**INTRODUCTION**

Logs provide a wealth of information to maintain, manage and troubleshoot IT systems. They provide a concise means of recording information at vital points during everyday operations.

In recent times however the sheer amount of log data generated by modern systems, in addition to the complexities of parallel processing and containerisation makes manual analysis of log data almost impossible. So, in such cases we turn to solutions such as Machine Learning to leverage statistical models to do most of the heavy lifting for us.

In this project I will train a model to detect anomalies in the given log data, as well as provide a means of providing analysis for the same.

**THE DATA**

The data consists of:

1. Kibana logs:

The kibana logs used here consist of 38,000 individual log events occurring over a 24-hour period. It contains information regarding the reason for logging, metadata about its logging process along with the timestamp for the same.

1. Metrics:

These metrics, recorded approximately every second, give us an aggregated view of the usages of individual processes as they fluctuate with time.

**DATA PREPARATION**

Before we can start an in-depth analysis of the dataset, we have to perform some basic preprocessing operations to convert it into a more convenient format to work with.

The JSON data is converted to pandas DataFrames for further operations.

Merging kibana and metric logs - Having log data with timestamps accurate down to the millisecond, very few logs will actually have an exact match on timestamps - just 22 out of 38,000 actually. So we resort to merging the kibana logs with metric data of the *nearest timestamp* rather than looking for a perfect match.

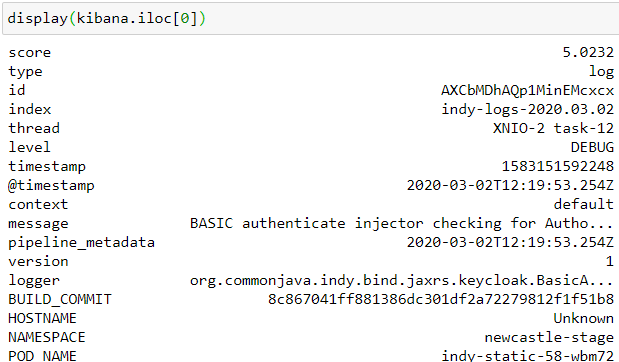
Additionally, the metrics tend to have a fair amount of missing values in records, so we will fill in with the metric value from the record just previous to it. In doing so, we assume that since the data is a time series no change in the reading has occurred since the last seen instance.

**EXPLORATORY ANALYSIS**

Exploratory analysis is an important step in understanding the data that we are working with and getting a better insight into what could be the most useful features to pass along to the machine learning algorithm.

**Kibana Logs**

Let’s see the kibana data that we’re working with:



kibana record from dataframe

We start out by dropping any columns which have only one unique value as they won’t provide any information to our algorithm, then will investigate the features more in detail:

* timestamp: UNIX timestamps which will be used to denote the order of log events.
* index: index given to each record, which on analysis depends on the date of logging.
* Score : On analysis this field remains constant for a given date, however, while this does not distinguish between records logged on the same day, it could be useful in anomaly detection across different days.
* Id: A unique alphanumeric id given to each log.
* POD\_NAME

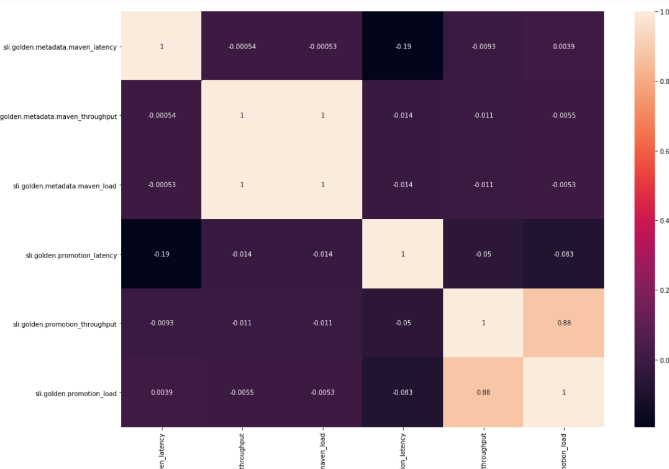
As the name suggests, ‘POD\_NAME’ gives the name of the pod that made the log record. However, since our metrics are all aggregated it wouldn’t make sense to tag it to a specific pod so this column is also not required.

* level: This tells us the nature of the log entry and will be widely used in later analysis.
* logger: Provides additional information about the log data.
* thread: The type of thread that the log pertains to
* Message: Tells us why the log was created and can be used to analyse anomalous conditions during operations.

**Metrics logs**

The metrics dataframe we created earlier has 162 fields per record which make it difficult to investigate each individually, so we try to statistically determine which columns hold the most information.

We plot the correlation matrix, if two or more columns vary similarly with respect to time (ie have high correlation) each often doesn’t provide much more information than one another so only one may be retained with the same net effect.



Correlation heatmap of metric data.

For example, we see how many of the latencies have high correlations with each other and hence we may drop most of these.

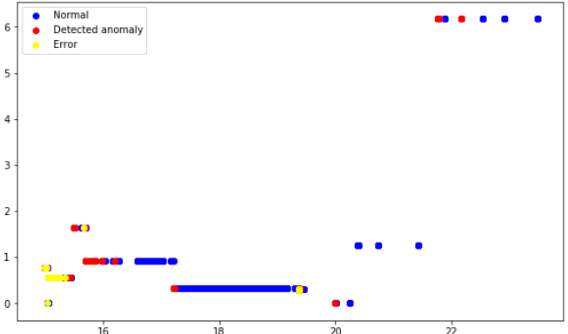
**MODEL SELECTION**

Of the models tested in the analysis notebooks an SVM approach was chosen for anomaly detection.

**One Class SVM**

A one class SVM is a commonly used unsupervised learning algorithm which detects anomalies by learning the boundaries of the “normal” data and then labelling the rest as anomalies.

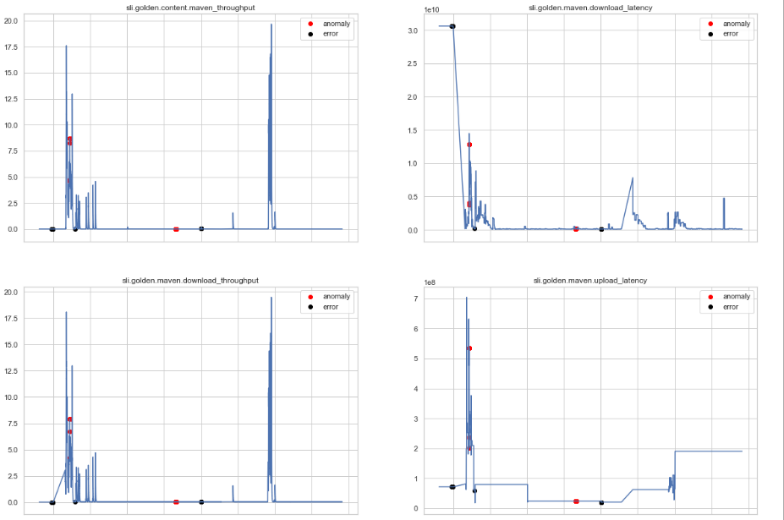
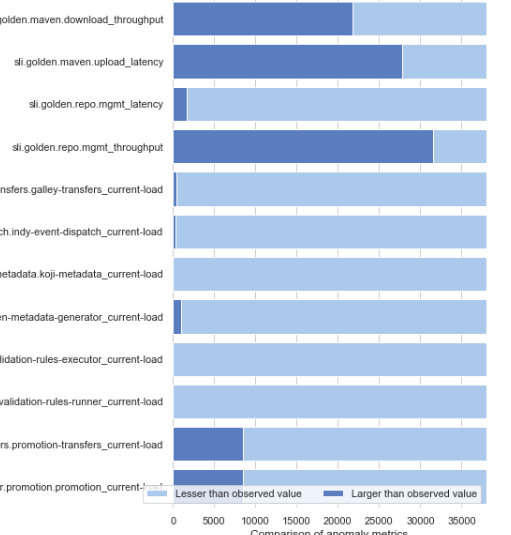
The anomalies detected by our model can be visualised it against metric data as:



**RESULTS**

The upside of adding metric data to the kibana log reports is that we’re now able to analyse the anomaly points to gain a basic insight of them. No longer are they just streams of numbers that fluctuate with time, but now we are able to drill down deeper into what activities are associated with the resource usages recorded by the metrics.

We can view when the anomalies occur against the metrics that we are tracking on a per-second basis.

* 1. anomalies plotted against metric data readings (b) ranks of the metric reading wrt. observed values

We can visualise how the metrics of a specific anomaly point compares in value to those normally seen in our logs, showing how the anomaly tends to have metrics towards the extremes of observed values. Finally using kibana logging data we may even graph the events leading up to an anomalies occurrence.



pie-charts showing the breakup of events prior to the occurrence of an anomaly based on their thread type and message