Aircraft Fault Detection through ML-based classification

1st Bhautik (Brian) Amin
College of Engineering
Drexel University
Philadelphia, PA, USA
ba555@drexel.edu

2nd Mevlut Bayram
College of Engineering
Drexel University
Philadelphia, PA, USA
mb3242@drexel.edu

3rd Atzael (Ben) Campos
College of Computing and Informatics
Drexel University
Philadelphia, PA, USA
abc347@drexel.edu

4th Richard Hong
College of Computing and Informatics
Drexel University
Philadelphia, PA, USA
rh824@drexel.edu

***Abstract*—CS 613: Machine Learning Final Project - Aircraft Actuator Fault Detection using modern machine learning techniques. Utilizing high fidelity simulations of aircraft flight with potential actuator failures, detection of stuck elevator jam events using both frequency and time domain data transformations techniques, implemented in Python.**

I. Introduction

The loss of flight control is one of the leading causes for fatal aircraft accidents. Actuator failures (The mechanisms that move control surfaces of an airplane) usually are the cause for the loss of flight control, and such a situation is hard to deal with [4] [5]. NASA has created an unmanned air vehicle (UAV) called the GTM (Generic Transport Model). The GTM is dynamically scaled from an actual transport aircraft. This small UAV has been tested in NASA's facilities, and a high fidelty model has been created for MATLAB Simulink [1]. It is noteworthy to mention that all of the aerodynamic data, actuator modeling, and engine performance in this simulation is very reliable, and the simulation results are very realistic. This opens up the opportunity to use this simulation model to generate data similar to that of a real aircraft.

In recent research papers, [2] and [3], an elevator (The actuator that controls an aircraft's pitch) jam problem is investigated in the GTM models, and it has been shown that the stuck elevator can be solved using a custom flight control system, and potential accidents can be halted. In these papers, it is assumed that the elevator is stuck at a certain time, with an unknown angle. A switching accommodation flight control system takes control from the nominal flight control system, and brings back the aircraft to the normal flight condition after the elevator is jammed. This assumption is valid because many commercial aircraft have sensors for their actuators, and if an actuator failure occurs the sensor may detect it and would be able to alert the custom control system. However, this is not the case for small planes or UAVs. Many small UAVs do not have feedback from their actuators, and if an actuator failure occurs by chance, it is nearly impossible to detect it.

In this project, the elevator jam problem is considered as an actuator failure. It is assumed that we do not get feedback from the elevator. The challenge is to detect the failure early enough so that the custom accommodation flight control system can take place and save the aircraft from a potential crash. As mentioned prior, the GTM model will be used for that purpose. The GTM h as full 12 state dynamics (features) as shown in the following equation:

$$
\begin{aligned}
&\dot{x}(t) = f(x(t),\ u(t)) \\
&x = \begin{bmatrix} V & \alpha & \beta & p & q & r & pN & pE & h & \phi & \theta & \psi \end{bmatrix}^T \\
&u = \begin{bmatrix} \delta_a & \delta_r & \delta_e & \delta_T \end{bmatrix}^T
\end{aligned} \tag{1}
$$

The 12 state variables are V: total speed (knots), α: angle of attack (rad), β: side slip (rad), p: roll rate (rad/s), q: pitch rate (rad/s), r: yaw rate (rad/s), pN: position North (ft), pE: position East (ft), h: the altitude (ft), ϕ: roll angle (rad), θ: pitch angle (rad), and ψ: yaw (rad). The 4 control inputs are δa: aileron (rad), δr: rudder, (rad), e: elevator (rad), and δT: throttle.

However, for the sake of simplicity we can separate aircraft dynamics as longitudinal and lateral dynamics. Also some states such as positions and altitude have no role on aircraft control ; that is why, they are omitted from the dynamic model. In elevator stuck cases, we will only consider longitudinal dynamics because the elevator deflection affects mostly longitudinal states. The longitudinal states are speed,

angle of attack, pitch angle and pitch rate. The longitudinal inputs are elevator and throttle.

$$x = \begin{bmatrix} V & \alpha & \theta & q & \end{bmatrix}^T$$
$$u = \begin{bmatrix} \delta_e & \delta_T \end{bmatrix}^T \quad (2)$$

This paper is organized as follows: We begin discussing data collection, and pre-processing. We will go onto dicussing about the methodology and implementation. Finally wrapping up with the results, conclusions, and future work.

A. Data Collection

For data collection, the GTM simulation is run 46 times with elevator stuck and no stuck cases. Stuck angle and stuck time is randomly chosen. Also, ascending and descending cases with three different climb rates are added both nominal and stuck cases to make the data more diverse and challenging.

The Figure 1 illustrates 46 cases where the first 15 cases are nominal cases. These no-stuck cases includes 200 ft. ascending and descending with three different climb rate scenarios. Rest of the cases are stuck cases but some of them have no ascending and descending scenarios. We tried to create realistic and difficult situation.Climb rate 3 is the highest climb rate we can choose.

index	Sim_time [sec]	Ascend/Descent time [sec]	Stuck Time[s]	Stuck Angle [deg]	Case
1	5	None	None	None	No stuck, no ascend or descend
2	100	0	None	None	Ascend 200 feet with ascend rate 3
3	100	9.17	None	None	Ascend 200 feet with ascend rate 1
4	100	5.19	None	None	Ascend 200 feet with ascend rate 1
5	60	1.3	None	None	Ascend 200 feet with ascend rate 2
6	60	2.76	None	None	Ascend 200 feet with ascend rate 2
7	60	8.75	None	None	Ascend 200 feet with ascend rate 3
8	60	14.22	None	None	Ascend 200 feet with ascend rate 3
9	100	0	None	None	Descent 200 feet with Descent rate 3
10	100	9.17	None	None	Descent200 feet with Descent rate 1
11	100	5.19	None	None	Descent 200 feet with Descent rate 1
12	60	1.3	None	None	Descent 200 feet with Descent rate 2
13	60	2.76	None	None	Descent200 feet with Descent rate 2
14	60	8.75	None	None	Descent 200 feet with Descent rate 3
15	60	6.13	None	None	Descent 200 feet with Descent rate 3
16	40	None	0	0.47	stuck, no ascend or descend
17	40	None	2.51	0.18	stuck, no ascend or descend
18	40	None	3.49	1.73	stuck, no ascend or descend
19	40	None	5.17	2.09	stuck, no ascend or descend
20	40	None	6.59	2.56	stuck, no ascend or descend
21	40	None	4.86	2.89	stuck, no ascend or descend
22	40	None	0.28	3.17	stuck, no ascend or descend
23	40	None	1.91	3.46	stuck, no ascend or descend
24	40	None	2.04	3.74	stuck, no ascend or descend
25	40	None	0.98	3.96	stuck, no ascend or descend
26	40	None	0	-0.14	stuck, no ascend or descend
27	40	None	0.26	-1.35	stuck, no ascend or descend
28	40	None	0.49	-2.49	stuck, no ascend or descend
29	40	None	1.28	-2.83	stuck, no ascend or descend
30	40	None	4.21	-3.63	stuck, no ascend or descend
31	40	None	2.82	-4.47	stuck, no ascend or descend
32	40	None	3.64	-6.59	stuck, no ascend or descend
33	40	None	7.21	-8.01	stuck, no ascend or descend
34	40	None	6.98	-10.03	stuck, no ascend or descend
35	40	0	1.32	0.5	Ascend 200 feet with ascend rate 1
36	40	0	2.75	3.08	Ascend 200 feet with ascend rate 2
37	40	0	5.23	-1.59	Ascend 200 feet with ascend rate 2
38	40	0	2.87	-4.36	Ascend 200 feet with ascend rate 3
39	40	0	3.45	1.32	Descent 200 feet with Descent rate 1
40	40	0	7.26	3.51	Descent 200 feet with Descent rate 2
41	40	0	6.84	-8.24	Descent200 feet with Descent rate 2
42	40	0	10.23	-1.98	Descent 200 feet with Descent rate 3
43	40	0	12.59	0.92	Ascend 200 feet with ascend rate 3
44	40	0	8.84	0.8	Descent 200 feet with Descent rate 3
45	40	0	27.69	1	Ascend 200 feet with ascend rate 3
46	40	0	11.22	1.07	Descent 200 feet with Descent rate 3

Fig. 1. Flight Data Simulation for Nominal and Stuck cases

The raw data contains information from all states (1) but we will consider only longitudinal states This part will be widely discussed in the following section. Also, apart from the master data we mentioned above, we crated new 5 cases. The table below show that elevator is stuck at various angle in different scenarios. The case 2 and case 4 are expected to be more challenging because elevator is stuck at very close to nominal elevator value, $\delta_e = 0.97$

Case #	Sim [s]	Stuck [s]	Stuck [deg]	Case
1	100	30	3	Ascend 200 ft , Rate 3
2	50	25	1	Ascend 200 ft , Rate 3
3	50	20	-4	Descend 200 ft , Rate 3
4	50	20	0.9	Descend 200 ft , Rate 3
5	50	20	0.2	Ascend 200 ft , Rate 3

These 5 cases are used to implement Time Domain Approach which will be discussed later.

B. Pre-processing

Once we gathered the 46 simulations individually, we used Pandas within Python to easily aggregate and add appropriate labels in the data. GTM does not automatically add the headers for us to view which features are what so we added them as well. As mentioned above, we will use longitudinal states in this project and the rest of states are omitted from the raw data. The data contains information from air speed, angle of attack, pitch angle and pitch rate. The label is binary 0 and 1 which respectively mean no-stuck and stuck conditions.

C. Basic Approach

The goal of this project is to solve a binary classification problem. Because class information is available, a supervised learning approach is taken for this project. From this, an novel method of data manipulation is investigated. The method is a frequency-domain based approach, where the data will be converted from the time-domain into the frequency-domain. Along with this, the original time-domain data will be utilized directly to serve as a comparison for the novel method.

II. Methodology

A. Logistic Regression

Logistic Regression is a widely used classifier that is easy to implement, and will serves as a fundamental method that is used in the beginning of most machine learning projects. This model can be very powerful if there is a semi-clear difference between classifications. For many projects, even if there is a small amount of data, this model is capable of producing great results and accuracy. Once there is more data added, and more features are introduced however - this model may not do it justice. According to [7], "Logistic regression is a fundamental classification technique. It belongs to the group of linear classifiers and is somewhat similar to polynomial and linear regression . . . Logistic regression determines the best predicted weights such that the function p(x) is as close as possible to all actual responses where n is the number of observations. The process of calculating the best weights using available observations is called model training or fitting." The Formula for logistic regression is as follows, given a typical linear classifier:

$$f(x) = b_o + b_1 x_1 + ... + b_r x_r \quad (1)$$

The Logistic regression function p(x) is the sigmoid function of f(x):

$$p(x) = \frac{1}{(1 + \exp(-f(x))} \tag{2}$$

Where the sigmoid function is the predicted probability that a given output is equal to 1 [7].

B. Naive Bayes

Naive Bayes Classification is one of widely used supervised classification techniques based on Bayes' theorem. Bayes' theorem can be formulated as follows:

$$p(C_k|\mathbf{x}) = \frac{p(C_k)\ p(\mathbf{x}|C_k)}{p(\mathbf{x})}. \, and, \text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}.$$

In this project, the raw dataset was shuffled and standardized in the preprocessing step. The preprocessed dataset for the Naive Bayes Classification contains 473046 rows of data, each with four continuous-valued features(Speed(tas), angle of attack(alpha), pitch angle(theta) and pitch rate(q)) followed by a binary class label (0=not-stuck, 1=stuck). The probabilities of not stuck and stuck can be simply calculated as follows:

$$P(non\_stuck|\mathbf{features}) =$$
$$P(non\_stuck)\ P(\mathbf{Speed}|non\_stuck)*$$
$$P(\mathbf{angle\_of\_attack}|non\_stuck)*$$
$$P(\mathbf{pitch\_angle}|non\_stuck)*$$
$$P(\mathbf{pitch_r ate}|non\_stuck)$$

$$P(stuck|\mathbf{features}) =$$
$$P(stuck)\ P(\mathbf{Speed}|stuck)*$$
$$P(\mathbf{angle\_of\_attack}|stuck)*$$
$$P(\mathbf{pitch\_angle}|stuck)*$$
$$P(\mathbf{pitch\_rate}|stuck)$$

Dealing with continuous data, in this project, Gaussian naive Bayes is applied. The Gaussian naive Bayes probability distribution of v given a class C_k is as follows:

$$p(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}}\, e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$$

This algorithm is accessible in Python through the scikit-learn library. It is adopted for this research. Nave Bases Classification requires a condition that the features are independent of each other. In this project, the four features(Speed (tas), angle of attack (alpha), pitch angle (theta) and pitch rate (q)) of the dataset are independent.

C. Decision Tree

The decision tree is a supervised machine learning algorithm for building classification systems or for establishing prediction algorithms. This method composes inverted tree-like nodes with a root node (the highest Information Gain attribute), internal nodes(attributes), and leaf nodes(classifications). The algorithm can efficiently handle massive and complex datasets. In this project, one of Python classifiers, DecisionTreeClassifier(), was applied. It can do both binary and ordinal/nominal data classification. It utilizes entropy for the information gain to recursively choose the most important feature, which has the highest information gain, as nodes of the tree. The formulas for Entropy and Information Gain are as follows,

$$Entropy\ H(X) = -\sum p(X)\log p(X)$$

$$Information\ Gain\ I(X,Y) = H(X) - H(X|Y)$$

In this project, the dataset contains 473046 rows, each with four features(Speed(tas), angle of attack(alpha), pitch angle(theta), and pitch rate(q)) followed by a binary class label (0=not-stuck, 1=stuck). Because it has a binary class label, every node in the tree has two branches(0=not-stuck, 1=stuck).

D. Random Forest

Random Forest is an ensemble technique that uses both regression and classification tasks with the use of many decision trees and a certain technique called Bootstrap Aggregation which is known as bagging. The approach that random forest follows is it picks a random data point from the training set, it builds a decision tree with that data point, chooses the number of trees you want to build and repeats the first two steps. For a new data point it makes each on of your trees predict the value of Y for the data point and assigns the new data point average across all of the predicted Y values [8]. There are some times when a random forest can be a poor choice. Random forests do not necessarily train well on smaller datasets, it fails to pick on the pattern. Random forest is a predictive tool and not necessarily a tool that is descriptive. You can also use a random forest feature importance graph that shows variable importance but it may not be sufficient enough to describe the relationship between the variables. The advantages of using a random forest algorithm are that since it involves the use of many decision trees the bias remains the same of one decision tree [9].

E. Neural Network

We implemented a neural network using Keras which runs on Tensorflow. The model we used was Sequential which is the default model but it's one to start on with. Neural networks is a popular type of machine learning subset that many use for supervised learning. "Keras is a high-level neural networks API, written in Python and capable of running on top of Tensorflow, CNTK, or Theano . . . The core data structure of Keras is a model, a way to organize layers. The simplest type of model is the Sequential model, a linear stack of layers.

For more complex architectures, you should use the Keras functional API, which allows you to build arbitrary graphs of layers" [6]. The algorithm for neural network can use either Stochastic Gradient Descent, Adam, or L-BFGS. Stochastic Gradient Descent or (SGD) updates parameters using the gradient of the loss function with respect to a parameter that needs adaption.

III. Implementation

A. Frequency Domain Transformation Overview

Digital signal processing (DSP) is a field within engineering dealing with the processing, modification, analysis, and study of signals. There are a wide variety of techniques used to characterize signals, a hallmark being the Fourier analysis [11]. In this project, the given data sets are made up of a series of simulations, consisting of several discrete time-domain signals (the feature data). With the aircraft performing similar maneuvers (flying level, climbing, descending, getting stuck) between each simulation/flight, the signals may exhibit some repeated pattern. Identifying such a pattern would be essential in understanding what sort of maneuver the aircraft is performing. Between every simulation, patterns in the signals may repeat themselves; the frequency of this pattern (the amount of times the pattern repeats itself) may be a useful identifier that could potentially aid in identifying what the aircraft is doing.

The main mathematical mechanism of Fourier analysis is that discrete time-domain data can be decomposed into its periodic components. A simple example of this would be a musical note can be broken down into its volume (amplitude) and harmonics (the various frequencies that make up that note) [11]. What the Fourier transform gives, is the frequencies that make up the given signal, as well as their amplitude - or how present they are in the signal. Moreover, this would make it easier to identify patterns in the data between the simulations. An efficient algorithm that does this transformation is the Fast Fourier Transform (FFT). In brief, this algorithm can decompose a given discrete time-domain signal into an array of frequencies [12]. This algorithm is available in Python through the use of the Scipy module. Figure 2 depicts a sample set of data from a simulation trial, and it's frequency domain counterpart can be seen in Figure 3.

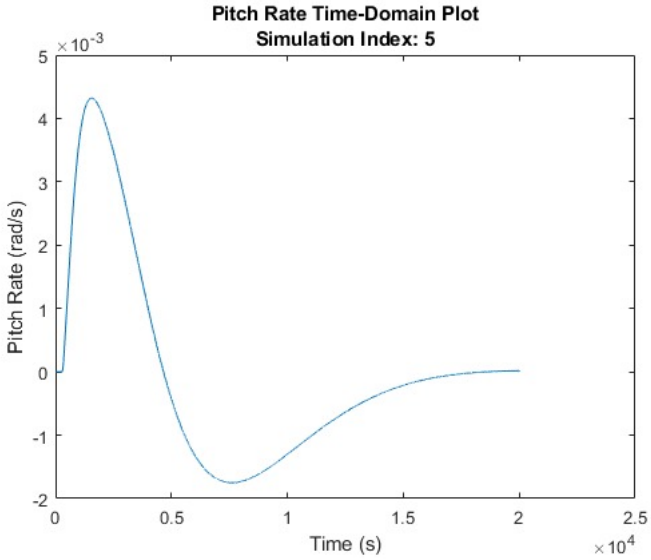

Fig. 2. Pitch-Rate Time-Domain Sample Plot

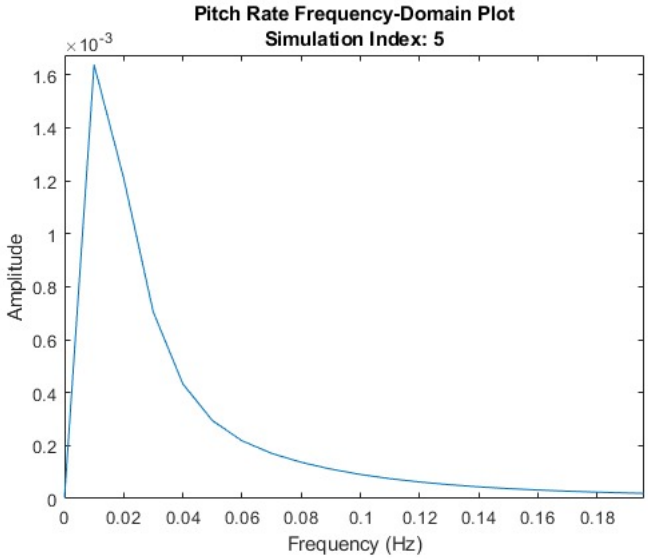

Fig. 3. Pitch-Rate Frequency-Domain Sample Plot

Referring to Figures 2 and 3, the time-domain plot represents the aircraft's pitch-rate, as it changes while the aircraft descends in altitude. In the frequency domain plot, the x-axis represents the various frequencies that compose this signal, the y-axis represents the amplitude, or how much of that frequency is present in the signal. The peaks from the frequency-domain plot is of particular interest, as they show the strongest patterns within a given feature. The main idea of this frequency-domain based classification is to extract these unique peaks from the data set, and use them as a new set of features for classification.

In order to do this successfully, careful consideration must be taken into account regarding the shape of the data set itself. When performing the FFT, the time-domain labels of the

data are no longer linked to their respective features - as new features are developed (the peak data). In order to resolve this, the data is grouped by simulation number, and that simulation number is then associated with a class label (Stuck or Not Stuck). By grouping the time-domain data by simulation number, managing the class labels is easier by utilizing a Python dictionary that matches simulation number to class label. After the data has been grouped and the dictionary has been formed, the simulation numbers are randomly shuffled then split into train and test sets, and then standardized. The data is moved based on simulation number into their new respective buckets. FFT is performed on all of the features for the training set. The new data formed from this operation has the same number of features, but the associated data are now frequency-domain values. Each data point is associated with a simulation number, which can be matched up to a class label using the dictionary object. The FFT spectrum for an entire feature can be quite large, and the only relevant information would be the large peaks in the data. Peak extraction is utilized to pick out the most important peaks from a given feature, and this aids in forming a new feature set that can be used for classification.

The main goal of peak extraction is to find large peaks of a given feature, and keep a record of the corresponding frequency, as well as that data point's associated class label. Figure 4 depicts the FFT plot of pitch-rate of the entire set of simulations:

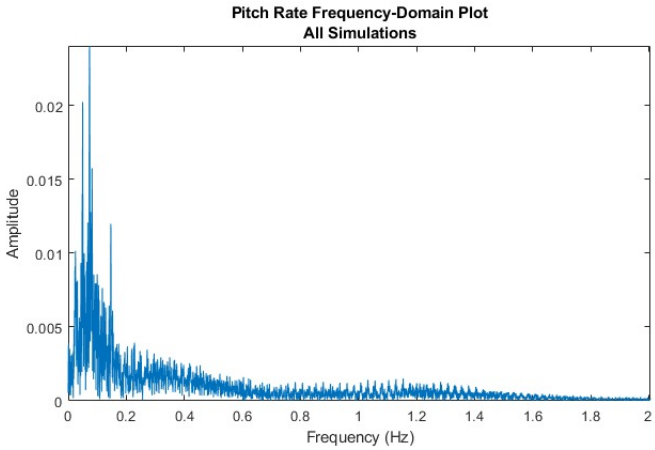

Fig. 4. Pitch-Rate Frequency-Domain Plot, All Simulations

There are many peaks in this plot, and in order to simplify the data - the first N highest peaks are used to construct the new simplified feature set. A threshold is set for minimum peak height, and the N highest peaks above this threshold are collected. In this project, 5 peaks were collected, as prior related work found that this would be an appropriate starting value [12]. The threshold for minimum peak height (mph) was calculated as such:

$$mph = signal_{min} + \frac{(signal_{max} - signal_{min})}{d} \quad (3)$$

Where d is a scaling parameter to set the threshold of the peak height, as per previous related work this value was set to 10 [12]. After this routine is completed, the newly constructed data follows the following format:

Simulation Number X	Feature 1, Peak 1	Feature 1, Peak 2	Feature 1, Peak 3	Feature 1, Peak 4	Feature 1, Peak 5	Class Label X
	Feature 2, Peak 1	Feature 2, Peak 2	Feature 2, Peak 3	Feature 2, Peak 4	Feature 2, Peak 5	
	...					

Fig. 5. Peak Extraction New Feature Structure

The newly structured data is now ready for training a classifier. The test data is run under the same series of operations as well.

IV. Experiments and Results

A. *Results with Fast Fourier Transform*

Three classifiers were trained and tested using the new FFT feature data. The first experiment conducted was logistic regression. Logistic Regression did not fair well, with an overall accuracy of 0.35. In the next section C, Receiver Operating Characteristic (ROC) curves and confusion matrices are developed to assess the performance of the various classifiers. The following tables provide numerical performance metrics:

Logistic Regression	
Precision	0.3333
Recall	0.2857
F-Measure	0.3076
Accuracy	0.3571
True Positive	2
False Negative	5
False Positive	4
True Negative	3

Random Forest	
Precision	0.6666
Recall	0.5714
F-Measure	0.6154
Accuracy	0.6428
True Positive	4
False Negative	3
False Positive	2
True Negative	5

Neural Network	
Precision	0.7142
Recall	0.8571
F-Measure	0.7999
Accuracy	0.7857
True Positive	6
False Negative	1
False Positive	2
True Negative	5

B. *Results Time Domain Approach (without FFT)*

Although the data initially looked as if it was time-domain dependent, when we trained and tested various supervised learning models - we learned that this may not be the case. Randomly shuffling and splitting the data does not impact the classification of the data. This can be seen through the tabulated performance metrics for the varying ML classifiers we trained and tested with. Multiple algorithms perform well with the longitudinal dynamic states.

Among these algorithms, the decision tree classifier performed the best in terms of speed and accuracy. The logistic regression accuracy on test data 0.78, while Naive Bayes gives 0.94 accuracy. Decision Tree and Random forest accuracy is almost 1 when running it through with test data. Section C highlights specific performance metrics found by analyzing ROC and confusion matrices for each of the models that were generated.

Logistic Regression	
Precision	1.0
Recall	0.5109
F-Measure	0.6763
Accuracy	0.7763
True Positive	36849
False Negative	35264
False Positive	0
True Negative	85569

Naive Bayes	
Precision	0.9988
Recall	0.8692
F-Measure	0.9295
Accuracy	0.9397
True Positive	62682
False Negative	9431
False Positive	70
True Negative	85499

Decision Tree	
Precision	0.9999
Recall	0.9993
F-Measure	0.9996
Accuracy	0.9996
True Positive	72066
False Negative	47
False Positive	5
True Negative	85564

Random Forest	
Precision	0.9998
Recall	0.9994
F-Measure	0.9996
Accuracy	0.9996
True Positive	72071
False Negative	42
False Positive	8
True Negative	85561

C. *ROC Curve Graphs for Fast-Fourier Transform Results*

Receiver Operating Characteristic or (ROC) graphs are metrics to evaluate classifier output quality. We implemented this metric to further gain insight into how our graphs compared against one another. ROC curves contain True Positive rate on the Y-axis and False Positive rate on the X-axis. The placement of axis indicates that the top left of the corner on the plot is the ”ideal” point - a false positive rate of zero, and a true positive rate of one. The area covered by the curve is the area between the orange line and the axis. The area that is covered between the orange line and the diagonal line is otherwise known as the Area Under the Curve or (AUC). The bigger the area covered the better the machine learning model is at distinguishing the given classes. There are several scenarios for (ROC) graphs.

1. Best - The best case is when there is a clear difference between the two classes. For a best case scenario if the orange line follows an upward right angle to the left side of the square this is a clear indicator that the model is predicting the correct probability of a class.
2. Random- In the event of a completely random case this is indicated when the orange line follows the diagonal blue line to a T. This is not necessarily the worst model but it does make a random guess.
3. Worst- A model is considered a worst case when it miss-classifies the classes. Completely opposite to the best case scenario, this means the polar opposite of a right angle on the left upper side. The worst case would mean a right angle on the right lower side of the graph which results in a 0 in the AUC [10].

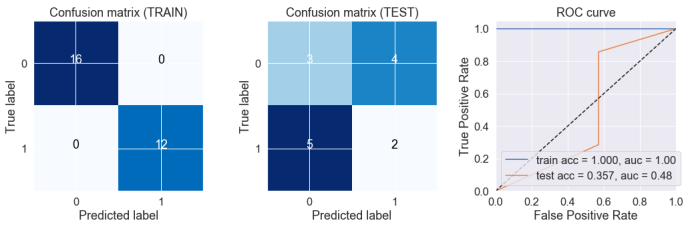

Fig. 6. Logistic Regression

Referring to 6, the ROC metrics showcase rather erratic changes in the test accuracy. Training accuracy is perfect at 1.0. There may be indication of overfitting. Which is logical in the sense that when the FFT transformation took place, the data and feature set for training shrank.

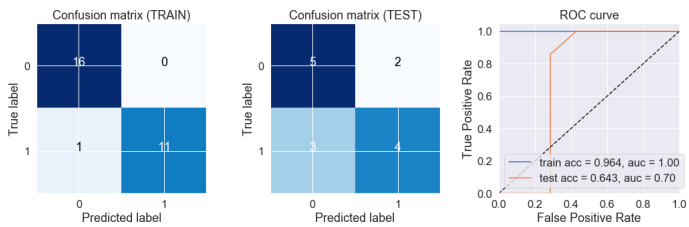

Fig. 7. Random Forest

Referring to 7, the ROC metrics show a sudden spike that holds a better accuracy compared to the training set of the

logistic regression, but a normal ROC curve is smooth.

D. ROC Curve Graphs for Results Time Domain Approach

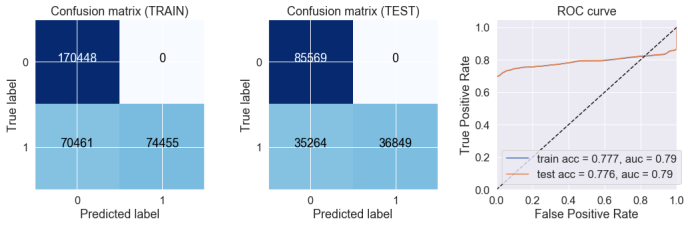

Fig. 8. Logistic Regression

Referring to 8, Logistic Regression ROC metric indicates somewhat of a curve that is straight, with an accuracy of 0.776 this has been a good improvement compared to a frequency domain approach. The training accuracy is also at the same accuracy score indicating a good fit.

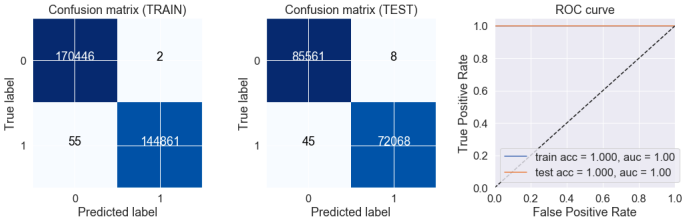

Fig. 9. Random Forest

Random Forest ROC metric shows very accurate results in predictions in both training and testing sets. This algorithm is deemed one of the best case scenarios for ROC curve.

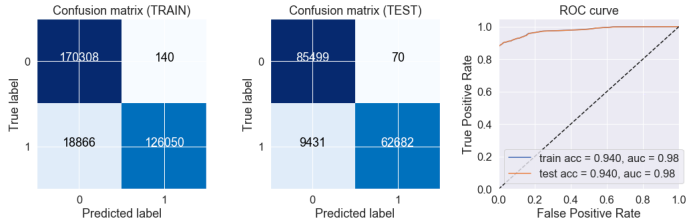

Fig. 10. Naive Bayes

Referring to 10, Naive Bayes ROC metric is another close to the ceiling graph that is a good indicator of a strong model. This algorithm is third best right behind Random Forest and Decision Trees.

Refering to 11, Decision Tree ROC metric is the same as Random Forest metric in that it has perfect scores in both training and testing sets.

V. Conclusions

Moreover, a total of 46 simulations were ran using a non-linear high fidelity GTM Simulink model. Only the longitudinal states of the aircraft were considered as features,

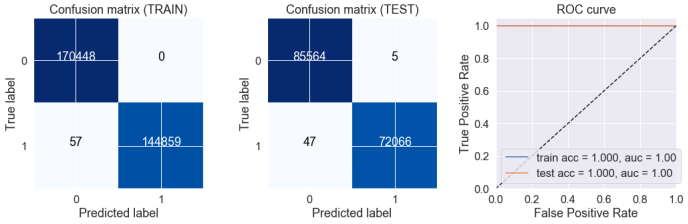

Fig. 11. Naive Bayes

and it can be seen that these are enough to be used in the investigation of elevator failure detection problems. Two different methodologies were considered: a frequency domain approach, and a time domain approach. Both implemented multiple machine learning supervised learning classification algorithms. The metrics from the frequency domain approach proved to be impractical in terms of performance. It can be seen that larger data sets would be needed in order for the frequency domain approach to work properly, as the operation of moving the data into the frequency domain can be seen as a feature and data reduction. This may not be necessarily bad, as large data sets can also be impractical to handle. So there is a trade-off when attempting to perform a frequency-based classification. In addition to this, it should be noted that the frequency domain method does not provide relevant time information to when the elevator jam occurred; which would be vital in real life applications.

Whereas, utilizing the time domain data directly, it can be seen from the performance metrics that this works well. The decision tree performed well with the test data, as well as the additional new 5 sets of test data as well. The project was completed successfully with the algorithm detecting an elevator jam within less than 0.3 seconds in all of the new 5 test cases. Although accuracy can be low in some cases where elevator jam happens too close to the nominal elevator value, this may be a great start in the application of modern machine learning techniques in the classical field of aircraft flight controls.

VI. Future Works / Extensions

In future endeavours, it would be relevant to try to package one of the successful models into some real-time flight software. This will require careful consideration however. One main issue that was not covered in the scope of this project was the presence of noise in the signals. On most UAVs, the feature data will appear heavily noisy. Modifying the existing GTM model to incorporate simulated noise may be necessary to see how feasible it would be to use the existing generated models on a real flight test. In this case, the Fourier analysis route may be useful as noise can be easily be identified and filtered out as the noise will propagate at it's own unique frequency [11]. Careful consideration will also need to be taken for verifying the speed of the detection, as an elevator jam needs to be caught quickly to prevent fatal accidents. Another thing to consider for a future project would be to verify false positives

with the real-time version of the algorithm, to ensure the emergency flight control system does not kick in when the aircraft is already flying nominally.

REFERENCES

[1] GTM Design Sim
http://github.com/nasa/GTM_DesignSim

[2] B. Chang, M. Bayram, H. Kwatny and C. M. Belcastro. *2016 IEEE Conference on Control Applications (CCA), Buenos Aires*. [*Flight path and altitude tracking control of an impaired nonlinear generic transport model (GTM) aircraft with elevator jam failures*]. 2016, pp. 15-20.

[3] Chan, P. C., Chang, B. C., Bayram, M., Kwatny, H., Belcastro, C. M. *Asian Journal of Control*. [*Robust tracking control of an aircraft with critical actuator jam failures*]. 2016, pp. 15-20.

[4] Aviation Statistics,
https://www.ntsb.gov/investigations/data/Pages/aviation_stats.aspx

[5] NTSB/AAR-02/01
urlhttpshttps://www.ntsb.gov/investigations/AccidentReports/Reports/AAR0201.pdf

[6] Keras: The Python Deep Learning library,
http://keras.io/

[7] Logistic Regression in Python,
https://realpython.com/logistic-regression-python/what-is-classification

[8] Random Forest Regression in Python,
https://www.geeksforgeeks.org/random-forest-regression-in-python/

[9] Random forests explained intuitively,
https://www.kdnuggets.com/2019/01/random-forests-explained-intuitively.html/

[10] Understanding ROC Curves with Python,
https://stackabuse.com/understanding-roc-curves-with-python/

[11] Obeid, Iyad. [*Signals: Continous and Discrete*]. 2013, pp. 35-50.

[12] Machine Learning with Signal Processing Techniques,
urlhttpshttp://ataspinar.com/2018/04/04/machine-learning-with-signal-processing-techniques/