SSC24-WV-06

Applying Machine Learning to Equatorial Plasma Bubble Now Casting on CubeSats

Benjamin Lewis, Shawn Jones, Charles Swenson, Kevin Moon, Mario Harper
Utah State University
[Address (same for all)]; 435-557-1682
benjamin.lewis@usu.edu

ABSTRACT

Equatorial plasma bubbles are a space weather phenomenon that occur at low latitudes within the Earth's ionosphere. These bubbles are regions of low-density plasma that form at the base of the ionosphere and expand upward through the peak and into the topside. They form in the early evening and persist through the nighttime, stretching north and south along magnetic field lines and effecting a sector of longitudes a few degrees wide. These bubbles cause scintillations on radio signals that pass through them, disrupting the performance of systems, such as GPS, throughout the night in regions around the Earth. Due to their potential social impact, there is a desire to know in real time if equatorial plasma bubbles are occurring. This can be achieved using a small satellite in orbit with automated processing to locate bubbles by processing data from in-situ sensors. Inter-satellite communications to a LEO communications constellation allows the possibility of real time detection of equatorial plasma bubbles from such a satellite. Langmuir probes, impedance probes, and ion drift meters are all instruments capable of detecting plasma bubbles. Sensors such as the Scintillation Prediction Observations Research Task (SPORT) Brazil/US CubeSat mission routinely detect equatorial plasma bubbles. This work investigates the application of various machine learning techniques for the detection of plasma bubbles on a CubeSat using time series plasma density data from the SPORT instrument. This paper reviews four machine learning approaches: Auto-regressive Integrated Moving Average (ARIMA) models, Random Forests, Gradient Boosting Machines, and Recurrent Neural Networks (RNN) such as Long Short-Term Memory (LSTM) networks. The models are evaluated using common metrics derived from the confusion matrix (e.g. accuracy, sensitivity, positive predictive value, and F1 score), as well as the computational complexity of the model. Building off of this work, USU will implement a plasma bubble detection algorithm on a low-power edge computing device built for a CubeSat to provide near real-time plasma bubble reporting. USU's Low-power Array for CubeSat Edge Computing Architecture, Algorithms and Applications (LACE-C3A) project is being funded by NASA's University SmallSat Technology Partnership Program to develop an FPGA-based compute board that will be capable of performing machine learning algorithms on a CubeSat scale. The near real-time plasma bubble reporting will be one of the applications USU will demonstrate on LACE-C3A.

1 Introduction

Space weather phenomena affect the performance of space based communication and navigation systems. In particular, equatorial plasma bubbles are associated with scintillation on RF signals pass through the ionosphere where bubbles are present. These bubbles are depletions in the density of the plasma that propagates from the bottom edge through the layers of the ionosphere. They form near the Earth's equator in the evening after sunset and their trails persist till after midnight. The sheet like bubbles rapidly rise near the Earth's magnetic equator being a tens to hundreds of km wide in the east-westward directions but expanding 1000's of km north-southward along the magnetic field. Instruments, such as Space Weather Probes (SWP) on the joint Brazil-NASA Scintillation Prediction Observations Research Task (SPORT) mission provide in-situ measurements of ionospheric density while crossing these bubbles. Within these along track satellite observations the bubbles are primarily identified as sudden and significant drops in density of an order of magnitude or more. SPORT, a 6U CubeSat, was deployed into a mid inclination orbit at 51.6° from the international space station at 420 km on December 29, 2023 and reentered on October 10, 2024. The data set from June to October primarily covers observations of equatorial plasma bubbles at an altitude range between 280 to 380 km.

In this paper, we explore several machine learning approaches to identify plasma bubbles from

the SPORT density data. These approaches include Auto-Regressive Integrated Moving Average (ARIMA), random forests, gradient boosting machines, and Long Short-Term Memory machines (LSTM). The goal of this project is to find a machine learning model to operate in real time on spacecraft data from an instrument like SWP on SPORT. The spacecraft can then use this model to perform edge computing on the SWP data and detect plasma bubbles on the spacecraft. This can then be used to provide near real time detection of regions around the Earth where plasma bubbles are occurring. Post-flight, the models can be used for labeling data sections with bubbles to be sent down with high priority. This can reduce the amount of data down-linked to the ground.

2 Background

Plasma bubble formation has been widely studied. Yokoyama[1] and Hysell[2] constructed simulations that show the formation and propagation of plasma bubbles. Figure 1 from the Yokoyama paper[1] shows a simulation of plasma bubbles forming. The low density plasma pushes upwards through the layers of the ionosphere until they reach the peak of the ionosphere. There, the bubbles bifurcate into smaller structures as they propagate into the topside of the ionosphere. Plasma bubbles matching these models have been observed by the ROCSAT[3] and CHAMP[4] missions as well as the SPORT mission.

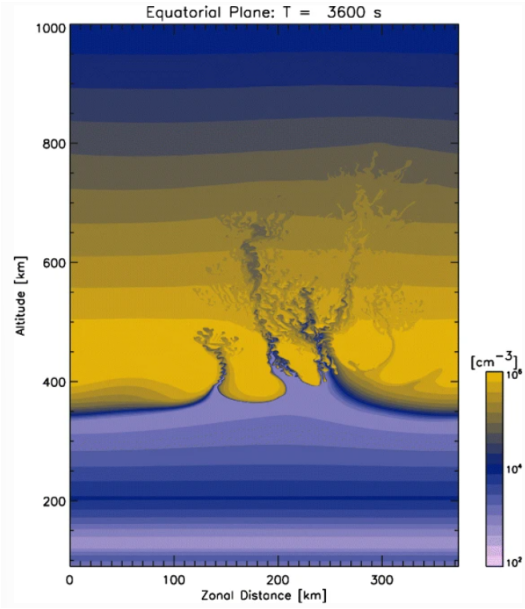

Figure 1: Simulation of Plasma Bubbles Forming[1]

There has been recent research on using machine learning to predict space weather effects.[5] Much of this research concerns predicting space weather indices, such as Kp, and Dst which are large scale measurements of the fluctuation in the Earth's magnetic field. Machine learning techniques were used to classify plasma bubbles by Thanakulketsarat,[6] Chandan,[7] and Reddy.[8] Thanakulketsarat used very high frequency radar images to detect plasma bubbles and built a convolutional neural network (CNN) and support vector machine (SVM) to classify plasma bubbles. Chandan used a random forest to classify bubbles using total electron count. Reddy used an extreme gradient boosting (XGBoost) model to predict the Ionospheric Bubble Index (IBI) using data from the Swarm spacecraft in a regression problem. Thanakulketsarat and Chandan were able to successfully classify different classes of plasma bubbles with accuracies of 93.08% and 97.6% respectively. Reddy was able to predict the IBI value with a root mean square error of 0.08. Thanakulketsarat faced a data imbalance issue that they remedied by reducing the dataset by throwing out data that does not have bubbles.

3 Data

The data we used for this project is electron density data collected from the Langmuir probe onboard the SPORT mission. The Langmuir probe consists of a needle-like probe that extends from the spacecraft. To measure the density the surface of the probe is held at a constant voltage and the current collected by the probe is measured at a 100Hz sample rate. This current can be calibrated into the electron density of the ionosphere. For this paper, we use the uncalibrated data as this is the data that will be available to the model when implemented on a CubeSat.

3.1 Cleaning the Data

The SPORT density data contains 391 sections of continuous data. These section represent a pass through the equatorial region. However, there are glitches in the data, data drops, and dropped packets. To clean these, we add in missing timestamps from the dropped packets and mark these samples as NaN. We then use a spike removal method where we have a moving window of 500 samples, and we calculate the standard deviation of the samples in the window. If a sample in the window is 3σ away from the mean, the sample is replaced by a NaN. We then use linear interpolation to fill the missing

samples and create a continuous section of data.

There is also noise in the data, specifically, a near-constant interference from another instrument. The spacecraft occasionally rotates in a way that accentuates the noise. We marked the sections that have this noise so that we can compare performance on the noisy samples. We did not remove these noise samples for two reasons. The first is that the samples in a bubble have a lot of high frequency data, and filtering out the noise could remove valuable information in the bubbles. The second is that the final product of this project is that we want to implement this on a satellite using raw data from an instrument similar to what was used on the SPORT mission, and we want to keep the data as similar as possible.

After cleaning the data, we were left with 360 sections of data, averaging 168, 969 samples per section, for a total of 60, 828, 851 samples. We then label the data. We had a scientist working with the SPORT data look through all of the sections of data and locate bubble regions in the data. All samples within a bubble region are labelled as class 1, while all other samples are labelled as class 0. After labelling, the data has 57, 623, 338 samples in class 0 (no bubble) and 3, 205, 513 samples in class 1 (bubble). A sample section of the data, including a labelled bubble section is shown in Figure 2.

The machine learning problem we propose is a binary classification problem. This could also be viewed as an anomaly detection problem, as there is a large class imbalance. Class 0 is the "normal" case, and class 1 is the anomaly.

3.2 Data Imbalance

After inspecting our data we found that approximately 5.27% of our data represents class 1 data. (class 1 is a positive detection of plasma bubbles.) Because of this large class imbalance, we need to find a way to create more balanced training and testing data sets. The fist thing we do is reduce the data set to only include sections that have at least one bubble. This reduces the data to 154 sections for a total of 26,534,472 samples, with 23,332,141 samples in class 0 and 3,202,331 samples in class 1. Approximately 12.07% of the reduced data is class 1.

To further remedy the class imbalance, we use two different approaches. The first is sample weighting, which weights each sample according to its class:

$$class\_weight = 1 - \frac{samples\_in\_class}{total\_samples}. \quad (1)$$

This will weight class 1 samples more than class 0 samples to help overcome the data imbalance.

Our second method is to generate synthetic data to augment the full dataset. In order to create synthetic density profiles (data) we used a three step approach. Step one, we learn the distributions of the delta density on give time steps for both class one and class two. Step two, we ran a Monte Carlo simulation to create new density profiles. Then smooth the data and combined the class one and class zero data. Finally, in step three, we learned the interference and noise patterns for our data and the added the patterns into the synthetic pass to make a more realistic density profile. The following notation will be used in this section to explain this approach:

Y = input labels
$\widetilde{Y}$ = Generated data labels
X = input data (density value from SPORT data)
$\widetilde{X}$ = Generated density
T = sampled time steps
d = Threshold for class transition
A = weights for each time steps
$x_{0|i}$ = new data point for the negative class
$\widetilde{T}$ = the set of sample points
$x_{1|i}$ = new data point for the positive class in each time divided region
$W_0 = a \forall t \in T$
$W_1 = a \forall t \in T$

3.3 Step One: Thompson Sampling to Learn Distributions

Thompson Sampling requires a threshold of success. In our case we look at the difference between two density samples, or delta densities, at given time steps (T). If the delta density is positive it counts as a success. We also split the data by class. By doing so we can isolate the independent density profiles. Thus the Thompson distributions are built as follows, where α is the success count and β is the

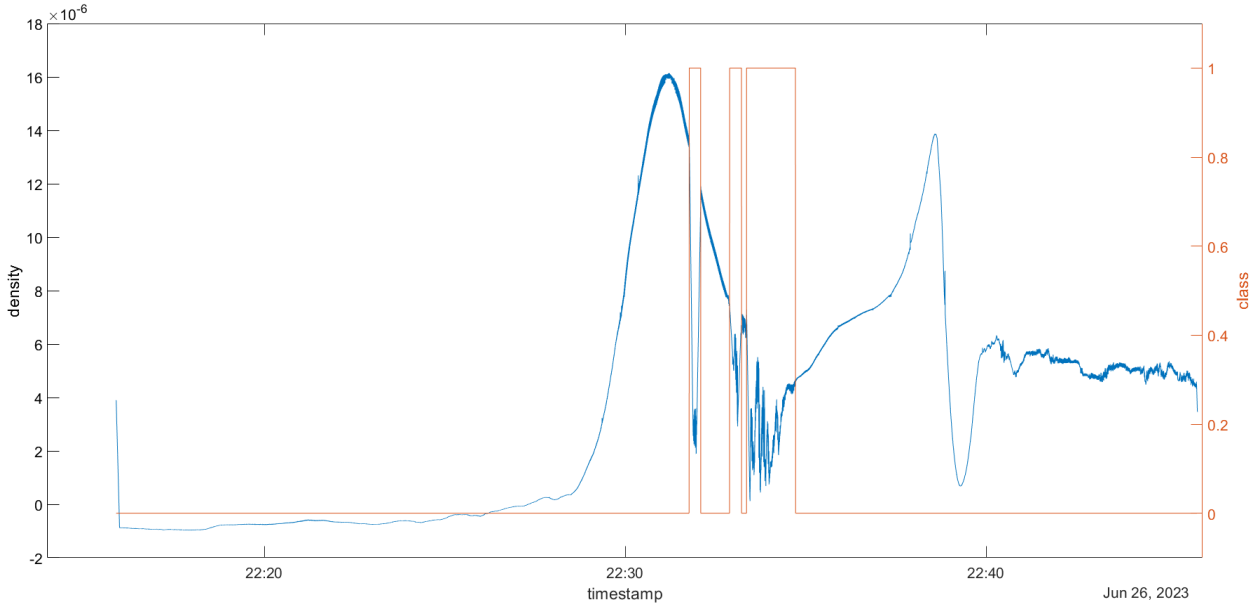

Figure 2: Sample Section of Data

failure count:

$$T = [1, 2, 3, 4, 5](minutes)$$

$$\alpha_{0|t,j} = \sum_{i}^{Y_j:i=0} \begin{cases} 1 & \text{if } \Delta(X_i - X_{i+t}) > 0 \\ 0 & \text{else} \end{cases}$$

$$\beta_{0|t,j} = \sum_{i}^{Y_j:i=0} \begin{cases} 1 & \text{if } \Delta(X_i - X_{i+t}) <= 0 \\ 0 & \text{else} \end{cases}$$

$$\alpha_{1|t,j} = \sum_{i}^{Y_j:i=1} \begin{cases} 1 & \text{if } \Delta(X_i - X_{i+t}) > 0 \\ 0 & \text{else} \end{cases}$$

$$\beta_{1|t,j} = \sum_{i}^{Y_j:i=1} \begin{cases} 1 & \text{if } \Delta(X_i - X_{i+t}) <= 0 \\ 0 & \text{else} \end{cases}.$$

We noticed that the data often has regions where it follows different distributions. So we decided to divide the data into 'time regions' ($\widetilde{T}$), where we build separate distributions for each subdivided region. Thus we build distributions for each time step from point $x(T)$ and then we also divided the larger data set by class, and time from when the pass began collecting data ($\widetilde{T}$). In our case $\widetilde{T}$ is defined as follows:

$$\widetilde{T} = [5, 10, 15, 20, 25, 30](minutes).$$

3.4 Step two: Monte Carlo Simulation

At this point we have learned distributions for how the density should increase or decrease for given time steps for each of the larger time regions. We have done this for both class 0 and class 1. With this, we have almost everything needed for a Monte Carlo simulation. We now need to calculate the average delta density. Doing this will allow us to center the learned distribution, then scale them by the average. We find the average delta density using the following formulas:

$$\mu_{0|t,j} = \frac{1}{n_0} \sum_{i}^{Y:i=0} \Delta(X_i - X_{i+1})$$

$$\mu_{1|t,j} = \frac{1}{n_1} \sum_{i}^{Y:i=1} \Delta(X_i - X_{i+1}).$$

With this information we can run a Monte Carlo simulation. For each new point we will pull one of our Thomas distribution sets depending on if we are going to assign class 0 or class 1 to the point:

$$\widetilde{Y} = \forall j \epsilon \widetilde{T} \forall i \in j \begin{cases} y_i = 0 \text{ if (random float} < d) \\ y_i = 1 \text{ else} . \end{cases} \tag{2}$$

Thus we will be generating a set of new points based on desired sample rate for our data. This is what is meant by $\forall i \in j$ in Equation 2. Each new $\widetilde{X}$ will be built as a standard Monte Carlo simulation. Thus:

$$\widetilde{X}_{i+1} = \delta \widetilde{X}_{y_i|j} * x_i, \tag{3}$$

where

$$\delta x_{0|j} = \sum_{i}^{T_j} w_{i0}\left(\mathcal{B}_{0|i,j}(\alpha_{0|i,j},\beta_{0|i,j}) - \frac{\alpha_{0|i,j}}{\alpha_{0|i,j}+\beta_{0|i,j}}\right)\mu_{0|i,j}$$

$$\delta x_{1|j} = \sum_{i}^{T_j} w_{i1}\left(\mathcal{B}_{1|i,j}(\alpha_{1|i,j},\beta_{1|i,j}) - \frac{\alpha_{1|i,j}}{\alpha_{1|i,j}+\beta_{1|i,j}}\right)\mu_{1|i,j}.$$

We center the Beta distribution at zero where we set our threshold, and then we multiply by the average delta density to then get a new time set density. We repeat this for each distribution corresponding to our current time region. In our case we have five distributions for each time region. We also provided a parameter W, which is the weights for each time step. We have defined W such that it sums to 1. Thus the new density can be built as follows:

$$\widetilde{X} = \forall j \epsilon \widetilde{T} \forall i \epsilon j \begin{cases} x_i = \delta x_{0|j} * x_{i-1} \text{ if } (y_i == 0) \\ x_i = \delta x_{1|j} * x_{i-1} \text{else} \end{cases} . \tag{4}$$

Note: j denotes a larger time region, and i denotes each time step distribution we learned in *Step One*.

After producing our Monte Carlo random walk, we smooth the class 0 data to get a smooth transition on our base density profile. We do this using a simple low-pass filter. Then we shift the class 1 data down to make sure that our generated "bubbles" match the overall trend of class 0. Then finally we have our new data set with labels, density, and times.

3.5 Results of data Generation (Step One Step Two)

We were able to generate approximately 4.5 gigabytes of data, about 600 sections of continuous data and approximately $10,800,000$ samples. The class balance in this data is approximately 40 percent class 1. The class balance is a tunable parameter that can be adjusted to increase the class balance if needed. A generated pass is compared with an actual pass that has no bubbles in Figures 3 and 4. Generated and actual passes with varying amounts of bubbles are shown in Figures 5 - 8.

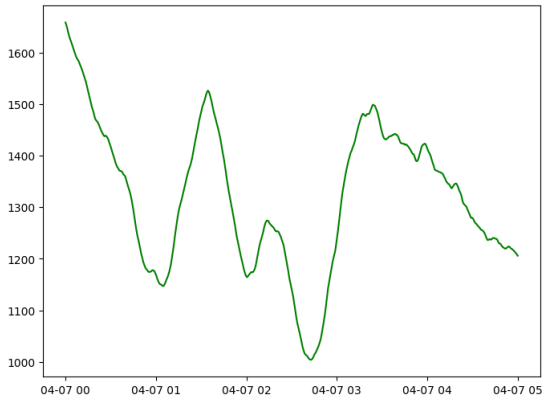

Figure 3: Generated Pass with no Bubbles

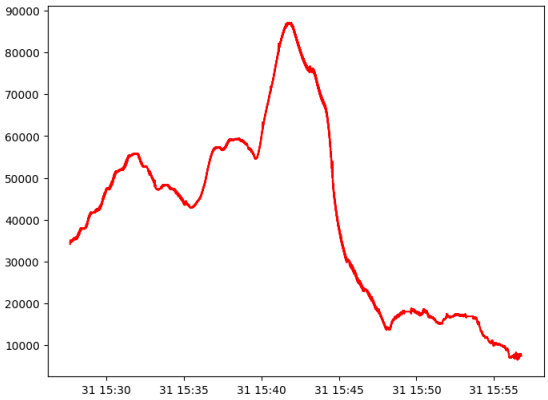

Figure 4: Actual Pass with no Bubbles

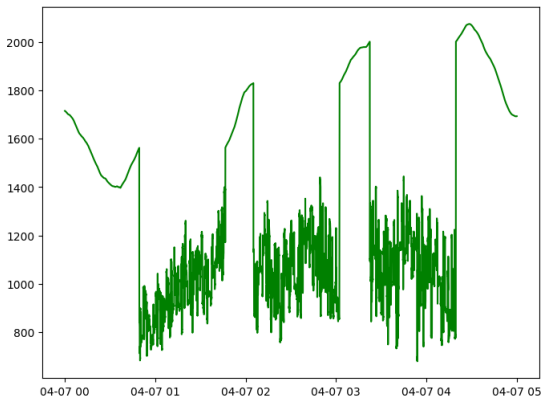

Figure 5: Generated Pass with Few Large Bubbles

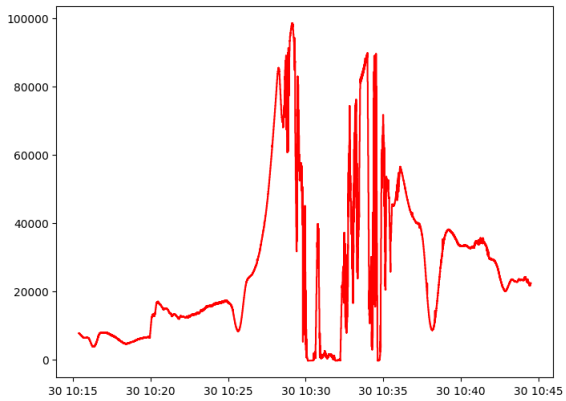

Figure 6: Actual Pass with Few Large Bubbles

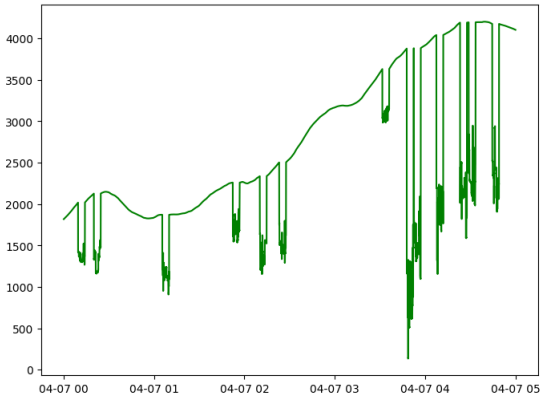

Figure 7: Generated Pass with Many Small Bubbles

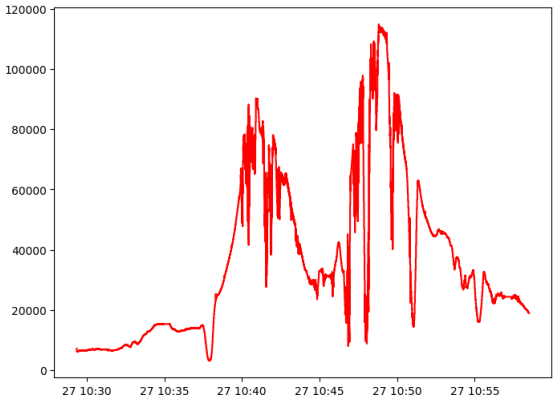

Figure 8: Actual Pass with Many Small Bubbles

3.6 Step Three: Learning Interference Patterns

The synthetic data reflects trends and profiles of the actual data. However, in the actual data there are interference patterns present. We want to add these interference patterns to the synthetic data to better simulate the patterns in the actual data. This makes it significantly harder to classify bubbles and thus increases the robustness of our classification algorithms. The first step to learning the interference pattern is to de-trend the Sport data to isolate the pattern. This was done by using a high pass filter on pass that did not have bubbles present. Examples of the de-trended data from various section are shown in Figures 9 - 11.

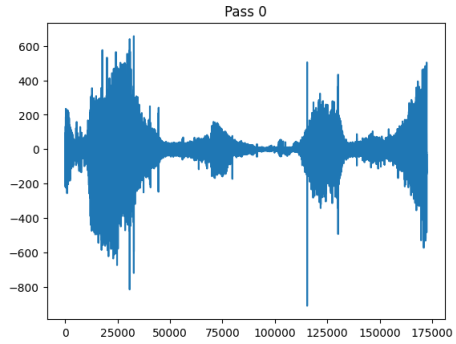

Figure 9: Example of De-trended Data

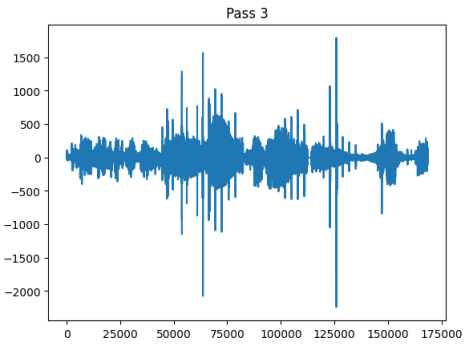

Figure 10: Example of De-trended Data

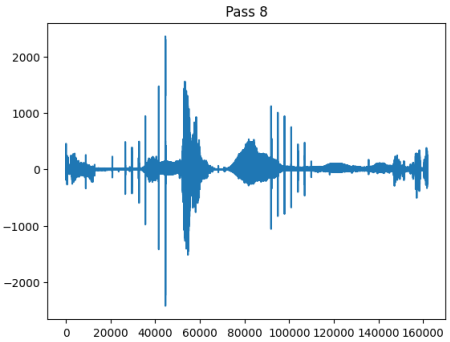

Figure 11: Example of De-trended Data

After the data was de-trended, we used an ARIMA model to try and learn the interference pattern in the data. The ARIMA model was trained by feeding it all the de-trended samples.[9] An example of an interference pattern generated by the ARIMA model is shown in figure 12.

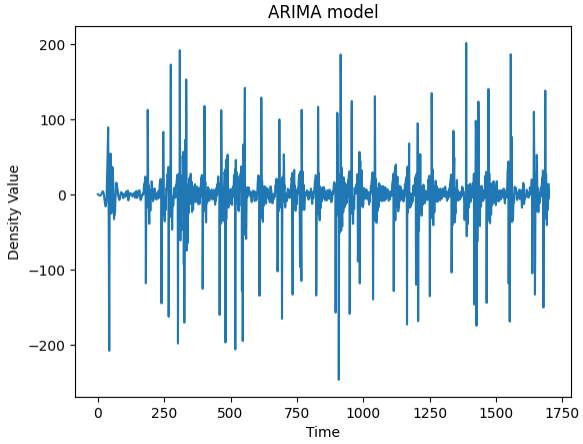

Figure 12: Output Pattern of the ARIMA Model

Finally, we combined the results of the ARIMA Model (*Step Three*) for interference with the results for the Thomson Sampling with a Monte Carlo simulation (*Step One* and *Step Two*). Resulting in synthetic data that had the trends of plasma density both with bubbles and with out bubbles, while also including realistic interface partners to maintain robustness our or classification algorithms. The following figures 13 - 15 show the resulting density profiles generated with this combined approach.

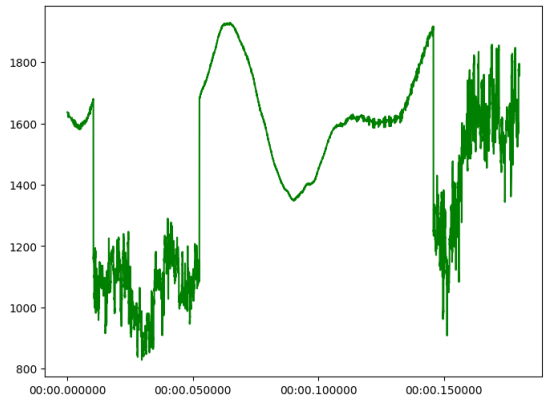

Figure 13: Generated with an Even Class Distribution

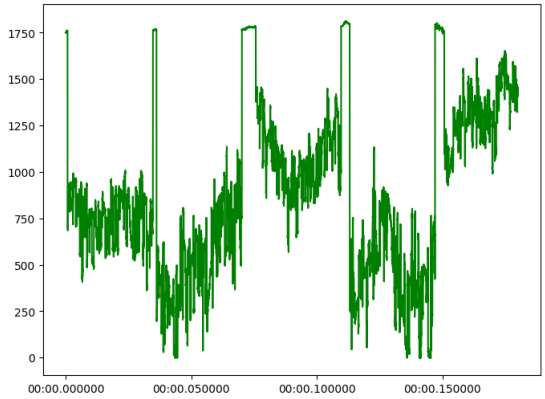

Figure 14: Generated with an Class Distribution Skewed to Class 1

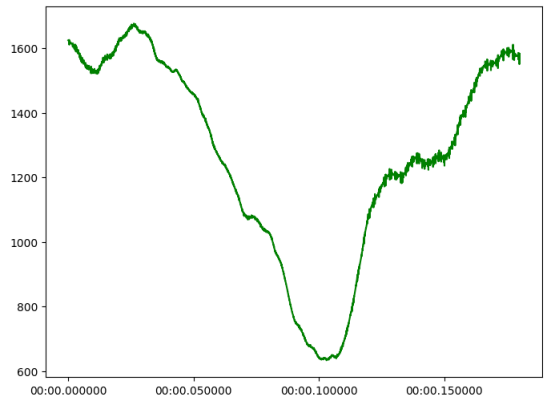

Figure 15: Generated with an Class Distribution Skewed to Class 0

4 Machine Learning approaches

To perform classification on the data, we tested four different machine learning approaches, ARIMA, Random Forests, XGBoost – a common gradient boosting machine –, and LSTM networks. To evaluate the models, we are using the following metrics: accuracy, precision, recall, f1 score, and ROC AUC score.

4.1 Scoring Metrics

To explain the scoring metrics we use, we introduce the confusion matrix. For a binary classification problem, this matrix consists of four values, as seen in Table 1. TN (true negative) is the total number of instances of class 0 that were correctly identified as class 0. FP (false positive) is the total number of instances of class 0 that were incorrectly identified as class 1. FN (false negative) is the total number of instances of class 1 that were incorrectly identified as class 0. TP (true positive) is the total number of instances of class 1 that were correctly identified as class 1. Given this, $TN+FP+FN+TP$ represents the total number of samples. From these values, we can calculate accuracy, precision, recall, and F1 scores, as shown in equations 5 - 8.

Accuracy is an overall measure of how good the model is at predicting things correctly. Accuracy is how many guesses the model got correct over the total number of samples.

Precision is a score that tells us how precise the class 1 guesses are. Precision is the total number of times the model guessed class 1 correctly over the total number of times the model guessed class 1.

Recall tells us how good the model is at catching all instances of class 1. Recall is the total number of times the model guessed class 1 over all true instances of class 1. Both precision and recall can be flawed metrics. If a model only guesses one instance of class 1, and is correct, the precision score will be 1. Similarly if a model only guesses class 1, the recall score will be 1.

To balance the precision and recall scores and mitigate their individual limitations, we use the F1 score. The F1 score is the harmonic mean of precision and recall, ensuring that it is high only when both precision and recall are high. This provides a more balanced measure of a model's performance in predicting the positive class. These metrics, particularly precision, recall, and F1 score, offer better insights when evaluating a model on an imbalanced dataset.

	Predicted Negative (class 0)	Predicted Positive (class 1)
Negative (class 0)	TN	FP
Positive (class 1)	FN	TP

Table 1: Confusion Matrix

$$accuracy = \frac{TN + TP}{TN + FP + FN + TP} \tag{5}$$

$$precision = \frac{TP}{TP + FP} \tag{6}$$

$$recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1 = 2\frac{precision \cdot recall}{precision + recall} \tag{8}$$

The final score we use is the Receiver Operating Characteristic - Area Under Curve (ROC AUC) score. This metric can be applied to any model that provides a probability of a sample belonging to class 1. It is derived from the ROC curve, which plots the false positive rate (FPR) against the true positive rate (TPR) as the threshold for classifying a sample as class 1 varies. The FPR and TPR are calculated using equations 9 and 10. The ROC AUC score is the area under the curve formed by comparing the TPR to the FRP as the probability threshold increases. The higher the score, the better, with 1 being perfect classification. An example ROC curve is shown in figure 16. The ROC AUC score gives an idea of how good the probability predictions of a model are before applying a threshold. The ROC AUC score gives a overall measure of how well a model performs without having to perform thresholding. We use the ROC AUC score in all contexts where it is applicable.

$$FPR = \frac{FP}{TP + FN} \tag{9}$$

$$TPR = \frac{TP}{TP + FN} = recall \tag{10}$$

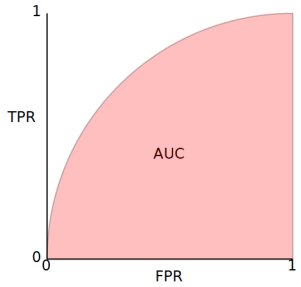

Figure 16: ROC Curve

For our application, we would rather incorrectly report a bubble than miss a bubble, so we weight the recall score higher than other scores. However, since the recall score is flawed by itself, we also include the other scores to find a model that is biased towards detecting bubbles, but is still a good classifier overall.

4.2 ARIMA

ARIMA models are a well used technique to perform predictive forecasting of time-series data.[9] To use ARIMA models to perform classification of time-series data, we train an ARIMA model for each class. To build the training set for these models, we divide the training data according to class and use each subset to train the respective model. To test the data, we feed the density data into each model and use them to predict the next sample. We then test how accurate each prediction is to the actual density value and label the class according to which model predicts the real data better. This formulation is not as sensitive to class imbalance. As long as there is enough data to train a good ARIMA model for each class, the class imbalance does not matter.

4.3 Random Forests

Random forests are a common machine learning approach that have been used in a wide variety of machine learning contexts. Random forests are a combination of tree predictors that are all trained independently.[10] With a large number of estimators, the majority of the estimators tend to agree on the correct classification. In order to use random forests to classify the time series data, we take a window of data around a sample point as the input vector to the random forest.

4.4 XGBoost

Extreme Gradient Boosting (XGBoost) is one of the most common gradient boosting methods used. XGBoost trains trees on the data like a random forest, but instead of training each tree independently, the trees are trained sequentially so that the next tree improves on the deficiencies of the previous tree.[11] Since XGBoost uses decision trees as the base classifier like random forests, we take a window of data around a sample point as the input vector to the XGBoost.

4.5 LSTM

The LSTM is a type of Recurrent Neural Network (RNN) that alleviates the vanishing gradient problem.[12] RNNs work by using the output of a neuron as part of the input. The LSTM truncates the gradient and introduces a forget gate that reduces vanishing and exploding gradients. Since RNNs, and subsequently the LSTM, have a memory and retain past data, they are naturally formulated to perform well on time-series data.

4.6 Feature Engineering

For the random forest and XGBoost methods, we want to include the time dependencies of the data, so to do this, for each density sample, we create a window of data that includes samples from before the sample, and after the sample. Our window size in 101 samples, 50 samples before the current samples and 50 samples after the current sample. Example windows of both classes are shown in figures 17 and 18.

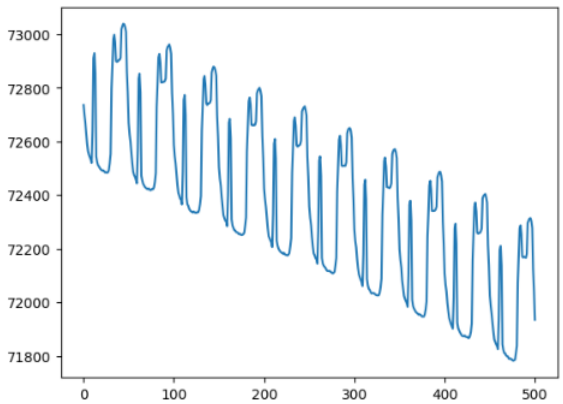

Figure 17: Sample Window of Class 0

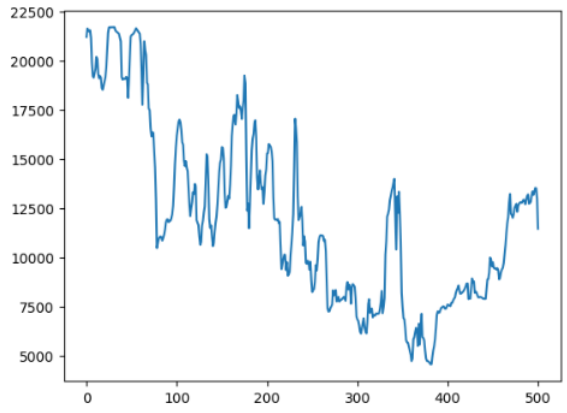

Figure 18: Sample Window of Class 1

In addition to windowing, we decided to perform some feature engineering to try to improve performance. We apply the following pre-processing to the windowed data to create different features for the classifier to use: Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT), Daubechies wavelet db3, and the auto-correlation.

4.6.1 Fast Fourier Transform (FFT)

We noticed that the bubble data has a lot more high frequency components than the non-bubble data. The FFT expresses the windowed data as a function of frequency. An example of this transformation applied to the sample windows from each class is shown in figures 19 and 20 with the DC value removed to better visualize the data. We can see that the bubble data has more high-frequency information than the non-bubble data. This difference in scale can be easily used by a decision tree to classify the data.

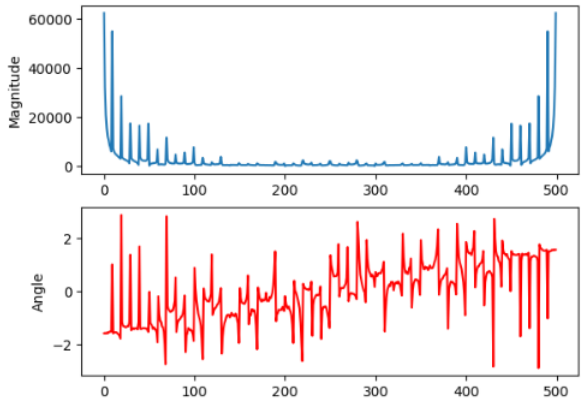

Figure 19: Sample Window of Class 0 with FFT Applied

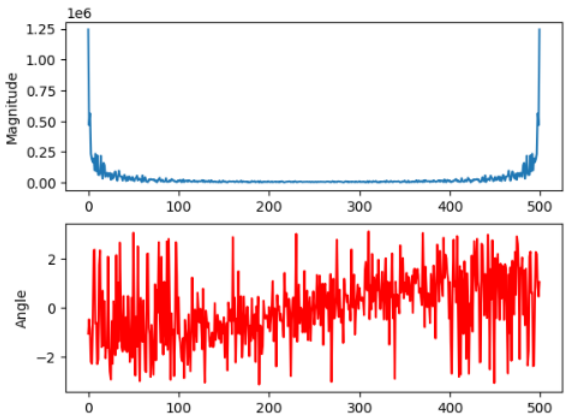

Figure 20: Sample Window of Class 1 with FFT Applied

4.6.2 Discrete Cosine Transform (DCT)

The FFT gives us frequency information, but it produces a complex value, This makes this windowing approach more difficult and computationally expensive when implementing on hardware. A similar transform than outputs a real number is the DCT. The DCT is a transformation that uses the cosine function as the basis function rather than the complex exponential like the FFT does. An example of this transformation applied to the sample windows from each class is shown in figures 21 and 22. Similar to the FFT, the bubble class has a lot more high frequency information that the non-bubble class.

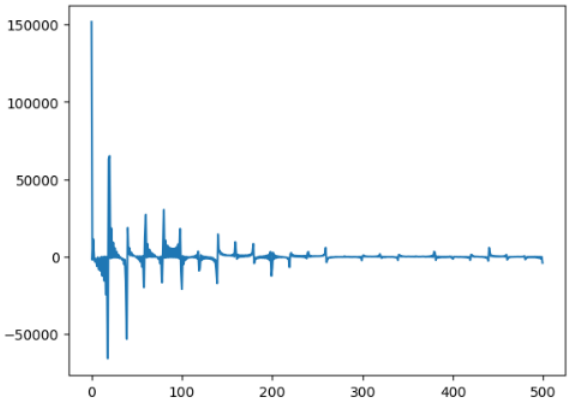

Figure 21: Sample Window of Class 0 with DCT Applied

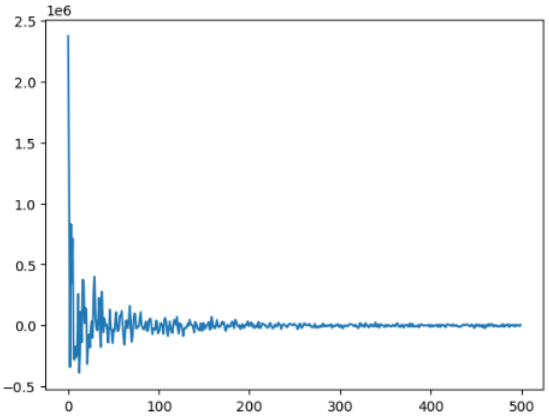

Figure 22: Sample Window of Class 1 with DCT Applied

4.6.3 Daubechies Wavelet Transform

The FFT and DCT both give frequency information, but no temporal information. This may cause errors in the samples transitioning between bubble and no bubble. The wavelet transformation includes high frequency information and preserves temporal information. The Daubechies wavelet transformation involves passing the window trough a specific low pass filter and down sampling to form the "average" samples. The window is also passed through a specific high pass filter and down sampling to form the "difference" samples. An example of the Daubechies 3db wavelet transform applied to the sample windows from each class is shown in figures 23 and 24.

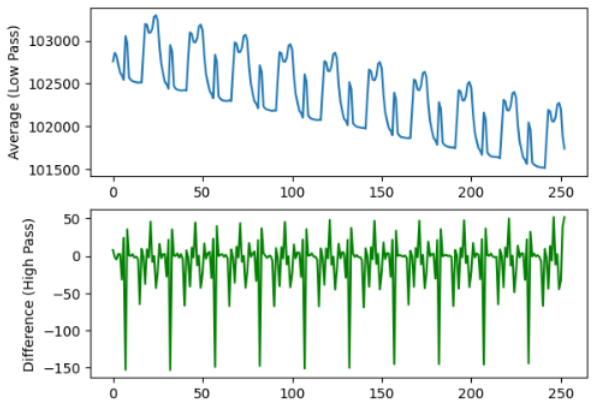

Figure 23: Sample Window of Class 0 with Daubechies 3db Applied

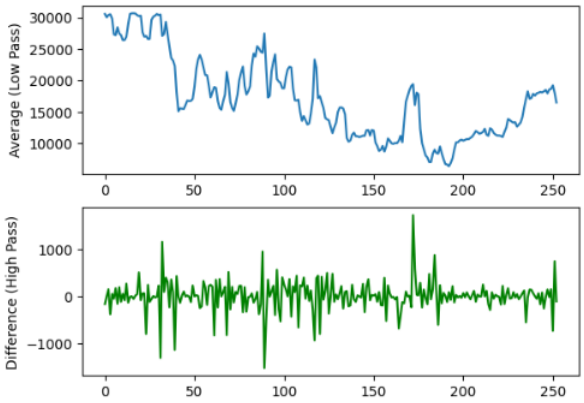

Figure 24: Sample Window of Class 1 with Daubechies 3db Applied

4.6.4 Auto-correlation Function

From studying the data, we can see that the class 0 has a repeating pattern of interference. The pattern changes throughout different sections, but always has approximately the same period. This would show up in the auto-correlation function. We used the cyclic auto-correlation function. This is calculated by using the FFT as seen in equation 11 where * refers to the complex conjugate and $IFFT$ refers to the inverse fast Fourier transform. The auto correlation tells us how much the window looks like itself at different time lags. An example of the auto-correlation function for a sample window from each class is shown in figures 25 and 26. We can see that there are spikes at multiples of 50 lags in the class 0 data that are absent in the class 1 data.

$$auto(x) = IFFT(FFT(x) \cdot FFT(x)^*) \tag{11}$$

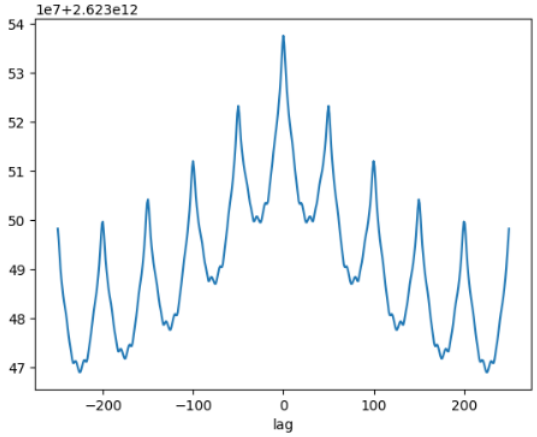

Figure 25: Auto Correlation of Class 0 Sample Window

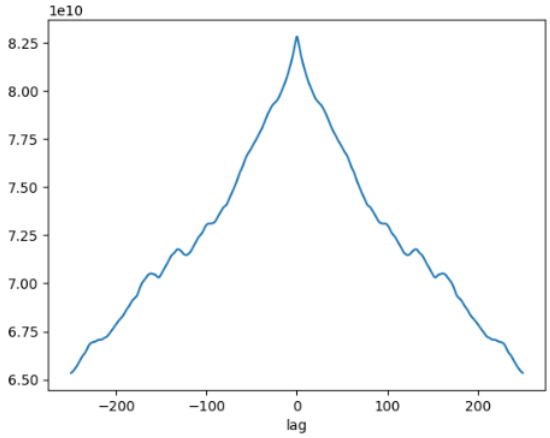

Figure 26: Auto Correlation of Class 1 Sample Window

5 Hyperparameter Tuning Results

We started evaluating the models by performing hyperparameter tuning on each model type. To do this, we created a small subset of the data to use for training and testing. We selected 10 sections of data with a variety of bubble types. We then combined this with synthetic data until the class balance was approximately 20%. We then built a test set out of only data sections selected from the real data. We used these sets to evaluate the models. Since the class balance is approximately 20%, a model is only performing better than randomly guessing if the accuracy of the model is greater than 80%.

The results for each model are shown below after performing hyperparameter tuning. We then compare the results of the best hyperparameters from each model. When analyzing the models, the accuracy score should be higher than 80% if the model is performing well.

5.1 ARIMA

We performed hyperparameter tuning on the ARIMA models using the parameter grid shown in table 2. The performance results of the ARIMA models are shown in table 3. The ROC AUC score does not apply to the ARIMA model, as the model does not output a probability, just a class label.

Table 2: ARIMA Parameter Grid

P	5	10	15
Q	0	5	10

Table 3: ARIMA Model Performance

(P, Q)	accuracy	precision	recall	F1
(5, 0)	0.486	0.251	**0.689**	0.368
(10, 0)	0.833	0.793	0.315	0.451
(15, 0)	0.833	0.793	0.315	0.451
(5, 5)	**0.836**	0.804	0.325	**0.463**
(10, 5)	0.834	0.800	0.315	0.452
(15, 5)	0.834	0.789	0.320	0.455
(5, 10)	0.834	0.801	0.313	0.450
(10, 10)	0.834	0.820	0.303	0.443
(15, 10)	0.831	**0.849**	0.273	0.413

From the results, we can see that the ARIMA model is able to perform classification better than randomly guessing, as the accuracy is higher than 80%. However the recall score is low while the precision score is high. This means that the model is good at not detecting false bubbles, but misses many bubbles. The best ARIMA model has parameters $P = 5, Q = 5$ with the highest accuracy and F1 score as well as the second highest recall score.

5.2 Random Forest

We performed hyperparameter on the random forests as well. We tuned on all window types using 3 fold cross validation and the parameter grid shown in table 4. The results and the best hyperparameters for each window type are shown in tables 5 and 6. Table 5 shows the best model parameters for a given window type. Table 6 shows the results of the model with the best parameters for each window type. In tables 5 and 6, the different window types are specified as follows: A - Raw window, B - FFT window, C - DCT window, D - Daubechies db3 window, E - Auto-correlation window.

Table 4: Random Forest Parameter Grid

number of estimators	100	200
max depth	5	10
min samples split	2	5
min samples leaf	2	4

Table 5: Random Forest Best Models

	number of estimators	max depth	min samples split	min samples leaf
A	100	10	5	2
B	200	10	2	2
C	100	10	2	2
D	200	10	5	4
E	200	10	2	2

Table 6: Random Forest Model Performance

	accuracy	precision	recall	F1	ROC AUC
A	0.841	0.683	0.499	0.577	0.723
B	**0.933**	**0.839**	**0.859**	**0.849**	0.964
C	0.929	0.833	0.842	0.838	**0.966**
D	0.915	0.846	0.743	0.791	0.937
E	0.810	0.595	0.400	0.479	0.679

From the results, we can see that the FFT window performs the best for the random forest model, followed closely by the DCT. Both the FFT and the DCT give information on the frequency content of the window. This means that for the random forest model, analyzing the frequency content of the density is one of the best ways to perform classification.

5.3 XGBoost

We performed hyperparameter tuning on all window types using 5 fold cross validation and the parameter grid shown in table 7. The results and the best hyperparameters for each window type are shown in tables 8 and 9. In tables 8 and 9, the different window types are specified as follows: A - Raw window, B - FFT window, C - DCT window, D - Daubechies db3 window, E - Auto-correlation window.

Table 7: XGBoost Parameter Grid

number of estimators	100	200	300
max depth	3	5	7
learning rate	0.01	0.1	0.2

Table 8: XGBoost Best Models

	number of estimators	max depth	learning rate
A	300	7	0.2
B	300	3	0.01
C	100	7	0.1
D	300	7	0.2
E	300	7	0.2

Table 9: XGBoost Model Performance

	accuracy	precision	recall	F1	ROC AUC
A	0.920	**0.807**	0.831	0.819	0.935
B	**0.925**	0.791	0.893	**0.839**	0.961
C	0.907	0.728	**0.912**	0.810	**0.966**
D	0.912	0.761	0.865	0.809	0.955
E	0.842	0.675	0.522	0.589	0.744

Analyzing this, we can see that the FFT and DCT windows performed the best. From this, we can tell that the frequency content of the density is critical in determining whether there is a bubble or not. This is similar to the results from analyzing the random forest model. For our particular application, we want to skew more towards reporting false bubbles over missing bubbles. As such, we weight a good recall score over other scores, so we would choose the DCT model as our best XGBoost model.

5.4 LSTM

We trained an LSTM on the hyperparameter tuning set. We trained three different small architectures to test the efficacy of the LSTM model. To prepare the data for the LSTM, we broke the input data into sequences similarly to how we sequenced the data for the random forest and XGBoost models. For this model we used a sequence length of 101 samples. We trained three LSTM with structures are seen in table 10. The results for each architecture is shown in table 11.

Table 10: Model Architectures

Model #	Layer	Output Size	Parameter #
1	LSTM	3	60
	Dense	1	4
2	LSTM	5	140
	Dense	1	6
3	LSTM	50	10400
	Dense	1	51

Table 11: LSTM Evaluation Metrics

model	accuracy	precision	recall	F1	ROC AUC
1	0.846	0.838	0.360	0.503	0.536
2	0.843	0.814	0.360	0.499	0.567
3	**0.874**	**0.896**	**0.477**	**0.623**	**0.737**

Analyzing this, we can see that the LSTM is able to classify the data, and that larger LSTM models tend to perform better that the small models. However the recall score is low while the precision score is high. This means that, like the ARIMA model, the model is good at not detecting false bubbles but misses many bubbles.

5.5 Comparison

The best results from each model are shown in table 12.

Table 12: Best Models of Each Type

	ARIMA	Random Forest	XGBoost	LSTM
accuracy	0.836	**0.933**	0.907	0.874
precision	0.804	0.839	0.728	**0.896**
recall	0.325	0.859	**0.912**	0.477
F1	0.463	**0.849**	0.810	0.623
ROC AUC	NA	0.964	**0.966**	0.737

We can see that the random forest model and the XGBoost model perform well with high scores in all five scores. The recall scores of the ARIMA and LSTM models are low, indicating that these models miss a lot of bubbles without detection.

We found that adding the synthetic data significantly improved the performance of the models trained on the hyperparameter tuning set. For example, the best performing XGBoost model trained on the real dataset with sample weighting is compared to the best XGBoost model trained using the data augmented with synthetic data in table 13.

Table 13: Comparing Results from Best XGBoost Models

	augmented data set	real data only
accuracy	0.907	0.847
precision	0.728	0.602
recall	0.912	0.877
F1	0.810	0.714
ROC AUC	0.966	0.879

6 Training on The Full Data Set

We then trained the models on the full dataset. We built two versions of the full dataset, the first set is the SPORT data reduced to just the sweeps that have bubbles. This dataset is approximately 12% class 1 (bubbles). Sample weighting is applied to this set to help overcome the data imbalance issue. To build the second training set, we added synthetic data to the SPORT data until the class balance was approximately 40%.

We divided the full dataset into a training set, a validation set, and a test set. The training set is 70% of the data, and the validation and test sets are both 15% of the data. When building the test dataset, we ensure that the test set does not include any synthetic data or data that was used to perform hyperparameter tuning. Doing this means that the test set balance is approximately 10% class 1. This means that when evaluating a model, an accuracy of 90% or lower is less accurate than always guessing class 0. Because of this imbalance, the precision, recall, and F1 scores are more informative than accuracy in evaluating a model. After evaluating the models on the test set and forming predictions, we applied a median filter to smooth the noise in the results. Since the median filter does not give a probability score, the ROC AUC score does not apply to this analysis.

6.1 ARIMA

We trained an ARIMA model using the best parameters from our hyperparameter tuning on the full data set. The ARIMA model has parameters $P = 5$, $D = 1$, $Q = 5$. We then put the output of the ARIMA prediction through a median filter to smooth the results. We found that the ARIMA model trained on the full dataset without synthetic data performed better than the model trained on the full dataset augmented with synthetic data. The re-

sults of the ARIMA model trained on the full dataset without synthetic data is shown in table 14.

Table 14: Results from ARIMA Model Trained on the Full Dataset with Median Filter

Filter size	accuracy	precision	recall	F1
1	0.9232	0.3268	**0.1694**	**0.2232**
13	**0.9343**	**0.4804**	0.1078	0.1761
55	0.9332	0.4057	0.0562	0.0987

We noticed that the recall score of the ARIMA model is significantly lower that the recall scores from hyperparameter tuning. We compare the Random Forest model trained on the hyperparameter tuning set tested on the test set from the full dataset in table 15. Analyzing these models, we can see that the ARIMA model trained on the full dataset has a higher accuracy and the model trained on the hyperparameter has higher precision, recall, and F1 score. This means that the hyperparameter model is better at identifying class 1 examples than the model trained on the full dataset, but does make more mistakes overall.

Table 15: Results from ARIMA Model Trained on the Hyperparameter Tuning Set with Median Filter

Filter size	accuracy	precision	recall	F1
1	0.9091	0.5290	**0.2498**	**0.3394**
13	**0.9205**	0.7692	0.2139	0.3347
55	0.9184	**0.7749**	0.1796	0.2916

6.2 Random Forests

We trained a Random forest model using the best parameters for hyperparameter tuning shown in table 16 on the full dataset. We found that the Random Forest model trained on the full dataset without synthetic data performed better that the model trained on the full dataset augmented with synthetic data. The results of the model trained on the full dataset without synthetic data is shown in table 17.

Table 16: Random Forest Best Parameter Grid

number of estimators	max depth	min samples split	min samples leaf
200	10	5	2

Table 17: Results from Random Forest Model Trained on the Full Dataset with Median Filter

Filter size	accuracy	precision	recall	F1
1	0.9639	0.9135	0.4928	0.6402
13	0.9636	0.9135	0.4931	0.6405
55	**0.9641**	**0.9172**	**0.4934**	**0.6416**

We noticed that the recall score of the Random Forest model is significantly lower that the recall scores in the hyperparameter tuning set. We compare the Random Forest model trained on the hyperparameter tuning set tested on the test set from the full dataset in table 18. Analyzing this we see that the model trained on the full dataset without synthetic data has higher accuracy and precision while the model trained on the hyperparameter tuning set has higher recall and F1 score. This means that the model trained on the hyperparameter tuning set catches more bubbles at the expense of missclassifying a few sections of class 0 as class 1.

Table 18: Results from Random Forest Model Trained on the Hyperparameter Tuning Set with Median Filter

Filter size	accuracy	precision	recall	F1
1	0.9156	0.5305	0.8436	0.6514
13	0.9159	0.5315	0.8458	0.6528
55	**0.9163**	**0.5329**	**0.8486**	**0.6547**

6.3 XGBoost

We trained an XGBoost model on the full dataset with the best parameters from hyperparameter tuning, shown in table 19. We found that the XGBoost model trained on the full dataset without synthetic data performed better that the model trained on the full dataset augmented with synthetic data.

Table 19: XGBoost Best Parameter Grid

number of estimators	max depth	learning rate
100	7	0.1

After training the model, we tested on the test data set and fed the results through a median filter to reduce noise in the results. The results of different median filter sizes are shown in table 20.

Table 20: Results from XGBoost Model Trained on the Full Dataset with Median Filter

Filter size	accuracy	precision	recall	F1
1	0.9530	0.8867	0.3197	0.4700
13	0.9533	0.9040	0.3170	0.4694
55	**0.9537**	**0.9200**	**0.3171**	**0.4716**

We noticed that the recall score of the XGBoost model is significantly lower that the recall scores in the hyperparameter tuning set. We compare the XGBoost model trained on the hyperparameter tuning set tested on the test set from the full dataset in table 21. Analyzing this we see a similar scenario as the Random Forest model. The model trained on the full dataset without synthetic data has higher accuracy and precision while the model trained on the hyperparameter tuning set has higher recall and F1 score. This means that the model trained on the hyperparameter tuning set catches more bubbles at the expense of miss-classifying a few sections of class 0 as class 1.

Table 21: Results from XGBoost Model Trained on the Hyperparameter Tuning Set with Median Filter

Filter size	accuracy	precision	recall	F1
1	0.9347	0.6806	0.8644	0.7616
13	0.9352	0.6818	0.8675	0.7635
55	**0.9358**	**0.6838**	**0.8703**	**0.7659**

6.4 LSTM

Since the LSTM performs better on a larger dataset, we re-did the hyperparameter tuning on the LSTM model. The data was sequenced into sequences 101 samples long, the same as in the small hyperparameter tuning set. We tested six different model architectures with model sizes seen in table 22. These models were trained on the full dataset for 1 epoch and compared in table 23. When training the LSTM, the full dataset augmented with synthetic data performed better than training on the full dataset without synthetic data.

Table 22: LSTM Model Architectures for Full Dataset

Model #	Layer	Output Size	Parameter #
1	LSTM	50	10400
	Dense	1	51
2	LSTM	100	40800
	Dense	1	101
3	LSTM	50	10400
	LSTM	25	7600
	Dense	1	26
4	LSTM	100	40800
	LSTM	50	30200
	Dense	1	51
5	LSTM	50	10400
	LSTM	40	14560
	LSTM	30	8520
	LSTM	20	4080
	LSTM	10	1240
	Dense	1	11
6	LSTM	100	40800
	LSTM	75	52800
	LSTM	50	25200
	LSTM	25	7600
	LSTM	10	1440
	Dense	1	11

Table 23: LSTM Evaluation Metrics

model	accuracy	precision	recall	F1	ROC AUC
1	**0.900**	**0.322**	0.057	0.007	0.467
2	**0.900**	0.286	0.043	0.006	0.460
3	0.880	0.278	0.176	0.018	0.569
4	0.885	0.311	**0.187**	**0.019**	**0.584**
5	0.883	0.226	0.101	0.011	0.556
6	0.896	0.287	0.073	0.009	0.517

Analyzing this round of hyperparameter tuning, we can see that while the smaller models have high accuracy, the low recall score indicates that the model is missing many of the bubbles. Model number 4 scores highest on recall and still scores high on all other scores. We take the architecture of model 4 and train it on the full augmented dataset for 10

epochs. The results of this LSTM with a median filter applied are shown in table 24.

Table 24: Results from LSTM Model Trained on the Full Dataset with Median Filter

Filter size	accuracy	precision	recall	F1
1	0.8512	0.2929	0.4178	0.3444
13	0.8512	0.2933	0.4185	0.3449
55	**0.8514**	**0.2944**	**0.4211**	**0.3465**

We noticed that in all of the other model types, a model trained on the hyperparameter tuning data set tends to have a higher recall score than the model trained on the full dataset. Because of this, we decided to train an LSTM model with the same architecture on the hyperparameter tuning set for 10 epochs and test it on the same test set as the full dataset. The results from the LSTM trained on the hyperparameter tuning dataset are shown in table 25. Analyzing the LSTM results, we can see that the model trained on the full dataset has a higher recall and F1 score, while the model trained on the hyperparameter tuning set has higher accuracy and precision. Interestingly, this is the reverse of the trend seen in the other models, where using the hyperparameter tuning set tends to increase recall and the F1 score at the expense of accuracy and precision.

Table 25: Results from LSTM Model Trained on the Hyperparameter Tuning Set with Median Filter

Filter size	accuracy	precision	recall	F1
1	0.8771	0.3054	0.2456	0.2723
13	**0.8772**	**0.3056**	**0.2459**	**0.2725**
55	0.8770	0.3047	0.2448	0.2715

6.5 Comparing Models

We now compare the best model from each model type and dataset in table 26. Analyzing these results, we can see that the tree-based methods tend to have the highest scores overall. We can also see that the models trained on the hyperparameter tuning set tend to have a much higher recall score at the cost of lowering the accuracy and precision.

Table 26: Final Results of Models Tested on the Full Dataset

Model	accuracy	precision	recall	F1
ARIMA Full Dataset	0.9232	0.3268	0.1694	0.2232
ARIMA Hyperparameter Dataset	0.9091	0.5290	0.2498	0.3394
Random Forest Full Dataset	**0.9641**	0.9172	0.4934	0.6416
Random Forest Hyperparameter Dataset	0.9163	0.5329	0.8486	0.6547
XGBoost Full Dataset	0.9537	**0.9200**	0.3171	0.4716
XGBoost Hyperparameter Dataset	0.9358	0.6838	**0.8703**	**0.7659**
LSTM Full Dataset	0.8514	0.2944	0.4211	0.3465
LSTM Hyperparameter Dataset	0.8772	0.3056	0.2459	0.2725

Figure 27 presents a detailed analysis of a sample data section, comparing actual class labels with predicted class labels from our XGBoost model trained on the hyperparameter tuning dataset. In this section, we observe an extensive region of plasma bubbles followed by several smaller bubble regions. The model successfully identifies most of the extensive bubble region, though it does not capture it continuously. Additionally, the model incorrectly labels some non-bubble regions as bubbles. Despite these inaccuracies, our primary goal is to determine the occurrence of plasma bubbles over a large time scale. The goal of the plasma bubble detection application on the ITA-SAT2 satellite that this model is being developed for is to detect if there is any plasma bubble interference as the satellite passes through the equatorial region. Detecting the entire bubbles is

not as important as detecting that there is a bubble in the equatorial region. Therefore, even though the model does not detect every part of the bubble, it effectively identifies the general areas where bubbles are present.

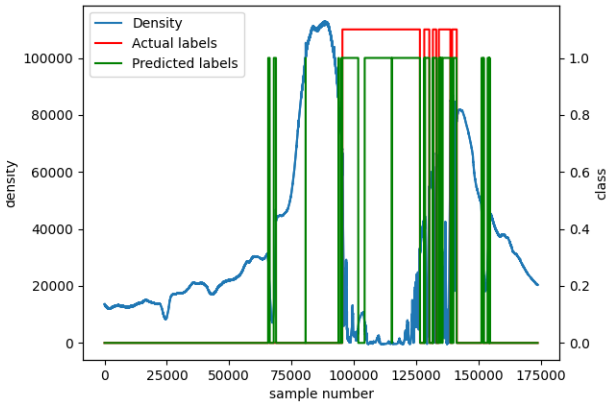

Figure 27: Section of Density Data with Actual Labels and Predicted Labels from the XGBoost Model Trained on the Hyperparameter Tuning Set

7 Model Complexity

The goal of building this model is to implement it on an FPGA on a small satellite. Because of this we want to prioritize simple models that can be easily implemented on a satellite. We analyze the complexity of each model.

7.1 ARIMA

A ARIMA model defined as ARIMA(P,D,Q) requires $P+Q$ multiplications and $P+Q+D-2$ additions. Our approach has two ARIMA models, and the results of each model is compared. The equation to calculate the total number of floating point operations (FLOP) required for our ARIMA approach is shown in equation 12. Our best model had parameters $P=5$, $D=1$, $Q=5$. This gives a total of 39 FLOPs per sample.

$$FLOP = 2*(2P+2Q+D-2)+1 \tag{12}$$

7.2 Random Forest

A Random Forest model uses decision trees to classify a sample point. To evaluate the trees, we need to make at most the maximum depth of the tree comparisons per tree, then find the majority of the tree results. The number of comparisons, or FLOPs required for a random forest is shown in equation 13. Our best model has 200 estimators with a maximum depth of 10. This means according to equation 13 that the best random forest evaluation requires approximately 2200 FLOPs per sample.

$$FLOP = \#estimators * max\_depth + \#estimators \tag{13}$$

7.3 XGBoost

The XGBoost model is similar to the Random Forest model, as it requires the evaluation of of a set of decision trees, then finding the majority of the results. The total FLOPs for an XGBoost model is approximated by equation 13 Our best XGBoost model has 100 estimators and a maximum depth of 7. This means that, according to equation 13, the XGBoost model requires approximately 800 FLOPs per sample.

7.4 LSTM

The LSTM architecture consists of various LSTM layers followed by a dense layer. The total number of FLOPs per LSTM layer is calculated by equation 14 where h is the size of the output layer and d is the size of the input layer and the total number of FLOPs per Dense layer is calculated by equation 15 where m is the size of the output layer and d is the size of the input layer. The total size of each layer in our LSTM architecture is shown in table 27. Using these equations, we calculate that the LSTM requires 284100 FLOPs per sample.

$$FLOP_{LSTM} = 16*(d*h+h^2+h) \tag{14}$$

$$FLOP_{Dense} = 2*d*m \tag{15}$$

Table 27: Size of LSTM Layers

Layer	Input Size	Output Size
LSTM	1	100
LSTM	100	50
Dense	50	1

We compare the complexity of all different model types in table 28. We can see that the LSTM is orders of magnitude more complex than any other model. The LSTM is more difficult to implement on hardware than any other model. Additionally, the tree-based models, random forests and XGBoost, use comparisons which are implemented as subtraction operations in hardware. This operation is simpler to implement in custom hardware on an FPGA than multiplications.

Table 28: Comparison of Model Complexity

Model	FLOPs per sample
ARIMA	39
Random Forest	2200
XGBoost	800
LSTM	284100

8 Conclusion

We successfully trained several machine learning models to classify equatorial plasma bubbles as observed in ionospheric density data from a satellite in a mid inclination orbit. Among these models, the Random Forest and XGBoost methods achieved the highest scores. Our best-performing model is the XGBoost model, trained using our hyperparameter tuning set, with an accuracy of 93.58% and an F1 score of 0.7659. These models are also significantly less complex than the LSTM model, making them simpler to implement in hardware. The best-performing Random Forest and XGBoost models used the FFT and DCT windows, respectively, indicating that analyzing the frequency content of the SPORT density data is an effective method for locating plasma bubbles. This is consistent with the approach used by SPORT data scientists, who look for regions in the density data where the density drops and appears "noise-like."

Another significant finding is that adding synthetic data improved model performance on small scales but not on large scales. This suggests that a small amount of synthetic data enhances the model's understanding, but too much synthetic data degrades performance. Additionally, models trained on the hyperparameter tuning set consistently achieved higher recall and F1 scores. This indicates that, in terms of recall, the larger dataset causes overfitting, thereby reducing performance.

8.1 Future Work

Looking forward, we plan to continue improving our plasma bubble detection model. In this paper, we only used the density data collected by the Langmuir probe on the SWP instrument. The SWP instrument also includes an electric field probe, a magnetometer, and an impedance probe that calculates plasma density. Plasma bubbles are detectable in data from these probes as well. Incorporating data from these additional probes may enhance the performance of the plasma bubble classifier.

We also plan to implement our model on an FPGA. We will deploy the architecture on the LACE-C3A board to perform plasma bubble classification on a low-power computing device. The LACE-C3A board will be flown on the ITA-SAT2 mission alongside a new version of the SWP instrument, where it will classify bubbles using data from the SWP instrument.

References

[1] Yokoyama, T. A review on the numerical simulation of equatorial plasma bubbles toward scintillation evaluation and forecasting. Prog Earth Planet Sci 4, 37 (2017).

[2] Hysell, D. L., Kirchman, A., Harding, B. J.,Heelis, R. A., England, S. L., Frey, H. U.,& Mende, S. B. (2024). Using ICON satellite data to forecast equatorial ionospheric instability throughout 2022.Space Weather, 22, e2023SW003817. (2023)

[3] Su, S.-Y., H. C. Yeh, and R. A. Heelis (2001), ROCSAT 1 ionospheric plasma and electrodynamics instrument observations of equatorial spread F: An early transitional scale result, J. Geophys. Res., 106(A12), 29153–29159, doi:10.1029/2001JA900109.

[4] C. Stolle, H. Lu¨hr, M. Rother, and G. Balasis, "Magnetic signatures of equatorial spread F as observed by the CHAMP satellite", 2006

[5] Camporeale, E. (2019). The challenge of machine learning in Space Weather: Nowcasting and forecasting. Space Weather, 17, 1166–1207. https://doi.org/10.1029/2018SW002061

[6] Thanakulketsarat, T., Supnithi, P., Myint, L.M.M. et al. Classification of the equatorial plasma bubbles using convolutional neural network and support vector machine techniques. Earth Planets Space 75, 161 (2023). https://doi.org/10.1186/s40623-023-01903-7

[7] Chandan Kapil, Gopi K. Seemala, Machine learning approach for detection of plasma depletions from TEC, Advances in Space Research, Volume 73, Issue 7, 2024, Pages 3833-3844, ISSN 0273-1177, https://doi.org/10.1016/j.asr.2023.04.042.

[8] Reddy, S. A., Forsyth, C., Aruliah, A., Smith, A., Bortnik, J., Aa, E., Kataria, D. O., & Lewis, G. (2023). Predicting swarm

equatorial plasma bubbles via machine learning and Shapley Values. Journal of Geophysical Research: Space Physics, 128(6). https://doi.org/10.1029/2022ja031183

[9] Box, G. E. P., Jenkins, G. M. (1968). Some recent advances in forecasting and control. Journal of the Royal Statistical Association: C, XVII, 91-109.

[10] Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

[11] Chen, T., & Guestrin, C. (2016). XGBoost. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. https://doi.org/10.1145/2939672.2939785

[12] Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.