Anomaly Detection in Machine Data:

Designing the Data Acquisition and Preparation Pipelines

February 1, 2018

Team Members:

Garrett Cheung

Michael Galarnyk

Jared Goldsmith

Jillian Jarrett

Orysya Stus

Advisors:

Yoav Freund (UCSD)

Chad Holcomb (Solar Turbines)

Raw Data Sources

Machine data. This data comes from the turbine packages themselves. Solar Turbines' packages are units comprised of compressors, combustors, turbines and application specific components. The machine data is primarily sensor readings. More specifically, they are primarily temperature and pressure readings, but also include vibration/displacement data, speeds, programmable logic controller values, etc. Solar currently stores 1 hour data (used to generate alerts) and 10 minute data. A single package has between 200-600 tags (columns), depending on model, control system, application type, etc.

While there are ~1,900 connected packages worldwide, we are exploring two engine models across a 2 year period (12/2015- 12/2017). Model 1 is comprised of 33 packages and Model 2 is comprised of 39 packages. For these packages, we're taking a subset of features that exist across the entire fleets (only tags that exist for all 33 Model 1 packages or all 39 Model 2 packages). Of those, we are reducing our dataset to include package basics and features used to generate alerts. For Model 1, that reduces our dataset to 146 features, and 77 features for Model 2. Specifically, we are looking at 1hr and 10 minute resolution data. Column names had to be anonymized and a data dictionary has been generated and distributed.

Failure data. We are in the process of requesting data from field service repairs and engine overhauls. Such information has potential to be useful for generating labels and differentiating normal from anomalous operating behavior. Due to the proprietary nature of this data, it had to be requested and anonymized by employees with higher clearances than us. As such, the size, format, and composition of this data has yet to be determined.

Table1. Raw data sources

Dataset Name	Source	Destination	Acquisition Notebooks, Code, Documents	Data Size & Format	Other Notes
1hr_machine_data	Solar Turbines servers	AWS S3	Available on our github: https://github.com/mGalarnyk/AnomalyDetectionMachineData	~1gb (421,204 rows) in 72 csv files	Anonymized, confidential (data cannot be made publicly available). Data dictionary available
10min_machine_data	Solar Turbines servers	AWS S3	Available on our github: https://github.com/mGalarnyk/AnomalyDetectionMachineData	~6gb (3,107,256 rows) in 72 csv files	Anonymized, confidential (data cannot be made publicly available). Data dictionary available.
failure_data	Solar Turbines servers	AWS S3	None yet	TBD	Not yet acquired. Had to be requested and anonymized by Solar Turbines employees with higher clearances

Data Exploration, Cleaning, Wrangling and Engineering

Understanding data format. A data dictionary was generated to exhaustively document data structure, meaning, and format. In the datasets, the column headers define a tag, i.e. SUBSYSTEM_C_TAGTYPECOUNTER, where the SUBSYSTEM refers to the subsystem is usually associated with, C denoted whether or not a tag is calculated versus measured (C means calculated, as is an optional argument), TAGTYPE refers to the type of measurement being recorded, and COUNTER is an incremental counter to differentiate tags with the same subsystem and tagtype. This data format will be referenced throughout this project.

Data quality and cleaning. Model1 and model2 hourly data was initially loaded into a Jupyter notebook to determine the data quality, the shape of the data and null counts were noted and feature plots generated. At this moment, we define complete records as timestamps per package which are not null for any single tag. Model1 has 163,544 total records and 145 tags with 46.5% (76,833 timestamps) being complete records while model2 has 257,732 total records and 77 tags with 64.6% (166,500 timestamps) being complete records. Some tags which are largely null include: sc_c_pct_e1 (51.2% null for model1 and 35.4% null for model2) and sc_pct1 (19.0% null for model1 and 22.1% null for model2). With the guidance of our domain experts, we will might need to drop certain tags using a null value threshold, unless the tag in question holds very high important in alerts (ie. a tag might typically be null but if a value is detected than an anomaly is present).

Furthermore, feature plots were generated for each tag as a line graph showing the trend of the certain tag over time, with all the packages for the model considered. These feature plots are useful in understanding which tags has more variance in trends and which tags are more stable (low variance) as well as visually seeing spikes/dips in trends signifying potential anomalies. An example feature plot is shown:

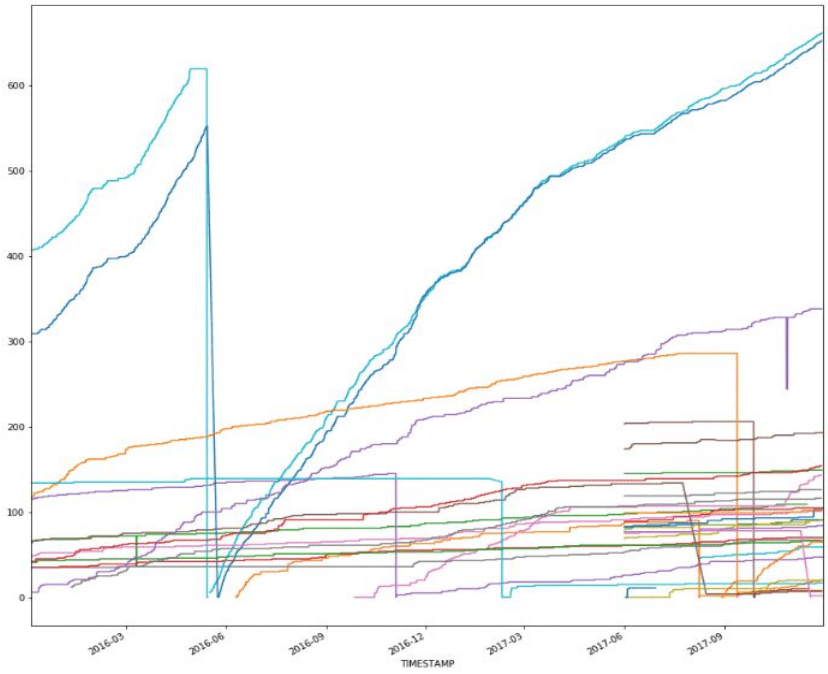


Figure 1. A feature plot of SUM_ENG_ST (engine starts) is shown from model1. Each line represents a distinct package in the model.

Feature correlations. Highly correlated features can be helpful in identifying anomalies. For example, if a particular tag for a package begins to fail or vary from its normal trend we can assume that a correlated tag will follow suit. Correlation heatmaps were generated for model1 and model2. The top 10 correlating pairs were mostly comprised of temperature tags, which is expected considered the temperatures should rise simultaneously. It will be interesting to examine the correlations from different subsystems as we understand how an engine's health is determined.

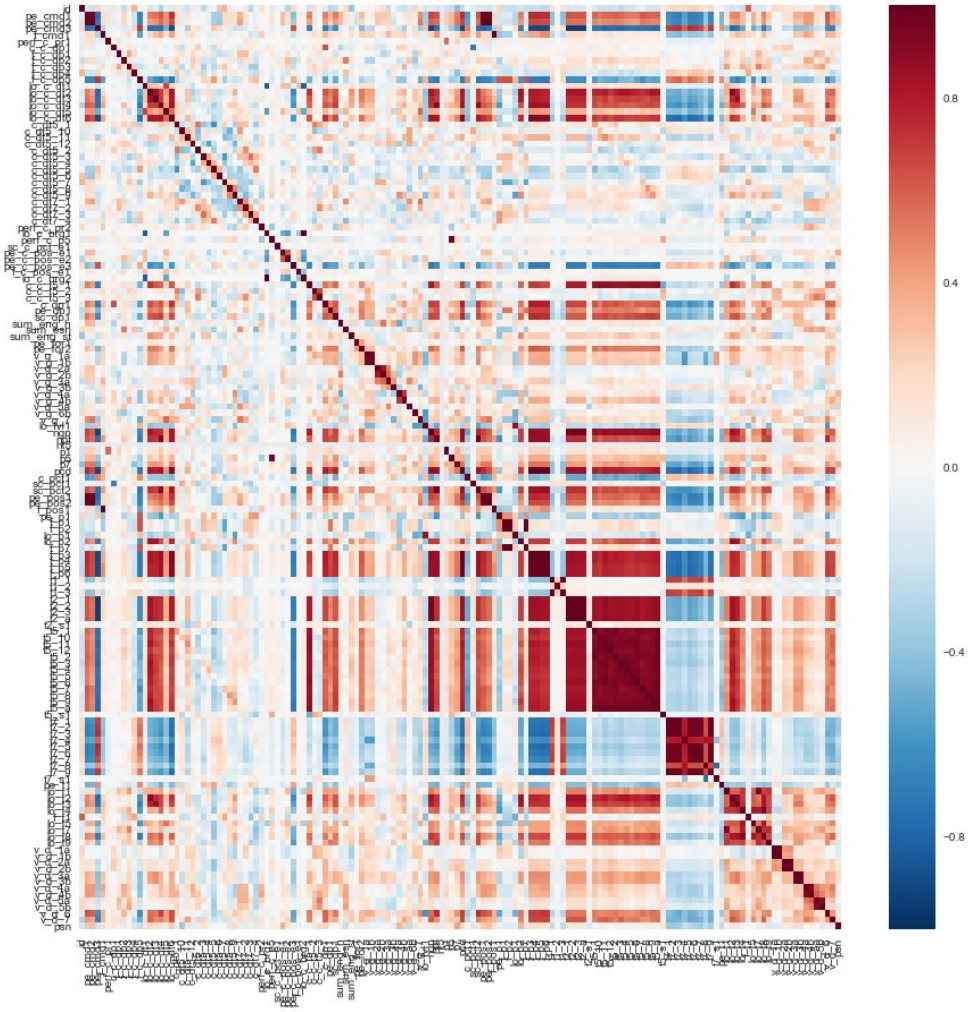


Figure 2. Correlation Heatmap Generated for Model1.

PCA results. Null data was dropped and an initial PCA computed on Model1. It was found that the first eigenvector explained a majority of the variance. Further examination will be necessary to understand if this is true or if this is an artifact of the normalization completed when moving the data off of the Solar network. All the tags were normalized to be between 0 and 1 (excluding outliers), with the exception of one tag (engine starts). We will recompute the PCA without engine starts being considered, and determine if the resulting PCA will change.

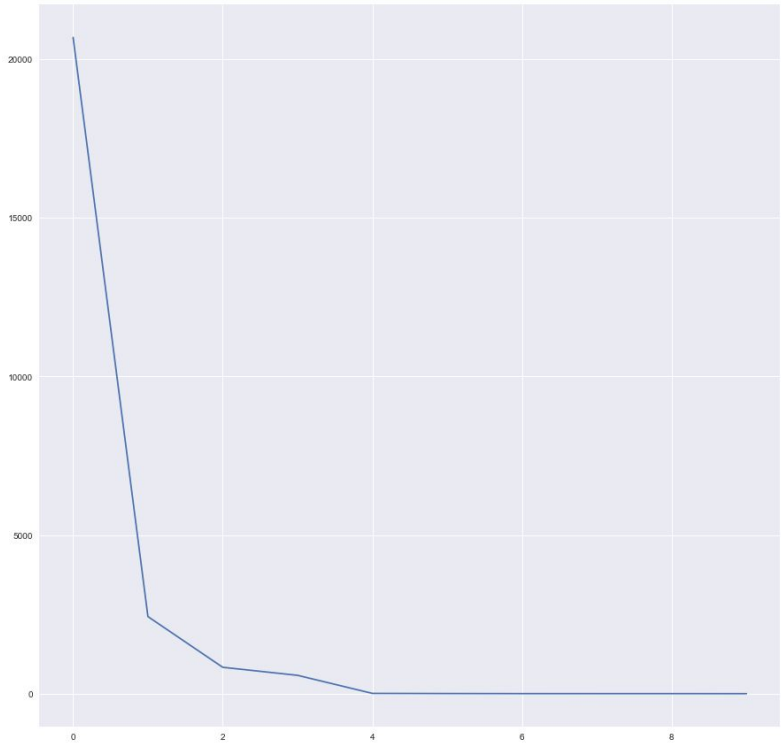


Figure 3. PCA Results for Model1. The first eigenvector explains a majority of the variance.

Data Preprocessing Approach

Data Pulling. Machine data can be pulled from Solar Turbines databases using an internal python library. This module allows users to retrieve machine data values for a specified date range in the form of a pandas dataframe. Both 1hr and 10 minute resolution was pulled. Some of the columns are composites of other features. These columns are generated and stored for the 1 hour dataset, but not in the 10 minute. To compensate, these columns were generated for the 10 minute dataset using their defined formulas.

Data Filtering. The machine data was then filtered by a series of conditions, per our domain experts' suggestions. First, to reduce variability within our very complex dataset, we are considering only data where the package is running close to its capacity (on load conditions). Second, we considered that some of the packages run only on gas fuel, while other can be run on liquid or gas. Again, for consistency, data points where the machines are running on liquid fuel were removed. Finally, because not all machines contain the exact same columns (due to different configurations and use cases), columns which are not common across all packages are removed as well. This also applies to the composite columns mentioned above.

Data Normalization. Solar Turbines mandated we normalize our data by factors that would remain unknown to any non-Solar employees. Many of our features are used to generate alerts: notifications to customers and Solar employees that specific parameters had crossed some sort of limit. These limits are numbers defined by Solar Turbines' engineers and are values under which packages are deemed healthy. However, crossing a global limit does not necessarily imply that the package was failing or malfunctions: often alerts are noted and no action is taken. Not all features are used to generate alerts. Those that do, however, were normalized by their global high limit values.

For columns without global limits, domain experts were consulted and data was normalized by factors suggested by them. Two features were not normalized: engine starts (the number of time an engine has started) and timestamps. While the normalization was a requirement of Solar Turbines, it offers some consistency and rules of thumb to our team members. After normalization, for most columns, expected values are generally between 0 and 1.

Data Anonymizing. Column names were aliased based on their subsystem and tagtype, as described in the Understanding Data Format section above. A data dictionary was also created for basic descriptions of each column.

Note: Due to the proprietary nature of our data, our 'raw' dataset is not available. We are using what we were able to move off the Solar network as our 'raw' data set, even though it has been extensively processed, as detailed above. As such, for our table of 'processed' data attributes, see Table1 above.

Approach for storing processed and/or integrated data

The anonymized data from Solar is and will continue to be stored in a PostgreSQL database. The current volume of data does not require a distributed storage solution, and, at this stage, ease of use is paramount. The 1 hour data is currently hosted on an AWS EC2 t2 large instance. The 10 minute data will soon be hosted on the instance as well, after testing query performance on the currently hosted data.

Data transformations and other engineered features will be cached in a key-value store that allows schema-less inserts. The complete set of features and transformations we'll be working with is currently unknown and expected to be constantly changing. Avoiding schema definitions for every new transformation or combination of transformations will save time. In addition, having an indexing strategy for transformations and derived attributes can enable faster team-exploration of the data and save on repeated computations. Because of the high dimensional nature of the anonymized data set, the derived data has the potential to be many times the volume. Moving away from relational databases provides scalability to handle working with that volume as well as flexibility in the way we store data structures.

Processed Data Location: PostgreSQL, AWS EC2
Import Code: /src/storage/postgres.py (subject to change)
Input: 1hr_machine_data
Destination Tables:
sensor_readings_model1_1hr
sensor_readings_model2_1hr
Input: 10min_machine_data
Destination Tables:
sensor_readings_model1_10min
sensor_readings_model2_10min

Approach for Feature Engineering and Data Modeling

While we have yet to dive into the realm of feature engineering and data modeling, we are beginning to explore potential avenues. Possible feature sets are included in the table below. None have been executed or stored at this time.

Table2. Feature sets

Feature	**Input Datasets (Dependencies)**	**Destination**	**Related Notebooks, Code, Documents**	**Provisional Data Size**	**Other Notes**
Team-made composite features	1hr and/or 10min datasets	AWS (TBD)	None yet	TBD	Features derived from combinations of other features. Not yet constructed.
Independent variables	1hr and/or 10min datasets	AWS (S3)	Available on our github: https://github.com/mGalarnyk/AnomalyDetectionMachineData, Also detailed in our data dictionary	TBD	Several features that can be used as general indicators of package health and suggested as independent variables for exploration of other features have been indentified in our data dictionary. These are part of our original datasets
Subsystems	1hr and/or 10min datasets	AWS (TBD)	Suggestions are detailed in our data dictionary	TBD	The 1hr and 10minute machine data can be grouped by subsystem. Chad and Jillian mapped out each feature to a subsystem, suggested for grouping, exploring, analyzing sets of features from our raw data
Overlapping tags	1hr and/or 10min datasets	AWS (TBD)	None yet	TBD	Some tags can be used to validate or identify the health of other sensors. Specific groupings have yet to be established.
Kernel transforms	1hr and/or 10min datasets, all other feature sets	AWS (TBD)	None yet	TBD	Not yet constructed
Anomaly labels	1hr and/or 10min datasets, all other feature sets	AWS (TBD)	None yet	TBD	Not yet constructed

Approach for Data Access

See data pipeline schematic below for context. Our data will be accessible as follows:

Anonymized Data

Import Interface

import_directory(directory, table, force_refresh)

import_csv(path, table, force_refresh)

Query Builder

Filtering

.not_null(column_name)

.psn(serial_number)

.model(model_number)

.frequency(f) # ‘1hr’ or ‘10min’

.min_time(timestamp)

.max_time(timestamp)

.remove_duplicates(columns)

Projection

.select(feature_list)

.select_subsystem(name)

Data Transformations

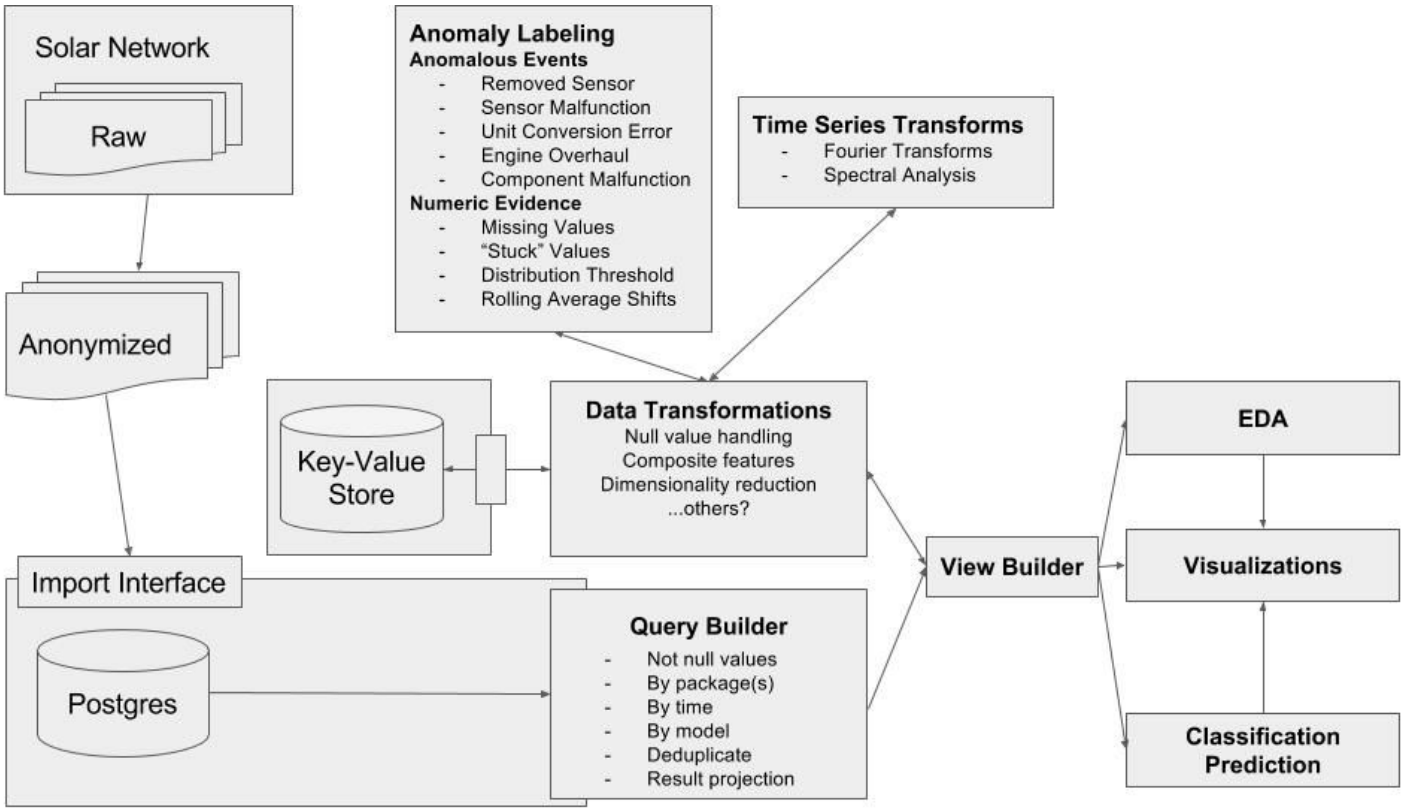
.insert(query_params, [(transformation_name, transformation_params)], results)

.fetch(query_params, [(transformation_name, transformation_params)])

Programmatic access enables interfaces that abstract away storage details. In the case of data transformations, we’re still assessing storage needs. Coding against the abstractions proposed above allows the underlying storage to change as needed for scalability.

Data Pipeline

The overview of our pipeline below highlights major components and general data flow.



Set up for our data environment

One of the most important parts of this project is the ability to collaborate with fellow team members and advisors. In result, we decided to utilize Google Colaboratory to make a live notebook. While this tool allows for collaboration, it has a drawback which our team is addressing: after users are idle on the notebook for 90 minutes, the virtual machine used to host the notebook is reclaimed. Consequently, packages need to be reinstalled every time when returning to the notebook. This isn't a major drawback as most commonly used libraries are preinstalled on the virtual machine. The rest can be installed in the first cell of the notebook. A more problematic drawback is that this makes it more difficult to work with flat files. Fortunately, the PostgreSQL database on AWS EC2 should allow for an efficient loading of data. An unexpected benefit of using Colaboratory is that there is no need to set up python environments and or worrying about different versions of python and various libraries.

Additionally, our bookkeeper Michael has been working extensively with Kevin Coakley to set up our AWS account.

Team Members Contributions

- Garrett Cheung (Data Engineer)
    - Preprocessed Solar Turbines data (pulling,filtering,normalizing,anonymizing)
    - Investigated data sizes for raw and published data
- Michael Galarnyk (Bookkeeper)
    - Correspondence and set up regarding our AWS account
    - Set up Google Colaboratory to facilitate team collaboration
- Jared Goldsmith (Record keeper)

    - Maintained our github repository
    - Preliminary EDA
    - Designed our data pipeline
- Jillian Jarrett (External Team Coordinator)
    - Wrote and published our data dictionary
    - Oversaw team correspondence and meeting scheduling with Yoav
- Orysya Stus (Internal Team Coordinator)
    - Constructed a demo notebook for data exploration with our advisor
    - Summarized EDA findings