University of Nevada, Reno

Department of Computer Science Engineering

Design Document

November 29th, 2016

**GE Machine Learning:**

Patrick Kelley

Brian Gaunt

Harrison Stanton

Samson Haile

**Instructors:**

Eelke Folmer

Devrin Lee

**External Advisors:**

Denver Liu

Thomas Avant

James McDuffie

# Table of Contents

Abstract 2

Introduction 3

High-Level and Medium-Level Design 4

Detailed Design 8

User Interface Design 12

Annotated References 18

Contribution of Team Members 19

Glossary Updates 20

# Abstract

The goal of this project is to use machine learning to detect potential future failures in machines used by the clients of General Electric. The team plans to implement multiple techniques for comparison and to choose the most effective one, ensuring confidence in our proposed method. To accomplish this, the team will have to collect data from a rotor kit supplied by General Electric; this data will be gathered by analyzing sensor data output by the rotor kit in normal conditions as well as simulated failure conditions. The difficulty of collecting this data will play a large role in the accuracy of the tested algorithms, and is likely to constrain the team’s decision. The team will also have access to the Fleet API, which assists in getting the data from the rotor kit.

# Introduction

For this Software Engineering class our group will be working with General Electric(GE). The problem as presented by GE is to detect and predict error modes for turbines. Currently, the process is handled by GE’s employees: They have employees who go out, pull the data, and then provide information on why errors are happening. The purpose of our project is to design and implement a machine learning algorithm to predict and classify failure modes on their turbines. Having a system in place to monitor the turbines greatly reduces the amount of time that GE has to spend on maintenance.

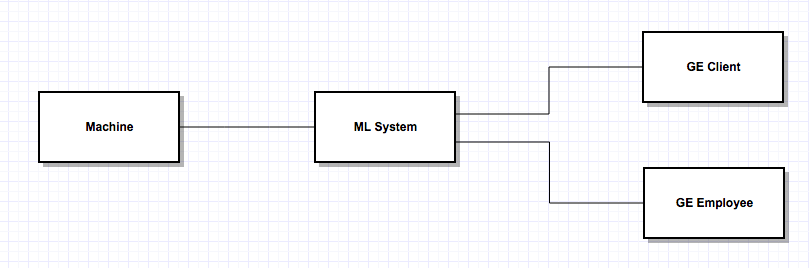
Our system will be able to monitor for error modes and be able to alert the client without the need for one of GE’s employees to manually comb over the data. Our plan to achieve this is as follows: First we will gather data from their rotor testing units for various levels of failure modes as well as proper operation. This will be done through usage of the Fleet API they are providing to us. More details on the API can be found below. After we obtain the data, we must then parse the data in a way that allows for usage in supervised learning techniques. This step will include labeling the data(in order to include the state of the turbine along with the corresponding data for that time period). Depending on the models that we employ after this step, we may need to further parse the data in order to develop more robust features.

We will then attempt to create the model using the data we gathered: Our intention for this stage is to take full advantage of existing libraries and software to help us build our models; some such software we have looked at to help us at this stage include sklearn, OpenCV, Keras and TensorFlow. We will experiment with several different models to find the most apt tool for the job. Although the robustness and popularity of neural networks is prominent, we may choose to use other models since the amount of data we generate may fall short for training. One such model that may work well with a limited amount of data is random forests. If the results from our initial attempts are lacking, we modify the model and gather more data if needed. We will continue to modify our model until we achieve a relatively low error rate. After we reach this stage we will then continue to add on functionality for other error modes. At the end of this project, it is our goal to be able to accurately predict and classify different error modes for GE’s turbines.

Since writing the Specification Document, the team has thoroughly designed the structure of the program; this structure is the focus of this Design Document. Additionally, the team has made progress in using the rotor kit supplied by GE, and can begin to generate data for training.

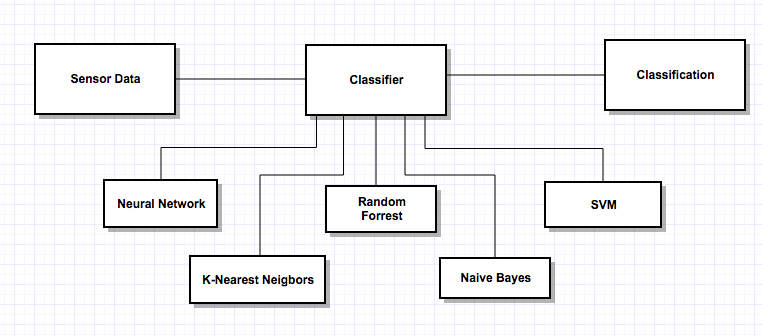
# High-level and Medium-level Design

System-Level Diagrams



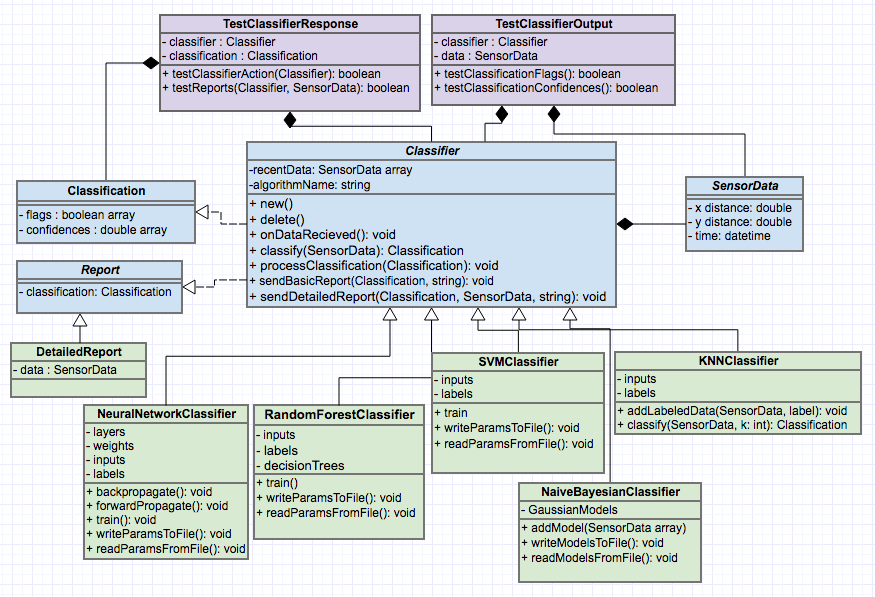
*Figure 1: Context Diagram*

Our system takes in sensor data from a machine, then informs the GE employee and client if there is a problem. The GE employee/client can also request a report of the machine’s status.



*Figure 2: Architectural Design*

Our design uses the Strategies design pattern, where the interface of the classifier is agnostic of the underlying algorithm.



*Figure 3: Class Diagram*

Classes

**TestClassifierResponse**: This class is used for testing the classification output of the classifier.

**TestClassifierOutput**: This class is used to test the measures taken by the classifier based on its classification.

**Classifier**: This class takes in sensor data and classifies it. If it finds a dangerous condition, the class will notify the GE Client and a GE employee.

**SensorData**: This class contains the sensor data output by the machine.

**Classification**: This class contains the classification data returned by the classifier.

**Report**: This is a simple summary of the machine’s state that contains the information included in the classification class.

**DetailedReport**: This report includes everything from the Report class plus the sensor data that corresponds to the classification.

**NeuralNetworkClassifier**: This is an implementation of the classifier using the neural network algorithm.

**RandomForestClassifier**: This is an implementation of the classifier using the random forest algorithm.

**SVMClassifier**: This is an implementation of the classifier using the support vector machine algorithm.

**KNNClassifier**: This is an implementation of the classifier using the k-nearest neigbors algorithm.

**NaiveBayesianClassifier**: This is an implementation of the classifier using Bayesian classification.

Selected Methods

**TestClassifierResponse:testClassifierAction**: This method will test that the classifier takes the correct action given a classification(i.e. Sends a report or not).

**TestClassifierResponse:testReports**: This method will test that the classifier is sending correct reports given the classification and sensor data. Note: the difference between this and the testClassifierAction method is that this tests the content of the report, rather than the presence of it.

**TestClassifierOutput:testClassificationFlags**: This test the array of flags in the returned classification from the Classifier:classify method.

**TestClassifierOutput:testClassificationConfidence**: This test the array of confidences in the returned classification from the Classifier:classify method.

**Classifier:new**: This is the basic constructor for the classifier. Note that each type of classifier will have it’s own initialization based on the algorithm.

**Classifier:delete**: This is the basic destructor for the classifier. Note that each type of classifier will have it’s own destructor based on the algorithm.

**Classifier:onDataRecieved**: When the classifier receives sensor data from the machine, it will trigger this function. This function will respond by calling the classify and processClassification functions.

**Classifier:classify**: Given sensor data, this method will return a classification(which consists of two arrays containing danger flags and confidences, respectively).

**Classifier:processClassification**: Given a classification, this method will determine what action to take. In particular, it will decide whether or not to notify the GE employee and client.

**Classifier:sendBasicReport**: Given a classification, send a summary of the data to the specified destination.

**Classifier:sendDetailedReport**: Given a classification and the corresponding sensor data, send a summary of the data to the specified destination.

**NeuralNetworkClassifier:writeParamsToFile**: Save the parameters learned during training to a file. This enables the network to read the parameters back in the future so it can classify without re-training. The other learning algorithms(*SVMClassifier*, *RandomForestClassifier*) have a similar function.

**NeuralNetworkClassifier:readParamsFromFile**: Read parameters back from a previously saved file. The other learning algorithms(*SVMClassifier*, *RandomForestClassifier*) have a similar function.

**NaiveBayesianClassifier:writeModelsToFile**: This is very similar to the *NeuralNetworkClassifier:writeParamsToFile* method, but a distinction is made here because the model is quite different in this case, as is simply summarizes the data by class and only this small amount of data is stored in the file.

**NaiveBayesianClassifier:readModelsFromFile**: This method is very similar to the *NeuralNetworkClassifier:readParamsFromFile*, but a distinction is made here for the same reason as described in *NaiveBayesianClassifier:writeModelsToFile*.

**NeuralNetworkClassifier:train**: Based on the inputs and outputs of the training data, the algorithm uses the backpropagation algorithm to iteratively update the ‘weights’ in it’s ‘layers’.

**NeuralNetworkClassifier:classify**: Using the forwardPropagate method, the classifier runs a datum through the network and returns its classification.

**RandomForestClassifier:train**: This algorithm trains many decision trees based on the training data.

**RandomForestClassifier:classify**: To classify, the algorithm runs the input data through the decision trees, then returns the average prediction.

**SVMClassifier:train**: This algorithm trains by maximizing classification rate, but minimizing the complexity(or ‘VC dimension’) of its grouping.

**SVMClassifier:classify**: Find the group of best fit for the input data.

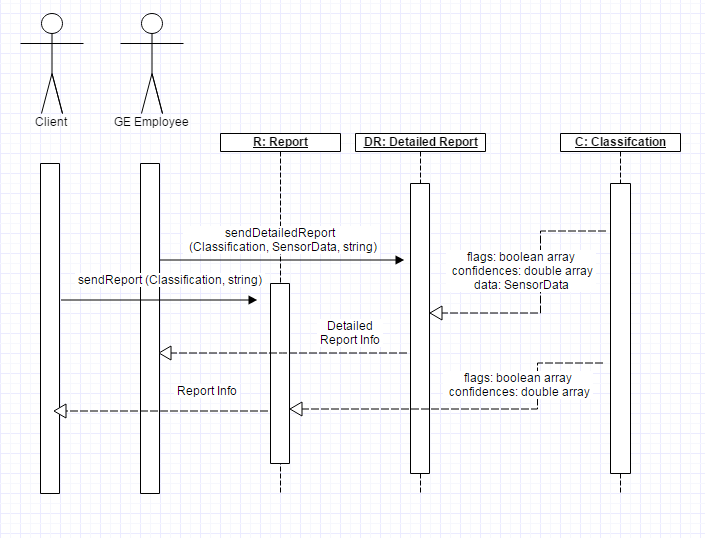
**NaiveBayesianClassifier:addModel**: Given a set of data for a class, find the mean and standard deviation for the data. Then add this model(along with it’s label) to the list of stored models.

**NaiveBayesianClassifier:classify**: Based on the stored models, find the most probable class for the input data.

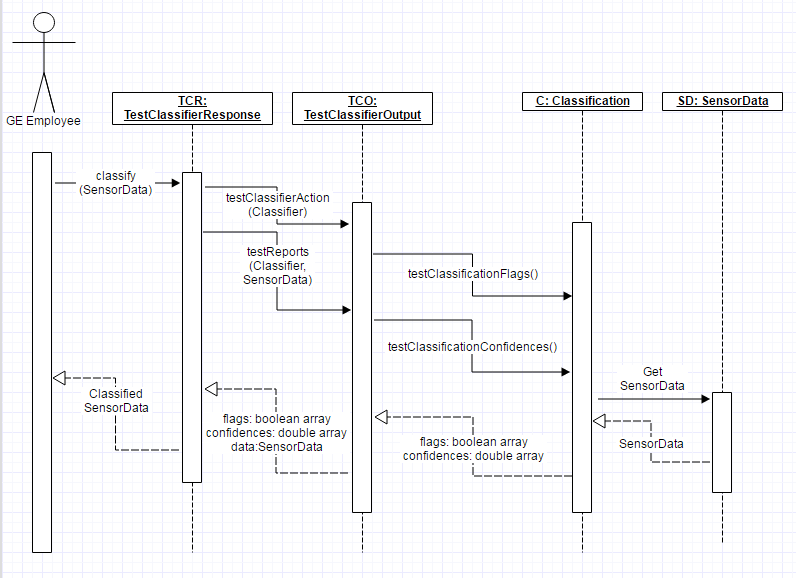
**KNNClassifier:addLabeledData**: This method adds datum to the data the classifier is using to classify data.

**KNNClassifier:classify**: Given integer k, find the k data points that are closest to the input. Return the mode of the contained classes.

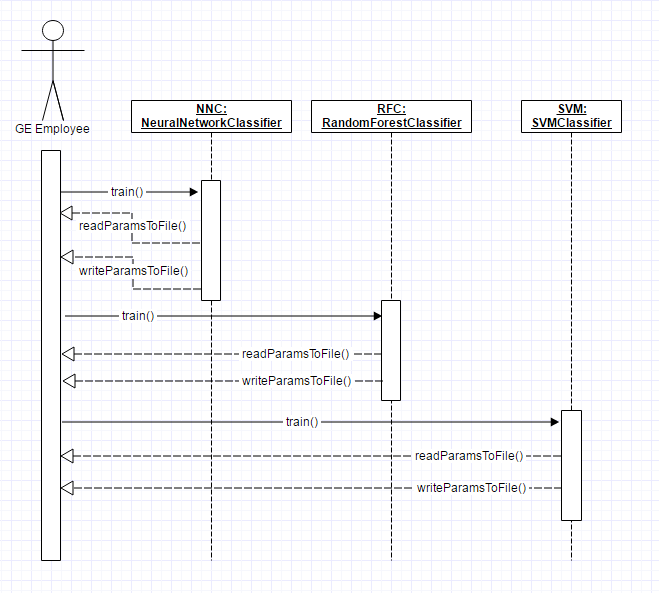
# Detailed Design



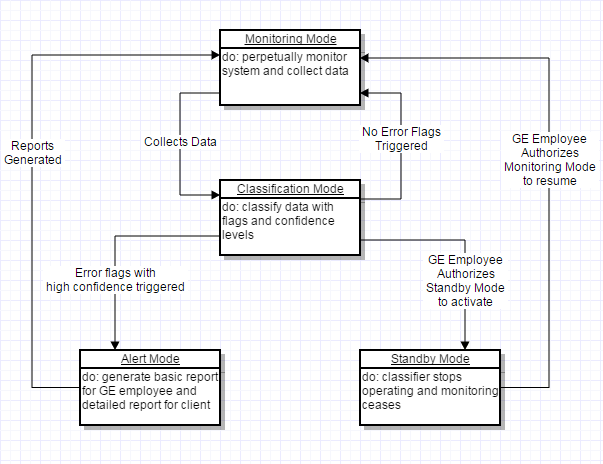
*Figure 4: Sequence Diagram - Reports*



*Figure 5: Sequence Diagram - Testing*

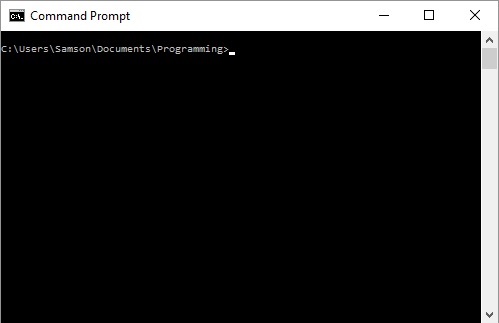


*Figure 6: Sequence Diagram - Training*

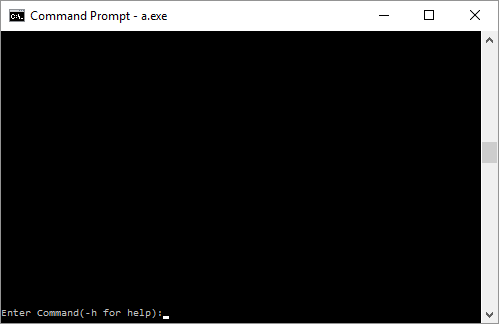


*Figure 7: State Diagram - Classifier Modes*

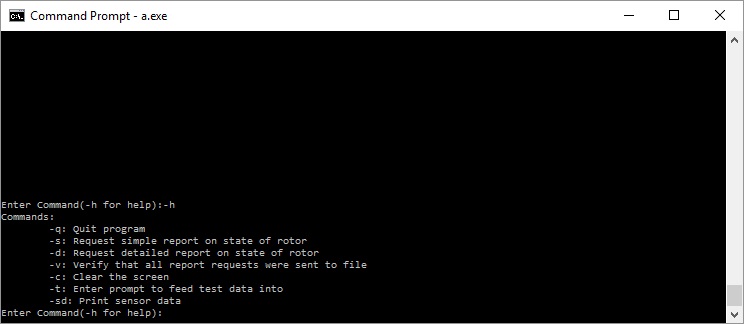
# User Interface Design



*Figure 8: The Machine Learning algorithm will utilize a CLI(command line interface), comparable to Command Prompt.*



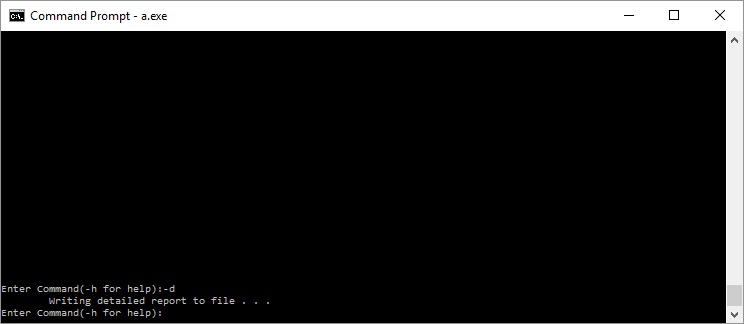
*Figure 9: The user will be able to enter appropriate commands in order to perform various tasks*



*Figure 10: Sample dialogue listing commands given prompt for help from the user.*



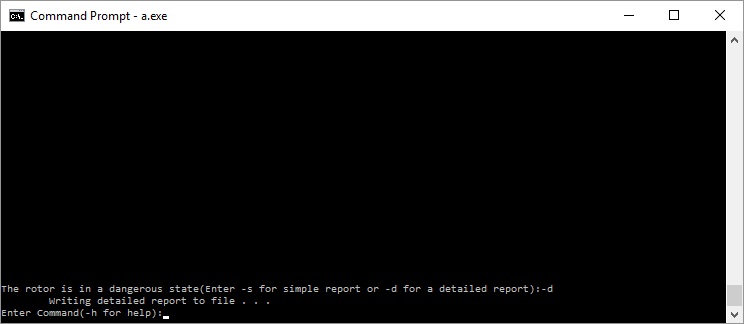
*Figure 11: The program is capable of writing reports to the file, regardless of whether the monitored turbine is in a dangerous state or not.*



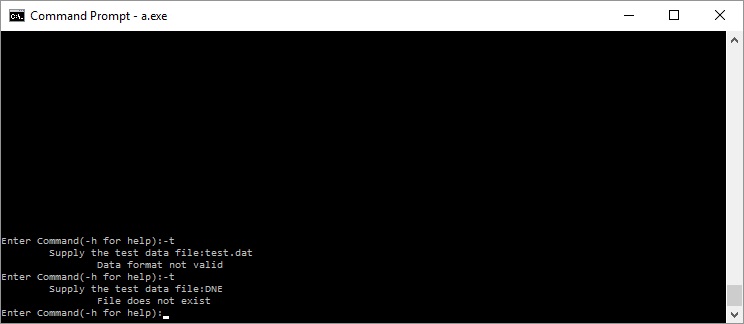
*Figure 12: Instead of a simple report, a detailed report can be specified which will include information that is not central to the classification of the dangerous state, but relevant to the monitored rotor.*



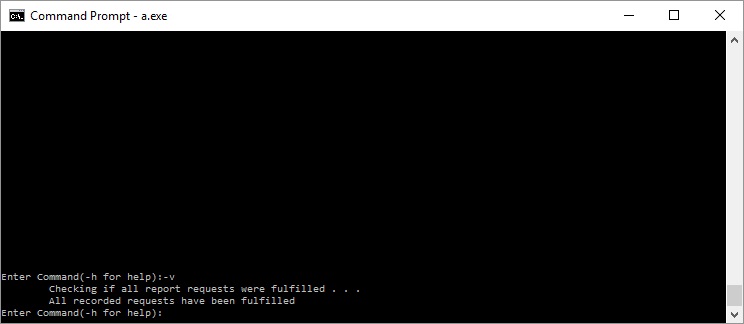
*Figure 13: The user will receive dialogue that informs them that the turbine enters a dangerous state. The user will then have the option to select which type of report they would like to receive based upon the rotor.*



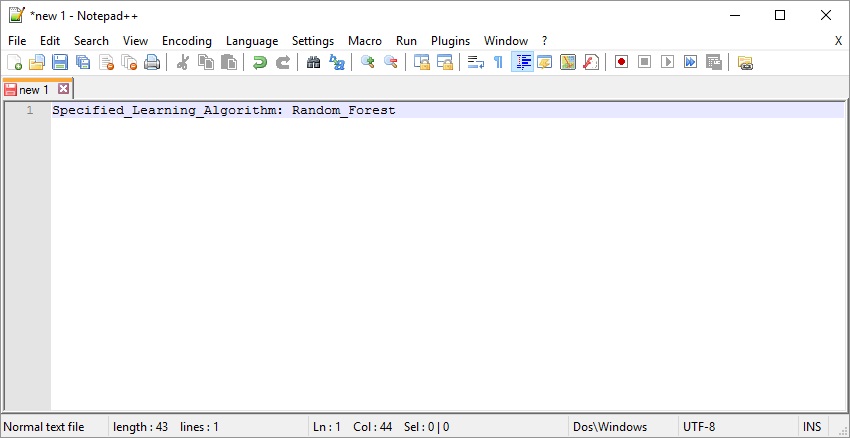
*Figure 14: The program requires the user to request a report in the event of the turbine entering a dangerous state. Upon the request, the program will write the report to file as usual*



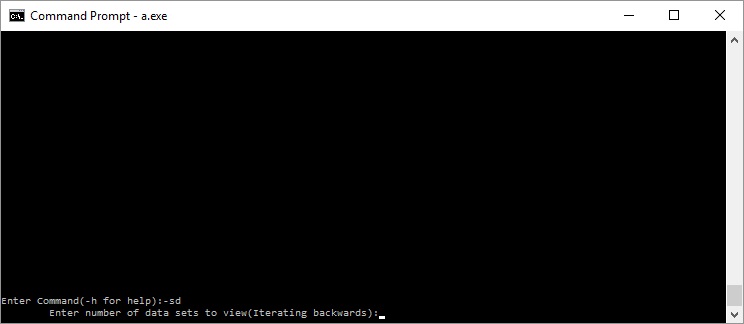
*Figure 15: Prompt for supplying test data is shown. Test data will be used by GE employees to validate the effectiveness of the system. Proper error-checking will be performed by the program depending upon the filename specified.*



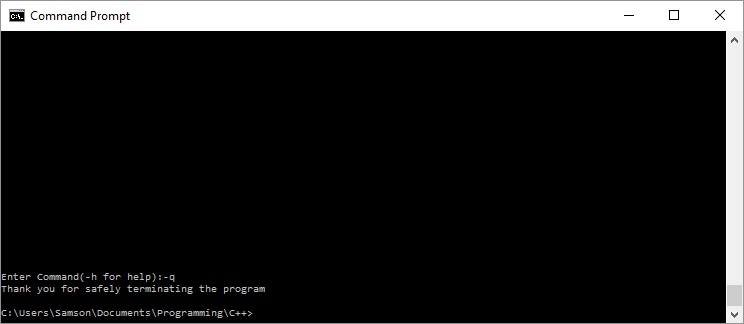
*Figure 16: Return prompt for verification of requests being fulfilled. The program checks it log kept during runtime of all requests made and compares the file writes recorded in the log to those present in the same directory as the running program.*



*Figure 17: The user can specify the machine learning algorithm used by the system to classify dangerous states of the turbine via modifying the fields of the configuration file that the program uses. The program will take precautions to prevent usage of corrupted configuration files, and restore corrupted configuration files.*



*Figure 18: Prompt for which the user responds to in order to display sensor data with regards to the rotor being monitored. The data is displayed starting with the most recently collected/stored data set, iterating through the data sets backwards in time until the xth data set from the first, for which x is indicated by the user.*



*Figure 19: The user can safely terminate the program using the -q command. Terminating the program by force would be ill-advised as some data could get lost.*

# Annotated References

* <https://www.gemeasurement.com/condition-monitoring-and-protection/software/ges-system-1-condition-monitoring-and-diagnostic> - This website describes in the purpose and performance of the System 1 Conditioning Monitoring Software that we will be using for our testing. The website outlines the System 1’s in terms of user experience, capability, accessibility, and embedded expertise.
* <http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/> - This is a tour of machine learning algorithms that describes the some of the different algorithms we may develop for this project, including supervised learning, decision trees, random forests, and neural networks which are defined in the glossary.
* <http://www.commtest.com/what_s_new/product_releases/index> - This describes the purpose of the VBX manager and how it will relate to our system. The instruments we are using are SCOUT instruments, and we can use VBX manager to configure and communicate between the instruments and the System 1 software.
* <https://coreos.com/fleet/docs/latest/api-v1.html> - This outlines the documentation and functionality of the Fleet API that we will be using in our project. It contains information such as how to create, modify, list, and delete unit entities, as well as describe how error communication will work.
* <http://www.w3schools.com/js/js_json_intro.asp> - This contains an introductory course into the inner workings of the JSON file format. The website contains information on what a JSON is, the different components of it, as well as different examples that flesh out the syntax of how the JSON file will look.
* <https://sourcemaking.com/design_patterns/strategy> - This page describes the “Strategies” design pattern employed by our design. The page first describes the motivation for the strategies design pattern and the structure of the pattern. Then the author gives a few examples of the pattern and a few guidelines to follow as well as a checklist for putting together the design. The system’s design is explained in the *High-Level and Medium-Level Design* section, and is illustrated in *Figure 2*.

# Contributions of Team Members

Patrick Kelley:

Abstract

High-Level and Medium-Level Design

Relevant References/Glossary Terms

Brian Gaunt:

Detailed Design

Annotated References

Harrison Stanton:

Introduction

Samson Haile:

User Interface Design

# Glossary

* Machine Learning(ML): A branch of artificial intelligence where the software is “trained” by processing data and adapting itself to operate better on that data.
* JSON: Stands for JavaScript Object Notation. This is a simple data format that is commonly used for sending information on the internet.
* API: Application Program Interface; the set of routines, protocols, and tools that denotes how software components should interact.
* Neural Network: A machine learning classifier that uses multiple layers of functions chained together to train a model to map given inputs to an output layer.
* Random Forest: A machine learning classifier that constructs decision trees to divide and classify the data.
* Decision trees: A technique which classifies data by subdividing it into smaller parts.
* SVM(Support Vector Machine): A type of learning algorithm that tries to minimize the complexity of its decision boundary by maximizing the distance between data points and the decision boundary
* KNN(k-Nearest Neighbors): A classification algorithm that classifies based on the data points near the given datum.
* Bayesian Classification: Statistical classification based on Baye’s rule
* Supervised Learning: A machine learning method of training models from labeled training data.