End-to-End Solution For IoT devices Predictive Maintenance and Management

Chuangxin Xia, Risheng Wang, Yifan Li
[cxa25, rishengw, yla570]@sfu.ca
http://www.devxia.com/crouching_tigers

1. Motivation and Background

IoT, Internet of Things, is just more than just a fad. According to data from Google Trends[1], the first interest spike on Internet of Things happened back in early 2014. It has pretty much been growing non-stop ever since. More interestingly, the world had about 12.5 billion IoT devices in 2012, it grew to 25 billions in 2015. It's projected that there will be over 50 billion by 2020[2]. The rise of Internet of Things is related to everyone and it's for the greater good. The world is already surrounded by IoT devices in diverse fields.

With that amount of growth, it definitely comes with some problems. Among with reliability, privacy, and other concerns, maintainability and longevity of the devices has always be the top issues. Manufacturer wants to know potential issues with a certain device before it comes out on the market. Service department wants to plan their labor, resources more efficiently and have an easy to to access device status in real time.

Due to the nature of IoT devices, manufacturers have been designing and implementing their own management and maintenance solution. However, one obvious drawback of those exclusive solutions is the overhead and switching cost. Companies using multiple brands and types of products need to pay those overhead on an ongoing basis, whether it's license fees or cost of training for their technicians to use different systems. Not to mention the struggle to keep track of numerous systems simultaneously. All of the existing solutions only do preventive maintenance instead of making an actual accurate prediction on the future states of the devices. Preventive maintenance is as simple as setting a timer based on historical data. In most scenarios, it is both inefficient and ineffective.

2. Problem Statement

We are trying to tackle the following questions and challenges:

- How to make accurate prediction regarding future state of the device?
- How to detect anomalous behavior that may lead to machine failures based on our prediction?
- How can we easily extend our solution for different devices with various measurements?

Our ultimate goal is to build a non-exclusive end-to-end IoT device predictive maintenance and management solution, from on-field devices all the way to a web console. The solution should be easy-to-use, flexible for all devices, and scalable to accommodate future growth.

---

[1] "Google Trends search on the " internet of things " and " IoT " keywords ...." https://www.researchgate.net/figure/Google-Trends-search-on-the-internet-of-things-and-IoT-keywords_fig1_319311693. Accessed 14 Apr. 2019.

[2] "An accelerating IoT adoption rate will benefit the digital economy ...." 29 Nov. 2016, https://delta2020.com/blog/150-an-accelerating-iot-adoption-rate-will-benefit-the-digital-economy. Accessed 14 Apr. 2019.

3. Data Science Pipeline

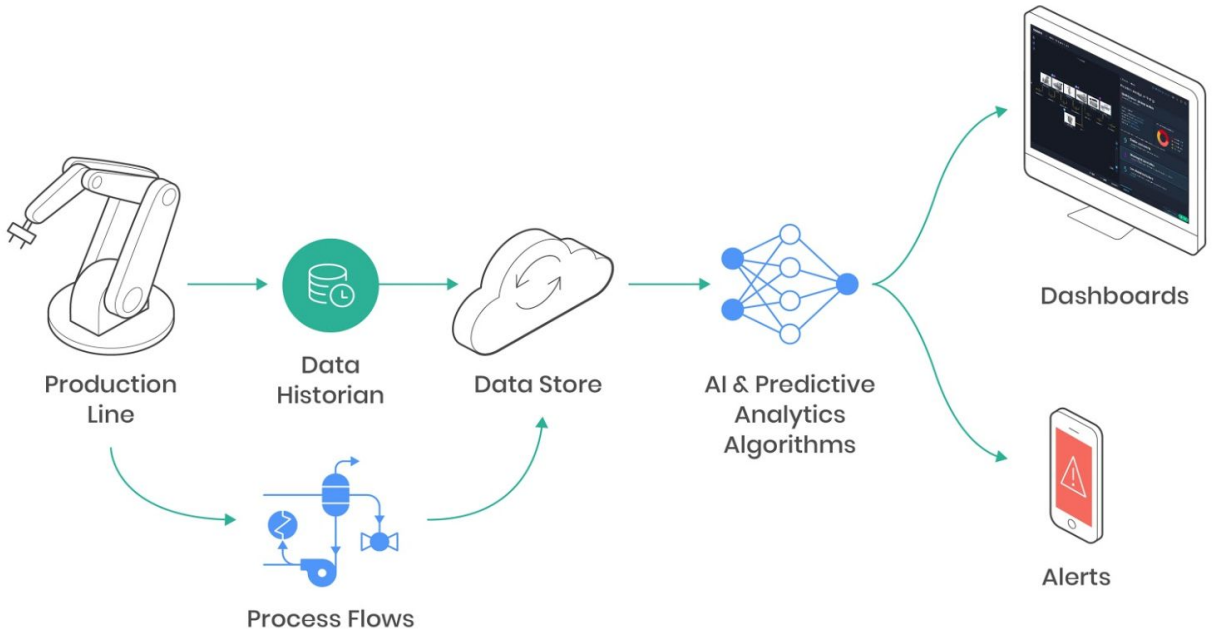


Fig. 1 Data Science Pipeline[3]

Generally speaking, data science pipeline refers to a sequence of processing and analysis steps applied to data. In our project, we managed to simulate the entire process of data collecting, data integration, data preprocessing and manipulation, data analysis and result display, and divide them into four main parts: Data Collection, ETL/EDA, Data Analysis, Streaming Server and Web Console. Figure 1 explains how our pipeline looks like for this project.

3.1 Data Collection:

In our project, we used data collected from a sensor attached to an automated and manufacturing device called Selective Compliance Assembly Robot Arm (SCARA) which captures position and calibration at each time stamp. The data collected represents some tasks that the SCARA arm is performing – for example, tightening screws – and then returns to a ready position (Fig. 2).

[3] "Predictive Maintenance in IIoT Projects, Moving Away From Preventive ...." 31 Jan. 2019, https://internetofthingswiki.com/moving-from-preventive-to-predictive-maintenance-in-iiot-projects/1348/. Accessed 14 Apr. 2019.

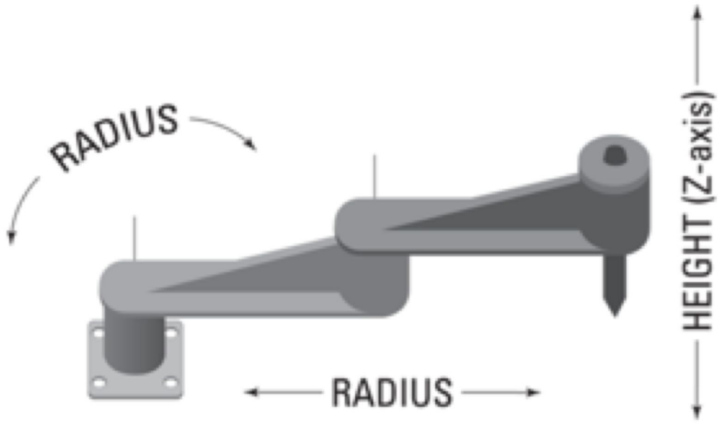


Fig. 2 SCARA Arm

In detail, our original data includes 2013 .dat files in total and the format of data is xml. Each row of data represents one feature of the status of the arm at one time stamp. Figure 3 shows an example of the raw data in xml format.

```
<?xml version="1.0"?>
<ArrayOfHistoricalTextData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
http://www.w3.org/2001/XMLSchema">
  <HistoricalTextData>
    <TagName>::[scararobot]Ax_J1.ActualPosition</TagName>
    <Status>0</Status>
    <TagValue>59.354080</TagValue>
    <TimeStamp>2016-07-22T10:02:54.0872165-04:00</TimeStamp>
  </HistoricalTextData>
  <HistoricalTextData>
    <TagName>::[scararobot]Ax_J1.PositionCommand</TagName>
    <Status>0</Status>
    <TagValue>0.000000</TagValue>
    <TimeStamp>2016-07-22T10:02:54.0872165-04:00</TimeStamp>
  </HistoricalTextData>
  <HistoricalTextData>
    <TagName>::[scararobot]Ax_J1.PositionError</TagName>
    <Status>0</Status>
    <TagValue>0.367542</TagValue>
    <TimeStamp>2016-07-22T10:02:54.0872165-04:00</TimeStamp>
  </HistoricalTextData>
```

Fig. 3 Original Data in XML format

We made use of pyspark SQL to accomplish the task of data integration on what we collected. Figure 4 displays a part of the cleaned data in CSV format. In total there are 28 columns standing for different features of the arm, including actual positions, position commands and position errors, etc.. Furthermore, the data contains more than 82,000 timestamps, which in total takes up to 260 megabytes' space on the disk.

TimeStamp	::[scararobot]Ax_J1.ActualPosition	::[scararobot]Ax_J1.PositionCommand	::[scararobot]Ax_J1.PositionError
2016-07-22T07:21:38.917-07:00	60.829575	0	0.374652
2016-07-22T07:03:55.513-07:00	46.359734	0	0.343566
2016-07-22T07:33:51.456-07:00	56.809727	0	0.37224
2016-07-22T07:17:56.835-07:00	2.763756	0	-0.34155
2016-07-22T07:29:32.501-07:00	30.450562	0	0.32391
2016-07-22T07:11:30.501-07:00	73.516205	0	0.459702
2016-07-22T07:15:07.467-07:00	86.591927	0	0.464994
2016-07-22T07:27:58.231-07:00	-6.698466	0	0.344448
2016-07-22T07:22:38.402-07:00	-3.876948	0	0.306594
2016-07-22T07:30:59.769-07:00	45.416771	0	-0.345384
2016-07-22T07:08:24.414-07:00	63.152836	0	0.37863

Fig. 4 Integrated Data as a CSV table

3.2 ETL/EDA

3.2.1 ETL (PySpark SQL)

Our original data, as shown in Figure 3, is in XML format and is hard to interpret. Therefore, we want to transform into a dataframe to get better insight. Since there are 2013 files in total, we want to do a bulk conversion and PySpark came to our sight for its capability to perform the transformation on all the files in one time.

We utilized a spark xml package to accomplish the data loading. Once the files are loaded, we got a narrow and long table that had 28 metrics associated with each time stamp. In general, it indicates that first 28 rows was for timestamp 1, then second 28 rows for timestamp 2 and so on. Apparently, what we need is a data structure in which each row is a timestamp and each column is a metric.

We used a group function called '**pivot**' from PySpark SQL which pivoted our data on the column 'TagName', and transformed into a new dataframe which contains 28 feature columns and 1 timestamp column for each data point, as shown in Figure 4.

3.2.2 EDA (Pandas)

Upon finishing data ETL, we obtained our initial view of the data. Figure 5 shows some straightforward visualized analysis regarding the arm, and from the picture we would get several visualization and find obviously erroneous data and outliers.

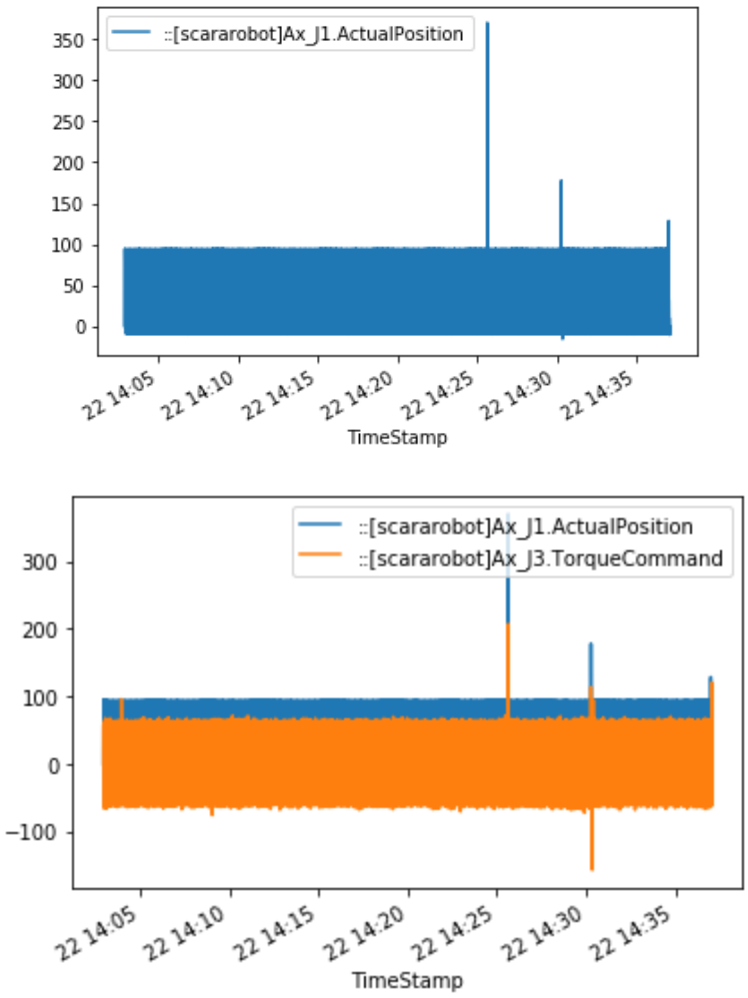


Fig. 5 Data Plotting

Our journey does not stop here. By finishing this project, we would like to find a solution for the industry so that they could predict abnormal behaviors of their devices ahead of time. Furthermore, it should not be only limited to certain features of the device, but the comprehensive status we want to predict.

To achieve this goal, we used pandas library for EDA work. There were more choices for us. Alternatively, we also tried performing feature selection by using sklearn SelectKBest and Chi2 modules. Unfortunately, the original data contains negative values that were not compatible with the library, and once converted to their absolute value, we were faced with an issue of information loss, which could lead to unpredicted or meaningless results.

We digged deeper into the true meaning of the data. For every SCARA arm, when there occurred an outlier, it occurred across all features in the timestamp. This indicates that when something irregular happened to the device, it affected the entire measurements. As a result, we attempted to aggregate the values in each column to produce a sum value for prediction,

and to act as a single threshold of acceptable normal behavior. Moreover, we found that some features had nothing to do with the device status. We dropped those features, including position commands, torque feedbacks and scan time, to further streamline our pipeline.

3.3 Data Analysis

Our data came from a SCARA arm recoding the positions of the arm so it does not contain extra meanings besides that. Therefore, our data analysis part was uniquely focused on exploring the trends of data, and predicting future status of the arm.

To analyze our data, we mainly used three tools: **pandas**, **tensorflow** machine learning modules and **sklearn** library.

Pandas is useful in exploring basic distributions and internal structures in the data, while tensorflow took most part in training, testing and predicting. We created a LSTM RNN by tensorflow network constructor, and trained the model with over 80,000 lines of data. Lastly, we use sklearn embedded algorithms to approach anomaly detection. The details about the algorithms will be explained in the upcoming Methodology section.

3.4 Model building and model serving

We store our cleaned data on **Google Cloud**. **Google co-lab** is the platform we choose for model building. We perform our model building using Tensorflow and sklearn. Once we built our model, we deploy it onto **Google Cloud Machine Learning Engine(GC MLE).** After serving our model live, we achieve restful API endpoint so that we can get prediction about machine states in real time.

3.5 Web console / dashboard

The dashboard is where all of the previous work comes together. It communicates with our model serving on **Google Cloud Machine Learning Engine** and our **Node.js** IoT device status server running on **Amazon Web Service**. Users can start or pause live prediction on machine status for the specified future via interactive user experience. Our Node.js server has the purpose of simulating real world devices, their heartbeat and uptime.

4. Methodology

We applied our tensorflow network on Colaboratory, a free Jupyter environment supported by Google. Besides a LSTM RNN prediction model, we also achieved anomaly detection by applying Sklearn embedded algorithms.

4.1 Google Colab

Google Colab is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. We chose to use Google Colab instead of mere Jupyter notebook because tensorflow is already pre-installed in the Colaboratory. Moreover, since it runs on cloud, we could use GPU device provided by Google Cloud freely to train our model, and share the files with teammates in avoid of any issues that might occur when we want to integrate our code and data.

4.2 Tensorflow Deep Learning Module

Inspired by works done by Justin Brandenberg on his Streaming Predictive Maintenance[4] and XiaoFeng Xie and his teammates on IoT Data Analytics Using Deep Learning[5], we came up with the idea to use Tensorflow RNN module to accomplish our task on data prediction.

Compared to regression models and classification models, RNN is good at sequence to sequence predicting, especially when it is hard to find any standard structure inside the data. Our data is from SCARA arm, which records its position information, and this information contains very little standard structure in it. RNN is proved to meet our requirement in this project.

Instead of a single ReLU activation node applied in original RNN module, our team adopted LSTM cell to improve the performance of the network. We used traditional conventional LSTM structure, as explained in the Figure 6, and adopted 2 layers, 100 hidden nodes for each layer, and Tanh algorithm as activation.

---

[4] "Streaming Predictive Maintenance for IoT using TensorFlow - Part 1 ...." 26 Feb. 2018, https://mapr.com/blog/streaming-predictive-maintenance-for-iot-using-tensorflow-part-1/. Accessed 14 Apr. 2019.

[5] "IoT Data Analytics Using Deep Learning." 13 Aug. 2017, https://arxiv.org/abs/1708.03854. Accessed 14 Apr. 2019.

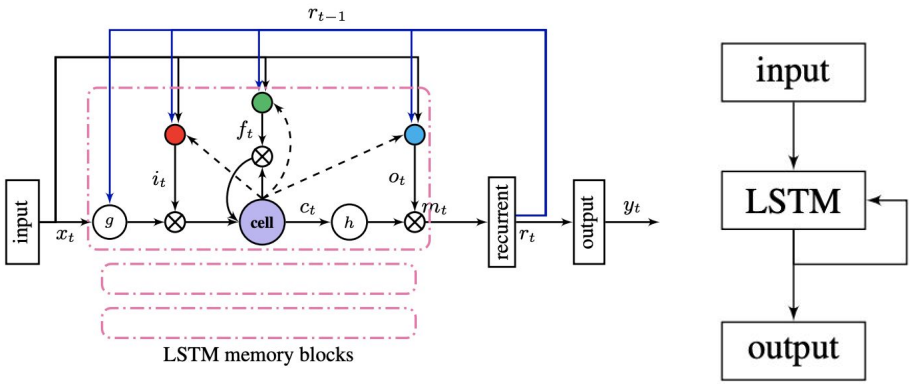


Fig. 6 LSTM RNN architecture[6]

To train the model, we used 2,000 epochs, and a learning rate of 0.005. We used the sum of square of difference between predicted value and actual value to compute loss, and the process of training is to minimize this loss. The equation is as following:

$$Loss = \sum \left(V_{pred} - V_{actual} \right)^2$$

In addition, to avoid over-fitting, we added a bias value to the outputs of the model as a regularizer.

4.3 Sklearn Anomaly Detection Library

4.3.1 OneClassSVM

With One-class SVM model, the support vector model is trained on data that has only one class, which is known as the “normal” class. It infers the properties of normal cases and from these properties can predict examples that are unlike the normal examples. The main difference between the OneClass SVM and the original SVM is that in one-class SVM the only given information is the normal samples of the same single class, whereas in the original SVM information on both normal samples and outlier samples are given.

The one-class SVM estimates the boundary region that comprises most of the training samples. If a new test sample falls within this boundary, it will be classified as of normal class, otherwise it will be recognised as an outlier. Suppose the training set contains m samples, the one-class SVM will make prediction basing on the assumption that the origin in the transformed feature space belongs to the negative or outlier class. The training stage consists of following steps: firstly, projecting the training samples to a higher dimensional feature space, and then separating most of the samples from the origin by a maximum-margin hyperplane which is as far

[6] "Long Short-Term Memory Recurrent Neural ... - Research - Google." https://research.google.com/pubs/archive/43905.pdf. Accessed 14 Apr. 2019.

away from the origin as possible. In order to determine the maximum-margin hyperplane, it needs to deduce its normal vector w and a threshold ρ by solving the following optimization problem:

$$min\ \frac{1}{2}|w|^2 + \frac{1}{vm}\sum_{i=1}^{m}\xi_i - \rho$$

Here, it introduces the "slack variable" ξ_i to avoid the overfitting. Parameter v is a unique parameter for OneClass SVM which controls the upper bound ratio of outliers among all training data.

4.3.2 IsolationForest Tree

IsolationForest or iTree is a model-based method that explicitly isolates anomalies rather than profiles normal instances. The method takes advantages of anomalies that are few and different which make them more susceptible to be isolated than normal points. This method builds a binary search tree (BST) like structure and anomalies are those samples which have short average path lengths on the iTree. The task of anomaly detection is to provide a ranking that reflects the degree of anomaly. The anomaly score s of an instance x is:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

Where $E(h(x))$ is the average length of $h(x)$ from a collection of isolation trees and h(x) of a point x is measured by the number of edges x traverses an iTree from the root node until the traversal is terminated at an external node. c(n) is the average path length of unsuccessful search in Binary Search Tree and n is the number of external nodes.

5. Evaluation

We want to evaluate our project in three aspects: how is the performance of our model, how well is the prediction and the value of our product.

5.1 Training Loss

We used the sum of square of difference between predicted value and actual value to compute training loss. Figure 8 are plots of training loss with one layer RNN and 10,000 epochs and two layer RNN and 2000 epochs respectively. The value of loss was recorded every 10 epochs.

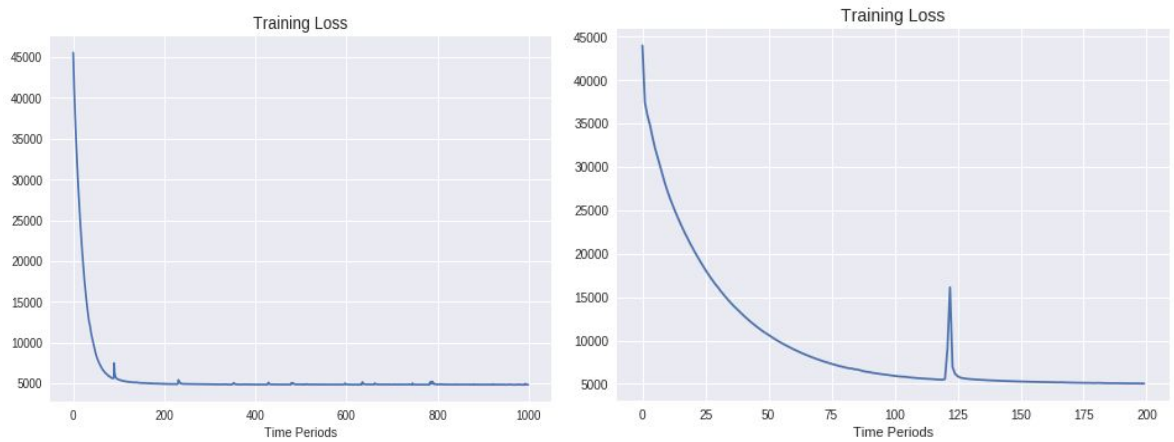


Fig. 7 Training Loss with 1-layer and 10,000 epochs (left), and 2-layer and 2,000 epochs (right)

From the plots we could see that training loss with 1-layer of RNN (right picture) drops a little faster than the one with 2-layer of RNN (left picture).

5.2 Prediction Test

Figure 9 is the Prediction vs. Actual plot we produced with our test data.

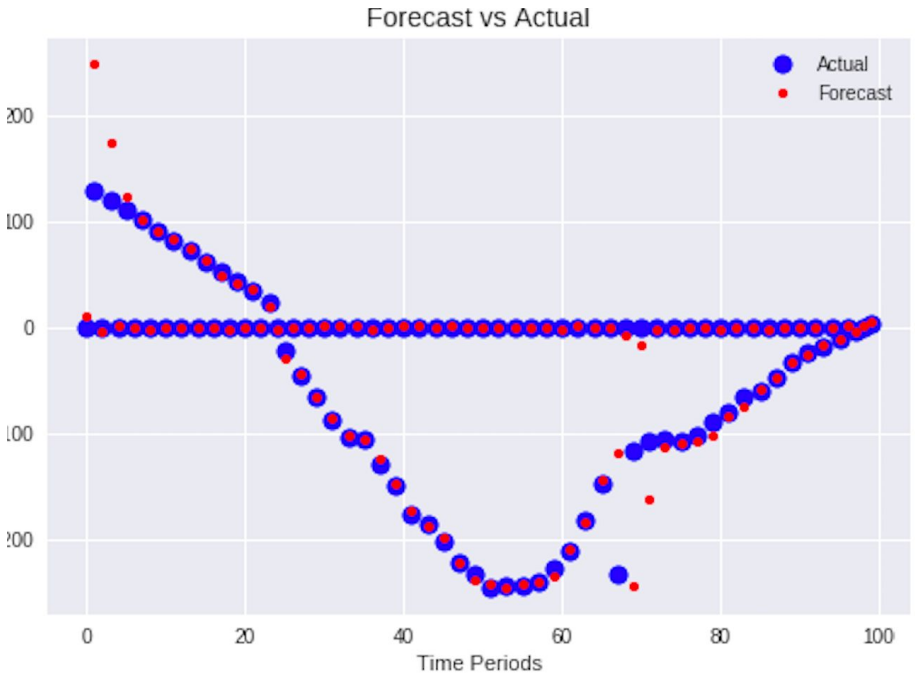


Fig. 8 Prediction Test

It is obvious that our model had done a great work on predicting future status of the device. Figure 10 shows a lined plot with results of anomaly detection.

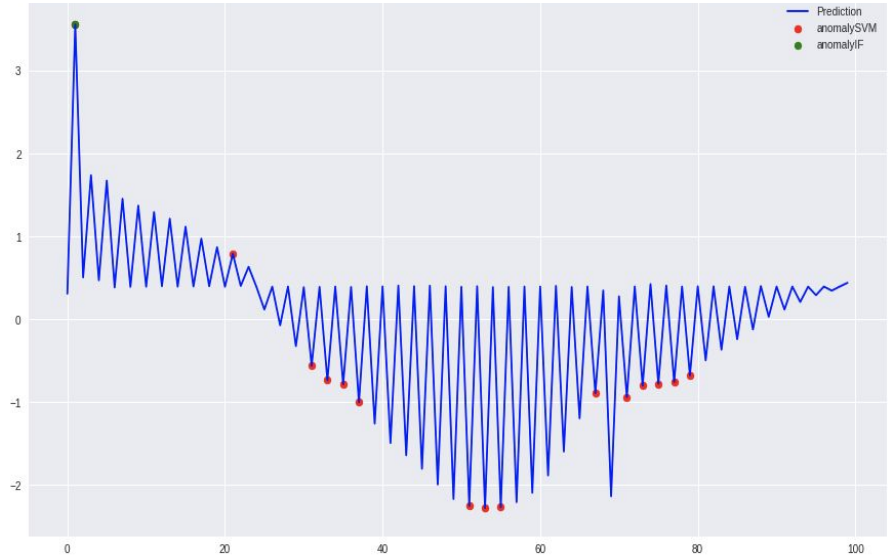


Fig. 9 Lined Plot with Anomaly Detection

Where red points stand for the results from One-Class SVM, and green one the Isolation Tree.

5.3 Scalability and flexibility

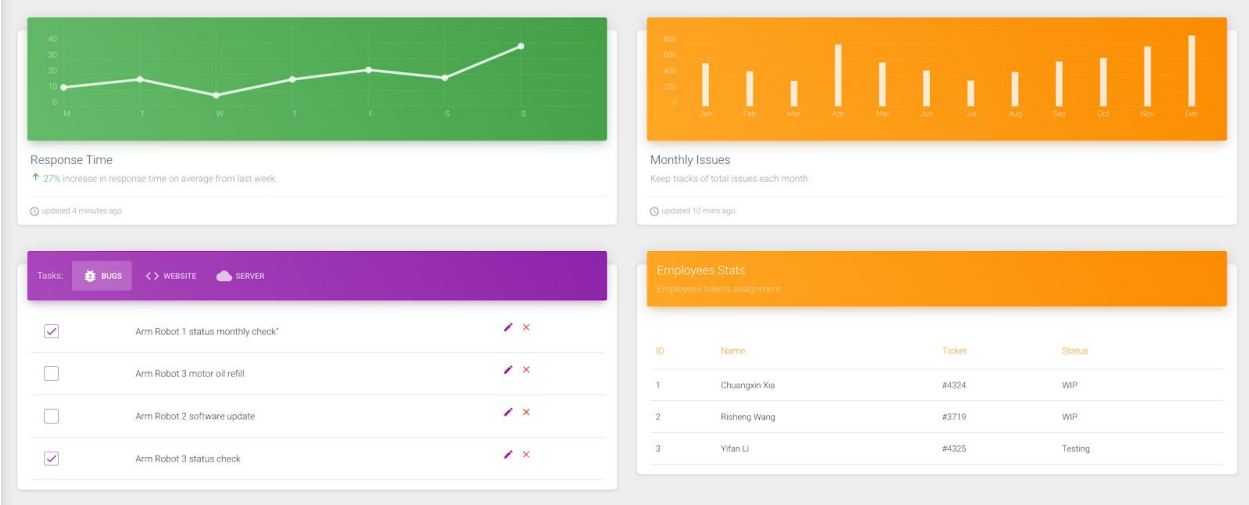
All the technologies that we handpicked, ranging from ETL, EDA part using pySpark, to LSTM RNN in Tensorflow, even to anomaly detection using Sklearn, are flexible enough and can be reapplied easily onto different IoT devices with a totally distinct set of features. We designed the whole pipeline to be future-proof. Using restful API for communicating between the web console and servers allows the project to be adaptive for future growth as well.

6. Data Product

Our final product is the user-friendly dashboard for checking device future states and device heartbeat implemented using React.js. It connects with our model served on Google Cloud Machine Learning Engine and Node server on Amazon Web Service EC2. The idea behind our dashboard is for maintenance crews to have an all-in-one place to anticipate possible future machine failure and real time status. Visualization was implemented using Chartist.js, and the layout is responsive, which means users can access it across all platform and devices.
The frontend is hosted at http://www.devxia.com/crouching_tigers.

Imaging yourself to be a part of the maintenance crew, the dashboard is the only place you need to check during your work. Clicking on the green “START LIVE” button, the dashboard will start live streaming from Google Cloud and AWS. If you see something wrong on the visualization or the machine status, click on the red “PAUSE LIVE” button. The chart and heartbeat check will stop updating and will show last updated time for future reference.

At this point, technicians can choose to issue a ticket and get right to work. The dashboard is also connected with Google Map APIs to users need to enable location based tracking.



7. Lessons Learned and Future Work

From working on this project, we learned that real world datasets are usually messy and unstructured. Approaches for doing ETL and EDA work should always keep flexibility and scalability as the top priority. We also learned that making a fully-automated pipeline is tough since not all technologies provide easy-to-access API for communication.
Here's the list of things we would like to work on in the future:

- adding a database for persisting a ticket/issue tracking system.
- automating all the stages in the data science pipeline.
- implementing finer parameter tuning with a more diverse dataset.
- experimenting more neural network structures for a more robust model.

8. Summary

In this project, we want to achieve an end-to-end solution for IoT device predictive maintenance and management. We performed ETL and EDA using pySpark, incorporated feature selection and anomaly detection on top of prediction neural network model trained using LSTM RNN. Those layers ensure our prediction to be precise. Our dashboard console communicate with live Node.js server and live model served on Google Cloud Machine Learning Engine while providing an interactive user experience and easy-to-interpret data visualization. The whole pipeline was built with flexibility and scalability in mind.