CS230 Project Report
Cloud Anomaly Detection

Dhruv Lakhani
dhruvsl@stanford.edu

Rishabh Kumar
rkumar92@stanford.edu

June 2020

1 Abstract

On services deployed on cloud, situations can arise which can cause the operating state to abruptly go out of normalcy. Consider a situation where the number of graphic images uploaded to a platform like Twitter spikes on a Christmas eve, or a popular keyword suddenly disappears from Google search. These out-of-the-normal states are anomalies. There are other common situations also where such anomalies occur quite frequently: actions of malicious agents like bots and spammers, after a new software release, etc. We tried to approach this problem with various existing techniques, and custom-built models. As part of this project, we identified models which worked better than others but none of the techniques performed as good as the existing statistical techniques, which lead us to investigate why. This report documents our experiments and investigation.

2 Introduction

In this section we will talk about introduction to Cloud Anomaly Detection. As we discussed earlier, early detection of anomalies is crucial for supporting use-cases for all stakeholders like availability of user data, availability of the cloud service itself, detection of adversaries like bots or spammers, some incident in data centers etc. This was one of major motivation for both of us as we both work on cloud services which face these problem quite often. As part of this project we developed deep learning models to see if we could possibly bring improvements over existing statistical techniques for detecting anomalies in cloud services. We used Yahoo dataset to come up with an anomaly detection model. We did some thorough statistical analysis for time-series data. We applied existing Twitter [9] statistical algorithm to establish a baseline before applying a deep learning based solution.

3 Related Work

Much of the existing deep learning work on time-series data is either related to forecasting or custom fitted to the needs of the targeted problem (as described in [2]). Along with high precision, there are other challenges with anomaly detection for cloud metrics: the data could be more granular across the time dimension and different detections are required for different time scales (short timescale for large noticeable abruptions, long timescale for more silent issues) . For this reason, cloud systems have developed a lot of custom statistical techniques, e.g. [9]. Few papers such as [6] and [10] present a forecasting solution which focuses on predicting uncertain events, a technique which works very well for distributing driver workload based on the number of trips – a dataset much less granular than cloud metrics. Twitter has adopted statistical approaches built on ESD as described in [9] and an improvement based on recursive-ESD [7]. Other papers talk about exploring anomalies in multidimensional space such as [8], but our focus is on industrial application to detect anomalies in cloud service metrics.

4 Dataset and Related details

As mentioned before we used Yahoo dataset, which has synthetic and real time-series with labelled anomalies. The dataset is divided into 5 benchmark folders and in total it contain 371 files.

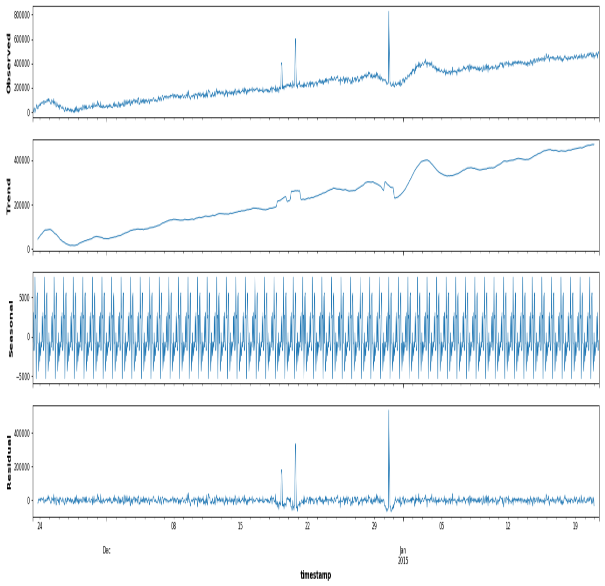

Figure 1: Cumulative time-series analysis

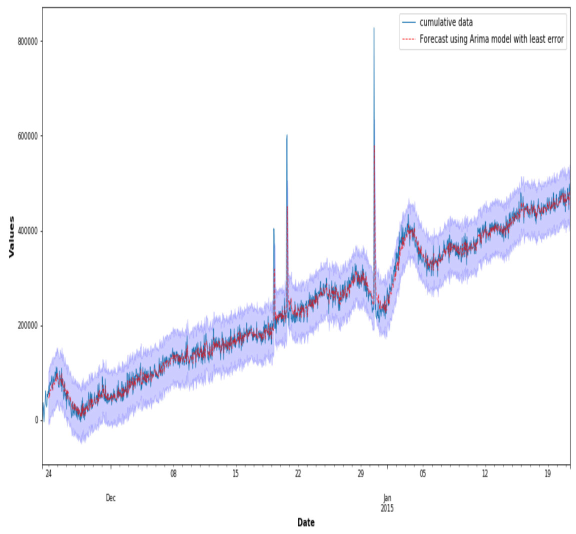

Figure 2: Confidence Interval using ARIMA

Each file has hourly time series data along with labelled anomalies for a quarter. One benchmark contains real data (from an actual cloud service) while others have synthetic data. Real data based benchmark consists of timestamp, value and anomaly boolean field while synthetic ones have, in addition to the above mentioned fields, varying noise and trends with three prespecified seasonalities. We also did cumulative time-series analysis on dataset w.r.t seasonality & trend which can be seen from the following figure 1

4.1 Analysis

We ran more than 50 ARIMA models to see which one is the closest fit to the time-series which we were analyzing. We analyzed all types of models which includes both additive and multiplicative seasonal effect and closest model was found to be ARIMA(1, 1, 1)x(0, 0, 1, 12). This analysis also helped in giving us a relevant confidence interval as can be seen from 2. We are planning to use these intervals for comparison w.r.t. the one predicted by our method.

4.2 Preprocessing and Selection

As we didn't have time-series for multiple years we decided to use all benchmark folder and split the whole data into a 80:10:10 split. 80% for training, 10% for validation and the remaining 10% for testing. We did time-series values normalization using Scikit's MinMaxScaler.

5 Methods

We ran Twitter algorithm described in [9] on Yahoo dataset to see some preliminary results so that we can use it later to compare our own algorithm's performance as baseline model. We also produced a vanilla LSTM, which performed decently on a small dataset. However, when we attempted to run the method on a larger data-set it performed poorly: the model had high MAE on a few sequences.

5.1 Deep Anomaly Detection using auto encoder

We structures overall approach for time-series based anomaly into following sub-tasks:

- **Forecasting:** We designed a deep prediction model for forecasting the next time-series element.
- **Confidence interval estimation:** We followed a Bayesian approximation based technique to estimate the confidence interval for the forecasting prediction.

5.1.1 Autoencoder Forecasting

With Vanilla LSTM model we deduced that the model had high bias and we attempted to fix it via deeper network. We tried increasing the size of the time-series segments that were being trained, increasing the number of network layers, and increasing the number of neurons in each layer. But deeper models didn't produce significantly better results. Upon investigation, we realized that in the approach our model is trying to learn a many-to-one function (since the output is a single next value). This approach does not work very well for time-series data, like the one we are trying to model. The reason being that data from production metrics is often driven by factors which cannot be deterministically modelled. Hence a single anomaly point which is a part of the time-series can have large effects during the backwards propagation phase. It was evident that instead a many-to-many model would be a more meaningful choice for efficient learning.

A many-to-many sequence forecasting model would be confusing for forecasting elements, and there were other issues also related to incorporating other hand-engineered features present with the dataset – like the seasonality data. So we built a multi-stage prediction pipeline with two stages:

1. First we ran an auto-encoder model as described on left side of 3. The auto-encoder model auto-encoded a given segment into an embedding, which is used to forecast the next M elements of the series. We train this auto-encoder model for a large number of epochs.
2. Once we have obtained the auto-encoder model, we used the embeddings produced by the encoder in above as input to a forecasting model which predicts the next element. This model is also provided with other features present in the data. We train the layers of the forecasting model based on the input data. The model is described on the right side of 3.

The above 2-stage approach helps forecast the next element. To predict the next sequence (next few elements), we follow an iterative approach, where the next element predicted serves as input for the next prediction. We use this prediction method and evaluate loss with respect to the original time-series for all series present in the validation/test sets.

5.1.2 Deep DNN and BiRNN Forecasting

We also tried TFLearn's DNN and BiRNN libraries. For DNN we experimented with multiple layers and most optimum hyper parameters were 5 layers with 100 neurons each, and was trained for 600 steps with a batch normalization along with the Adam optimizer. For BiRNN We tried ELU activations along with 100 neurons and 5 layers, but even after introducing temporal nature we were not able to reduce lot of true negatives for both approaches.

5.1.3 Mixed RNN Forecasting

As seen from above auto encoder model above, results don't look promising compared to statistical results. We tried one another approach inspired from [5], where paper talks about predicting sales data on instacart and we saw data similarity between customer sales and anomaly dataset, we tried similar approach on our dataset to come up with better precision and Recall.

In this mixed RNN model, we define $\rho(\text{t}) \in [0, 1]$ to be a function of the time interval between two particular observations in our time series sequence. This value was like a weighted decay and decreased as difference in time increases so that effect of previous time series value diminishes as we proceed with RNN model. This model was basically controlled temporal inference based. ρ allowed us to define a continuous RNN model which adapts to varying time spans between anomalies.

5.1.4 Confidence Interval Estimation

To estimate confidence interval we take inspiration from [4] and [3] and use bayesian approximation. The total uncertainity of the forecasted value can be estimated by the following formula:

$$\mathbb{V}ar[y|x] = \mathbb{V}ar[\mathbb{E}[y|\mathcal{M}, x]] + \mathbb{E}[\mathbb{V}ar[y|\mathcal{M}, x]]$$

,

where y is the forecasted point, x is the input sequence and $\mathcal{M}$ is the trained model.

The parameters on the right hand side are evaluated with the following approach:

1. $\mathbb{V}ar[\mathbb{E}[y|\mathcal{M}, x]]$ is the model uncertainty. To estimate this we add an MC dropout layer (as presented in [4], [3], and [10]) after each layer of the forecasting model. The model uncertainty,

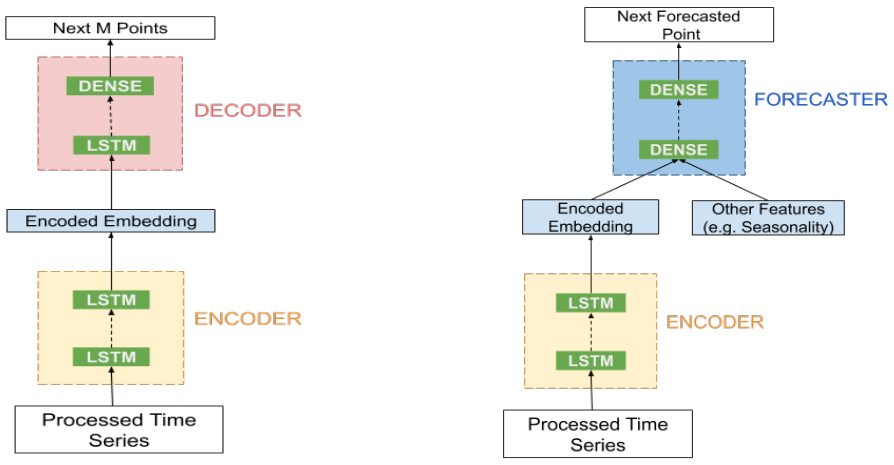

Figure 3: Left: Autoencoder for time-series. Right: Forecasting for time-series

Anomaly %	Precision	Recall
0.005	0.85	0.51
0.01	0.92	0.75
0.02	0.801	0.79
0.03	0.78	0.79

Table 1: Behavior of Precision and Recall with Hybrid ESD algorithm

Model	Precision	Recall
Vanilla LSTM	0.40	0.31
Autoencoder based LSTM	0.46	0.36
TFLearn DNN	0.40	0.39
TFLearn BiRNN	0.40	0.40
Mixed RNN	0.35	0.52

Table 2: Deep Learning model results

σ is estimated as the variance of the forecasted values for a given time-series x:

$$\sigma(x) = \frac{1}{M}\sum_{m=1}^{M}(\mathcal{M}'(x)_m - \overline{\mathcal{M}'(x)}\ )^2$$

, where x is an input time-series, $\mathcal{M}'(x)_m$ is the forecasted value of the model with dropout layers ($\mathcal{M}'$) at the mth experiment, and $\overline{\mathcal{M}'(x)}$ is the mean of all M experiments. We observed that the choice of x determined the uncertainity, so to provide an improvement over this we tried to average this over a validation set: $x_1, x_2, ...x_V$:

$$uncertainty = \frac{1}{V}\sum_{v=1}^{V}\sigma(x_v)$$

,

2. $\mathbb{E}[\mathbb{V}ar[y|\mathcal{M}, x]]$ is the noise of forecasted values w.r.t. to the actual values. This is estimated on a validation set: $x_1, x_2, ..., x_V$, and $y_1, y_2, ..., y_V$ as:

$$\frac{1}{V}\sum_{v=1}^{V}(\mathcal{M}(x_v) - y_v)^2$$

, where $\mathcal{M}(x_v)$ is the forecasted value of the model actual model (without dropout layers).

6 Results

We ran the experiment on a few different models and calculated the precision and recall based on confidence intervals derived from the above method. Table 1 & 2 documents the results along with some statistical algorithms. On the right of 4, we plot MAE against the number of epochs for different approaches.

We obtained confusion matrix for Twitter's Seasonal Hybrid ESD algorithm and results are included in 1. We saw high precision and Recall for statistical algorithm. As seen from results in 2, we can see that autoencoder based LSTM approach performs better than the naive vanilla LSTM. However, we still see that the performance of deep learning based algorithms is not better than the statistical approaches. Upon more careful investigation we see that the deep learning based

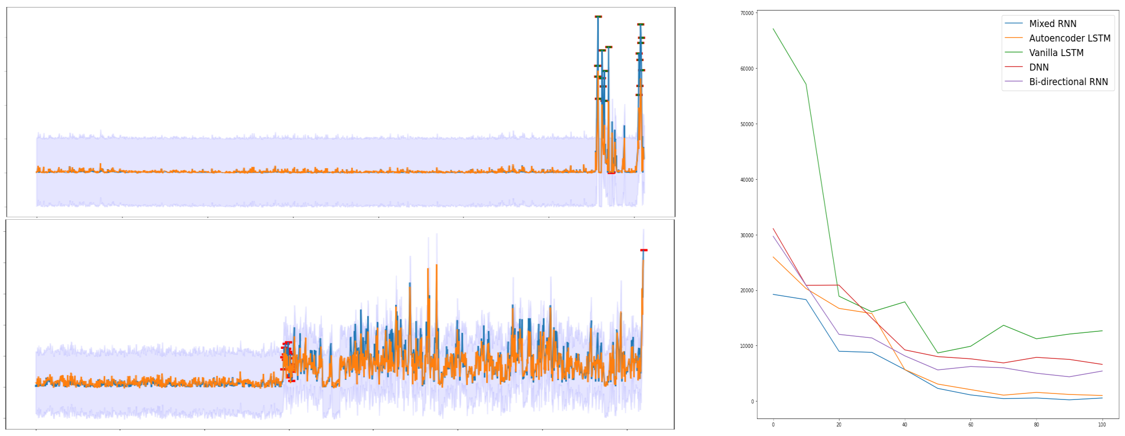

Figure 4: Left: Predicted anomaly for 2 time-series: blue line indicates the actual time-series, orange line is the forecasted time-series, red squares indicate actual anomalies, light blue region is the confidence region, green circles are the predicted anomalies. Top-Left: A time-series for which most of the anomalies were covered. Bottom-Left: A time-series for which none of the anomalies were covered. Right: MAE against number of epochs for various approaches

anomaly detector performs very well on negatives (a data point is not an anomaly) but performs poorly on positives (a data is an anomaly), as can be seen on the left of 4. The reason being two-fold:

1. The number of non-anomaly data points is much greater than anomaly data points, and hence they are likely to produce skewed results.
2. The model still produces relatively high MAE which indicates deviation of the model from the actual results. This was verified by looking at a few time-series that produced very high MAE. MAE increases the inherent noise of the model and increasing the confidence interval. Due to this we observe a lot of anomaly data points to be classified as non-anomaly data-points.

DNN and BiRNN results didn't provide advantages over auto encoders due to the limited set of available features. However we observed decreased precision for Mixed RNN but our recall increased by good amount and this indicates better detection of local anomalies via this approach.

7 Conclusion & Future Work

Here are important conclusion which we noted via dataset analysis and various techniques:

- Our statistical analysis showed that Yahoo data aligns pretty well with one of ARIMA multiplicative model. Our baseline model was based on statistical algorithm on Twitter's real time data and we observed that it performed really well on Yahoo data. Major reason for this was good detection for both global and local anomalies and it was possible due to ESD algorithm which decomposes various seasonal and trend component of time-series.
- Autoencoders provided a way for us to go deeper and provide better accuracy since we were able to get feedback from more number of points but still was far behind in performance.
- Limited features attributed to poor performance of BiRNN and DNN approaches.
- We tried Mixed membership RNN model and it was motivated by the fact that as time proceeds between observations, the next observation may be better modeled as independent from initial distribution of time-series. This gave us better recall and it captured a lot of local anomalies but it performed poorly on precision as it predicted few false local anomalies too due to increase in local effect of time-series.
- We concluded that we need lot of data for e.g few years, if we want deep neural network techniques to perform better than statistical one as 1 quarter of data is not enough to capture information for hidden units. Additionally Cloud data aligns very well with ARIMA models but more data will help in learning various hidden parameter which are not detected by ARIMA models.

8 Contributions

Both of us worked together collaboratively on almost all aspects.

References

[1] Benchmark dataset. https://research.yahoo.com/news/announcing-benchmark-dataset-time-series-anomaly-detection.

[2] R. Chalapathy and S. Chawla. Deep learning for anomaly detection: A survey. *CoRR*, abs/1901.03407, 2019.

[3] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2015.

[4] Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks, 2015.

[5] C. M. K. W. Ghazal Fazelnia, Mark Ibrahim and J. Paisley. Mixed membership recurrent neural networks. 2018.

[6] N. Laptev, J. Yosinski, L. E. Li, and S. Smyl. Time-series extreme event forecasting with neural networks at uber. 2017.

[7] A. P. M. Ryan and C. Mahoney. Real-time anomaly detection for advanced manufacturing: Improving on twitter's state of the art. 2019.

[8] R. J. H. Priyanga Dilini Talagala and K. Smith-Miles. Anomaly detection in high dimensional data. 2019.

[9] O. Vallis, J. Hochenbaum, and A. Kejariwal. A novel technique for long-term anomaly detection in the cloud. In *HotCloud*, 2014.

[10] L. Zhu and N. Laptev. Deep and confident prediction for time series at uber. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, Nov 2017.