Machine Learning for Aviation Data

A Thesis Submitted to the Committee on Graduate Studies
in Partial Fulfillment of the Requirements for the Degree of Master of Science
in the Faculty of Arts and Science

TRENT UNIVERSITY

Peterborough, Ontario, Canada

Applied Modelling and Quantitative Methods M.Sc. Graduate Program

May 2022

Abstract

Machine Learning for Aviation Data

Yang Meng

This thesis is part of an industry project which collaborates with an aviation technology company on pilot performance assessment. In this project, we propose utilizing the pilots' training data to develop a model that can recognize the pilots' activity patterns for evaluation. The data will present as a time series, representing a pilot's actions during maneuvers. In this thesis, the main contribution is focusing on a multivariate time series dataset, including preprocessing and transformation. The main difficulties in time series classification is the data sequence of the time dimension. In this thesis, I developed an algorithm which formats time series data into equal length data.

Three classification and two transformation methods were used. In total, there are six models for comparison. The initial accuracy was 40%. By optimization through resampling, we increased the accuracy to 60%.

Keywords: Multivariate Time Series Classification, K-NN, Time Series Forest, Machine Learning, Data Mining

Acknowledgements

I wish to thank various people for their contributions to this thesis. Without you all, it would not be possible for me to finish this thesis.

First of all, my family, who respected my choice and supported me without hesitation. Thank you for everything.

The CAE Inc. - the best team I have been lucky enough to join. I could not conduct the research smoothly without your great effort on the guiding and mentoring. Jean-François Delisle and Laurent Desmet - working with you was fun and thanks for helping for my research and daily life.

I would like to thank my supervisor Dr. Sabine McConell and Dr. Richard Hurley. Thank you for providing me such valuable opportunity to work alongside both of you. Your suggestions and encouragement supported me throughout the thesis study. Finally, thank you to Trent University and Mitacs for financial support. Without funding I could not afford my life during my research process.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

The study of time series data was ubiquitous in all aspects of life. There are many examples of data around us, such as stock prices, voice tracking, electrocardiogram analysis and so on. Machine Learning can be applied to the field of time series for research and analysis, for example for human motion tracking analysis. In 2003, Koegh collected the actor's hand position movement data to classify two classes – with gun or without gun [1].

Due to its broad application prospects and flexibility, the *Time Series Retrieval* problem had sparked interest in the field of data mining. In the early 1990s, many researches focused on this field. For example, R. Agrawal proposed a methodology for time series similarity search [4]. Since then, many well-known research communities contributed to Time Series Data Mining, such as groups at California University and University of East Anglia [3].

In terms of time series classification research, most work focused on *Uni-*

variate Time Series Classification (UTSC [5]). However, with the popularity of mobile electronic equipment and the needs of industry, the need for *Multivariate Time Series Classification* (MTSC) became increasingly prominent. It can be utilized in many fields, such as judging people's movements by capturing actions for example. In 2016, a research group performed four activities while wearing a smart watch. The watch collected 3D accelerometer and a 3D gyroscope data [3].

With the achievements in MTSC, more innovation progressed in the field of aviation as well. According to Bryan et al [6], data mining techniques were utilized in aviation safety, specifically for anomaly detection. The most recent project development was ROCKET [7], in which researchers utilized random convolutional kernels for time series classification. The research was funded by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development for aviation purposes.

1.2 Objects

A time series is a series of sampled values of a set of variables at consecutive time points. The d dimensional time series of length n is denoted as

$$x_i(t) : [i = 1, 2, ..., d; t = 1, 2, ..., n] \tag{1.1}$$

Where, i denotes dimension and t denotes the time. If $d = 1$, then we can categorize the problem as *Univariate Time Series* (UTS). In contrast, if $d >1$, we define it as *Multivariate Time Series* (MTS). We can also use a matrix to

represent a MTS:

$$\begin{bmatrix} X_{1,1} & ... & X_{1,n} \\ : & : & : \\ : & : & : \\ X_{d,1} & ... & X_{d,n} \end{bmatrix} \tag{1.2}$$

There were many classic example data sets of univariate time series. The most widely utilized one was GunPoint [1]. In this data set, researchers used sensors to track hand position for each time point and generate the record for two classes. The data can be used for supervised machine learning.

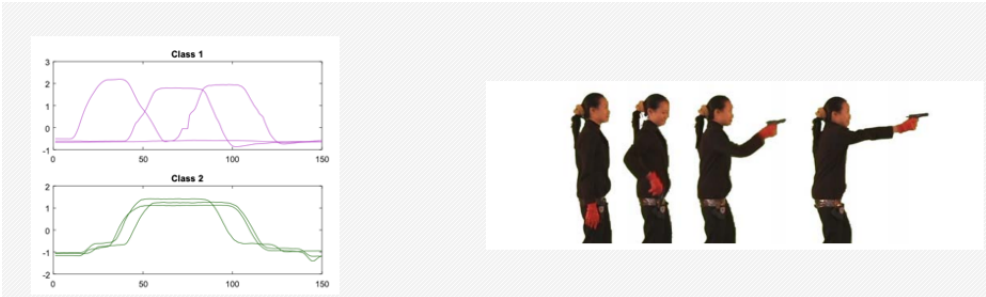


Figure 1.1: Time Series Data Set for GunPoint [1]
The left graph shows the hand position data with gun and without gun. The right graph shows the person's real movement

In Figure 1.1, time series was not just a list of pure data, the relation between each time point cannot be ignored. This feature made many statistical methods for studying variance unsuitable for time series.

In addition to this, time series also posed many problems that other numerical data did not have, such as how to measure the similarity between two series. Unlike numerical values that can be compared by mathematical algorithms, time series data can have a delay or a transfer for each time point. Based on the above problem, researchers proposed a more effective solution for

one-dimensional time series. In 2001, Keogh proposed a methodology named *Dynamic Time Warping* (DTW) [5], which improved the accuracy of time series analysis by reducing the effect of time delay and warping.

Researchers tried to expand to multi-dimensional time series based on one-dimensional time series data mining approaches. However, there are many problems and challenges still in need of exploration.

- Time series data were always collected with noise and redundancy. For example, the sound wave had pause, delay, and unclear pronunciation in the voice tracking;
- Multi-dimensional caused an increase in the amount of data, and time series cannot reduce the dimensionality using traditional approaches. The Curse of Dimensionality was very prominent. If the dimensionality was reduced, the multi-dimensional time series structure were changed, and some information are missing;

These problems appeared repeatedly in this thesis research and became the main focus. Since the project was focused on aviation training data, there were multiple dimensions with a high volume of records. In addition, the time series data had various lengths, which caused another problem.

1.3 Thesis Outline

In this thesis, I discuss my research in Chapter 3 to Chapter 5. Chapter 2 mainly focused on the background, relevant terminology, algorithms and the concepts which were used in the thesis. It explored the state-of-art techniques that were used in time series classification for univariate and multivariate time

series. The concepts gave insight into distance measures such as *Euclidean Distance* (EE) and *Dynamic Time Warping* (DTW). The techniques already developed by previous research were explored as well, such as *HIVE-COTE model* [8], *TS-CHIEF* model [9], *ROCEKT* [7] and the best achievement of those techniques. After introducing the practices, I discussed their advantages and disadvantages and comparisons. Some of the methods achieved high accuracy in certain data sets, however, they had limitations dealing with certain problems as well.

Following the background introduction, I introduced the data sets used in this research in Chapter 3. This part began with the meta data of the data sets from the industry partner. Followed by the general introduction, I discussed the preprocessing part, which included restructuring of the data and summary statistics. In this part, I expanded the discussion of difficulties of multivariate time series classification, such as the *dimensionality curve*. Another difficulty to modeling with the data set was using *Principal Component Analysis* (PCA) to reduce the features. I discussed how to transfer each time series data set to a list and store it in a column.

Chapter 4 discussed two models used in this thesis. The first one is *K Nearest Neighbor* (K-NN), and the second one is *Decision Tree* based classification method.

After talking about the models and techniques that were utilized, results were presented using the previous techniques in Chapter 5. Final results were not ideal because of low accuracy and computational complexity. To improve the model performance, ensemble methods were used.

Chapter 6 was the conclusion for the whole thesis which discusses the

achievements and limitations in the experiment.

1.4 Summary

In this chapter, I described the structure of the thesis, and clarified the main target output of the project. A brief summary of each chapter is included as well. In the following chapters, I will discuss each topic in more depth.

Chapter 2

Background

There was a long history of research and applications for time series data. In early years, time series research focused mainly on statistical modeling and forecasting. In this period, time series was always considered as a linear function with the dependent variable *time*.

With the development of machine learning techniques, researchers began to pay more attention to the overall meaning of the time series rather than just the continuity of time points. More research topics were beginning to flourish, such as identifying the patterns, preprocessing with noise, time window selection and describing the process, etc. Another important direction in this field was the measurement of time series similarity. Unlike numeric values that can be compared directly, there were many warping and alignment issues in the comparison of two time series. In the early 2000s, time series research was expanded to multidimensional classification and clustering.

The subsection is based on the development of time series classification: from *Univariate Time Series Classification* to *Multivariate Time Series Clas-*

sification. Following those sections, several subsections are discussed, such as *Similarity-based Techniques* and *Interval-based Technique*. The *Similarity Measures* is another topic which is important in time series classification.

2.1 Univariate Time Series Classification

Research in *Univariate Time Series Classification* was based on the traditional time series or statistic modeling of serial data. Given the structure of a time series data set, it was always considered for linear regression. The original research focused on time series forecasting, such as stock price prediction [10], the analysis of heart rate [11], voltage disturbances analysis [12] and so on.

Unlike statistical models, machine learning for time series extended the target of the time series analysis. Since Keogh [13] utilized an algorithm to describe the distance between two samples and classified them, more difficult problems required classification of large quantities of time series data. The most recent research in this field included human activity recognition [14], health and medical data from electrocardiograms (ECG) [15] and electric device identification [16]... University of California Riverside (UCR) compiled a TSC dataset as sample data. The algorithm which was first used with TSC dataset was 1-NN. In the early research period, the distance between test sample and training sample time series decided the class.

Time Series Classification focused on prediction of a label $y \in \{1, ...c\}$, where c was the number of target classes. For univariate time series classification, there was only one dimension that can be represented as $T = (X_1, X_2, ...X_i)$

2.1.1 Similarity-based Techniques

Similarity-based Techniques were based on the distance between pairs of time series. These usually use *1-Nearest Neighbour techniques* (1-NN) [13] with distance measurement and with various warping window size selection methods [17]. Time series warping was the most crucial problem for modeling. Warping means the similarity between two temporal sequences may vary in speed. For example, similarities in walking could be a scenario of time series warping, a person may walk faster or slower, and movement of data by a certain time point were different. How to measure the time series with the effect of speed became the priority task. In the next subsection, similarity measurement was discussed in more depth with how it affected the model accuracy and efficiency. Since *Similarity-based Techniques* were the most direct ways for humans to understand, there were many developments in this field during the last two decades, including *WeightedDTW* (WDTW) [18], *Weighted DDTW* (WDDTW) [18], and measures based on subsequences, such as *Move-Split-Merge* (MSM) and *Longest Common Subsequence* (LCSS) [19].

Ensemble methods were also developed to optimize 1-NN classifiers. Bagnall et al used multiple 1-NN classifiers with diverse similarity measures [16], the overall best performance achieved by Bagnall was more than 95%. The ensembles benefited classifications by reducing the variance of the model and improving overall accuracy.

The advantages of *Similarity-based Techniques* were obvious. Since the technique only considered the distance directly, it was the simplest and easiest algorithm to understand and compute. However, the disadvantages cannot be ignored. It was slow during training: if leave-one-out cross-validation was used

in evaluation, the time complexity for n time series which has m time points for each is $O(m^2 * n^2)$ [20].

2.1.2 Interval-based Techniques

Unlike the distance or *Similarity-based Techniques* that considered every time point, *Interval-based Techniques* focused on groups of time points. These algorithms separated the entire series into a set of intervals and applied transformations to these intervals to obtain a new feature vector. The new vector can be treated as a categorical variable and used to train a traditional machine learning algorithm, such as a *Decision Tree* and *Random Forest*.

There were many applications that utilized these techniques. *Time Series Forest* (TSF) [21] applied the mean, standard deviation and slope to transfer a set of randomly chosen intervals, and then trained a *Decision Tree* by using the new feature vector. The algorithm was repeated to learn an ensemble model. Other algorithms utilized interval-based techniques was *Random Interval Spectral Ensemble* (RISE) [22], *Time Series Bag of Features* (TSBF) [23] and *Learned Pattern Similarity* (LPS) [24].

These algorithms can overcome the disadvantage of Similarity-based Techniques. Since they decreased the time points to only k intervals, for n time series, the training complexity is $O(k * m * n^2)$ in which k is far less than m.

However, although the interval transformations decreased the time complexity dramatically, the loss of information during preprocessing was difficult to manage. The interval transformations gave up meaningful information such as some minor changes and caused a bias of the model.

2.1.3 Shapelet-based Techniques

Different from the above two approaches discussed which took the whole time series into consideration, *Shapelete-based Techniques* focused on partial data. A shapelet was an unusual pattern which can be crucial identification for each class. In another words, " shapelet-based algorithms seek to identify sub-sequences that maximally indicative of class members irrespective of where they occur in a sequence " [25].

The first version of a shapelet algorithm was developed by Ye in 2009 [25], in which the researcher enumerated all possible sub-sequences and tried to find the 'best' one. In this experiment, researchers used the *Information Gain* as criteria to assess how to split the data. After getting the 'best' shapelet, a distance threshold was used as a decision criterion for a *Decision Tree*. Although the shapelet-based technique can ignore the noise and improve the robustness in time series classification, it is still a very slow algorithm with time complexity of $O(n^2 * m^4)$, in which n is the number of time series and m is the length of each one.

Given the disadvantage of shapelet-based techniques, many approaches tried to optimize this technique to speed up the training process. Instead of enumerating all possible shapelet candidates, researchers tried to minimize the search process. These algorithms included *Fast Shapelets* (FS) [26] and *Learned Shapelets* (LS) [27].

2.1.4 Dictionary-based Techniques

Dictionary-based techniques transfered time series data into bags of words [15]. These techniques were good at dealing with time series with noise and

finding the recurring patterns. The most widely discussed method was *Bag-of-SFA-Symbols* (BOSS) [28]. The BOSS model was a structure-based similarity measure which deducted noise from the raw time series. The algorithm first defined a proper split window for each time series and transformed the sub-sequence time series to words. This process can maximize the unique patterns of the time series and ignore the noise. BOSS was faster than other algorithms for time series which had many repeated patterns.

However, the model's applicability was also very limited. It performed well for some pattern extraction, such as using a sensor to detect the surface of objects. If the time series had no repeated obvious patterns, the model was not efficient.

2.2 Similarity Measures

Since *Time Series Retrieval* gained wide attention, the discussion about time series similarity continued. There were many particularity of the data cannot be ignored. Here, it cannot be ignored that the particularity of time series data. For example, some *Time Series Retrieval* was only for a certain interval, and others needed to be considered the whole waveform. Therefore, the applications of similarity measurements varied based on the scenario.

Currently, the most mature algorithms regarding similarity measurement include *Euclidean distance* (ED), *Dynamic Time Warping* (DTW) [5], *Longest Common Sub-sequence* (LCSS) [29], *Edit Distance on Real Sequence* (EDR) [30] and *Edit Distance with Real Penalty* (ERP) [31].

2.2.1 Euclidean Distance

Euclidean Distance (ED) is the most traditional and straightforward measurement for similarity of time series. It was widely implemented in many experiments before other methods became popular. *Euclidean Distance* was the "ordinary" straight-line distance between two points in *Euclidean* space [32]. The line represented the distance between two points. It can be represented as the function (2.1):

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.1}$$

In this function, x and y were two time series, x_i and y_i represented the value at the same time point i. The approach summed each value of the distance between directly opposite points of two time series. The time consumption was also low since it only iterated one time. Figure 2.1 showed the ED calculation:

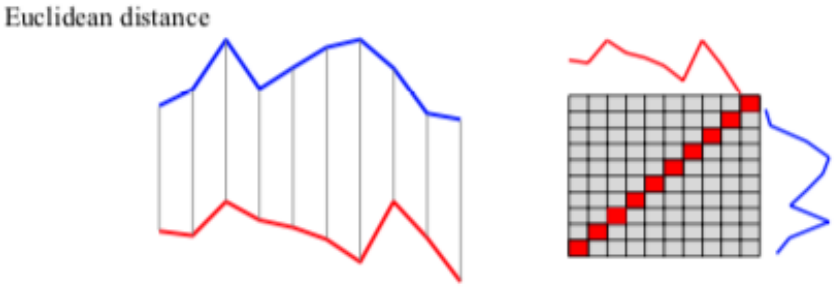


Figure 2.1: Euclidean Distance with Time Series Data Set [1]
The blue and red line represent two time series. To calculate the *ED*, the distance between each pair if opposite point are accumulated. The right graph is the visualization of the path.

There were some experiments that used this method for time series data sets. In the paper [33], K. Yang et al used *ED* and *K-means* to cluster time series

data. They achieved an outstanding performance with low time consumption. In Z. Bank's paper [34], *ED* is used to identify the similarity between two time series.

However, *ED* also had deficiencies. *ED* did not have perception of time series delay and transfer. In other words, because the distance was straight between two opposite points, any prior and late position was not considered. Many real time-series data with noise was not matched against each other precisely. To solve this difficulty, researchers proposed other measurements.

2.2.2 DTW

Dynamic time warping (DTW) was a technique to find " an optimal alignment between two given (time-dependent) sequences under certain restrictions " [13]. It warps two sequences x and y non-linearly to cope with time deformations and varying speeds in time dependent data [35].

Different than *ED* which directly summed the distance between two opposite time points, DTW considered the points value in a matrix, which was called *local cost matrix* (LCM). The matrix size (i, j) decided how many time points were considered in DTW, where the value of time point (i, j) was calculated between x_i and y_j. Next, a warping path W was determined which could be the minimum values of all distances. There were more considerations of LCM in different conditions , such as the boundary condition, continuity, and monotonicity. The total distance for a path W was obtained by summing the individual elements (distances) of the LCM that the path traverses. To obtain the DTW distance, a path with a minimum total distance was required. It can be represented by Equation 2.2

$$d_{cum}(i,j) = d(x_i, y_i) + min\left\{d_{cum}(i-i, j-1), d_{cum}(i-1, j), d_{cum}(i, j-1)\right\} \tag{2.2}$$

After obtaining each minimum distance for all time point positions, the total distance was the root of sum:

$$d_{DTW}(x,y) = min\sqrt{\sum_{k=1}^{K} w_k} \tag{2.3}$$

Figure 2.2 showed an example

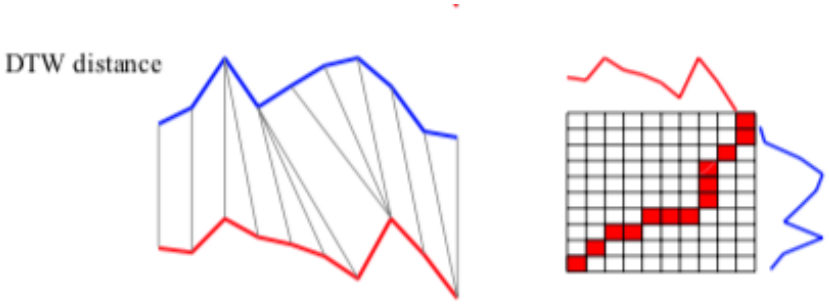


Figure 2.2: DTW with Time Series Data Set [1]
The blue and red lines represent two time series. *DTW* calculates the distance in a matrix: the points prior and after determine the most closest points. Graph on the right is the visualization of the path.

Since the strength of *DTW's* in dealing with time series delay and transfer, there were many modifications and optimizations contributed by researchers. Keogh [5] optimized and expanded the utilization of *DTW*. Keogh proposed *Derivative Dynamic Time Warping* (DDTW) in 2001, which converted the time series into a series of order differences. This avoided having a single point of one series to map to many points in another series, which can negatively impact

the DTW distance. In 2011, *Weighted Dynamic Time Warping* (WDTW) was proposed by Jeong et al. [18], which added a weight function to the LCM to increase the penalty. This optimization reduced the bias for shape matching.

However, even though DTW reduced the bias of time series transform problems, there were still many imperfections. The time and space complexity of finding the DTW distance between two time series was O (mn), where n and m are the lengths of the two input sequences.

The substantial complexity made DTW a high computational complexity process, hence different methods were proposed in the literature to speed up the distance measure. In 2007, Salvador and Chan proposed a method [36] called *FastDTW* which reduced the time complexity of DTW by considering fewer time points in time series. The algorithm first reduced the dimensions by averaging adjacent pairs of points, then found the minimum warping path on the reduced time points, and lastly calculated the distance based on the warping path. The time complexity of *FastDTW* is O $(max(m, n))$ [37], where n means the number of points in LCM.

Even though *FastDTW* already improved the efficiency compared with traditional DTW, many researchers still criticized the algorithm because of its complexity. Whether DTW was a powerful time series analysis method was still under discussion. T. Rakthanmanon [37] tested *FastDTW* with trillions of time series and determined that *FastDTW* took the least amount of time. H. Ding [17] reviewed 800 related papers, concluding that DTW was still the best method for the measurement of time series similarity.

2.2.3 Other Similarity Measurements

Although *ED* and *DTW* were two mainstream measurements in time series classification, other methods were also considered in some scenarios, such as *Longest Common Sequence* (LCSS) [38]. Instead of considering the whole time series, *LCSS* was based on a solution using pattern matching.

Another measurement which was discussed widely was *Edit Distance with Real Penalty* (ERP) [39]. Similar to *LCSS*, *ERP* also calculated the sub of a series but also included a constant penalty which was applied to non-matching parts. It can be treated as the development of *LCSS* which considered both matching and non-matching points.

Those techniques offset some of the disadvantages of *EE* and *DWT*. However, they had their own limitations. They were fast when applied to partial classification or pattern recognition, but not universally applicable.

2.3 Multivariate Time Series Classification

With the foundation of similarity measurements, the time series classification problem can be converted into an ordinary data mining problem. By using various preprocessing techniques, such as transfer to categorical variables, the univariate time series classification can be realized by many mature machine learning algorithms, such as *Decision Trees*.

Multivariate time series (MTS) classification received wide attention over the past decades. People can track time series data in more fields, such as human behaviour, medicine, or media. *Multivariate time series* were developed from techniques for *Univariate time series* classification. Since *Multivariate time*

series had more dimensions, there were many approaches that try to transfer *Multivariate time series* into a univariate format. One important preprocessing step was to reduce the number of dimensions.

2.3.1 Dimension Reduction by PCA

When talking about dimension deduction, the first algorithm came out was *PCA*, which was widely applied in various problems. *Principle Component Analysis* (PCA) was first proposed by Karl Pearson in 1901 [40]. *PCA* was a linear algebra technique for continuous attributes. It achieved dimensionality reduction by establishing an orthogonal coordinate system and capturing the maximum variation of data.

Although *PCA* was used for many applications to reduce the dimensions, using PCA to realize time series dimensionality reduction still had many difficulties. The main reason was that time series had data in order which PCA ignored. *PCA* only catched the variance between datasets. The sequence of time series had no effect on the PCA results.

A simple experiment to prove the limitations of *PCA* for time series dimensionality reduction was done as follows. First, I created two datasets D_1 and D_2, which had same the length of n. Each of them had two variables X_1, X_2. In D_1, X_1 was monotonically increasing with a degree of 1, and X_2 was a set of random variables that obey the standard normal distribution. In D_2, X_1 was monotonously decreasing, the degree of the decrease was 1, and X_2 was also a set of random variables subject to standard normal distribution. I performed *PCA* on D_1 and D_2 and used one dimension to represent them. The dimensionality reduction resulted D_1 and D_2 were the same. In fact, the orders

of the two datasets were different. This result illustrated the feature that *PCA* ignored the order and only extracted the variance.

In Figure 2.3, it was easy to see that with time series dimension variables in D_1 and D_2 are different, though, after PCA preprocessing in one variable, D_1 and D_2 became the same in Figure 2.4.

Although *PCA* cannot realize dimension deduction of time series datasets, it was still a very effective method for analyzing variance. There were many experiments discussed that applied *PCA* to *MTS* for comparing the similarity. Z. Wang [41] used *PCA* as one similarity measurement which efficiently caught the similarities between multiple variables. E. Keogh [42] combined *DTW* and *PCA* for a new approach of a dissimilarity measure for multivariate time series. The main idea of that approach was to use *PCA* to maximize the variance and then realize the classification.

2.3.2 Vertical Preprocessing

From the above discussion, we realize that dimensionality deduction was difficult to realize in most time series datasets. In this condition, how to convert multivariate time series to univariate became the crucial preprocessing problem.

Since the multivariate time series structure was discussed in the previous chapter, each multivariate time series was a matrix with multiple variables and time dimensions.

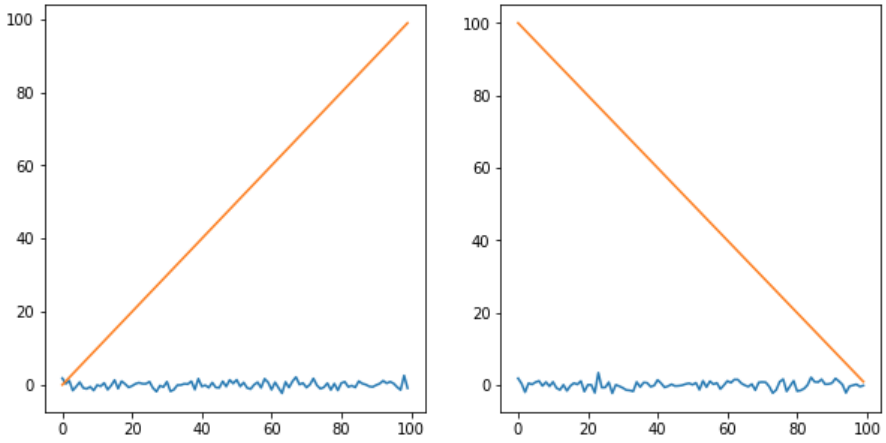


Figure 2.3: Comparison of Original D_1 and D_2 Dataset

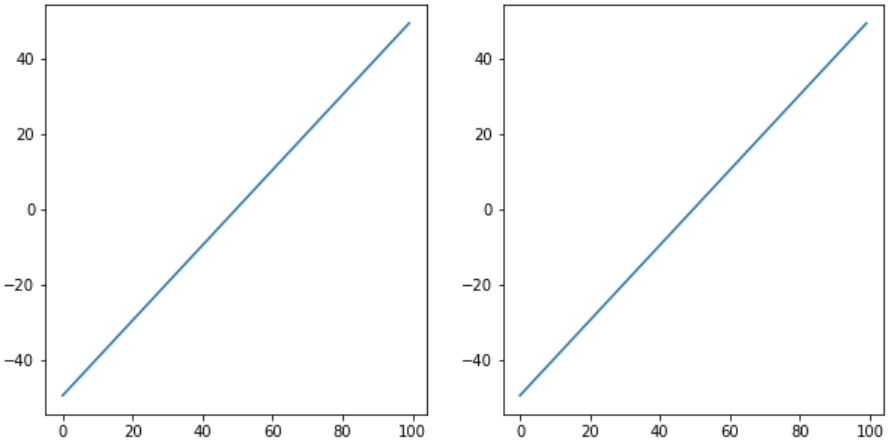


Figure 2.4: Comparison of D_1 and D_2 Dataset after PCA preprocessing
The original D_1 and D_2 with time dimensions are different in the above two graphs; though after PCA preprocessing in one variable, D_1 and D_2 become the same in the below graphs. This figure shows us that PCA only catches the variance of variables without sequence. That is why the two dataset's PCA results are the same.

$$\begin{bmatrix} X_{1,1} & ... & X_{1,n} \\ : & : & : \\ : & : & : \\ X_{d,1} & ... & X_{d,n} \end{bmatrix} \tag{2.4}$$

To classify multivariate time series, the most direct approach was to convert it to the univariate time series. In Equation 2.5, the i means the ith time series sample with the length of t and d variables.

$$x_i(t) : \{X_{1,1}, ...X_{1,t}, ...X_{d,1}, ...X_{d,t}\} \tag{2.5}$$

To concatenate the multivariate time series to a long univariate time series, each variable should be in same scale. The most common way to normalize various scales of data set was *Standardization*. The purpose of *Standardization* was to make the entire set of values have specific properties. If $\bar{x}$ was the average value of the attributes and s_x was their standard deviation, *Standardization* was the equation to create a new variable x' with a mean value of 0 and a standard deviation of 1.

$$x' = \frac{x - \bar{x}}{s_x} \tag{2.6}$$

The *Vertical preprocessing* method can be realized easily without an increase of the time complexity. It was also easy to understand as it utilized a direct way to compare each time series. Although the *Vertical preprocessing* can transfer multivariate time series in a direct way, it changed the shape of each variable. The disadvantage of standardization was to change the original value.

For example, the Basic Motion [43] dataset in Figure 2.5 needed to be *Standardized* by using Equation 2.6 in the first step. Then each time series was concatenated in a long time series.

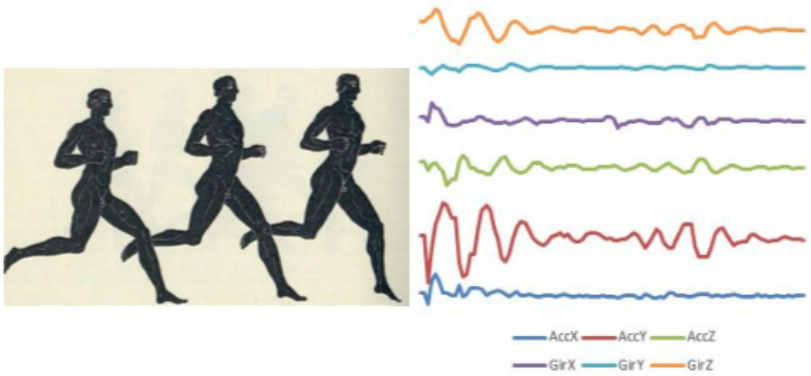


Figure 2.5: Basic Motion dataset[1]
Persons performed four activities whilst wearing a smart watch. The watch collects 3D accelerometer and a 3D gyroscope data. Each line is one dimension

2.3.3 Horizontal Preprocessing

Although *Vertical preprocessing* for multivariate time series can easily transfer multiple dimensions to a single dimension, it changed the scale of variables and *Standardization* decreased the differences between values. Researchers developed another method to maintain the original data. Instead of transferring multivariate to univariate, *Horizontal preprocessing* methodology was a method which direct classified each variable simultaneously and assembled each classification result. To show the logic, Basic Motion database in Figure 2.5 was taken as an example, the *Horizontal preprocessing* classified each dimension as univariate time series and then assembled the result.

The method can well make up the disadvantage of *Vertical preprocessing*,

which changed the original data value, but the cost was the increase of calculation. The sequence of each variable was also a key factor of the classification results. As a consequence, this method can gain a good performance when the data size was small with few variables.

Bagnall proposed a model named COTE [16] in 2015 which considered all possible methods in using a platform named COTE, which combined *Vertical preprocessing* and *Horizontal preprocessing*.

2.3.4 Image Topology Structure

Each multivariate time series sample had two dimensions, which can be classified as image. Using this idea, *MTS* was treated as image variables. Classification method such as *Convolutional Neural Network* (CNN) can be used. The most state-of-the-art technique was *ROCKET* [7], which was developed in 2019. *ROCKET* was not only focusing on a single representation, such as shape or variance, but used a *CNN* which can capture all features together.

There were many advantages to using *CNN* for time series classification. A *CNN* was able to successfully capture the spatial and temporal patterns through the application trainable filters and assigns importance to these patterns using trainable weights. It was good at taking input with multiple dimensions. *CNN* model identified the most significant patterns for samples with a reduction of noise. Z.Cui [44] used a *Multi-Scale Convolutional Neural Networks* for time series classification and achieved a high overall accuracy rate of 90%.

However, even though *CNN* can be the most reliable practice in time series classification, the disadvantage was still obvious. The computation complexity of CNN was high. Thus, CNN cannot be utilized to data with a large volume.

2.4 Evaluating the Models

Model performance will be evaluated in many ways. The first should be the overall accuracy score. Other than that, a Confusion Matrix will be implemented to show the performance for each class as well.

The Confusion Matrix shows the performance of the classification algorithm with different classes. The other measures are calculated based on the confusion matrix. In this thesis, Accuracy, Precision, Recall and $F - score$ will be calculated from the Confusion Matrix (see table 2.1).

Table 2.1: Confusion Matrix

	Positive	Negative
Positive	True Positive (TP)	False Negative (FN)
Negative	False Positive (FP)	True Negative (TN)

$$Recall = \frac{TP}{FN + TP} \tag{2.7}$$

The Recall shows the proportion of positive tests correctly predicted by the classifier

$$Precision = \frac{TP}{FP + TP} \tag{2.8}$$

The Precision shows how reliable the predict class

$$F - score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{2.9}$$

The $F - score$ is the mean of Recall and Precision

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.10}$$

Accuracy is the proportion of the total number of the correctly identified class that actually belongs to the class. It is an overall score to measure the classifier performance. However, accuracy is not an ideal measurement when classes are not balanced.

2.5 Summary

This chapter presented a literature review of the time series classification development and the most state-to-art techniques. The early history of time series research was focused on linear regression, which was a statistic model to describe the relationship between time and value. Research of time series classification started from linear algorithms and evolved into a univariate time series classification. With the demand of industry in many fields, the multivariate time series were mentioned more generally in academic research.

Chapter 3

Dataset and Environment

Since time series research gained more attention in the early 1990s, the demand for experimental data was becoming more and more urgent. However, in those days, it was difficult to obtain data from the web or other resources. To solve the issue and be inspired by the increasing contributions of the UCI (University of California Irvine) Archive to the Machine Learning Community, Koegh Folias developed the UCR (University of California Riverside) time series datasets archive in 2002 [2]. The first version of the archive contained only 45 datasets, which increased to 85 in 2015. As of the Fall of 2018, the archive expanded to 128 data sets. Currently, UCR Time Series Archive has become the largest depository for time series classification. As a consequence, the majority of algorithms in time series classification were developed by using the archive data. The format can be described as follows:

- Each time series has the same length
- The data is normalized

- Individual datasets are small
- The split for training and testing data already exists

Each time series dataset had various number of columns and rows in UCR archive. However they all had same data structure. There was one example from UCR archive which showed the structure. The structure of the UCR archive can be represented in a two-dimensional table as showned in Figure 3.1. Each row was a single time series with the class at the beginning, and there were 151 columns, meaning there were 151 time points for each sample. The column header 1,2,3...indicated the time point with the normalized value.

	0	1	2	3	4	5	6	7	8	9	...	
0	1	-1.12500	-1.13130	-1.13830	-1.14670	-1.13860	-1.14140	-1.14370	-1.14440	-1.15490	...	-1.
1	2	-0.62696	-0.62592	-0.62754	-0.62633	-0.62409	-0.62471	-0.62501	-0.62418	-0.62402	...	-0.
2	2	-2.00120	-1.99960	-1.99950	-1.99920	-1.99900	-2.00030	-1.99840	-2.00070	-2.00000	...	0.
3	1	-1.00460	-0.99984	-0.99525	-0.99202	-0.99120	-0.98756	-0.99647	-0.99688	-0.99867	...	-1.
4	1	-0.74263	-0.74377	-0.74390	-0.74487	-0.74475	-0.74536	-0.74708	-0.74661	-0.74671	...	-0.

Figure 3.1: GunPoint Data Set from UCR Archive [2]
Each row represents one movement by 151 time points. The class 1 and 2 means With Gun or Without Gun

Motivated by the UCR time series datasets archive, the University of East Anglia developed the Multivariate Time Series Archive in 2018 which can be found at the website *www.timeclassification.com* [3]. There were 30 datasets from a wide range of real-life problems. The UEA multivariate time series archive inherited the format of UCR, in which each single time series is equal length.

The Multivariate Time Series archive can be stored in a 'Table' format as well, but in each column, there was a certain length of time series data instead

of a single value. The column header $'dim0'$, $'dim1'$... described the different features; it's structure was shown in Figure 3.2.

	dim_0	dim_1	dim_2	dim_3	dim_4	dim_5
9	0 -0.407421 1 -0.407421 2 2.355158 3...	0 1.413374 1 1.413374 2 -3.928032 3...	0 0.092782 1 0.092782 2 -0.211622 3...	0 -0.066584 1 -0.066584 2 -3.630177 3...	0 0.223723 1 0.223723 2 -0.026634 3...	0 0.135832 1 0.135832 2 -1.946925 3...
24	0 0.383922 1 0.383922 2 -0.272575 3...	0 0.302612 1 0.302612 2 -1.381236 3...	0 -0.398075 1 -0.398075 2 -0.681258 3...	0 0.071911 1 0.071911 2 -0.761725 3...	0 0.175783 1 0.175783 2 -0.114525 3...	0 -0.087891 1 -0.087891 2 -0.503377 3...
5	0 -0.357300 1 -0.357300 2 -0.005055 3...	0 -0.584885 1 -0.584885 2 0.295037 3...	0 -0.792751 1 -0.792751 2 0.213664 3...	0 0.074574 1 0.074574 2 -0.157139 3...	0 0.159802 1 0.159802 2 -0.306288 3...	0 0.023970 1 0.023970 2 1.230478 3...
7	0 -0.352746 1 -0.352746 2 -1.354561 3...	0 0.316845 1 0.316845 2 0.490525 3...	0 -0.473779 1 -0.473779 2 1.454261 3...	0 -0.327595 1 -0.327595 2 -0.269001 3...	0 0.106535 1 0.106535 2 0.021307 3...	0 0.197090 1 0.197090 2 0.460763 3...
34	0 0.052231 1 0.052231 2 -0.54804...	0 -0.730486 1 -0.730486 2 0.70700...	0 -0.518104 1 -0.518104 2 -1.179430 3...	0 -0.159802 1 -0.159802 2 -0.239704 3...	0 -0.045277 1 -0.045277 2 0.023970 3...	0 -0.029297 1 -0.029297 2 0.29829...

Figure 3.2: BasicMotion Data Set from UEA Archive in DataFrame Format [3] Each row represents one sample. One sample has 5 features. In another word, one sample has 5 time series happen simultaneously

Since the dataset used in this thesis was a multivariate time series, I was inspired by the UEA dataset archive for the data structure. Though there was still a certain gap between my actual data and the ideal model, the UEA archive gave good guidance on model establishment. How to transfer the actual data to an archive format data was described in the following section.

3.1 Analysis of Pilot Performance Data

The dataset used in this thesis came from industry partner CAE Inc. CAE is dedicated to aviation technology. It is one of largest flight simulator manufacturer in the world. Their main products are flight simulators which assist pilot trainees in familiarising themselves with the flight environment before flying. CAE's main business is building simulators and developing aviation training

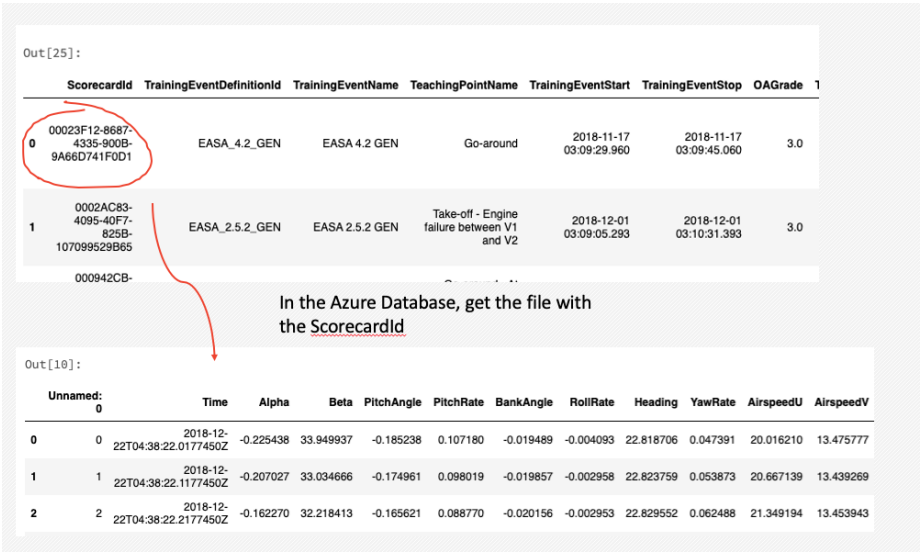
centers for their clients all over the world.

My research was focusing on flight training data. The flight training data was collected directly from each simulator. Each session was stored in one table with time points as one dimension. The data recorded all performance using different variables, such as speed, altitude, flight angle.

A metadata table that contained the different training events including the overall information such as the start time, end time and scored that achieved. The data was stored in an AZURE cloud environment. For each record, a log file recorded specific data from the telemetry of the simulator. Around 4000 parameters of different types were records at about a frequency 10HZ. Unlike archive datasets, there were separate tables for each training session. In each table, the variables were listed as a time sequence. The structure of the dataset can be seen in Figure 3.3. I can use the attribution **ScoreId** to reserve the corresponding time series dataset from certain training sessions. The class label is the scored as the **GAGrade** in the metadata.

The line graphs of each time series can be seen from Figure 3.4 to Figure 3.7. Since the dataset was a multi dimensional data structure, it was shown in each class by same feature. Figure 3.4 to Figure 3.7 were four score classes – Score 1.0, 2.0, 3.0, and 4.0. The x-axis was time and the y-axis was ground speed. It can be seen from the graph that at the beginning the ground speed was low. It rose quickly in the next period. Then the speed was stable until the end after reaching peak speed.

Some differences were observed from the graph easily. Score 4.0 took less time to reach peak speed than other scores. This might be one reason for the score difference. Other than that, Score 4.0 had more samples than other three



Out[25]:

	ScorecardId	TrainingEventDefinitionId	TrainingEventName	TeachingPointName	TrainingEventStart	TrainingEventStop	OAGrade	T
0	00023F12-8687-4335-900B-9A66D741F0D1	EASA_4.2_GEN	EASA 4.2 GEN	Go-around	2018-11-17 03:09:29.960	2018-11-17 03:09:45.060	3.0	
1	0002AC83-4095-40F7-825B-107099529B65	EASA_2.5.2_GEN	EASA 2.5.2 GEN	Take-off - Engine failure between V1 and V2	2018-12-01 03:09:05.293	2018-12-01 03:10:31.393	3.0	
	000942CB-							

In the Azure Database, get the file with the ScorecardId

Out[10]:

	Unnamed: 0	Time	Alpha	Beta	PitchAngle	PitchRate	BankAngle	RollRate	Heading	YawRate	AirspeedU	AirspeedV
0	0	2018-12-22T04:38:22.0177450Z	-0.225438	33.949937	-0.185238	0.107180	-0.019489	-0.004093	22.818706	0.047391	20.016210	13.475777
1	1	2018-12-22T04:38:22.1177450Z	-0.207027	33.034666	-0.174961	0.098019	-0.019857	-0.002958	22.823759	0.053873	20.667139	13.439269
2	2	2018-12-22T04:38:22.2177450Z	-0.162270	32.218413	-0.165621	0.088770	-0.020156	-0.002953	22.829552	0.062488	21.349194	13.453943

Figure 3.3: Data Structure for Pilot Performance Assessment
The above table is meta data of all samples and their classes. Each sample has a table with time series details given in the table below

classes and Score 1.0 has the least samples. Those factor could be a main reason in the testing results which discussed in Chapter 5.

There were 8696 records in the current database for a 6-month collection. In the whole dataset, there were 28 training events, such as Go-around, Take-off. Engine-Out-Landing, etc. For each of the training events, there were multiple tests. Each of the training event log files contained 164 features, such as Ground Speed, Altitude. These features consisted of various telemetry measures collected at regular intervals during a simulation exercise. As the first phase of the modeling, I started using a single training event – Take-off, as selected by industry partner. Since there were too many features to be manipulated, the partner selected a subset of features using their domain knowledge. In their documentation [45], those features can represent a flight status. For time series classification purposes, there was a subset of features of the dataset. The

descriptions were listed in Table 3.1.

There is a total of 673 records in four classes. The four classes can be seen in Table 3.2

3.2 Software

All preprocessing and modeling parts in this thesis are performed using Python Version 3.7. Various external libraries for Python were used in this thesis as well. I used a `DataFrame` which is a spreadsheet format from the pandas library. Another library used for time series classification is `sktime`, which is developed by the Alan Turing Institute research group [46]. It is an open-source Python library for machine learning with time series. The version of `sktime` used here is 0.4.1. The library supports:

- Forecasting
- Time series classification
- Time series regression

The main development idea of `sktime` is to extend existing machine learning capabilities based on `scikit-learn` [46]. `Sktime` development team aimed to design and implement an API (application programming interface) for time series. The library includes all aspects for time series classification, including data analysis, data regularization, and state-of-the-art algorithms for time-series classification.

Table 3.1: Table of test captions and labels

Column	Data type	Description
AirspeedTrue	Double	The speed of simulated aircraft relative to the surrounding air mass
AltitudeAGL	Double	Direct distance measured between the simulated aircraft and the ground
PitchAngle	Double	The vertical angle between the nose of the aircraft and the horizon
BankAnglem	Double	The vertical angle between a wingtip of the aircraft and the horizon
AngleOfSideslip	Double	The origin of the relative wind
Engine1N1	Double	Engine1 indication of the engine thrust through the measure of the rotational speed of the low-pressure engine spool
Engine2N1	Double	The engine2 indication of the engine thrust through the measure of the rotational speed of the low-pressure engine spool

The features selected by industry partner, which can represent the situation of an aircraft, explanation for each feature from [45]

Class	Count
Score 1.0	57
Score 2.0	163
Score 3.0	145
Score 4.0	308

Table 3.2: Class Distribution

3.3 Preprocessing

Given the previous discussion, the data needs to be transformed so that supervised machine learning algorithms can applied.

3.3.1 Proportional Scaling

The time series contained in the UEA archive has some common features: they are of equal length. This is a priori condition for most time series classification problems. The goal is to standardize each time series for comparison. If the lengths are different, it is difficult to be implemented the similarity measurement. However, datasets in this project have various length. To visualize the time series in each class, we use one training event with one attribute to show four classes for exploration purposes in Figure 3.4 to Figure 3.7.

Another aspect that needs to be considered is timing. For example, if a pilot operates the simulator in a shorter time to get to a certain altitude level, he will get a better score. However, this is not the only reason for attitude. From our partner document, reaching the target altitude could also hinge on other variables such as, aircraft gross weight, air temperature, or thrust settings. In this thesis, there are selected features from partner already. With this condition, I need to maintain the basic shape of each time series and transfer them to the

same length. In order to solve this problem, I adopt the stratified sampling idea from statistics, which is a method of sampling from a population which can be partitioned into sub populations. *Proportional Scaling* was proposed by Kanatani in 1997 [47] for motor control applications. The main idea is to use as few samples as possible to represent the whole population. The algorithm separates the whole time series into k pieces and gets the value at point k. If I take the time point as measurement dimension, I can set the stratified sampling to a certain percentage of the whole time series. For example, if the whole time series using n time points, we can extract k values sequentially: the value on the $1th$ time point, the value the $2th$ time point ... utile the value of the kth time point. The algorithm is shown in Equation 3.1

$$x_{sample}(t) : \{X_{\frac{1*n}{k}}, X_{\frac{2*n}{k}} ... X_{\frac{(k-1)*n}{k}}, X_n\} \tag{3.1}$$

In Equation 3.1, n represents the whole time series. With the sampling method, I can generate a sample that can represent the real time series. If the number of points is not divisible by the sampling number k, I round the result of the division. By using Equation 3.1, no matter how many time points the time series data has, it can be transferred to a certain number of lengths. For example, if there is a time series with 1000 time points, and we want to use 100 time points to represent it. By using the Equation 3.1, we will use the time point $10th$, $20th$. $30th$ ... $1000th$ time point to represent the whole time series.

To show an example: I randomly select one time series (take-off with a ground speed feature). The original data set is 858 time points in length, and the goal is to transfer it to 100 time points length. By using Equation 3.1, I will take the $\frac{1*858}{100}$th, $\frac{2*858}{100}$th... $\frac{99*858}{100}$th values to generate a new time series

data. Since $n = 858$ cannot be divided by k =100, I use the round value of each point. Therefore, I use the $9th$, $18th$ ... $849th$ values instead of the whole time series. If comparing the preprocessed data with the original one, they have similar shapes. Figure 3.8 and Figure 3.9 show a comparison of the original time series and after proportional scaling preprocessing in 100 time points.

Figure 3.8 shows that the value increases from time point 0 to time point 400, with a really small fluctuation on the peak value. Figure 3.9 shows that the new time series data with *Proportional Scaling* has the same shape as the original data. When comparing these two graphs, the transformed data set has the same shape as the original one.

If the time series' monotonicity is constant, which means it always increases or decreases, I can use few points to represent the series. Otherwise, we need to improve the sensitivity to resampling, which means we need more points.

With this algorithm, the various length time series data can be preprocessed into series of the same length. There are advantages of this method:

- The algorithm can use few samples to represent the various length time series, which can improve the efficiency for the time series classification
- The algorithm can be adjusted to various scaling needs
- The scaling can be performed at various resolutions within one series.

3.3.2 Transfer Time Series Data Structure

Unlike the normal data mining problem using a two-dimensional matrix, multivariate time series have more dimensions. How to organize classification algorithms on each sample is another difficult problem, since the project data

is stored in multiple tables and needs to be reorganized into a two-dimension matrix.

The start of the data structure adjustment is to load all test samples into a list structure `inputList`. Based on the selected features and sampling points defined, cut each time series with one feature and certain time points separately and save them in a list structure `outputList` as output in Algorithm 1. The next step is to expand each time series with a single value and generate a nested table in a `DataFrame` structure. Here I used the function `fromLongtoNested` to realize this step. The process is shown in Figure 3.10.

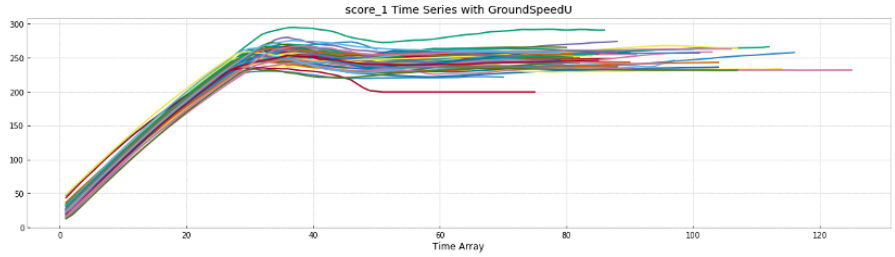


Figure 3.4: Score 1.0 Time Series with Ground Speed

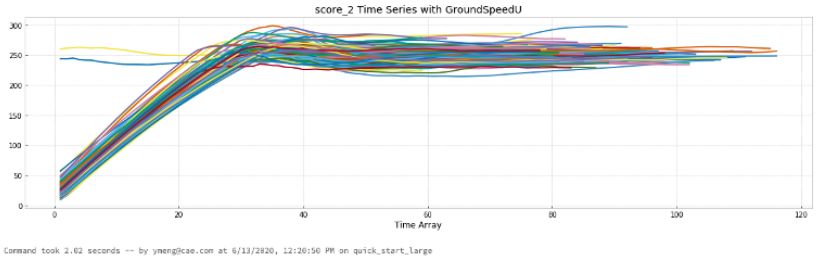


Figure 3.5: Score 2.0 Time Series with Ground Speed

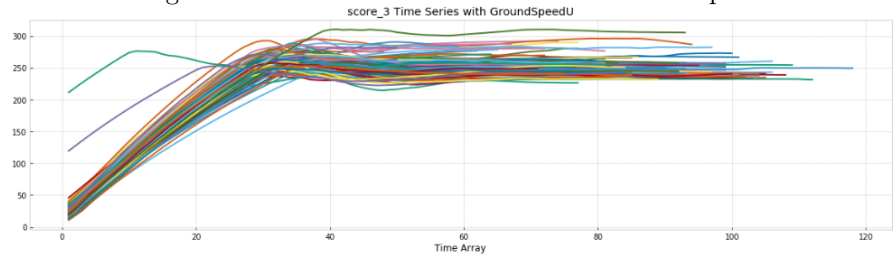


Figure 3.6: Score 3.0 Time Series with Ground Speed

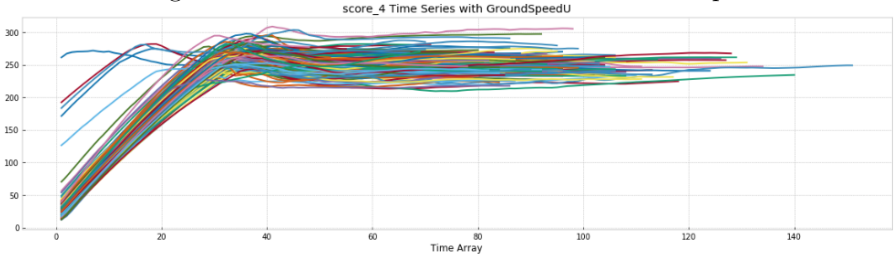


Figure 3.7: Score 4.0 Time Series with Ground Speed

Figure 3.4 to Figure 3.7 show Score 1.0, 2.0, 3.0, and 4.0 with Ground Speed feature. Score 1.0 samples are very similar at the beginning, and the time of peak speed is almost the same. Score 2.0 and Score 3.0 have similar amounts of samples and trends. Score 4.0 has more samples and the peak value within the first 20 time points

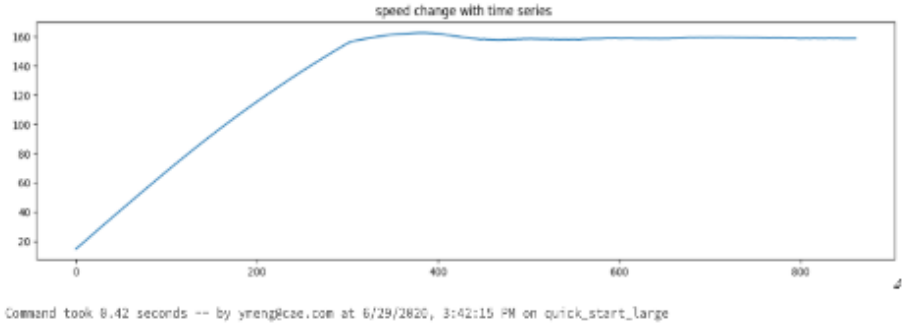


Figure 3.8: Original Time Series Data Set

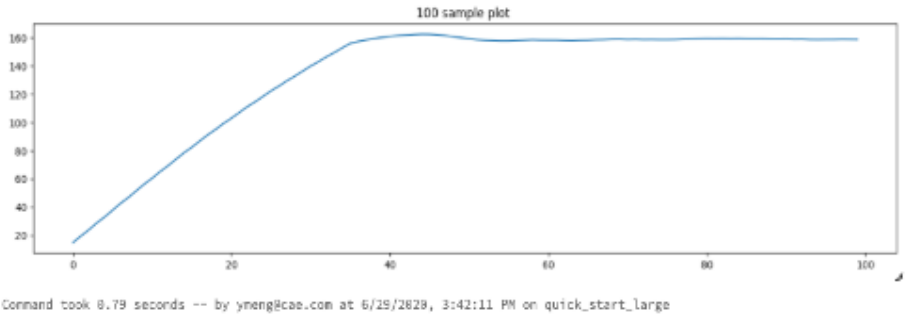


Figure 3.9: Proportional Scaling Preprocessing with 100 Time Points

The values in Figure 3.8 increase from Time Point 0 to Time Point 400, with a really small fluctuation on the peak value, it becomes flat until the end. In Figure 3.9, the new time series data with *Proportional Scaling* has the same shape as the original data

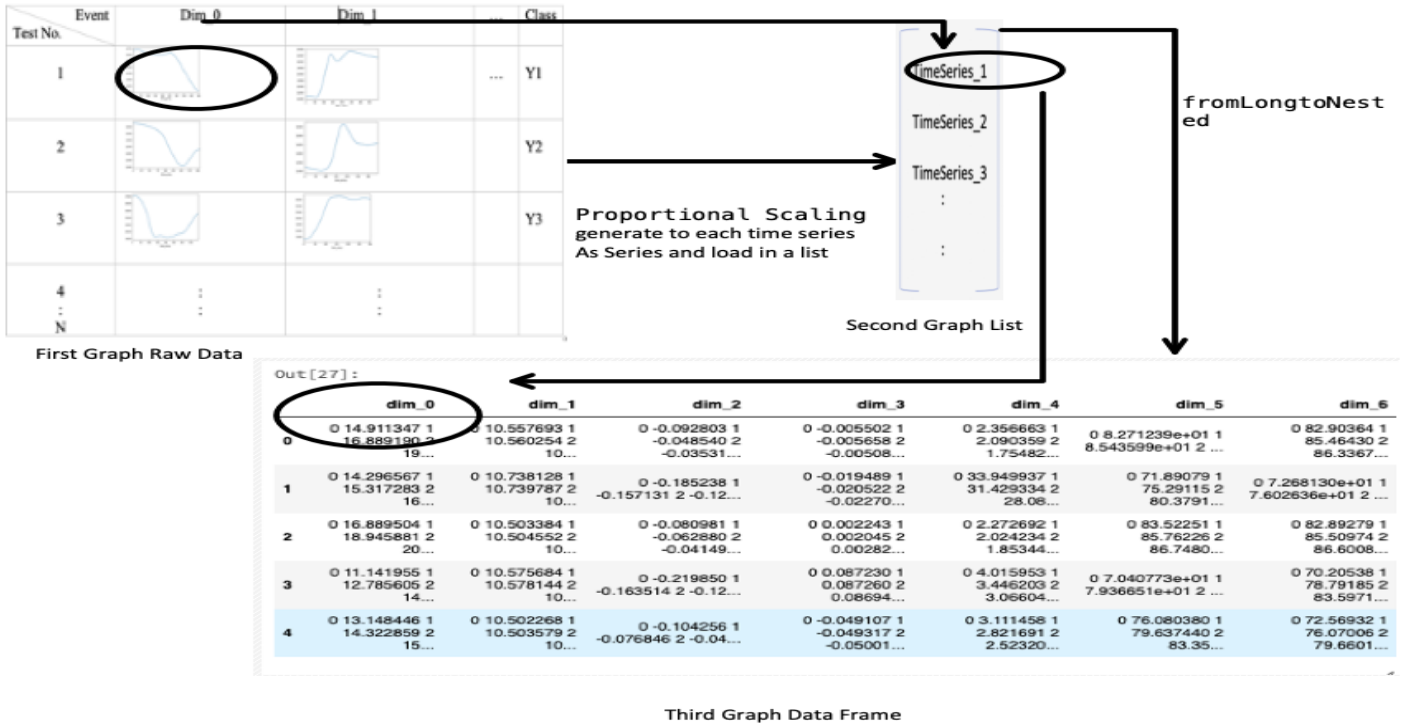


Figure 3.10: Data Transformation Process

The first graph shows the original data structure. Each record has a time series on each dimension. The second graph is the first step of transformation which cuts each time series to the same length and stores it in a list structure. The third graph shows transformations of the list into a `DataFrame` by invoking `fromLongtoNested`

Figure 3.10 shows the raw data stored in a list with each log file as a table. One table represents one test with various features. In the first step, I cut them into a single feature with the same length by using Proportional Scaling methods. The algorithm is shown in Algorithm 1. The output of this step is univariate time series data with a single feature. In the next step, I invoked the function `fromLongtoNested` and the output of this step is a `DataFram` which has a time series in each column. The final output of this procedure is a 'Table' which has each time series inside of column unit.

```
Data: inputList
Result: Generate Splite Sampling Time Series
outputList ← [] ;
for test j in inputList.length do
    for feature f on selectFeatures do
        for timepint i in samplingNumber do
            outputList.add(inputList[t][f][round(inputList[t].length * i /
            samplingNumber)])
        end
    end
end
```

Algorithm 1: Generate Long List Algorithm

3.4 Summary

This chapter is an introduction of the dataset used in this thesis. The preprocessing methods is also presented. Using well-developed and innovative python library, I transfer the raw data to a specific structure for classification. The preprocessing step *Proportional Scaling* is proposed in this chapter. This algorithm is dedicated to use few time points to represent the whole series other than the original data. The algorithm will solve the problem of various length in time series classification and improve the efficiency.

Chapter 4

Models

Mitacs partner company CAE.Inc used Deep Learning on Time Series Classification for their project - Training Event Detection [45]. They used *Long Short Term Memory networks* (LSTMs) [48] to build the model. The result was 81% accuracy, but the computational complexity was high. Training the model process took two days with twenty nodes. It showed a significant limitation for this work. Deep Learning model was efficient and cannot be applied for this particular example.

To improve the efficiency and applicability to here, the use of traditional classification models is necessary. Chapter 2 introduced each model with their advantages and disadvantages. In this project, the whole time series which with time points and values needs to be compared as a vector. There are no repeated particular patterns. Based on the characteristics of our dataset, similarity-based technique and interval-based technique were chosen.

Similarity-based technique and interval-based technique are two widely used models for Time Series Classification. The similarity-based technique is to cal-

culate the distance between two values on every two opposite time points and sum them together, using a brute force method. Interval-based technique transfers partial time series into categories and use a decision tree for classification. These techniques were introduced in Chapter 2.

The advantage of the interval-based technique is that it has a lower computational complexity. Rather than use the whole series, the interval class of algorithms selects one or more dependent intervals of the series. If the time series has outstanding features, such as peak points and fluctuation, the interval-based technique will perform better. In Chapter 2, I compared the training complexity between similarity-based technique and interval-based technique. The interval transformations will result in meaningful information such as some minor changes and will cause a bias of the model.

However, some information will be lost. Unlike a interval-based technique, a similarity-based technique maintains the original data. The disadvantage of the similarity-based technique is a higher computational complexity. The more time points, the more computing time is required.

4.1 Model 1: 1-NN with DTW

1-NN is the most intuitive classification algorithm. Keogh provided significant insight in this field [5] [49] [50]. It is widely applied to UCR data in many applications.

Keogh discussed how to measure the time series distance first. He discussed that, other than using traditional Euclidean distance, DTW will improve the accuracy of the time series classification. The disadvantage is that DTW will

also increase the computational complexity. Keogh also tried to develop other algorithms such as FastDTW, WDTW, and DDTW. Keogh's insisted that the 1-NN is the best method for time series classification [34].

K-NN, which is a traditional classification method, uses the *Lazy Leaner* strategy, which means it will not classify the data during the training process. The parameter k refers to the number of nearest neighbours to include in the majority of the voting process. The value of k vary from 1 to many. If the k is too small, the test samples will be under-classified. If k is too large, the test samples will be over-classified. If the time series has noise, the result will be opposite.As k increases, the test samples will be classified more accurately.

Result: k-NN Classification Algorithm

$thenearestneighbour \leftarrow k$;

$theTrainingDataset \leftarrow D$;

for *each test instance z = (x',y'))* **do**

Compute z = x',x, the distance between z and every sample, (x,y) $\in$ D;

Select $D_Z \subseteq$ D , the set of k closest training example to z;

y' = argmax$\sum(x_i,y_i) \in D_z$ I(v = y_i)

end

Algorithm 2: K-NN Classification Algorithm [51]

The test instance is classified by the majority of its nearest neighbors through voting.

The Elastic Ensemble Model was developed based on 1-NN combined with time series similarity measurement. The algorithm is described below. Since

the development of DTW which was proposed in early 1990, the techniques and discussions applied to this area have seen substantial growth. These include DTW and its derivatives products and edit distance-based measures, such as longest common subsequence. Some methods perform better on specific time-series datasets samples but not all. As a consequence, another ensemble idea was proposed by Lines and Bagnall [52].

```
Result: 1-NN Classification Algorithm
bestsofar ← inf ;
for i in length(TRAINclasslables) do
    comparetothisobject ← TRAIN(i,:) ;
    distance ← DWT distance;
    if distance < bestsofar then
        predictedclass = TRAINclasslabels(i) ;
        bestsofar = distance ;
    end
end
```

Algorithm 3: 1-NN Classification Algorithm with DTW [8]

The 1-NN Classification Algorithm with DTW will use DTW as the similarity measurement and calculate all distances between test instances and training dataset, then using the closest training sample class as the classification result.

An ensemble of classifiers is a combination of base classifiers with their decisions. The first version of the ensemble in time series data mining was developed by Deng et al [21]. They compare 1-NN with Euclidean distance and 1-NN with DTW. Their results showing improvement in accuracy.

4.2 Model 2: Time Series Forest (TSF)

Time Series Forest (TSF) is the most representative method of the interval-based TSC techniques. The generation process is a top-down, recursive strategy that is similar to a standard decision tree. It was first proposed by Deng in 2013 [21]. TFS was developed to summarize the features from intervals of a time series. Instead of using the whole time series, TSF focuses on every possible sampling interval. For example, if a time series' length is m, there will be $m(m-1)/2$ possible intervals. TFS takes a Random Forest-like approach. Based on mean, standard deviation, and slope, the algorithm will find the most likely subsequence as features to be used in classification.

Result: buildTS(A list of n cases length m, T = (X, y))

the number of trees, k; the minimum interval length, p; the number of intervals per tree, r

$k \leftarrow 500$ as default, $p \leftarrow 3$, $r \leftarrow \sqrt{m}$ i $\leftarrow 1$;

Let $\mathbf{F} = (F_1 ...F_k)$ be the trees in the forest;

while $i < k$ *and time Remaining* **do**

    Let S be a list of n cases (s_1 ... s_n);

    for $j \leftarrow$ *1 to r* **do**

        b $\leftarrow$ randVetween(1, m-p);

        e $\leftarrow$ randBetween(b+p, m);

        for $i \leftarrow$ *1 to n* **do**

            $s_{t,3(j-1)+1} \leftarrow$ mean(x_t,b,e);

            $s_{t,3(j-1)+2} \leftarrow$ standardDeviation(x_t,b,e);

            $s_{t,3(j-1)+3} \leftarrow$ slope(x_t,b,e)

        end

    end

    F_i.buildTimeSeriesTree(S,y)

end

The value of k can be vary. The default of the k is set by the development team to 500

Algorithm 4: TSF algorithm [8]

4.3 Model 3: Random Interval Spectral Ensemble (RISE)

RISE is also a tree based interval ensemble similar to TSF. However, there is an improvement in the efficiency compared to TSF. RISE only uses a single interval for each tree compared to multiple trees of TSF. RISE also employs spectral features rather than statistical analysis. At the same time. RISE also uses Fast Fourier Transform (FFT) and Auto Correlation Function (ACF) to transfer the same interval for each series. There is also an improvement to the original RISE which used the partial autocorrelation function and autoregressive model features. These make RISE faster than traditional TFS.

Result: buildRISE(A list of n cases of length m, **T** = (**X**,y)

the number of trees, k; the minimum interval length, p; the number of

intervals per tree, r

$k \leftarrow 500$ as default, p $\leftarrow$ min(randomNumber n, m/2) ;

Let **F** = (F_1 ...F_k) be the trees in the forest ;

while $i < k$ *and time Remaining* **do**

    buildAdaptiveTimeModel();

    if $i = 1$ **then**
        r $\leftarrow$ m

    else

        max $\leftarrow$ findMaxIntervalLength();

        r $\leftarrow$ findPowerof2Interval(p,max);

    end

    T' $\leftarrow$ removeAttributesOutsideofRange(**T**,**b**,**r**);

    S $\leftarrow$ getSepctralFeatures(**T'**);

    F_i.buildRandomTreeClassifier(**S**,y);

    updateAdaptiveModel(r);

    i $\leftarrow$ i+1

end

The value of k can be various. The default of the k is set by develop

team as 500

Algorithm 5: RISE algorithm [8]

4.4 Column Concatenate Transform

In Chapter 3, Proportional Scaling was discussed. The propose of Proportional Scaling is to transform the various length time series and to keep the

most significant features.

After the Proportional Scaling, the data used in this project can be investigated using transformation and classification algorithms. In this research, not only the classification algorithm is adopted, but the transfer algorithm is also an important step. There are two main strategies implemented: *Horizontal* and *Vertical*. The *Vertical* method is to concatenate each time series to a long series, and *Horizontal* is to classify by each feature and vote. The majority class will be the testing class. The precondition to use those two transform methods is cut each time series to the same length and load each time series in a *DataFrame* structure. This preprocessing step is shown in Figure 3.10.

Algorithm 6 shows how to change data structure form a multi-dimension to a . The input *DataFrame* is a table with (m,n) dimensions. The algorithm will transfer to an output *DataFrame* with (m*n,1) dimensions which can implement the univariate time series classification.

Result: Concatenate Each Feature to Long Time Series

a dataframe InputTable(m,n), each time series t

Output ← NewDataFrame(m*n,1);

for t *in InputTable* **do**
| NewDataFrame.add(t)

end

Algorithm 6: Concatenate Algorithm

4.5 Column Ensemble Transform

The Column Ensemble Transfer is a horizontal processing method that uses a different logic from the vertical processing method. Unlike to concatenate each

dimension, Column Ensemble retains the original data structure. As it is shown in Figure 4.1, the test samples will be classified by each of feature: $feature_1$, $feature_2$, $feature_3$, $feature_4$ simultaneously. The results will be voted by ensemble method.

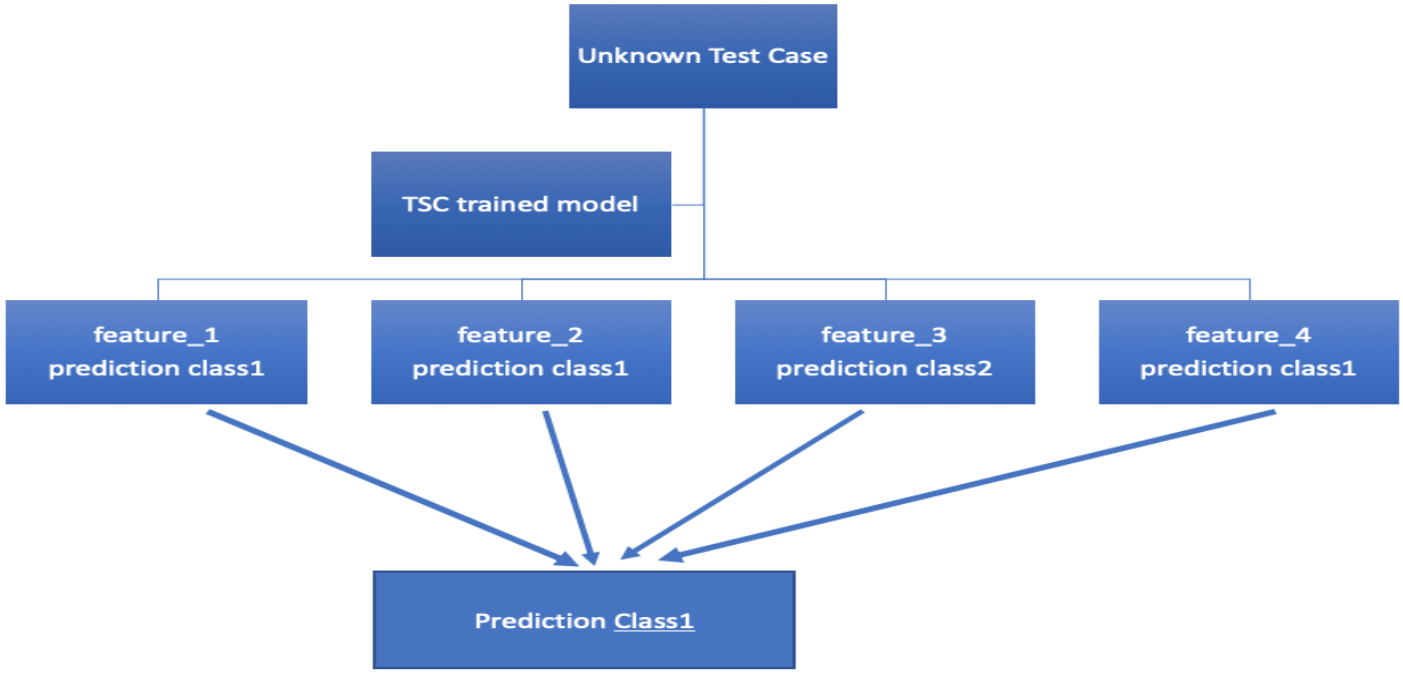


Figure 4.1: Column Ensemble Algorithm
The test samples will be classified by each of feature:
$\text{feature}_1, feature_2, feature_3, feature_4. The results will be voted by ensemble method and get the prediction class$

4.6 Implementation of Models

The industry partner has developed a deep learning model by using LSTMs. In their report, they state that LSTMs architecture aims at predicting a sequence from another input sequence [45]. The model aims at compressing and exacting the most relevant information across the input signal.

Different from their model, I use the three models with two preprocessing methods. I first use *Proportion Scaling* to cut each time series to the same length, then load them in a 2-dimension table. I used the models implemented sktime with the preprocessing methods. The results and comparison will be discussed in the next chapter.

4.7 Summary

In this chapter, I mainly illustrated each model algorithm used in this thesis. Due to the characteristics of the project data, I choose three models—1-NN, TSF, and RISE. Since the data used in the project has multiple features, two transformation methods are employed to transfer the original data to univariate time series. The two transformation methods are considered from different aspects: Column Concatenate and Column Ensemble. The outputs of two transformation methods are univariate time series which can be applied classification model.

Chapter 5

Results and Optimization

The aim of this study was to train a model that can correctly classify four score classes: Score 1.0, Score 2.0, Score 3.0, Score 4.0. Each score class represented the trainees' performance. In this chapter, the results of three classification methods and two preprocessing methods were presented. The results were discussed using accuracy score, precision, recall and f1-score. k-fold Cross Validation was also used in the evaluation section. Each dataset was divided into training data and testing data. The test size was 0.25, which meant 75% data was used as training data while the remaining 25% was used as testing data.

5.1 Evaluating the Models

5.1.1 Evaluation for 1-NN with DTW using Concatenation Transform

The first model was 1-NN with DTW using Concatenation Transform, with a 0.31 overall accuracy score, see Figure 5.1. Even though the average accuracy score of four classes exceeded the random likelihood, each class behaved differently.

```
from sktime.classification.distance_based import KNeighborsTimeSeriesClassifier
steps = [
    ('concatenate', ColumnConcatenator()),
    ('classify', KNeighborsTimeSeriesClassifier())]
clf = Pipeline(steps)
classifier = clf.fit(X_train, y_train)
clf.score(X_test, y_test)

```

```
Out[19]: 0.31333333333333335
```

The overall accuracy rate of 1-NN with DTW using Concatenation Transform is 31%

Figure 5.1: 1-NN with DTW using Column Concatenate Transform Accuracy Score

According to Table 5.1, the precision for Score 1.0 was 7%, Score 2.0 24%, Score 3.0 13% and Score 4.0 37%. Thus, Score 4.0 had the greatest accuracy in this model. Recall reflected the percentage of correctly classified given that instances belong to the classified class: Score 1.0 only had 5% correct hits, while Score 2.0 had 24%, Score 3.0 had 13% and Score 4.0 had 37%. F1-score was the weighted mean of precision and recall, and the best score was 0.37 for Score 4.0. Our classifier reported 0.06, 0.25 and 0.13 for Score 1.0, Score 2.0 and Score 3.0 respectively.

In Figure 5.2, the Confusion Matrix shows the number of each class cor-

rectly predicted. Score 4.0 had the highest correct predictions. The majority of incorrect predictions happened when Score 1.0, Score 2.0 and Score 3.0 were predicted as Score 4.0. Some of Score 4.0 samples were predicted as Score 2.0.

Class	Precision	Recall	F1-score	Support
Score 1.0	0.07	0.05	0.06	19
Score 2.0	0.24	0.26	0.25	39
Score 3.0	0.13	0.13	0.13	30
Score 4.0	0.37	0.37	0.37	62

Table 5.1: 1-NN with DTW using Column Concatenate Transform for each class

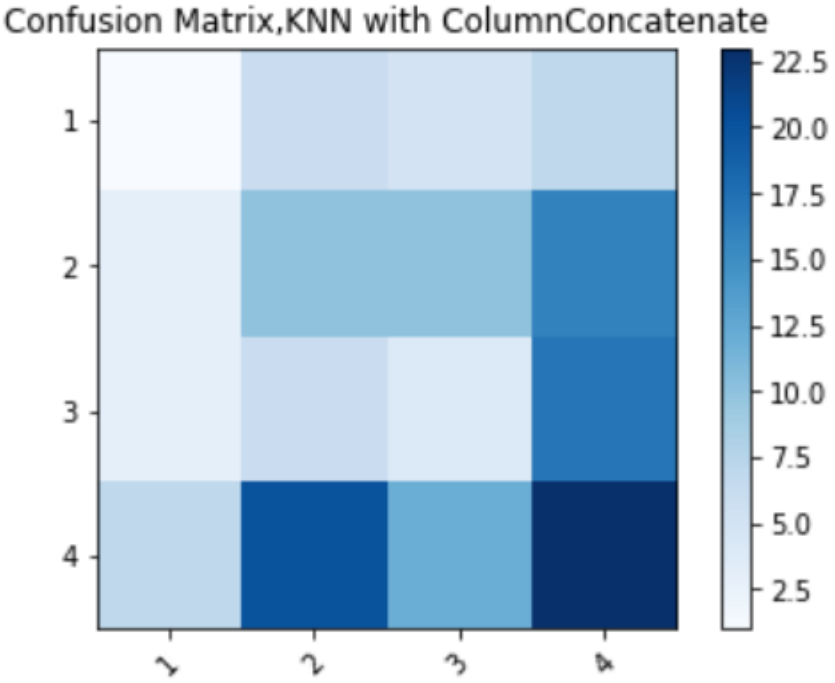


Figure 5.2: 1-NN with DTW using Column Concatenate Transform Confusion Matrix

5.1.2 Evaluation for TSF using Concatenation Transform

The second model was TSF which used Concatenation Transform. The overall accuracy score for this model was 0.39, indicating that 39% of test samples were correctly predicted. It performed better than the first model, since the first model's overall accuracy was 0.31.

```
from sklearn.model_selection import cross_val_score
steps = [
    ('concatenate', ColumnConcatenator()),
    ('classify', TimeSeriesForestClassifier(n_estimators=100))]
clf = Pipeline(steps)
classifier = clf.fit(X_train, y_train)
clf.score(X_test, y_test)

```

```
Out[13]: 0.38666666666666666
```

Figure 5.3: TSF using Column Concatenate Transform Accuracy Score
The overall accuracy rate of TFS using Concatenation Transform is 38%

From Table 5.2, it can be observed that the precision for Score 1.0 was 100%, which was the highest compared to all other class. Score 2.0 was 39%, Score 3.0 was 14% and Score 4.0 was 42%. As a result, Score 1.0 had the highest precision in this model. The recall for this model was quite different. Score 1.0 only had 5% correct hits, while Score 2.0 had 18%, Score 3.0 had 10% and Score 4.0 had 74%. Score 4.0 had a highest recall rate. F1-score was the weighted mean of precision and recall, and the best score was 0.54 for Score 4.0. Our classifier reported 0.10, 0.25, and 0.12 for Score 1.0, Score 2.0 and Score 3.0 respectively.

For the confusion matrix (Figure 5.4), it can be seen that Score 4.0 still had more correctly predicted samples than other classes. Score 1.0 had the lowest

number of correct predictions. In this model, the misclassification still occured when Score 1.0, Score 2.0 and Score 3.0 were predicted as Score 4.0.

Class	Precision	Recall	F1-score	Support
Score 1.0	1.00	0.05	0.10	19
Score 2.0	0.39	0.18	0.25	39
Score 3.0	0.14	0.10	0.12	30
Score 4.0	0.42	0.74	0.54	62

Table 5.2: TSF using Column Concatenate for each class

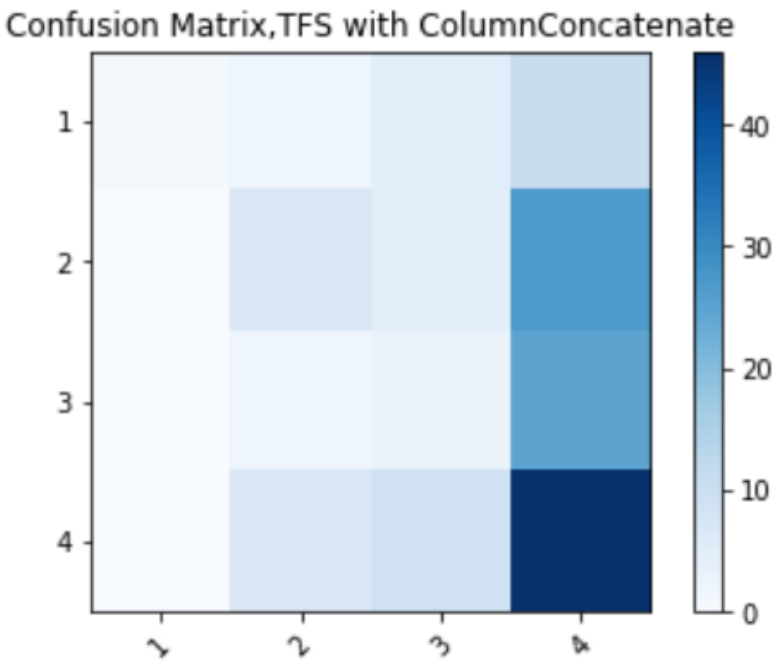


Figure 5.4: TSF using Column Concatenate Confusion Matrix

5.1.3 Evaluation for RISE using Concatenation Transform

RISE had the highest computational complexity compared to the previous two models, which meant it took the longest time to run. However, it had a higher accuracy score than other models. The accuracy score was 0.4, which meant overall that 40% of test samples were predicted correctly. It was better than the previous two models.

```
from sktime.classification.frequency_based._rise import RandomIntervalSpectralForest
steps = [
    ('concatenate', ColumnConcatenator()),
    ('classify',  RandomIntervalSpectralForest())]
clf = Pipeline(steps)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
Out[24]: 0.4
```

Figure 5.5: RISE using Column Concatenate Transform Accuracy Score
The overall accuracy of rate RISE using Concatenation Transform is 40%

From Table 5.3, the precision for Score 1.0 was 0%, which was lowest compared to other classes. Score 2.0 was 22%, Score 3.0 was 17% and Score 4.0 was 42% which represented the percentage of correct classification. Score 4.0 still had the highest precision. The model did not predict Score 1.0 well. It cannot predict Score 1.0 because of the amount of samples was too small. This situation was mentioned in Chapter 3 that the dataset was imbalanced, since most trainees got Score 4.0 and only a few of them got Score 1.0. The result also explained that our models cannot predict imbalanced datasets correctly. It had bias in the prediction.

Recall in this model was quite different. Score 1.0 had 0% correct hits, while Score 2.0 had 5%, Score 3.0 had 3% and Score 4.0 had 92% respectively. In that case, Score 4.0 had the highest recall rate. The highest f1-score was 0.58

for Score 4.0. The classifier reported 0, 0.08 and 0.06 for Score 1.0, Score 2.0 and Score 3.0 respectively.

The confusion matrix of RISE using Concatenation Transform was similar with the TFS using Column Concatenate. The result showed that Score 4.0 had more correctly predicted samples than other classes. Score 1.0 had the lowest number of accurate prediction. In this model, the misclassification happened when Score 1.0, Score 2.0 and Score 3.0 were predicted as Score 4.0.

Class	Precision	Recall	F1-score	Support
Score 1.0	0.00	0.00	0.00	19
Score 2.0	0.22	0.05	0.08	39
Score 3.0	0.17	0.03	0.06	30
Score 4.0	0.42	0.92	0.58	62

Table 5.3: RISE using Column Concatenate for each class

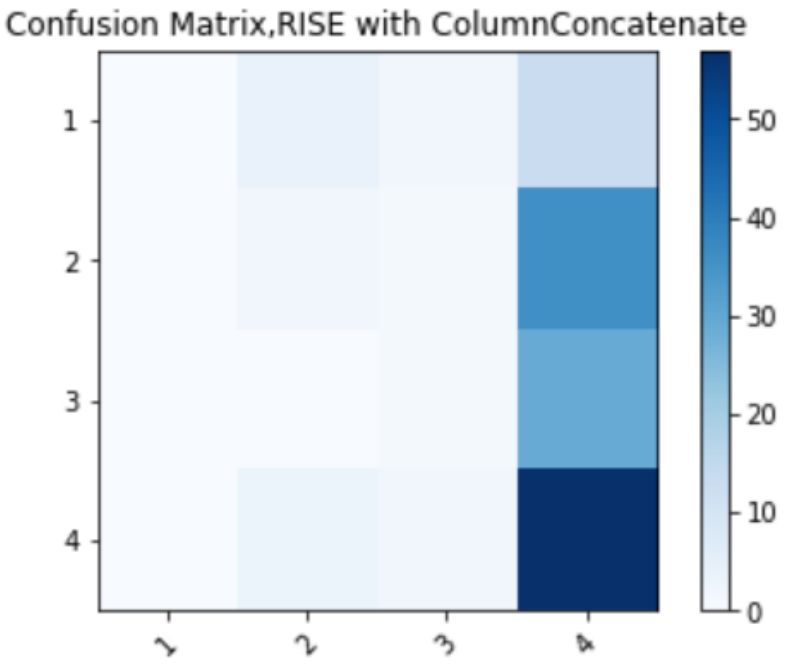


Figure 5.6: RISE using Column Concatenate Transform Confusion Matrix

5.1.4 Evaluation for 1-NN with DTW using Column Ensemble Transform

The previous three models were transformed by concatenating columns together. Other than generating a long list, the Column Ensemble Transform applied classification on each feature and assembled the most possible one. The 1-NN with DWT by using Column Ensemble Transform achieved a 0.29 accuracy, which meant 29% of test samples were predicted correctly. It had a lower accuracy score than 1-NN with DTW using Column Concatenate.

In Table 5.4, the precision for Score 1.0 was 0%, which was lowest compared to all other classes. Score 2.0 was 28%, Score 3.0 was 20% and Score 4.0 was 41% which represented the percentage of correct classification. Score 4.0 had

```
clf = ColumnEnsembleClassifier(estimators=[
    ('classify', KNeighborsTimeSeriesClassifier(), [0]),
  ("BOSSEnsemble3", BOSSEnsemble(max_ensemble_size=5), [3])
])
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
Out[34]: 0.2866666666666667
```

Figure 5.7: 1-NN with DTW using Column Ensemble Transform Accuracy Score

the highest precision. The model cannot predict Score 1.0 well. The recall in this model was quite different. Score 1.0 had 0% correct hits, while Score 2.0 had 28%, Score 3.0 had 23% and Score 4.0 had 42% respectively. As a result, Score 4.0 had the highest recall rate compared all other models. The best f1-score is 0.41 for Score 4.0. Our classifier reported 0, 0.28 and 0.22 for Score 1.0, Score 2.0 and Score 3.0 respectively.

1-NN with DTW using Column Ensemble Transform cannot predict Score 1.0 properly. The result was similar with RISE using Concatenation Transform. Neither of them can predict Score 1.0 correctly. The reason for this can be a result of imbalance in the sample amount. Score 1.0 had less samples than other class. In the confusion matrix, Score 4.0 had more correct predicted samples than other classes. Score 2.0 had a specific number of correct prediction as well.

Comparing the result with 1-NN with DTW using Concatenation Transform's result, Concatenation Transform showed better than Column Ensemble with K-NN model.

Class	Precision	Recall	F1-score	Support
Score 1.0	0.00	0.00	0.00	19
Score 2.0	0.28	0.28	0.28	39
Score 3.0	0.20	0.23	0.22	30
Score 4.0	0.41	0.42	0.41	62

Table 5.4: 1-NN with DTW using Column Ensemble for each class

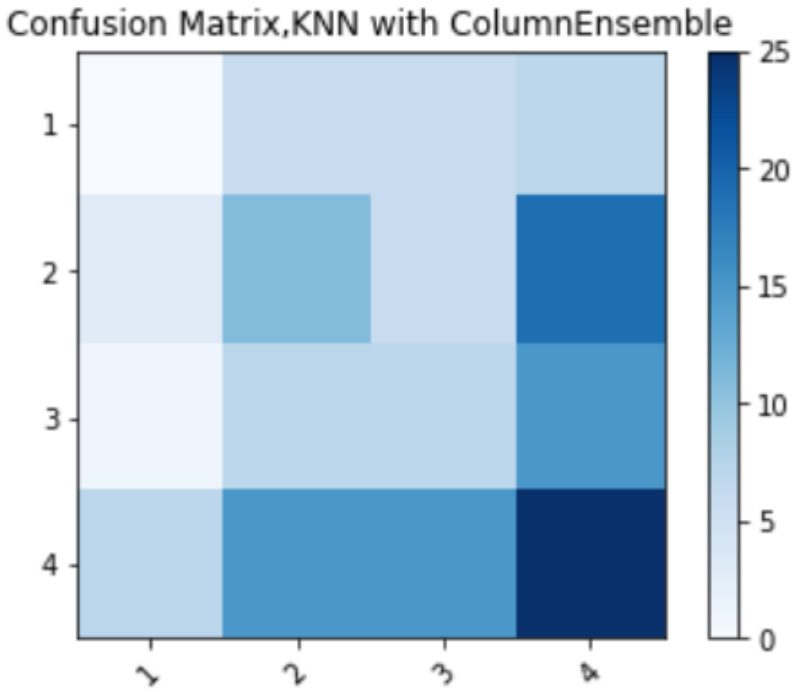


Figure 5.8: 1-NN with DTW using Column Ensemble Transform Confusion Matrix

5.1.5 Evaluation for TSF using Column Ensemble Transform

The accuracy score of TSF using Column Ensemble Transform was 0.35, it meant 35% test samples were predicted correctly. It achieved better perfor-

mance than 1-NN with Column Ensemble but lower than TFS using Concatenate Transform.

```
clf = ColumnEnsembleClassifier(estimators=[
    ("TSF0", TimeSeriesForestClassifier(n_estimators=100), [0]),
  ("BOSSEnsemble3", BOSSEnsemble(max_ensemble_size=5), [3])
])
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
Out[31]: 0.35333333333333333
```

Figure 5.9: TSF using Column Ensemble Transform Accuracy Score
The overall accuracy of rate RISE using Concatenation Transform is 35%

In Table 5.5, precision for Score 1.0 was 0%, which was lowest compared to all other classes. Score 2.0 was 21%, Score 3.0 was 0% and Score 4.0 was 40%. Score 4.0 still had the highest precision in this model. It did not predict neither Score 1.0 nor Score 3.0 well. The recall was different with previous models. Score 1.0 had 0% correct hits, while Score 2.0 had 31%, Score 3.0 had 0% and Score 4.0 had 60% respectively. In this case, Score 4.0 still had the highest recall rate F1-score was 0.48 for Score 4.0, and 0, 0.25 and 0 for Score 1.0, Score 2.0 and Score 3.0 respectively.

In conclusion, TSF using Column Ensemble Transform can predict neither Score 1.0 nor Score 3.0, since they both achieved low in all three evaluation measures. It meant even the overall accuracy was not bad, this model cannot be used to prediction of all classes.

From Figure 5.10, Score 1.0 and Score 3.0 were all 0. Score 4.0 still had more correctly predicted samples than other classes.

Class	Precision	Recall	F1-score	Support
Score 1.0	0.00	0.00	0.00	19
Score 2.0	0.21	0.31	0.25	39
Score 3.0	0.00	0.00	0.00	30
Score 4.0	0.40	0.60	0.48	62

Table 5.5: TSF using Column Ensemble for each class

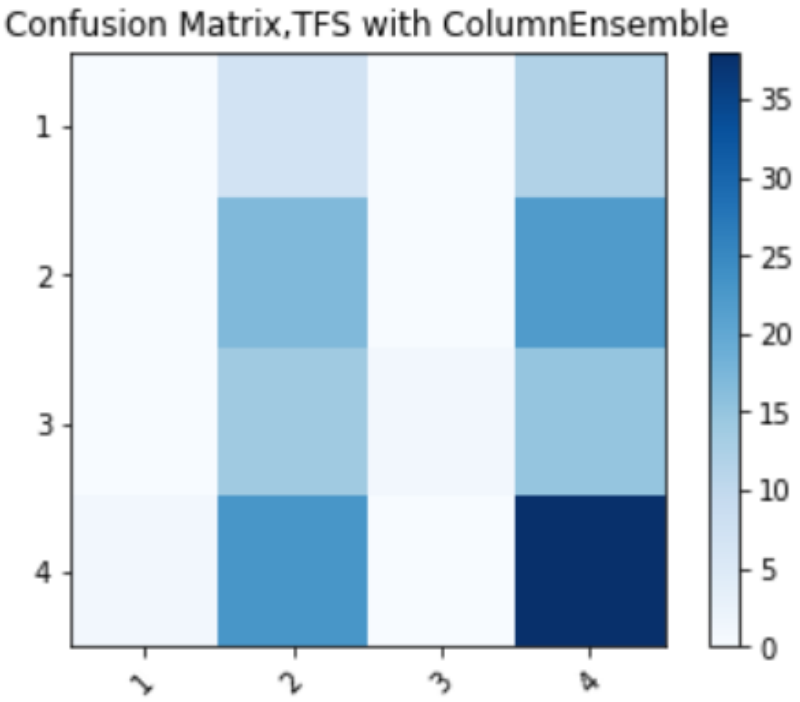


Figure 5.10: TFS using Column Ensemble Transform Confusion Matrix

5.1.6 Evaluation for RISE using Column Ensemble Transform

The accuracy score of RISE using Column Ensemble Transform was 0.27, which meant 27% of test samples were predicted correctly. The accuracy rate

was lower than other models. Considering its highest computational complexity, this model was not considered as applicable model in this project. Because of this, the model was not evaluated by other evaluation measures.

```
from sktime.classification.frequency_based._rise import RandomIntervalSpectralForest

clf = ColumnEnsembleClassifier(estimators=[
    ('classify',  RandomIntervalSpectralForest(), [0]),
  ("BOSSEnsemble3", BOSSEnsemble(max_ensemble_size=5), [3])
])
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
Out[13]: 0.26666666666666666
```

Figure 5.11: RISE using Column Ensemble Transform Accuracy Score
The overall accuracy of rate RISE using Concatenation Transform is 26%

5.1.7 Overall Comparison

Table 5.6 summarised the cumulative analysis of all versions. The highest accuracy in this table was approximately 0.4. TFS, RISE outperformed K-NN in terms of precision. For both transformation processes, TFS had a higher accuracy. Column Ensemble was superior to RISE with Column Concatenation. Column Concatenate preprocessing had higher average accuracy score than Column Ensemble preprocessing. In conclusion, Column Concatenate was a better option for our project in terms of preprocessing.

Table 5.6: Overall Accuracy Comparison Through All Models

Transform Method	K-NN	TFS	RISE
Column Concatenate	0.31	0.39	0.4
Column Ensemble	0.29	0.35	0.27

After comparing all models with each class, it can be found that Score 1.0 had the lowest performance except TSF using Column Concatenate. Score 4.0 had more correctly predict samples in the Confusion Matrix. Score 1.0, on the other hand, had the least accurate forecast. It meant that certain models cannot predict Score 1.0, and misclassified other classes in Score 4.0 during testing.

The next step was optimization. I tried to use more preprocession methods to reduce misclassifications.

5.1.8 Computational Complexity

The analysis of the model is not limited to the accuracy rate, but also the computational complexity. The computational complexity or simply complexity of an algorithm is the amount of resources required to run it. To measure that,

the *Big O* is implemented. In computer science *Big O* notation is used to classify algorithms according to how their run time or space requirements grow as the input size grows [53].

In Chapter 2, each classification algorithm was discussed. The time and space complexity of finding the *KNN with DTW* distance between two time series is $O(m^2 * n^2)$ [20], where n and m are the lengths of the two input sequences. Unlike *KNN with DTW* takes every time point to calculate, *TFS* and RISE transfer the whole time series in sub sequences. Since they decrease the time points to only k intervals, for n time series, the training complexity is $O(k * m * n^2)$ in which k is far less than m.

The two preprocessing algorithms also need to be compared. Column Concatenate is a linear function which the only scan a list of time series size, and the computational complexity is $O(m*n)$, where m is the number of dimensions and n is the lengths each time series [46]. Column Ensemble is a way of generating each classifier with one dimension time series. The computational complexity of Column Ensemble is decided by the max amount of time of classifier, which should be $O(Max(m * n))$.

The running time can be matured by using computer timing function. The overall comparison of different classifiers for training and testing with 10-fold cross validation is listed as the table by minutes.

Table 5.7: Overall Running Time Comparison Through All Models by Minutes

Transform Method	K-NN	TFS	RISE
Column Concatenate	3.92	0.86	4.49
Column Ensemble	5.18	7.06	7.25

From Table 5.7, TFS with Column Concatenate has the lowest running time

which is 0.86 Minutes . RISE with Column Ensemble has the highest computational complexity, which 7.25 Minutes. Considering all classifiers' performance, TFS with Column Concatenate can be considered as the most efficient one. With comparing between two transformation methods, it can be found that Column Ensemble's average result is higher than Column Concatenate. Therefore, if the accuracy is similar, Column Concatenate is better for our experiment.

5.2 Optimization

From the initial experiment, the performances of the classifiers were not satisfactory, where the highest score was only 0.4. This cannot be utilized in real applications. Based on the analysis of the data feature – the imbalance of four classes, the following optimization was utilized.

The main reason for the poor performance of the model could be the imbalance of classes. Since the trainees were professional pilots with years of experience, the majority of them achieved the full score. Figure 3.4 and Figure 3.7 showed that Score 4.0 had more samples than other classes. In the first test, Score 4.0 had a higher accuracy rate than other classes. In conclusion, the misclassification happened because of the imbalance of sample amount of each class.

Another reason for the models' poor performance might be the differences between each class were minute, which indicated that classifier algorithm cannot identify the threshold of each class. For example, the standards of Score 2.0 and Score 3.0 might be similar. It might cause confusing during the model training and testing.

To improve the performance, only Score 1.0 and Score 4.0 were used in second experiment. `Imbalanced-learn` package in python was also used for resampling purpose. The strategy was to take more samples from the minor class and fewer samples from the major class.

In Table 5.8, the overall accuracy scores all increased. Not only did the accuracy scores increase, the performance for each class was also better. From Table 5.9 to Table 5.14, the performance of Score 1.0 classification showed a continuous improvement. By resampling Score 1.0 and Score 4.0 using imbalance learn in similar size, the models can predict two classes properly.

Table 5.8: Overall Accuracy Comparison Through All Models With Imbalance Resampling

Transform Method	K-NN	TFS	RISE
Column Concatenate	0.53	0.66	0.66
Column Ensemble	0.60	0.53	0.43

After comparing previous results, an improvement was outstanding here. Table 5.10 to Table 5.15 showed precision and recall were balanced after resampling. However, some issues still existed in current experiment. The first was that number of samples was small, especially after resampling. In order to obtain more accurate results, the models still needed more data for training purpose. The second was that only one maneuver was used in the thesis, which might not reflect the real situation. The third was that the select features were also selected.

Class	Precision	Recall	F1-score	Support
Score 1.0	0.42	0.45	0.43	11
Score 4.0	0.67	0.63	0.65	19

Table 5.9: 1-NN with DTW using Column Concatenate Transform after optimization of each class evaluation

Class	Precision	Recall	F1-score	Support
Score 1.0	0.62	0.62	0.62	13
Score 4.0	0.71	0.71	0.71	17

Table 5.10: TSF using Column Concatenate after optimization of each class evaluation

Class	Precision	Recall	F1-score	Support
Score 1.0	0.36	0.36	0.36	11
Score 4.0	0.63	0.63	0.63	19

Table 5.11: RISE using Column Concatenate after optimization of each class evaluation

Class	Precision	Recall	F1-score	Support
Score 1.0	0.43	0.55	0.48	11
Score 4.0	0.69	0.58	0.63	19

Table 5.12: 1-NN with DTW using Column Ensemble Transform after optimization of each class evaluation

Class	Precision	Recall	F1-score	Support
Score 1.0	0.25	0.27	0.26	11
Score 4.0	0.56	0.53	0.54	19

Table 5.13: TFS using Column Ensemble Transform after optimization of each class evaluation

Class	Precision	Recall	F1-score	Support
Score 1.0	0.36	0.36	0.36	11
Score 4.0	0.63	0.63	0.63	19

Table 5.14: RISE using Column Ensemble Transform after optimization of each class evaluation

5.3 Summary

This chapter showed the results of the six models presented in Chapter 4. However, the first result was not ideal, an optimization strategy was utilized to improve the models' performances. Even though the overall accuracy was better after applying the optimization strategy, issues still remained.

Chapter 6

Conclusions and Future Work

There is a long history in time series data mining, especially in time series classification, and there are many outstanding practices and approaches. However, due to the special structure of the time series dataset, the preprocessing techniques are inadequate for the application of classifiers. In the previous experiments, the time series datasests which commonly used in experiments were formatted in equal length. In this project, the difficulty was to utilize the chosen classifier on a real aviation dataset which had various lengths.

In Chapter 4, *Proportional Scaling* was developed, which used fewer points to represent the whole time series. The purpose of this preprocessing was to cut each time series in same length. At the same time, *DataFrame* data structure stored all time series. Those two steps made it possible to utilize the classifier models. In this experiment, based on our data type, three classifiers and two

time series transformation methods were selected. In total, six different models were created. The results of models were presented and compared in Chapter 5. However the first results were not ideal, an optimization strategy was implemented.

This was an innovative practice in time series classification, especially in the aviation industry. The future work could be explored more on the preprocessing, such as how to adjust sampling numbers, how to improve the classifier efficiency, and how to deal with a huge amount of data in the real world.

Bibliography

[1] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh, "The UCR time series archive." [Online]. Available: http://arxiv.org/abs/1810.07758

[2] E. Keogh and T. Folias, "The ucr time series data mining archive, 2002," *URL http://www. cs. ucr. edu/eamonn/TSDMA/index. html*, vol. 7.

[3] A. Bagnall, H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh, "The UEA multivariate time series classification archive, 2018." [Online]. Available: http://arxiv.org/abs/1811.00075

[4] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient similarity search in sequence databases." Springer Verlag, 1993, pp. 69–84.

[5] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping," in *Proceedings of the 2001 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, pp. 1–11. [Online]. Available: https://epubs.siam.org/doi/10.1137/1.9781611972719.1

[6] B. Matthews, S. Das, K. Bhaduri, K. Das, R. Martin, and N. Oza, "Discovering anomalous aviation safety events using scalable data mining al-

gorithms," *Journal of Aerospace Information Systems*, vol. 10, no. 10, pp. 467–475, 2013.

[7] A. Dempster, F. Petitjean, and G. I. Webb, "ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels." [Online]. Available: http://arxiv.org/abs/1910.13051

[8] A. Bagnall, M. Flynn, J. Large, J. Lines, and M. Middlehurst, "A tale of two toolkits, report the third: on the usage and performance of HIVE-COTE v1.0." [Online]. Available: http://arxiv.org/abs/2004.06069

[9] A. Shifaz, C. Pelletier, F. Petitjean, and G. I. Webb, "TS-CHIEF: a scalable and accurate forest algorithm for time series classification," vol. 34, no. 3, pp. 742–775. [Online]. Available: https://doi.org/10.1007/s10618-020-00679-8

[10] A. A. Ariyo, A. O. Adewumi, and C. K. Ayo, "Stock price prediction using the arima model," in *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*. IEEE, 2014, pp. 106–112.

[11] Z. Őri, G. Monir, J. Weiss, X. Sayhouni, and D. H. Singer, "Heart rate variability: frequency domain analysis," *Cardiology clinics*, vol. 10, no. 3, pp. 499–533, 1992.

[12] Y. H. Gu and M. H. Bollen, "Time-frequency and time-scale domain analysis of voltage disturbances," *IEEE Transactions on Power Delivery*, vol. 15, no. 4, pp. 1279–1284, 2000.

[13] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping," in

Proceedings of the 2001 SIAM international conference on data mining. SIAM, 2001, pp. 1–11.

[14] H. F. Nweke, Y. W. Teh, M. A. Al-Garadi, and U. R. Alo, "Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges," *Expert Systems with Applications*, vol. 105, pp. 233–261, 2018.

[15] J. Wang, P. Liu, M. F. She, S. Nahavandi, and A. Kouzani, "Bag-of-words representation for biomedical time series classification," *Biomedical Signal Processing and Control*, vol. 8, no. 6, pp. 634–644, 2013.

[16] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with COTE: The collective of transformation-based ensembles," vol. 27, no. 9, pp. 2522–2535, conference Name: IEEE Transactions on Knowledge and Data Engineering.

[17] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.

[18] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu, "Weighted dynamic time warping for time series classification," vol. 44, no. 9, pp. 2231–2240. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S003132031000484X

[19] D. S. Hirschberg, "Algorithms for the longest common subsequence problem," *Journal of the ACM (JACM)*, vol. 24, no. 4, pp. 664–675, 1977.

[20] A. Shifaz, C. Pelletier, F. Petitjean, and G. I. Webb, "Ts-chief: A scalable and accurate forest algorithm for time series classification," *Data Mining and Knowledge Discovery*, pp. 1–34, 2020.

[21] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," vol. 239, pp. 142–153. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0020025513001473

[22] J. Lines, S. Taylor, and A. Bagnall, "Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles," vol. 12, no. 5, pp. 1–35. [Online]. Available: https://dl.acm.org/doi/10.1145/3182382

[23] M. G. Baydogan, G. Runger, and E. Tuv, "A bag-of-features framework to classify time series," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 11, pp. 2796–2802, 2013.

[24] M. G. Baydogan and G. Runger, "Time series representation and similarity based on local autopatterns," vol. 30, no. 2, pp. 476–509. [Online]. Available: http://link.springer.com/10.1007/s10618-015-0425-y

[25] L. Ye and E. Keogh, "Time series shapelets: a new primitive for data mining," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 947–956.

[26] T. Rakthanmanon and E. Keogh, "Fast shapelets: A scalable algorithm for discovering time series shapelets," in *Proceedings of the 2013 SIAM International Conference on Data Mining*. Society for

Industrial and Applied Mathematics, pp. 668–676. [Online]. Available: https://epubs.siam.org/doi/10.1137/1.9781611972832.74

[27] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning time-series shapelets," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 392–401.

[28] P. Schäfer, "The BOSS is concerned with time series classification in the presence of noise," vol. 29, no. 6, pp. 1505–1530. [Online]. Available: http://link.springer.com/10.1007/s10618-014-0377-7

[29] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *Proceedings 18th international conference on data engineering*. IEEE, 2002, pp. 673–684.

[30] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 491–502.

[31] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, 2004, pp. 792–803.

[32] H. Anton and C. Rorres, "Elementary linear algebra:: John wiley& sons," *Inc, New York, USA*, vol. 9, 1994.

[33] M. R. Berthold and F. Höppner, "On clustering time series using euclidean distance and pearson correlation," *arXiv preprint arXiv:1601.02213*, 2016.

[34] E. Keogh, L. Wei, X. Xi, M. Vlachos, S.-H. Lee, and P. Protopapas, "Supporting exact indexing of arbitrarily rotated shapes and periodic time series under euclidean and warping distance measures," *The VLDB journal*, vol. 18, no. 3, pp. 611–630, 2009.

[35] P. Roelofsen, "Time series clustering," *Vrije Universiteit Amsterdam, Amsterdam*, 2018.

[36] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[37] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12*. ACM Press, p. 262. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2339530.2339576

[38] M. A. Rahim Khan and M. Zakarya, "Longest common subsequence based algorithm for measuring similarity between time series: a new approach," *World Applied Sciences Journal*, vol. 24, no. 9, pp. 1192–1198, 2013.

[39] D. Zhang, W. Zuo, D. Zhang, H. Zhang, and N. Li, "Classification of pulse waveforms using edit distance with real penalty," *EURASIP Journal on Advances in Signal Processing*, vol. 2010, pp. 1–8, 2010.

[40] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[41] K. Yang and C. Shahabi, "A pca-based similarity measure for multivariate time series," in *Proceedings of the 2nd ACM international workshop on Multimedia databases*, 2004, pp. 65–74.

[42] Z. Bankó and J. Abonyi, "Correlation based dynamic time warping of multivariate time series," *Expert Systems with Applications*, vol. 39, no. 17, pp. 12 814–12 823, 2012.

[43] A. Bagnall, H. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh, "The uea multivariate time series classification archive, 2018. arxiv e-prints," *arXiv preprint arXiv:1811.00075*, 2018.

[44] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," *arXiv preprint arXiv:1603.06995*, 2016.

[45] J.-M. D. Maxie Claveau and K. Krishan, "Capstonr project report - opction c: Aircraft training event detection," 2019.

[46] M. Löning, A. Bagnall, S. Ganesh, V. Kazakov, J. Lines, and F. J. Király, "sktime: A unified interface for machine learning with time series," *arXiv preprint arXiv:1909.07872*, 2019.

[47] K. Kanatani-Fujimoto, B. V. Lazareva, and V. M. Zatsiorsky, "Local proportional scaling of time-series data: method and applications," *Motor Control*, vol. 1, no. 1, pp. 20–43, 1997.

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[49] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: A survey and empirical demonstration," p. 23.

[50] M. W. Kadous and C. Sammut, "Classification of multivariate time series and structured data using constructive induction," *Machine learning*, vol. 58, no. 2-3, pp. 179–216, 2005.

[51] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Pearson Education India, 2016.

[52] J. Lines and A. Bagnall, "Time series classification with ensembles of elastic distance measures," vol. 29, no. 3, pp. 565–592. [Online]. Available: http://link.springer.com/10.1007/s10618-014-0361-2

[53] A. Mohr, "Quantum computing in complexity theory and theory of computation," *Carbondale, IL*, vol. 194, 2014.