TerumoBCT Anomaly Detection For Preventative Maintenance Analysis and Documentation

Sean Flood, Erik Pohle, Simon Koeten, Ruijiang Ma

University of Colorado at Boulder

May 2021

Appendix

0. Project Setup and Installation

This project was a massive undertaking for our team and posed numerous questions which we have not encountered prior to this project. First and foremost, we wanted to create an environment that would allow our team to run analytics in a uniform environment. We knew that in order to achieve this, we would need to have a standardized environment that would be usable on a variety of different machines and platforms. After some research, we determined that Docker was the solution. This proved to be more difficult than anticipated, and as such, we struggled for several weeks in setting up a docker environment, and after a surprising amount of tweaking, we were able to get a docker environment running on our Windows machines. The next step for us was to install the environment onto each team member's machine. This effort started well, but soon ran aground due to unanticipated problems. The biggest problem was a recent update for Apple computers which categorically dropped 32-bit support. As a result, Apple's Big Sur update ended up breaking the docker environment for all of our team members who used Mac, and, as a result, we had to pivot towards using an Anaconda virtual environment.

In order to set up and instantiate the project we have, the first step is to clone the GitHub repository. Afterward, Docker Desktop must be installed. Once Docker Desktop is installed, navigate to the location of the repository in the command prompt. Once there, type the command ‘make dev-start’ to start building the docker image. Building the Docker image takes a few minutes. Once complete, in the same command prompt, type the command ‘docker ps’. You should see two containers running.

```
CREATED          STATUS          PORTS                    NAMES
5 seconds ago    Up 4 seconds    127.0.0.1:32781->8888/tcp   CSCI-4308-TerumoBCT_jupyter_derekthomas
15 seconds ago   Up 14 seconds                            CSCI-4308-TerumoBCT_bash_derekthomas
```

On your machine the container ids and the names of the images and running containers will be different, i.e. they will have your username rather than derekthomas. In addition, the local ports will be different as well. That is expected. Now, in order to go to the Jupyter instance, look for the IP address and port that is associated with the container named 'CSCI-4308-TerumoBCT_jupyter_XXXXX'. The IP should be localhost, but the port will change. Go to that URL on the browser of your choice. Once there, Jupyter Lab should be waiting and ready to go! In order to stop the containers, you can use the command 'make dev-stop'. If you are done with the docker environment as a whole, you can delete all images from your computer using the command 'docker system prune'. At this juncture, the environment should be ready to go!

1. Recommendations for Future Development

Due to the size of the dataset required for this project, it is recommended that future development teams prepare a framework for processing big data, and this document contains recommendations on how to do so, and subsequently how to begin development. The primary change that is needed to produce quality results for this project is that a big data processing framework, such as Spark or Dask, that allows for reliable processing of the data is needed. While querying the SQL server is an easy task that any machine can do, a representative sample of the entire database is needed for meaningful analysis, and SQL is not natively equipped to perform such an operation (this can be done in SQL with sophisticated custom queries, but this is not how sampling is usually performed). The data set that is being explored for the purposes of this project is 1.9 GB in size, which has proved to be too large for in-memory (read: loaded into RAM) operations on most standard computers.

The 1.9 GB dataset was obtained by downloading numerous tables from the SQL server and joining the relevant tables that are needed to access alarm data. After downloading, file sizes are reduced and the files are serialized as a feather file. This process ran on a very high-end machine, e.g 2020 I9 processor w/32 GBs RAM, however, it would not run on laptops and older desktop computers. In an attempt to standardize our working environment, we used Docker containers to manage our development environment. This, however, had some unintended side effects that future teams should be aware of.

The addition of Docker had the unfortunate effect of increasing the required system resources to process the data, and the increased complexity added ended up hindering our efforts further. This is not to say that Docker should not be used for future local development projects, but care should be employed when setting it up, as it will not always work out of the box.

This points towards the primary thesis of this section, which is that a framework should be set up on the Google Cloud Platform (GCP) that accomplishes the following tasks:

1) Provides a clone of the data set on the SQL server
2) Has Spark, or another similar big data processing framework, installed, configured, and running
3) Has Jupyter notebooks set up to interface with Spark, which runs on GCP

Putting in the time and money to migrate this project to the cloud would mitigate the problems that we have encountered in reliably accessing the data and performing analytics without overwhelming local hardware. The cost of migrating to the cloud would not be large, as the primary reason for doing so is to gain access to a uniform set of more powerful machines, and the actual uptime for the cloud services would be small during development. It is worth

noting that Amazon Web Services (AWS) is less expensive than the Google Cloud Platform, but more difficult to use.

Migrating the product to the cloud will provide a foundation for consistent development that was lacking using low-performance personal machines. With the migration complete, the next focus would be cleaning and transforming the data into a suitable form for analytics. Our team approached this by flattening out the star schema organization of various tables, with the goal of having a single data set to work with. However, the complexities of a business-grade database proved to be greater than anticipated, and as a result, we may not have accessed the most useful data in the database. As such, it is recommended to devote sufficient manpower to ensure that the data is cleaned and transformed appropriately. Apache Spark is a big data analytics engine that is suitable for this task, and it crucially allows for user manipulation via SQL. That is, while Spark stores information as a RDD (Resilient Distributed Dataset), which is essentially a method to perform actions on data in parallel on many machines, Spark has a SQL API that allows for data manipulation without touching the underlying complexities of an RDD.

This means that a knowledgeable SQL programmer could transform the data, and do some of the cleanings, without learning a new skill set. This would be advantageous due to the fact that SQL expertise is already present at Terumo, and that expertise could be used to ensure that the best data is being used for analytics. Our development team realized late in our work cycle that we may have not been using the elements from the dataset, so it would be wise to confirm that the dataset is optimal before proceeding to the analytical phase. It is worth noting that all of the SQL queries used should be saved for later use, as this process will need to be automated when the project is moved to production.

Once the transformed and mostly cleaned dataset is present on the cloud, standard cleaning operations will need to be performed to drop rows with missing values, test values, etc. With this complete, the data can then be saved in a static database that will not be updated until the end of development. Unless major changes are made to Reveos or its firmware, a static database should be sufficient for most of the development cycle, as it is assumed that the fundamental problem will not change substantially. That is, while the causes of anomalies in Reveos machine performance will change, the set of variables that captures said anomalies will remain consistent.

The final item that may be useful to future development teams is that grouping the data by different time intervals, by the day, the week, the month, etc, is a process that needs to be performed on the data before applying Machine Learning techniques. For this reason, it would be wise to parameterize code so that datasets that are grouped in different intervals can be easily inputted into the chosen ML techniques.

2. Data Collection and Reduction

Connecting to the database is relatively easy, as the function simply takes an argument with the IP address of the SQL server, the database type, user id, and password. Once the connection is established, all files are saved as a serialized object to disk in a feather format.

```
dim_blood_product = pd.DataFrame(pd.read_sql_query("""SELECT * FROM v_dim_blood_product""", conn))
dim_blood_product.to_feather("../data/raw/dim_blood_product.feather")
del dim_blood_product
gc.collect()
```

1. Code snippet for gathering data from SQL server. Found in "Getting Data From DB.ipynb".

Saving the results of the SQL queries in a Pandas dataframe that is then written to disk as a feather file allows for the initial download of data from the SQL server to only occur one time. Furthermore, the serialized files can be easily shared, which allows our team to easily replicate the files without the need of setting up a VPN connection for each team member. Note the calls to delete the original database file, and the call to the garbage collector, gc.collect() - These operations are crucial on low-end machines to prevent errors occurring due to a lack of memory.

It is also recommended that the dataset undergoes variable type reduction. That is to say, when possible, the data types within each column should be reduced to their smallest possible binary representation.

```
def reduce_mem_usage(df, make_sparse=False):
    start_mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage of dataframe is :",start_mem_usg," MB")
    # First drop empty columns
    df.dropna(axis=1,how='all', inplace=True)
    for col in df.columns:
        # Print current column type
        print("******************************")
        print("Column: ",col)
        print("dtype before: ",df[col].dtype)
        if str(df[col].dtype) in ["object", "string"]:
            print(df[col].nunique(), len(df[col]), (df[col].nunique() / len(df[col])))
            if (df[col].nunique() / len(df[col])) < 0.5:
                df.loc[:,col] = df[col].astype('category')
        elif str(df[col].dtype).lower() == "int64":
            # Make Integer/unsigned Integer datatypes
            mx = df[col].max()
            mn = df[col].min()
            try:
                if mn >= 0:
                    if mx < 255:
                        df[col] = df[col].astype(np.uint8)
                    elif mx < 65535:
                        df[col] = df[col].astype(np.uint16)
                    elif mx < 4294967295:
                        df[col] = df[col].astype(np.uint32)
                    else:
                        df[col] = df[col].astype(np.uint64)
                else:
                    if mn > np.iinfo(np.int8).min and mx < np.iinfo(np.int8).max:
                        df[col] = df[col].astype(np.int8)
                    elif mn > np.iinfo(np.int16).min and mx < np.iinfo(np.int16).max:
                        df[col] = df[col].astype(np.int16)
                    elif mn > np.iinfo(np.int32).min and mx < np.iinfo(np.int32).max:
                        df[col] = df[col].astype(np.int32)
                    elif mn > np.iinfo(np.int64).min and mx < np.iinfo(np.int64).max:
                        df[col] = df[col].astype(np.int64)
            except ValueError:
                df[col] = df[col].astype("Int64")
                # Make float datatypes 32 bit
        elif str(df[col].dtype) == "float64":
            df[col] = df[col].astype(np.float32)

        if (len(df[col].dropna()) / len(df[col]) < 0.25) and make_sparse:
            df[col] = pd.arrays.SparseArray(df[col], dtype = df[col].dtype)
        # Print new column type
        print("dtype after: ",df[col].dtype)
        print("******************************")

    # Print final result
    print("___MEMORY USAGE AFTER COMPLETION:___")
    mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage is: ",mem_usg," MB")
    print("This is ",100 * mem_usg / start_mem_usg,"% of the initial size")
    return df
```

2. Code snippet for reducing data types for memory usage. Found in “Merging Data.ipynb”.

We found it useful to convert data types from int64 to int32 or lower when possible. Doing so allowed for much more data to be loaded to RAM and a more efficient processing of the models. This also reduced the memory footprint significantly for each pandas dataframe, allowing for lower performance machines to compute some form of statistical analysis.

3. Data Analysis Methods & Results

Once the data collection and cleaning had been performed and the data types had been optimized, we began to perform data analysis methods. Initially, we started by printing a small subset of the dataset and began to look into the different columns along with prominent values that seemed anomalous. A few of those columns we initially looked at included procedure duration minutes, run duration minutes, basin temperatures, and each alarm type.

As we were tasked with finding cases where anomalous figures were appearing, we decided to focus on finding patterns where an exception alarm would be raised. Our main focus included the previously mentioned columns, so we began by computing statistics for each of the columns we found interest in. We started by analyzing the column related to the number of alerts being thrown. We found that on average, 1/3rd of runs would produce at least one type of alert, so we started to look for any commonalities for when an alert was triggered. To our eyes, nothing seemed anomalous in that regard, so we moved on to looking at exception alarms instead.

Total Number of Alerts	7,189,876
Average Alerts Per Procedure	0.33083848895035606
Standard Deviation of Alerts	0.838588508313418

3. Table of statistics for 'number_of_alerts' column.

By performing a similar technique, when looking at the exception alarms, our team found that a significant change in basin temperature from the start of a run to the end of a run produced a greater chance of an exception alarm being triggered. We noted this and continued our investigation.

Another interesting finding our team had was the duration of each procedure. We wondered if longer procedure times also increased the amount of exceptions that would be raised during a cycle. We looked further into this possibility by looking at the dataframe for instances where the procedure time was above average and if the alarms were being triggered more commonly as a result. Although we did not find any direct pattern there, we assumed that such a column would be very helpful for training models for detecting anomalous behavior.

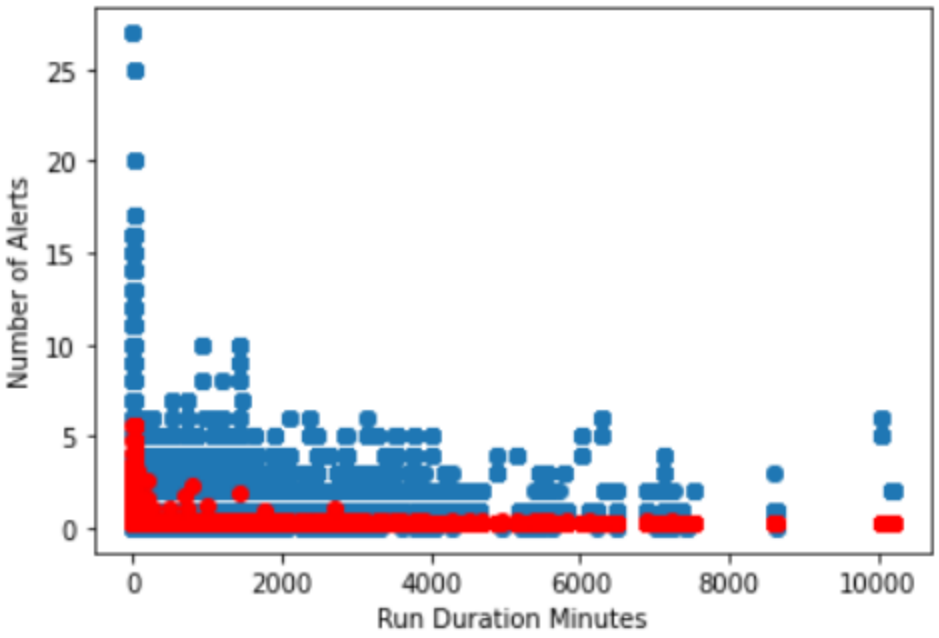
After getting a better understanding of the data we were working with and identifying patterns capable of being seen, we gathered our notes and began brainstorming the different types of models that would work given the data and our goal.

4. Regressive Classification Models

Regression analysis is a predictive modeling technique that analyzes the relation between the target or dependent variable and independent variable in a dataset. The different types of regression analysis techniques get used when the target and independent variables show a linear or non-linear relationship between each other, and the target variable contains continuous values. These regression techniques then get used mainly to determine the predictor strength, forecast trend, time series, and in case of cause & effect relation.

We started by creating the most common regression models, linear regression and logistic regression. Most of our group's experience came from using linear and logistic regression, and as such, we decided to start our initial modeling endeavors by using the simplest of the two, linear regression.

For our regression models, rather than building them ourselves, we used the sklearn package from scikit-learn. As is good practice, we initially started by setting up and separating our test data from our train data. Once we had properly created our subsets of data, we selected the columns we felt appropriate for beginning with. Initially, our subset of columns proved to be far too large for any of our computers to compute, so we reduced the number of features we planned on using to just a few per model. For the linear regression model, we had little hope it would accurately predict anomalies, due to its limited scope. For our first linear model, we trained it on the feature ‘run_duration_minutes’ against ‘number_of_alerts’. As expected, this model performed rather poorly, only successfully predicting an alert on 0.2% of procedures. We then repeated the same model with a different subset of features, this time using ‘run_duration_minutes’ to predict ‘number_of_exception_alarms’. Similar to the previous model, this model did not fare any better with this subset of features either, successfully predicting an exception alarm on 1% of procedures.

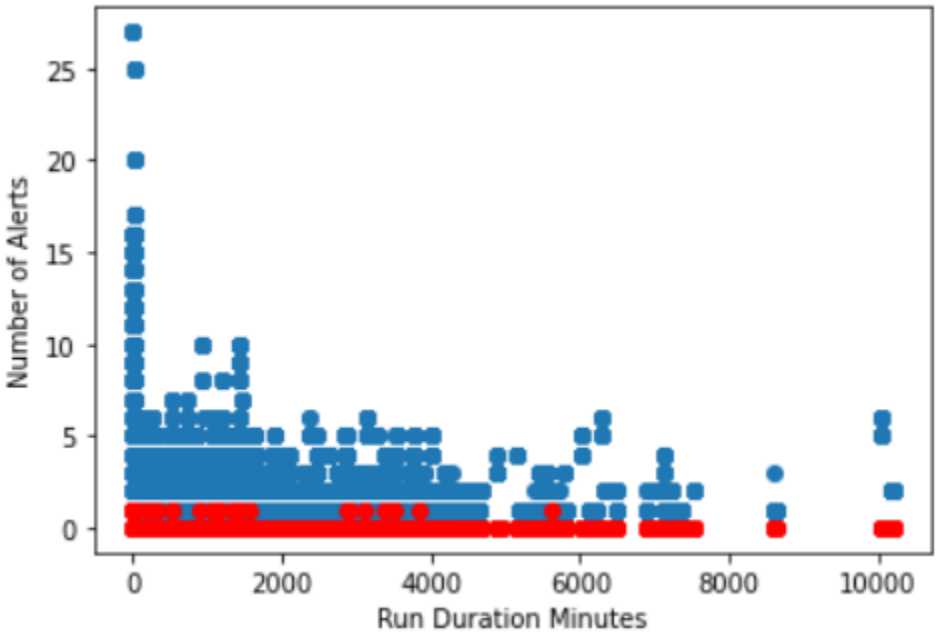


4. Graph representing the number of alerts in relation to run duration. Blue points represent the test data and red points represent the predicted number of alerts given a run duration from the model. Found in "Linear Regression.ipynb".

Next, we created a logistic regression model similar to the linear regression model. Once again, rather than creating our own logistic regression model, we used the model provided by sklearn and trained that instead. Upon using the logistic regression model, our computers started to prove to be a hindrance. Up to this point, loading the data into their respective dataframes utilized nearly 95% of the ram on our best computer, and training a model increased the consumption of ram and CPU significantly. As a result, our models were limited to a few selected features per run.

We began by using the same features as the linear regression model had, using 'run_duration_minutes' as a feature to predict 'number_of_alerts'. Immediately, we noticed an improvement in our models' accuracy scores. By using a logistic regression model, our accuracy for predicting 'number_of_alerts' went from 0.2% to an overwhelming accuracy of 80%. The

same could be said for when we trained the model against 'number_of_exception_alarms'. We worked on trying to improve our model by using different solvers since initially we were only using the provided 'saga' solver. However, we noticed no discernible improvement to the accuracy of the model when using different solvers like the 'liblinear' solver.



5. Graph representing the number of alerts in relation to run duration. Blue points represent the test data and red points represent the predicted number of alerts given a run duration from the model. Found in "Logistic Regression.ipynb".

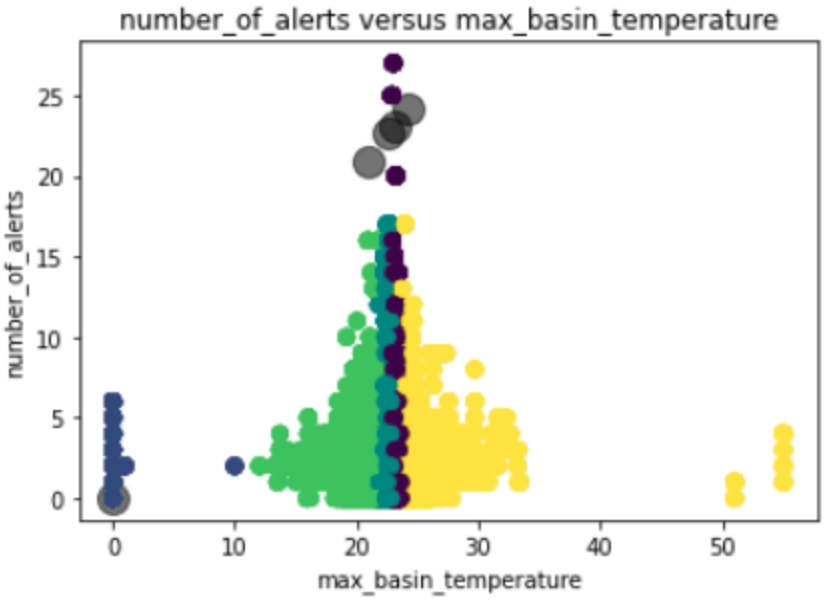
5. Clustering Classification Models

K-Means is a specific clustering algorithm that partitions all observations or data into K different clusters. It does so by initializing all data into a cluster (randomly or methodically) and then calculating the mean of that cluster, creating a centroid for a cluster. Then it determines if each data point is closer to another cluster's centroid, if it is, changes its group to the new one, otherwise leave it in the original group. Once all data is regrouped, it recalculates the mean thereby changing the centroids, and the algorithm repeats the total process until a stopping

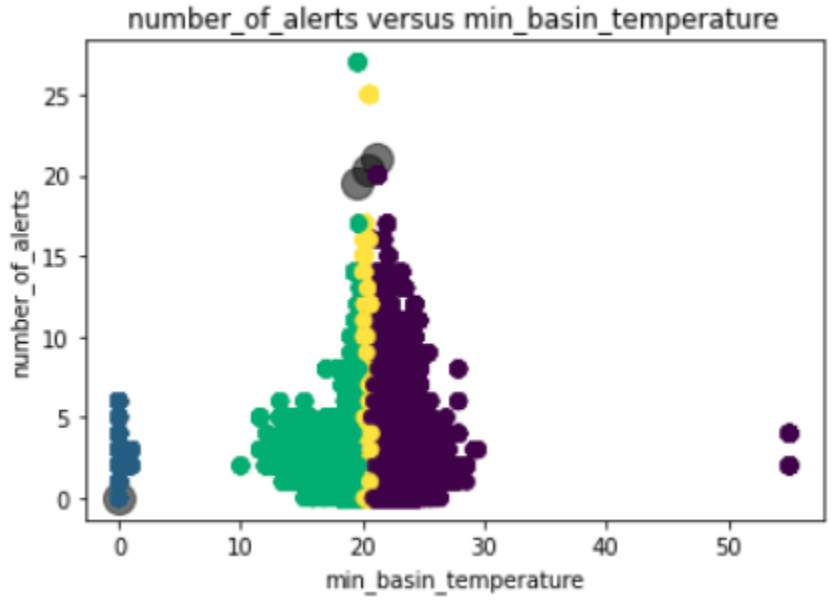
condition is met. A stopping condition could be a set number of iterations of regrouping and recalculating means, or determining if the difference between centroids from consecutive iterations is minimal.

In our use of K-Means, we use the implementation from Sklearn. Additionally, because of the large amount of data, we had to be selective with which elements of the data we used. K-Means can be used with more than 2 elements, but due to our computational constraints, we kept the number of elements to 2. Specifically, we paired a lot of different columns of data with the number of alerts. We did this to see if there was any relationship between errors and a specific event happening. The specific columns we used in combination with a number of alerts were: product volume, minimum temperature, maximum temperature, temperature out of range exceptions, run duration, and procedure duration. With more computation power, you could consider pairing a number of alerts with minimum temperature, maximum temperature, and temperature out of range exceptions to get greater insights into how temperature is affecting alerts.

Taking a look at our procedure, we first hit our temporary dataframe with a .described() function to see the spread of data. We then call K-Means on the data, playing around with the number of clusters to fit each group into a distinct set of sub-data groups, such that each group is distinct enough to be on its own. Once K-Means is run, we then take the results and put them into a scatter plot, organizing data by color, giving each cluster its own color.



6 . Graph representing the number of alerts in relation to maximum basin temperature. Different colors represent different colors. Found in "k-means.ipynb".



7. Graph representing the number of alerts in relation to minimum basin temperature. Different colors represent different colors. Found in "k-means.ipynb".

From the above data, we can see the temperature in relation to the number of alerts following a sort of standard distribution. It is important to note that a majority of the data points will lie in that center cluster, which may get hidden in the graph. Similarly, we can see the data points with the most alerts again lie in the middle of the pack, perhaps showing the more catastrophic errors due to another reason, rather than extreme temperatures. Further, in both graphs above, we can see clusters on the edges of each graph, at temperatures around 0 degrees and above 50 degrees. Due to the discontinuous flow of the temperature readings, perhaps those extremes are due to errors in temperature readings or procedural errors.

6. Ensemble Classification Models

On top of the other models and techniques we used, we also used an ensemble classification model. Simply put, ensemble models attempt to learn not just one classifier, but rather a whole set of classifiers, also known as an ensemble of classifiers. The model uses all these classifiers and then aggregates the results from those classifiers. After all the results are grouped and added up, the model uses the most common result as its prediction. We tried three different ensemble methods from the SKLearn package: Bagging Classifier, AdaBoost Classifier, and the Gradient Boosting Classifier, but we will focus on the Bagging classifier.

To start, we will focus on the bagging classifier and the results attained. The bagging classifier forms a class of algorithms that are all used to build several instances of a black box estimator which then run on subsets of the training data. After training on that data, the model collects each algorithm's individual predictions to then form a final prediction. In our own testing, we ran into some issues using this classifier, however, we were able to get a model that completed training. We trained the model on the same subset of features as used in the previous

two models, using a mix of ‘run_duration_minutes’, ‘min_temperature_basin’, in order to predict ‘number_of_alerts’ and ‘number_of_exception_alarms’. In the end, we noticed that the results for the bagging classifier ensemble model were lacking. Not only were the accuracy results significantly worse than both the regression models and the clustering models, they also took a large amount of time to run. We decided that focusing on these models would not prove worthwhile, given the other models that we tested.

When attempting to implement the other two classifiers, Adaboost and Gradient Boost, we ran into similar problems, except we never could get a model to finish training when using them. We recommend that more research be done specifically with these two classifiers in mind since they appear to be more well suited for the data we were using to train on.

7. TensorFlow Classification Models

TensorFlow is the open-source library for a number of various tasks in machine learning. Included in TensorFlow is Keras, a high-level API that is built in Python and is tightly integrated into the TensorFlow ecosystem. Keras is a much easier and more user-friendly tool than TensorFlow, which is why we focused on using that rather than the latter.

To start using Keras, we first investigated the different models and how each model is created. There are a number of different models that can be created and used with Keras, and after our initial investigation, we slowly learned the different layers and when to use each layer. For example, we started by making a Keras model that was sequential, that is to say, each layer within that model would be run one after the other, where the output of one layer becomes the input for the next layer and so on. The first layer we have the model run is the Input layer, which simply builds a Keras model just through the inputs and outputs of the model. The second layer

is the dense layer, which allows the model to create a deeply connected layer in the neural network, where each of the neurons of the dense layers receives input from all neurons of the previous layer. The third layer is a 1D convolution layer, which creates a convolution kernel that is convolved with the layer input over a single spatial dimension to produce a tensor of outputs. These outputs then go to a dropout layer. This layer is perhaps one of the more important layers, since the dropout layer helps reduce overfitting in neural network models. If the model overfits, then the model would not work for more generalized data, making it useless. Following the dropout layer, we have another 1D convolution layer, which performs the same computations as the previous 1D layer, but this time without any of the data that may have potentially produced some overfitting. The outputs from the previous 1D convolution layer then goes to the 1D convolution transpose layer, which does the same computation as the 1D convolution layer but transposes the results. Once again, we follow that layer with a dropout layer to help prevent any overfitting that may be occurring within the model. Finally, we follow the dropout layers with two 1D convolution transpose layers.

While seeming complex, Keras is actually very easy to use and train once the model has been defined. The next step was to pick an optimizer that we would want to use with our Keras model. We tested our Keras model with a few different optimizers but found the best results came when we used the Adam optimizer. Initially, we were using the Adagrad and SGD optimizers, but never were able to get a complete run of the model successfully training, rather the model seemed to train endlessly. When we switched to using the Adam optimizer, our Keras model still took a while to run, easily running for over 12+ hours, but we did manage to get a successful result.

```
model = keras.Sequential(
    [
        layers.Input(shape=(train_data.shape[1], train_data.shape[2])),
        layers.Dense(16, input_shape = (8,)),
        layers.Conv1D(
            filters=32, kernel_size=7, padding="same", strides=2, activation="relu"
        ),
        layers.Dropout(rate=0.2),
        layers.Conv1D(
            filters=16, kernel_size=7, padding="same", strides=2, activation="relu"
        ),
        layers.Conv1DTranspose(
            filters=16, kernel_size=7, padding="same", strides=2, activation="relu"
        ),
        layers.Dropout(rate=0.2),
        layers.Conv1DTranspose(
            filters=32, kernel_size=7, padding="same", strides=2, activation="relu"
        ),
        layers.Conv1DTranspose(filters=1, kernel_size=7, padding="same"),
    ]
)
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss="mse")
model.summary()
```

8. The Keras model that we created and implemented, along with each layer. Found in "TensorFlow.ipynb".

Despite this, however, actually fitting the model to our data took even more computation time, and resulted in us not being able to attain any results due to the time we lacked. We recommend that this Keras model be investigated further, as we believe the results from the model may be insightful for future development purposes.

```
history = model.fit(
    x_train,
    x_train,
    epochs=50,
    batch_size=128,
    validation_split=0.1,
    callbacks=[
        keras.callbacks.EarlyStopping(monitor="val_loss", patience=5, mode="min")
    ],
)
```

9. The Keras model being fit to the training data. Found in "TensorFlow.ipynb".

8. Anomaly Detection and Preventative Maintenance

Anomaly detection is the first step towards implementing a strategy that can direct analyst attention towards new problems in the Reveos system, and when combined with other techniques can allow for preventative maintenance programs to function. Often defined as “the process of finding patterns in data that do not conform to expected behavior ”, an understanding of the different types of anomalies, and methods applicable to said types is a crucial first step in implementing a system to automate the detection of anomalies.

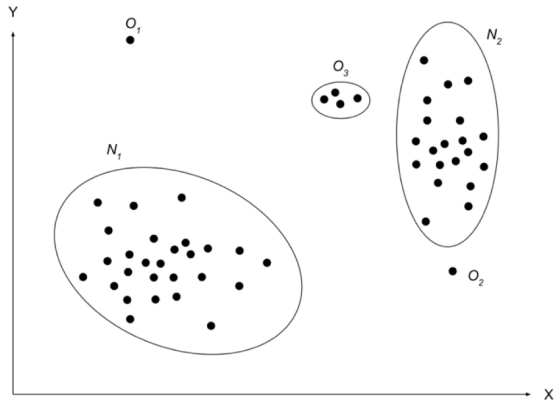


Fig (1) A basic set of anomalies {O1, O2, points in O3} in a 2d data set.
Image courtesy of Chandola, Banerjee and Kumar

A variety of data science and Machine Learning techniques are applicable to anomaly detection past the neural networks (TensorFlow), regressive, clustering, and ensemble methods employed by our team. These methods include classification, clustering, nearest neighbor and statistical models, along with information theory and data mining techniques. The nature of the data, along with the absence or presence of labels, and the desired type of anomaly to detect all play a role in selecting the proper technique.

There are three broad categories of which anomalies can be categorized. The first, and most simple is a *point anomaly*, which is a single point, representing a vector, which is outside the boundary of the region defined as normal. In figure 1, O1 and O2 are representative of point

anomalies, along with the points contained within O3. Since O3 only contains 4 points, and that number is much less than the points found in N1 and N2, it is still classified as anomalous even though the points within are not singletons.

The second broad class of anomaly are *contextual anomalies* which are, as the name suggests, data vectors that are only anomalous within a specific context, which is depicted in figure 2.

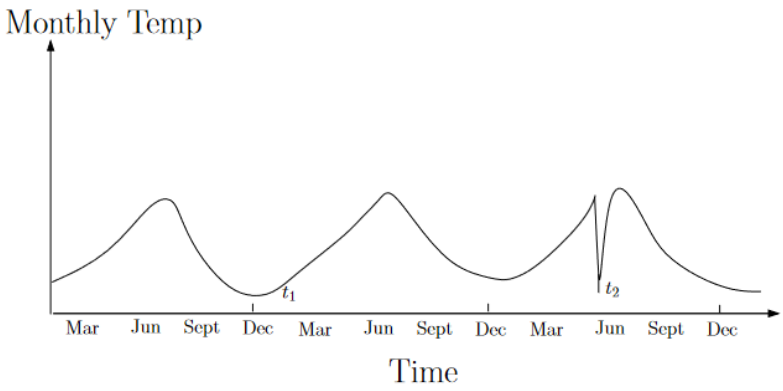


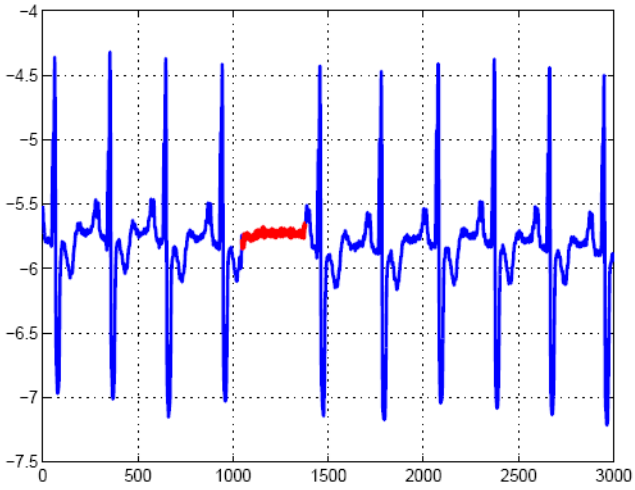
Fig (2) A contextual anomaly, t2, present in a time series. Note that t1 is of the same value, but in a different and non anomalous context.
Image courtesy of Chandola, Banerjee and Kumar

A contextual anomaly is defined by two sets of attributes, its *contextual attributes* and its *behavioral attributes*. Behavioral attributes are simply the value of the variable of interest, which in the above example is the monthly temperature. Contextual attributes are used to determine the context, which in the above example is time. A possible contextual attribute for the above example would be if the change between two successive time slices is past a defined threshold. Contextual anomalies are usually explored in either time series data, or spatial data. Since the Reveos system does not have a spatial element for this problem, time series data would be the class of data relevant to Reveos.

The final class of anomaly to be examined is a *collective anomaly*, which shares some similarities with contextual anomalies. Whereas a contextual anomaly looks in its immediate

neighborhood to determine the context, a collective anomaly looks at the entire data set, which is depicted in figure 3.

Fig (3) A collective anomaly, in red, present in a time series. Note that a value within the expected range continues for an abnormally long time. Image courtesy of Chandola, Banerjee and Kumar



Collective anomalies are generally applied to the context of sequence data, graph data, or spatial data. Within the realm of Reveos, spatial data would likely not have an application, but sequence data and graph data certainly could. It is worth noting that if graph data were used that a significant amount of thought would need to be expended towards representing the system as a graph.

The level of supervision required for anomaly detection is dependent on the availability of labels for the dataset used. If no labels are present, and one can reasonably make the assumption that normal instances are far more frequent than anomalous, *unsupervised anomaly detection* can be used. Unsupervised is certainly viable for Reveos, however, it may be necessary to segregate the data into very broad categories, such as high throughput and standard throughput, depending on a very general use case for the system. This segregation would be necessary if it was determined that the aforementioned use case categories result in markedly

different rates of anomalies occurring, which this author believes to be possible, but has not verified if that is the case.

If it is possible to obtain labels for only normal behavior, then it is possible to perform *semi-supervised anomaly* detection. While it may seem strange at first glance to only have normal behavior labeled, this could be accomplished by curating a dataset that corresponds to machines that are maintained at regular intervals with trained staff, perhaps from customers with a history of excellent performance, or a Terumo pilot facility.

Finally, if labels are available for normal and anomalous classes, then *supervised anomaly* detection may be used. The fully labeled data is then used to build a predictive model which is used for the classification of data points. This is often a difficult model to implement because the datasets are usually highly imbalanced in favor of the normal class, and it is also difficult to obtain a fully labeled dataset. One possible way to obtain a labeled dataset at low cost would be to use the maintenance dates and codes to select data that prompted an emergency maintenance event, which could then be labeled as anomalous. The viability of adopting a supervised approach would depend almost entirely on rather it is reasonably possible to label the anomalous data.

For all methods of anomaly detection, it is worth considering over the long term whether the model would provide greater utility if it outputted a score instead of a label. That is, different events could be assigned scores that correspond to the severity of the anomaly. While outputting a score would make it easier for analysts to direct attention towards the most pressing issues, creating the score in a reasonable fashion presents a problem of its own, and as such it would be wise to deploy a scoring model after a classification model has been perfected.

9. Further Datasets

Employing more granular data, such as the signal level datasets collected by each Reveos device would offer the potential to gain further insight into low-level events, but at the cost of greatly increasing the complexity of the model. Since the number of variables present in such a dataset would likely be large, a technique such as principal component analysis (PCA) would need to be performed to see which variables are useful. PCA offers a method to determine which variables explain the variance in the data, or in other words, which variables are useful in building a model. It would be very useful to have a polished model of some kind using the log-level data to compare the results between the two. It is quite possible that variance in the signal data is effectively captured by the log data, as mapping signals to logs could represent a reduction of dimensions that retains most of the information.