AFRL-RY-WP-TR-2022-0146

SEQUENTIAL LEARNING WITH VERY NON-UNIFORMLY SAMPLED TIME SERIES

Michael Gregg
The Ohio State University

APRIL 2022
Final Report

STINFO COPY

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
April 2022	Thesis	12 April 2022	12 April 2022

4. TITLE AND SUBTITLE

SEQUENTIAL LEARNING WITH VERY NONUNIFORMLY SAMPLED TIME SERIES

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
N/A	N/A	N/A
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
N/A	N/A	N/A

6. AUTHOR(S)

Michael Gregg

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ohio State University
281 W. Lane Ave.
Columbus, OH 43210

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'SREPORT NUMBER(S)
Air Force Research Laboratory, Sensors Directorate Wright-Patterson Air Force Base, OH 45433-7320 Air Force Materiel Command, United States Air Forces	AFRL/RYWE	AFRL-RY-WP-TR-2022-0146

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

PAO case number AFRL-2022-1701, Clearance Date 12 April 2022. M.S. final project report submitted to The Ohio State University, Columbus, OH, towards a Master of Science in Electrical and Computer Engineering. This work was funded in whole or in part by Department of the Air Force. The U.S. Government has for itself and others acting on its behalf an unlimited, paid-up, nonexclusive, irrevocable worldwide license to use, modify, reproduce, release, perform, display, or disclose the work by or on behalf of the U. S. Government. Report contains color.

14. ABSTRACT

Time series classification problems implement supervised machine learning techniques to analyze temporally ordered data and classify new sequential data. Time series classification has grown in popularity as access to time series data has increased in recent years, and the problems have appeared across a wide spectrum of applications such as audio recordings, medical signals, and weather prediction. Generally, an assumption is made that the temporal ordering is uniformly or close to uniformly sampled. However, there are important applications where this is not the case. This project looked at a dataset that was a very non-uniformly sampled time series with the task of classification of three labels. The dataset was also quite large and required very high dimensional features. These considerations encouraged the use of sequential learning techniques. Sequential learning refers to machine learning models that have sequences of data as the input or output. The goal of this project was to identify pre-processing techniques and approaches for generating sequences that would be helpful for this classification task. If successful, the results could help give insights to similar sequential learning problems. The data were first standardized over the entire dataset. The data as given had large gaps of time where no samples resided, called “dead” zones, that were artificially filled in by a process of interpolating and zero-mean padding. A relative time encoding feature was also created to help the predictor interpret the amount of time between bursts of data. Decimation was performed to maintain the sequence length for a window while simultaneously increasing the duration of time that it represented. A jointly optimal predictor was determined as (D, N, P, S) = (8, 644616, 250, S/8) where D represents the decimation factor, N represents the number of sequences used in training, P represents the window length, and S represents the stride. It was found that there exists an approximate duration of time, roughly equal to 2120 samples, that results in the best performance for this classification.

15. SUBJECT TERMS

time series classification

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
Unclassified	Unclassified	Unclassified	SAR	24

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Christopher Ebersole	

Sequential learning with very non-uniformly sampled time series

M.S. Final Project Report

Submitted by:

Michael Gregg

Master of Science in Electrical and Computer Engineering

The Ohio State University, Columbus Oh

Advised by:

Professor Philip Schniter

Department of Electrical and Computer Engineering

The Ohio State University, Columbus Oh

Table of Contents

Abstract

Time series classification problems implement supervised machine learning techniques to analyze temporally ordered data and classify new sequential data. Time series classification has grown in popularity as access to time series data has increased in recent years, and the problems have appeared across a wide spectrum of applications such as audio recordings, medical signals, and weather prediction. Generally, an assumption is made that the temporal ordering is uniformly or close to uniformly sampled. However, there are important applications where this is not the case.

This project looked at a dataset that was a very non-uniformly sampled time series with the task of classification of three labels. The dataset was also quite large and required very high dimensional features. These considerations encouraged the use of sequential learning techniques. Sequential learning refers to machine learning models that have sequences of data as the input or output. The goal of this project was to identify pre-processing techniques and approaches for generating sequences that would be helpful for this classification task. If successful, the results could help give insights to similar sequential learning problems.

The data were first standardized over the entire dataset. The data as given had large gaps of time where no samples resided, called "dead" zones, that were artificially filled in by a process of interpolating and zero-mean padding. A relative time encoding feature was also created to help the predictor interpret the amount of time between bursts of data. Decimation was performed to maintain the sequence length for a window while simultaneously increasing the duration of time that it represented.

A jointly optimal predictor was determined as (D, N, P, S) = (8, 644616, 250, S/8) where D represents the decimation factor, N represents the number of sequences used in training, P represents the window length, and S represents the stride. It was found that there exists an approximate duration of time, roughly equal to 2120 samples, that results in the best performance for this classification.

Introduction

Time-series classification problems are common tasks for modern machine learning techniques and have applications across a variety of subjects. Frequently these problems have data that is uniformly or close to uniformly sampled in time. However, non-uniformly sampled data also has important applications. In this project, a very non-uniformly sampled dataset was considered. The main goal of this project was to learn how to construct useful data from a software that outputs very non-uniformly sampled data for effective classification by implementing sequential learning techniques.

It will be useful to define some terminology before proceeding:

Dataset terminology

- **Scenario**: A full-duration output from the software. It is comprised of several scenario-sequences.
- **Scenario-sequence**: The scenario through the lens of one of four data-channels given one of three labels. A more detailed description of these labels is given in the detailed dataset section.
- **Duration (s)**: The length of time within a given scenario.
- **Window:** The number of samples collected in a single instance. The window was shifted through an entire scenario-sequence to create part of the dataset.
- **Stride:** The number of samples that the window was shifted between adjacent instances of data preparation.

This document will proceed as follows: first, the methodology section will outline the technical approach, highlight important aspects of the data, describe the multiple pre-processing steps that were taken, and delineate the experiments. Then, the results and discussion section will consider the implications of the experiments and consider avenues for future extensions of this project. Finally, the main takeaways will be reviewed in the conclusion section.

Methodology

This project began with constructing the data from the software that was given. There were many peculiar aspects of the software output, and many techniques were used to manipulate the data to best fit the classification task. All the data processing and experiments were performed in python.

Detailed Dataset

The scenario-sequences output four synthetic data-channels that contain several features. One of three labels would be manually selected before running the software and thereby assigned to the entire scenario-sequence. This would affect the behavior of the data in a known way.

Label explanation

- **Low**: Had the most predictable pattern without any randomness.
- **Medium**: Implemented a pseudo-random pattern, where most of the pattern was random, but was repeated over time.
- **High**: Used a completely random pattern.

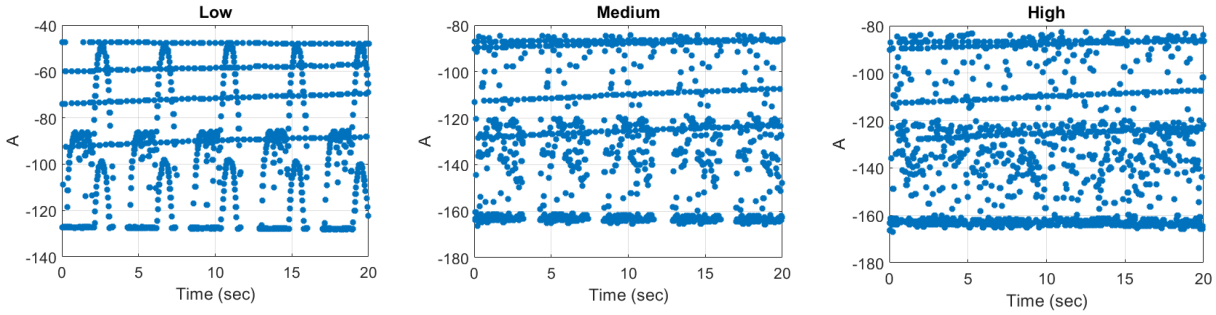

Figure 1: Label examples. Each of these figures show a typical pattern for a distinct label and each were made from the same scenario. The leftmost image corresponds to low label, the middle figure corresponds to the medium label, and the rightmost figure corresponds to the high label.

Data Preprocessing

Data preprocessing is crucial for applications of machine learning. This step improves the quality of the data such that useful information can be more readily learned by predictors. The data used in this project contained several interesting problems to consider when preparing the data for training, and several methods were employed to handle these complications.

Standardization

The features of the data varied significantly from one another, so the data were standardized. In particular, column-wise standardization over the features for the entire dataset was performed before any sequences were constructed.

It is worth noting that this technique might have resulted in look-ahead bias. The section sequence generation will describe how the data was segmented into smaller pieces, and this is where the potential snag with look-ahead-bias stemmed from. Consider a sequence that was generated in the first quarter of the scenario-sequence. That sequence will have the mean and variance of its features determined by values outside of that sequence since the data were standardized over the entire set. This might give the model some insights about the future of the data, thereby biasing the model.

There was some consideration into accounting for the look-ahead bias by standardizing over individual sequences instead of over the entire dataset. This idea would follow from image processing tasks where images are typically standardized individually. There was not enough time to include an experiment delving into this idea for this project, but it would make for an interesting experiment in the future.

Interpolation and Zero Padding

The output data consisted of samples in short and intermittent bursts. These bursts were always followed by a period of time that contained no samples which will be

referred to as “dead” periods. This pattern repeated for the entire duration of the scenario and was present for every feature.

The idea for using a sliding window seemed natural. Scenarios contained tens of thousands of samples, and most machine learning techniques for time series would struggle with memory issues for a sequence of that length. This could be handled by using a sliding window to slice up the scenarios. However, this created a new issue. Any sequence constructed from a window would not be able to represent the dead periods in time. This was because the window was a fixed number of samples and would be shifted by the stride which was a fixed number of samples, meaning that the end of one burst of data would be followed immediately by the beginning of the next burst of data in a window. Typically, deep learning networks expect evenly or almost-evenly spaced samples. The resulting temporal spacing would likely be problematic for deep learning frameworks.

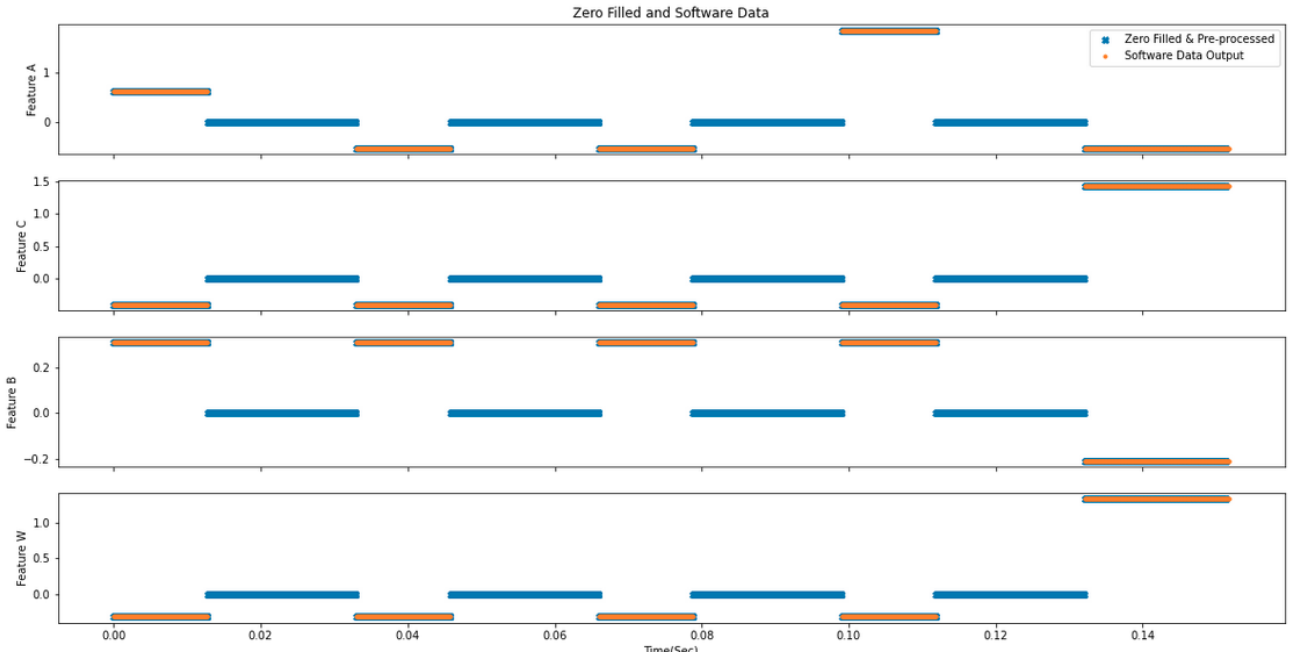

Figure 2: Interpolation and Zero Padding example

Values of zero (i.e., the mean) were padded to fill in the dead zones with a temporal spacing determined by calculating the average sampling rate between the burst of samples before and after the dead zone. It was suspected that doing this would help the models learn that it was in a transitional state between two bursts of relevant information. The outcome of this step can be seen in figure 2 for all features. The orange lines show how the data was spaced before any processing. The blue lines are the zero-mean values that were added during this stage. The final data that the models were presented with was the combination of the orange and blue data.

Time Encoding

There were still concerns that the models would have difficulty interpreting the relative time between bursts of samples. This motivated the creation of a relative time encoding feature which was constructed per sequence (the section sequence generation will describe the details of what constitutes a sequence.) For the time-encoding process, the last sample of the window was considered to be the "current" time step, and all prior samples were considered to be in the past. The index of the "current" time step was subtracted from the indices of all other time steps individually to form a vector that starts from a negative number and linearly increases to 0.

Relative Time Encoding Example

Consider a sequence with the time indices $[t_1, t_2, t_3, t_4, t_5]$. The relative time encoding feature would be constructing by taking the "current" time step index and subtracting it from the time index of all the other time steps. The resulting relative time encoding feature would be t = [-4, -3, -2, -1, 0].

Decimation

The goal of decimation for this project was to maintain sequence length while increasing the duration of time that a sequence represented. In doing so, the duration of a scenario could be extended without having to weaken the computational performance. For example, consider a window of 750 samples with 5 features. Having a decimation factor D=4 would approximately represent the same amount of time as a window of 3000 samples with the same 5 features without decimation. However,

the D = 4 sequence would have 5 * 750 = 3750 inputs, and the sequence without decimation would have 5 * 3000 = 15,000 inputs which increases the computational complexity by four. Of course, the decimation process also effectively throws away a relative portion of the data, so that was taken into consideration as well.

Scipy.signal.decimate was used to perform the decimation. First, high frequencies were filtered out with a low-pass filter to prevent aliasing. In particular, the ftype='iir' was used which employs a Chebyshev type I filter. The decimation was then applied to each feature.

Sequence Generation

Scenarios were longer than what was generally expected by machine learning techniques. A typical length for a scenario with a duration of 20 seconds was approximately 150,000 samples. This motivated the use of a sliding window to slice scenarios into more manageable pieces. After a window of data had been segmented, the window was moved forward in time through the dataset by an integer valued number of samples called the stride. After the dataset had been fully segmented, the resultant sequences were then stacked to use as inputs for the predictors.

Generalized Univariate Sequence Generation Example

Consider the following time-series: $t_1, t_2, \ldots, t_{N-1}, t_N$, where t_i is the i^{th} time sample. A window of five samples with a stride of one sample would result in the following sequences: $x_1=[t_1, t_2, t_3, t_4, t_5]$, $x_2=[t_2, t_3, t_4, t_5, t_6]$, …, $x_{N-4}=[t_{N-4}, t_{N-3}, t_{N-2}, t_{N-1}, t_N]$ where x_i is the i^{th} generated sequence. These sequences would then be given the proper corresponding label y_i.

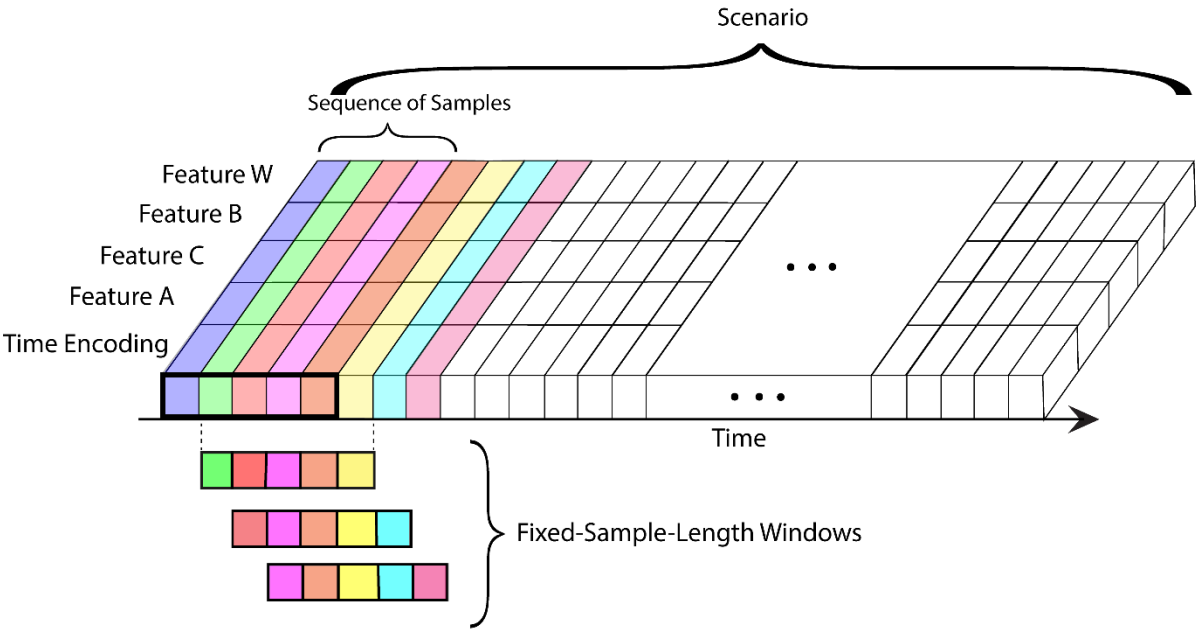

Figure 3: Sequence Generation Visualization

Figure 3 shows how this idea was extended to the multivariate case. Every sample now included information on features A, B, C, W, and the time encoding. This is shown by the connected blocks that are sharing the same color. A sequence was still made with a sliding window of a fixed number of samples, but the result was now a multivariate structure. A fully processed sequence with a window of 250 samples can be seen in figure 4.

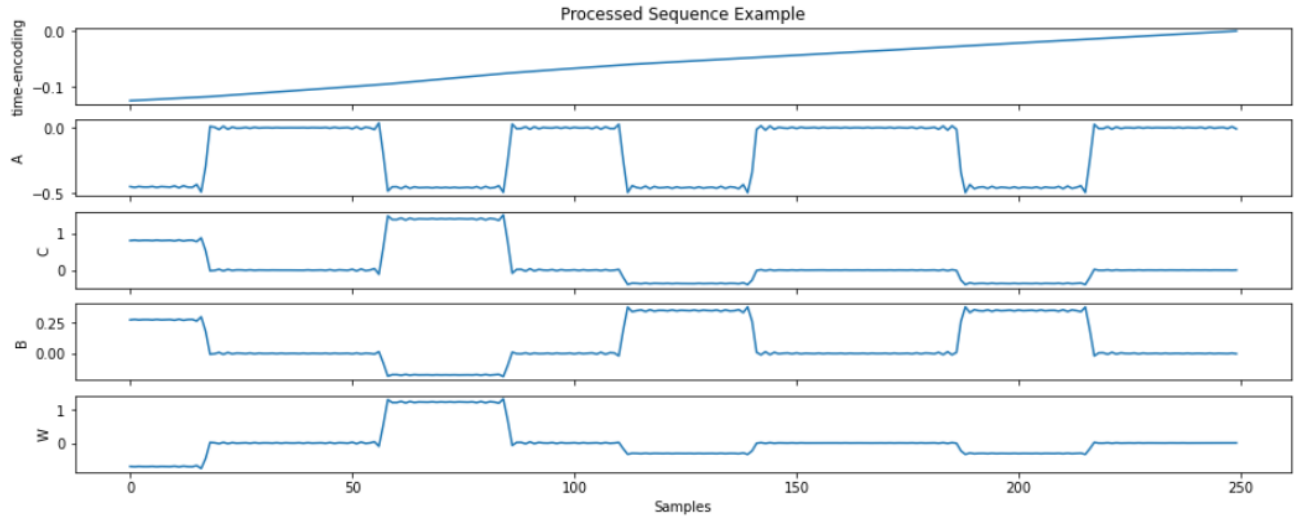

Figure 4: Processed Sequence Example

Experiments

Decimation Factor and Model

The dataset necessary for this classification problem proved to be quite large. Without decimation, a window of 750 samples represented approximately 38 milliseconds, and it was conjectured that much longer durations would be necessary for this task. Therefore, determining if the duration of the scenario could be lengthened without significantly increasing the amount of data through decimation was important. Additionally, this experiment was performed with several predictors to get an idea of which seemed promising. The details of the experiment were as follows:

Models used:

- Random Forest
- XGBoost
- GRU
    - Hidden states: 16
- Time Series Transformer
    - Attention heads: 8
    - Attention dimension = 128
    - GELU activation in feed forward network
    - 128 hidden units in feed forward network

The following parameters were used:

- Decimation factors: [1, 2, 4, 8, 16, 32]

Training and testing details:

- Training set: ~200,000 sequences
- Test sets: ~20,000 sequences each
- Deep learning models were trained for 20 epochs
- Optimizer: Adam
- Batch size of 64 for TST, 512 for GRU

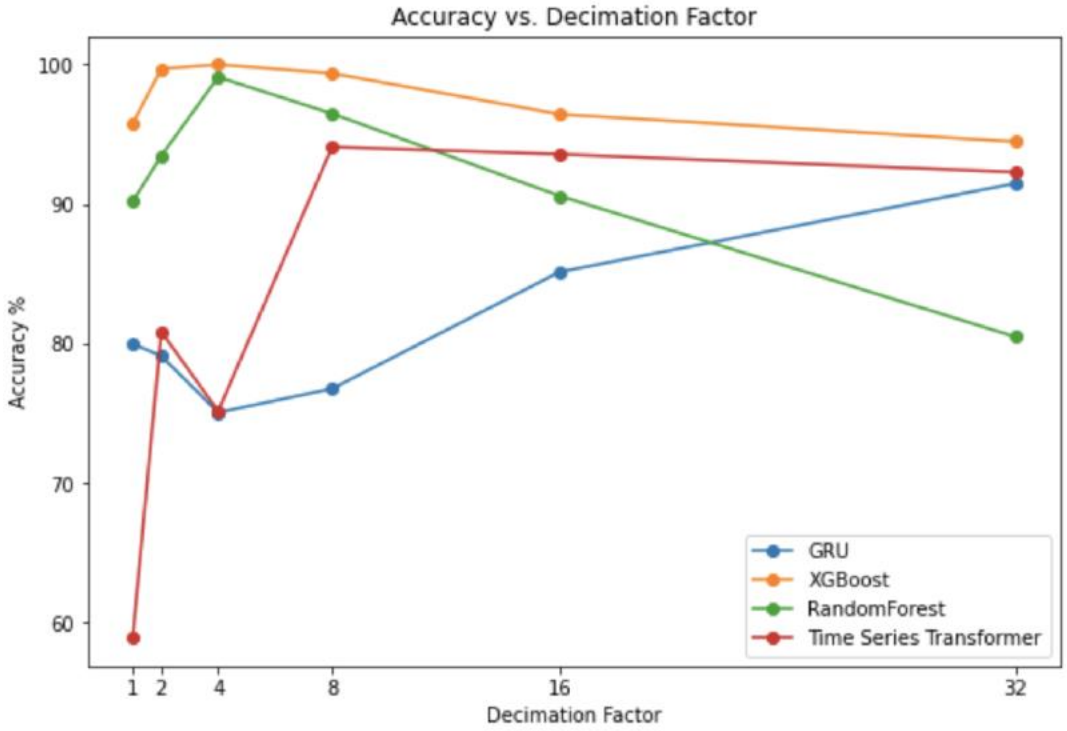

Figure 5: Decimation Factor and Model Plot

The results from this experiment can be seen in figure 5. The GRU struggled with this classification. It was suspected that the GRU was not tuned optimally, but later attempts made to tune the GRU did not improve things drastically. Similarly, it was likely that the time-series transformer could have been tuned better. The transformer was computationally expensive, was quite a complicated model, and was already pushing the computational resources that were available at the time to achieve the accuracy shown. It was hard to justify attempts to achieve better results with the GRU or the TST when the tree methods were performing well out of the box.

XGBoost outperformed all other models for every decimation factor. Additionally, both decision-tree predictors performed well for small decimation factors, and both showed similar trends in performance overall. Both start with decent performance for D=1, increase to their peak performance at D=4, and then monotonically decrease in performance for all higher decimation factors.

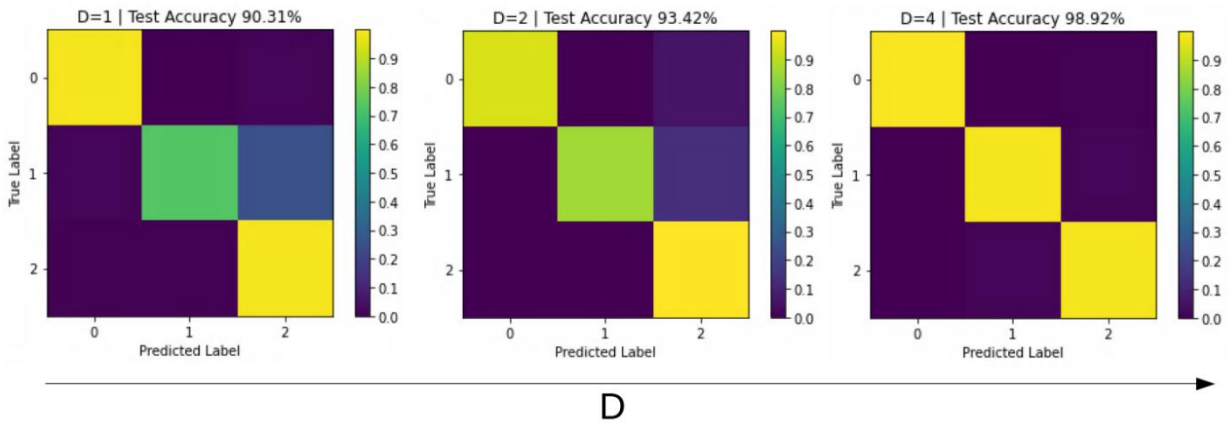

Figure 6: Decimation factor trends for the random forest classifier

Figure 6 shows confusion matrices for the random forest classifier for decimation factors 1, 2, and 4. The random forest classifier was chosen to illustrate this since the trend was more pronounced than it was for XGBoost, even though XGBoost performed better overall. This figure demonstrates that the predictors had difficulties distinguishing between the medium and high labels (labels 1 and 2 in this figure, respectively). This can be explained by observing figure 1 which shows that the medium and high labels were visually much more similar to one another than either label was to the low label.

The results of this experiment should be taken with a grain of salt. There was a constraint in the experimental setup that was later determined to be unfair for the experiment. This constraint was that each training set would have approximately 200,000 sequences and that each test set would have approximately 20,000 sequences. The original idea was that trying to hold the number of sequences constant would allow for a fairer evaluation and comparison between the different amounts of decimation. This was achieved by using proportionately more scenarios for different decimation factors. If D=2 resulted in half of the scenario data being thrown away, then achieving approximately the same number of constructed sequences as D=1 could be achieved by using twice as many scenarios. This approach went against the formalization of most machine learning problems. Typically, a set amount of data would be given, and the goal would be to complete the task as well

as possible with only that amount of data. Therefore, the following experiments were conducted while adhering to that constraint.

Goal: Joint optimization of the predictor (D, N, P, S)

In this experiment, a set amount of data was considered. The details of that dataset are as follows:

Training set:

- 40 scenarios
    - 480 unique scenario-sequences (40 scenarios * 3 labels * 4 data-channels)

Testing set:

- 32 scenarios
    - 384 unique scenario-sequences (32 scenarios * 3 labels * 4 data-channels)

The task now was to determine the optimal processing parameters given this data. The model used for this experiment was XGBoost with 'gpuhist' to make use of the XGBoost fast histogram algorithm. XGBoost was chosen for simplicity and because it performed better than all other algorithms in the first experiment.

Problem Formalization

The experiment was formalized as follows:

Given:

- **M:** number of scenario sequences
- **L:** length of each scenario sequence

Jointly optimize:

- **D:** decimation factor

- **N:** number of sequences used for training
- **P:** length of a window
- **S:** stride

Under the following constraints:

$$S \geq 1, integer \ (1)$$

$$D \geq 1, integer \ (2)$$

$$\frac{L}{D} \geq \frac{N}{M-1} * S + P \ (3)$$

Constraint (3) prevents the sequences from looking beyond the last sample of the scenario sequence since $\frac{L}{D}$ represents the length of a scenario after decimation and $\frac{N}{M}$ represents the number of window sequences extracted from each scenario sequence.

Window length and Decimation factor

The goal of this experiment was to determine the optimal tradeoff between window length and decimation factor. For this experiment, the stride was held constant as one half of the window length to correspond with common spectrogram conventions. Additionally, the number of sequences was tied to the decimation factor which meant that this setup allowed for examining the tradeoff between window length and decimation. Both the training and testing sets were decimated with the same factor for each D value.

The following parameters were used:

- D factors: [1, 2, 4, 8, 16, 32]
- P values: [250, 500, 750, 1250, 1500]

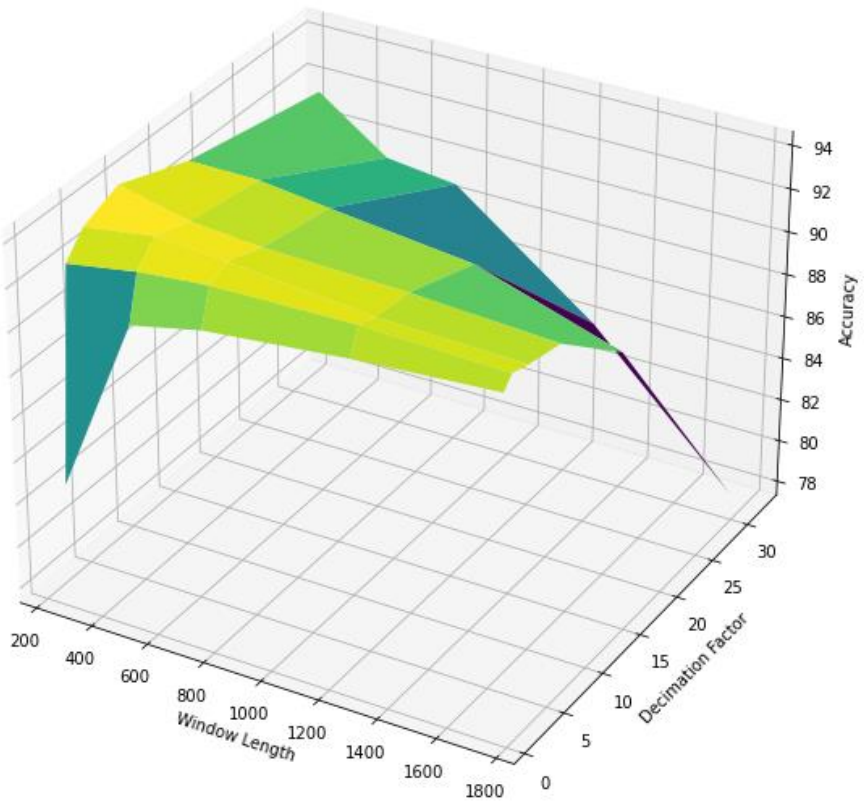

Figure 7: Window length vs. decimation factor surface plot

The two highest peaks of this surface plot occur at (D, P) = (8, 250) and (D, P) = (4, 750) with accuracies of 94.29% and 94.08%, respectively. The results of this suggested that further exploration of this tradeoff would be necessary. Figure 7 shows that the peak of this surface was on the edge of the plot with the smallest P values. This motivated further exploration with smaller window lengths. Additionally, D = 32 was performing poorly enough to justify omitting it from further experiments.

The following parameters were used for a second experiment:

- D factors: [1, 2, 4, 8, 16]
- P values: [50, 100, 200, 250, 400, 500, 750, 1250, 1500]

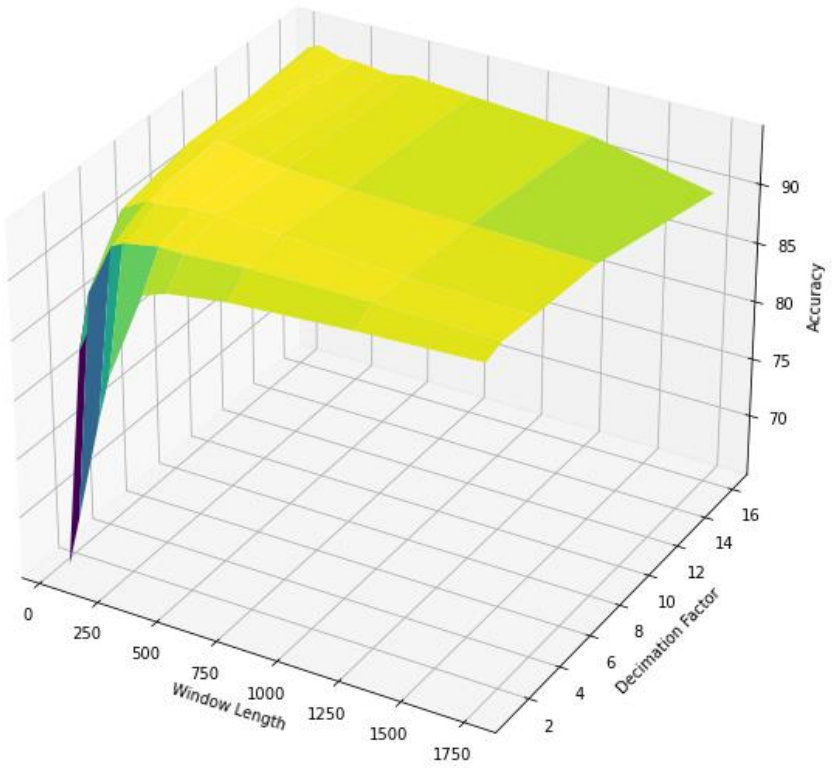

Figure 8: Window length vs. decimation factor surface plot -further exploration

Figure 8 shows the updated surface plot. The highest peaks have not changed, but the plot now shows a ridge with a shallow peak on top. That peak corresponds to (D, P) = (8, 250). However, an interesting observation can be seen if the peak value per decimation level is observed.

```
3
4  Axg1 =  [65.36,69.69,78.69,82.64,90.28,91.01,91.73,92.39,92.85]
5  Axg2 =  [82.44,87.66,92.00,92.55,93.12,93.10,93.41,93.55,93.37]
6  Axg4 =  [86.62,92.32,93.48,93.58,94.02,94.08,93.86,93.27,93.10]
7  Axg8 =  [92.13,93.62,94.11,94.29,93.76,93.53,93.26,93.09,92.72]
8  Axg16 = [93.15,93.57,92.94,93.02,92.64,93.01,92.61,91.88,89.76]
9
```

Figure 9: Accuracies for window length vs decimation surface plot

Figure 9 shows the accuracies for every point on the surface plot in figure 8. The notation "Axg1" represents the accuracy of XGboost for D =1, "Axg2" represents the accuracy of XGboost for D=2, etc. The columns represent the P values used in this experiment with the leftmost column representing P=50 and the rightmost column representing P = 1500. The peak accuracy for each decimation factor was highlighted

revealing a trend that suggests that the larger the decimation factor, the smaller the optimal window length needs to be to achieve the highest accuracy.

Stride Experiment

Now that an optimal pair of (D, P) had been identified, the goal was to find the corresponding jointly optimal stride value. The window length and decimation factor experiment had used the spectrogram convention of S = P/2, so the other stride values were also defined in terms of the window length.

The following parameters were used:

- S values: $[\frac{P}{32}, \frac{P}{16}, \frac{P}{8}, \frac{P}{4}, \frac{P}{2}, \frac{2P}{3}, P]$ = [8, 16, 31, 62, 125, 167, 250]

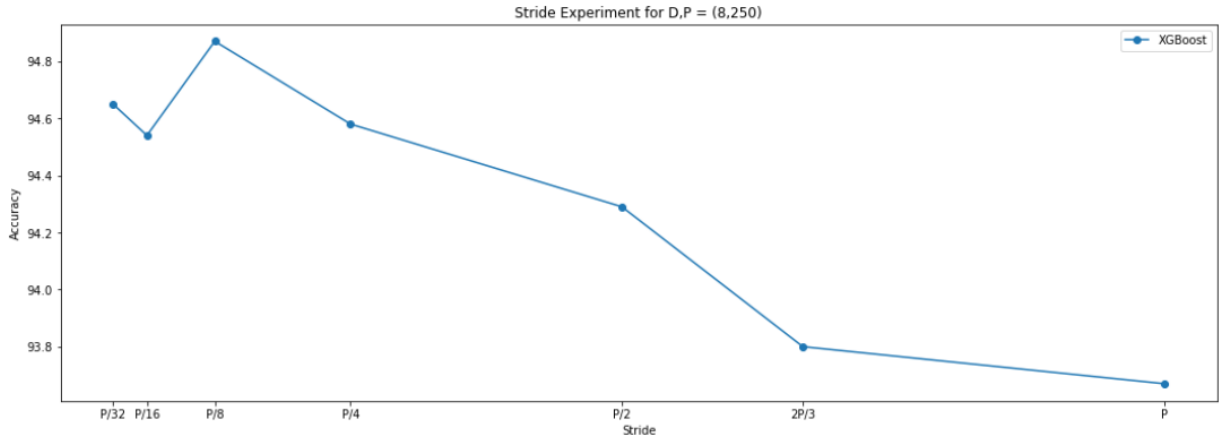

Figure 10: Stride experiment for (D, P) = (8,250)

Figure 10 shows that the spectrogram convention can be improved upon for the task. Strides that were longer than P/2 performed exclusively worse than S = P/2, and strides that were shorter than P/2 performed exclusively better than S=P/2. The highest performing S value was S = P/8 with an accuracy of 94.87%.

Results and Discussion

The results from the window length and decimation experiment shown in figure 9 suggest that there is an approximately optimal time duration for classification, and that there are several different ways to reach it through the joint selection of D and P. This can be seen by considering the highest performing P value for every D.

Table 1: Best window length for every decimation factor

D	P_{best}	$D*P_{best}$	Avg $D*P_{best}$
1	1500	1500	2120
2	1250	2500	
4	750	3000	
8	250	2000	
16	100	1600	

The multiplication of D and P represents the time duration of a given sequence. Table 1 shows the window length for every decimation factor that resulted in the highest performance. Table 1 also shows those two values multiplied together to show the approximate corresponding time duration, as well as the average of all five. This analysis suggests that 2120 samples is representative of an optimal time duration for a sequence to represent for the given data. The correspondence is only approximate due to the quantization of the grids used for (D, P). Additionally, since the dataset is very non-uniformly sampled in time, there is not a direct correspondence to the amount of time that takes place between two adjacent samples. Even so, this is still an interesting insight about the data that could be explored further for future work.

Figure 10 suggested that strides shorter than P/2 perform better for the given data than S = P/2. This makes sense intuitively as decreasing the stride means that there would be more sequences for training. What is more interesting is that there was a point where the decreasing the stride hurt the performance. The peak accuracy occurred at S=P/8 while both S=P/16 and S=P/32 performed worse. This might

suggest that adding too much redundancy in the sequences harms the predictor's ability to classify. Though it might also simply suggest that there are diminishing returns for decreasing the stride. That is still useful information to have since decreasing the stride increases the computational expenses, which should be avoided if the performance is not increasing as well. With the optimal stride identified, the results of these experiments suggested that the optimal predictor for this data is (D, N, P, S) = (8, 644616, 250, S/8) = (8, 644616, 250, 31).

Future Work

As previously mentioned, an experiment to determine the optimal data standardization technique might be fruitful. This project only explored standardization over the entire dataset, but standardization over an individual window or over the bursts of data might be interesting avenues for future experiments.

This paper suggested the existence of an optimal number of samples for a sequence to represent, but that representation was only approximate due to the quantization of the (D, P) grid. Future work could increase the resolution of that grid to get a stronger understanding of what that optimal time duration truly is.

This project did not do extensive work to fine-tune every predictor that was used, and most of this project was about exploring the manipulation of the dataset as opposed to determining which type of predictors work best. Experiments that implement and improve on other predictors such as GRU or TST would be interesting.

Conclusion

Time-series classification defines a series of problems that use supervised machine learning techniques to classify temporally ordered data and have manifested in a variety of applications. These problems typically assume that the data are uniformly or close to uniformly sampled in time. This project analyzed a set of very non-uniformly sampled data through the implementation of sequential learning methods and preprocessing techniques such as standardization, interpolation, zero padding,

decimation, and relative time encoding. The experiments suggest that there exists a time duration equivalent to approximately 2120 samples for a sequence that results in the highest performance, and that the choice of (D, N, P, S) = (8, 644616, 250, S/8) was the jointly optimal predictor for the task.

References

1. Scipy.signal.decimate documentation https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.decimate.html
2. Time series transformer documentation https://timeseriesai.github.io/tsai/models.TST.html
3. XGBoost documentation https://xgboost.readthedocs.io/en/stable/
4. Random forest classifier documentation https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
5. GRU documentation https://pytorch.org/docs/stable/generated/torch.nn.GRU.html