Victor Allen
CS549
Assignment04

Introduction

PULSE-M (Performance & Utilization Link 16 Signal Energy - Monitor) is a pulse density monitor created by Timber Sweet Consulting to monitor Link 16 networks and to ensure that they meet all requirements. It is used to monitor transmissions and to notify users if there is any potential interference to navigation or other important or potentially sensitive equipment or location(s). It does this by allowing the user to compare their activity against known "Network Monitoring" tools, which helps to identify anything from rogue users to split networks and other inconsistencies that may be of importance to the operator.

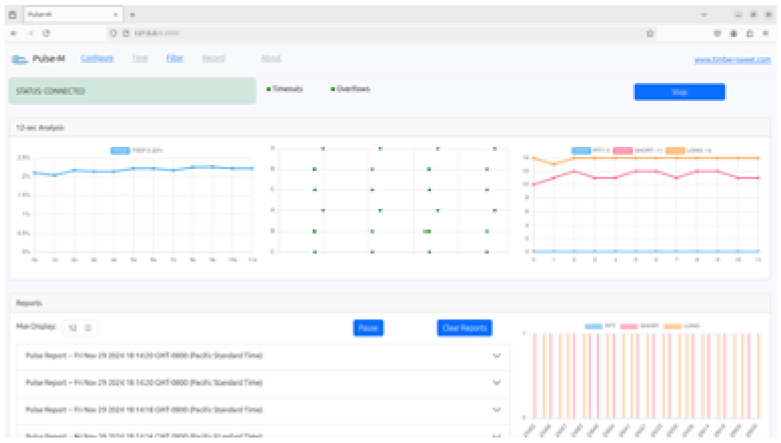
An important part of the operation of PULSE-M is to determine the start time of a transmission given the transmission's data detected by the PULSE-M. It is important here to make a distinction. Time is not a continuous value in this scenario. Instead there are many different known transmission start times and we are trying to identify the correct one based on the transmission data. This distinction makes this particular problem not a regression problem, like is normal for problems involving the prediction of traditionally continuous values, and makes it a classification problem. The purpose of the PULSE-M(L) project is to apply machine learning algorithms and approaches to correctly identify and classify the start time of a transmission based on the features provided.

Data Acquisition

In order to obtain the dataset, a BATS-D (Battlefield Awareness and Targeting System - Dismounted) radio was set up, and every transmitter in

the radio was known before the test was conducted. It was used to send messages and pulses out that were captured by a Software Defined Radio designed to listen for pulses. Each pulse was captured and recorded and the relevant information was recorded using CFAR (Constant False Alarm Rate) detection techniques and theory. This is significantly above my pay grade, for all I am aware the radio field is black magic. The random noise category in the dataset is also introduced here. As the result of radio pulses bouncing on and off objects in the environment, useless and interferetory data can be recorded and must be differentiated from useful radio pulses.

An around 12 second snapshot was taken, and each pulse was manually inspected and labeled with the appropriate message start time. The original motivation for this problem arises from the want to avoid writing an algorithm that does the same job, however it it somewhat ironic that some form of algorithm was needed to properly label the data. In the original dataset (`data.csv`) the pulses intended to be random noise were left unlabelled. I wrote a brief program to assign them all the value of zero somewhat arbitrarily as its super far from all of the other labels.



Pulse Report -- Fri Nov 29 2024 18:14:20 GMT-0800 (Pacific Standard Time)

Pulse Report -- Fri Nov 29 2024 18:14:20 GMT-0800 (Pacific Standard Time)

Pulse Report -- Fri Nov 29 2024 18:14:18 GMT-0800 (Pacific Standard Time)

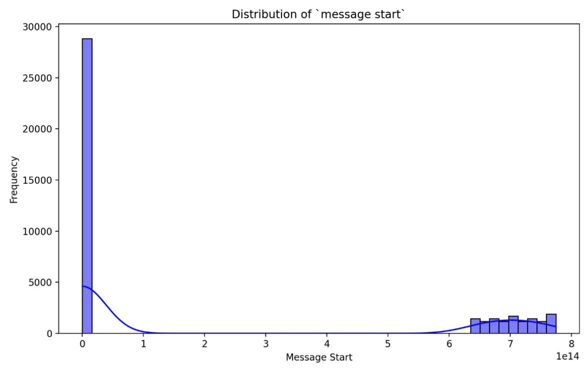
An Overview of the Data

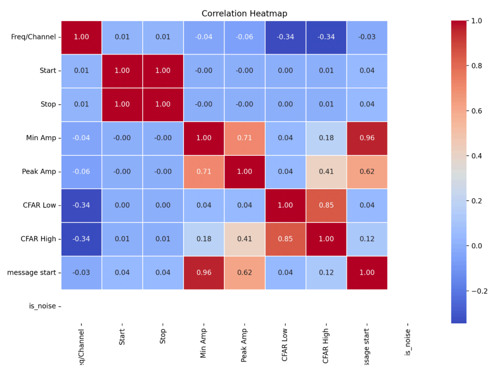
The data has 41413 different observations. Of those, 12590 are actual transmissions and the rest is noise. There are 32 individual start times, ranging from `639218752193141` to `774453126171891`, meaning that including the random noise, there are 33 different possible categories that a single observation could fall into. There are seven different features that are suspected to influence the start of a message, and each datum in the CSV is organized in this way:

`Freq/Channel,Start,Stop,Min Amp,Peak Amp,CFAR Low,CFAR High,message start.`

Note that messages to be classified as noise are assigned a `message start` value of zero.

By far the most common category was random noise, as such it is important that the models are trained with random noise, in order to recognize and classify it, but also if it becomes overbearing the number of training samples shown should be reduced to increase accuracy on the other end.





This correlation heat map shows the strength of the (linear) correlation between variables. It shows that the strongest predictors of the `message start` field are the `Min` and `Peak Amp` Fields. (The main diagonal is to be ignored as a feature is expected to have a perfect correlation with itself). However there may be other non linear trends that are not reflected here and that will be discovered by including and excluding certain features in the training process.

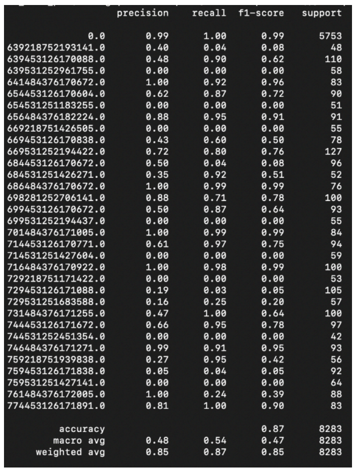
Model Analysis

I chose to evaluate two models that I think had the most interesting or surprising results. I had suspected initially that a Neural Network would be a great selection for this problem, as I had seen its success in machine learning projects like classifying handwritten digits and I thought that the high number of possible categories and the complexity of the data would make a neural network a great fit. As I will explore, there I think the data was too convoluted for the network to process.

On the other hand, before this project, I was completely unaware of Random Forest trees and how they worked. However, upon studying how they work and the conditions in which they work best including massive, convoluted datasets, it was a natural choice for this problem. As will be seen it performed admirably. With a more accurate dataset, it is possible that the results would be perfect. Further, given that one of the features of the model is that overfitting is rare, we can be confident that the model will perform well on data that it has not seen before.

Metrics for analysis are standard for classification tasks. It will include the overall accuracy of the model, meaning the total number of correct classifications that the model makes. However equally important will be the precision and recall rates for individual categories, as the accuracy can miss important imperfections in the output like mis-identifying entire categories that will not be missed when the precision and recall for that category is examined. Further the size and complexity of the model will be evaluated and attempts will be made to minimize its size and complexity to make it less computationally intensive.

Neural Network



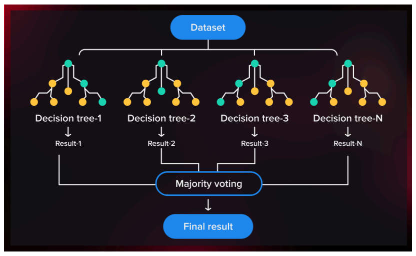
```
                   precision    recall  f1-score   support

             0.0        0.99      1.00      0.99      5753
639218752193141.0       0.40      0.04      0.08        48
639453126170088.0       0.48      0.90      0.62       110
639531252961755.0       0.00      0.00      0.00        58
641484376170672.0       1.00      0.92      0.96        83
654453126170604.0       0.62      0.87      0.72        90
654531251183255.0       0.00      0.00      0.00        51
656484376182224.0       0.88      0.95      0.91        91
669218751426505.0       0.00      0.00      0.00        55
669453126170838.0       0.43      0.60      0.50        78
669531252194422.0       0.72      0.80      0.76       127
684453126170672.0       0.50      0.04      0.08        96
684531251426271.0       0.35      0.92      0.51        52
686484376170672.0       1.00      0.99      0.99        76
698281252706141.0       0.88      0.71      0.78       100
699453126170672.0       0.50      0.87      0.64        93
699531252194437.0       0.00      0.00      0.00        55
701484376171005.0       1.00      0.99      0.99        84
714453126170771.0       0.61      0.97      0.75        94
714531251427604.0       0.00      0.00      0.00        59
716484376170922.0       1.00      0.98      0.99       100
729218751171422.0       0.00      0.00      0.00        53
729453126171088.0       0.19      0.03      0.05       105
729531251683588.0       0.16      0.25      0.20        57
731484376171255.0       0.47      1.00      0.64       100
744453126171672.0       0.66      0.95      0.78        97
744531252451354.0       0.00      0.00      0.00        42
746484376171271.0       0.99      0.91      0.95        93
759218751939838.0       0.27      0.95      0.42        56
759453126171838.0       0.05      0.04      0.05        92
759531251427141.0       0.00      0.00      0.00        64
761484376172005.0       1.00      0.24      0.39        88
774453126171891.0       0.81      1.00      0.90        83

        accuracy                            0.87      8283
       macro avg        0.48      0.54      0.47      8283
    weighted avg        0.85      0.87      0.85      8283
```

The results of the Neural Network were surprising. Given that this is a fairly complex dataset with literally tens of thousands of observations, and the suspected complexity of the relationships, I initialized the model to be pretty complex, it had a 33 neuron input layer, with two hidden layers of 256, and 128 neutrons respectively with standard activation functions and algorithms like batch normalization and dropout applied to prevent overfitting. If one were merely concerned with the total accuracy of the network, 87%, while not great, is acceptable. However, a closer inspection of the report shows that most of the accuracy comes from the correct identification of random noise, and that the classification of the correct `message start` time for data that are not random noise is actually quite a bit less. I wrote a program to filter out all random noise from the dataset, resulting in a CSV called "`filtered_data.csv`", leaving only the 32 actual start times as categories. When testing was performed on this dataset, there were similar results to the table above. However without the noise classification, the accuracy of the model dropped to the high 40s - 60s. I tried several things, including increasing the complexity of the model, both by increasing the number of neurons in a layer and by increasing the amount of layers in the model, but nothing increased the recall and precision of important categories in significant ways.

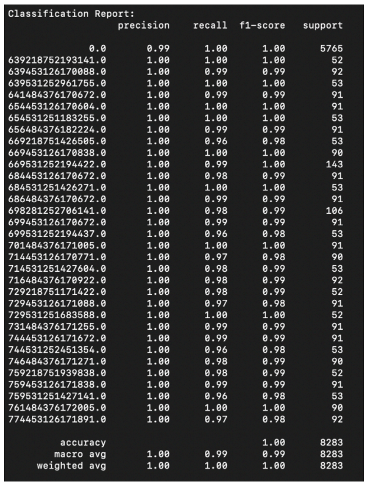
I was informed that there were some inconsistencies and mis-classifications in the data set itself, which I think could be helping to cause the inaccuracies that are present but I suspect they will not completely explain them. It is also worth noting that training a good Neural Network requires good data, a lot of experience, and resources. It is more of an art than a science and it’s possible someone more experienced could create a better algorithm.

Random Forest Classifier

The random forest classifier was a huge surprise. A model not taught in class, it works by using multiple decision trees. Each tree is trained on a random subset of the data and uses a random selection of features for splitting at each node, which introduces diversity in the “forest”. Predictions are made by averaging the outputs of all trees, (usually a majority vote for classification). It reduces overfitting, makes better generalizations and performs particularly well in high dimensional and noisy datasets, making it a perfect candidate for this dataset.



When trained on the whole dataset, the Random Forest Tree performed almost perfectly, as there were almost no errors, the accuracy was perfect (or close very close to) and the recall and precision for individual categories was never less than 96%, which is well within the “acceptable” range, especially when one considers that the overall accuracy of the model was so close to 1.



```
Classification Report:
                    precision    recall  f1-score   support

               0.0       0.99      1.00      1.00      5765
639218752193141.0       1.00      1.00      1.00        52
639453126170088.0       1.00      0.99      0.99        92
639531252961755.0       1.00      1.00      1.00        53
641484376170672.0       1.00      0.99      0.99        91
654453126170604.0       1.00      1.00      1.00        91
654531251183255.0       1.00      1.00      1.00        53
656484376182224.0       1.00      0.99      0.99        91
669218751426505.0       1.00      0.96      0.98        53
669453126170838.0       1.00      1.00      1.00        90
669531252194422.0       1.00      0.99      1.00       143
684453126170672.0       1.00      0.98      0.99        91
684531251426271.0       1.00      1.00      1.00        53
686484376170672.0       1.00      0.99      0.99        91
698281252706141.0       1.00      0.98      0.99       106
699453126170672.0       1.00      0.99      0.99        91
699531252194437.0       1.00      0.96      0.98        53
701484376171005.0       1.00      1.00      1.00        91
714453126170771.0       1.00      0.97      0.98        90
714531251427604.0       1.00      0.98      0.99        53
716484376170922.0       1.00      0.98      0.99        92
729218751171422.0       1.00      0.98      0.99        52
729453126171088.0       1.00      0.97      0.98        91
729531251683588.0       1.00      1.00      1.00        52
731484376171255.0       1.00      0.99      0.99        91
744453126171672.0       1.00      0.99      0.99        91
744531252451354.0       1.00      0.96      0.98        53
746484376171271.0       1.00      0.98      0.99        90
759218751939838.0       1.00      0.98      0.99        52
759453126171838.0       1.00      0.99      0.99        91
759531251427141.0       1.00      0.96      0.98        53
761484376172005.0       1.00      1.00      1.00        90
774453126171891.0       1.00      0.97      0.98        92

         accuracy                           1.00      8283
        macro avg       1.00      0.99      0.99      8283
     weighted avg       1.00      1.00      1.00      8283
```

Given the overall accuracy of the model, and in the interest of making the model lighter and easier to run, I tired cutting out features in various combinations and investigated their accuracy (note that the results for the accurate model were similar to the above report in each category):

Only including `Min` and `Peak Amp`	`Accuracy Score: 0.7048171`
Above and including `CFAR Low` and `High`	`Accuracy Score: 0.7053000`
Above with `Start` and `Stop`	`Accuracy Score: 0.996378`
With only `Start` and `Stop`	`Accuracy Score: 0.303996`
With `Start, Stop, Min` and `Peak Amp`	`Accuracy Score: 0.996740`

It seems of the 7 features, the most important indicators for accuracy were only the `Min` and `Peak Amps` and the `Start` and `Stop` values. While including the other values does raise the accuracy of the model, the gain is negligible at best meaning that there is likely little to no relationship between them and the `message start`.

Conclusions

I think the biggest takeaway from this experiment is the difference in success from classifying noise and actual categories. Most models (I had trained a couple more but found the two featured ones to be the most interesting. See the code in the repository for other models) found it significantly easier to create a model that accurately classified noise or not noise than it was to classify each datum according to its category. The disparity, I think, highlights the importance of quality data in machine learning.

The contrast between the neural network and the Random Forest classifier highlights the importance of selecting the right algorithm for the problem at hand. While neural networks may excel in certain domains, such as image or natural language processing, they may not always be the best choice for structured datasets, especially when the data quality is not ideal. I had assumed that it might be useful based on its in class description and my understanding of how it worked. On the other hand, Random Forest classifiers are often more robust to noise and inconsistencies, making them a reliable choice for real-world problems.