 12:12:39,004 INFO o.a.j.JMeter:
Setting JMeter property: threads=10
2020-01-01 12:12:39,004 INFO o.a.j.JMeter:
Setting JMeter property: loops=50
2020-01-01 12:12:39,009 INFO o.a.j.JMeter:
Copyright (c) 1998-2018 The Apache
Software Foundation
2020-01-01 12:12:39,009 INFO o.a.j.JMeter:
Version 4.0 r1823414
2020-01-01 12:12:39,010 INFO o.a.j.JMeter:
java.version=1.8.0_171
2020-01-01 12:12:39,010 INFO o.a.j.JMeter:
java.vm.name=Java HotSpot(TM) 64-Bit
Server VM
2020-01-01 12:12:39,010 INFO o.a.j.JMeter:
os.name=Linux
2020-01-01 12:12:39,010 INFO o.a.j.JMeter:
os.arch=amd64
2020-01-01 12:12:39,010 INFO o.a.j.JMeter:
os.version=1.2.2-4.8.6.x86_64
2020-01-01 12:12:39,010 INFO o.a.j.JMeter:
Max memory     =1073741824
2020-01-01 12:12:39,010 INFO o.a.j.JMeter:
Available Processors =24
```

3.2 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)
```

3.3 Principal Component Analysis

PCA – Principal Component Analysis is used to determine the minimum number of features to be fetched for a maximum variance. 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 ratio of each of the components. We select only those components which result in maximum covariance and ignore those data/features which have less covariance.

```
pca = PCA(n_components=7)
pca.fit(X_train_normalized)
X_reduced =
pca.transform(X_train_normalized)
pca.explained_variance_ratio_.cumsum()
[0.81011924 0.97088256 0.99269478
0.99984594 0.99999399 0.99999975 1.]
```

In the above sample, the cumulative variance is shown. We can see that first 3 rows are enough to capture the maximum variance. Thus, we set the n_components to 3 for our future calculation.

3.4 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 the *fit()* function to predict whether the given metric is anomalous or not.

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

```
rng = np.random.RandomState(42)
clf =
IsolationForest(max_samples=no_of_rows,
      max_features=2,
      contamination=0.1,
      behaviour='new',
      random_state=rng)
clf.fit(X_train_normalized)
y_pred_train = clf.predict(\
               X_train_normalized)
scores = np.round(\
         clf.decision_function(\
         X_train_normalized),2)
```

Once the model is executed, we get a score for each data set. Any score less than 0 means that it is an anomalous row. Lesser the value, farther it is away from the good set of values. We separate out such runs and mark them as anomalous.

```
-------------------------------------
Anomalous Run among last past 100 runs:
-------------------------------------
-------------------------------------
     total  time  throughput  average  min
max  err  Score
48   5000  2037       2.5     4064  408
5862    0   -1.2
-------------------------------------
```

4. CONCLUSION

The methods used for performance testing varies but the intention remains the same. It helps us to demonstrate that our application meets certain pre-defined performance criteria under various conditions and environments. It also helps to determine the performance bottlenecks. In the cloud world, performance testing cannot be neglected. The ISOLATION forest algorithm can be applied to multiple interdependent performance metrics and can help us automatically detecting all kinds of performance anomalies. This will reduce the error-prone manual intervention for analyzing the performance tests output. This analysis can be integrated with the actual run of performance test itself so that the output of the test is not just the numbers but also the analysis, the score about how good or bad the overall performance is.

5. REFERENCES

[1] Sudipto Nandan, "Anomaly detection in code base, International Journal of Advance Research, Ideas and Innovations in Technology, 5.2 (2019). www.IJARIIT.com"

[2] Christopher M. Bishop "Pattern Recognition and Machine Learning"

[3] Henrik Brink, Joseph W Richards, Mark Fetherwolf, "Real World Machine Learning"

[4] Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar, "Foundations of Machine Learning"

[5] Aurelien Geron, "Hands-On Machine Learning with Scikit-Learn & TensorFlow"

[6] Wiki: "Principal Component Analysis" , https://en.wikipedia.org/wiki/Principal_component_analysis

[7] Scikit Learn: "Outlier Detection", https://scikit-learn.org/stable/modules/outlier_detection.html#isolation-forest

BIOGRAPHY

Sudipto Nandan
Consulting Member Technical Staff,
Oracle Corporation