Time Series Classification Under Different Types of Noise

Aonghus Condren

A thesis submitted in partial fulfilment of the degree of

BSc. (Hons.) in Computer Science

Moderator: Thach Le Nguyen

Supervisor: Georgiana Ifrim

UCD School of Computer Science

University College Dublin

April 2019

Project Specification

Project ID: 39702

Academic Supervisor: Georgiana Ifrim

Project Mentor: Thach Le Nguyen

Subject: Machine Learning, Time Series Classification Project Type: Applied Research Software Requirements: Python Hardware Requirements: PC/laptop Target audience: Suitable for both streams Preassigned: No

General Information: Time series data is numeric data that arrives over time. For example, human motion sensors monitor movement over time (e.g., steps, jump, physio exercise), the electricity demand profile of electrical devices can be monitored over time, sensors placed in soil can monitor the soil temperature, humidity, etc. every minute or hour of the day. In many applications we need to classify time series data into groups, to understand for example if a physio exercise is done correctly or not.

There are several effective approaches for time series classification, from simple comparison-based classifiers (e.g., 1-Nearest Neighbour with Euclidean distance), to complex deep learning models (e.g., Fully Convolutional Neural Network). Not enough is known though about how these approaches behave in the presence of noise in the data. For example, some techniques expect equal length time series for training and prediction, and a pre-processing step has to be carried over the data if the collected time series have variable length. Time series can also be affected by shift, translation, scaling, stretching, missing values, etc. The goal of this project is to study how the existing state-of-the-art time series classification methods are affected by different types of noise.

Core:

-Study the time series classification literature and decide on 3 common types of noise that affect time series data and some appropriate transformations for inducing that type of noise in time series (e.g., variable length, shift, scaling).

-Apply the 3 different types of noise-inducing transformations to the UCR time series classification benchmark. -Study the time series classification literature and select 5 state-of-the-art classifiers to evaluate. -Compare the 5 state-of-the-art classifiers on the original and distorted UCR benchmark, and discuss how different types of noise affects each classifier.

Advanced:

-Extend the portfolio of noise inducing transformations, e.g., variable length, shift, scaling, stretching, translation, etc.

-Extend the evaluation of state-of-the-art classifiers and the discussion of how different types of noise affects these classifiers.

Data & Analysis:

Time series classification benchmarks: http://timeseriesclassification.com

The scale is in Mb. It is static data, presented as labelled time series (i.e., a sequence of numeric values over time and an associated class label). It needs to be accessed by the student, but the supervisor also has a local copy. The analysis involves applying different transformations to the data (e.g., amplitude scaling, shifting)

and applying existing time series classifiers to the transformed data. The code for existing time series classifiers is available online as open source code and the supervisor also has a local copy.

Abstract

In this report, we will be examining the challenge of time series classification and in particular, how classification methods deal with noisy data. Time series classification has practical uses in many fields, ranging from environmental or health science to stock market analysis and satellite imagery data. It's, therefore, hugely important. Time series data from the real world is often noisy and imperfect and while many new classification methods have been proposed in recent years less research has been done to analyse the impact of noise on time series classification. Noise makes the classification of time series more difficult as it can distort the shapes in the time series that are used to classify them. In our research we will test several different classification methods, namely:

- Dynamic time warping (DTW)
- Bag of Symbolic-Fourier-Approximation symbols (BOSS).
- Word extraction for time series classification (WEASEL)
- Linear models + symbolic representations (Mr-SEQL)
- Fully convolutional networks (FCN)

To examine how each classifier is affected by different types of noisy data we use the CBF dataset from the UCR time series classification benchmark as a model to create test datasets. The noisy features we choose to apply to these datasets are, Gaussian noise, Variable length time series, Amplitude scaling and Signal shape time warping. We apply noise at incremental levels of saturation in the datasets, to better understand the effect the noisy parameters have. We find that the classifiers have certain strengths and weaknesses to the various types of noise.

Table of Contents

1 Introduction

A Time Series (TS) is a sequence of data points taken at successive, equally spaced points in time. Time series data, therefore, can be gathered from any source that collects data over time. Time series data contains vast amounts of information, analysis on this data can reveal vitally important statistics, trends and other characteristics. The real-world applications are therefore vast. Time series data is encountered in biomedicine (Argyro Kampouraki, 2009), in human activity records (Nweke et al, 2018) and even in steel quality prediction (Mehdiyev N et al. , 2017). The wide variety and availability of time series data coupled with its applications to real-world problems means that the time series data is becoming more prevalent. We can see this through the patterns of development of Time Series databases, as shown in Figure 1. This figure was taken from the site db-engines, which is a website dedicated to gathering all manner of current information about databases. The graph shows the historical trend of the most popular databases, and we can see that time series databases are by far the most popular. The rankings are generated by taking the best three systems per category and calculating the average of their ranking scores. Then to allow comparison the initial value is normalized to 100.

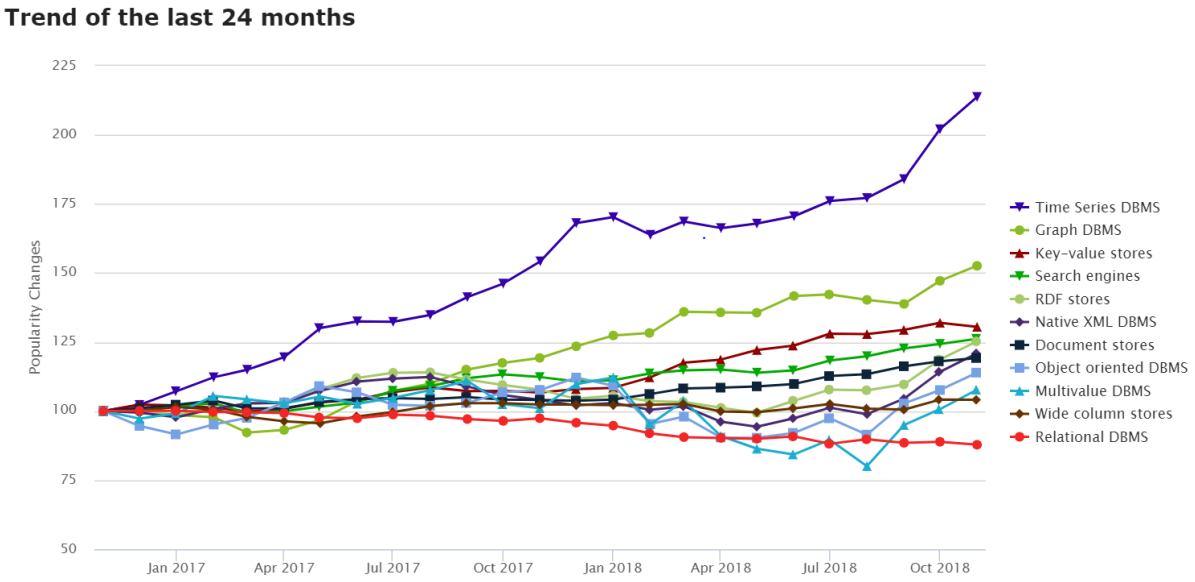

Figure 1. Source: https://db-engines.com/en/ranking_categories

With such a prevalent amount of data comes the problem of Time Series Classification (TSC). Time series classification is the task of determining which predefined class a time series belongs to and classifying or labelling the time series with that class. For example, a classification task for human activity could be to classify the instances when a person wearing a heartbeat monitor is running, jogging or walking based on their heart rate. A typical step in solving the classification problem is training a method to be able to determine the trends in data and to correctly classify them. This is a non-trivial task as a classifier must be able to differentiate between noise in the time series and the actual trends that it must use to classify. This is a prevalent problem in classification and there are hundreds of new classification methods being proposed, in an attempt to produce the most accurate classifier (Bagnall, 2017). One of the most popular methods of classification is DTW. When used with a nearest-neighbour search method, DTW has been shown to perform quite well and makes an

excellent baseline method that holds up against many other classification methods (Bagnall, 2017). There are also more complex deep-learning methods such as Fully Convolutional Networks which also performs well and achieves high accuracy (Fazle K et al, 2017). Other newer methods such as Word Extraction for Time Series Classification (WEASEL), are also both fast and accurate (Schäfer et al, 2017). WEASEL is a symbolic method that operates by transforming the time series into vectors using a sliding-window approach, it then analyses the feature vectors utilizing a machine-learning approach.

The University of California, Riverside (UCR) time series classification archive (Keogh et al, 2018) contributes greatly to the large amounts of classifiers being created in recent years (Bagnall, 2017). The UCR benchmark is a superset of synthetic and real data from a range of applications. These datasets were created to give an alternative to TSC research being done on a single dataset. The UCR archive provides 85 distinct datasets on which one can run their classifiers and it has become hugely popular with researchers with more than 3000 downloads as of 2015. The UCR archive is used across many projects (Bagnall, 2017), (Hassan I F et al, 2018). However, the UCR archive is, by the admission of its authors a poor proxy for real-world data. As the data is pre-processed it implicitly contains assumptions about the data, for example, that all the time series are of equal length (Hu, 2013), which is not the case in many applications.

While there are many projects testing the accuracy of classifiers there has been much less research done on the effect noise has with regards to time series classification. Since the UCR datasets are expansive they are a good basis for testing the accuracy of classifiers, however as the datasets contain a baseline level of noise many of these classifiers have not been tested thoroughly with regards noise. The question of how noise affects time series classification is an important one as real-world data is, likely, noisy and imperfect. As solving real-world problems could be viewed as the ultimate use for classifiers, the manner in which classifiers are able to maintain accuracy while dealing with noisy data becomes a very interesting and important question. The effect that noise has on time series classification has been analysed in other research, its effect has been shown regarding the baseline DTW classifier and the symbolic BOSS classifier (Schäfer, 2015). This research shows that the BOSS method is highly invariant to noise, while DTW suffers from high levels of noise.

In testing the BOSS and DTW classifiers one type of noise was used. In our report, we use several different types of noise in order to understand how well each classifier deals with noise. We also want to see if in fact there are certain noise types that produce a particularly difficult time series to classify.

In order to assess the classifiers, we created datasets modelled off the CBF dataset from the UCR archive. By altering the parameters of the code used to generate the CBF dataset we are able to induce different types of noise into the time series. The different types of noise we will employ to assess the classifiers are the following:

- *Gaussian noise –* Gaussian noise is generated by varying the standard deviation of a Gaussian distribution when generating some of the time series. This is the baseline noise that is in the CBF dataset. The CBF time series are generated using a standard Gaussian with a mean of 0 and a standard deviation of 1.

- *Variable Length noise –* Variable length noise is found often in real-world data, this is somewhat overlooked in the UCR dataset however as all series there are of fixed length. Rather than having all the time series of equal length i.e. containing the same number of data points in the series, we build time series with varying numbers of points in the series.

- *Amplitude Scaling noise –* we will vary the scalar value of the time series to increase or decrease the signal shape size. This will replicate real-world examples that we often find when using electronic sensors. Fluctuations can scale the signal to higher or lower than normal, we can reproduce this by, for example, increasing the variation of the amplitude scaler which would simulate a positive or negative spike in the frequency.

- *Signal Shape Time warping –* We will warp the length of the signal shape so that will be more variation in the duration that a signal shape is prominent on the time series. This will represent variations that are often found in time series. By increasing the period for which a signal shape is pronounced on a time series we can replicate real world scenarios. For example, a fit joggers heart rate taking longer to increase than an unfit jogger, despite them jogging at the same pace.

To assess how each classifier deals with the noisy data we create datasets for each of the noise types we are using. To this end, we induce noise incrementally at three different levels, 20%, 40% and 60%, this will highlight how increasing levels of noise affect the classifiers. We also create a CBF dataset without altering any of the original parameters to act as a baseline accuracy for the classifiers. In total, we create 13 distinct datasets with each containing different levels and types of noise. Each dataset is split into train and test partitions to allow for an unbiased test of each model's accuracy. In creating these datasets, we follow the design of the existing dataset from the UCR benchmark which has 30 training example TS and 900 test TS. We create the datasets with a fixed length of 128 for the time series, with our variable length dataset being the exception to that rule.

2 Background Research

2.1 Times Series Classification

In almost all scientific fields, measurements are performed over time. The collection of this kind of data leads to the formation of a time series. Time series classification is concerned with assigning one of a set of predefined classes to a time series. A time series is an ordered sequence of real-values that have been recorded over time. We can denote a time series with the following definition:

Definition 2.1. *$T = (v_1, v_2, \ldots, v_n)$, where T is a vector of values, n is the length of the time series and vi represents numeric values collected at each time stamp, i = [i, n].*

Time series classification is concerned with assigning one of a set of predefined classes to a time series. The following definition defines time series classification:

Definition 2.2. *Given an unlabelled time series T, assign it a single class c_i such that left $\{c_i\}$ is a member of C, where C is a set of predefined class.*

While this is a relatively straight forward task for humans, it remains non-trivial for computers. Humans can ignore small fluctuations (noise) and focus instead on the shape of patterns in a time series, where computers find it more difficult (Esling, 2012).

2.1.1 Time Series Datasets

The UCR archive has played a vital role in furthering the field of time series classification. It provides researchers with a common benchmark on which to test their classification methods (Bagnall, 2017). As mentioned, the UCR time series classification archive is a superset of synthetic and real datasets from a range of applications. In our project, we will be using the design of the CBF (Cylinder, Bell, Funnel) dataset from the UCR archive (Y. Chen, 2015).

The CBF dataset is comprised of simulated data. It contains three distinct signal shapes that form the three predefined classes of the dataset (Saito, 1994). In his research, Saito discusses the creation of each of the distinct signal shapes that are found in the dataset. Each shape is created using a baseline Gaussian noise (discussed below) and an offset term which is unique to each of the classes. These offset terms produce the "Bell", "Cylinder", and "Funnel" shapes. The "cylinder" signal shape is categorised as two step-edges in the TS with constant values in the centre between them, the "bell" signal shape is a positive ramp around the centre followed by a step edge and the "funnel" signal shape is a step edge followed by a negative ramp around the centre of the time series. We can see an example of each type of signal shape in Figure 2 below. It shows each class of time series without any noise applied. Using this basis as our predefined classes we apply noise to the datasets we'll use in our research.

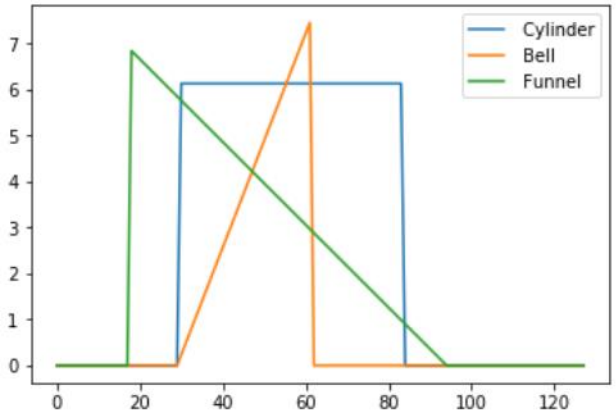

Figure 2 – an example of each class found in CBF

2.2 Noise in Time Series

Time series data from the real world is often messy and imperfect. We can refer to these imperfections as noise in the time series. Noise in a time series distorts the event in the TS that we wish to classify, and this has a negative effect on the attempts to classify the time series. Figure 3 below shows, the effect of applying noise to the TS under the baseline CBF assumptions of noise. When compared to Figure 2 above we see injecting even a small amount of noise affects the TS.

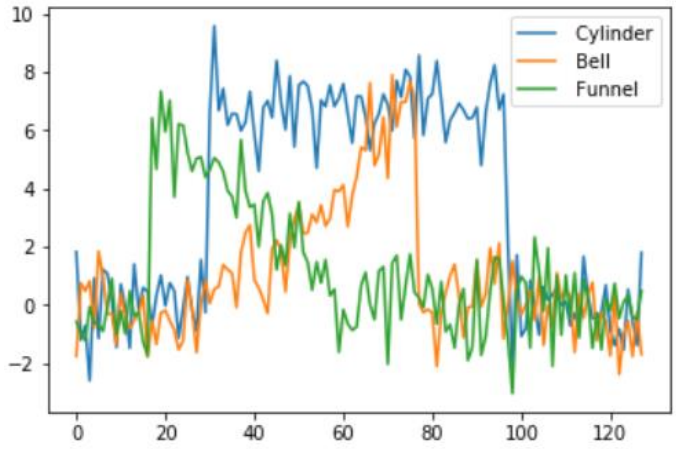

Figure 3 – Each class of TS with baseline noise

There are many different types of noise from white noise, a process most frequently seen in time series in which data points at different times are not correlated (Kang, et al. 2014), to horizontal alignment each of which can have adverse effects on the classification accuracy. The noise that we will induce in our datasets has been chosen as it best represents the types of noise one would most expect to find in real-world time series.

The UCR time series database is an excellent resource for testing classifier accuracy, but, when examining the impact of noise on the accuracy of classifiers it is less helpful. As the data is pre-processed it means that the noise levels in the datasets do not approximate real world or noisy data well (Hu, 2013). There are also some types of noise that are frequently found in the real world which are not represented in the data, for example, there is no data of variable length in the UCR datasets.

2.2.1 Types of Real-World Noise

Variable Length – A variable length dataset is that which has time series of various lengths. Variable length data is often found in real-world examples of time series data (Arash Jalalian, 2012). In this paper it discusses how the use of dynamic time warping can be used to classify TS of variable length, it also proposes another method (GDTW-P-SVM) which combines potential support vector machines(P-SVMs) and Gaussian dynamic time warping (GDTW). This classification method enables fast and efficient classification on both variable length and fixed length TS.

Gaussian Noise – Gaussian noise is a statistical noise that has a probability density function that's the same as that of the normal distribution. We find this distribution frequently throughout nature making it a good representation for real-world noise when applied to a time series.
The probability density function P of a Gaussian random variable z is:

Definition 2.2.1. $p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$ where z represents the value, μ is the mean value and σ is the standard deviation.
The standard Gaussian noise applied to the CBF dataset above is Gaussian noise in the range of [0,1].

Amplitude Scaling – Amplitude scaling occurs frequently when we're dealing with time series that were generated from electronic noise. Random fluctuations in the noise caused by the electronic device, for example, is where amplitude scaling comes into effect(Batista. G et al, 2011). Amplitude scaling involves multiplying the amplitude of a signal by a real number (Cuff, 2011).

Signal Shape Time Warping – Signal shape warping is used to account for natural variations in time series data. This is particularly true with human activity (Ada Wai-chee Fu, 2006), for example, to account for the acceleration of a group of runners. Signal shape warping is applied to the signal shape of a given time series and involves either reducing or extending the length of time the signal shape is pronounced on a time series.

2.3 Classification methods

2.3.1 Dynamic Time Warping

Dynamic time warping is a measure of distance between two time series. It's used to align two time series. To do this we try and find the optimal path between them. Suppose we have two series $A = (a1, a2, \dots, an)$ and $B = (b1, b2, \dots, bn)$, to find the optimal path we create a matrix M such that M = A*B. The path between the two time-series then is a series of points, or pairs of indices that denote the traversal of the matrix *(Berndt, D.J et al,1994)*. To find the best match between these time series we simply need to find the path that minimises the distance between them. The overall distance of the path is the total sum of distances between each individual element on the path. While finding the optimal path there are several criteria that must be satisfied *(Sakoe, H.*1978)

- The path may not turn back on itself, both indexes must increase or remain the same.
- The path can advance by at most one step at a time.
- The path begins at the bottom left of the matrix and ends at the top right.

These constraints restrict the movements of the warp path and so limits the number of paths that need to be considered. It has been shown that using the 1-Nearest neighbour classification method with DTW distance to classify the neighbours produces very accurate classification results (Xiaopeng Xi, 2006). Due to this, DTW is a common baseline method for TSC, which is the purpose we use it for.

The k-nearest neighbour algorithm can be simply explained as follows:

A positive integer K is declared along with a given data point b, we examine the K nearest data points to b, whatever the most common classification is for the nearest points, we classify b as (Bronshtein, 2017).

2.3.2 Bag of Symbolic-Fourier-Approximation Symbols

The BOSS model uses several key building blocks which allows for fast and accurate performance (Schäfer, 2015). First, it extracts sliding windows of fixed length for a given time series. This windowing method breaks up the time series into a series of sequences.

Then the Symbolic Fourier Approximation (SFA) (Schäfer M. H., 2012) is applied to each sequence generated by the windowing function. SFA utilizes a finite alphabet of symbols to represent the data from the sliding windows. It provides low pass filtering and quantisation to help to reduce noise from the time series. This produces sequences of symbols (SFA word) which represents each sliding window.

Using a histogram of the SFA words then indicates the structural similarity. By using a string representation, it allows for string matching algorithms to be applied which has additional noise reducing effects.

When it comes to classification of the data, it's based on the Quantisation intervals we compute the SFA words for both the train and test data. We retrieve the precomputed and labelled quantization intervals from a training dataset. In short, it transforms the input time series with SFA, and uses the generated histogram for classification.

2.3.3 Word Extraction for Time Series Classification

The WEASEL classifier is a successor to the BOSS classification method. It is both fast scalable and accurate when classifying time series data (Schäfer et al, 2017). The WEASEL method introduces several novel ideas which allow it to operate almost as accurately as the COTE classification method (which is considered the overall best method), but at a much faster rate of training and classifying. WEASEL was created to operate in the following manner.

Firstly, WEASEL extracts different length normalized windows from the time series. Using the Fourier transform (breaks down a time function into the frequencies that make it up) each window is approximated. Those Fourier coefficients are discretized into a word with the use of information gain binning, this also chooses discretization boundaries to separate the time series classes.

From this, a bag-of-patterns is created by, built using bigrams and unigrams (usual method only looks at unigrams). The Bag-of-Patterns encodes the unigrams, bigrams and variable length windows. Chi-Squared, a method of feature selection, is then applied which filters unimportant words. With this done WEASEL then builds a discriminative feature vector, to which a logistic regression is applied.

The WEASEL classifier draws on previous work from the bag-of-patterns method (J. Lin, 2007), but enhances it using Fourier transformation (Högqvist., 2012)

2.3.4 Fully Convolutional Network

Fully Convolutional Network (FCN) models are regarded as state-of-the-art models when it comes to classification (Karim, 2017). FCNs classify time series using a series of layers. The first layer of an FCN is its convolutional layer. This computes two functions creating a third which is then passed through to the following layers. The batch normalisation block (Ioffe, 2015) follows from the convolutional layer and then the ReLu (Rectified linear unit) activation layer. Convolution is executed by sliding the filter over the input, the features are selected through this process. In every location, the operations are performed to sum the result onto a feature map. The convolution operation uses three 1-dimensional kernels, using this the final network is built using three convolutional blocks, which are stacked one on top of the other, each with its own filter size. Batch normalisation is applied to speed up the convergence speed of the functions and also, to help to generalise them. Once the features pass the convolution layer, they are fed into the global average pooling layer (M. Lin, 2013). This layer reduces the weights of the features which then makes it easier for the final SoftMax layer to produce labels for the features, i.e. to classify the data.

2.3.5 Linear Models and Logistic Regression

Time series classification using sequence learner (Mr-SEQL) is a classification method that uses linear models combined with multiple symbolic representations of time series to form a competitive and accurate classifier (Nguyen T.L. et al, 2018). Mr-SEQL uses two symbolic transformation techniques to represent the time series to a symbolic representation. These are SFA (Schäfer M. H., 2012), which is described above, and Symbolic Aggregate Approximation (SAX) (Lin et al, 2007).

In the first step of SAX transformation the time series in z-normalized and split into a number of equal length sequences, then each sequence is replaced with a mean value. These results form a piecewise aggregate approximation (PAA) vector. Using a given alphabet a look up table is then created with each symbol in the alphabet being associated with an interval (a range of values). Finally, each entry from the PAA vector is replaced by a symbol from the alphabet using the look up table that was created leaving a symbolic representation of the original time series.

Multiple symbolic representations can then be fed to the sequence learner (SEQL). SEQL is an implementation of a greedy coordinate-wise gradient descent technique which classifies the symbolic representations of our time series.

3 Implementation

3.1 Data Generation

The main source of data generation was the CBF dataset which was found in the UCR time series archive (Keogh et al, 2018). The UCR archive is an archive of pre-processed datasets The CBF dataset is made up entirely from synthetic time series data. There are three distinct classes found in the dataset which are differentiated by the signal shapes Cylinder, Bell and Funnel. Each shape is created using a baseline Gaussian noise and an offset term which is unique to each of the classes. Using a Python adaption of the algorithms proposed (Saito, 1994), we create time series with three distinct classes using the signal shapes. To produce each shape, we use the following methods.

Algorithm 1. Cylinder Shape:

$$[\,(6+n)*xab(t)\ +\ random.normalvariate(0.0,1.0)\ for\ t\ in\ range(1,tslength\ +\ 1)]$$

Algorithm 2. Bell Shape:

$$[\,(6+n)*xab(t)*(t-a)/(b-a)\ +\ random.normalvariate(0.0,1.0)\ for\ t\ in\ range(1,tslength+1)]$$

Algorithm 3. Funnel Shape:

$$[\,(6+n)*xab(t)*(b-t)/(b-a)\ +\ random.normalvariate(0.0,1.0)\ for\ t\ in\ range(1,tslength+1)]$$

In the presented algorithms n is a random normal value on the normal probability distribution where the mean is equal to 0 and the standard deviation of 1, 'a' is a random value in the range [16,32], 'b' is a random value in the range [32,96] plus 32 (the maximum range of a) and 'tslength' is equal to 128, i.e. the length of the time series. Every iteration of the algorithm generates a single value on the time series, the for-in loop ensures that the algorithm is run 128 times to give us our time series of fixed length. xab(t) is a function call that takes the current value of 't' as an input and returns the value 1 if 't' falls between the range of 'a' and 'b' and 0 if it does not. When 1 is returned the offset term is applied to the value 't' which then forms one of the distinct signal shapes for the time series.

Algorithm 1 is used to create the "Cylinder" shape, the offset term, $(6+n)*xab(t)$, when applied simply adds a constant of $(6+n)$ to the given data point in the TS. This produces the two step edges, as $(6+n)$ is applied to the data points of the time series only while 't' is in the range of [a,b]. In Figure 4 we can see the baseline time series that this algorithm produces.

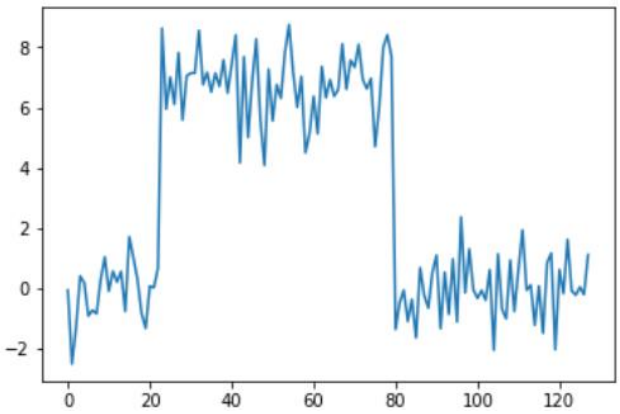

Figure 4 – An example of the 'Cylinder' signal shape

To create the "Bell" signal shape we use Algorithm 2. This algorithm uses the offset term $(6 + n) * xab(t) * (t - a)/(b - a)$. This produces a positive slope followed by a step edge. When the offset term is first applied to the time series values 't' = 'a' (because the offset term is applied only when 't' is in the range of [a, b]), therefore, (t-a) = 0, for the first value the offset term is applied to. So as the value of 't' increases by 1 after each iteration of the algorithm, the value of the fraction $(t - a)/(b - a)$ also increases, creating the positive ramp. A step edge is formed once 't' > 'b' as the offset term is no longer applied to the data. See Figure 5 for an example of such a signal shape in a time series.

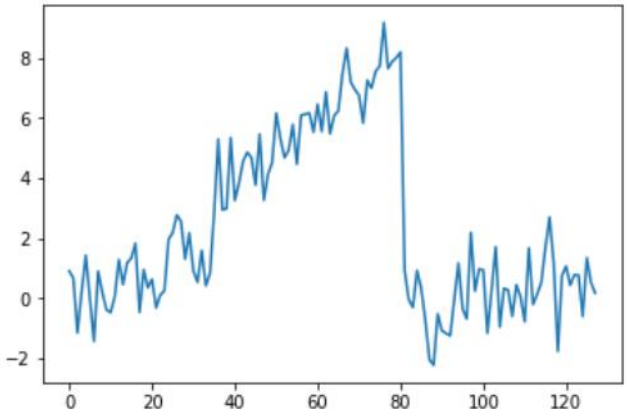

Figure 5 – An example of the 'Bell' signal shape

Algorithm 3 forms our final signal shape which is the "funnel" shape. The offset term, $(6 + n) * xab(t) * (b - t)/(b - a)$, forms a step edge which is followed by a negative ramp. When the offset term is being applied to the values in the time series 't' is in the range of [a, b]. So, when we look at the first value the offset term is applied to, the equation $(b - t)/(b - a)$, will produce the highest value as 't' = 'a' and 'b' is the maximum value the offset term is applied to, thus forming the step edge. While 't' trends upwards as we advance along the time series the equation $(b - t)/(b - a)$ produces smaller and smaller values, which when applied to the rest of the offset term then produces our negative ramp in the time series. Figure 6 shows an example of such a time series where algorithm 3 has been applied.

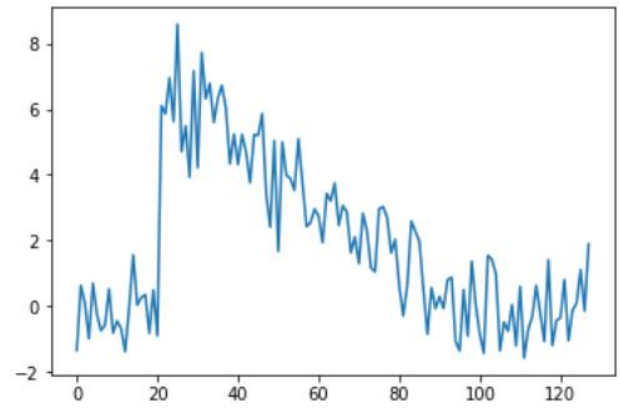

Figure 6 – An example of the 'Funnel' signal shape

These are the signal shapes that we want our models to classify. The goal of using the CBF dataset is to be able to reliably test how different models deal with the different challenges of the noisy data, as

we are trying to see how each of the classifiers, DTW, BOSS, FCN, WEASEL and Mr-SEQL react to different noise constraints. Therefore, it's important to feature a single data source while constructing our datasets as the only variation we want between our datasets is the type of noise applied. This means that the noise we induce in the data will be the only factor influencing the classification methods accuracy, as variation in data sources could bias our results.

We created several datasets with each modelling the initial CBF dataset from the UCR archive. The aim of having these datasets was to test the model's accuracy when each of our noise types (Gaussian noise, variable length noise, amplitude scaling noise, and signal shape time warping noise) were introduced into the data. These types of noise were chosen because they are all frequently found in real-world data and so understanding how the classification methods handle these noise types may produce some interesting results and implications. In each case we created the signal shapes in the same manner the only differences between the datasets then, is the amount of noise and type of noise added.

To better understand how noise affects the classification methods, we opted to generate datasets with incrementing amounts of noise. We generated noise to the following levels:

- Noise is applied to 20% of the data
- Noise is applied to 40% of the data
- Noise is applied to 60% of the data

We choose to observe the accuracy of the classifiers at these levels as noisy features have been observed at similar levels in other work (Bagnall, A. 2017). 40% is a common level of noise to apply To do this we created 3 datasets for each of our 4 types of noise (Baseline noise must remain a single dataset, as there is no additional noise we can incrementally increase). In each of the datasets, 20, 40 or 60 percent of all the TS will be noisy TS. In each of the datasets, 20, 40 or 60 percent of all the TS are noisy TS. Table 1 below shows the breakdown of this. We can see the number of noisy TS and the number of baseline TS that will be in each of the datasets.

Type of Time Series	Noise at 20%	Noise at 40%	Noise at 60%
Baseline	960	720	480
Noisy	240	480	720

Table 1: Breakdown of the number of noisy/baseline TS in each dataset

By creating datasets with varying amount of noise for each noise type we use, we can see, for example, at what level of noise does classifier accuracy being to degrade.

When creating the datasets to test the models we created training and testing datasets, this allows us to train the model on the data and have some reserved data that the model in question will have never seen before, so it provides an impartial test on the accuracy of the models. To generate the testing and training datasets, rather than partitioning a single dataset into test/train, we create two datasets one to represent the training data and one to represent the testing data. We generated the datasets this way to ensure that the percentage of noise is consistent between the two datasets. We do this for each of the noisy datasets, for our baseline data however, as there is no noise injection, we create it by partitioning a single dataset.

When generating the training and test data we wanted to model our data to be similar to the CBF dataset. However, we created the training data to contain 300 time series rather than the 30 in the original CBF dataset as we felt 30 was too little training data.

We created the datasets to test different noise types in the data and how they affect the models. Each dataset is made up of 1200 time series, which are then split into a training set of 300 time series and a test set of 900 time series. We use a large test set to ensure that any misclassifications do not appear to be more significant than they are. Each of the distinct classes of the CBF dataset, Bell, Funnel and Cylinder, appear 400 times in each dataset. This is to ensure that no class is over or under-represented in the datasets.

When creating these datasets, we used the following Python code:

```
CYLINDER=1
BELL=2
FUNNEL=3

def Keoghcbf(o, tslength=128):
base = tslength / 8

""" This is the Cylinder-Bell-Funnel (CBF)
   N. Saito, Local feature extraction and its application
 using a library of bases. PhD thesis, Department of Mathematics,
  Yale University, 1994."""

  a = random.randint(base,base*2)
  b = base*2 + random.randint(base*2,base*6)
  n = random.normalvariate(0.0,1.0)

   def xab(t):
    if ( (t>=a) and (t <=b) ):
      return 1.0
 return 0.0
  if(o == CYLINDER):
    return [(6+n)*xab(t) + random.normalvariate(0.0,1.0) for t in
range(1,tslength + 1)]
  elif(o == BELL) :
    return [(6+n)*xab(t)*(t-a)/(b-a)+ random.normalvariate(0.0,1.0) for t
in range(1,tslength + 1)]
  elif (o == FUNNEL):
    return [(6+n)*xab(t)*(b-t)/(b-a)+ random.normalvariate(0.0,1.0) for t
in range(1,tslength + 1)]
  else :
return None
```

Snippet 1 – code to create CBF inspired synthetic time series data. Source: (Saito, 1994)

This code is a function that must be called upon to generate a single time series of length 128. When we call it, we must specify which signal shape we want to apply to our time series. For example, calling the function as 'cbf(1)' will return a time series with a cylinder shape. This function is what we use to create each of our datasets, we can induce different types of noise by changing some parameters of the function (explained below).

In order to determine the accuracy of the classifiers, we need to know to which class a TS belongs. Without this information, the classifiers would be unable to know if their prediction of the TS was correct, and so, unable to produce an accuracy score. The function in Snippet 1 above generates only TS without a class label, therefore, we must apply a label to each TS ourselves. To do this we add the

predefined class label (1, 2 or 3) to the beginning of each TS. Figure 7 below shows an example of classifiable TS.

class	0	1	2	3	4	5	6	7	8	...
1	0.253512	0.771842	1.640482	0.412237	1.227898	0.093639	-1.016611	0.623133	-0.169205	...
2	1.598154	-2.393694	-0.877214	-0.529318	-0.259262	-2.394686	-0.335624	0.950562	-1.291309	...
3	-0.026574	2.833155	-2.392277	0.936553	-2.547324	-0.080960	0.917525	0.379414	-1.273411	...

Figure 7 – Time series represented with their corresponding class label.

As we are generating TS to be classified, we need to declare which predefined class each TS belongs to so the classification methods can take them as input.

3.1.1 Baseline Dataset

The first dataset to be created was made using for loops to call on the unaltered code from snippet 1. To create time series data for each class, we created and ran three for loops specifying one of the three classes for each for loop. We ran each for loop 400 times and stored the results in three distinct datasets. Then for each dataset, we added a column to the beginning of the dataset to denote the class (1=Cylinder, 2=Bell, 3=Funnel) that the time series belongs to. Then we joined each dataset together to create a single dataset containing all the time series for each class. The final step was to then split it into our test and training sets and shuffle the rows of each dataset, ensuring that the classes appeared in random order.

The training and testing datasets contain standard normal noise as used in the CBF dataset. These datasets will provide a benchmark for the accuracy of the models. With this benchmark accuracy rating, we can compare how well each model operates with its performance under the baseline conditions of the CBF dataset.

3.1.2 Gaussian noise dataset

We created to test how the classification models deal with noise is a dataset that contains varying degrees of Gaussian noise. The more noise that's added the harder it is to identify the different classes as the signal shapes can get lost in the white noise if the level is very high, Figure 8 below is a clear example of this. We see how the TS with Gaussian noise applied appears much more erratic when compared to the baseline TS. To vary the levels of Gaussian noise found in the time series we change the standard deviation of the random value in the normal distribution found in algorithm 1, 2 and 3. For example, in this case $random.normalvariate(0.0,1.0)$ the standard deviation is 1. By altering these features, we can increase the levels of noise found in the time series.

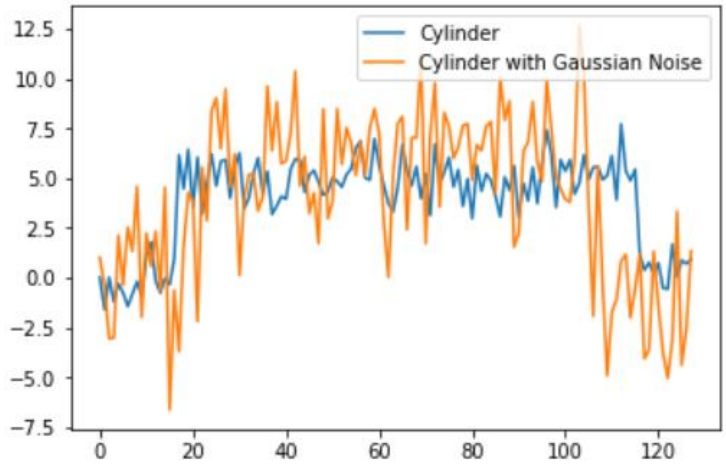

Figure 8 – A time series with added Gaussian noise(orange) with a baseline time series (blue).

We created 6 distinct datasets, with each dataset containing TS from a single class. Each class is represented by two datasets one for training data and one for testing data. We began by altering the code in snippet 1(or more specifically algorithms 1, 2 and 3). We add an extra parameter to the function call, we then use this parameter to define the standard deviation, so we have $random.normalvariate(0.0, sigma)$. To generate TS with higher levels of Gaussian noise we use a standard deviation of 3. This increases the possible range of noise that is applied to the dataset. With this increased standard deviation, we get higher noise levels as there is more deviation from the baseline TS.

To populate each of our datasets, we ran two for loops, the first for loop calls the altered version of the function from snippet 1, with baseline parameters. The second for loop does the same only we call the function with the noise-inducing parameter (sigma = 3) to produce our noisy TS. This creates 6 datasets with all the noise we wanted to induce on the TS. Then we added a column containing the class labels of the TS to the beginning of the dataset to denote the class that each TS in the dataset belongs to.

We then joined our three training datasets to create our single training dataset containing 300 TS that has all the time series with the various levels of Gaussian noise. We do the same for our testing datasets to create our final testing dataset, containing 900 TS. Finally, we shuffled the order by which the time series appear in the dataset so that all TS of a given class don't appear at once.

3.1.3 Variable length dataset

We created the dataset to contain time series of variable length. The initial length of each time series was 128, which is denoted by 'tslength' in Snippet 1. To change the length of the time series we can simply change the value of 'tslength'. In Figure 8 we see a time series that's twice the normal length of the baseline TS. To create our variable length TS, we changed the definition of 'tslength' to be defined upon each call of the function rather than a constant value. We added the parameter 'length' to the function call and then defined 'tslength' as `tslength = random.randint(128,length)`. When we create the variable length TS, we define length to be 256, this allows the time series to be up to double the original length of the baseline time series. We did this to ensure there was an element of

randomness to the lengths of the TS, doing this also helps to emulate the randomness found in real-world data.

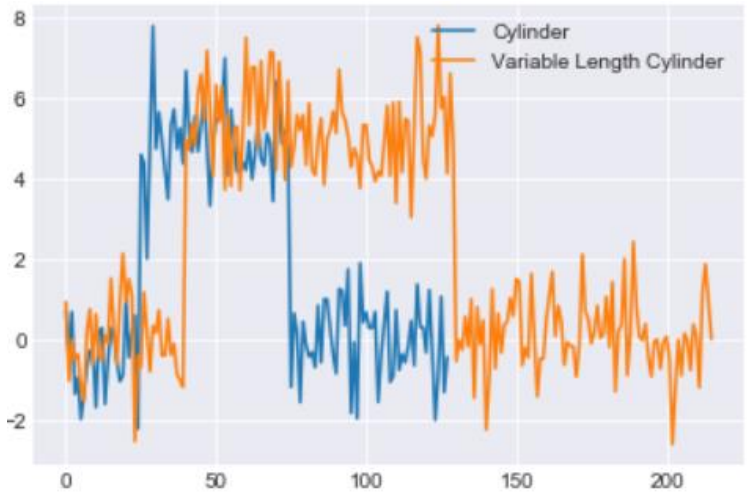

Figure 8 – Baseline Time series(blue) alongside a time series with a length of 256, double the fixed baseline length.

We created our variable length time series by writing and appending to a file. To create our dataset for variable length we cannot use a data frame to store our values as it would automatically try to create fixed length time series by adding 0 values to every time series that is shorter than the longest time series in the data frame. We instead write the time series to a file, allowing us to maintain variable length between the time series.

To create our test and train files, we use for loops to add the time series data to the file. To populate the training file, we use 2 for loops. The first for loop adds the baseline fixed length TS and the second for loop adds the variable length TS to the training file. To create our test file, we use that same method.

Before each time series, we must define the class, so we also write the class to the beginning of each time series we add to the dataset. In Algorithm 5 we see how the for loop writes each time series, using Bell (class 1) as an example, to the dataset as it iterates.

Algorithm 5.

```
out_string = ""
out_string = "1," + str(cbf(1,256))
out_file.write(out_string)
```

Using this method to populate our files encases each TS in our file in square brackets. For the final step, we remove these brackets and are left with classifiable TS of variable length.

When applying variable length noise to the data, we had to consider if we wanted to scale the signal shape with the length of the TS. By examining Snippet 1 we can see that the values for ‘a’ and ’b’ are determined by the length of the TS (tslength). These values ‘a’ and ‘b’ determine the starting and ending points for a signal shape on a TS. We opted to alter that definition slightly so as not to scale the signal shapes with the length of the TS. Instead, we kept the values for ‘a’ and ‘b’ operating under baseline assumptions (tslength = 128). We did this because we were concerned that we may inadvertently be offsetting the signal shape and so, inducing more noise of a different variety. As we were only trying to examine variable length noise, we felt that this may bias our results. A real-world example of where this might occur would be, for example, two sensors recording the same speech

pattern, but where one of the sensors gathers input for a longer period after the pattern has been given. This creates two time series of the same pattern of considerably different lengths.

3.1.4 Amplitude Scaling Dataset

We created a dataset that scales the signal shape data points to varying degrees. By scaling the signal shape in each of the three classes to different degrees it creates a challenge for the classifier. Because although the signal shapes are still in the time series, they may be much larger or smaller than the signal shapes in the initial unaltered dataset. In the algorithms 1, 2 and 3 above, we see *that* `(6+n)`, is the baseline value given to the time series of each class. We can alter the value of n, which is defined by `n = random.normalvariate(0.0,1.0)`, to increase the amount of noise which is applied to the dataset. To induce the amplitude scaling noise we set n to be `n = random.normalvariate(0.0,3.0)`. This increases the noise in the dataset that is applied to the signal shape as it increases the possible range of noise that can be applied. By applying the amplitude scaling noise in this way, we are able to emulate the same form of noise that would be created, in the real world, due to sensory overload. In Figure 9 we see an example of how this noise affects the signal shape.

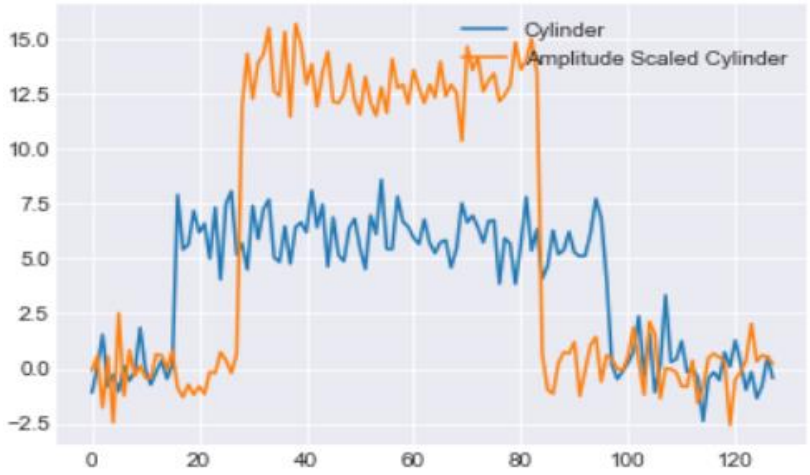

Figure 9 – Baseline time series (blue) alongside noisy scaled time series (orange).

Originally, we sought to alter the induced noise by altering the numerical value 6 i.e. rather than 6+n use 9+n. Ultimately, we decided against this approach opting instead to alter the value for n. We altered the n value because we could induce a greater element of randomness to the noise that we applied. We also had some concerns about that possibility of creating sub-classes in our dataset by introducing a more limited range of noise.

To create the final test and train datasets, there were several steps involved. Firstly, we initialise 6 datasets, 2 for each of the classes Bell, Cylinder and Funnel. The two datasets for each class represent the partitioning of the training and testing data. The 3 training datasets each contain 100 TS. We used 2 sets of for loops to populate each dataset with the first loop inputting standard baseline TS, and the second adding our noisy TS to the dataset. The 3 testing datasets each contain 300 TS. Again, we used 2 for loops to populate these with the first for loop populating the datasets with baseline TS and the second for loop adding the noisy TS to each dataset. We then added a column to the beginning of each of these 6 datasets to denote the specific class that the time series belongs to. We now have TS that can be classified by the classification methods.

To create the final 2 test and train datasets we merge the 3 training and testing datasets respectively. Finally, we shuffle the TS so that the classes and the noisy TS appear in a random order in both datasets. We now have a training dataset of 300 TS and a testing dataset of 900 TS, both of which contain some TS with amplitude scaling noise and represent the 3 classes equally.

3.1.5 Signal Shape Time Warping Dataset

We created the Signal shape time warping dataset by increasing and decreasing the range of data points that the offset term effects on a time series. In Snippet 1 it's the values for ‘a' and ‘b' that affect the length of the signal shape, so by altering these values, we can induce signal shape time warping noise on the time series.

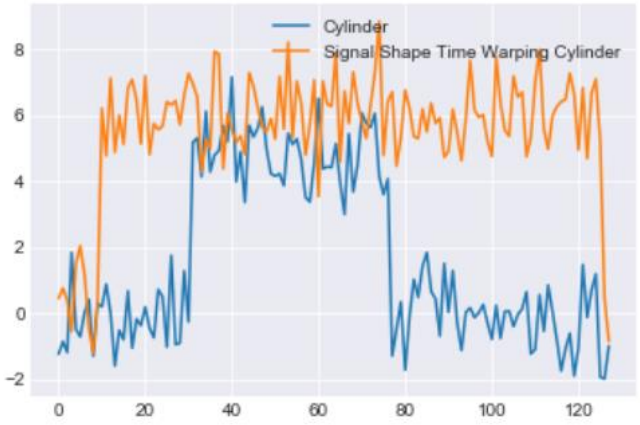

Figure 10- Baseline time series (blue) alongside noisy signal shape time warped time series (orange).

In figure 10 above, we see the effect that lengthening the range of ‘a' and ‘b' has on the signal shape.

The signal shape time warping noise levels we choose to apply to the time series are the same for each of the three classes, as the possible ranges are limited, they are the following:

A) ‘a’ in the range [10, 20] and ‘b’ in the range [5, 25]

B) ‘a’ in the range [1, 33] and ‘b’ in the range [65, 95]

Selecting these parameters allows us to, in the case of the A parameter, significantly shorten the period in which the signal shape of the time series is pronounced on the TS. In the case of the B parameter significantly increase the length of time the signal shape is pronounced on the TS. We created 6 distinct datasets, 2 for each class, these two datasets again represent the training and testing partition. Each dataset was created using an altered version of the code in Snippet 1 by adding additional parameters to the function call, see Snippet 2. These additional parameters define the possible ranges of ‘a’ and ‘b’, based on the input we give during the function call. For example, to generate a TS with a cylinder signal shape and baseline noise we would use *cbf(1,16,32,32,96)*

```
def Keoghcbf(o, ai,aii,bi,bii, tslength=128):
base = tslength / 8

  a = random.randint(ai,aii)
  b = aii + random.randint(bi,bii)
  n = random.normalvariate(0.0,1.0)
```

Snippet 2: Altered CBF code for Signal Shape Time Warping dataset generation

When generating the noisy data, we created half the TS using the parameters from A above and the other half using the parameters from B. This was done to maintain a good variation of in the signal shape time warping noise that was applied to the TS.

Using 2 for loops for each of our datasets we generated the training dataset, with one for loop populating the dataset with noisy TS and the other adding TS with baseline noise. Then we added a column to the beginning of each of the datasets which contained class labels to denote the class that the time series belongs to. We then merged our three training datasets (one for each class) to form our final training dataset. We do the same for our three testing datasets and are left with our final testing dataset. Finally, we shuffled the order in which the time series appear in the datasets to ensure that the classes are distributed randomly across the datasets.

3.2 Classification and Initial Results

To generate the classification accuracies, we need to train each classifier using the training data and then test the classifier using the test data. We use our generated datasets to do this as we have created a training and testing dataset to represent each type of noise, for each percentile of noise applied to the baseline data. As we have 13 datasets (counting test/train as one for each type and percentage of noise), we ran each of the classifiers 13 times, once for each dataset. Once the TS in the given dataset have been classified (once the classification method has finished), the output of the method is the classification accuracy for the given data. Then we store that accuracy score and rerun the classification method using the next test/train dataset as input to be classified. So, each time we run the classifier we re-train and re-test it for the new TS data.

The accuracy score from each of the classification methods can be seen in the tables below. Each table represents the accuracy of each classification method when applied to each of our datasets. Each table shows the classification accuracy for each of the classifiers for a single noise type.

	DTW	FCN	BOSS	Weasel	Mr SEQL
Baseline	0.996	0.998	0.998	0.997	1.000
Gaussian Noise at 20%	0.982	0.953	0.984	0.984	0.979
Gaussian Noise at 40%	0.946	0.93	0.964	0.976	0.966
Gaussian Noise at 60%	0.918	0.893	0.964	0.97	0.952

Table 2 – The accuracy rating of each classifier when applied to the baseline dataset and the generated dataset containing time series with Gaussian noise.

	DTW	FCN	BOSS	Weasel	Mr SEQL
Baseline	0.996	0.998	0.998	0.997	1.000
Variable length Noise at 20%	---------	---------	0.998	0.999	1.000
Variable length Noise at 40%	---------	---------	1.000	0.999	1.000
Variable length Noise at 60%	---------	---------	0.998	1.000	0.998

Table 3 – The accuracy rating of each classifier when applied to the baseline dataset and the generated dataset containing time series with variable length noise.

	DTW	FCN	BOSS	Weasel	Mr SEQL
Baseline	0.996	0.998	0.998	0.997	1.000
Signal Shape Time Warping Noise at 20%	0.981	0.995	0.967	0.982	0.989
Signal Shape Time Warping Noise at 40%	0.977	0.992	0.960	0.979	0.973
Signal Shape Time Warping Noise at 60%	0.974	0.981	0.956	0.978	0.966

Table 4 – The accuracy rating of each classifier when applied to the baseline dataset and the generated dataset containing time series with signal shape time warping noise.

	DTW	FCN	BOSS	Weasel	Mr SEQL
Baseline	0.996	0.998	0.998	0.997	1.000
Amplitude Scaling Noise at 20%	0.973	0.979	0.993	0.990	0.987
Amplitude Scaling Noise at 40%	0.953	0.967	0.982	0.978	0.978
Amplitude Scaling Noise at 60%	0.943	0.949	0.977	0.976	0.971

Table 5 – The accuracy rating of each classifier when applied to the baseline dataset and the generated dataset containing time series with amplitude scaling noise.

3.3 Visualising and Analysing the Results

When we examine the classification accuracies on TS with Gaussian noise in Figure 11 below, we can see that DTW and FCN take the most significant loss of accuracy. FCN is the worst effected classifier with the accuracy score dropping by 10.6% from the baseline noise to noise at 60%, followed by DTW seeing a loss of accuracy of 7.7% under the same conditions. WEASEL shows the highest invariance to Gaussian noise scoring the highest accuracy of all classifiers on each level of noise. WEASEL suffers a loss of accuracy of just 2.7% from its baseline accuracy rating. Mr-SEQL and BOSS are also highly competitive, with their respective accuracies dropping by 4.8% and 3.4% from baseline noise to noise at 60%. These results appear to show that it is the symbolic approach that is best at correctly classifying TS data to which Gaussian noise has been applied. WEASEL is the most resilient of the

symbolic methods to Gaussian noise, as when noise applied to 40% or 60% of the TS data, WEASEL outperforms all other methods. It's only the when noise is applied to 20% of the data do, we see the BOSS classifier achieving the same accuracy score of 98.4%.

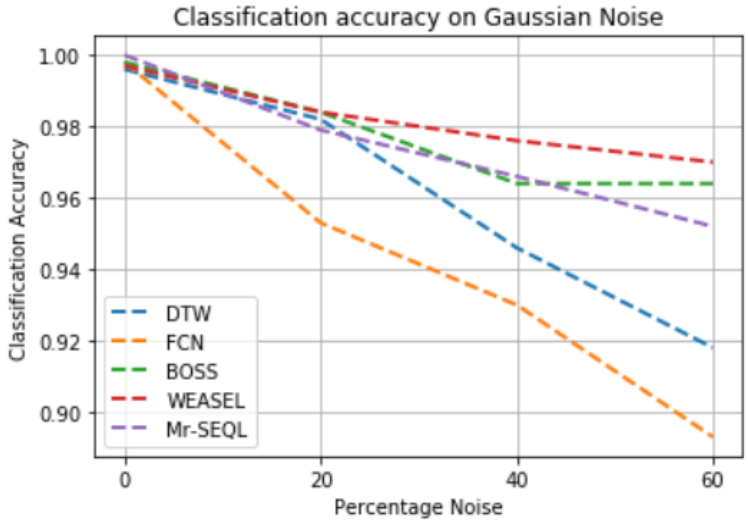

Figure 11 – Graph showing the accuracy of each classifier at each percent of Gaussian noise applied to the TS.

Variable length noise appears to be the easiest noise to handle for our symbolic classification methods. However, neither our distance measure (DTW) nor our deep learning method (FCN) can classify variable length TS. Figure 12 below shows the accuracies of BOSS, WEASEL and Mr-SEQL, it's important to note however, that the x-axis begins at 0.997 so the largest difference in classification accuracy is 0.3%. WEASEL shows the highest invariance to variable length noise as the accuracy actually increases by 0.3% from baseline noise to noise at 60%. The BOSS method maintains the same accuracy score at each noise percentile, bar noise at 40%, where we see an increase of 0.2% in its accuracy score. Mr-SEQL is the only classifier whose accuracy score drops below the baseline score. This occurs when noise is at 60%, as we see a drop of 0.2%, up to that point however there was no difference in classification accuracy. The fact that the results vary so little on variable length noise suggest a comparatively easy classification test. This is likely due to our choice to not scale the signal shape with the length of the time series, and so our application of variable length poses less of a challenge to the classification methods.

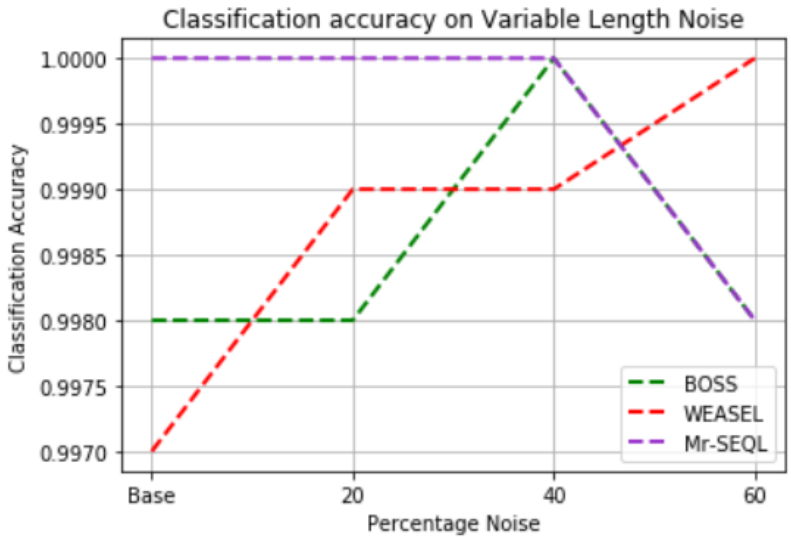

Figure 12 - Graph showing the accuracy of each classifier for each percentage of variable length noise applied to the TS.

When we examine the effect of signal shape time warping noise, we drastic differences in the classification accuracies of our methods. In Figure 13 we see that it's the BOSS method that takes the most significant drop in accuracy with a 4.2% drop in total. FCN, the deep learning method shows the highest invariance to signal shape time warping noise as its accuracy only degrades by 1.8% when noise is at applied to 60% of the TS. FCN further shows its invariance to this form of noise when we examine the accuracy scores of the classifiers when noise is at 40%, as it suffers only a 0.7% loss of accuracy, outscoring the WEASEL classification method by 1.3%. DTW and WEASEL share similar accuracy score across each noise levels with the largest difference being just 0.4% when noise is at 60%. It is only for signal shape time warping that we DTW preforming competitively when compared to the symbolic methods of classification. Mr-SEQL shows itself to be able to handle signal shape time warping noise quite effectively when the noise levels are at 20% as its accuracy score is just 0.6% lower than that of FCN, which is our highest performing classifier in this case. However, as the levels of noise increase, we can see that Mr-SEQL struggles to correctly classify correctly the TS, as when noise is at 60% Mr-SEQL accuracy score drops by 4.4%.

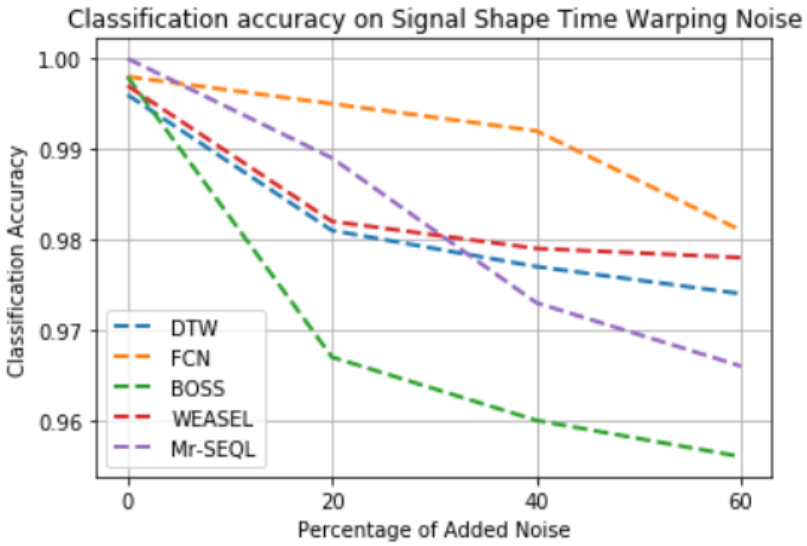

Figure 13 - Graph showing the accuracy of each classifier for each percentage of signal shape time warping noise applied to the TS.

When we examine the effect of Amplitude scaling noise on the classification methods, we see that the symbolic methods of classification show the greatest invariance to amplitude scaling noise. Figure 14 shows how each classifier is affected by amplitude scaling noise at each percentage. We see that it's our distance measure method, DTW that struggles the most with this type of noise, taking a 5.2% loss of accuracy between baseline noise and noise at 60%. The accuracy of the FCN classification method follows a similar trend to the DTW classification method as in both cases when the levels of noise increase, they consistently degrade in terms of accuracy. FCN does, however, maintain a slightly better accuracy score across all levels of amplitude scaling noise, yet FCN is outperformed by all three of the symbolic methods. The symbolic methods all perform very similarly, BOSS and WEASELs accuracy scores both degrade by 2.1% from baseline to 60% noise. There is never more than a difference of 0.5% between the 3 symbolic methods at any given level of noise. This shows a high level of competitiveness however, it is the BOSS method that is the best performing method on TS affected

by amplitude scaling noise. It maintains a slightly higher accuracy score across each level of noise, however the difference in classification accuracy at 60% is 0.1% which is minute.

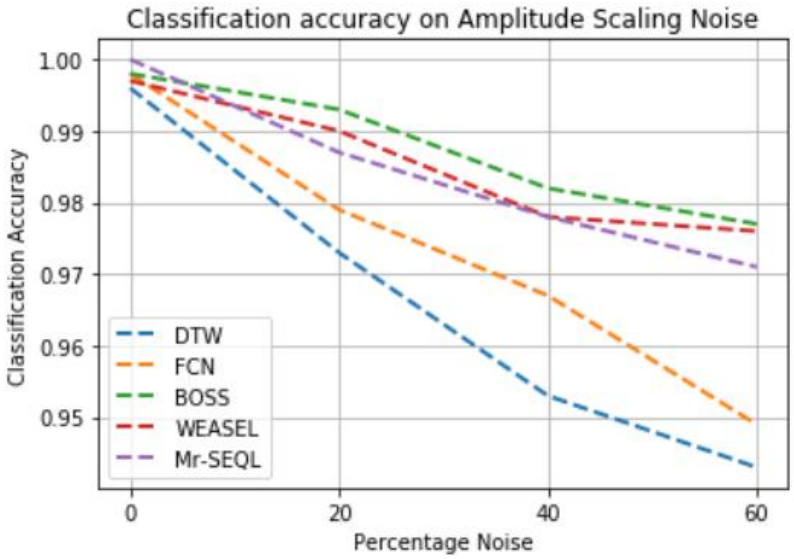

Figure 14 - Graph showing the accuracy of each classifier for each percentage of amplitude scaling noise applied to the TS.

The DTW classification method struggles the most when trying to classify noisy TS. DTW cannot classify variable length noise and has no outstanding noise type that it performs exceptionally well in. We can see this clearly in figure 14 below, which compares the average classification accuracy of all the classifiers across each type of noise we injected into our baseline data. It is only for signal shape time warping that we see the DTW method perform competitively compared to the WEASEL method and the Mr-SEQL method, yet all these methods are outperformed by the deep learning method accuracy on TS with signal shape time warping noise applied.

The FCN method of classification, a deep learning method, is the best performing classifier on time series with signal shape time warping applied. From Figure 14, we can see that FCNs deep learning approach fails to accurately classify TS with Gaussian noise applied, performing significantly worse than each of the symbolic methods, as well as our distance measure method DTW.

The symbolic approach has the best invariance to Gaussian noise. WEASEL is the top preforming symbolic method for Gaussian noisy data, and in fact across all noise types, with the exception of amplitude scaling noise. Although WEASEL is the successor to the BOSS approach, we can see that when classifying noisy time series, the different approaches employed by these classification methods has an impact on their ability to classify different types of noise.

From Figure 14 we can see that our symbolic methods have their own respective strengths and weakness for different types of noise. The BOSS classification method is shown to be the highest performing symbolic method on time series data affected by Amplitude scaling noise, as it achieves the highest average accuracy. This is the only case where BOSS outperforms the WEASEL classification method. However, the approach used by the BOSS method has its drawbacks, as the BOSS method is shown to be susceptible to misclassification when signal shape time warping is applied to the TS data.

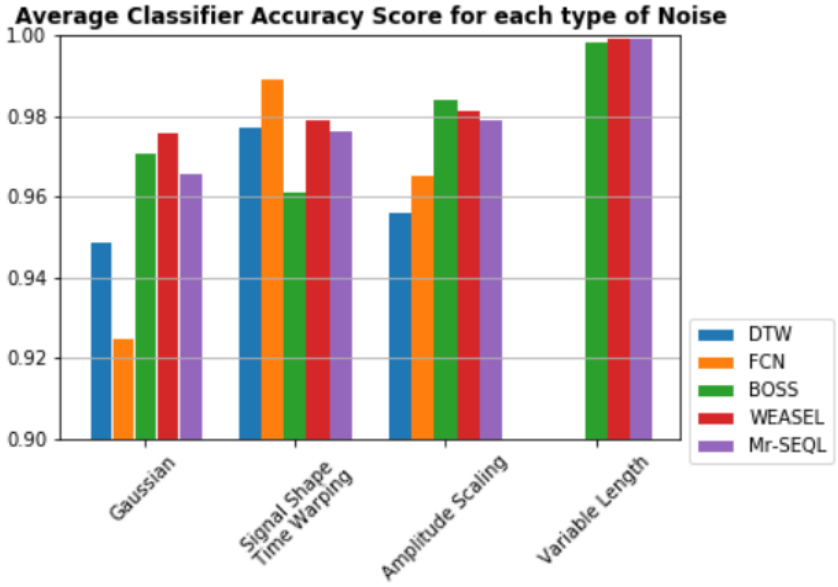

Figure 14- Comparison of the average accuracy for each of classifiers on each type of noise

The above analysis of the results is based on the accuracy scores of the classifiers from a single set of datasets. While this was, in part, due to time limitations it remains a major oversight of our project. When we generate the datasets the levels of noise vary slightly. This is due to the element of randomness we used when injecting the time series with noise, which we did to avoid generating subclasses by implicitly declaring the levels of noise that are injected into the TS. This means that the exact accuracy score of the classifiers may fluctuate on subsequent iterations of our datasets.

To avoid this, we should have generated our datasets and tested the classifiers on them multiple times, then averaged those results to come to our final accuracy ratings for the classifiers. This would account for the random element of the noise applied and the possible differences in classification accuracy.

While the accuracy scores should be not considered to be an exact indication of the classifier's performance, we can however, expect the overall trends of the classifiers accuracy scores to remain consistent. As we tested each of the classifiers using the same 13 datasets, the comparable results between the classifiers show how each method deals with noise under the same conditions. Therefore, on subsequent iterations of the datasets, while the exact accuracy levels may vary, we could expect to see a comparable loss or gain in accuracy across each of the classifiers.

3.4 Future Work

During the process of answering the questions proposed in this report, it became apparent that there is a large amount of future work that can be done with regards to this project. Some of the major aspects of work that we will do in the future are the following.

Working with real-world data.

The data used during this project was generated to simulate real-world data. As we are examining how these classifiers could be expected to perform in the real world it would be interesting to use

real-world data. The results from an experiment such as these could then be used to refine our own simulated real-world data in further studies.

Include more types of noise.

In particular, we like to examine how introducing TS with missing data points might affect the accuracy of the classifiers. This is another type of noise that is common in real-world data and it would be interesting to see how classifiers handle it.

3.5 Conclusion

In our work we examined how the accuracy of five TSC methods would be affected by the introduction of different types of noise to time series would affect their ability to accurately classify those time series. The individual classification methods we examined were, a baseline method, DTW, three symbolic methods, BOSS, WEASEL and Mr-SEQL, and finally, a deep learning method FCN. We introduced four noise types that are frequently found in real-world time series, namely, Gaussian noise, amplitude scaling, signal shape time warping and variable length noise.

Our research suggests that the fundamental approach of a classifier has a substantial impact on how those classifiers will perform on certain types of noisy data. Symbolic methods appear to outperform deep learning methods for both TS with Amplitude scaling noise and TS with Gaussian noise applied. However, those same symbolic methods are outperformed by the deep learning method when signal shape time warping is applied to the data. Neither the deep learning method nor the distance measurement method can classify variable length data, yet each of the symbolic methods appear highly competitive on variable length TS.

When choosing a classifier, consideration should be given as to what types of noise may be most likely to appear on the TS, as there are significant differences in accuracy depending on the type of noise that is present.

3.6 Acknowledgements

I would like to express my very great appreciation to my supervisor, Dr Georgiana Ifrim and my moderator, Thach le Nguyen for their valuable and constructive suggestions, as well as their continued support throughout the entirety of this research project.

4 Bibliography

Ada Wai-chee Fu, E. K. (2006). Scaling and Time Warping in Time Series Querying.

Arash Jalalian, S. K. (2012). GDTW-P-SVMs: Variable-length time series analysis using support vector machines. The University of Newcastle, School of Electrical Engineering and Computer Science.

Argyro Kampouraki, G. M. (2009). Heartbeat Time Series Classification With Support Vector Machines. IEEE.

Bagnall. (2017). Bagnall A, The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances.

Bronshtein, A. (2017). A Quick Introduction to K-Nearest Neighbors Algorithm. Retrieved from https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7

Batista, G., Wang, X., Keogh, E.J.: (2011) A Complexity-Invariant Distance Measure for Time Series. In: SDM. SIAM / Omnipress.

Cuff, P. (2011). Signals and Systems. Retrieved from https://www.princeton.edu/~cuff/ele301/files/lecture1_2.pdf

D. J. Berndt and J. Clifford. (1994) Using dynamic time warping to find patterns in time series. In KDD Workshop.

Esling, P. (2012). Time-Series Data Mining. ACM Computing Surveys.

Fazle K et al. (2017). Fully Convolutional Networks for Time, LSTM.

Hassan I F et al. (2018). Deep learning for time series classification: a review.

Högqvist., P. S. (2012). SFA: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In Proceedings of the 2012 International Conference on Extending Database Technology, (pp. 516-527).

Hu, B. (2013). Time Series Classification Under more Realistic Assumptions.

J. Lin, E. J. (2007). Experiencing SAX: a novel symbolic representation of time series. In Data Mining and knowledge discovery (pp. 107–144).

Kang, Y., Belušić, D. and Smith-Miles, K. (2014). Detecting and Classifying Events in Noisy Time Series. Journal of the Atmospheric Sciences, 71(3), pp.1090-1104.

Mehdiyev N et al. (2017). Time series classification using deep learning for process planning: A case from the process industry.

Nweke et al. (2018). Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges. Expert Systems with Applications.

Patrick Schäfer et al. (2017). Fast and Accurate Time Series Classification with WEASEL.

Saito, N. (1994). Local Feature Extraction and Its Applications Using a Library of Bases. Department of Mathematics, Yale University.

Schäfer, P. (2015). The BOSS is concerned with time series classification in the presence of noise.

Sakoe,H. and Chiba, S. (1978) Dynamic programming algorithm optimization for spoken word recognition.

Xiaopeng Xi, E. K. (2006). Fast Time Series Classification Using Numerosity Reduction.

Y. Chen, E. K. (2015). The UCR time series classification archive. Retrieved from www.cs.ucr.edu/~eamonn/time series data/