UiT The Arctic University of Norway

Faculty of Science and Technology

Department of Physics and Technology

Imputation and classification of time series with missing data using machine learning

Comparing kNN, SVM, and TCN at classifying time series after imputing missing data.

Vilde Fonn Dretvik

FYS-3941 Master's thesis in applied physics and mathematics…June 2021

Contents

Foreword

After having struggled and worked on my master thesis during brain fog, long-time fatigue, and chronic depression I'd like to thank the following:

My family. My friends. The Dungeons and Dragons (5e) group I was in. The student club Imladris, with their slogan: "when normality takes too much effort". My general physician. My psychiatrist. My supervisor.

Thank you

Chapter 1

Introduction

Time series classification is a general problem with a wide group of potential applications. Some of these problems are extra challenging to classify as they contain multivariate time series, time series with missing values, and sometimes both. This work studies a specific example of such multivariate time series with missing data by using a clinical data set containing blood sample values from patients that have undergone surgery, from which risk of infection in the surgical wound follows. Selected methods for ways to handle the missing data and classification of this data is explored in this work. The methods are additionally tested on synthetic data to test their robustness.

1.1 Motivation

After an operation, a patient can get a surgical site infection (SSI). Hospitals do blood tests before and after the operations to try to detect these infections. The same blood tests are not taken every day as some patients are more healthy and require fewer tests. As the tests are multivariate time series, it is difficult to perform good statistical learning and accurate prediction when there are gaps in the time series. A method to try to remedy those gaps is to use imputation. A selection of imputation methods is tested in this paper on data of infected and healthy patients who have undergone gastrointestinal surgery. This is done using two popular methods for time series classification: the k Nearest Neighbors (kNN) classifier and the Support Vector Machine (SVM) classifier in combination with Dynamic Time Warping (DTW). A relatively new deep learning algorithm called the Temporal Convolutional Network (TCN) is compared against the two older algorithms.

1.2 Research Questions

The research question in this project are:

1. Which imputation method is best suited as pre-processing of multivariate time series with missing data before classification?

2. How does the Temporal Convolutional Network (TCN) classifier compare with methods such as K Nearest Neighbour (KNN) and Support Vector Machine (SVM) utilising Dynamic Time Warping (DTW) when classifying multivariate time series data with missing data?

1.3 Outline

The theory is presented in chapter 2, starting with what time series classification is, and known methods to do it, followed by a summary of distance based methods, definitions of various types of missing data, an overview of the kNN, SVM, TCN classifiers for time series, and score metrics for classification, in that order. After going through the theory, the methodology is presented in chapter 3 explaining how the processing the data was done and how the classifiers was implemented. Experiment parameters is found in chapter 4, and results with discussion are presented in chapter 5, then finally conclusions are drawn in chapter 6.

Chapter 2

Theory

This chapter presents theory for time series classification. It covers distance-based methods using the multi-dimensional Euclidean distance and the more advanced dynamic time warping distance. The latter is developed specifically for time series, allowing efficient and meaningful comparison of time series with relative temporal shifts. It also introduces two well-known classifiers that are used in this work: the k nearest neighbor (kNN) classifier and the support vector machine (SVM) classifier. A newer deep learning method used for time series classification is then introduced afterward, called temporal convolution network (TCN). Finally, some common performance measures for classifiers are presented, namely the confusion matrix, accuracy, precision, recall, specificity, balanced accuracy, and the F_1 score.

2.1 Time Series Classification

2.1.1 What is a time series

A time series is defined as a series of numerical values indexed in time order. This can for instance be measurements of stochastic variables taken over time. Different from a timeline, which can be either continuous or discrete, the time series can be seen as a sequence of discrete measurements taken at discrete points in time. The time series can be a one-dimensional or multidimensional stochastic process [1, 2, 3]. The independent variable is the time t when the measurements were taken, which can be defined as the sequence $\mathbf{t} = \{t_0, t_1, t_2, ...\}$ of discrete measurements. These time stamps are often sampled at regular intervals, and can have intervals ranging from fractions of a second to years. In this paper, the interval is 1 day. In a one-dimensional time series, the dependent variable is a vector of scalar values X sampled at time t, which can be defined as $\mathbf{X} = \{X_{t_j}\}\forall t$, where j is the index of discrete time t; i.e. each measurement has an associated time stamp, and can be indexed as $X_0, X_1, X_2, ...$, etc. If one takes several measurements at the same time, the time series will be multivariate or multidimensional, as seen in the figure below. In the SSI dataset used this work, each patient has its own multidimensional set of time-series data, and similarly for the uWave dataset for each trial experiment.

A time series is generally seen as correlated, as the previous measured value X_{t-1} will depend in some way on the previous value X_{t-2}, X_{t-3}, and so on ([1], p. 9).

2.1.2 What is classification

Classification is the task of grouping data into known classes. It is the supervised counterpart of clustering, where the groups or clusters are assumed unknown. Supervised

Figure 2.1: Representation of a general multi-dimensional time series measurements as a matrix. Row i indicates a specific measurement or test, manifested as a scalar value, and the column j is an instant of the measurements along the time axis t, with one value of j for each time index.

$$\mathbf{X} = \begin{Bmatrix} X_{0,0} & \dots & X_{0,j} \\ \vdots & \ddots & \vdots \\ X_{i,0} & \dots & X_{i,j} \end{Bmatrix}$$

learning has access to labeled data, while unsupervised learning finds hidden patterns, and may later assign labels to them based on further analysis and interpretation. The learning is done by finding parameters or parameter functions (kernels) that best describe a decision boundary which separate data points. These boundaries are either a point if the data are one-dimensional, a line if they are two-dimensional, or a surface if they are three-dimensional. If the data have more dimensions, they are separated using a high-dimensional manifold. Once the data points are separated, a predicted label is assigned to them. Depending on the method, in a two-class data set the given labels are usually $\{-1, 1\}$ or $\{0, 1\}$. Certain algorithms prefer the use of certain labels due to their underlying math. The data sets used in this paper is labeled using $\{0, 1\}$ where for the SSI data, a patient with SSI is indicated with the label 1, and a non-SSI patient is labeled with a 0. The uwave data used was relabeled, where pattern 1 is labeled as 0 and pattern 2 is labeled as 1.

2.1.3 Methods for time-series classification

There are several classifiers that can be applied to time series data. They can largely be grouped as time domain distance-based, differential distance-based, dictionary-based, shapelet-based, interval-based, and ensemble-based classifiers, according to [4]. The time domain distance-based classifiers include those based on the weighted dynamic time warping (WDTW)[5] distance and the Time Warp Edit (TWE)[6] distance, and also the Move-Split-Merge (MSM)[7] classifier. Time domain distance classifiers utilize elastic distance measures that are allowed to stretch or warp the time axes when comparing data points. The important dynamic time warping distance (DTW) will be presented in Section 2.2.2. The differential distance-based classifiers include those based on the Complexity Invariant Distance (CID)[8], the Derivative DTW (DDTW)[9] distance, and the Derivative Transform Distance (DTD_C)[10]. Differential distance-based classifiers utilize the first-order difference between a series, and the most successful versions of them combine both time and difference distances. Dictionary-based classifiers include the Bag of Patterns (BOP)[11], the Symbolic Aggregate Approximation - Vector Space Model (SAXVSM)[12], the Bag of SFA Symbols (BOSS)[13], and DTW Features (DTW_F)[14]. Dictionary-based classifiers represent a series as a word by reducing the dimensionality through a transformation, then the distribution of words is discretised into values by sliding windows over the distributions. Each value is assigned a symbol from a dictionary of size s. This also reduce the dimensionality of the series. There are different shapelet-based classifiers based on Fast Shapelets (FS)[15], the Shapelet Transform (ST)[16], and Learned Shapelets (LS)[17]. A shapelet-based classifier tries to find and use sub-sequences of time series data or candidates of such snippets and slide them along the time series to find the shortest distance between them and the data. Among the interval-based classifiers we find the Time Series Forest (TSF)[18], the Time Series Bag of Features (TSBF)[19], and the Learned

Pattern Similarity (LPS)[20] method. These classifiers find features from intervals of the time series, then use them to train a Support Vector Machine (SVM)[21]. Three types of ensemble classifiers are: Elastic Ensemble (EE), Collective of Transformation Ensembles (Flat-COTE)[22]. They are ensembles or combinations of several simple classifiers, which vote on the result of the classification. This is not an exhaustive list. There are also some types of classifiers not listed above.

Some classifiers are regression-based and use parametric models such as the auto-regressive (AR) model, the auto-regressive moving average (ARMA) model, or the auto-regressive integrating moving average (ARIMA) model. Others may use a Hidden Markov Model (HMM).

Deep learning techniques can also be used for time series classification [23, 24], but have yet to gain popularity for this use according to [25]. Some deep learning method architectures that have been used for time series classification are mentioned in [26] and [25]. These architectures include the: fully connected (FC) neural networks (a.k.a. multi-layer perceptrons)[27, 28], Multi-scale Convolutional Neural Network (MCNN)([29]), Recurrent Neural Network (RNN)[30], Time Le-Net (t-LeNet)[31], Multi Channel Deep Convolutional Neural Network (MCD CNN)([32],), Time Convolutional Neural Network (Time-CNN)[29], Time Warping Invariant Echo State Network (TWIESN)[33], Multivariate Long Short-Term Memory Fully Convolutional Network (MLCN), Time Series Attentional Prototype Network (TapNet)[34], and Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE)[35].

According to [36], time series classification can be grouped into two main categories: feature-based and distance-based. Distance-based classification is by the authors further divided into reduction-based, purely distance-based, and parametric distance-based. Tables 4 through 7 in [36] sums up the properties and differences well between several methods. Among them are the methods DTW, SVM, and kNN.

This project uses the kNN classifier and the SVM classifier with the DTW distance as the chosen distance measure. According to the taxonomies mentioned above, these methods can be grouped as time domain-distance based or . The SVM and kNN methods can further be categorised as feature based mixed with the purely distance-based DTW.

The motivation for use of the kNN classifier in this work is that it is easy to use and known to give good results for time series classification [37, 38]. The SVM method is known to have comparable results for some data sets and kernels. The DTW distance and variations of it is known to function well as a distance function for time series and can be used both in the kNN method and as input to the kernel function of the SVM.

2.1.4 Modification of classification methods for use with time series

Time series are subject to the curse of dimensionality [39, 40]. This is because of their high-dimensional nature. When you increase the dimensionality, all datapoints become far apart in feature space and therefore Euclidean distance becomes less of use, as the distance between all data points will be very high. If one has n number of points in $\mathbb{R}^d$, then the expected difference between the minimum and maximum distance between a reference point and random points approaches zero when compared to the minimum distance, as the number of dimensions d goes toward infinity:

$$\lim_{d \longrightarrow \infty} E\left(\frac{dist_{max}(d) - dist_{min}(d)}{dist_{min}(d)}\right) \longrightarrow 0 \tag{2.1}$$

Another issue that complicates classification of time series is that visual analysis tools are typically made for 2-3 dimensions, and if there are more dimensions, then it is difficult to intuitively understand the result. Higher-dimensional data can usually be reduced in dimensionality with methods such as kernel principal component analysis (kernel PCA) [41], Fisher discriminant analysis (FDA) [42], or T-distributed stochastic neighbor embedding (t-SNE) [43], but with time series the reduction of dimensions or factors using said methods can result in critical loss of information, correlation, and causation. A solution to compare time series (of different length) is to use the dynamic time warping distance, which is explained further down, and which works with both k Nearest Neighbor and support vector machines.

2.2 Distance-based methods

There are several methods for comparing time series and other types of data using different types of distances. Two of the most used distance or dissimilarity measure is the Euclidean distance, and the Dynamic Time Warp (DTW) distance .

2.2.1 Euclidean distance

The multi-dimensional Euclidean distance is given as

$$D_{ij} = \sqrt{(\mathbf{x_i} - \mathbf{x_j})(\mathbf{x_i} - \mathbf{x_j})^T} \tag{2.2}$$

where D is an NxN matrix with column width and row height equal to the number of dimension, and x_i, x_j are row vectors to compare against each other.

However, the Euclidean distance measure is known to be sensitive to offset, amplitude scaling, noise, phase shifts, and temporal distortion. Standardizing the data-set can help against offset and amplitude scaling, but not the remaining issues [44]. A different method for handling the other problems is needed. A known method is using dynamic time warping.

2.2.2 Dynamic time warping distance

The beneficial properties of the dynamic time warping distance can be explained as follows: "Dynamic time warping (DTW) allows local contraction and expansion of the time axis, alleviating the alignment problem inherent with Euclidean distance. It allows for comparison of signals with different shapes. In the past, DTW was widely used in speech recognition and more recently in various time series data mining applications. It is a reasonable choice if prior knowledge about the data at hand is limited" [44].

Patients may have similar SSI data, but infection-related events occur at different times compared to each other. Finding similar "shapes" that are stretched or compressed can therefore be difficult on a timeline. Certain restrictions and rules make it possible to compare these shapes [45]:

1. All indexes on the timeline must be matched with one or more indexes on another timeline.

2. The first index must be matched with the corresponding first index on the other timeline. Accordingly, the same goes for the last index.

3. The mapping of the indices from the first sequence to indices from the other sequence must be monotonically increasing, and vice versa.

Following these restraints and rules gives the smallest and most correct dissimilarity when comparing the two timelines. The shortest distance between index samples within a window is used. The pseudo-code for computation of the DTW distance between a time-series s of length n and a time series t with length m, and window w so the comparisons keep within a certain index distance of each other, is given below. [45, 46, 47, 48].

```
int DTWDistance(s: array [1..n], t: array [1..m], w: int) {
    DTW := array [0..n, 0..m]

    w := max(w, abs(n-m)) // adapt window size (*)

    for i := 0 to n
        for j:= 0 to m
            DTW[i, j] := infinity
    DTW[0, 0] := 0
    for i := 1 to n
        for j := max(1, i-w) to min(m, i+w)
            DTW[i, j] := 0

    for i := 1 to n
        for j := max(1, i-w) to min(m, i+w)
            cost := d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],    // insertion
                                        DTW[i  , j-1],    // deletion
                                        DTW[i-1, j-1])    // match

    return DTW[n, m]
}
```

The indices of the time series is then warped with the optimized path. The warping is non-linear.

The data is multidimensional, so the distance calculation function used in the cost calculation 'cost' for the cumulative distance matrix DTW is the multidimensional Euclidean distance, mentioned above.

There are many variations on the DTW algorithm that attempt to improve on it but there are too many to list here. The code used in the library to compute the DTW distances is very similar to the above pseudo-code, but also outputs the optimised path, specifying the indices that are aligned after time warping, which is not of interest at this time.

2.3 Missing Data

The data being used have entries with values missing. When doing clustering and computing distances, that is rather undesirable. Trying to do calculations on values that does not exist, is not possible. There are several reasons why data values may be missing, and ways to work around that.

2.3.1 Types of missingness

Data may be Missing At Random (MAR), Missing Completely At Random (MCAR), or Missing Not At Random (MNAR) [49, 50, 51]. The data set can contain a mixture of all three categories. If the data is MCAR, then the probability of missing data is equal for all the data points. The cause of data being missing is unrelated to the collected data.

Statistics often use MCAR to simplify assumptions. Data that is MAR have an equal probability of being missing only within *groups* of observed data and is therefore a broader category than MCAR. MAR data is more realistic and general scenario than MCAR. If the data is neither MCAR or MAR, then the data is MNAR. If data is MNAR, then the missingness varies for reasons that are unknown to us [50], but can with experimentation be extrapolated (like how with black holes we cannot directly observe them, but their effect on light and gravity can be observed and detected [52, 53]). This further means that missingness can be informative and be exploited to detect matters of interest.

Using the notification and formulas from [50] a more precise mathematical definition can be made of types of missingness. The $n \times p$ matrix $\mathbf{Y}$ contains the data points for p variables and n units (meaning features or dimensions) in the sample from a selection (in the generic sense). A matrix $\mathbf{R}$ is defined to store the location of missing data in Y, with their elements denoted as r_{ij} and y_{ij} respectively where $i = 1, ..., n$ and $j = 1, ..., p$. If y_{ij} is observed then $r_{ij} = 1$ and if y_{ij} is missing then $r_{ij} = 0$. The distribution of $\mathbf{R}$ can be dependent on the ordered pair of observed and missing values $\mathbf{Y} = (\mathbf{Y}_{obs}, \mathbf{Y}_{mis})$. Let ψ be a vector that contains missing data model parameters. A general model can then be expressed as $Pr(\mathbf{R}|\mathbf{Y}_{obs}, \mathbf{Y}_{mis}, \psi)$. If $\mathbf{R} = \mathbf{0}$ denotes that all elements of $\mathbf{R}$ are zero and all values are missing, $\mathbf{R} = \mathbf{1}$ denotes that all elements of $\mathbf{R}$ equal one and all values are observed in $\mathbf{Y}$, then a condition for data that is MCAR then become

$$Pr(\mathbf{R} = \mathbf{0}|\mathbf{Y}_{obs}, \mathbf{Y}_{mis}, \psi)_{MCAR} = Pr(\mathbf{R} = 0|\psi) \tag{2.3}$$

The MCAR data depend only on some parameters or factors within ψ. If the data is MAR then

$$Pr(\mathbf{R} = \mathbf{0}|\mathbf{Y}_{obs}, \mathbf{Y}_{mis}, \psi)_{MAR} = Pr(\mathbf{R} = \mathbf{0}|\mathbf{Y}_{obs}, \psi) \tag{2.4}$$

and depend only on the observed values and the factors in ψ. Data that is MNAR depend then on all the conditions and the condition cannot be reduced:

$$Pr(\mathbf{R} = \mathbf{0}|\mathbf{Y}_{obs}, \mathbf{Y}_{mis}, \psi)_{MNAR} = Pr(\mathbf{R} = \mathbf{0}|\mathbf{Y}_{obs}, \mathbf{Y}_{mis}, \psi) \tag{2.5}$$

meaning the MNAR data also depend on information that has not been observed.

In [54] a point is made that electronic health records are more complex than what can be captured by the categories of MCAR, MAR, and MNAR traditionally used, as they contain a mixture of the three categories. If data is missing, this could be due to either lack of collection or lack of documentation. A lack of documentation would be a mixture of MAR and MCAR, while a lack of collection would be considered MNAR. For the SSI data used in this work, if data is MAR or MCAR, this may be because someone ordered tests that were not taken because of administrative error, or the tests were taken, but for one reason or another the results was not registered or measurable, maybe because of equipment malfunction or human error. It may also be that the medical practitioner in charge of the patient did not find it necessary to take tests, either because they deemed the test not necessary, or due to personal bias (conscious or unconscious). If the medical practitioner choose to not take tests, this can be seen as data that is MNAR and can be informative and of interest to a classifier. Other examples of data MNAR might be a sensor or transducer that under certain circumstances fails to register a sample, or have data missing due to bad energy supply, environmental noise (like a car passing by or a sunbeam), or failing components. Inspired by [55] one can see in figure 2.2 that the SSI data have informative missingness in the sense that there is some correlation between missing data of tests and incidence of SSI. The figure will be discussed in section 5.1.

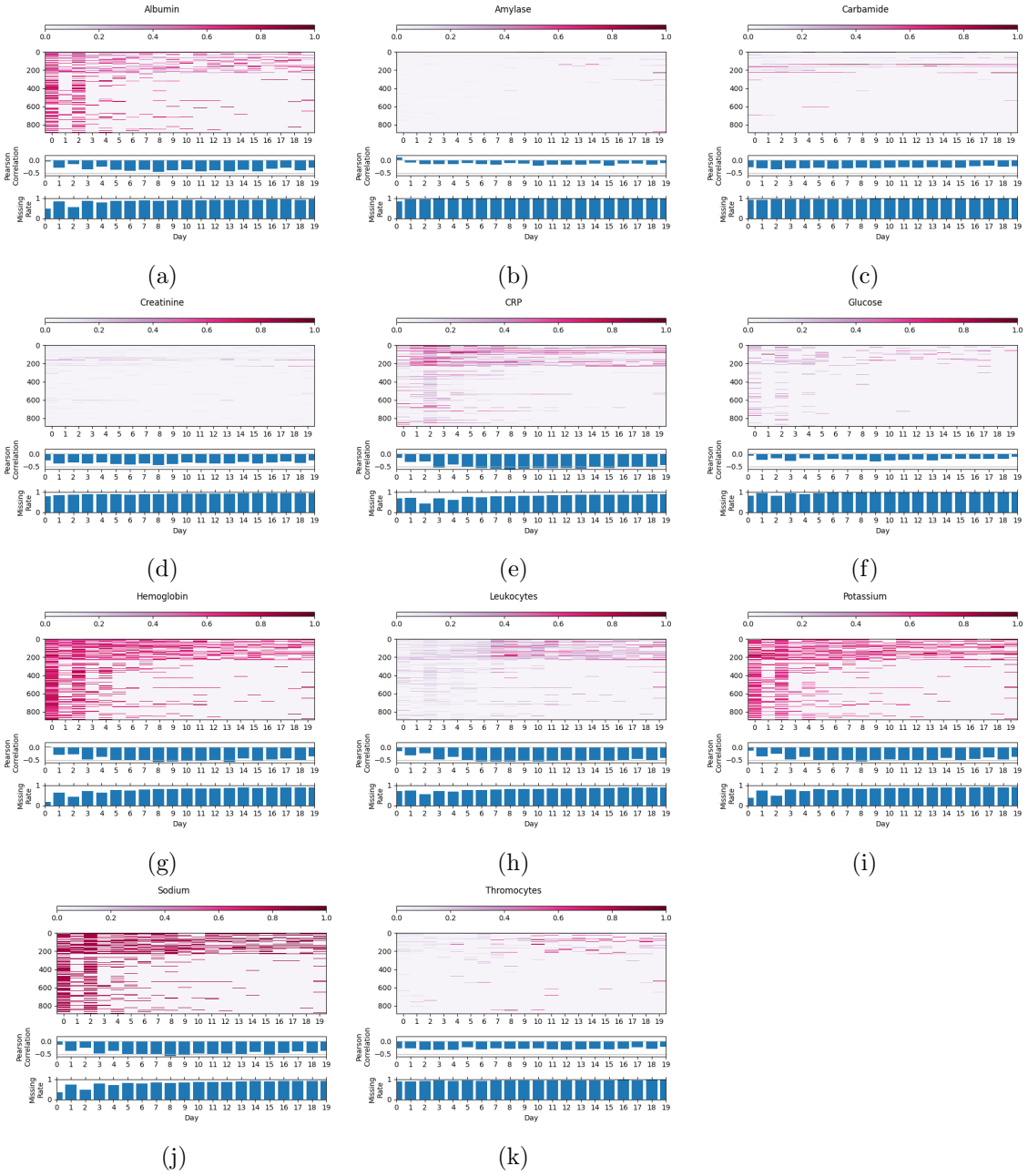


Figure 2.2: Plot demonstrating how missing data can be informative. From (a) to (k) the figures show respectively for albumin (a), amylase (b), carbamide (c), creatinine (d), CRP (e), glucose (f), hemoglobin (g), leukocytes (h), potassium (i), sodium (j), and thrombocytes (k). The upper images show the normalised data for that blood test, the middle a bar-plot of the Pearson correlation between missing data and SSI for that day, and the lowermost show a bar-plot of missing rate for that day. More dark colour in the upper image indicate higher values, while a lighter colour indicate lower values.

2.3.2 Imputation

When data is missing, it is difficult to correctly classify the result. Some types of research may filter out samples with these missing data, but this is not possible when working with e.g. health data, as it is not continuous and there are often time gaps between each data registration that have unknown values. One solution is to fill in the data that is missing. This is called imputation and several options exist for imputing. The ones used here are zero-padding, last observation carried forward (locf), mean, median, minimum, maximum, and combined imputation.

For locf, one carries forward the last valid observation to the next entry. When doing imputation using the mean, median, minimum, and maximum value, the imputed value is extracted from the data set X with a row vector $\mathbf{x_{ni}}$ such that $\mathbf{x_{ni}} \in \{x_{ni} : x_{nij} \text{ not } \emptyset\}$, where i is the dimension, j the time axis of testing, and $\emptyset$ stands for empty or missing data entry, for each n patient of N patients.

It is also possible to use multiple imputation values and combine them, taking the average at time t. In this work, this has been dubbed the 'combined' imputation method.

2.3.3 Augmentation of data with missingness features

A general method can be used to pre-process the data and augment it. These augmented data increase the number of features extracted from a data set which can be exploited by different classifiers. This method will be tested with the classifiers presented in the coming sections below.

Similar to [55] a masking vector and a time interval count on the missing data to augment the data is employed, but the approach of the implementation differs and is a bit simpler. Contrary to [55], the time stamp is ignored due to the data used in this work simply not having them, and the mask is set to true when data is missing instead of when data is there. The time interval counter starts at zero at time $t = 0$ and each time data is not observed in dimension d the time interval counter is incremented until data is again observed, at which point the time interval counter is reset. When a new missed observation of data is detected, the time interval counter is incremented from zero again. In other words, if the data is $x_{d,t} \in \mathbf{X}$ at time index t in dimension d, the masking vector $m_{d,t} \in \{0, 1\}$, and time intervals $\delta_{d,t} \in \mathbb{R}$ then

$$m_{d,t} = \begin{cases} 1 & \text{if } x_{d,t} \text{ is not observed} \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

$$\delta_{d,t} = \begin{cases} \delta_{d,t-1} + 1 & t > 0, m_{d,t-1} = 1 \\ 0 & t > 0, m_{d,t-1} = 0 \\ 0 & t = 0 \end{cases} \tag{2.7}$$

The application of the augmented data is described in section 3.1.3 using formula (2.6) and (2.7). If the data had a timestamp $s_t \in \mathbb{R}$, for when the tth observation was made where $s_1 = 0$, then the implementation used in [55] would be used. The equations would

then be

$$m_{d,t} = \begin{cases} 1 & \text{if } x_{d,t} \text{ is observed} \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

$$\delta_{d,t} = \begin{cases} s_t - s_{t-1} + \delta_{d,t-1} & t > 0, m_{d,t-1} = 0 \\ s_t - s_{t-1} & t > 0, m_{d,t-1} = 1 \\ 0 & t = 0 \end{cases} \tag{2.9}$$

though applying masking on either observed or missed data comes down to preference, data set, and classification method, and the conditions in formula (2.7) and (2.9) must be adjusted accordingly. Using the wrong condition might result in NaN values for the loss function.

2.4 k Nearest Neighbor classifier (kNN)

The kNN classifier is a distance-based classifier. It is an algorithm that works by finding the nearest k neighbors with known labeling and assign the label of the sample to be classified to be the same as the majority of the closest k neighbors. It is a bit different than other classification methods in that there is no training step involved in building a model. Instead, the data points themselves are the model, and the classifier compares them to infer the predicted label [56]. The pseudo-code below describes roughly the kNN algorithm, given the data defined as x, and y defined as the true labels.

kNN pseudocode

```
    for n of N patients:
        for m of N patients:
            if not n == m:
               d[n,m] = distance between x[n] and x[m]
            else:
                d[n, m] = 0
        end for
    end for

    k = desired odd k
    for each patient n of N patients:
        indexes = argsort(d[n, :])
        indexes of nn = indexes[1:k+1] #given indexing start at 0
        labels of nn = y[indexes of nn]
        predicted label[n] = most common( labels of nn )
    end for
```

If the DTW distances are pre-computed, the classification, i.e. assignment of labels in the lower-most for-loop, take little time.

If there are several ks to be tested, then a loop can be added around the lowermost for-loop to test for all k. It is assumed that k is odd numbered.

Mathematically, the above pseudocode can be expressed as the following. If the distance function is the Euclidean DTW distance DTW_E, $\mathbf{X_n}$ and $\mathbf{X_m}$ are data for patients n and m, where $n = 0, 1, ..., N$ and $m = 0, 1, ..., N$ given that there are N patients, then

$$\mathbf{D}_{n,m} = \begin{cases} DTW_E(\mathbf{X_n}, \mathbf{X_m}) & \text{if } n \neq m \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

and the estimated integer label $\hat{y}$ for patient n is

$$\hat{y}_n = \text{round}\left(\frac{1}{k}\sum_{i=1}^{k+1} y_{a_i}\right) \tag{2.11}$$

where

$$a = \text{argsort}_m(\mathbf{D_{n,m}}) \tag{2.12}$$

with the true label y, a as the index of the sorted set of values of the DTW distance matrix $\mathbf{D}$ along axis m (second axis), and k is the number of nearest neighbors considered.

2.4.1 The effect of different values of k

A value of $k = 1$ makes it effective and quick to find that one nearest neighbor, but the algorithm is sensitive to noise. Using a higher value of k slightly increases the computation time (if distances have been pre-computed), but is more robust. Too high a value of k is however counter-productive.[57, 58, 59]

2.4.2 Weakness of using kNN classifier

If the data is highly skewed toward one class, then the assignment of labels will be drawn to this "majority" class, since most of the neighbors of any point will mostly belong to that class. Assigning weights to the nearest neighbors before computing the most common label can mitigate this.[60]

There are also other important weaknesses of the kNN classifier. The kNN classifier does not learn any parameters, and therefore it is necessary to store the whole data set. Importantly, at inference time, a new sample must be compared against the whole training set to perform the classification. This process is computationally expensive, but can be mitigated to some degree by e.g. storing the distances of the training set in a distance grid matrix and then padding the grid with the distances to the new sample that is to be classified. Feature reduction is also possible to do in order to reduce computation cost when computing the DTW distances, but the data set used in this project consist of very short time series.

2.5 Support Vector Machine classifier (SVM)

The SVM classifier is a classifier that tries to optimally separate points into classes. It does this by finding a decision boundary that separates them, with a margin on both sides that controls the acceptable error through a parameter C. The margin is chosen such that it maximally separates the classes. This is called a "maximum-margin hyperplane" [61]. A linear SVM uses a linear kernel function which produces a line or hyperplane, while a non-linear SVM uses a non-linear kernel resulting in a curve or manifold.

2.5.1 The general SVM

A visual representation of the SVM method with hard margins can be seen in figure 2.3. Let the hyperplane separating the classes be defined as

$$\mathbf{w} \cdot \mathbf{x} - b = 0 \tag{2.13}$$

where w is the normal vector to the hyperplane, $\mathbf{x}$ is a stochastic vector containing a data point, and b is bias. Labels are defined as y_i where each y_i belong to its corresponding

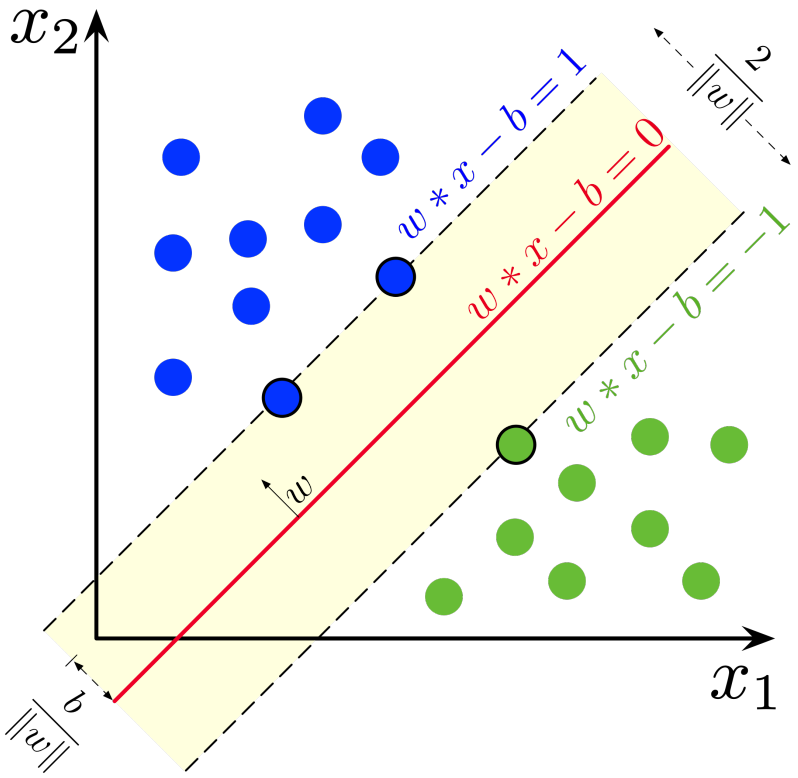


Figure 2.3: A 2D plot showing a max-margin line separating two classes. By Larhmam - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=73710028

vector x_i. The figure shows the hard margins that are solved for $wx - b = +1$ and $wx - b = -1$, and the width of the total margin between these lines is $\frac{2}{||\mathbf{w}||}$. The line separate is used to separate and classify each class. If a data point is above the boundary represented by $wx - b = +1$, it is classified as a positive result with the label 1, and respectively anything on or below $wx - b = +1$ is classified as a negative with the label 0. The offset from the origin to the hyperplane along the vector w is $\frac{b}{||\mathbf{w}||}$. The optimization problem can be summed up as

$$\min_{||\mathbf{w}||} \qquad y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 \text{ for } i = 1, ..., n \tag{2.14}$$

The $\mathbf{x_i}$ that lie on the margins are called support vectors, hence the name Support Vector Machine.

The margins used in this paper for classifications are soft margins that allow for imperfect classification of the training data points. This means that we seek a margin that allows a certain amount of data points from either class inside the margin, or on the wrong side of the margin with respect to the class label, as described in [3], [44] and [61]. This relaxation is referred to as slack and the amount of slack is determined by the parameter $C > 0$, which controls the trade-off between incorrectly classified labels and maximising the width of the margins. The value of C is a hyper-parameter which must validated and is usually through a grid search and sampled on a logarithmic scale.

2.5.2 The kernel trick

The data might not be linearly separable, and may require a non-linear decision boundary in order to achieve good classifier performance. This can be solved by going to a higher dimension using something called the kernel trick.

A kernel is a similarity function for pairs of data points specified beforehand by a user. The kernel trick utilizes the fact that many machine learning algorithms depend on the data only through inner products between data points, and not through the coordinates of individual data points. It is therefore possible to transform the data implicitly by replacing the inner product with some kernel function that corresponds to an inner product in a transformed space. For instance, the linear inner product known as the scalar product or dot product can be replaced by the Gaussian kernel function, also known as the radial basis function kernel, or a polynomial kernel. Hence, instead of determining the explicit transformation and computing the individual coordinates of the data, one can compute the inner product between all pairs of it through the kernel function. Oftentimes, this is known to save computation time.

In the higher dimension the points can be separated using a linear hyperplane or manifold. The points are mapped from input space to feature space $\mathrm{x} \longrightarrow \phi(\mathrm{x})$ by the kernel function. The linear SVM can be expanded to a non-linear SVM by substituting the linear inner product with other non-linear inner products [62, 63]. The non-linear kernel function used in this work is the Radial Basis Function (RBF) [64]. The natural choice of RBF kernel is usually the Euclidean distance, but if one want to use a distance-based classifier this can be switched out with the DTW distance if one want to use the DTW distance for time series. This is done by substituting the RBF:

$$K(\mathbf{x_i}, \mathbf{x_j}) = e^{-(\gamma \cdot ||\mathbf{x_i} - \mathbf{x_j}||^2)} \tag{2.15}$$

with the function

$$K(\mathbf{x_i}, \mathbf{x_j}) = e^{-(\gamma \cdot DTW(\mathbf{x_i}, \mathbf{x_j})^2)} \tag{2.16}$$

which is similar to it, where the kernel function $||\mathbf{x_i} - \mathbf{x_j}||$ is being substituted with the Euclidean DTW distance[44]. The parameter γ is a scaling value for the kernel, and x_i and x_j are data points to be compared.

2.5.3 Parameter optimization of C and γ

The parameters of C and γ can be found in different ways [61, 65]:

1. Grid search: Pairwise testing of sets of values for C and γ on the validation set using cross-validation or Bayesian optimization to find the minimum parameter values that give the smallest classification error [61, 66]. The prarameter γ is usually predetermined in a list, while C is exponentially increased by a factor of 10, starting from 1.

2. Set a heuristic rule for the value of γ and search for C: To determine γ in a heuristic way one assume that the median of the distance values, the mode value (typical value) of the histogram, or a function of one of these two is a "typical distance" which can be the scale value σ in the RBF kernel, where

$$\gamma = \frac{1}{\sigma^2} \tag{2.17}$$

   A suitable C can then be found through a linear search with respect to a score metric, like balanced accuracy and the F_1 score on the validation set. This is a simplification of grid-searching.

2.5.4 Weaknesses of the SVM

The SVM method has some weaknesses. It needs the input data to be fully labeled, it is by itself only suited to two-class problems, and the values of the parameters are not easy to interpret after training [61, 67]. There is a type of SVM that don't need full labeling called semi-supervise SVM (S3VM) [68] where only some of the data is labeled and the non-labeled data is inferred from the labeled. If there are multiple classes, one class at a time is selected and compared against the rest, though this is not relevant for this text. If the data is highly biased toward one class it can present an issue of correctly classifying, but this can to some degree be mitigated using either class weighting or sample weighting.

2.6 Temporal Convolution Network

Temporal Convolutional Networks (TCN) are a newer type of deep learning network [69] which utilises dilated convolution to perform classification or regression on sequential series of data. In the case of time series data, the dilated convolution is also causal. Convolution has been used for a long time to classify time series [70, 71, 72], but due to computation and memory limitations at the time it could only be used for small networks and limited training data. In the last decade it has gained more popularity as the availability of computation power, memory, and labeled data has increased and made convolutional neural networks (CNN) viable competitors to other machine learning methods [73]. It was shown in 2004 that CNNs could be made faster [74] by using GPUs, but what made CNN popular was a network made in 2012 called AlexNet [75, 76] which utilized GPU computation to classify a huge data set called ImageNet[77]. CNNs have been applied to many other disciplines like signal processing, filtering, linear data transformation, detection, feature extraction and generation, auto-encoding, imputation [62, 78], and has even

been used to win board games like chess and go against human opponents with the help of self-reinforcement learning.

A CNN consist of several layers or units made up of a convolutional layer, an activation layer, some manner of pooling, and a fully connected (dense) layer, in addition with dropout, normalisation layers, and other techniques. CNNs have been adapted into residual neural networks (RNN) [79, 80], gated recurrent unit (GRU) [81] networks, long short-term memory (LSTM) [82, 83] networks, and recently the TCN. A type of network (which does not necessarily need to be a CNN) has been used to detect patterns without labeled data using what is called generative adversarial networks (GAN), where one network tries to generate fake data that seem real, while another network tries to detect whether its own input data is fake or real, though much work remains to make such networks safe in certain contexts [84]. In this section the various parts of the TCN will be explained starting with convolution, then residual networks, followed by dilation, and finally regularization.

2.6.1 Convolution

Convolution is a mathematical operation where a weight filter function or kernel is multiplied element by element with some data structure like a time series and summed. For the 1D case it can be written generally as

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.18}$$

where s is a one-dimensional signal as function of t produced from input x weighted with w [62, 85]. Convolution can also be applied on 2D data like images [86, 62], but the TCN works with 1D time series. When training a machine learning network, the weights of the filter in w are flipped (that is, the order is reversed) when dot-multiplying during forward feeding of the network. During back-propagation the weight filter is transposed. Convolution is similar to cross-correlation; the only difference is that convolution flips the filter, whereas cross-correlation does not.

There are several advantages and strengths in using CNNs. They have more efficient memory use and statistical efficiency compared to dense matrix multiplication, because they are using weight sharing and can with the help of dilation or pooling expand the receptive field of neurons/perceptrons. Weight sharing means that they use same parameter for more than one function in the model. I.e., in place of learning one parameter for each separate location in e.g. an image, the same weights are applied to every location. This makes detection of patterns translation invariant.

As the hidden layers of the CNN updates it automatically extract features of the data, and each layer extract more abstract features as the data propagate deeper into the network. The filter kernels used for feature extraction are many. A way to vary them using the same principle of convolving can be by changing the padding of the data or the stride of the function. Types of padding are: valid, same, full [62], and causal. Causal padding will be explained in section 2.6.3. Stride can be adjusted to down-sample data thus reducing the amount of locations visited by a kernel.

2.6.2 Residual Network

Just as convolutional networks are an old idea and concept, networks with time delay are just as old [79, 87, 88]. Over time the idea have been improved into the residual neural

networks we use today which use skip connections [80] and residual blocks which will be explained in this section.

There is a general principle that defines a residual network. They have some kind of looping where an output from one or several layers are connected to the input of another previous layer, making them recurrent. This allows the network to learn connections between current and previous input by giving memory to the network. It is limited how far into the past the network can remember, but this depends on the amount of block units, and in the case of so-called gated units [89, 90] they can also forget and learn when to forget. A gated unit uses recurrence to train a "valve" or gate parameter to control the flow and strength of the temporal connections letting a recurrent layer work at different time scales and adapt the network structure to the input.

The advantage of residual networks is that the model always has the same size irregardless of the input sequence length due to state-based transition between units. It is possible to use the same function f for transition between states using the same parameter at each stride/time step, allowing the parameters to learn parameters/pattern which apply to all steps and series lengths instead of one model for each combination of step iteration. This simplifies the model and reduces the number of parameters needed. The network then makes generalization possible due to the parameter sharing. The disadvantage with the reduced number of parameters is that it makes optimization more difficult because one is re-using the same parameters at different time-steps. Learning the parameters take a long time because of this. It is also difficult to predict/classify when there are missing values, as mentioned in section 2.3.

When a residual network is trained for a long time to learn the long-term dependencies of a time series, the gradient back-propagation through the network might vanish or explode. A vanishing gradient occurs when the input to the activation function is such that the derivative of the activation function is close to zero, in which case adaptation and learning in preceding layers stop. An exploding gradient is the opposite case, when the derivative of the activation function becomes too big, which can make the updating of weights unstable. Vanishing and exploding gradients is a known problem with RNNs. There are known methods for mitigating this, like the spectral radius limit used in Echo State Networks (ESN)[91], use of leaky units [92, 93], or one can use skip-connections [94, 95, 96]. The TCN uses skip-connections. Skip connections are used to preserve, stabilize, and smooth the gradient so it reduce the chance of vanishing or exploding, and reduce the chance of getting stuck in a local minimum if the gradient is jagged. Skip connections are made by adding a connection from the input to each residual block as shown in figure 2.4, but one could add connections in multiple ways for other types of networks.

As mentioned earlier, the TCN has skip connections, as can be seen in figure 2.4 in the residual block "layer" on the right. The input is added to the connection between each residual block. The residual block will be explained later. After the input passes through the residual blocks we get a number of filter-outputs. The number of filter outputs depends on the list of dilations for each block. Dilations will be explained in the next section. Each residual block has a dilation value associated with it. If the list of dilations consist of the values $\{1, 2, 4, 8, 16, 32\}$ (one value for each residual block, minus one, in this case 5 blocks) then the number of filter outputs will two times the last entry value in the list, making the number of filter outputs 64. The filter-outputs are passed in a cascading manner passed into (i) a layer normalization layer [97]. Any normalisation layer is fine: batch normalisation, layer normalisation, or weight normalisation, but because of batch normalisation values being dependent on the size of a batch, it's worth considering other normalisation schemes. The normalized values then (ii) passes into a dropout layer [98]

before passing into (iii) a dense layer of neurons/perceptrons. The output from that then (iv) go into an activation layer with rectified linear units (ReLU). Then we repeat step (i)-(iv) with a different dense layer (no recursion or parameter sharing) followed by layer normalization, then dropout, and finally the output neuron/perceptron.

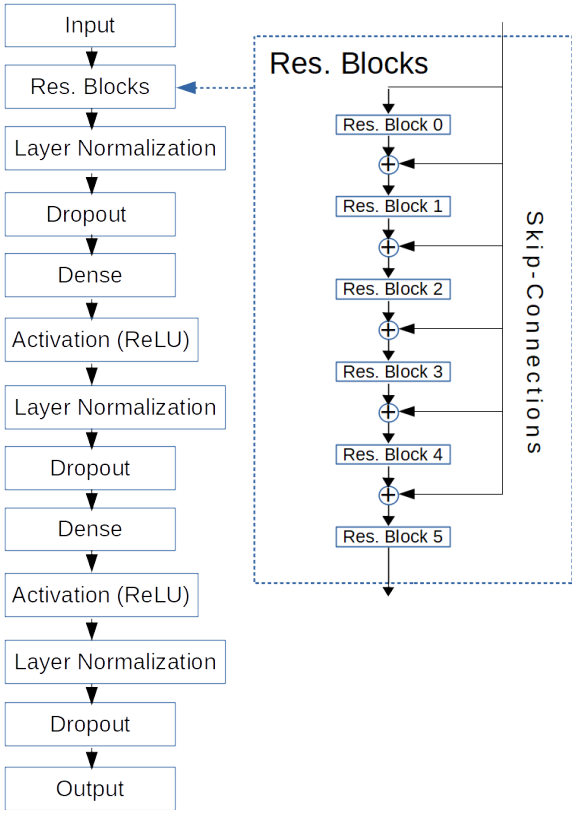


Figure 2.4: The architecture used in this paper for the TCN classifier. The TCN in this work have 6 residual blocks corresponding to the dilations $\{1, 2, 4, 8, 16, 32\}$. The number of residual blocks depends on the number of dilations, so a different network than the one in this work may have more or less. The general architecture of the residual blocks can be seen in Figure 2.5.

A general representation of the residual blocks used in figure 2.4 can be seen in Figure 2.5. The flow of information splits into two paths from the input into the block. In the left path of the figure, the function $\mathcal{F}(\mathcal{X})$ represents the convolutional filtering in the block which is adapted during training. In the right path, an identity operator is applied to the input $\mathcal{X}$, followed by an optional 1×1 convolution that is used for dimensionality reduction when the number of feature maps in $\mathcal{X}$ and $\mathcal{F}(\mathcal{X})$ do not match. The copy of $\mathcal{X}$ passed on to the right path is used to conserve the gradient during back-propagation. The first layer is (i) a dilated causal convolution. Dilation and causal padding will be explained in section 2.6.3. Then follow (ii) the activation layer (ReLU), (iii) layer normalisation, and (iv) dropout. Then (i)-(iv) follows again in a cascading manner. The two convolutional layers in the residual block have shared parameters. The output of the right and left paths are then added together. The output of a single residual block can therefore be seen as the mathematical expression

$$o(\cdot) = \mathcal{F}(\mathcal{X}) + \mathcal{X} \tag{2.19}$$

Each block stacks the dilation and expands the field of view of the neurons that follow after the last residual block.

The receptive field of a hidden unit, how much of the input it sees, in a TCN can be calculated as:

$$1 + N_s \sum_{n=0}^{N_b - 1} (\mathbf{d}[n] \cdot k[n] - 1) \tag{2.20}$$

where N_s is the number of stacks, N_b is the number of residual blocks per stack, $\mathbf{d}$ is a vector containing the dilations of each residual block in one stack, $\mathbf{k}$ is a vector containing the lengths of the filters of each residual block in one stack, and n is a residual block index [99].

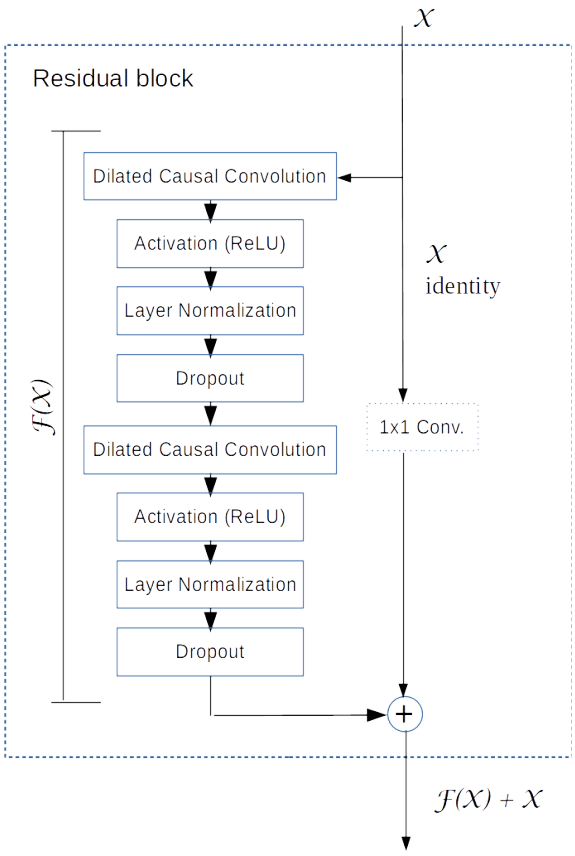


Figure 2.5: The residual block used in TCN. The skip connection passes on a copy of the identity matrix of X. The $1x1$ convolution is used to harmonize the dimensions of the outputs of the two paths. Figure inspired by original ResNet in [80] and figure used in [100].

2.6.3 Dilation

A TCN uses dilation as an alternative to pooling [101, 62] to reduce the computation and memory cost of the network while keeping the information about the input data. Dilation is a way of expanding the receptive field of the network by having the convolutional layer look at cells or indexes of the input with added steps in-between, as illustrated in figure 2.6. This is done in order to detect patterns at different time scales, with some values being dependent on earlier occurrences of an event or value. An obvious way to illustrate this is by looking at e.g. sentences, where the next word is affected both by previous words and future words due to syntax. Another example is weather and climate patterns where

the current weather is dependent on many factors earlier in time. Each residual block in a TCN have two causal dilated convolution layers with the same dilation factor within their block. Being causal means that the convolution layer only take into consideration past and present values, and cannot use future values.

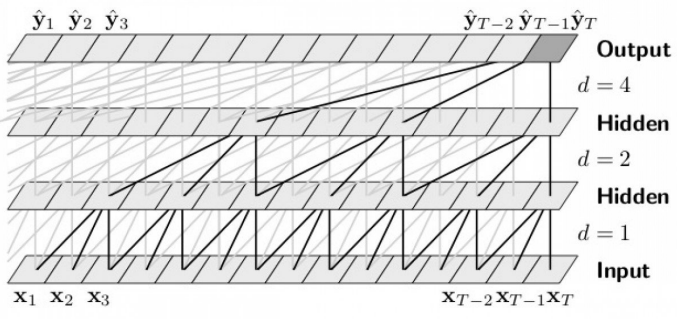


Figure 2.6: Shows how the receptive field of a neuron expands through dilation, where the bottom is the input and the top the output. Image source: [100], derived from [102].

In figure 2.6 one can see an example of how dilation works. Starting at the bottom and going up, one can see that from the input to the first layer of convolutional filtering there are connections down to each index of the input with dilation $d = 1$. Then in the next layer above where the dilation value is $d = 2$, there are fewer connections between that one and the next layer above it, which has a dilation value of $d = 4$. The output would then sample from every fourth index of the final filter layer. Looking at the figure as a whole, it is then easy to see how the receptive field of a single output neuron is expanded while retaining the information of the input and keep the memory usage low.

With the filter being of a fixed size, in order to look into the past when convolving the input, it is necessary to pad the data to retain the constant size of the input. This is done by padding with zeros in each layer of convolution at the beginning of each time series in each channel. This forces the convolution to be causal and only look into the past. In this work we cannot use data from the future as patient data need to be present for real-time analysis.

In deep learning, another word used for filters is kernel. Being short for convolutional kernels, these are different from the kernel functions mentioned under the discussion of SVMs. Instead of kernel functions, convolutional networks utilize stacks of matrices with weights that slide and act as filters when multiplied with the data (hence convolution filter). Kernels determine the receptive field of a convolution layer, which depend on their size and position of the weights. The weights in a kernel need not be symmetrically arranged. Kernel size can be found by using the formula:

$$K_s = k_w \cdot k_h \cdot c_in \cdot c_out + c_out \tag{2.21}$$

where K_s is the kernel size, k_w the width of the kernel, k_h the height of the kernel, c_in the number of input channels (feature maps), and c_out the number of output channels. Stacking smaller kernels is preferred for convolution networks compared to stacking larger ones. Smaller kernel widths (and height) reduce the number of parameters needed for training, and forces the filters to adapt more, while stacking the kernels increases the depth of the network and the number of channels. A deeper network generalizes its training data better and abstracts the data more. [62, 103, 104, 105]

This work uses Keras to implement the TCN. Keras is a code library which simplifies and abstract parts of Tensorflow, another code library used to code many machine learning

applications in the Python code language. In the Keras implementation of TCN dilations, the receptive field can be calculated as

$$R_{field} = 1 + 2 \cdot (K_{size} - 1) \cdot N_{stack} \cdot \sum d_i \tag{2.22}$$

where K_{size} is the kernel size, N_b is the number of residual blocks, and $\mathbf{d}$ is a vector containing the dilations of each residual block in each stack [99], and the sum is over all residual blocks. A more detailed explanation of dilation and how to compute padding, the number of layers, and number blocks can be found in [106].

As a sidenote, just as pooling can be used to locate objects and patterns with class activation maps through global average pooling [28], this can also be done in a similar manner with TCNs and other convolution networks with dilation by using attention weights [107]. Both methods increase detection of patterns and allow non-experts to visually inspect the decision making of the network and how much emphasis/attention it puts on certain periods of time and data (e.g. blood samples over time), but this has not been used in this work.

2.6.4 Regularization

In order for the network to be stable and generalized for its purpose, several well-known methods are employed to regularize it. In order to avoid having the gradient go to zero, known as the vanishing gradient problem, rectifier linear units (ReLU) are used as the activation function. As there are only two classes in the problems considered in this work, activation functions such as softmax or sigmoid functions are not employed at the output, and a simple linear output and check on whether the sign of the output is positive or negative can be used. Though, if there are more than two classes, it is natural to use either softmax or sigmoid at the output. To constrain the parameter weights, the activation function need to be clipped or scaled. This was chosen to be done using layer normalisation, as previously mentioned in section 2.6.2. In order to avoid over-fitting, and to make the network generalize better, one can use dropout [98] to remove connections or zero out entire channels during back-propagation. Dropout prevents over-fitting by forcing the active neurons to learn without relying on neighboring neurons, and act as a bagging mechanism where several network models can be tested quickly. Another good way to make the network more generalized and increase the amount of training data is to add noise to your data. All of these methods make the network model more robust.

The network also needs to be initialized, and the weights must be adapted. This is achieved by using one of the most common optimizer algorithms called the adaptive average momentum (ADAM), although there are several other alternatives, of which the most common are listed in [108].

2.7 Score metrics for classifiers

There are several methods for scoring the performance of a classifier. Some of them are the confusion matrix, accuracy, precision, recall, specificity, balanced accuracy, and the F_1 score. The aforementioned score metrics are briefly commented on below along with their weaknesses.

2.7.1 Confusion matrix

A confusion matrix is a type of table used to compare each predicted label with their corresponding true labels of the data. It can be used to check if there is an unbalance

of label assignments, and the cells of the matrix can be used for computing several score metrics. The row values signify the true labels, and the columns the predicted values. When comparing labels, one can look at it as an AND operation (i.e. intersection, $A \cap B$) between the predicted labels and true labels. As an example, for a confusion matrix CM, if the predicted label is 0, AND the true label is 0, then this gives a TRUE value, and is added to CM_{00} of the confusion matrix, and called a True Positive (TP). If the predicted label is 0, AND the true label is 0, then this give TRUE, and is added to CM_{11}, and called True Negative (TN). However, if the true label is 0, AND the predicted label is 1, then we have a False Positive and it is added to CM_{01}. It is similar when we have a False Negative (FN), when the true label is 1, AND predicted label is 0. This is repeated for all predicted and true labels and summed up for each cell in the confusion matrix. A weakness of only using a confusion matrix is that you cannot tell at a glance how good of a fit the result from a classifier is.

Figure 2.7: Example of how a confusion matrix looks. TP stands for True Positive, FP for False Positive (a.k.a. Type II error), TN is True Negative, and FN is False Negative (a.k.a. Type I error)

	Predicted 0	Predicted 1
True 0	TN	FP
True 1	FN	TP

A type I error is another name for a false positive while a type II error is the opposite, known as a false negative. If one define the null hypothesis H_0 that a given data point is class 1 (positive), then a type I error would be labeling a point which is another class as class 1, making it a false positive. A type II error would be mislabeling a data point which is class 1 as another class.

2.7.2 Accuracy

If there are N patients, then the accuracy of the classification will be

$$\text{Accuracy} = \frac{\text{sum of correctly labeled}}{\text{total number of patients}} = \frac{TN + TP}{N} = \frac{TN + TP}{TN + TP + FP + FN} \tag{2.23}$$

If the amount of labels is highly skewed toward one class, either in the data set or predicted, this score measurement is deceiving. One might get a high accuracy, but it may not be precise and sensitive enough.

2.7.3 Precision

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.24}$$

The randomness, random error, of the classification can be determined by the precision of the classifier. A weakness with using only either precision or accuracy is that something can be accurate, but not precise. This means that there is a huge spread between the target centered around it. It can be precise, but not accurate, meaning a small spread, but a systematic error can have shifted the accuracy. A good classifier should have both good accuracy and precision. If it's not good, one can experience either a type I or type II error.

2.7.4 Recall

True positive rate (TPR), also called sensitivity, tells us that we have a detection, but not what is detected. It tells how much the classifier avoid false negatives. A higher recall indicates lower type II error rate.

$$\text{Recall} = TPR = \frac{TP}{TP + FN} \tag{2.25}$$

2.7.5 Specificity

Specificity is also called True negative rate (TNR). A high specificity means the classifier can find specific classes with high certainty. A higher value indicate lower type I error rate.

$$\text{Specificity} = TNR = \frac{TN}{TN + FP} \tag{2.26}$$

The classifier should be both sensitive (have good recall) and have good specificity. Having good specificity means nothing if a test or classifier is not sensitive enough to detect what it needs to.

2.7.6 Balanced accuracy

$$\text{Balanced accuracy} = \frac{TPR + TNR}{2} \tag{2.27}$$

If data is imbalanced, with many data points of one label, and a lot less of another, then accuracy become misleading to use. Balanced accuracy is the mean of true positive rate and true negative rate, recall (sensitivity) and specificity.

2.7.7 $\mathbf{F}_1$ score

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.28}$$

The F_1 score is defined as the harmonic mean of precision and recall. This score measurement covers the weakness of only using precision or recall (sensitivity) by themselves.

Chapter 3

Method

3.1 Data

Two different data sets are employed in testing the algorithms. The first one is the SSI data used in [109], which naturally contains missing data. It consists of data from 883 patients with 11 different blood tests taken over 20 days, with information MCAR and MNAR. It was structured as a row for each patient, with 11 blood tests of different variables taken every day over a period of 20 days. Blood tests consisted of values for hemoglobin, leukocytes, CRP, potassium, sodium, creatinine, thromocytes, albumin, carbamide, glucose, and amylase. The data was already zero-imputed when received. In order to easier perform indexing, the data was reshaped into an array with shape (number of patients, number of tests, number of days).

To compare the machine learning results, a second data set was chosen. This set is the uwave data from [110], which is made up of complete data recorded with an accelerometer while a subset of people do various predetermined gestures. It originally has 8 different classes, but for this work only two were chosen, the first and second gesture pattern, to be more manageable. The training and test data was concatenated, shuffled, and later split as described below in order to have more control of the sizes used for training, validation, and testing. Each sample's data consist of x, y, and z spatial coordinate values recorded over 315 time-steps, for a total number of 1119 recorded three-dimensional time series. Similar to the SSI data, the uwave data was re-shaped into (number of samples, number of dimensions, number of time-steps).

In order to compare the two data sets using imputation, the uwave data needed to be manipulated by removing data while imposing some form of informative missingness. This was done by a stochastic procedure where the dimensions and time steps of each sample to be removed was drawn randomly from normal and uniform distributions. The dimensions selected for deletion were drawn from a normal distribution. The time indexes selected for "deletion" (set to zero) from any given dimension was then sampled from a uniform distribution. Depending on the class and dimension, each "deletion" start index had a number r of consecutive "deletions" associated with it. In this manner, informative missingness should appear as stochastic, but at the same time be detectable by a machine learning algorithm. The deletion procedure is presented mathematically in the following manner:

1. The dimension d_i to delete from is determined by drawing a random number

$$\theta \sim N(\mu, \sigma^2) = N(p_d \cdot n_d, n_d/4)\,, \tag{3.1}$$

where $N(\mu, \sigma^2)$ denotes a normal distribution with mean μ and variance σ^2. Here, the mean $\mu = p_d \cdot n_d$ and variance $\sigma^2 = n_d/4$ are parameterized by the number of dimensions n_d and a parameter p_d. By setting the parameter p_d to a number between 0 and 1 you can scale which dimension θ is centered on. Next, the dimension to delete from is determined as

$$d_i = \begin{cases} \text{round}(\theta) & \text{if } 0 \geq \theta \leq n_d \\ 0 & \text{if } \theta < 0 \\ n_d & \text{if } \theta > n_d \end{cases} \tag{3.2}$$

where round$(\cdot)$ is the round-off function.

2. The number of deletions for the dimension selected in the previous step, denoted n_{missing}, is drawn as

$$n_{\text{missing}} \sim N(p_t \cdot n_t, n_t/4) \tag{3.3}$$

where n_t is the number of time indices in dimension d_i, i.e. the length of the time series, and p_t is a scaling factor between 0 and 1 used to determine the expected number of time indices that will be missing.

3. Then, the time indices for the starting point of the n_{missing} deletions are drawn as

$$t_{\text{missing}} = \lfloor \tau \rfloor, \text{ where } \tau \sim U(0,1) \cdot n_{\text{missing}} \tag{3.4}$$

with $\lfloor \cdot \rfloor$ as the floor function.

4. Lastly a parameter r indicates how many consecutive indexes to remove, starting from index t_{missing} in the given dimension.

In retrospect, it could have been easier and more elegant to draw from discrete distributions in place of the normal distribution, but the chosen approach is functional

The parameter values of p_t for class 1 (relabeled as 0 for the sake of implementation) were chosen as $\{0.01, 0.015, 0.02\}$ for the three dimensions (the spatial coordinates x, y and z), while for class 2 (relabeled as 1) the parameters were arbitrarily chosen to be in the reverse order as $\{0.02, 0.015, 0.01\}$. The parameter p_d was chosen to be $\{0.33, 0.5, 0.66\}$ and kept the same for both classes. These parameters for p_d ensure that the μ in the normal distribution in point 1 used to select the dimension was centered on each of the dimensions. The r parameter values were set to $\{20, 40, 60\}$ and $\{11, 13, 17\}$, respectively, for the three dimensions of class 1 and 2. Combined together this result in a stochastic multivariate distribution of missing values with informative missingness, since each class has distinct parameters that can be utilized for discrimination by a classifier. This procedure results in a random amount of missing values each time it is run. The manipulated uwave data set utilized is one particular realization of this simulation procedure, which has a missing rate of 34.2% for class 1 and 13.3% for class 2, with an average missing rate of 23.7% when combined. This is not a very high missing rate, and certainly not as high as in [109], but should be enough as a proof of concept.

3.1.1 Standardization of the data

Some dimensions in the multidimensional data set might have much larger values than values in other dimensions. This is problematic because some dimensions of the data will be given more impact in the analysis based on the range or scaling of the variable, and

not its information content. The solution is to normalize each variable of the data, i.e. scale the range to be between 0 and 1 (or -1 if negative). Given that there are $n = 1, ..., N$ multivariate time series $\mathbf{X}_n = \{x_{nij}\}$, then each feature value can be scaled with respect to the maximum for the given dimension d over all time steps t, and the data can be normalized using the formula

$$x_{ndt} = \frac{x_{ndt}}{\max\limits_{(n,t)}(x_{ndt})} \quad \forall n, \forall d, \forall t \tag{3.5}$$

The data set should then be standardized.

3.1.2 Imputation of the data

The same imputation methods were used for the classifiers. Imputation methods were selected from the pre-determined options of zero, locf, mean, median, minimum, maximum, and combined imputation, in that order. Since the data already was zero-imputed, this method was covered by the original dataset. The other imputation methods were implemented by searching for the zero values, and replacing these with the appropriate value (locf, mean, median, min, or max).

3.1.3 Augmentation with missingness features

The classifiers were also tested on augmented data as described in section 2.3.3 using formula (2.6) and (2.7). The missingness indicator mask and missingness interval counter was concatenated to the imputed data to form new feature vectors. If the masking matrix $\mathbf{M} = \{m_{dt}\,;\,\forall\, d, \forall\, t\}$ and the time interval matrix $\mathbf{\Delta} = \{\delta_{dt}\,;\,\forall\, d, \forall\, t\}$, then the augmented data matrix $\mathbf{X}^{\mathrm{T}}_{\mathbf{aug}} = (\mathbf{X}_{\mathrm{imp}}, \mathbf{M}, \mathbf{\Delta})^{\mathrm{T}}$, where $\mathbf{X}_{\mathrm{imp}}$ are imputed values. This increases the number of feature vectors by three times and replace the missing values. The augmented data also needed to be scaled, and this was performed using equation (3.5). The augmented data could then be used to make distance matrices using Euclidean distance or DTW distance for the kNN and SVM classifier. The TCN classifier is able to use the augmented data as is.

3.1.4 DTW distances of the data

After imputation, the multivariate time series were sent pairwise into the DTW function for each combination of pairs between subjects. The DTW function makes use of the multidimensional Euclidean distance as a cost function. This gave a matrix D with distances between each pair of subjects. The distance matrix for each imputation method was then written to a file to speed up later computations and testing of code, as computing the DTW distances is quite time consuming.

3.1.5 Splitting the data

The split of data into training, validation and test sets is done as follows: The distance matrix for the given imputation method is read from file. Then a 1D array of indices is made with the same length as the height and width of the distance matrix, i.e. the total number of labels in the data set. A method called StratifiedKFold [111] is then used to split the indices for the kNN data sets. The method StratifiedKFold shuffles the N indices before the split, and stratification ensures that the percentage of each class label will be the same as in the full data set for the subsets of indices. The data sets used for

SVM classification was split using the method StratifiedShuffleSplit[111]. A more detailed explanation of the splitting for the kNN and SVM classifier follows below.

3.2 kNN classifier

A description of how the kNN classification is done follows next. The distance matrix D is split into a training/validation set with $N_{tr} + 1$ data points and a test set with N_{test} data points, a total of $N_{tr} + N_{test} + 1 = N$ data points. Both the imputed data set and the data set augmented with a combination of missingness features and imputation are classified using the kNN.

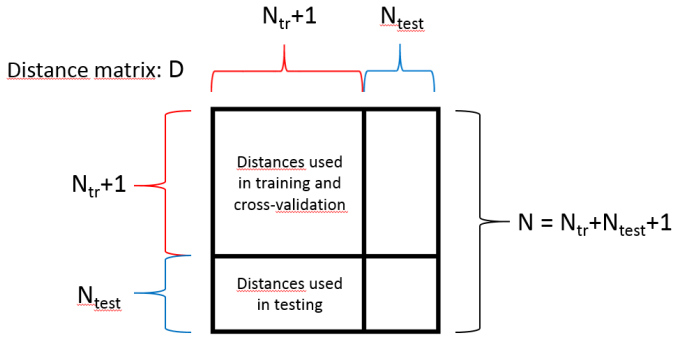


Figure 3.1: The proportion of distance matrix used for validation/training set and test set for the kNN classifier.

3.2.1 Validation and training set

After splitting the data set into one set for training and cross-validation and one set for testing, the distance matrix can be partitioned into block matrices, as illustrated by Figure 3.1. When validating the kNN model, we use the block matrix representing distances between data points in the training and validation set. Remember that the kNN model has a different nature than most machine learning algorithms and is not explicitly trained, since the model consists of the data points in the training set. We have chosen to use use leave-one-out cross-validation (LOOCV), where validation is carried out iteratively by extracting one data point at the time from the combined training and validation set, using this as a validation point, and the remaining data points as training set. The row of the block matrix that represents distances between the given validation data point and corresponding training set in a LOOCV iteration is used as input to the validation, since the k nearest neighbours are determined from these distances.

3.2.2 Test set

Nearest neighbor classification is then done on the test set. The value of k was set to be the one found to be best in the combined training and test set. The most common label among the k nearest neighbours is assigned to be the predicted label. The true test labels were extracted earlier. Finally, the confusion matrix for the best k was computed along with the accuracy, F_1-score, and balanced accuracy.

3.3 SVM classifier

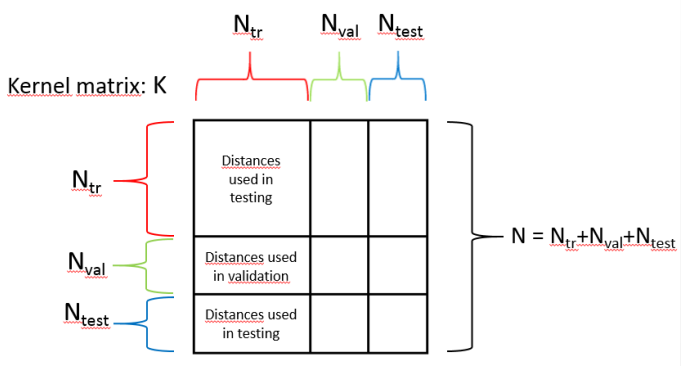


Figure 3.2: The proportion of the matrix D used for training, validation, and test kernel.

The SVM classifier is used on both imputed data sets and data sets augmented with a combination of imputation and missingness features. The full data set consist of N samples, for which a DTW distance matrix D has been pre-computed. The DTW distances for a given imputation method are sent into the Gaussian kernel function along with the γ (gamma) value for that method; gamma being the median DTW distance of the training set. This way we can pre-compute a kernel matrix K containing the kernel function for all pairs of data points. After splitting the data set into training, validation, and testing, the kernel matrix can be partitioned into block matrices in the same manner as the distance matrix was for the kNN algorithm, as illustrated by Figure 3.2. The true test labels y were also split accordingly, to be the same as each column height. When training the SVM model, we use the block matrix representing the kernel functions between data points in the training set. Then we use the off-diagonal block matrix that represents kernel functions between one training data point and one validation data point as input to the validation. Finally, the off-diagonal block matrix that represents kernel functions between one training data point and one test data point is used as input to the testing, which provides the final performance measures. A value for C to be used on the training set was found by doing a grid search on the validation set for 50 values by increasing C from 0.1 to 2.1 with logarithmic step size values followed by re-training the SVM with this C, and classification of the validation set. The result of each value of C was scored by computing the confusion matrix, precision, recall, specificity, balanced accuracy, and F_1-score. The scores of precision, recall, and specificity is left out in the plot in the result section as they are baked into balanced accuracy and F_1. The best value of C was then selected by using the F_1-score. This value of C was then used to classify the training set.

3.3.1 Finding γ

The DTW distances for the training set, test set, and validation set was written to a file, then a new program was written to read the distances into a histogram function and plotted to visually inspect the distribution of distances. The peak of the histogram was found to be close to the median of the DTW distances. A heuristic rule was created in the original program, where γ was set to be the median of DTW distances in the training set.

3.3.2 Finding C

In order to find the value for C that gives the highest score on the chosen metrics, we do a grid search on the validation set by testing several increasing values of C. This was done by re-training the training set while switching out the value of C for each increment, followed by prediction on the validation set. After having predicted for all values of C on the validation set, the C that gave the best score got chosen. If several values of C gave the maximum score, only the first one was chosen to be used on the test set.

3.3.3 Weighting

As the data can be heavily skewed toward one class with over-representation of it in the data set, it was necessary to do some weighting in order for the SVM algorithm to be effective. Otherwise, it would not know how to separate the two classes properly. The chosen method was to use sample weighting. Sample weighting is done by re-scaling C with the weight for that sample:

$$C[i] = C \cdot w_s[i] \tag{3.6}$$

where $C[i]$ is the re-scaled C and w_s[i] would be the sample weight for sample i. A higher value for w_s[i] implies more importance should be attributed to that sample. The weights were assigned using the heuristic:

$$w_s[i] = \begin{cases} \frac{N_{tr}}{N_1} & , \quad y[i] \text{ is } 1 \\ 1 & , \quad \text{otherwise} \end{cases} \tag{3.7}$$

where N_{tr} is the total number of labels in the training set, N_1 is the number of labels in the training set labeled 1, and $y[i]$ is the label of sample i.

3.4 TCN classifier

A description of how the TCN classifier was implemented is given. Due to the TCN being a somewhat new algorithm, which has not been used on the SSI data set earlier, a recaption of the architectural choices follows. The TCN classifier will use the augmented data set.

3.4.1 Trying out different architectures for SSI data

A simple TCN architecture was first attempted, using only the residual blocks, skip connections, dropout, class weighting (using the same weight heuristic that was used in the SVM classifier), and a single hidden layer of neurons between the residual blocks and the output, as a proof of concept. The data was augmented with missingness mask and interval counter. The Hinge loss [112] was used as objective function, because it was described to work well with maximising the margin between classes; It was later swapped with the well-known cross-entropy loss function. Normalisation was not used at this time, but was added at a later point. Varying the number of perceptrons did not seem to have much impact on the accuracy and validation set accuracy, though over-fitting could be seen.

A second layer of hidden neurons, activation, and dropout was therefore added before the output. This increased the loss and reduced the training set accuracy, but increased the validation accuracy, resulting in better generalisation. A problem with using the Hinge loss was then found, as varying the number of perceptrons in the hidden neuron layers then had little impact on the variance of the validation. Switching to cross-entropy helped.

This resulted in a new issue, namely that sometimes the gradient would vanish or explode. As mentioned in section 2.6.2, this was due to the activation function being linear, but layer normalization helped against this, although it increased the computation time.

After having tested out different types of layers, it was necessary to narrow down the number of neurons in the hidden layers before the output. This was done by varying the combination of hidden layer widths using a grid search, as shown in figure 3.3 and the tables in appendix B. This will be discussed in section 5.5.1. One can see that there are many possible and viable combinations of hidden neural layers. If more combinations of neurons were to be tested, then a randomized search [113, 114] could have been used, as this has the potential of finding the optimal combination.

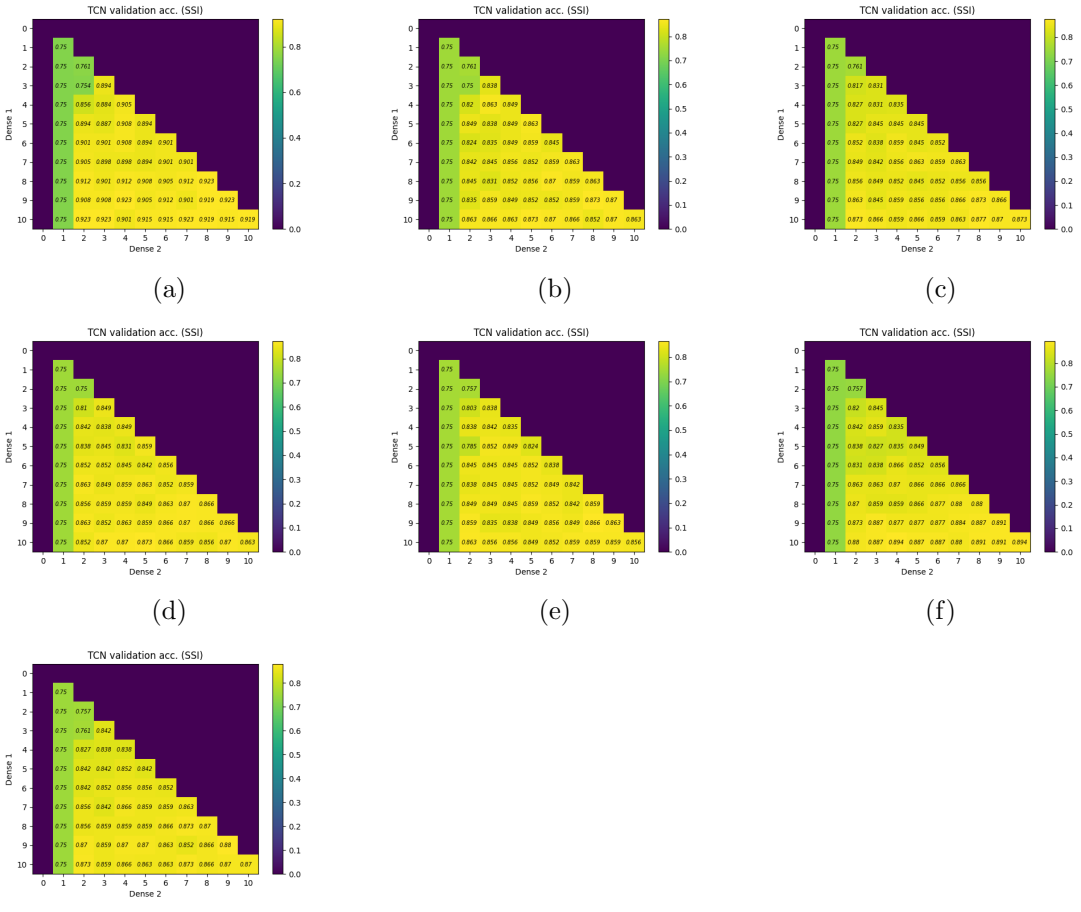


Figure 3.3: Median of the accuracies of the validation sets for combinations of different number of neurons in the two layers before the output for the TCN on the SSI data. From upper right to lower left is (a) zero-imputed, (b) locf-imputed, (c) mean-imputed, (d) median-imputed, (e) minimum-imputed, (f) maximum-imputed, and (g) combine-imputed data used. Vertical axis is the number of neurons in the first layer, and the horizontal axis is for the second layer. Each cell show the median after 20 runs with randomized sets without replacement and early stopping with maximum best epoch. Limit of tolerance for early stopping was set to 20 epochs. The numbers for the validation results can also be found in appendix B. The darkest areas are values which have been ignored.

3.4.2 Trying out different architectures for the uwave data with missingness

Using what worked for the SSI data, the same basic architecture setup was used for the uwave data with synthetic missing data. The number of residual blocks did not need to be changed, but the number of neurons in the dense layers still had to be validated. The validation for the uwave data was not as extensive as for the SSI data due to time constraints. The number of validation runs was reduced from 20 to 10, the patience for the early stopping (i.e. the maximum number of epochs executed in wait for a new minimum value of the loss function) was reduced from 20 to 10, and the number of neurons in each layer was checked for every even number of neurons. This reduced the computation time of the TCN validation process from about one week per imputation method to about 25-30 hours per imputation method. A figure showing an overview of the validation results for the TCN on the uwave data can be seen in figure 3.4. The values of each cell can be found in appendix B.

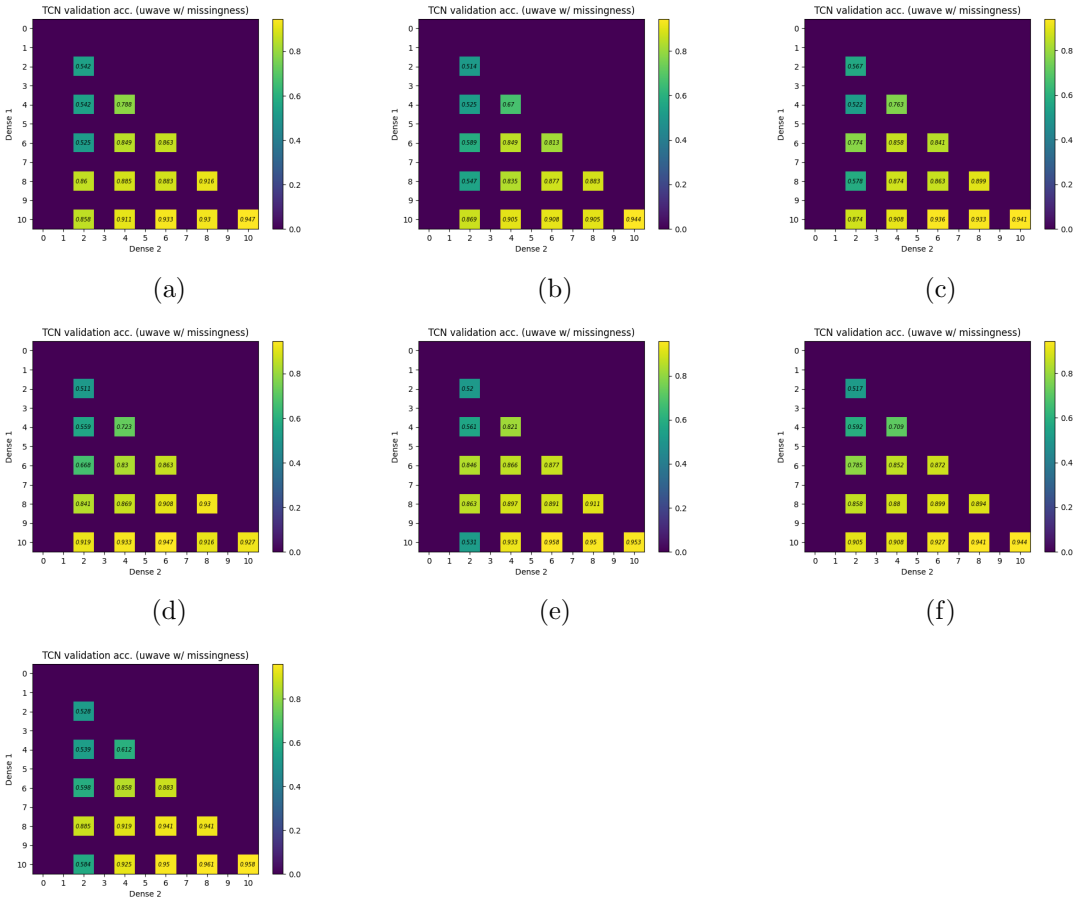


Figure 3.4: Median of the accuracies of the validation sets for combinations of different number of neurons in the two layers before the output for the TCN on the uwave data with synthetic missingness. From upper right to lower left is (a) zero-imputed, (b) locf-imputed, (c) mean-imputed, (d) median-imputed, (e) minimum-imputed, (f) maximum-imputed, and (g) combine-imputed data used. Vertical axis is the number of neurons in the first layer, and the horizontal axis is for the second layer. Each cell show the median after 20 runs with randomized sets without replacement and early stopping with maximum best epoch. Limit of tolerance for early stopping was set to 20 epochs. The numbers for the validation results can also be found in appendix B. The darkest areas are values which have been ignored.

Chapter 4

Experiments

4.1 Equipment and language

The experiments are carried out an a laptop with processor: AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx 2.10 GHz. The laptop runs a 64-bit Windows operating system, with an x64-based processor. Python version 3.8.1 has been used for general programming, along with Keras 2.4.3 for deep learning.

4.2 The experiments

The experiments were done using a distance matrix of DTW and ED distances after having used imputation over one dimension (one type of blood test) at a time. The imputation methods were zero, locf, mean, median, minimum, maximum, and a combined imputation.

4.3 kNN parameters

SSI data

Validation and test sets were made as described earlier. The validation lists of indices are of length 442 and the lists of test indices of length 441. The kNN classifier was used with odd values of k ranging from 1 to 21 . It was applied to the full data set with all imputation methods and for both normal feature vectors and the augmented feature set.

Synthetic missing data

Same procedure as with the SSI data, but with a different length of validation lists with length 560 for the training and validation indices, and 559 used for testing on both the normal data and the augmented data.

4.4 SVM parameters

SSI data

The data set was split into training set with data from 618 patients, from which the DTW distances and the kernel functions that go into in a kernel matrix of shape (618×618) are computed; a validation set with data from 133 patients, from which a kernel block matrix of shape (133×618) is computed; and a test set with data from of 132 patients, from

which another kernel block matrix with shape (132×618) is computed. A suitable gamma was selected using the median value of the DTW distances between patients for the given imputation method in the training set. Best value of C was found using was found using grid search on the validation set. Used F_1-score to select a C value. Additionally a few stop criteria was set: the maximum of iterations the SVM could try to converge were 10000, and a minimum tolerance of 1E-3 difference was used for the validation set and the test set had a minimum tolerance of 1E-6 difference between each iteration.

Synthetic missing data

The same procedure was used for the uwave data with synthtic missing data, with an added pre-processing step in order to counter a glitch with the particular data set that was discovered during preliminary experiments. When splitting the distance matrix and the corresponding labels using the StratifiedShuffleSplit method, a synchronised shuffling of the indice labels and the distance matrix values had to be made beforehand. The training set indices was of length 783 resulting in a kernel of size (783×783), with a validation set with 168 label indices giving a validation set of shape (168×783). The length and shape of the test set was the same as the validation set.

4.4.1 TCN parameters

SSI data

The data was augmented and divided into training data and test data by a 80/20 percentage split. The training data was split further, where 20% of the training set (16% of the full data set) was used for validation by the Keras network training method. The learning rate of the optimizer was set to 0.0001 and was assumed to be sufficient but did not have time to validate. Dilations used was 1, 2, 4, 8, 16, and 32, in that order. Dropout rate was set to 0.1 for all layers. Class weights was set equal to 1 for class 0 (non-SSI), and $N_{tr}/N_{y=1}$ where $N_{y=1}$ are the number of training labels marked class 1 and N_{tr} are the total number of training labels. The loss function used was the binary cross-entropy function. Early stopping parameters were set to a minimum Δ (required improvement of loss function) of 0.01 and a patience (how many epochs to wait) of 10, with the maximum number of epochs set to 250. Both validation loss and validation accuracy was monitored. The early stopping mechanism stores the weights for each epoch, and if the algorithm stops early because the validation fail to improve, the best weights stored by the early stopping mechanism is used. The number of neurons was determined through validation. The complete process was run 10 times for each imputation method.

Uwave with missing data

The same parameters used for the SSI data was used on this data set, with one difference being that the labels and data had to be synchronously shuffled before splitting the data, same as with the SVM.

Chapter 5

Results and discussion

Histograms of the DTW distances are shown below in 5.2.1 and Euclidean distances in 5.2.2 for SSI data. The same kind of DTW distance and Euclidean distance histograms are shown in section 5.6.1 and 5.6.2, respectively, for the uwave dataset. All these plots are shown to explain and justify the value of γ used by the SVM classifier and to capture the difference between the imputation methods, as experienced by the distance-based classifiers.

Informative missingness is discussed in section 5.1. The differences between the histograms of DTW and Euclidean distance is shown in 5.2.3 for the SSI data and in section 5.6.3 for the uwave data. Then in 5.3 and 5.7 the kNN score metrics for the test set is shown for the SSI and uwave data, respectively, along with plots of the F_1 scores for their validation sets. Thereafter, in 5.4 (SSI) and 5.8 (uwave data), are tables for the SVM classifier shown followed by plots showing balanced accuracy and F_1 score below. What is designated as non-SSI has been assigned with the label 0, and SSI has been labeled as 1. Each validation run is independent from each other and starts with training from start.

5.1 SSI missing data

As mentioned in section 2.3, missing data can be informative; lacking data that may be MAR, MNAR, or MCAR, or a mix of MAR and MCAR. Some of this informative missingness can be seen in figure 2.2 which shows an image matrix for the data for each blood test type, a bar plot with Pearson correlation between missing data and incidence of SSI, and the missing rate. The data is as mentioned earlier done before gastrointestinal surgery, which is a collective term used for surgery on the esophagus, pancreas, liver, gallbladder, and the rest of the digestive system [115].

One can see in figure 2.2 that the missing rate of all the blood tests increase from start to end with some variation. Carbomide (c), creatinine (d), glucose (f), and thrombocytes (k) have the most data missing, which can be seen in both the data image matrix and the bar plot with missing rates. Overall, taking blood tests on day 0 and 2 seem common for 6 of the 11 blood tests: albumin (a), CRP (e), glucose (f), hemoglobin (g), potassium (i), and sodium (j). CRP increases on day 1 on a number of patients, and leukocytes increase on day 7 for several. The amount of thrombocytes seem to increase as time goes on for several patients, but there is no obvious common start day for when the increase happen.

One can also see that there is a weak positive correlation between infection and data missing on day 0 for amylase and hemoglobin. This is speculation from the author, but a possible cause for not taking blood tests on day 0 of amylase and hemoglobin could be due to pancreatitis with complications, which is an infection of the pancreas resulting

from digestive enzymes consuming the pancreas' tissue. A common cause of pancreatitis is alcoholism[116]. Alcohol is known to dehydrate the body, which in turn cause hemoglobin levels to decrease due to reduced plasma volume[117], while the infection in the pancreas increase the amylase levels. Common treatments for acute versions of this condition is to get hydrated with intravenous fluids and undergo fasting while resting. To get more accurate test results, a medical practitioner who knows the condition will probably wait to test hemoglobin and amylase levels. As these missing data possibly infer alcoholism, which is a sensitive topic, they may also affect a machine learning algorithm with access to health records used in areas such as social services, insurance cases, court cases, health treatment, and inadvertently cause discrimination.

5.2 Comparing distance measures

The kNN and SVM classifiers are distance-based, specifically using either DTW or Euclidean distances, and different imputation methods along with whether the data is augmented or not affect the range and distribution of the distance. It is therefore of interest to study histograms of the distance matrices used.

The distances are in the SVM classifier input to a Gaussian kernel, similar to the RBF kernel. We have parametrized the Gaussian kernel with a parameter γ, which can be related to the scale parameter σ as $\gamma = 1/\sigma^2$. The scale parameter σ can be interpreted as a typical or characteristic distance, and is chosen such that distances equal to σ or smaller are translated by the Gaussian kernel to similarities from $1/e$ up to 1. This means we should inspect the distances with this in mind; specifically: what is a typical distance value and can we by visual inspection find a way to relate it to any statistics of the distribution, such as the mean, median or mode value, such that σ (and γ) can be determined automatically? For instance, we may inspect figures 5.1 and 5.3. A useful statistic in this case could be the median value of the distances, and thus with the assumption that the median is a typical or characteristic value for the distance between time series from the same class, a heuristic rule was made to set the kernel parameter γ (gamma) of the Gaussian kernel used by the SVM classifier. The medians of each imputation method was also plotted in figures 5.1, 5.3, 5.2, and 5.4 for the SSI data to visually confirm that it looks like a typical value that could be used as a scale parameter σ to set the kernel parameter $\gamma = 1/\sigma^2$ in the Gaussian kernel. The statistics mentioned in this section can also be found in table 5.1 for both SSI data and the uwave data with synthetic missing data. Common to all the histograms is that they show a trail-off with very small counts of high values along the horizontal axis indicating the distances between some outliers, and the blue curve (distance between 0 and 0) is largest, with the orange curve (distance between 0 and 1) second largest, and the green curve (distance between 1 and 1) being the smallest. The SVM classifier use the same median shown in the histograms to find $\gamma = 1/\sigma^2$ and separate the blue and green curve, where σ is set to be the median.

5.2.1 DTW distance histograms of the SSI data

The histogram of the DTW distances looks quite different when comparing the zero-imputed data in figure 5.1a with the other imputation methods in figure 5.1. The histogram of the zero-imputed data has a sharp rise from distance 0 to around 5, then slopes down again before descending slowly and ending near distance 25. Around distance 12 there is a small rising peak in the slope, about the same distance where the other imputation methods have a second peak and their median. This small peak comes from the difference

Table 5.1: Median values from the histograms to compare the DTW and Euclidean distances on the imputed data, respectively $\mathbf{D}_{DTW}$ and $\mathbf{D}_{Euclidean}$ and their augmented versions. The columns are the different imputation methods. The total number of values in each distance matrix $\mathbf{D}$ is $883 \cdot 883 = 779,689$ for the SSI data, and $1119 \cdot 1119 = 1,252,161$ for the uwave data.

	zero	locf	mean	median	min	max	combination
SSI data:							
Median, $\mathbf{D}_{DTW}$	6.138	15.502	14.979	15.014	14.159	16.968	6.187
Median, $\mathbf{D}_{DTW\ aug.}$	33.480	34.596	32.555	32.596	32.539	36.612	32.228
Median, $\mathbf{D}_{Euclidean}$	7.475	8.813	7.451	7.474	7.274	8.40	6.968
Median, $\mathbf{D}_{Euclidean\ aug.}$	34.782	36.642	35.178	35.218	34.944	35.387	34.552
Synth. missing data:							
Median, $\mathbf{D}_{DTW}$	59.912	65.706	62.044	64.147	78.617	80.271	60.382
Median, $\mathbf{D}_{DTW\ aug.}$	47.500	49.032	48.287	64.147	53.444	51.401	47.709
Median, $\mathbf{D}_{Euclidean}$	59.912	65.707	62.045	64.147	78.619	80.271	60.382
Median, $\mathbf{D}_{Euclidean\ aug.}$	49.169	50.728	49.947	50.703	54.948	53.033	49.392

between class 1 (SSI) and class 0 (non-SSI), which can be seen from the orange line layered on top of the histogram. The LOCF imputation histogram shows two peaks, one main peak and another lower but sharper peak at DTW=10, while mean, median, minimum and maximum imputation show a steep peak around DTW=5 with a small bump halfway down the histogram slope around DTW=10. The author does not have an explanation for why there is a small bump along the sides of the non-zero-imputed distance histograms other than that it is a property of the dissimilarity from class 0 to every other class 0, as can be seen by the blue line in the figure 5.1 (b),(c), (d), (e), (f), and 5.5b.

The zero-imputed distances have a median around 7. I.e. most DTW distances of the zero-imputed data lie around this distance. The other imputation methods have much the same median, with LOCF imputation having the highest value. It is expected that the median of the distances shift, as substituting the missing data with non-zero imputation values shifts the distances. Overall the imputation methods have quite similar values of γ.

Other differences between the histogram for the DTW distances of zero-imputed data and the other histograms are also present. The small green curve representing the difference between class 1 and 1 is for the zero-imputation at distance 12, higher than the median, but shifts down to around the same distance as where the median lie at around distance 6.

The first and second "peak" is of about the same size for mean, median, and minimum imputation, but for maximum imputation the second peak lay quite higher due to a higher peak from the contribution of the orange line representing the difference between class 0 and 1. The combined imputation in figure 5.5b looks like a combination of the zero-imputed and the other methods, but the spread of the distances has been compressed and shifted, making a high peak near the median with a small "platform" protruding on the right side.

When it comes to the DTW distances of the augmented data in figure 5.2, the histograms look somewhat different compared to the non-augmented data in figure 5.1. The distances are more spread out and jagged with distances ranging up to 80. In the zero-imputed data in figure 5.2a there is a small top around 10, before a small decline between distance 20 and 30. Around distance 30 to 55 there is some plateauing with some jagged

bumps, before descending steadily and approximately linear. The median value is about at the start of the plateau. The other imputation methods are a bit more steep and smooth. The augmented LOCF-imputed data have two defined peaks at 20 and 40 with some smaller and lesser defined ones at its slopes with a small plateau around 60 before sloping down again with a long trail ending at around 100.

The other imputation methods look very much like the LOCF imputation histogram, but with a slightly more flat single top. Looking at the orange and blue curves they are a bit more blunt for the non-zero non-LOCF imputation methods, which can explain the other histograms not having a second peak. The combined imputation in figure 5.5d looks mostly like the histograms for the LOCF, mean, median, minimum, and maximum-imputed data. Different from the non-augmented data, the green curve representing the difference between class 1 and 1 stay fairly consistent at the same distances for all imputation methods.

5.2.2 Euclidean distance histograms of the SSI data

The histograms of the Euclidean distances can be found in figure 5.3, while the histograms for the augmented data of the same distance type can be found in figure 5.4. Similar to the zero-imputed data in the histogram for DTW distances, the zero-imputed Euclidean distances look jagged. First it starts with a jagged slope/plateau, then steeply inclines with a top close to distance 5, which is slightly off from the median, which was around 7, where a small plateau starts before the histogram descends slightly and rises to a small top around 11-12 before descending again. Again, the other imputation methods give more smoother histograms. The LOCF-imputed data gave a more bell-shaped form with its first peak near the median at approximate 8, with a second semi-peak or plateau around distance 11-12. The rest of the imputation methods gave histograms with the same basic shape as the LOCF-imputed data, but more steep and a slightly longer trail. The maximum imputation histogram have more of a bulge than a plateau. Looking at the blue line representing the distances between class 0 (non-SSI) and class 0 one can see that at the small second peak is a result of these differences internally between the non-SSI patients for the non-zero-imputed data. The orange line representing the difference between class 1 and 0 gets a peak close to the median and a platform which transition into a slope at distance 15. The green slope representing the distances between class 1 (SSI) and 1 is the smallest curve which for the zero-imputed data has a peak around distance 14, but shifts position down to about distance 6 for the other imputation methods.

Looking at the histogram of the augmented data histograms of the Euclidean distances one can see that the shape of the histogram of the zero-imputed data look quite similar to the DTW distances of the augmented data, though the maximum Euclidean distances are slightly larger than DTW distances for the zero-imputation. The other histograms are more bell shaped, but the LOCF data have more of a plateau than a peak, and the sides can be considered more "wavy". The biggest difference is that the distances are larger with the maximums being around 90, whereas for the non-augmented data had a maximum distance around 30. The green curve representing the distance between class 1 (SSI) and 1 stays about the same for each of the imputation methods with small differences, i.e. the curve for the zero-imputed data have a peak at around distance 50, while the peak is closer to 40 for the other imputation methods.

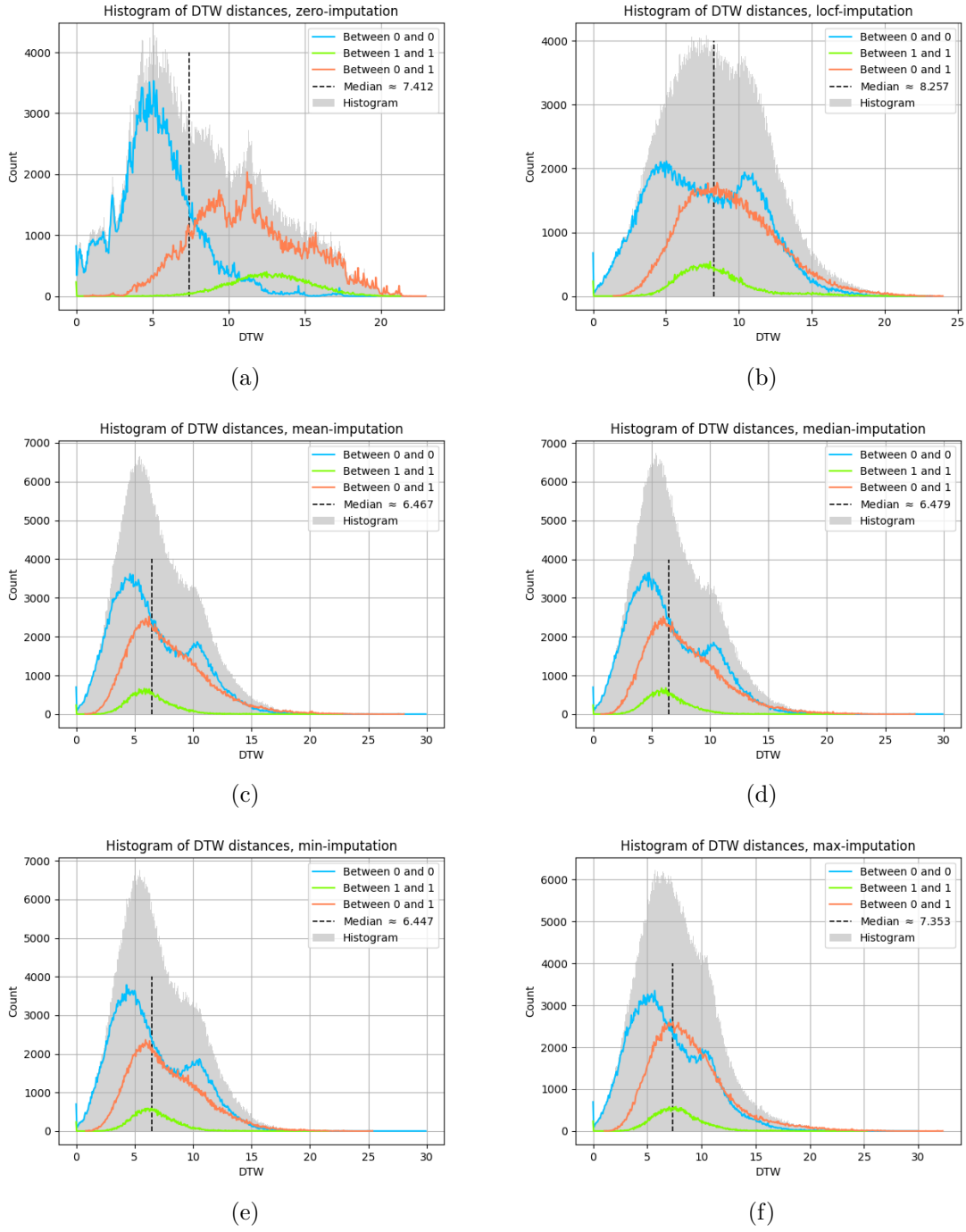


Figure 5.1: Histograms showing the DTW distances for different imputation methods, using Euclidean distance as a cost function on the SSI data set. (a) is zero-imputation with a median distance of 6.138, (b) is LOCF-imputation with a median distance of 15.502, (c) is mean-imputation with a median distance of 14.979, (d) is median-imputation with a median distance of 15.014, (e) is minimum-imputation with a median distance of 14.159, and (f) is maximum-imputation with a median distance of 16.968. The histograms show for all patient data, with the blue line showing the contribution from non-SSI patients, the green line the contribution from SSI patients, and the orange line the contribution from the difference between the two classes. Dashed line on histogram shows where on the horizontal DTW distance axis the median is. The histogram consist of $883 \cdot 883 = 779,689$ distance values distributed into 500 bins.

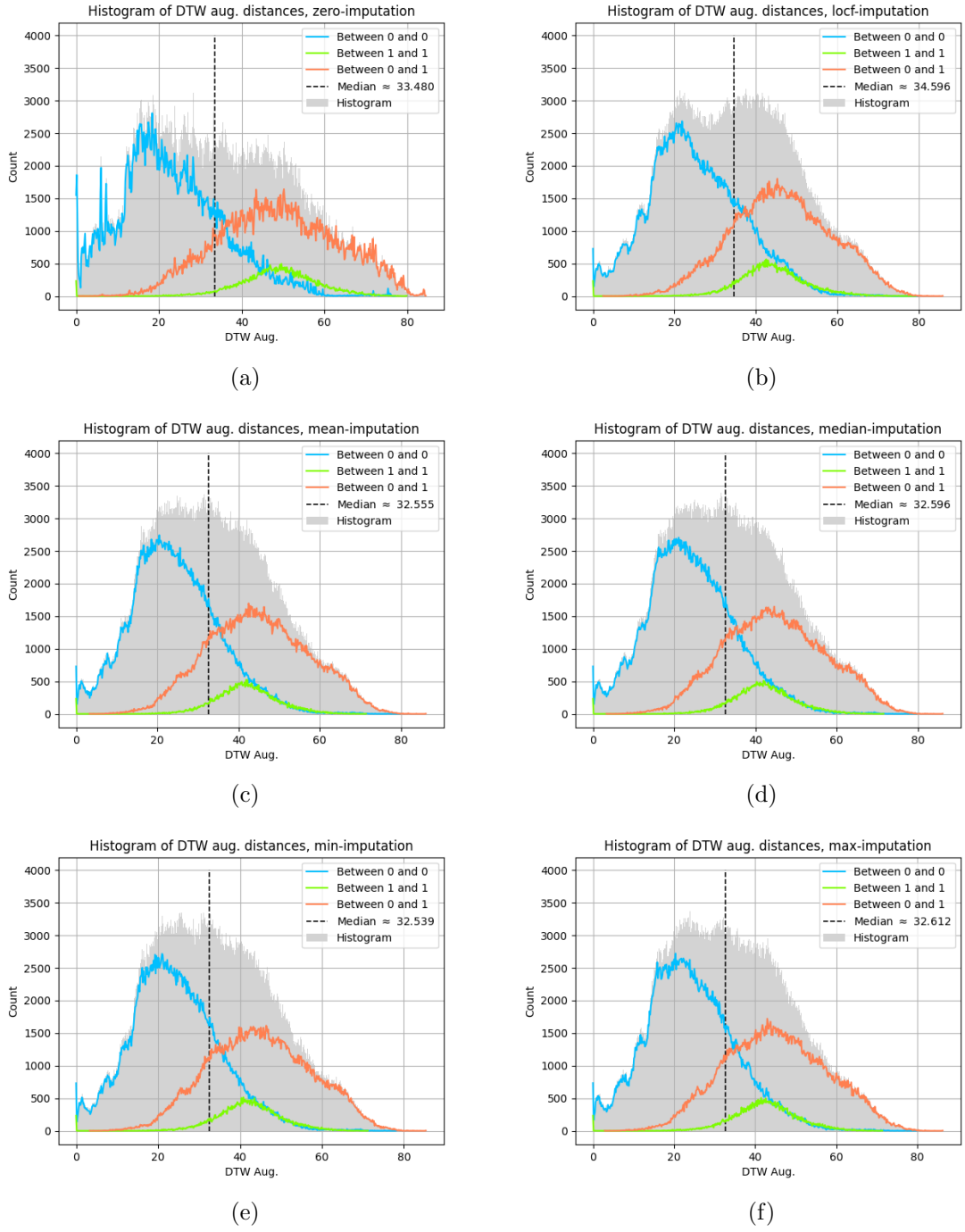


Figure 5.2: Histograms showing the DTW distances for different imputation methods using the augmented data on the SSI data set. The sub-figure (a) is zero-imputation with a median distance of 33.480, (b) is LOCF-imputation with a median distance of 34.596, (c) is mean-imputation with a median distance of 32.555, (d) is median-imputation with a median distance of 32.596, (e) is minimum-imputation with a median distance of 32.539, and (f) is maximum-imputation with a median distance of 32.612. The histograms show for all patient data, with the blue line showing the contribution from non-SSI patients, the green line the contribution from SSI patients, and the orange line the contribution from the difference between the two classes. Dashed line on histogram shows where on the horizontal DTW distance axis the median is. The histogram consist of $883 \cdot 883 = 779,689$ distance values distributed into 500 bins.

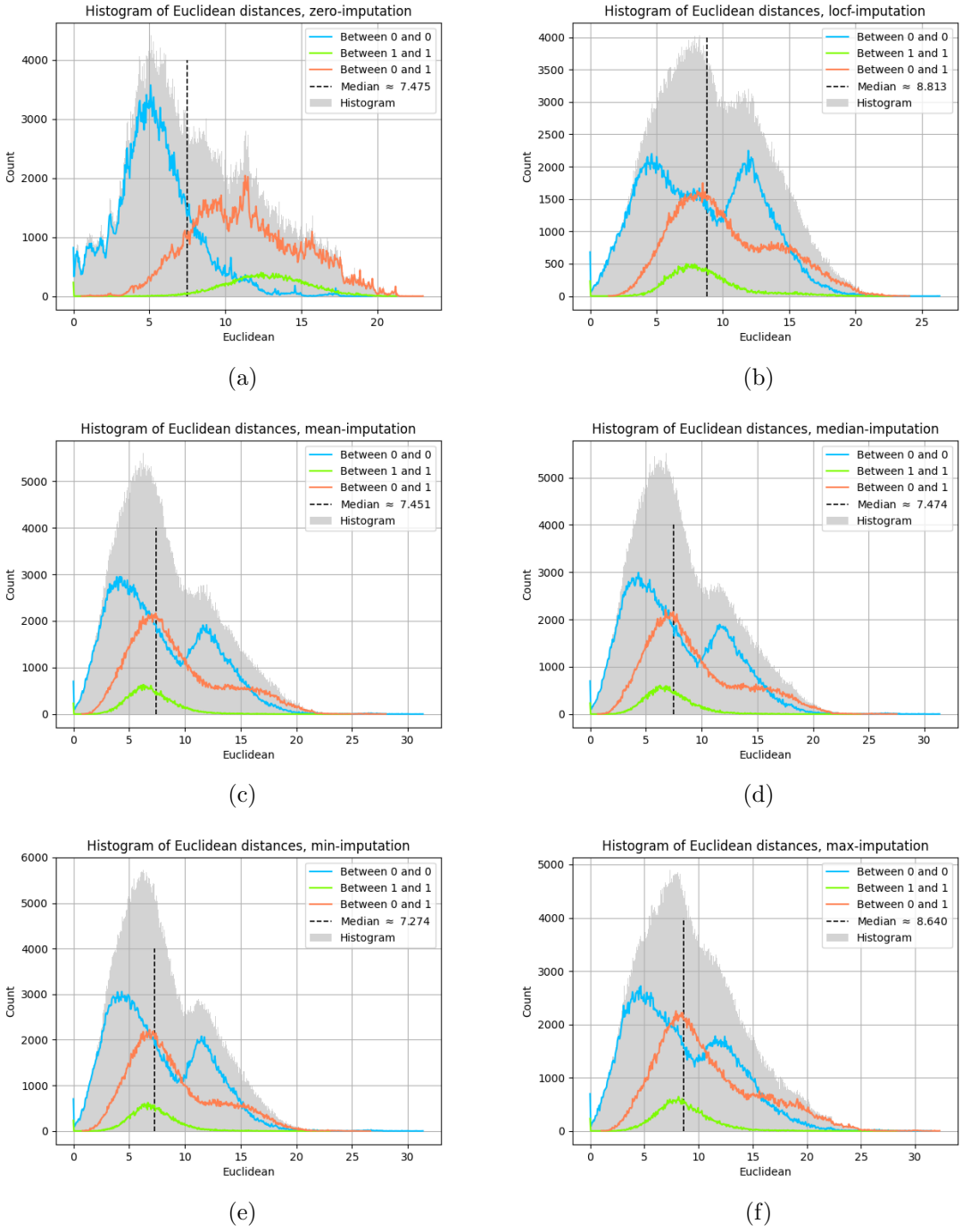


Figure 5.3: Histograms showing the Euclidean distances for different imputation methods on the SSI data set. The sub-figure (a) is zero-imputation with a median distance of 7.475, (b) is LOCF-imputation with a median distance of 8.813, (c) is mean-imputation with a median distance of 7.451, (d) is median-imputation with a median distance of 7.474, (e) is minimum-imputation with a median distance of 7.274, and (f) is maximum-imputation with a median distance of 8.640. The histograms show for all patient data, with the blue line showing the contribution from non-SSI patients, the green line the contribution from SSI patients, and the orange line the contribution from the difference between the two classes. Dashed line on histogram shows where on the DTW distance axis the median is. The histogram consist of $883 \cdot 883 = 779,689$ distance values distributed into 500 bins.

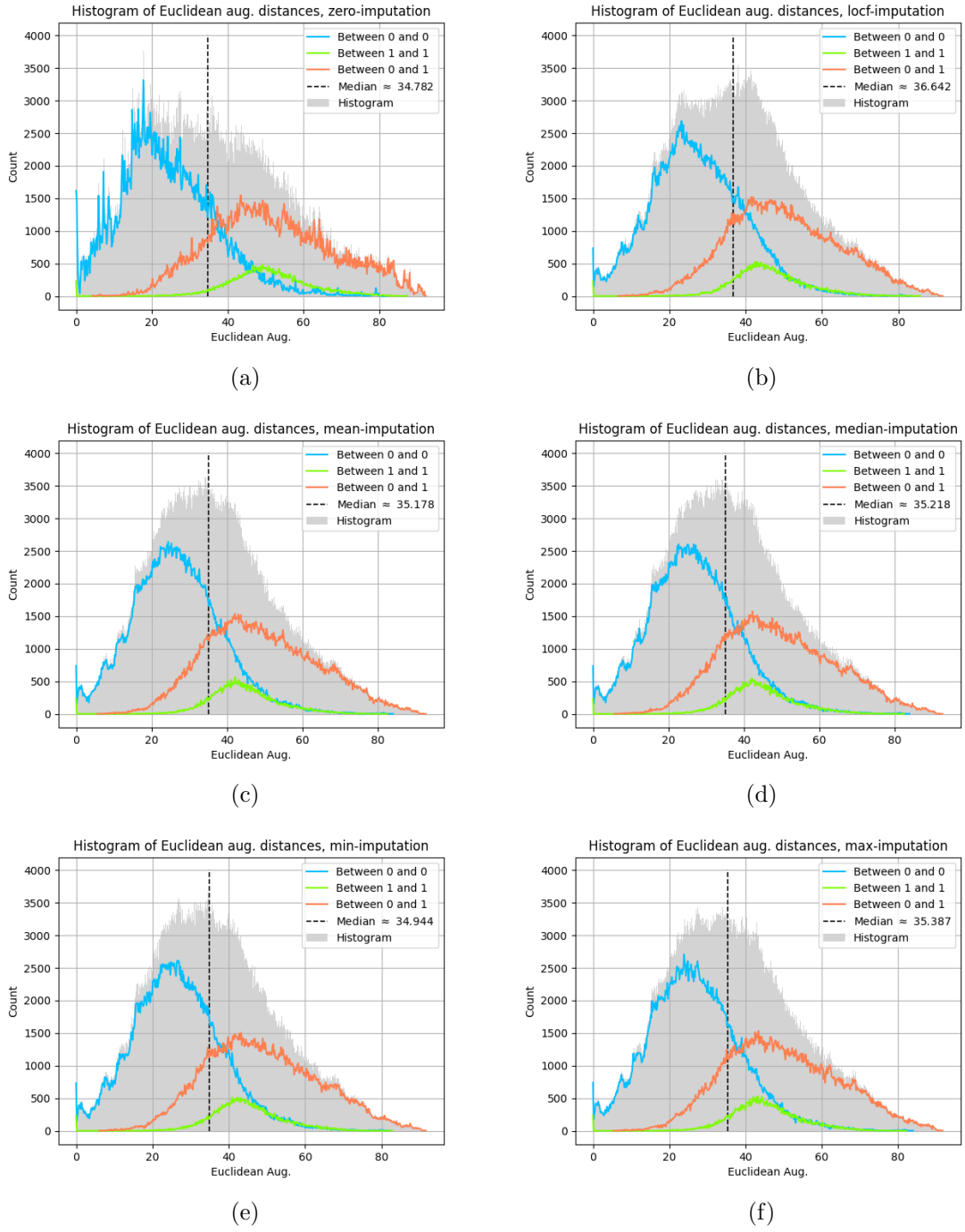


Figure 5.4: Histograms showing the Euclidean distances for different imputation methods using the augmented data on the SSI data set. The sub-figure (a) is zero-imputation with a median distance of 34.782, (b) is LOCF-imputation with a median distance of 36.642, (c) is mean-imputation with a median distance of 35.178, (d) is median-imputation with a median distance of 35.218, (e) is minimum-imputation with a median distance of 34.944, and (f) is maximum-imputation with a median distance of 35.387. The histograms show for all patient data, with the blue line showing the contribution from non-SSI patients, the green line the contribution from SSI patients, and the orange line the contribution from the difference between the two classes.. Dashed line on histogram shows where on the DTW distance axis the median is. The histogram consist of $883 \cdot 883 = 779,689$ distance values distributed into 500 bins.

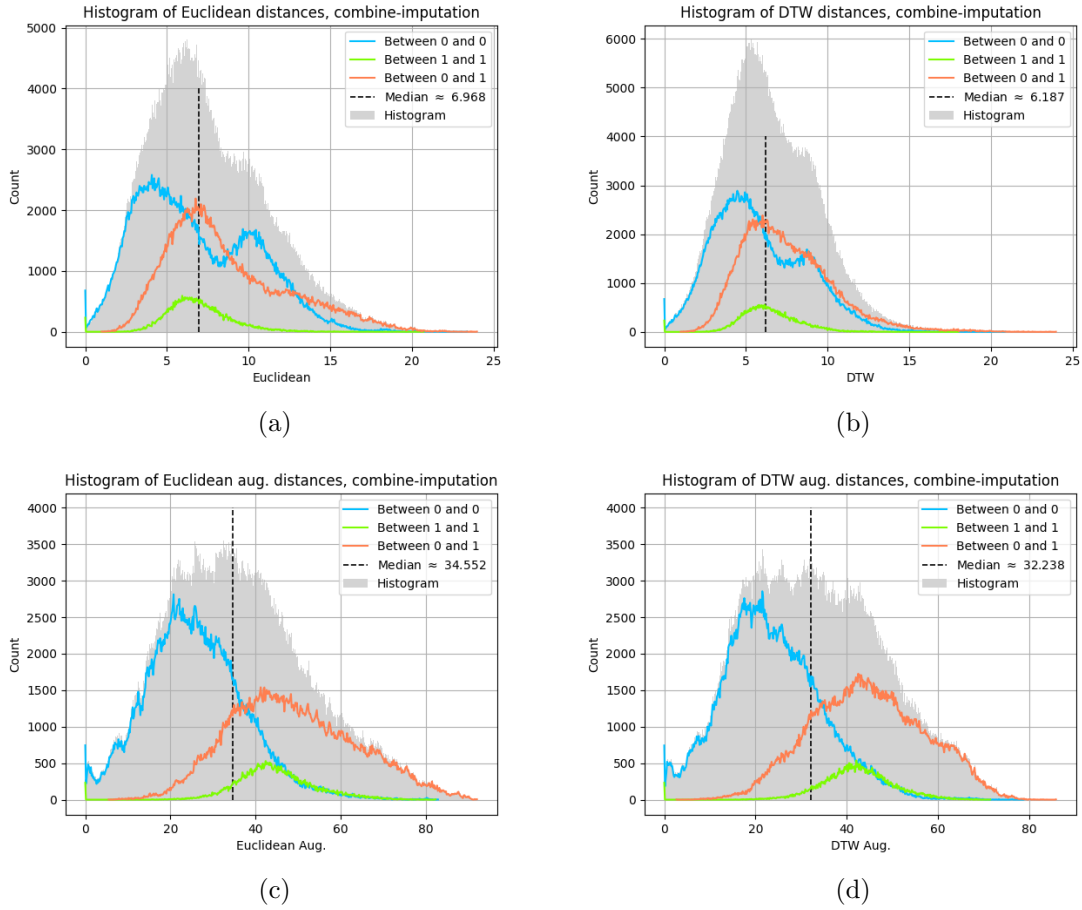


Figure 5.5: Histograms showing the Euclidean and DTW distances for the combined imputation using either the normal data or augmented data on the SSI data set. The subfigure (a) is Euclidean distance with normal data having a median distance of 6.968, (b) is DTW on normal data with a median distance of 6.187, (c) is Euclidean distance on the augmented data with a median distance of 34.552, and (d) is DTW used on the augmented data with a median of 32.228. The histograms show for all patient data, with the blue line showing the contribution from non-SSI patients, the green line the contribution from SSI patients, and the orange line the contribution from the difference between the two classes. Dashed line on histogram shows where on the DTW distance axis the median is. The histogram consist of $883 \cdot 883 = 779,689$ distance values distributed into 500 bins.

5.2.3 Some differences when comparing the distances of the SSI data

Figure 5.6, 5.8, 5.7, 5.9, and 5.10 show the scatter plots where each distance between two patients in the SSI data set is plotted against another type of distance from their respective distance matrices after imputation. Each distance matrix contain $883 \cdot 883 = 779,689$ values. If the distance measures produced the same distance, then the scatter points should appear to make a line. If one looks at figure 5.10, which is for the combined imputation, one can see the main shapes for each of the distance type comparisons. One can see that there are large differences between some of the distance types, though the DTW and Euclidean distances both for the original data and the augmented data in some cases remain fairly the same, in particular for the uwave data with missing values. The

biggest difference is between the same type of distance measure when comparing non-augmented with augmented data.

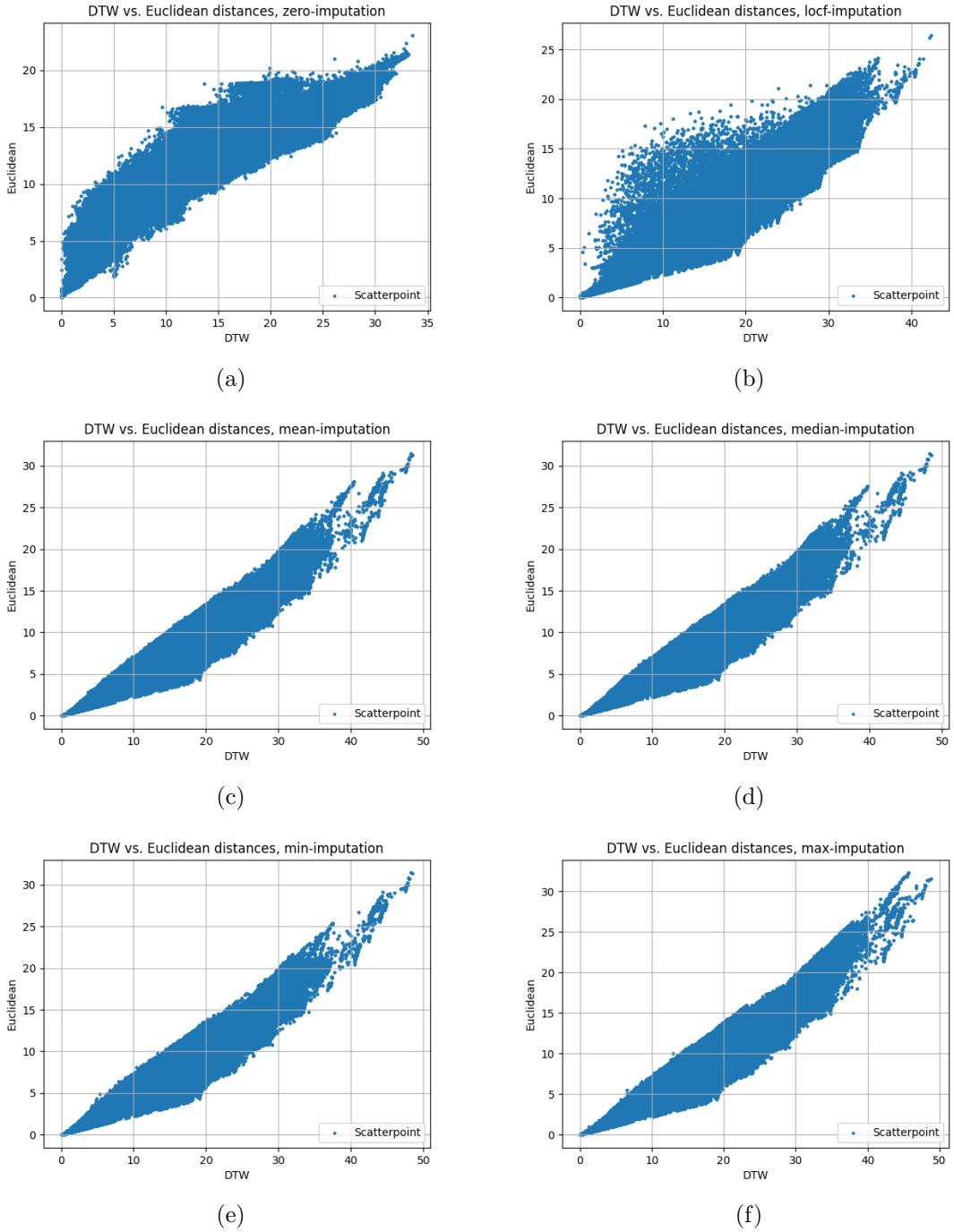


Figure 5.6: Scatter-plots comparing one-by-one DTW and Euclidean distances for different imputation methods on the SSI data set. The sub-figure (a) is zero imputation, (b) is LOCF imputation, (c) is mean imputation, (d) is median imputation, (e) is minimum imputation, and (f) is maximum imputation. Each plot contains 779,689 points.

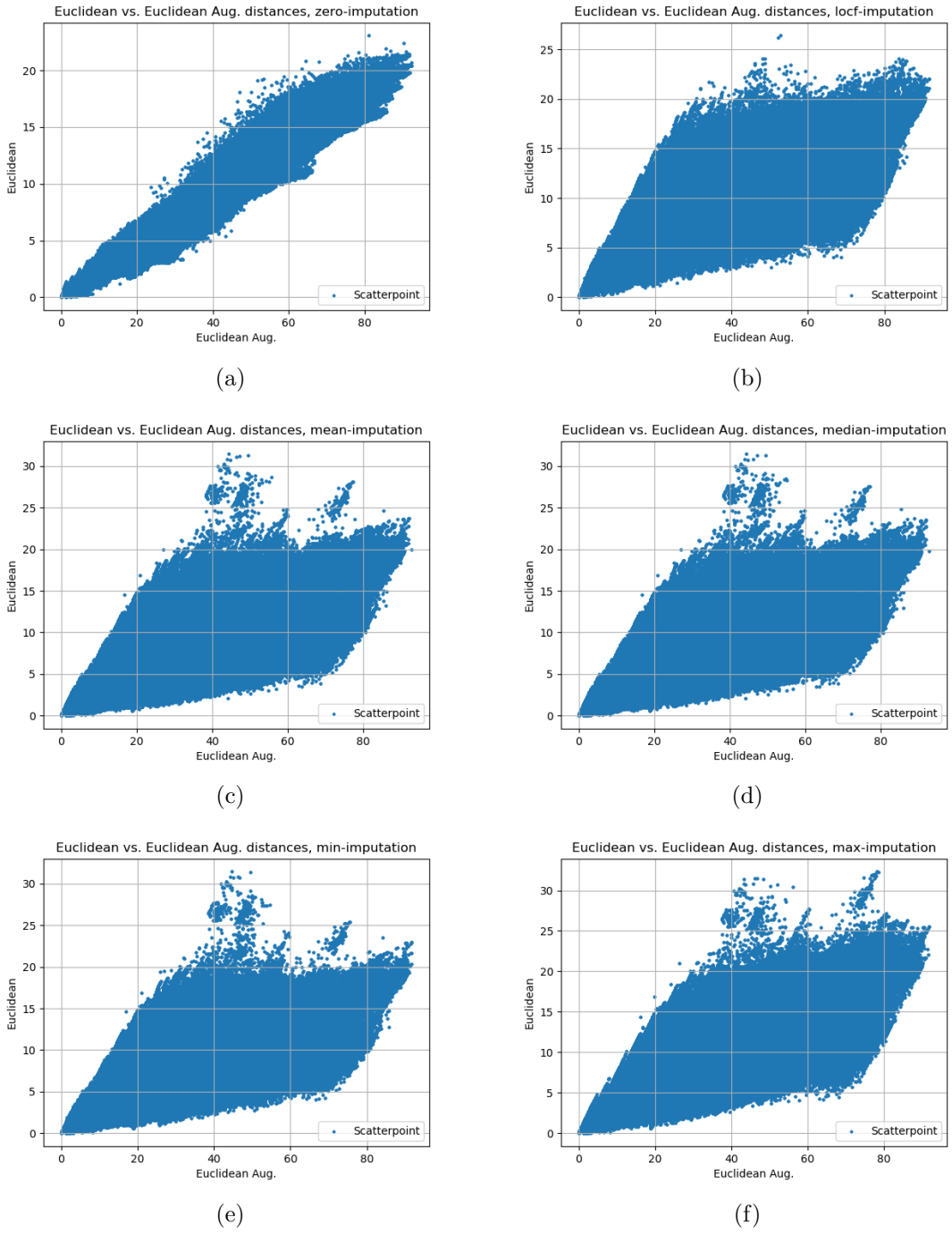


Figure 5.7: Scatter-plots comparing one-by-one Euclidean distances of the original with the augmented data Euclidean distances for different imputation methods on the SSI data set. The sub-figure (a) is zero imputation, (b) is LOCF imputation, (c) is mean imputation, (d) is median imputation, (e) is minimum imputation, and (f) is maximum imputation. Each plot contains 779,689 points.

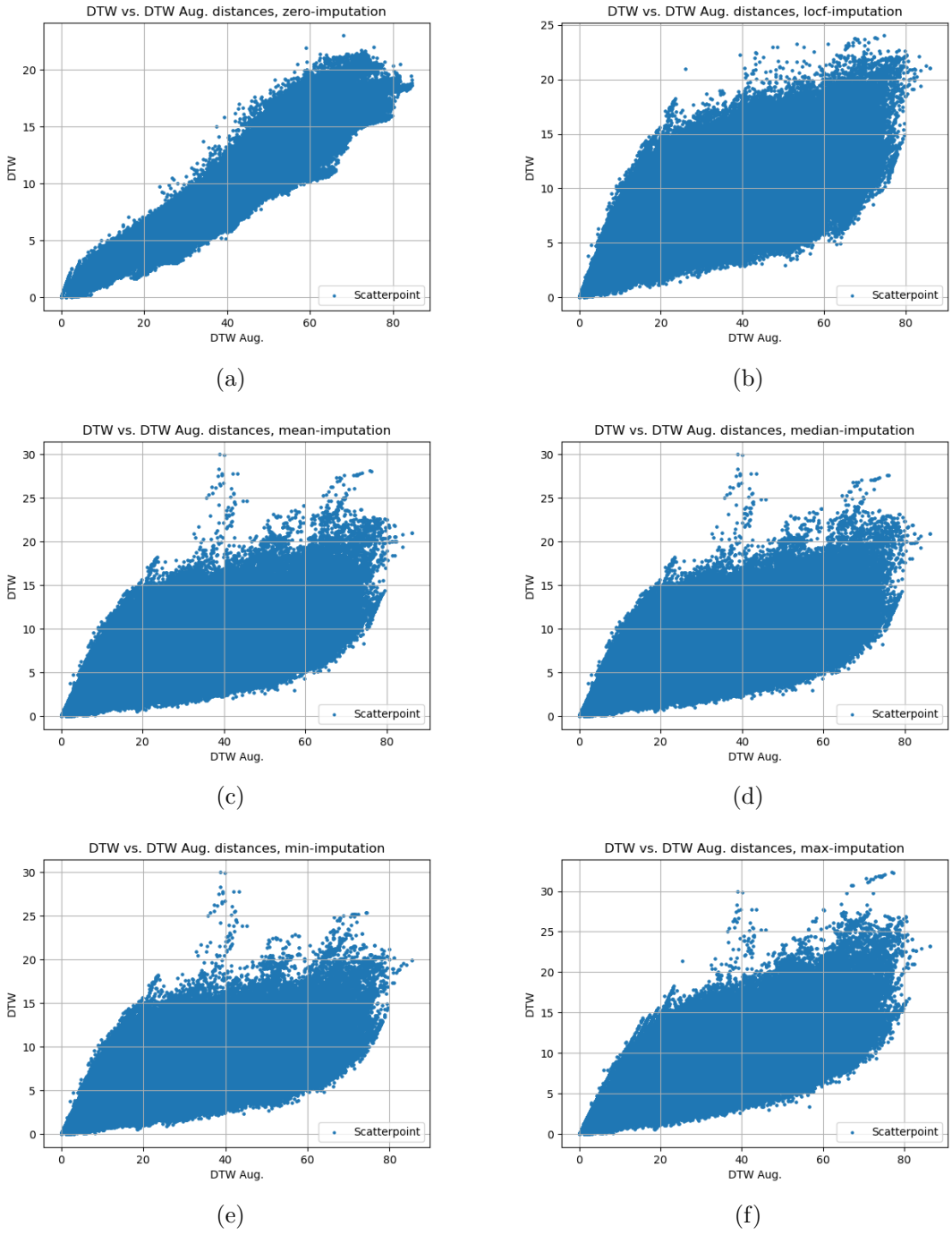


Figure 5.8: Scatter-plots comparing one-by-one DTW for different imputation methods using the normal data and the augmented data from the SSI data set. The sub-figure (a) is zero imputation, (b) is LOCF imputation, (c) is mean imputation, (d) is median imputation, (e) is minimum imputation, and (f) is maximum imputation. Each plot contains 779,689 points.

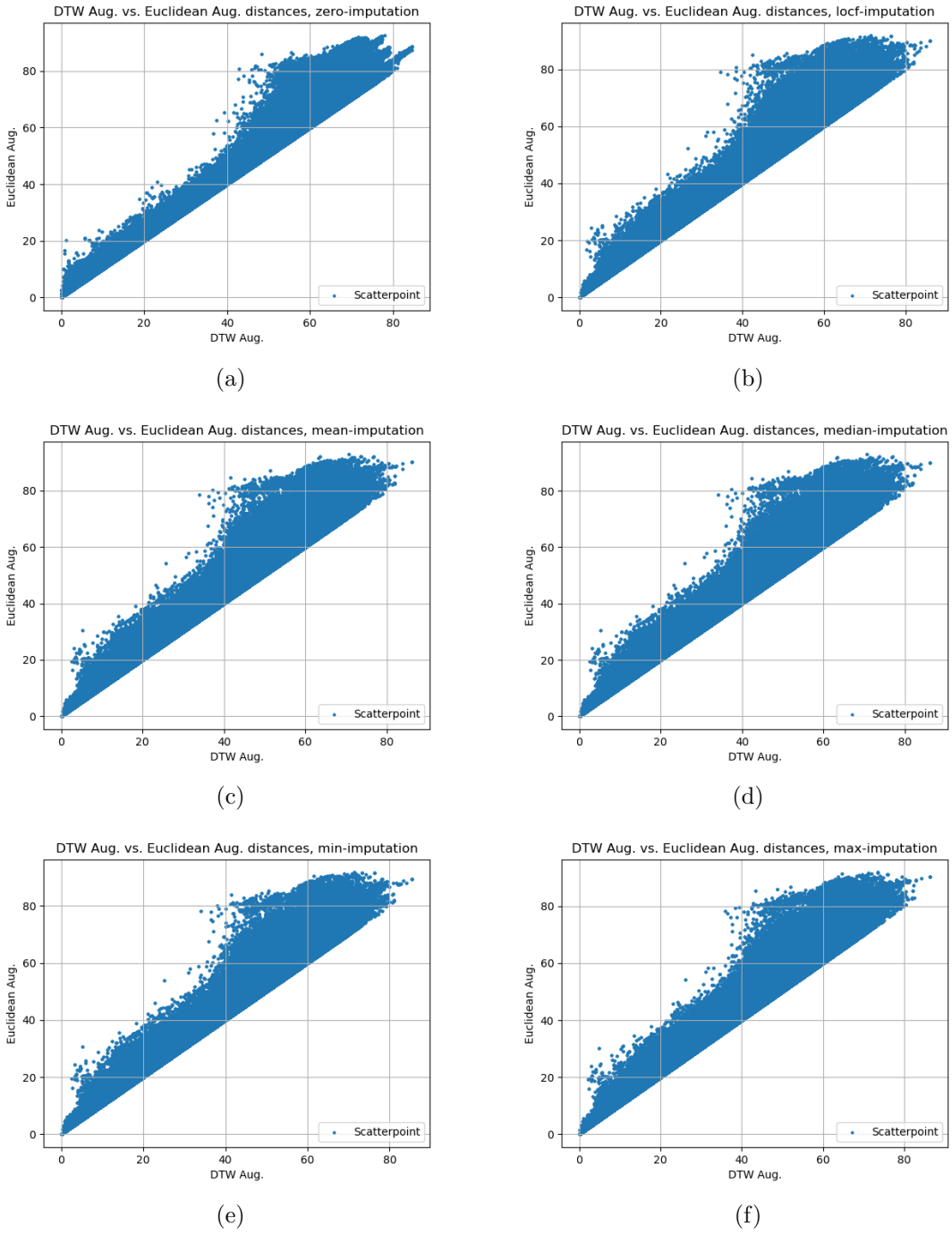


Figure 5.9: Scatter-plots comparing one-by-one the DTW and Euclidean distances distances of the augmented data for different imputation methods for the SSI data set. The sub-figure (a) is zero imputation, (b) is LOCF imputation, (c) is mean imputation, (d) is median imputation, (e) is minimum imputation, and (f) is maximum imputation. Each plot contains 779,689 points.

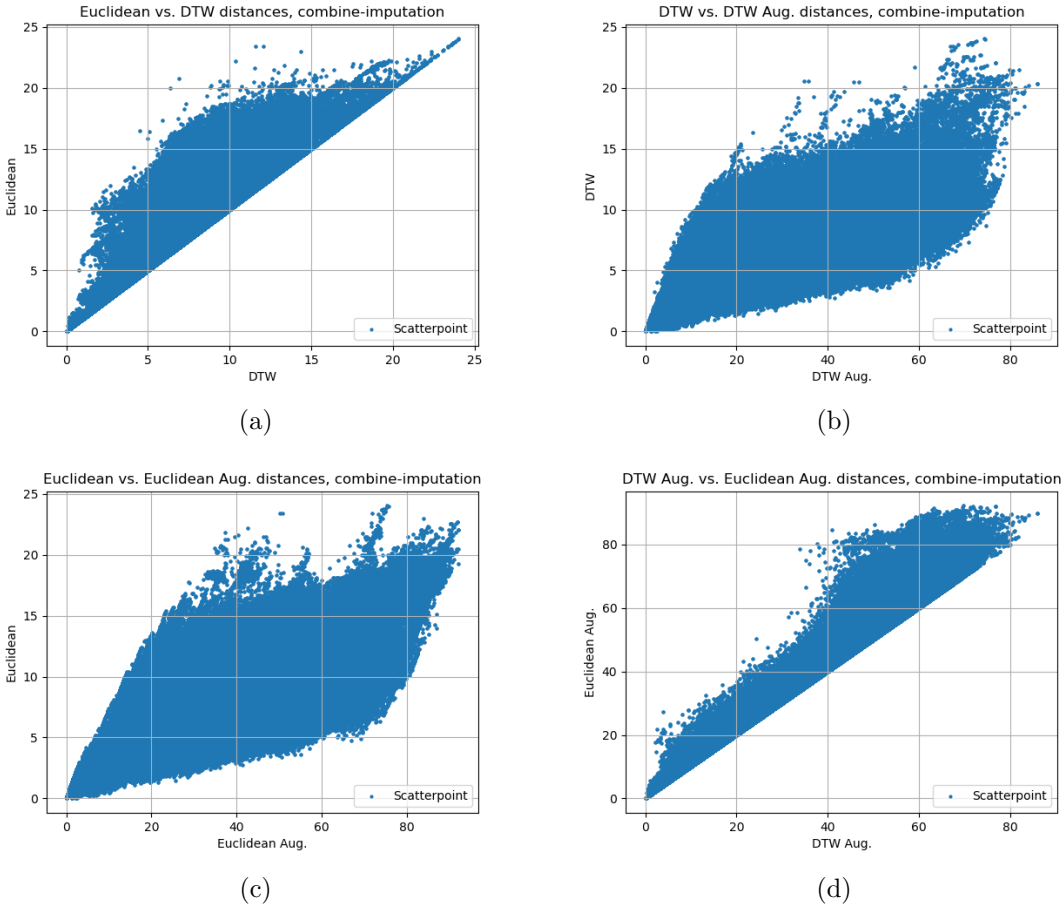


Figure 5.10: Scatter-plots comparing one-by-one the DTW and Euclidean distances on the combined imputation for the SSI data set. The sub-figure (a) is Euclidean distance of the normal data, (b) is comparing DTW with the DTW of the augmented data, (c) is Euclidean distance for normal data and augmented data, and (d) is Euclidean distance of the aug. data versus the same data using DTW. Each plot contains 779,689 points.

5.3 Results for kNN classification on the SSI data

A table of the kNN classifiers best k value, accuracy, balanced accuracy, and F_1 score is shown below for the test sets in table 5.2, followed by plots of the validation set mean balanced accuracy and F_1 scores for odd values of $k = 1$ to 21 in figure 5.11 for 20 different random validation sets drawn from the data set.

One can see in figure 5.11 how the trend across the various imputation methods and distance types appear quite steady with, surprisingly, the Euclidean distance on the augmented data giving the best results, followed closely by the DTW distance on the augmented data, before again the Euclidean and DTW on the non-augmented data, in that order. Each plot increases between $k = 1$ and $k = 3$ before either flattening out or slightly descending, where the clearest descent happens for the non-augmented DTW and Euclidean distances on the zero-imputed data. From table 5.2 one can see that the k which give the highest test score in 5.11 varies quite a bit.

The best imputation method for the DTW distance was the combined imputation with an F_1 score of 74.1 ± 2.9 with $k = 3.20 \pm 2.36$, although the zero imputation came close with an F_1 score of 74.0 ± 2.8. The standard deviation is slightly bigger for the combined imputation with 0.1 difference compared to the result of the zero imputation, but at the same time it also has a slightly larger mean by a difference of 0.1 making up for it, which is why the combined imputation can be better for the DTW distance. The worst result for the DTW distance on the non-augmented data was the minimum-imputed data with an F_1 score of 66.3 ± 2.3.

Applying the DTW distance on the augmented data on the other hand made the minimum-imputed data give the best result among the imputation methods for that group of test results, with a F_1 score of 80.9 ± 1.7. Augmenting the data improved the results for the other imputation methods as well.

Looking at the Euclidean distances on the non-augmented data one can see the same conclusions can be drawn as for the DTW distance on the non-augmented data. The best result for this combination of methods was achieved by the combined imputation with an F_1 score of 76.4 ± 2.6 followed closely by the mean imputation with 75.3 ± 2.2, and the worst result was the minimum-imputed data with an F_1 score of 71.1 ± 2.1.

The minimum imputation being the worst result for the Euclidean distance was again reversed by augmenting the data, same as for the DTW distance. The F_1 score for the minimum imputation was 81.3 ± 1.9, with the next best being the combined imputation with a F_1 score of 81.1 ± 2.1, compared to the worst result of 79.9 ± 1.8 for the median imputation, closely followed by the zero-imputation method with 79.9 ± 1.7.

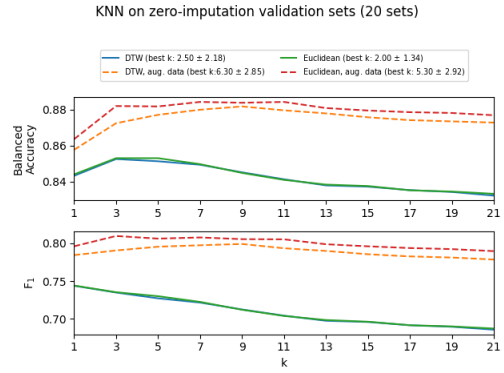
When splitting the data into training, validation and test set there was some problems: the choice of method used to split the data affects the outcome of the classification, the percentage of splitting. The data is biased towards non-SSI patients, and how the randomisation (shuffling) of the data is done affects the classification. Stratification was found to help keep the percentage of each class consistent across data sets and improve the results. Using stratified k-fold split with 2 splits and shuffling gave best results, but for the kNN this method of splitting does not follow the convention of having a larger portion dedicated to training and validation than to testing, although the author could not find a better method. Looking at the plots of F_1 scores in figure 5.11 for the kNN classifier, it is clear that some shuffles also gave better results than others, along with a large variance in outcomes. This is a result of the re-shuffling of the data, as described earlier in the method chapter.

The best imputation method and distance type for the kNN classifier on the SSI data

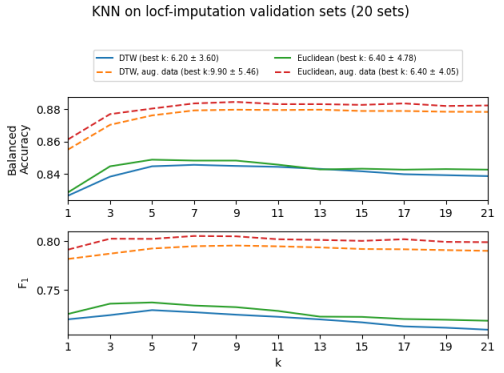
set was found to be minimum imputation on the augmented data with the Euclidean distance with an F_1 score of $81.3 \pm 1.9\%$.

Table 5.2: Score measures for a test set using the kNN classifier on the SSI data. Best k determined by validation set and used on the test set. Best result for each distance measure is marked with **bold** typing.

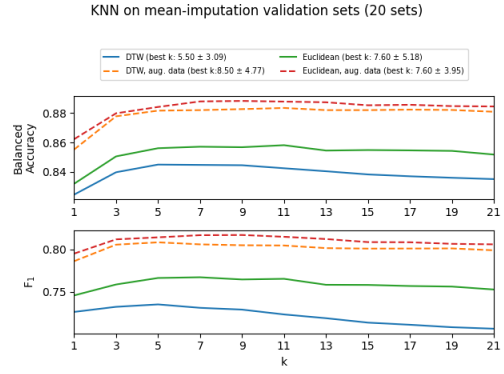
Dist. meas.	Imputation	k	Acc.[%]	Bal. acc. [%]	F_1 [%]
DTW	zero	2.50±2.18	63.6±4.5	84.6±1.3	74.0 ± 2.8
	locf	6.2±3.6	61.4±3.1	84.1±1.0	72.6±1.9
	mean	5.5±3.1	60.3±3.8	83.6±1.5	72.0±2.1
	median	4.80±4.24	58.6±4.3	83.0±1.5	71.1±2.3
	min	6.2±6.37	49.3±4.7	80.9±1.7	66.3±2.3
	max	6.20±5.95	54.8±5.1	82.0±1.6	69.1±2.7
	combine	3.20±2.36	63.9±4.9	84.8±1.5	$\mathbf{74.1 \pm 2.9}$
DTW aug.	zero	6.30±2.85	70.8±3.0	87.0±1.3	78.3±1.9
	locf	9.90±5.46	71.8± 2.8	87..4±1.1	79.0±1.9
	mean	8.50±4.77	72.5±2.7	87.7±1.0	79.4±1.7
	median	7.40±4.72	72.5±3.6	87.5±1.6	79.6±2.2
	min	6.50±4.24	74.3±2.4	88.1±0.9	$\mathbf{80.9 \pm 1.7}$
	max	7.80±4.92	72.2±2.9	87.6±1.1	79.2±1.9
	combine	8.50±5.09	73.8±3.1	88.1±1.2	80.4±2.1
ED	zero	2.00±1.34	64.6±4.2	84.9±1.3	74.6±2.6
	locf	6.40±4.78	62.4±3.0	84.3±0.9	73.2±1.9
	mean	7.60±5.18	65.6±3.7	84.9±1.5	75.3±2.2
	median	11.3±5.10	64.4±3.6	84.7±1.3	74.5±2.2
	min	13.10±5.74	58.7±3.9	83.6±1.0	71.1±2.1
	max	8.80±5.25	65.1±4.2	84.8±1.5	74.9±2.4
	combine	8.70±4.95	67.7±4.2	86.1±1.4	$\mathbf{76.4 \pm 2.6}$
ED aug.	zero	5.30±2.92	73.0±2.48	87.7±1.0	79.9±1.7
	locf	6.40±4.05	73.3±2.8	87.8±1.2	80.1±1.8
	mean	7.60±3.95	74.1±2.9	88.2±1.1	80.5±2.0
	median	7.70±5.49	72.8±2.5	87.5±1.0	79.9±1.8
	min	5.20±4.47	74.7±2.9	88.1±1.5	$\mathbf{81.3 \pm 1.9}$
	max	5.80±4.02	73.5±2.1	87.9±0.8	80.3±1.5
	combine	5.40±4.27	74.5±2.8	88.1±1.2	81.1±2.1



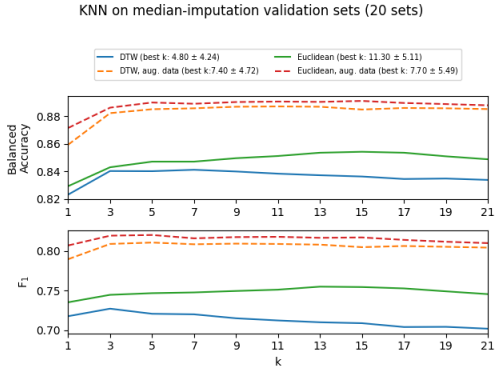
(a)



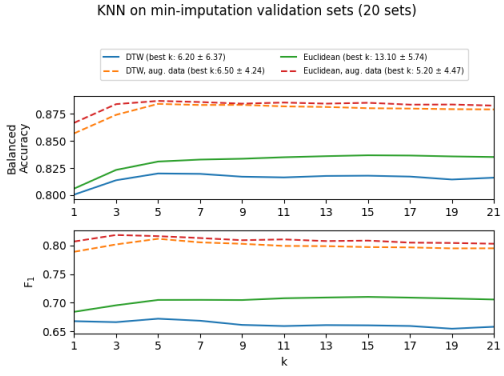
(b)



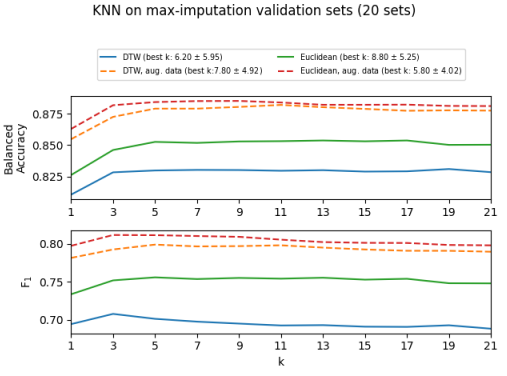
(c)



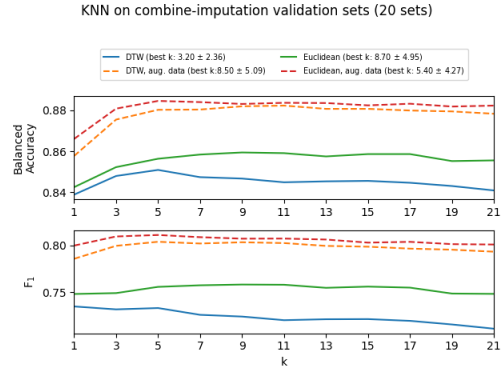
(d)



(e)



(f)



(g)

Figure 5.11: The F_1 scores of the validation sets after 20 shuffles and re-classifying each time on each k for (a) zero imputation, (b) LOCF imputation, (c) mean imputation, (d) median imputation, (e) minimum imputation, and (f) maximum imputation on the SSI data both with and without data augmentation.

5.4 Results for SVM classification on the SSI data

Plots of the score metrics balanced accuracy and F_1-score on the validation set is shown below for 50 values of C ranging between 0.1 and 10 with logarithmic spacing in figure 5.12. Below that, tables of the score metrics accuracy, balanced accuracy, and F_1 are shown for the test set in table 5.3.

From table 5.3 one can see that the best result of the DTW distance on the non-augmented data was the median imputation with an F_1 score of $89.1 \pm 10.8\%$, with the combined imputation close at $86.5 \pm 14.0\%$, and the worst result was the LOCF imputation with $81.3 \pm 23.3\%$.

With the augmented data, the DTW had the best result using the median imputation with an F_1 score of $88.3 \pm 6.7\%$, and the worst result being the minimum imputation with an F_1 score of 85.0 ± 12.8. The next best imputation method was the mean imputation with a F_1 score of $87.9 \pm 9.6\%$.

The Euclidean distance on the non-augmented data gave some nice results, with the mean-imputed data giving an F_1 score of $92.7 \pm 6.8\%$, the highest mean value result for SVM on the SSI data. Slightly worse was the median imputation with an F_1 of $91.0 \pm 6.7\%$, with the worst result for this classifier being the zero-imputed at $78.3 \pm 21.2\%$ with the Euclidean distance.

On the augmented data, the Euclidean distance got best result with the mean imputation giving an F_1 of $89.0 \pm 7.9\%$. The worst result for the Euclidean distance with augmented data was maximum imputation with an F_1 score of $79.8 \pm 13.9\%$.

Looking at the plots from the validation sets in figure 5.12, one can see that the there is a large variation in the mean values of the 20 validation sets represented by each of the plotted lines. It is difficult to determine which of the methods have the best overall results from the validation due to all the "noise", but it appears that the Euclidean and DTW distances did best overall when the plots are seen in isolation. During early testing, the imputation methods resulted in non-positive semi-definite kernels (a.k.a. an indefinite matrix [118]) for the SVM. An indefinite symmetric matrix has both positive and negative eigenvalues. The authors in [57] mention that using DTW distances to construct kernels result in indefinite kernels. This was checked using code for checking the eigenvalues of the SVM kernels. According to Mercer's theorem and the Karush-Kuhn-Tucker theorem, not having a positive semi-definite kernel matrix affects the SVM classifier and how well the equations converge. This might explain the large standard deviations seen in table 5.3 and the noisy plots in figure 5.12. Using another variation of SVM, like the ν-SVM, should give same result according to [3]. It uses a parameter p for controlling the amount of support vectors used for a more geometric interpretation of the data model.

The best result for the SVM on the SSI data was the mean imputation after using Euclidean distance on the non-augmented data, with an F_1 score of $92.7 \pm 6.8\%$.

Table 5.3: Mean values and results with uncertainty for test sets used for SVM classifier on the SSI data. Value of C found using grid search on validation sets. Gamma is the median value of the training sets. Best result for each distance measure is marked with **bold** typing.

Dist.	Imputation	C	γ	Acc. [%]	Bal. acc. [%]	F_1 [%]
DTW	zero	3.00±2.18	9.62±0.16	75.8±19.8	63.3±13.9	82.1±18.9
	locf	5.04±3.41	8.50±0.14	76.2±21.7	73.2±12.0	81.3±23.3
	mean	2.75±1.98	21.51±0.35	84.1±17.2	75.4±7.2	88.4±16.9
	median	3.30±2.50	21.39±0.40	83.6±13.4	77.9±10.0	**89.1 ± 10.8**
	min	4.29±2.85	21.54±0.37	76.3±20.9	75.7±12.9	82.2±19.9
	max	4.43±2.65	19.25±0.29	75.4±19.1	76.0±12.8	82.4±16.0
	combine	4.28±2.93	15.13±0.21	80.8±16.0	77.2±8.7	86.5±14.0
DTW aug.	zero	2.26±1.59	6.41 ±0.09	80.9±13.7	57.3±14.5	87.7±11.4
	locf	3.78±2.79	6.21±0.08	81.4±9.2	73.5±16.9	**88.3 ± 6.7**
	mean	3.27±2.22	7.04±0.11	81.9±12.2	78.0±13.2	87.9±9.6
	median	4.52±2.74	6.97±0.11	79.6 ±11.8	80.5±13.7	86.3±9.4
	min	4.36±2.67	6.93±0.10	78.1±15.6	72.8±17.6	85.0±12.8
	max	3.44±2.33	6.98±0.09	76.3±14.6	72.4±16.1	83.6±12.1
	combine	3.72±2.58	7.12±0.12	80.4±11.9	69.2±17.9	87.2±10.1
ED	zero	2.99±2.25	9.52±0.16	71.7±21.7	59.6±10.0	78.3±21.2
	locf	3.50±2.44	9.01±0.17	80.2±17.4	76.6±11.1	85.7±17.7
	mean	2.16±1.31	17.79±0.38	88.3±9.1	77.7±7.0	**92.7 ± 6.8**
	median	2.76±2.30	17.62±0.32	85.5±9.6	76.3±9.4	91.0±6.7
	min	4.01±2.66	18.64±0.30	77.9±17.5	76.5±8.7	84.1±15.7
	max	3.83±2.56	13.93±0.23	81.7±7.5	86.5±7.5	86.5±12.1
	combine	3.74±2.17	11.96±0.21	80.3±19.9	76.8±10.6	84.6±21.3
ED aug.	zero	3.25±2.78	7.08±0.11	77.3±15.8	62.1±17.8	84.4±14.4
	locf	3.82±2.47	6.29±0.08	72.4±19.8	69.7±21.8	80.3±17.8
	mean	3.11±2.09	6.97±0.10	83.1±10.4	77.6±12.5	**89.0 ± 7.9**
	median	4.23±2.80	6.93±0.09	77.4±14.5	75.5±21.0	85.1±10.5
	min	4.70±2.15	6.91±0.09	77.4±13.7	77.1±14.2	84.7±10.8
	max	3.70± 2.26	6.70±0.07	71.5±17.7	72.1±19.0	79.8±13.9
	combine	4.20± 2.67	7.13±0.10	76.8±19.9	69.1±22.7	83.6±17.3

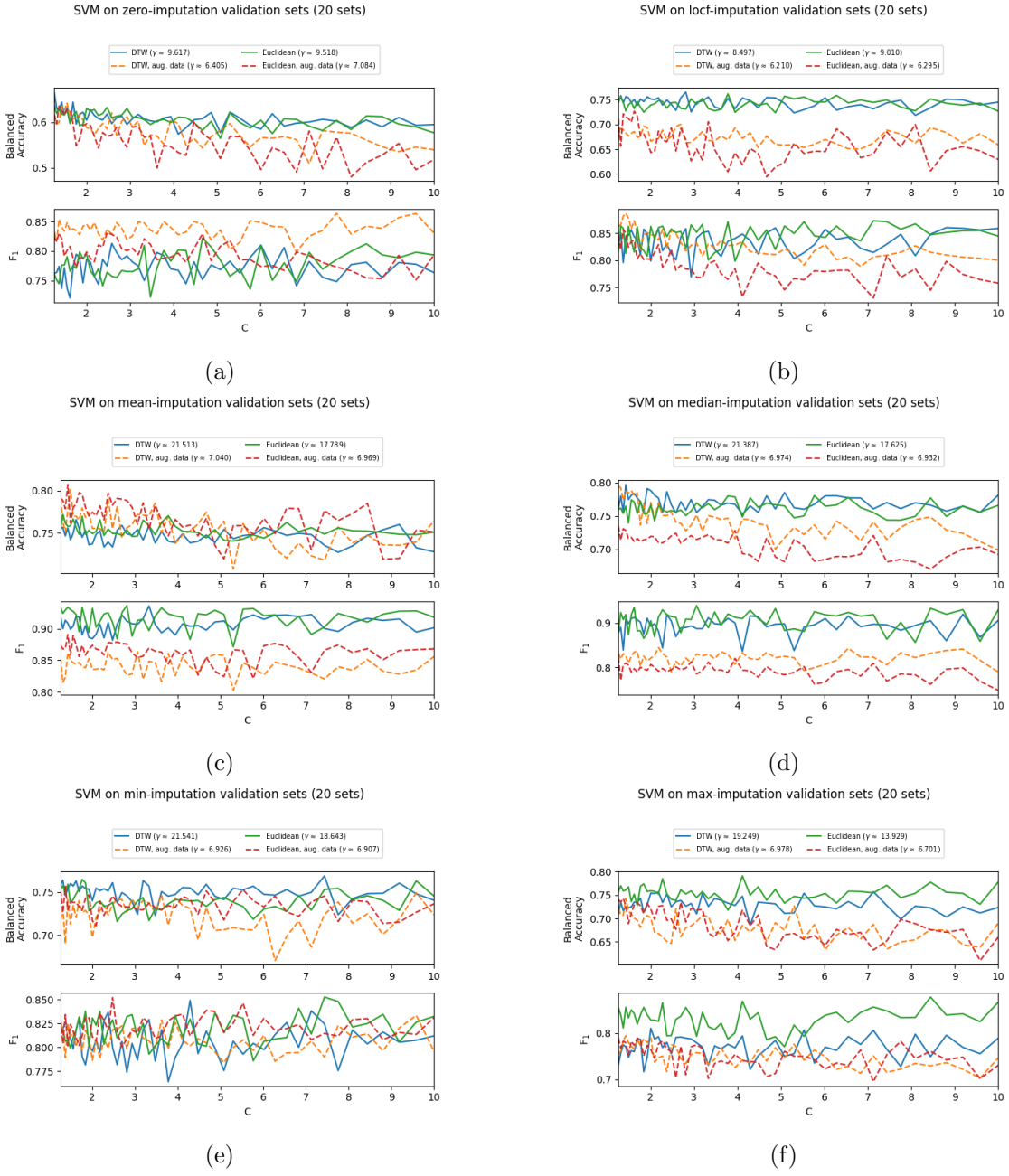


Figure 5.12: Comparison of the score metrics balanced accuracy and F_1 for SVM classification of (a) the zero imputed test data, (b) the LOCF imputed test data, (c) the mean imputed test data, (d) the median imputed test data, (e) minimum imputed test data, and (f) the maximum imputed data from $C = 0.1$ to $C = 10$ on the SSI data set.

5.5 Results for TCN on the SSI data

5.5.1 Experimentation with architecture

Looking at the validation results of figure 3.3 in section 3.4 and their values in appendix B, one can see that having at least 3 neurons in the first dense layer and 2 or more in the second dense layer gave quite good (¿ 80%) validation results for most of the imputed SSI data. The combination and LOCF-imputed data start giving good validation results

with 4 neurons in the first layer. As mentioned earlier, the TCN has only been applied on augmented data due to time constraints.

In this section the abbreviation (a, b) is used, where the numbers a and b in the parenthesis denote the numbers of neurons in the first and second dense/fully connected layers.

Each of the imputation methods have validation results where multiple dense layer configurations gave the same result, which is why only the best result with the lowest number of neurons in each layer will be mentioned. The best validation result for the zero-imputed data was found to be (10, 2) with accuracy 92.3%. LOCF-imputation had its best validation result with (9, 8) at 87.3%, with minimum imputation having its best validation result with the same number of neurons, but with 86.6% accuracy. The mean-imputed data's best validation result was with (10, 8) having accuracy of 87.7%. The median, maximum, and combined imputation had respectively their best validation results with (10, 5), (10, 4), and (9, 9) with the accuracy 87.3%, 89.4%, and 88.0% in afore-mentioned order.

The test results using the three best combinations from the validation results are shown in table 5.4. Each test comes from running and validating the TCN network from start using the number of neurons in the dense layers shown in the table. One can see from the values that the test results are poor, with high standard deviation signifying unstable values. This is because sometimes after initialisation the optimiser gets stuck in a local minimum of the loss function. Another possible reason may be that the data is smaller than what the TCN requires to generalise in a good way. Possible solutions are proposed in section 6.1.

The best imputation method with the current results was found to be the combined imputation method for the SSI data.

Table 5.4: Mean values and results with uncertainty for test sets used for TCN classifier on the SSI data. Number of neurons in the dense layers found using grid search on validation sets. NaN values have been ignored. The three best validation results are tested, with the best result marked in **bold**.

Imputation	Dense 1	Dense 2	Acc. [%]	Bal. acc. [%]	F_1 [%]
zero	10	2	72.3±23.9	51.8±12.1	41.3±17.9
	8	8	59.0±24.0	69.4±15.7	**54.4 ± 13.5**
	9	9	66.9±22.7	67.2±13.2	52.5±11.9
locf	9	8	79.7±23.8	77.3±14.2	**65.0 ± 12.7**
	9	2	73.4±24.1	51.0±13.8	37.9±22.4
	9	10	51.7±26.7	66.4±15.9	54.0±15.0
mean	10	8	64.7±22.7	66.2±13.9	**57.1 ± 11.9**
	10	2	73.7±20.8	61.9±14.5	56.9±24.0
	10	10	76.0±26.1	65.9±16.4	55.2±21.1
median	10	5	63.8±18.5	70.0±12.9	**58.3 ± 10.8**
	10	4	56.2±24.8	64±17.1	52.6±14.2
	10	3	70.9±23.5	50.4±14.7	51.0±15.6
minimum	9	8	37.9±26.3	56.7±21.7	45.4±19.6
	10	2	52.3±23.6	51.1±13.5	42.0±19.4
	9	9	68.9±20.2	77.1±21.2	**65.7 ± 21.1**
maximum	10	4	60.45±20.9	72.7±13.4	57.1±12.4
	10	8	52.8±18.8	62.6±11.6	49.1±14.8
	9	9	71.5±23.4	72.8±14.0	**58.3 ± 11.4**
combine	9	9	75.4±23.7	79.9±13.8	66.7±13.1
	9	2	40.1±26.3	51.0±15.9	42.9±14.6
	8	8	80.2±21.4	80.1±14.5	**66.9 ± 12.6**

5.6 Comparing distance measures on uwave data with missing data

The classifiers used are distance-based, specifically using either DTW or Euclidean distances, and different imputation methods along with whether the data is augmented or not affect the range and distribution of the distance. As with the SSI data, it is of interest to study histograms of the distances used.

5.6.1 DTW distance histograms of the uwave data with missing data

The histograms in figure 5.13 for the DTW distances look largely the same compared to each other, with some differences. For the zero-imputed data in (a) one can see that the histogram has a smooth outline with two tops, first one small around DTW=45 which dips slightly fused to one large peak which peaks at around DTW=65. The first peak has almost half the height of the large peak. The median of the histogram comes shortly after the small dip at DTW=59.9.

The LOCF-imputed data in (b) has largely the same shape, but the first peak has become slightly smaller and resemble more a slightly sloping plateau, and the median has shifted to 65.706.

The first peak or slope changes again for the mean-imputed data in (c), becoming slightly smaller, and the shape becoming marginally narrower compared to the LOCF-

imputed. The median for the mean-imputed data is around 62.0, smaller than the median for the LOCF-imputed, but larger than for the zero-imputed data.

Looking at the median imputation the peak disappears even more and transitions more into an extra slope at the side of the large peak. The median is slightly larger than for the mean imputation, ca. at distance 64.1.

For both the minimum and maximum imputation the first peak/plateau disappears and is entirely absorbed by the second/highest peak, which is now at around distance 80 for both of them, with the median being around 78.6 and 80.3 for the minimum-imputed and maximum-imputed respectively.

When examining the contribution of each of the two classes, the orange and blue line for class 1 and 2 respectively, one can see that the first peak comes from the contribution of class 1, while the plateauing comes from class 2. In minimum imputation and maximum imputation both are affected mostly the same, which is why the first peak and plateau disappear for these two imputation methods.

Figure 5.14 shows the histograms for the augmented data after imputation. One can see that the shape resembles more a steep cartoon volcano with a small first peak descending slightly to a a small platform at the top. The contributions from each class have about the same height for all the imputation methods but with some differences. The curves representing the distances between the different classes and each other have a "winding" platform shape between 35 and 40 for the green curve, 40 to 50 for the blue curve, and the orange ave two of these, the first between ca 35 to 40 and the second around 45 to 50. The peak of the orange the largest, with the green curve slightly shorter height, and a the lowest height is the blue curve. The curves have their peaks close to the median, but with some distance, where from left to right the green curve peaks first before the median, followed by the blue curve peaking after the median, with the green curve peaking last. These features are the same for all the DTW augmented uwave data histograms. The contribution from class 1 have a higher peak in LOCF-, mean-, and median-imputed data, and about the same height for the zero-imputed data.

5.6.2 Euclidean distance histograms of the uwave data with missing data

No significant difference between the shapes of the DTW histograms described in section 5.6.1 and the Euclidean histograms on the non-augmented data can be observed by the naked eye. There is a small difference in the median, where they differ on the third digit after rounding to three decimals, which can be seen in table 5.1 presented earlier that summarizes the medians. This is most likely due to the small percentage (23.7% average in total between class 1 and 2) of synthetic missingness applied on the uwave data used. With a larger amount of missingness there would most likely be more of a difference between both the shape and median of the two histograms. This can also be observed in figure 5.17 (a) and (b), which shows the Euclidean and DTW distance for the combined imputation method.

Figure 5.16 of the augmented data shows that the histograms form a steep partial bell form with the top splitting clearly into two peaks. These peaks are more distinct compared to the DTW on the augmented data. On the zero-imputed data the peaks are mostly the same height, with the second peak slightly more thick. The median is slightly larger with a distance of 49.169, and the maximum distances are sligthly larger compared to the DTW distances. The curves are distributed the same way as the augmented DTW data

For the LOCF-imputed data in figure 5.16b the second peak is a bit higher and more sharp than the first peak. This is reflected by the orange curve (distances between class 1

and 2) being fairly higher than the green curve (distance between 2 and 2). The median is a bit higher than for the zero-imputed data, with a median of 50.728.

Mean imputation and median imputation both have their highest peak being the second one, and their medians are close, with the mean imputation having a median distance of 49.947 while the median imputation have a median distance of 50.703

Minimum imputation with a median of 54.948 and 53.033 for the maximum imputation. Both have a steep cliff followed by an angled platform before rising into the first peak. The angled platform comes from the orange curve having a small peak, along with its main peak being shorter and more blunt. The first peak is higher for both, reflected by the orange curve having a higher peak than the more blunt blue curve.

Common to all the histograms in figure 5.16 is that the median seem to be directly in the center of the valley separating the two peaks. The blue and orange curves do not intersect at the median, except for the minimum and maximum imputation for the Euclidean distances on the augmented data.

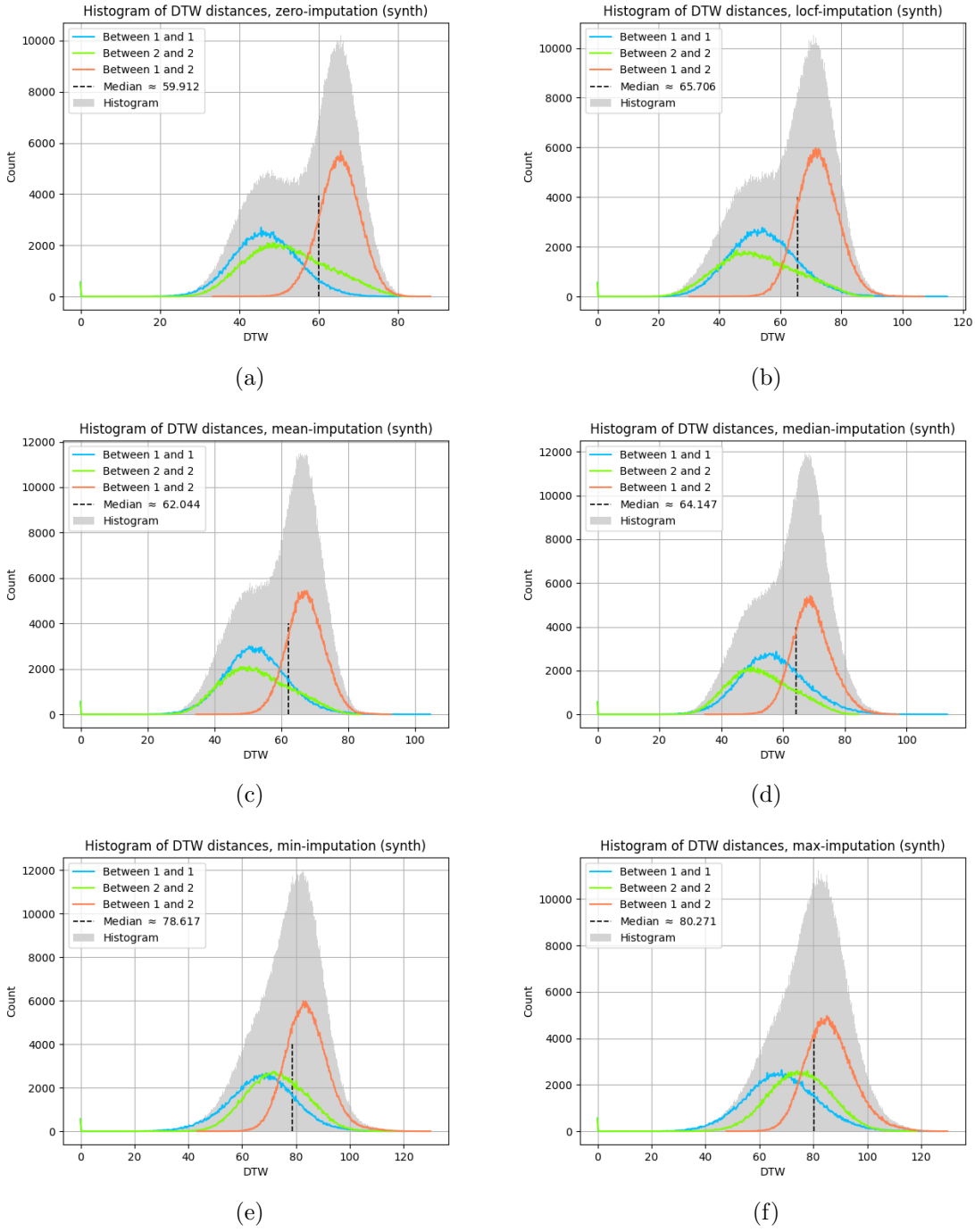


Figure 5.13: Histograms showing the DTW distances for different imputation methods, using Euclidean distance as a cost function on the data set with synthetic missing data. (a) is zero-imputation with a median distance of 59.912, (b) is LOCF-imputation with a median distance of 65.706, (c) is mean-imputation with a median distance of 62.044, (d) is median-imputation with a median distance of 64.147, (e) is minimum-imputation with a median distance of 78.617, and (f) is maximum-imputation with a median distance of 80.271. The blue line show the contribution from pattern of class 1, the green line the contribution from the pattern of class 2, and the orange line the contribution from the difference between the two classes. Dashed line on histogram shows where on the horizontal DTW distance axis the median is. The histogram consist of $1119 \cdot 1119 = 1,252,161$ distance values distributed into 500 bins.

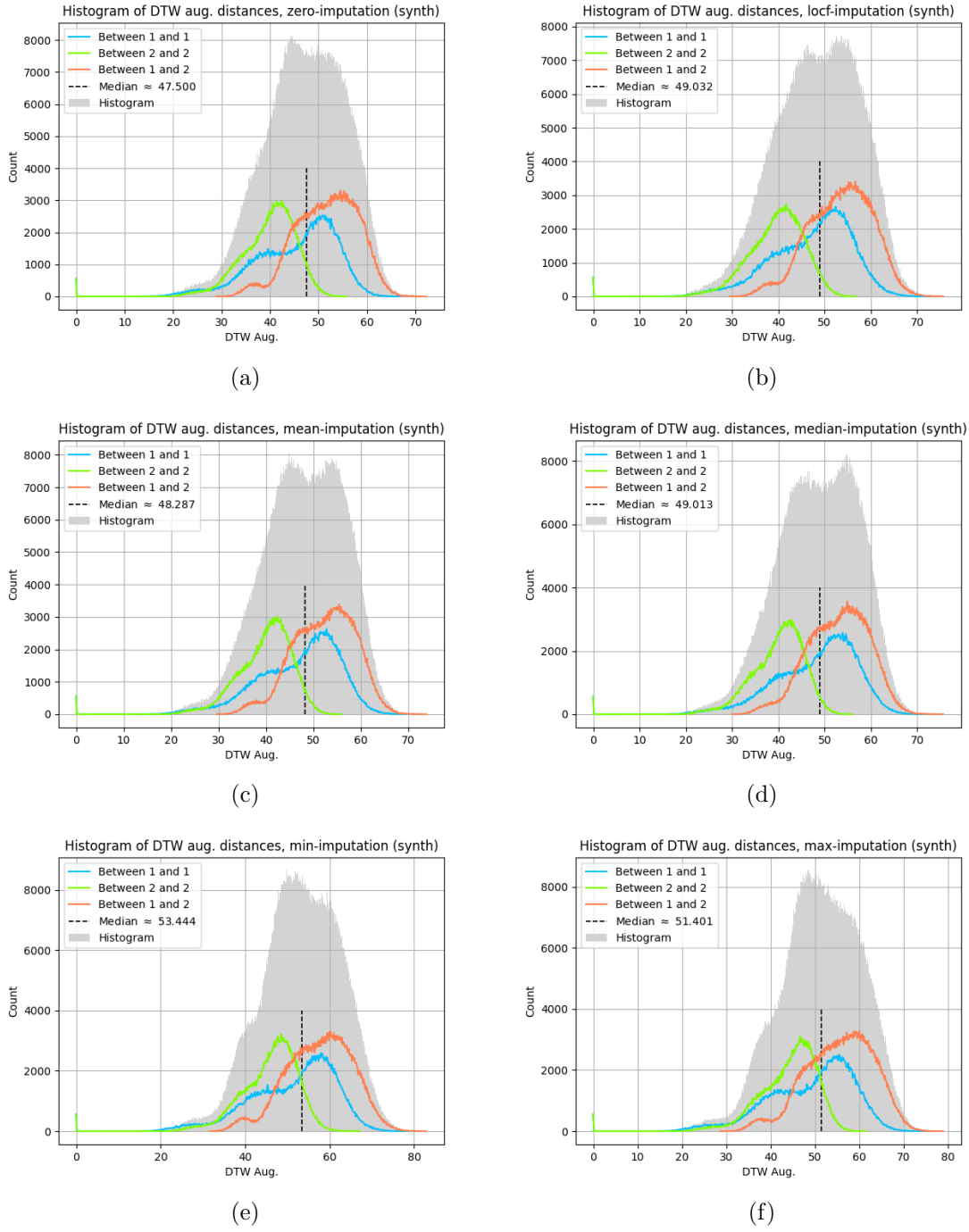


Figure 5.14: Histograms showing the DTW distances for different imputation methods using the augmented data on the uwave data with synthetic missing data. The sub-figure (a) is zero-imputation with a median distance of 47.500, (b) is LOCF-imputation with a median distance of 49.032, (c) is mean-imputation with a median distance of 48.287, (d) is median-imputation with a median distance of 64.147, (e) is minimum-imputation with a median distance of 53.444, and (f) is maximum-imputation with a median distance of 51.401. The blue line show the contribution from pattern of class 1, the green line the contribution from the pattern of class 2, and the orange line the contribution from the difference between the two classes. Dashed line on histogram shows where on the horizontal DTW distance axis the median is. The histogram consist of $1119 \cdot 1119 = 1,252,161$ distance values distributed into 500 bins.

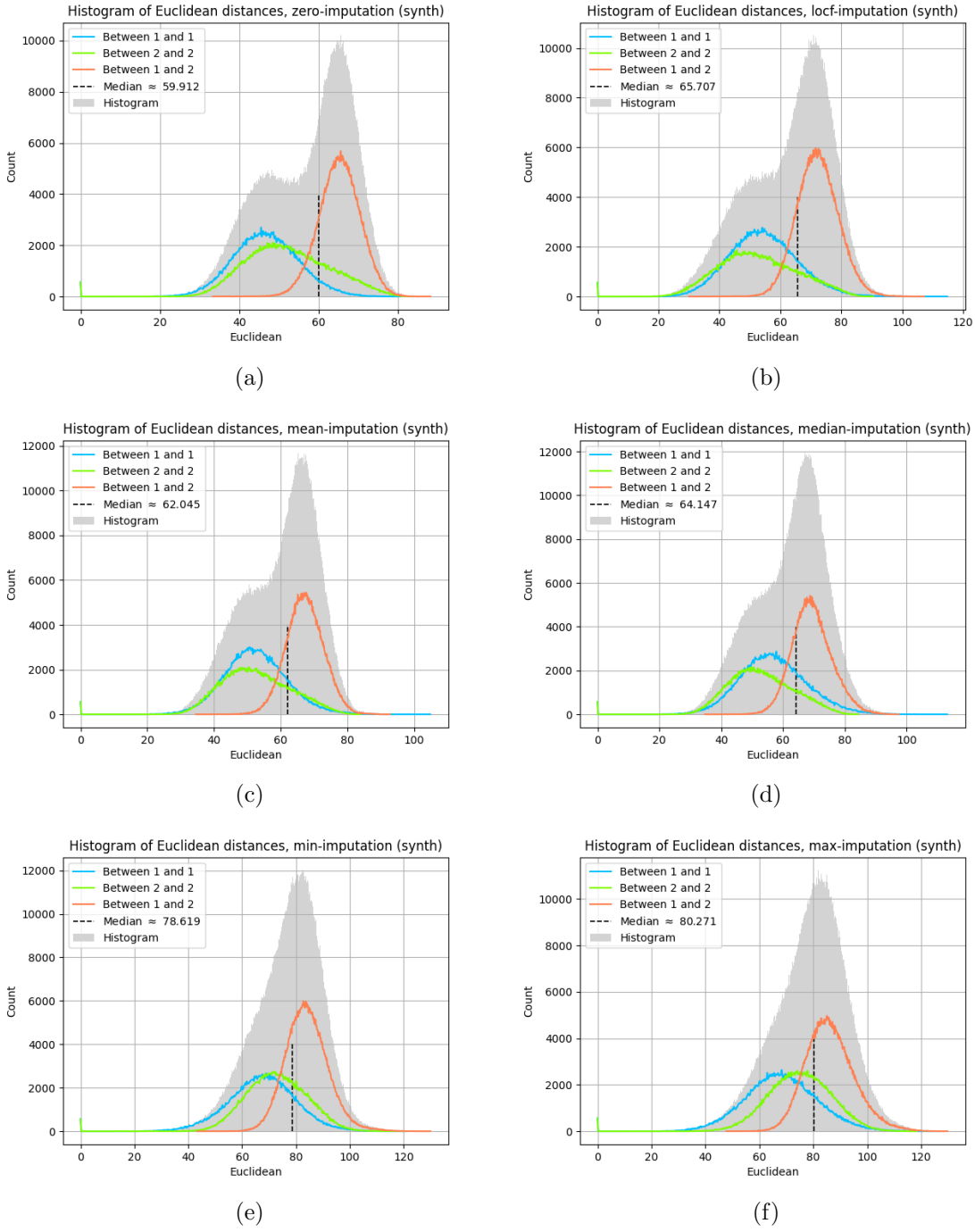


Figure 5.15: Histograms showing the Euclidean distances for different imputation methods on the uwave data with synthetic missing data. The sub-figure (a) is zero-imputation with a median distance of 59.912, (b) is LOCF-imputation with a median distance of 65.707, (c) is mean-imputation with a median distance of 82.045, (d) is median-imputation with a median distance of 64.147, (e) is minimum-imputation with a median distance of 78.619, and (f) is maximum-imputation with a median distance of 80.271. The blue line show the contribution from pattern of class 1, the green line the contribution from the pattern of class 2, and the orange line the contribution from the difference between the two classes. Dashed line on histogram shows where on the DTW distance axis the median is. The histogram consist of $1119 \cdot 1119 = 1,252,161$ distance values distributed into 500 bins.

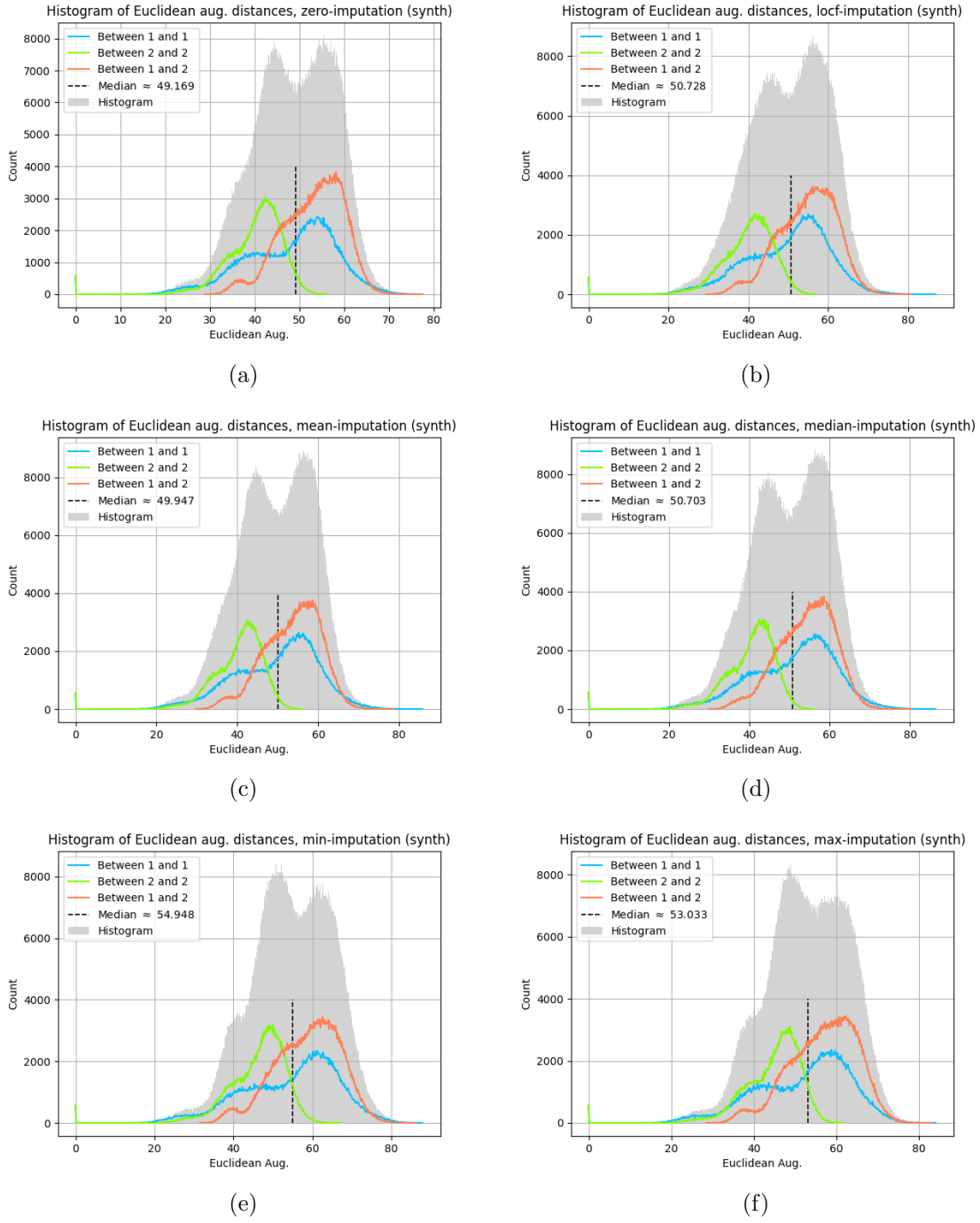


Figure 5.16: Histograms showing the Euclidean distances for different imputation methods using the augmented data on the uwave data set with synthetic missing data. The subfigure (a) is zero-imputation with a median distance of 49.189, (b) is LOCF-imputation with a median distance of 50.728, (c) is mean-imputation with a median distance of 49.947, (d) is median-imputation with a median distance of 50.703, (e) is minimum-imputation with a median distance of 54.948, and (f) is maximum-imputation with a median distance of 53.033. The blue line show the contribution from pattern of class 1, the green line the contribution from the pattern of class 2, and the orange line the contribution from the difference between the two classes. Dashed line on histogram shows where on the DTW distance axis the median is. The histogram consist of $1119 \cdot 1119 = 1,252,161$ distance values distributed into 500 bins.

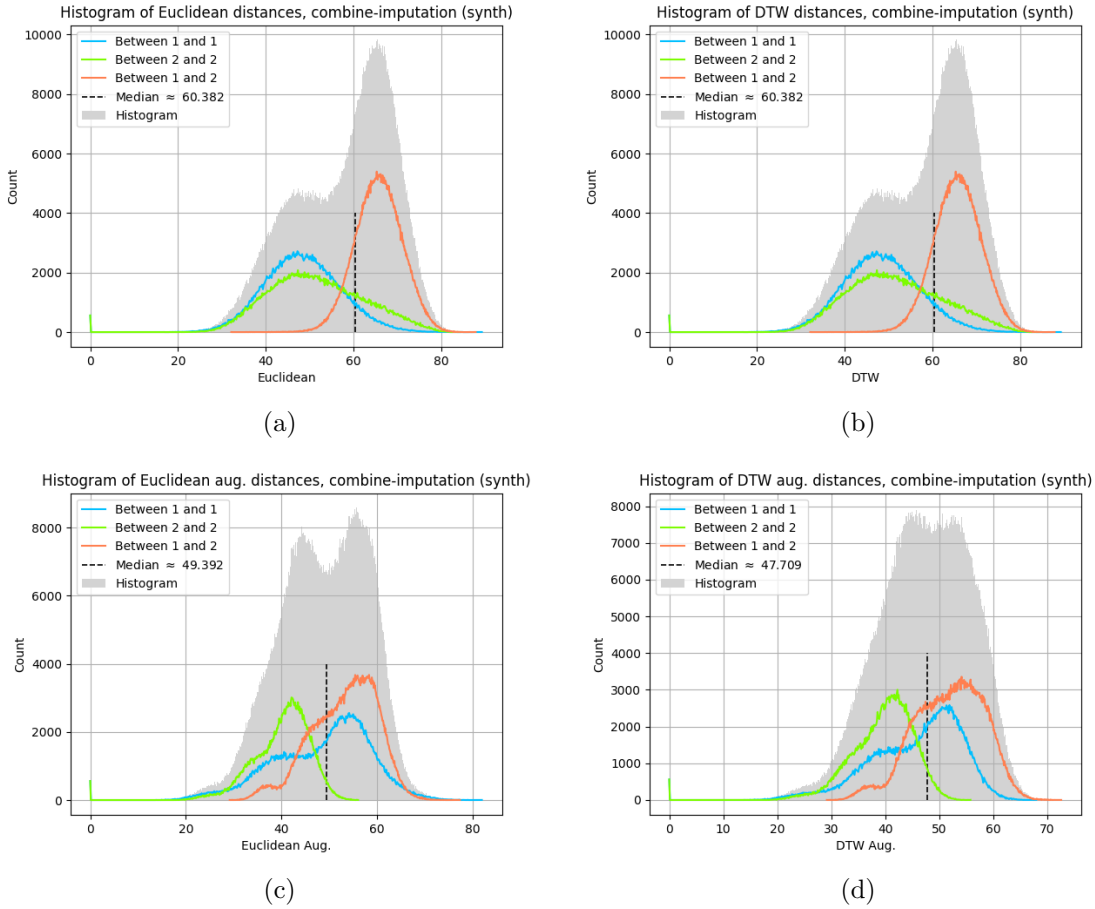


Figure 5.17: Histograms showing the Euclidean and DTW distances for the combined imputation using either the normal data or augmented data on the uwave data with synthetic missing data. The sub-figure (a) is Euclidean distance with normal data having a median distance of 60.382, (b) is DTW on normal data with a median distance of 60.382, (c) is Euclidean distance on the augmented data with a median distance of 49.392, and (d) is DTW used on the augmented data with a median of 47.709. The blue line show the contribution from pattern of class 1, the green line the contribution from the pattern of class 2, and the orange line the contribution from the difference between the two classes. Dashed line on histogram shows where on the DTW distance axis the median is. The histogram consist of $1119 \cdot 1119 = 1,252,161$ distance values distributed into 500 bins.

5.6.3 Some differences when comparing the distances of the missing data from the uwave data

Figure 5.18, 5.20, 5.19, 5.21, and 5.22 show the scatter plots where each distance between two multivariate time series from the uwave data set with missing entries is plotted against another type of distance from their respective distance matrices that was made after imputation. Each distance matrix contain $883 \cdot 883 = 779,689$ values. If the distance values would be the same, then the scatter-points should appear to make a line. Looking at figure 5.10, which is for the combined imputation, one can see the main shapes for each of the distance type comparisons. The distances seem fairly linear with some outlier scatterpoints appearing on the upperside of the apparent line, both in figure 5.18 and

5.22a, with the least difference seen in figure 5.18a from the zero-imputed data. This might be because of the low number of dimensions of the data, with only 3 dimensions, and fairly similar signal patterns between each multivariate time series.

The other distance comparisons appear to have more different distances, and look quite similar in shape to the shapes of the scatter plot comparisons for the SSI data, specifically the scatter plots in figure 5.22b and 5.22c look quite similar to (b) and (c) in figure 5.10. They have quite the wide scattering at the middle, and close together toward the edges in both directions; the best description of the shape would be a mix between a rhombus and an ellipse tilted 45 degrees clockwise. The last type of shape in 5.22a has a flat underside similar to (a), but the overside looks like a wave crashing down, meaning there are some distances which remain the same between the Euclidean and DTW distances on the augmented data. This wave shape appear after ca. distance 25, with the distances below that appearing to be the same.

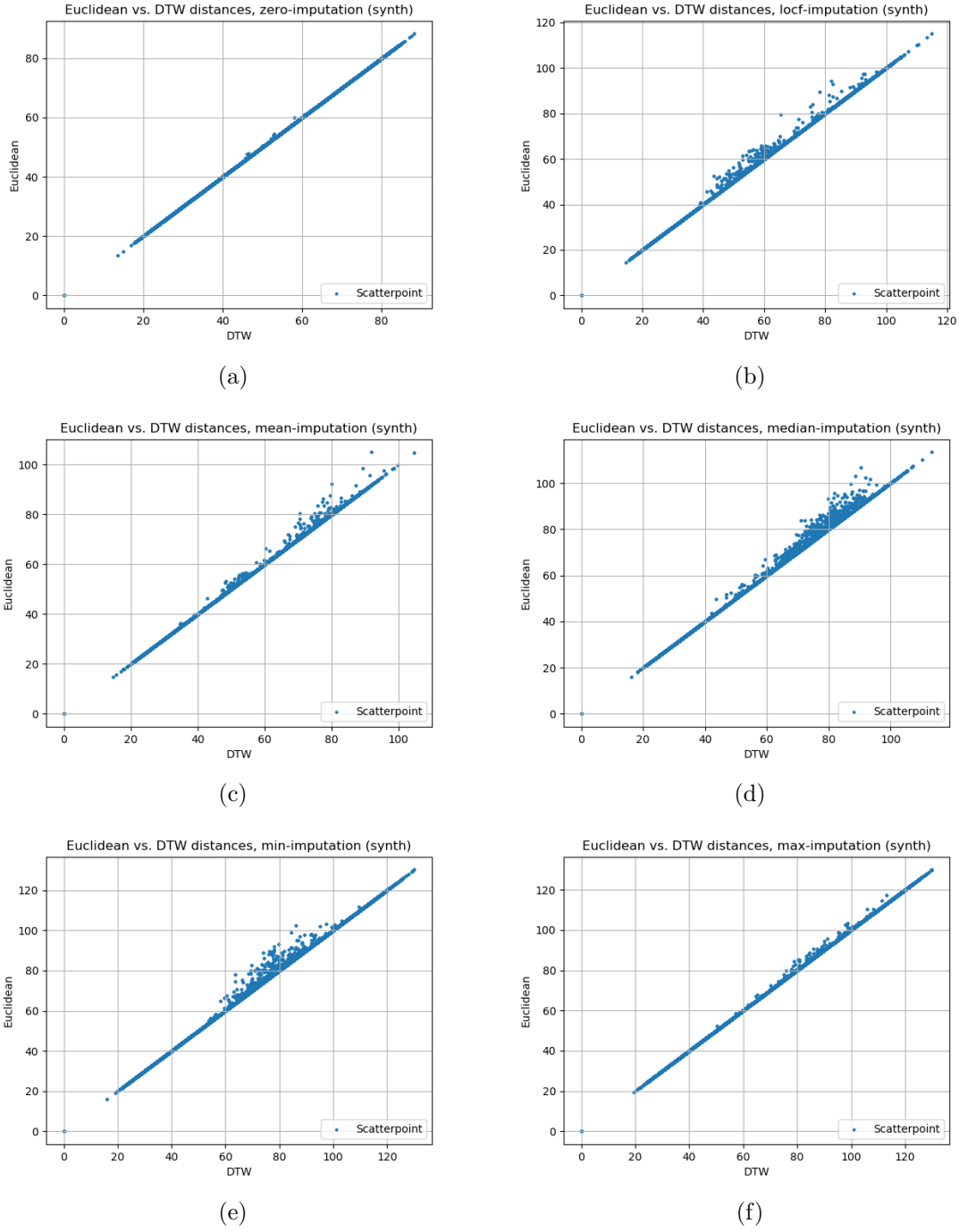


Figure 5.18: Scatter-plots comparing one-by-one DTW and Euclidean distances for different imputation methods on the uwave data with synthetic missing data. The sub-figure (a) is zero imputation, (b) is LOCF imputation, (c) is mean imputation, (d) is median imputation, (e) is minimum imputation, and (f) is maximum imputation. Each plot contains 1,252,161 points.

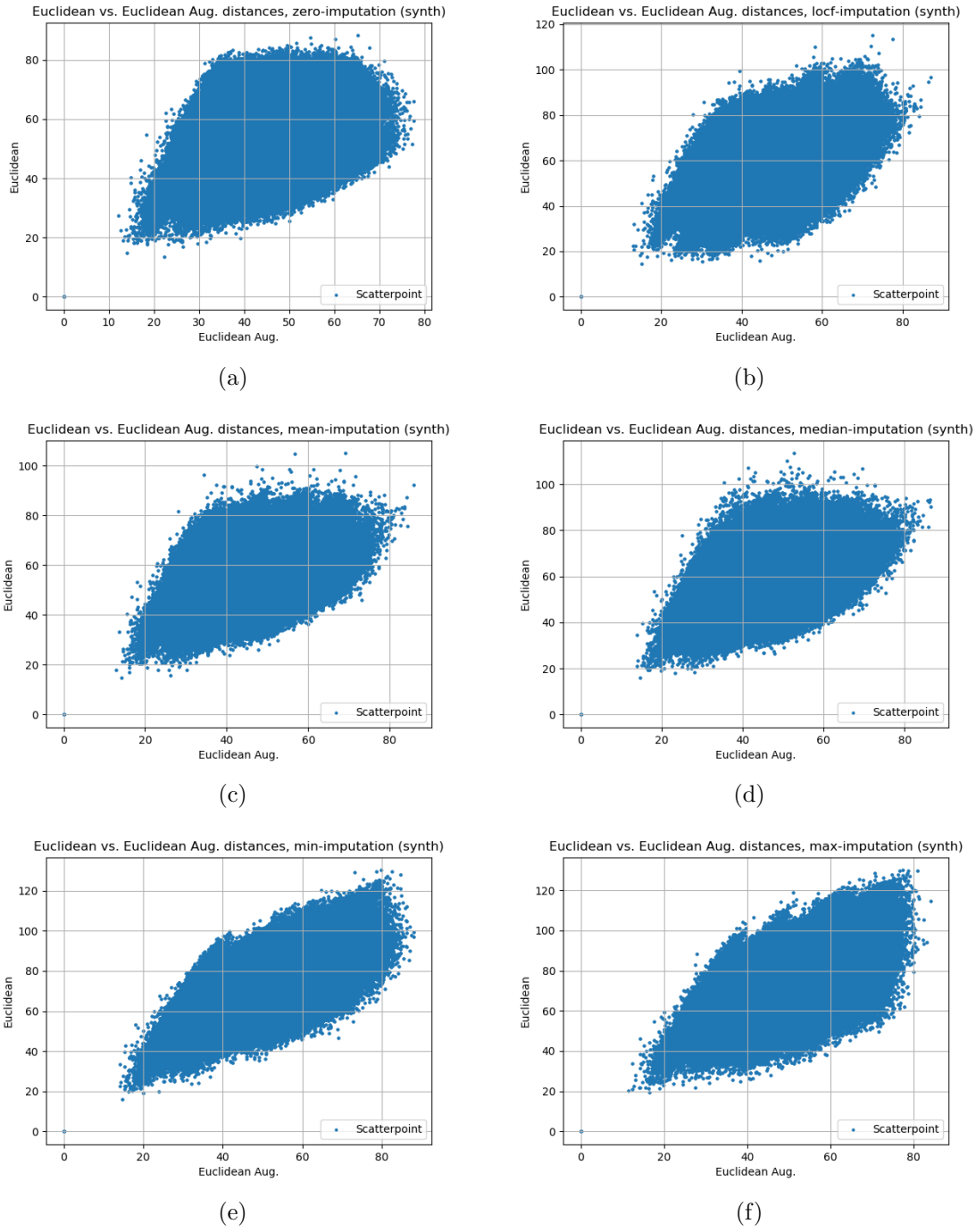
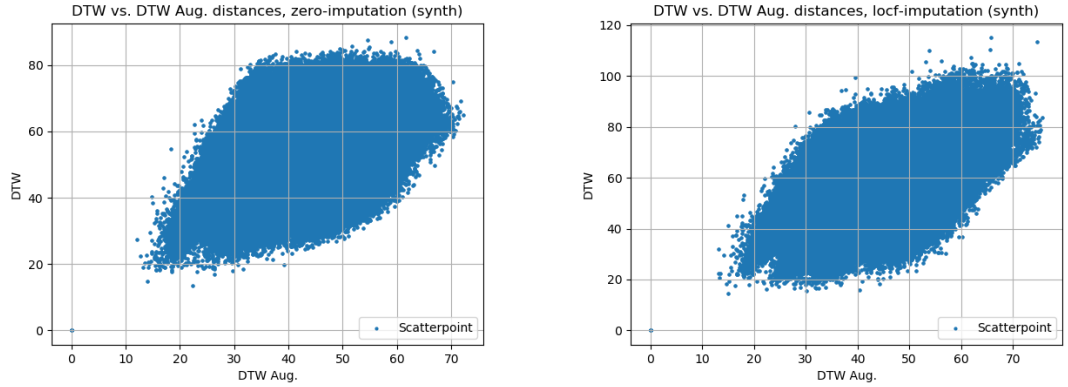
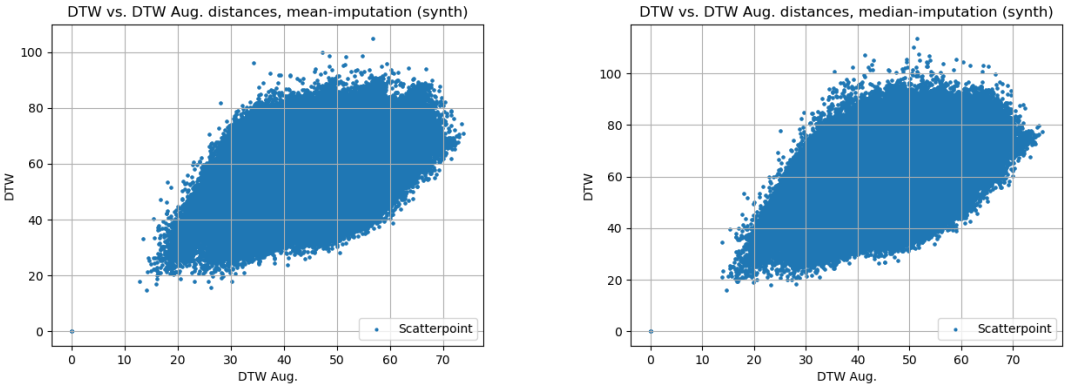


Figure 5.19: Scatter-plots comparing one-by-one Euclidean distances of the original with the augmented data Euclidean distances for different imputation methods on the uvawe data with synthetically missing data. The sub-figure (a) is zero imputation, (b) is LOCF imputation, (c) is mean imputation, (d) is median imputation, (e) is minimum imputation, and (f) is maximum imputation. Each plot contains 1,252,161 points.



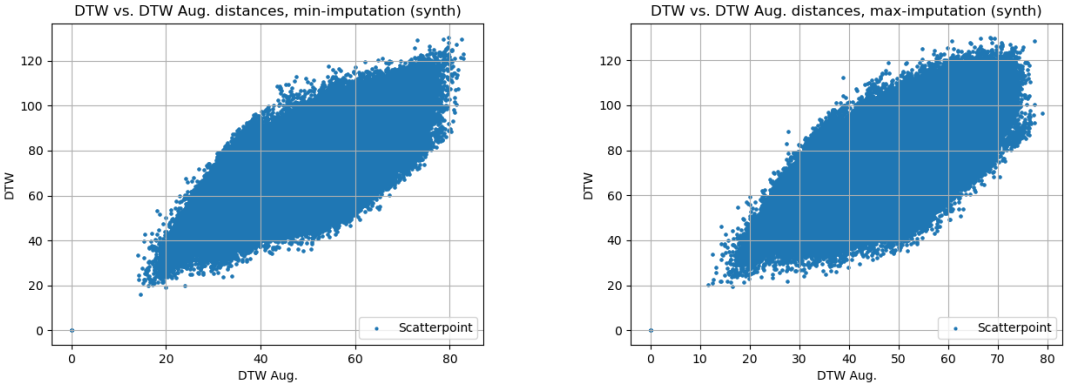
(a)

(b)



(c)

(d)



(e)

(f)

Figure 5.20: Scatter-plots comparing one-by-one DTW for different imputation methods using the the augmented and non-augmented uwave data with missing values. The sub-figure (a) is zero imputation, (b) is LOCF imputation, (c) is mean imputation, (d) is median imputation, (e) is minimum imputation, and (f) is maximum imputation. Each plot contains 1,252,161 points.

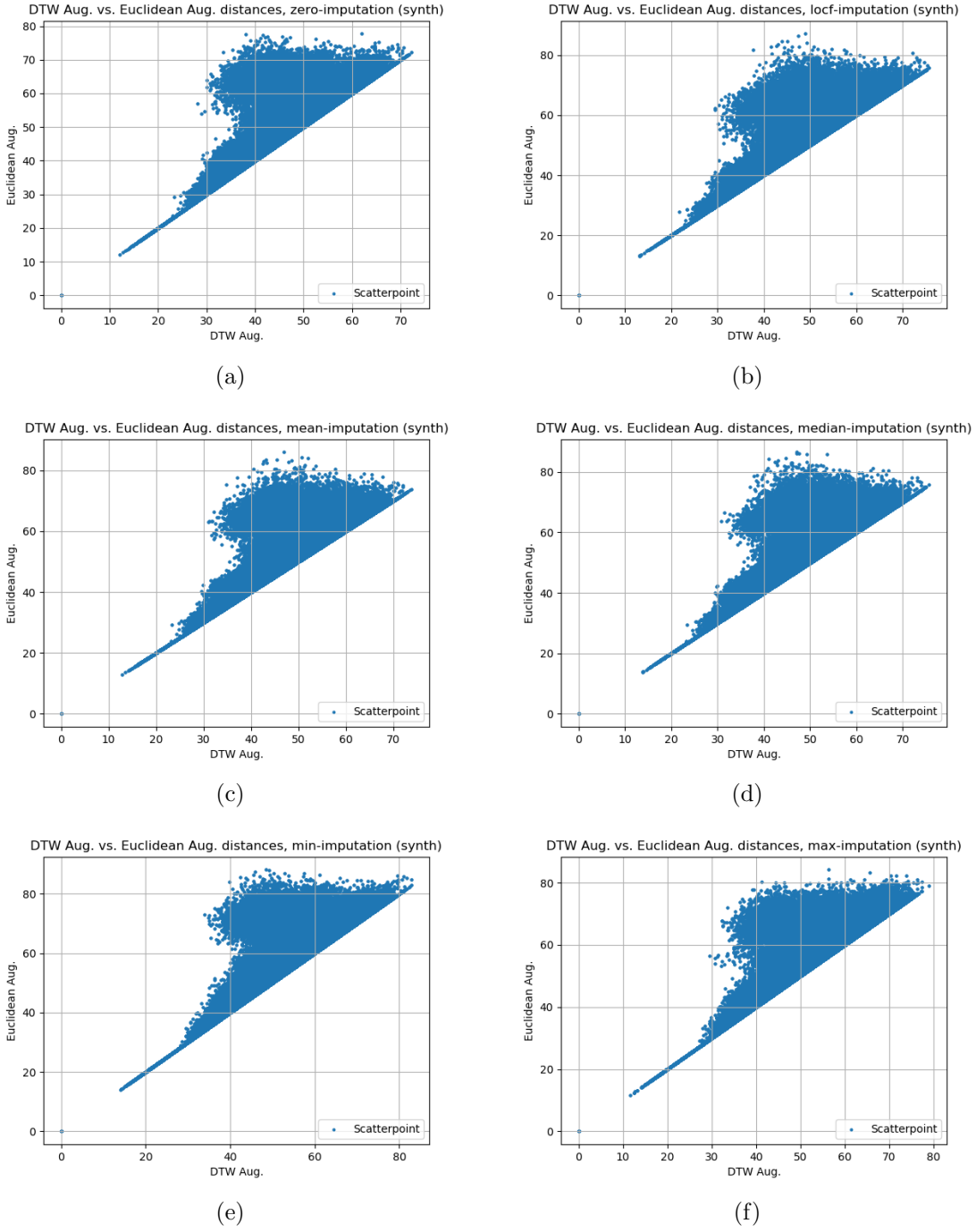


Figure 5.21: Scatter-plots comparing one-by-one the DTW and Euclidean distances distances of the augmented data for different imputation methods on the uwave data with synthetically missing data. The sub-figure (a) is zero imputation, (b) is LOCF imputation, (c) is mean imputation, (d) is median imputation, (e) is minimum imputation, and (f) is maximum imputation. Each plot contains 1,252,161 points.

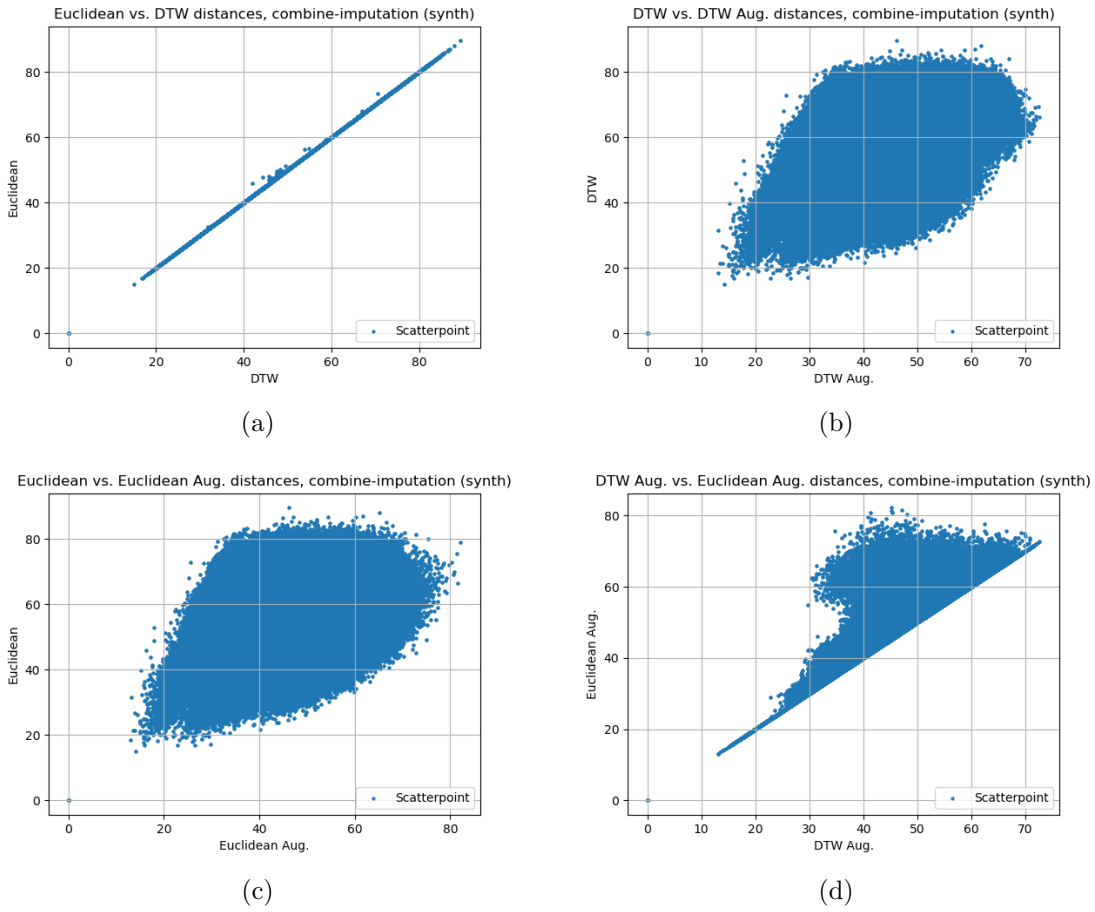


Figure 5.22: Scatter-plots comparing one-by-one the DTW and Euclidean distances distances on the uvawe data with synthetically missing data with the combined imputation. The sub-figure (a) is Euclidean distance of the normal data, (b) is comparing DTW with the DTW of the augmented data, (c) is Euclidean distance for normal data and augmented data, and (d) is Euclidean distance of the aug. data versus the same data using DTW. Each plot contains 1,252,161 points.

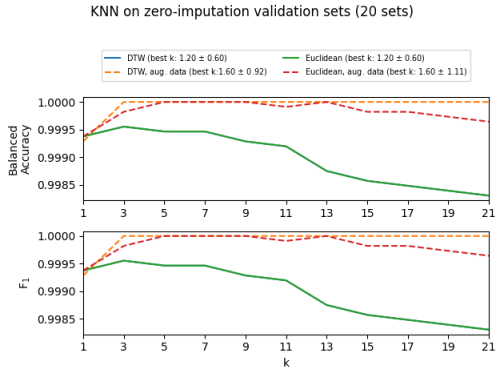
5.7 Results for kNN classification on uwave data with missing data

A table of the kNN classifier's best k value, accuracy, balanced accuracy, and F_1 score is shown below for the test set in table 5.5, followed by plots of the validation set F_1 scores for odd values of k from 1 to 21 in figure 5.23. One can see that all the results of k between 1 and 25 result in a mean balanced accuracy and F_1 scores from about 99% and close to 100%. This is a bit surprising, but might be due to the uwave data used are only for two classes and the average missingness only being around 23.7%. If all the eight classes from the uwave data was used instead of only two, the results might have been worse. The validation results for the Euclidean distance descends as k increases for all the imputation methods, except for LOCF and maximum imputation, which are somewhat level. The LOCF imputation has its F_1 score and balanced accuracy increase between $k = 11$ and 15 for the DTW and Euclidean distance on the augmented data, but after $k = 15$ the result for the Euclidean distance (augmented) decreases, while the result for the DTW (augmented) increases to nearly 100%. For the other imputation methods with the augmented data the results increase when $k = 3$ to 5 and either flattens out or descend around $k = 9$ and $k = 11$ and outwards.

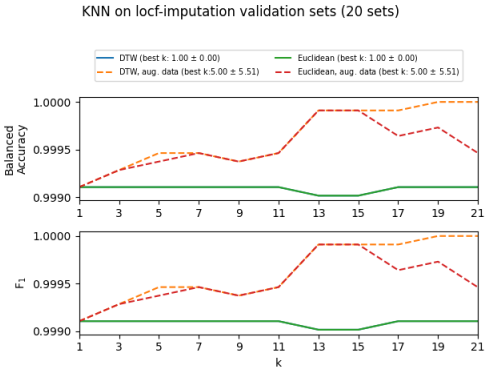
The best imputation method for the instance of uwave data with missing data used was found to be the maximum imputation with F_1 score $99.99 \pm 0.04\%$.

Table 5.5: Result of performing classification on 20 test sets using the kNN classifier on the uwave date with missing data. Best k determined by validation set and used on the test set. Best for each distance measure is marked with **bold** typing.

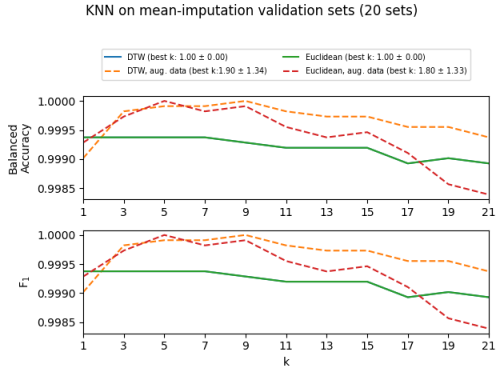
Dist. meas.	Imputation	k	Acc.[%]	Bal. acc. [%]	F_1 [%]
DTW	zero	1.2±0.60	99.85±0.06	99.85±0.06	99.85±0.06
	locf	1.00±0.00	99.87±0.10	99.87±0.10	99.87±0.10
	mean	1.00±0.00	99.86±0.11	99.86±0.11	99.86±0.11
	median	1.00±0.00	99.90±0.09	99.90±0.09	**99.90 ± 0.09**
	min	4.10±2.93	99.70±0.19	99.70±0.19	99.70±0.19
	max	4.10±4.31	99.39±0.26	99.39±0.26	99.39±0.26
	combine	1.10±0.44	99.84±0.096	99.84±0.096	99.84±0.096
DTW aug.	zero	1.60±0.92	99.91±0.09	99.91±0.09	99.91±0.09
	locf	5.00±5.51	99.91±0.09	99.91±0.09	99.91±0.09
	mean	1.90±1.34	99.89±0.13	99.89±0.13	99.89±0.13
	median	1.80±0.98	99.90±0.09	99.90±0.09	99.90±0.09
	min	2.60±2.24	99.94±0.13	99.94±0.13	99.94±0.13
	max	1.00±0.00	99.99±0.04	99.99±0.04	**99.99 ± 0.04**
	combine	2.20±1.47	99.93±0.12	99.93±0.12	99.93±0.12
ED	zero	1.20±0.60	99.84±0.06	99.85±0.06	99.84±0.06
	locf	1.00±0.00	99.87±0.10	99.87±0.10	99.87±0.10
	mean	1.00±0.00	99.86±0.11	99.86±0.11	99.86±0.11
	median	1.00±0.00	99.90±0.089	99.90±0.89	**99.90 ± 0.89**
	min	4.10±2.93	99.70±0.19	99.70±0.19	99.70±0.19
	max	4.10±4.31	99.39±0.26	99.39±0.26	99.39±0.26
	combine	1.1±0.44	99.84±0.10	99.84±0.10	99.84±0.10
ED aug.	zero	1.60±1.11	99.91±0.11	99.91±0.11	99.91±0.11
	locf	5.00±5.51	99.90±0.11	99.90±0.11	99.90±0.11
	mean	1.80±1.33	99.89±0.12	99.89±0.12	99.89±0.12
	median	1.80±0.98	99.89±0.09	99.89±0.09	99.89±0.09
	min	2.50±2.09	99.90±0.14	99.90±0.14	99.90±0.14
	max	1.20±0.60	99.97±0.06	99.97±0.06	**99.97 ± 0.06**
	combine	2.20±1.47	99.92±0.12	99.92±0.12	99.92±0.12



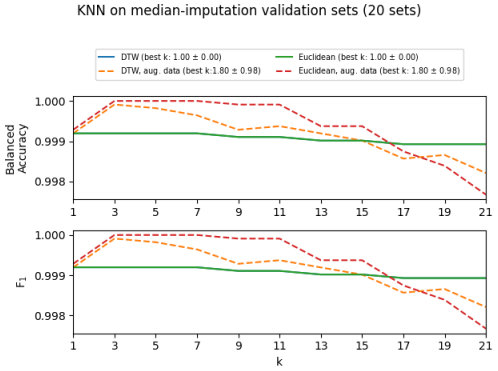
(a)



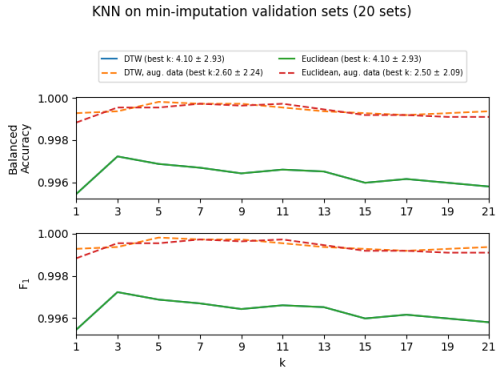
(b)



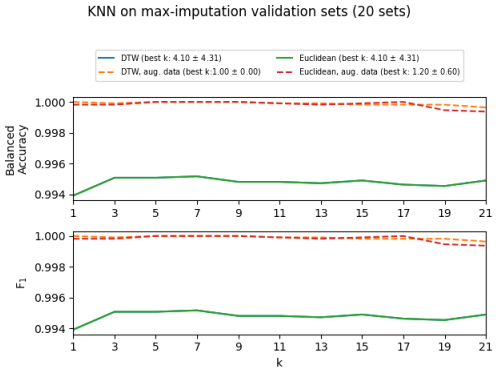
(c)



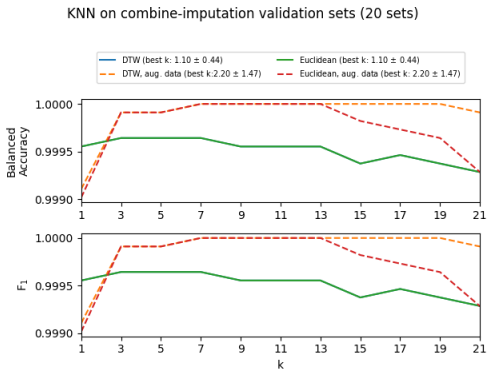
(d)



(e)



(f)



(g)

Figure 5.23: The F_1 scores of the validation sets after 20 shuffles and re-classifying each time on each k for (a) zero imputation, (b) LOCF imputation, (c) mean imputation, (d) median imputation, (e) minimum imputation, and (f) maximum imputation on the uwave data with synthetic missing data both with and without data augmentation.

5.8 Results for SVM classification on the uwave data with missing data

Plots of the score metrics balanced accuracy and F_1-score are shown below for the validation set for 50 values of C ranging from 0.1 to 10 with logarithmic spacing in figure 5.24. Below that, tables of the score metrics accuracy, balanced accuracy, and F_1 are shown for the test set in table 5.3.

One can see in figure 5.24 that overall the the Euclidean and DTW distance did about equally well on the validation sets for the non-augmented data with results ranging between 95% and 100%. Getting such high results was quite surprising, especially since the augmented data did slightly worse. The third highest validation results was overall achieved by the DTW distance followed by the Euclidean distance on the augmented data. It seems that augmenting the data for this data set with a low amount of missingness was detrimental for the SVM classifier when considering how well the classifiers did on the non-augmented data.

The validation plots can also be described as less noisy than for the SSI data, except for the maximum-imputed data. The author does not understand why the maximum imputation gave the noisy validation plots it did in figure 5.24f. This noise is also reflected by the test results in table 5.6. In table 5.6 the results for test sets show that the best imputation method for the DTW distance on the non-augmented data was minimum imputation with F_1 score $98.9 \pm 1.1\%$, with the LOCF-imputed data close at $98.6 \pm 2.3\%$. The worst result for the DTW distance was the combined imputation method with F_1 score $95.4 \pm 15.7\%$.

On the augmented data the DTW distance had its best result for the maximum imputation with an F_1 score of $97.4 \pm 2.2\%$, with the combined and zero imputation closely following at F_1 scores of $97.2 \pm 4.2\%$ and $97.0 \pm 6.2\%$, respectively. It should be noted that the upper limit of the F_1 score at 100% naturally limits the standard deviation, and should be kept in mind when one see that there are test results with very high standard deviation values among the results.

For the Euclidean distance on the non-augmented data the best test result was achieved by using combined imputation with F_1 score of $98.9 \pm 1.1\%$, followed closely by mean and LOCF imputation, in that order, with $98.9 \pm 1.5\%$ and $98.8 \pm 2.7\%$, respectively. The worst result for this distance was the maximum imputation with F_1 score $95.9 \pm 12.7\%$, which is a very high standard deviation when compared to many of the other results for the uwave data.

The overall worst results was achieved by the Euclidean distance on the augmented data, with the best result being produced by the combined imputation with F_1 score $91.7 \pm 6.0\%$, and the maximum imputation following closely with F_1 score $90.3 \pm 7.4\%$. The worst result was for the median imputation with F_1 score $86.2 \pm 7.2\%$ on the non-augmented data with the Euclidean distance.

Euclidean and DTW distances on the non-augmented data did equally well with the overall best results on different imputation methods, with minimum imputation giving the best result for DTW distance and combined imputation for the Euclidean distance both with F_1 score $98.9 \pm 1.1\%$.

Table 5.6: Mean values and results with uncertainty for test sets used for SVM classifier on the uwave data with synthetic missing entries. Value of C found using grid search on validation sets. Gamma is the median value of the training sets. Best for each distance measure is marked with **bold** typing.

Dist.	Imputation	C	γ	Acc. [%]	Bal. acc. [%]	F_1 [%]
DTW	zero	2.00±1.80	2.17±0.01	98.3±2.8	98.2±3.3	98.0±3.9
	locf	2.55±1.55	3.06±0.01	98.7±2.2	98.6±2.1	98.6±2.3
	mean	2.10±2.13	2.85±0.01	98.1±3.2	98.0±3.3	97.8±3.8
	median	1.56±0.51	3.12±0.01	98.4±2.5	98.4±2.5	98.4±2.7
	min	2.78±2.65	2.74±0.01	98.9±1.1	98.9±1.1	**98.9 ± 1.1**
	max	2.63±2.39	2.62±0.01	97.5±6.3	97.4±6.7	96.7±9.5
	combine	2.37±2.41	2.20±0.01	97.1±8.4	97.0±9.0	95.4±15.7
DTW aug.	zero	2.36±2.03	2.31±0.01	96.5±8.0	96.6±7.6	97.0±6.2
	locf	2.59±2.03	2.40±0.02	95.9±5.2	95.9±5.2	96.3±4.5
	mean	2.88±1.83	2.34±0.01	95.0±5.6	95.2±5.4	95.3±5.0
	median	2.93±2.26	2.39±0.01	91.6±12.0	91.7±12.1	93.1±8.7
	min	2.59±1.69	2.41±0.02	91.2±10.2	91.6±9.5	92.3±8.2
	max	2.73±1.47	2.36±0.01	97.4±2.2	97.5±2.1	**97.4 ± 2.2**
	combine	1.77±0.85	2.32±0.01	97.1±4.3	97.3±3.9	97.2±4.2
ED	zero	2.21±1.92	2.17±0.01	98.7±2.1	98.6±2.5	98.5±2.8
	locf	1.99±1.08	3.06±0.01	98.9±2.5	98.9±2.5	98.8±2.7
	mean	1.68±1.28	2.87±0.01	99.0±1.3	99.0±1.4	98.9±1.5
	median	1.94±1.92	3.12±0.01	98.1±3.1	98.1±3.1	98.1±3.0
	min	2.22±1.49	2.74±0.01	98.7±0.9	98.6±1.0	98.6±1.0
	max	2.18±1.42	2.62±0.01	97.1±7.7	97.1±8.1	95.9±12.7
	combine	2.44±2.29	2.20±0.01	98.9±1.1	98.9±1.1	**98.9 ± 1.1**
ED aug.	zero	3.68±2.12	2.50±0.02	88.4±11.6	89.0±10.8	89.9±9.2
	locf	2.95±2.00	2.96±0.02	88.4±10.0	88.8±9.5	89.9±7.8
	mean	4.50±3.20	2.96±0.02	85.1±12.6	85.8±11.6	87.4±9.4
	median	3.55±2.70	2.92±0.02	83.6±10.4	83.8±10.8	86.2±7.2
	min	3.07±1.87	2.57±0.02	85.7±10.2	86.1±9.5	87.7±7.7
	max	3.91±2.752	2.52±0.01	89.6±8.5	89.9±7.9	90.3±7.4
	combine	3.00±2.02	2.77±0.02	91.0±7.8	90.9±8.1	**91.7 ± 6.0**

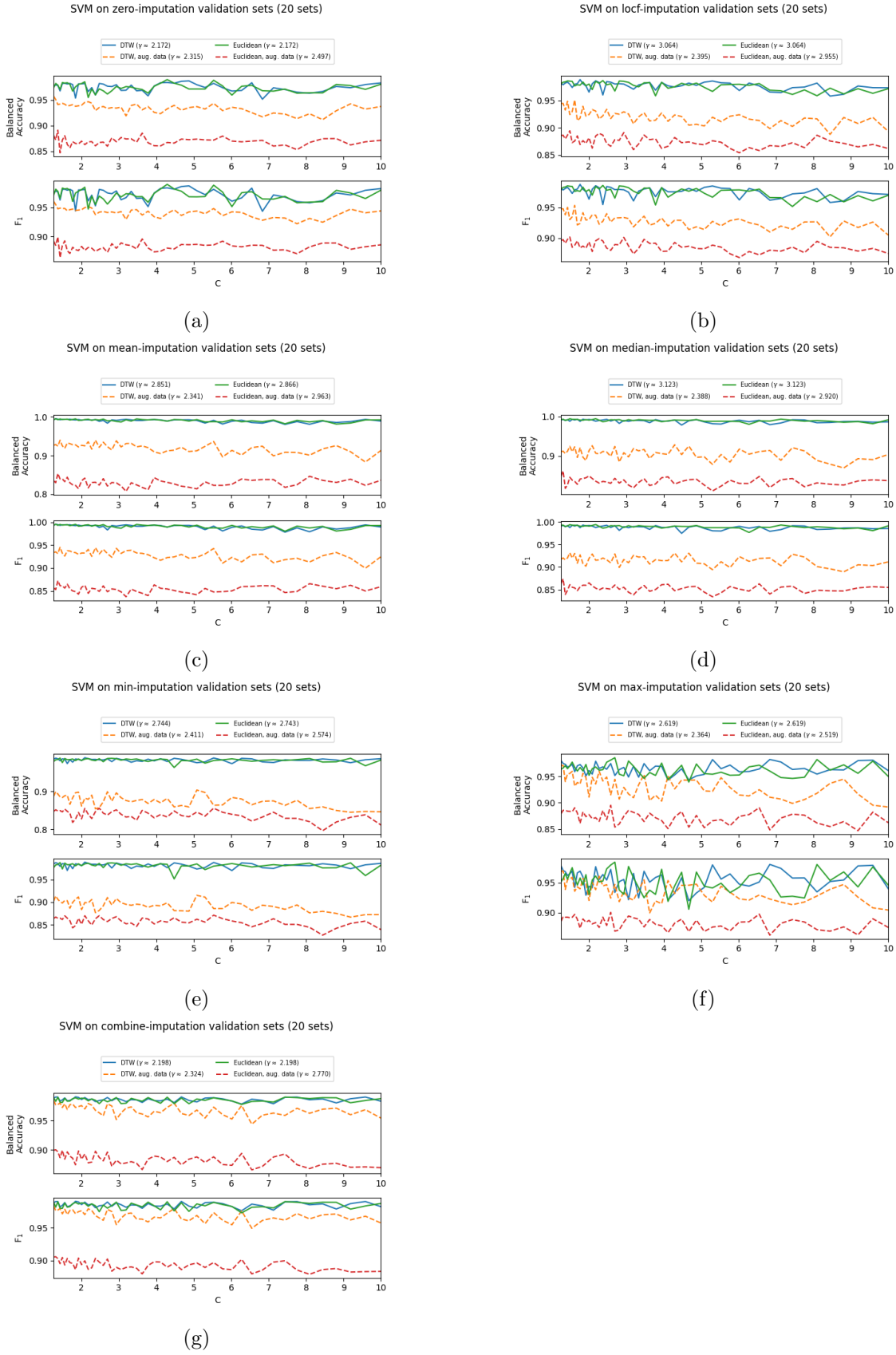


Figure 5.24: Comparison of the score metrics balanced accuracy and F_1 for SVM classification of (a) the zero imputed test data, (b) the LOCF imputed test data, (c) the mean imputed test data, (d) the median imputed test data, (e) mininmum imputed test data, and (f) the maximum imputed data from $C = 1$ to $C = 10$ for the uwave data with synthetic missing data both with and without augmentation.

5.9 Results for TCN on uwave data with missing data

The same TCN architecture was used as with the SSI data, while cutting down on the number of parameters that were validated due to time constraints. The author checked the number of dense layer neurons for every even number of neurons using grid search. The number of validation runs was reduced to 10 from 20, and the same for the patience of the early stopping (if no improvement on validation accuracy or validation loss the training stops early). Looking at the validation results in figure 3.4, one can see good results usually when there more than or equal to six neurons in the first dense layer and 2 or more neurons (but less or equal to that in the first layer) in the second dense layer. It appears that at least 4 neurons in the first layer is needed to get better results than guessing for this data set.

In the zero-imputed data the first good validation results (accuracy around 80% or more) are obtained when there are 8 neurons in the first dense layer and 2 neurons in the second dense layer, giving an accuracy of 85.8%. With 4 neurons in the first dense layer and 4 in the second layer the results are almost as good with 78.8%. The best validation results are higher. The best validation result for the zero-imputation was with 10 neurons in both the first and second dense layer with an accuracy of 94.7%. The second highest was with 10 neurons in the first dense layer and 6 in the second with an accuracy of 93.3%, and the third highest validation result was with 10 in the first dense layer, and 8 in the second.

The number of neurons in the first and second dense layer will from here on in this section be abbreviated as (number in first dense layer, number in second dense layer), i.e. (a, b), where a and b are the numbers in the dense/fully connected layers.

The validation results for the LOCF imputation was good for layers (10, 2) and (6, 4), where the result was 86.9% and 84.9%, respectively, but the best result was found for (10, 10) with an validation accuracy of 94.4. Second best result was (10, 6) with an validation accuracy of 90.8. The third best result was shared between (10, 4) and (10, 8) with an accuracy of 90.5%, but as one wants the simplest network with the least possible weights for tuning, (10, 4) was chosen for testing.

First good results for the mean imputation can be found for layers (10, 2) and (6, 4) with validation accuracy of 87.4% and 85.8%, respectively. The best validation result for this imputation method was (10, 10) with an accuracy of 94.1%. The second and third best validation results were respectively (10, 6) and (10, 8) with accuracies 93.6% and 93.3%.

Median imputation gave the first good validation results for (8, 2) and (6, 4) with 84.1% and 83.0% in that order. The best result was for (10, 6) where the result was 94.7%. Next best validation result for median imputation after that was (10, 4) with 93.3% followed by (8, 8) having 93.0%.

Minimum imputation has the second best validation result among all the imputation methods for the TCN with the uwave data at 95.8% with (10, 6). Next best for the minimum imputation is (10, 10) with 95.3%, followed by (10, 8) having 95.0% accuracy. The good results started at (6, 2) and (4, 4) for this imputation method with accuracy 84.6% and 82.1%

Maximum imputation starts getting good results with (8, 2) at 85.8% and (6, 4) with 85.2%. For this imputation method the best validation was (10, 10) at 94.4%, with (10, 8) following at 94.1 and (10, 6) with 92.7% after that.

Combined imputation has the highest validation result with 96.1% for (10, 8). The next best (a tie with the best for minimum imputation) is (10, 10) with 95.8%. This is

followed next by (10, 6) at 95.0%. The earliest good validation results can be seen for (8, 2) (same as maximum, median, and zero imputation) at 88.5% and (6, 4) (same as maximum, median, LOCF and zero imputation).

It is possible that increasing the number of neurons will give even better validation results, but that takes more processing time, and one can see that limiting the parameter search for the number of neurons in the dense layers to 10 already provides potentially good results.

Test results in table 5.7 come from re-training the network from start using the number of neurons in the dense layers found during validation. This results in different initialisation values for the weights. Same as with the SSI data, although slightly better, the good results in validation did not transfer properly over to the test sets. This can be seen by the low test set results compared to the validation set results, where e.g the zero-imputed data had a validation accuracy of 94.7% with (10, 10), while the test result for the same configuration had an accuracy of $57.4 \pm 6.5\%$. Although not giving as low results, this same discrepancy repeats itself for the other configurations, and with a high standard deviation. The only reason the author can think of for this is that although some test results are about equal to the accuracy during validation, they get overshadowed by bad initialisation, which causes the TCN to get stuck in a local minimum of its loss function and unable to get out. Possible solutions are proposed in section 6.1.

The best imputation method for this instance of uwave data with missing values was found to be the combined imputation method with F_1 score $86.1 \pm 9.0\%$.

Table 5.7: Median values and results with uncertainty for test sets used for TCN classifier on the uwave data with synthetic missingness. Number of neurons in the dense layers found using grid search on validation sets. The three best validation results are tested, with the best result marked in **bold**. NaN values have been ignored.

Imputation	Dense 1	Dense 2	Acc. [%]	Bal. acc. [%]	F_1 [%]
zero	10	10	57.4±6.5	57.4±6.5	68.7±16.5
	10	6	53.1±8.6	53.1±8.6	67.7±6.6
	10	8	67.4±12.1	67.4±12.1	**73.9 ± 7.7**
locf	10	10	79.7±9.4	79.7±9.5	81.5±5.5
	10	6	81.3±7.5	81.3±7.5	**82.6 ± 5.0**
	10	4	75.4±16.9	75.4±16.9	79.4±10.2
mean	10	10	78.6±9.9	78.6±9.9	82.0±5.8
	10	6	59.6±18.6	59.6±18.6	71.3±18.2
	10	8	79.2±14.5	79.2±14.5	**82.3 ± 2.4**
median	10	6	63.4±9.4	63.4±9.5	72.8±4.8
	10	4	51.8±11.8	51.8±11.8	67.5±5.9
	8	8	79.9±9.6	79.9±9.6	**81.8 ± 5.7**
minimum	10	6	54.2±10.9	54.24±10.9	67.8±8.0
	10	10	72.3±11.8	72.3±11.8	77.9±7.3
	10	8	78.8±12.3	78.8±12.3	**82.4 ± 7.7**
maximum	10	10	79.9±5.2	79.9±5.2	**83.0 ± 3.8**
	10	8	77.7±12.1	77.2±12.1	80.7±6.8
	10	6	67.9±14.8	67.9±14.8	67.9±14.8
combine	10	8	58.9±14.9	58.9±14.9	70.9±8.5
	10	10	81.0±12.2	81.0±12.2	83.4±7.4
	10	6	83.9±14.8	83.9±14.8	**86.1 ± 9.0**

Chapter 6

Conclusions

The author has used imputation to replace missing values in two data sets, one containing SSI data of 11 types of blood samples of patients over 20 days, and the other data set called uwave which contain 3D accelerometer data of several patterns made by subjects, where two patterns were selected. In the latter dataset data was stochastically removed in an informative way to simulate missing data. The DTW and Euclidean distances were computed for each data set to make distance grid matrices, and used to perform classification of the data using the kNN classifier and the SVM classifier. Furthermore, the missing data was augmented by using masking and time counting to exploit informative missingness, and used to classify the data using the same classifiers and distance methods mentioned earlier, in addition to a newer classifier called TCN, which used the augmented data.

The results of the validation and tests using the kNN classifier show that for this classifier DTW is unnecessary, and would only add to computation time, since Euclidean distance does comparable or better than it and is much faster. If the rate of missingness is too low (depends on the data set) then there is also no significant difference between using either DTW or Euclidean distances between the multivariate time series when considering the histograms in this work. The best imputation method for the kNN classifier on the SSI data was found to be the combined imputation on non-augmented data, while augmenting the data resulted in the minimum imputation being the best option. Using minimum imputation with the Euclidean distance on augmented gave the overall best result for the SSI data when using the kNN classifier. The best combination of imputation and distance type for the SVM on the SSI data was the mean-imputation on the non-augmented data.

Augmenting the data improved the result of the imputation method which did worst for the non-augmented data. It also improved the standard deviation of all the methods, and overall improved the mean F_1 score, although the mean of some performance measures worsened when compared against the corresponding method on non-augmented data for the SVM. The best imputation method varied depending on the classifier and whether or not the data was augmented, but overall the zero imputation was never declared as the best imputation for any of the combinations of distance type, augmentation and classifier.

The results of the TCN appear to be unstable. Selecting the best architecture configuration with the same number of neurons in each of the dense layers after validation did not guarantee good results. The results appear to depend on the initial values, with the weight adaption appearing to sometimes get stuck in a local minimum of the loss function during the start of the training, even after changing the learning rate. This signify that more work is needed on the TCN solution used. Due to time constraints the TCN was neither validated nor tested on non-augmented data, which makes it a bit more difficult

to compare it with the other classification methods, and it should therefore be examined further at a later time. The test results for the uwave data with missing values were better than for the SSI data. The author can think of two reasons for this; first reason is that the SSI data set is smaller with fewer samples than the uwave data set with missing data, and the second reason is that the missing rate for the SSI data is higher. It is known that deep learning methods prefer big data sets, and it is possible that the TCN would do better with bigger sets of data.

6.1 Further Work

There are some things that could be improved upon. A possibility for further work is to use the methods in this work as reference methods and compare them to more advanced methods. This includes some imputation methods like the one used in [119], which performs Gaussian imputation in the training stage. Using a Gaussian imputation would practically increase the size of the data sets (and it is known that deep learning methods need large data sets), which might help with the unstable results encountered with the TCN in this work. The TCN could also be improved in both interpretability and accuracy by e.g. using attention weights, similar to what has been done in [120]. Also, there is a need to narrow down the random initial parameters due to the random nature of the validation and test results observed during experimentation. An alternative would be to do pre-training and apply transfer learning [121] from a TCN with good initialization and use it to classify or predict on the other imputed data (augmented and/or non-augmented). Transfer learning has been applied with TCNs to predict data sets earlier (e.g. [122, 123]) and can improve generalisation for the network. A method for learning rate decay with some kind of scheduling for perturbing the gradient could also maybe help, similar to what has been done in [124], where they add perturbation to the Adam optimizer. It would also be interesting to examine at a later time if the TCN remains unstable when the data are not augmented, and if so, what improvements or changes could be made to the augmentation process.

An ensemble of the machine learning methods used in this work is also something that might improve results, as ensembles of weak learners are known to create a strong learner when combined [62].

Appendix A

A.1 Word list

A.1.1 Biology

[125]

- **Albumin:** Transport protein. Transport several types of steroid hormones, and the thyroid hormones.
- **Amylase:** Mainly slices starch into maltose (sugar), which consist of two glucose molecules. Found in pancreatic juice, and saliva.
- **Carbamide:** Urea.
- **Creatinine:** Has something to do with kidney function and how well it function.
- **CRP:** C-reactive protein. Increases substantially (more than a hundred-fold) with bacterial infection. Increase less if virus infection. Can be used to find out if infection is from bacteria or virus.
- **Glucose:** Monosaccharide used as primary energy source in the body's cells. Commonly called blood sugar. People with Diabetes mellitus have difficulty keeping glucose levels down to normal levels when suffering infection and inflammation. Physical and mental stress may also either increase or lower glucose levels.
- **Hemoglobin:** Molecule that transport O_2 on erythrocytes (red blood cells).
- **Leukocytes:** White blood cells. Part of the immune system. Phages. Several sub-types exist.
- **Potassium (K):** Mineral. Ion. Affects nerve function and muscle contraction together with Potassium and Magnesium. Transfer through cells by ion pumps. Regulated by kidney.
- **Sodium (^{23}Na):** Mineral. Ion. Affects nerve function and muscle contraction together with Potassium and Magnesium. Transfer through cells by ion pumps. Higher concentration increase fluid amount in body tissue.
- **Thrombocytes**: Platelets in the blood. Have the main function of clotting the blood to physically stop bleeding in blood veins and arteries, as long as the rupture is not too big. It is a major component in hemostasis.

A.1.2 Some abbreviations:

- **AR**: Auto-Regressive
- **ARMA**: Auto-Regressive Moving Average
- **ARIMA**: Auto-Regressive Integrated Moving Average
- **CM**: Confusion Matrix
- **DTW**: Dynamic Time Warp
- **ED**: Euclidean distance
- **FN**: False Negative
- **FP**: False Positive
- **kNN**: k Nearest Neighbor
- **LOCF:** Last Observation Carried Forward
- **MAR**: Missing At Random
- **MCAR**: Missing Completely At Random
- **MNAR**: Missing Not At Random
- **SSI**: Surgical Site Infection
- **TP**: True Positive
- **TCN**: Temporal Convolution Network
- **TN**: True Negative
- **TNR**: True Negative Rate
- **TPR**: True Positive Rate
- **SVM**: Support Vector Machine

Appendix B

TCN dense layer validation

Resulting values after doing validation on the number of neurons in the dense layers of the TCN shown in figure 3.3 and 3.4. Values are rounded to 3 digits and the value 0 means that combination has been ignored. Vertical axis is the number of neurons in the first dense layer, and the horizontal axis is for the second dense layer. Each validation run is independent from each other and starts with training from start.

B.1 SSI data

Each cell show the median after 20 runs with randomized sets without replacement and early stopping with maximum best epoch. Limit of tolerance for early stopping was set to 20 epochs. Several combinations gave the same results within their imputation methods. The tables are too large to give standard deviation.

Figure B.1: Validation accuracy of different combinations of dense layers in TCN for the zero-imputed augmented data shown in figure 3.3(a).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0.75	0	0	0	0	0	0	0	0	0
2	0	0.75	0.761	0	0	0	0	0	0	0	0
3	0	0.75	0.754	0.894	0	0	0	0	0	0	0
4	0	0.75	0.856	0.884	0.905	0	0	0	0	0	0
5	0	0.75	0.894	0.887	0.908	0.894	0	0	0	0	0
6	0	0.75	0.901	0.901	0.908	0.894	0.901	0	0	0	0
7	0	0.75	0.905	0.898	0.898	0.894	0.901	0.901	0	0	0
8	0	0.75	0.912	0.901	0.912	0.908	0.905	0.912	0.923	0	0
9	0	0.75	0.908	0.908	0.923	0.905	0.912	0.901	0.919	0.923	0
10	0	0.75	0.923	0.923	0.901	0.915	0.915	0.923	0.919	0.915	0.919

Figure B.2: Validation accuracy of different combinations of dense layers in TCN for the locf-imputed augmented data shown in figure 3.3(b).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0.75	0	0	0	0	0	0	0	0	0
2	0	0.75	0.761	0	0	0	0	0	0	0	0
3	0	0.75	0.750	0.838	0	0	0	0	0	0	0
4	0	0.75	0.820	0.863	0.849	0	0	0	0	0	0
5	0	0.75	0.849	0.838	0.849	0.863	0	0	0	0	0
6	0	0.75	0.824	0.835	0.849	0.859	0.845	0	0	0	0
7	0	0.75	0.842	0.845	0.856	0.852	0.859	0.863	0	0	0
8	0	0.75	0.845	0.831	0.852	0.856	0.870	0.859	0.863	0	0
9	0	0.75	0.835	0.859	0.849	0.852	0.852	0.859	0.873	0.870	0
10	0	0.75	0.863	0.866	0.863	0.873	0.870	0.866	0.852	0.870	0.863

Figure B.3: Validation accuracy of different combinations of dense layers in TCN for the mean-imputed augmented data shown in figure 3.3(c).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0.75	0	0	0	0	0	0	0	0	0
2	0	0.75	0.761	0	0	0	0	0	0	0	0
3	0	0.75	0.817	0.831	0	0	0	0	0	0	0
4	0	0.75	0.827	0.831	0.835	0	0	0	0	0	0
5	0	0.75	0.827	0.845	0.845	0.845	0	0	0	0	0
6	0	0.75	0.852	0.838	0.859	0.845	0.852	0	0	0	0
7	0	0.75	0.849	0.842	0.856	0.863	0.859	0.863	0	0	0
8	0	0.75	0.856	0.849	0.852	0.845	0.852	0.856	0.856	0	0
9	0	0.75	0.863	0.845	0.859	0.856	0.856	0.866	0.873	0.866	0
10	0	0.75	0.873	0.866	0.859	0.866	0.859	0.863	0.877	0.870	0.873

Figure B.4: Validation accuracy of different combinations of dense layers in TCN for the median-imputed augmented data shown in figure 3.3(d).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0.75	0	0	0	0	0	0	0	0	0
2	0	0.75	0.750	0	0	0	0	0	0	0	0
3	0	0.75	0.810	0.849	0	0	0	0	0	0	0
4	0	0.75	0.842	0.838	0.849	0	0	0	0	0	0
5	0	0.75	0.838	0.845	0.831	0.859	0	0	0	0	0
6	0	0.75	0.852	0.852	0.845	0.842	0.856	0	0	0	0
7	0	0.75	0.863	0.849	0.859	0.863	0.852	0.859	0	0	0
8	0	0.75	0.856	0.859	0.859	0.849	0.863	0.870	0.866	0	0
9	0	0.75	0.863	0.852	0.863	0.859	0.866	0.870	0.866	0.866	0
10	0	0.75	0.852	0.870	0.870	0.873	0.866	0.859	0.856	0.870	0.863

Figure B.5: Validation accuracy of different combinations of dense layers in TCN for the minimum-imputed augmented data shown in figure 3.3(e).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0.75	0	0	0	0	0	0	0	0	0
2	0	0.75	0.757	0	0	0	0	0	0	0	0
3	0	0.75	0.803	0.838	0	0	0	0	0	0	0
4	0	0.75	0.838	0.842	0.835	0	0	0	0	0	0
5	0	0.75	0.785	0.852	0.849	0.824	0	0	0	0	0
6	0	0.75	0.845	0.845	0.845	0.852	0.838	0	0	0	0
7	0	0.75	0.838	0.845	0.845	0.852	0.849	0.842	0	0	0
8	0	0.75	0.849	0.849	0.845	0.859	0.852	0.842	0.859	0	0
9	0	0.75	0.859	0.835	0.838	0.849	0.856	0.849	0.866	0.863	0
10	0	0.75	0.863	0.856	0.856	0.849	0.852	0.859	0.859	0.859	0.856

Figure B.6: Validation accuracy of different combinations of dense layers in TCN for the maximum-imputed augmented data shown in figure 3.3(f).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0.75	0	0	0	0	0	0	0	0	0
2	0	0.75	0.757	0	0	0	0	0	0	0	0
3	0	0.75	0.820	0.845	0	0	0	0	0	0	0
4	0	0.75	0.842	0.859	0.835	0	0	0	0	0	0
5	0	0.75	0.838	0.827	0.835	0.849	0	0	0	0	0
6	0	0.75	0.831	0.838	0.866	0.852	0.856	0	0	0	0
7	0	0.75	0.863	0.863	0.870	0.866	0.866	0.866	0	0	0
8	0	0.75	0.870	0.859	0.859	0.866	0.870	0.880	0.880	0	0
9	0	0.75	0.873	0.887	0.870	0.870	0.870	0.884	0.887	0.891	0
10	0	0.75	0.880	0.887	0.894	0.887	0.8870	0.880	0.891	0.891	0.894

Figure B.7: Validation accuracy of different combinations of dense layers in TCN for the combination-imputed augmented data shown in figure 3.3(g).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0.75	0	0	0	0	0	0	0	0	0
2	0	0.75	0.757	0	0	0	0	0	0	0	0
3	0	0.75	0.761	0.842	0	0	0	0	0	0	0
4	0	0.75	0.827	0.838	0.838	0	0	0	0	0	0
5	0	0.75	0.842	0.842	0.852	0.842	0	0	0	0	0
6	0	0.75	0.842	0.852	0.856	0.856	0.852	0	0	0	0
7	0	0.75	0.856	0.842	0.866	0.859	0.859	0.863	0	0	0
8	0	0.75	0.856	0.859	0.859	0.859	0.866	0.873	0.870	0	0
9	0	0.75	0.870	0.859	0.870	0.870	0.863	0.852	0.866	0.880	0
10	0	0.75	0.873	0.859	0.866	0.863	0.863	0.873	0.866	0.870	0.870

B.2 Uwave data with synthetic missing data

Each cell show the median after 10 runs with randomized sets without replacement and early stopping with maximum best epoch. Limit of tolerance for early stopping was set to 10 epochs. Several combinations gave the same results within their imputation methods. The tables are too large to give standard deviation.

Figure B.8: Validation accuracy of different combinations of dense layers in TCN for the zero-imputed augmented data shown in figure 3.4(a).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0.542	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.542	0	0.788	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0.525	0	0.849	0	0.863	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0.860	0	0.885	0	0.883	0	0.916	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0.858	0	0.911	0	0.933	0	0.930	0	0.947

Figure B.9: Validation accuracy of different combinations of dense layers in TCN for the LOCF-imputed augmented data shown in figure 3.4(b).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0.514	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.525	0	0.670	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0.589	0	0.849	0	0.813	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0.547	0	0.835	0	0.877	0	0.883	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0.869	0	0.905	0	0.908	0	0.905	0	0.944

Figure B.10: Validation accuracy of different combinations of dense layers in TCN for the mean-imputed augmented data shown in figure 3.4(c).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0.567	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.522	0	0.763	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0.774	0	0.858	0	0.841	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0.578	0	0.874	0	0.863	0	0.899	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0.874	0	0.908	0	0.936	0	0.933	0	0.941

Figure B.11: Validation accuracy of different combinations of dense layers in TCN for the median-imputed augmented data shown in figure 3.4(d).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0.511	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.559	0	0.723	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0.668	0	0.830	0	0.863	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0.841	0	0.869	0	0.908	0	0.930	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0.919	0	0.933	0	0.947	0	0.916	0	0.927

Figure B.12: Validation accuracy of different combinations of dense layers in TCN for the minimum-imputed augmented data shown in figure 3.4(e).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0.520	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.561	0	0.821	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0.846	0	0.866	0	0.877	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0.863	0	0.897	0	0.891	0	0.911	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0.531	0	0.933	0	0.958	0	0.950	0	0.953

Figure B.13: Validation accuracy of different combinations of dense layers in TCN for the maximum-imputed augmented data shown in figure 3.4(f).

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0.517	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.592	0	0.709	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0.785	0	0.852	0	0.872	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0.858	0	0.880	0	0.899	0	0.894	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0.905	0	0.908	0	0.927	0	0.941	0	0.944

Figure B.14: Validation accuracy of different combinations of dense layers in TCN for the combination-imputed augmented data shown in figure 3.4(g).

	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0.528	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.539	0	0.612	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0.598	0	0.858	0	0.883	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0.885	0	0.919	0	0.941	0	0.941	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0.584	0	0.925	0	0.950	0	0.961	0	0.958

Appendix C

Bibliography

[1] Robert H Shumway and David S Stoffer. *Time series analysis and its applications: with R examples*. Springer, 2017.

[2] Wikipedia contributors. Time series — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Time_series&oldid=977544968, 2020. [Online; accessed 27-September-2020].

[3] Konstantinos Koutroumbas and Sergios Theodoridis. *Pattern Recognition 4th Edition*. Academic Press, 2008.

[4] Anthony J. Bagnall, Aaron Bostrom, James Large, and Jason Lines. The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version. *CoRR*, abs/1602.01711, 2016.

[5] Young-Seon Jeong, Myong K. Jeong, and Olufemi A. Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44(9):2231 – 2240, 2011. Computer Analysis of Images and Patterns.

[6] Pierre-François Marteau. Time warp edit distance with stiffness adjustment for time series matching. *IEEE transactions on pattern analysis and machine intelligence*, 31(2):306–318, 2008.

[7] A. Stefan, V. Athitsos, and G. Das. The move-split-merge metric for time series. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1425–1438, 2013.

[8] Gustavo EAPA Batista, Eamonn J Keogh, Oben Moses Tataw, and Vinicius MA De Souza. Cid: an efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery*, 28(3):634–669, 2014.

[9] Eamonn Keogh and Michael Pazzani. Derivative dynamic time warping. *First SIAM International Conference on Data Mining*, 1, 01 2002.

[10] Tomasz Górecki and Maciej Łuczak. Using derivatives in time series classification. *Data Mining and Knowledge Discovery*, 26:310–331, 03 2013.

[11] Jessica Lin and Yuan Li. Finding structural similarity in time series data using bag-of-patterns representation. In Marianne Winslett, editor, *Scientific and Statistical Database Management*, pages 461–477, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[12] Pavel Senin and Sergey Malinchik. "sax-vsm: interpretable time series classification using sax and vector space model". 12 2013.

[13] Patrick Schäfer. The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29, 11 2015.

[14] Rohit Kate. Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery*, 30, 05 2015.

[15] Thanawin Rakthanmanon and Eamonn Keogh. *Fast Shapelets: A Scalable Algorithm for Discovering Time Series Shapelets*, pages 668–676. 05 2013.

[16] Aaron Bostrom and Anthony J. Bagnall. A shapelet transform for multivariate time series classification. *CoRR*, abs/1712.06428, 2017.

[17] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Learning time-series shapelets. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08 2014.

[18] Houtao Deng, George C. Runger, Eugene Tuv, and Vladimir Martyanov. A time series forest for classification and feature extraction. *CoRR*, abs/1302.2277, 2013.

[19] Mustafa Gokce Baydogan, George Runger, and Eugene Tuv. A bag-of-features framework to classify time series. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2796–2802, September 2013.

[20] Mustafa Baydogan and George Runger. Time series representation and similarity based on local autopatterns. *Data Mining and Knowledge Discovery*, 30:1–34, 07 2015.

[21] Klaus-Robert Müller, Alexander Smola, Gunnar Rätsch, Bernhard Schölkopf, J. Kohlmorgen, and V. Vapnik. *Using support vector machines for time series prediction*, pages 243–253. 01 1999.

[22] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with cote: The collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27:1–1, 09 2015.

[23] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering–a decade review. *Information Systems*, 53:16–38, 2015.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[25] Alejandro Pasos Ruiz, Michael Flynn, and Anthony Bagnall. Benchmarking multivariate time series classification algorithms. *arXiv preprint arXiv:2007.13156*, 2020.

[26] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *CoRR*, abs/1809.04356, 2018.

[27] Wikipedia contributors. Multilayer perceptron — Wikipedia, the free encyclopedia. "https://en.wikipedia.org/w/index.php?title=Multilayer_perceptron&oldid=1013769694", 2021. [Online; accessed 15-May-2021].

[28] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[29] W. Dai, C. Dai, S. Qu, J. Li, and S. Das. Very deep convolutional neural networks for raw waveforms. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 421–425, 2017.

[30] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[31] Thach Le Nguyen, Severin Gsponer, Iulia Ilie, Martin O'Reilly, and Georgiana Ifrim. Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. *Data Mining and Knowledge Discovery*, 33(4):1183–1222, 2019.

[32] Omer Berat Sezer and Ahmet Murat Ozbayoglu. Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. *Applied Soft Computing*, 70:525–538, 2018.

[33] Mantas Lukoševicius, Dan Popovici, Herbert Jaeger, Udo Siewert, and Residence Park. Time warping invariant echo state networks. *International University Bremen, Technical Report*, 2006.

[34] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

[35] Jason Lines, Sarah Taylor, and Anthony Bagnall. Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1041–1046. IEEE, 2016.

[36] Gian Antonio Susto, Angelo Cenedese, and Matteo Terzi. Time-series classification methods: Review and applications to power systems data. In *Big data application in power systems*, pages 179–220. Elsevier, 2018.

[37] K-R Müller, Alexander J Smola, Gunnar Rätsch, Bernhard Schölkopf, Jens Kohlmorgen, and Vladimir Vapnik. Predicting time series with support vector machines. In *International Conference on Artificial Neural Networks*, pages 999–1004. Springer, 1997.

[38] Himani Bhavsar and Mahesh H Panchal. A review on support vector machine for data classification. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 1(10):185–189, 2012.

[39] Wikipedia contributors. Curse of dimensionality — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Curse_of_dimensionality&oldid=970064949, 2020. [Online; accessed 22 June 2020].

[40] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In *International work-conference on artificial neural networks*, pages 758–770. Springer, 2005.

[41] Wikipedia contributors. Principal component analysis — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Principal_component_analysis&oldid=1022935748, 2021. [Online; accessed 15-May-2021].

[42] Wikipedia contributors. Linear discriminant analysis — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Linear_discriminant_analysis&oldid=1021740456, 2021. [Online; accessed 15-May-2021].

[43] Wikipedia contributors. T-distributed stochastic neighbor embedding — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=T-distributed_stochastic_neighbor_embedding&oldid=1020310146, 2021. [Online; accessed 15-May-2021].

[44] Steinn Gudmundsson, Thomas Philip Runarsson, and Sven Sigurdsson. Support vector machines and dynamic time warping for time series. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 2772–2776. IEEE, 2008.

[45] Wikipedia contributors. Dynamic time warping — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Dynamic_time_warping&oldid=979506999, 2020. [Online; accessed 19 Apr 2020].

[46] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[47] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:, 1994.

[48] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[49] Donald B Rubin. *Multiple imputation for nonresponse in surveys*, volume 81. John Wiley & Sons, 2004.

[50] Stef Van Buuren. *Flexible imputation of missing data*. CRC press, 2018.

[51] Joseph L Schafer and John W Graham. Missing data: our view of the state of the art. *Psychological methods*, 7(2):147, 2002.

[52] Kazunori Akiyama, Antxon Alberdi, Walter Alef, Keiichi Asada, Rebecca Azulay, Anne-Kathrin Baczko, David Ball, Mislav Baloković, John Barrett, Dan Bintley, et al. First m87 event horizon telescope results. iv. imaging the central supermassive black hole. *The Astrophysical Journal Letters*, 875(1):L4, 2019.

[53] Wikipedia contributors. Ligo — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=LIGO&oldid=1023611511, 2021. [Online; accessed 2-June-2021].

[54] Zhen Hu, Genevieve B Melton, Elliot G Arsoniadis, Yan Wang, Mary R Kwaan, and Gyorgy J Simon. Strategies for handling missing clinical data for automated surgical site infection detection from the electronic health record. *Journal of biomedical informatics*, 68:112–120, 2017.

[55] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.

[56] Stack Overflow forum members. "knn: training, and validation". https://stackoverflow.com/questions/10814731/knn-training-testing-and-validation, 2012. Accessed 27 May 2021.

[57] Amaia Abanda, Usue Mori, and Jose A Lozano. A review on distance based time series classification. *Data Mining and Knowledge Discovery*, 33(2):378–412, 2019.

[58] Stack Overflow users. Value of k in k nearest neighbor algorithm. https://stackoverflow.com/questions/11568897/value-of-k-in-k-nearest-neighbor-algorithm, 2012. [Online].

[59] STOR 390. Putting the k in k nearest neighbors. https://idc9.github.io/stor390/notes/cross_validation/cross_validation.html#knn_for_different_values_of_k, 2020. [Online].

[60] Wikipedia contributors. K-nearest neighbors algorithm — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=K-nearest_neighbors_algorithm&oldid=979506830, 2020. [Online; accessed 15 June 2020].

[61] Wikipedia contributors. Support vector machine — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=980412092, 2020. [Online; accessed 15 June 2020].

[62] Ian Goodfellow, Yoshua Bengio, , and Aaron Courville. *"Deep Learning.* The MIT Press, 2016.

[63] Wikipedia contributors. Kernel method — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Kernel_method&oldid=1012624126, 2021. [Online; accessed 21-May-2021].

[64] Wikipedia contributors. Radial basis function kernel — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Radial_basis_function_kernel&oldid=1010095405, 2021. [Online; accessed 21-May-2021].

[65] Wikipedia contributors. Hyperparameter optimization — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Hyperparameter_optimization&oldid=972537954, 2020. [Online; accessed 15 June 2020].

[66] Wikipedia contributors. Bayesian optimization — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Bayesian_optimization&oldid=959629080, 2020. [Online; accessed 24 June 2020].

[67] Wikipedia contributors. Probabilistic classification — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Probabilistic_classification&oldid=917869051, 2019. [Online; accessed 23 June 2020].

[68] Kristin Bennett, Ayhan Demiriz, et al. Semi-supervised support vector machines. *Advances in Neural Information processing systems*, pages 368–374, 1999.

[69] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks for action segmentation and detection. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 156–165, 2017.

[70] Terrence J Sejnowski and Charles R Rosenberg. Parallel networks that learn to pronounce english text. *Complex systems*, 1(1):145–168, 1987.

[71] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[72] Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. "phoneme recognition using time-delay neural networks". *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339, 1989.

[73] Isma Hadji and Richard P Wildes. What do we understand about convolutional networks? *arXiv preprint arXiv:1803.08834*, 2018.

[74] Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.

[75] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[76] Wikipedia contributors. Convolutional neural network — Wikipedia, the free encyclopedia, 2021. [Online; accessed 25-March-2021].

[77] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[78] Yale Chang, Jonathan Rubin, Gregory Boverman, Shruti Vij, Asif Rahman, Annamalai Natarajan, and Saman Parvaneh. A multi-task imputation and classification neural architecture for early prediction of sepsis from multivariate clinical time series. In *2019 Computing in Cardiology (CinC)*, pages Page–1. IEEE, 2019.

[79] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[81] Rahul Dey and Fathi M Salemt. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

[82] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[83] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89, pages 89–94. Presses universitaires de Louvain, 2015.

[84] Mathias Lechner, Ramin Hasani, Radu Grosu, Daniela Rus, and Thomas A Henzinger. Adversarial training is not ready for robot learning. *arXiv preprint arXiv:2103.08187*, 2021.

[85] James H. McClellan, Ronald W. Schafer, and Mark A. Yoder. *DSP First*. Pearson, 2017.

[86] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson, 2010.

[87] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and K Lang. Phoneme recognition: neural networks vs. hidden markov models vs. hidden markov models. In *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, pages 107–110. IEEE, 1988.

[88] Kevin J Lang, Alex H Waibel, and Geoffrey E Hinton. A time-delay neural network architecture for isolated word recognition. *Neural networks*, 3(1):23–43, 1990.

[89] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[90] Wikipedia contributors. Gated recurrent unit — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Gated_recurrent_unit&oldid=997015931, 2020. [Online; accessed 19-May-2021].

[91] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks*, 20(3):335–352, 2007.

[92] Michael C Mozer. Induction of multiscale temporal structure. In *Advances in neural information processing systems*, pages 275–282, 1992.

[93] Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pages 493–499, 1996.

[94] Tsungnan Lin, Bill G Horne, Peter Tino, and C Lee Giles. Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996.

[95] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[96] Nikolas Adaloglou. "intuitive explanation of skip connections in deep learning". https://theaisummer.com/skip-connections/, 2020.

[97] Paperswithcode.Com. "papers with code - layer normalization explained". https://paperswithcode.com/method/layer-normalization, 2021. Accessed 9 Apr 2021.

[98] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[99] Philippe Remy. Temporal convolutional networks for keras. https://github.com/philipperemy/keras-tcn, 2020.

[100] Julius Richter. Temporal convolutional networks for sequence modeling. https://dida.do/blog/temporal-convolutional-networks-for-sequence-modeling, 2020. published January 2020, last checked March 2021.

[101] Kouichi Yamaguchi, Kenji Sakamoto, Toshio Akabane, and Yoshiji Fujimoto. A neural network for speaker-independent isolated word recognition. In *First International Conference on Spoken Language Processing*, 1990.

[102] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[103] Sicara Chazareix. "about convolutional layer and convolution kernel". https://www.sicara.ai/blog/2019-10-31-convolutional-layer-convolution-kernel, 2020. Accessed 20 May 2021.

[104] Irhum Shafkat. "intuitively understanding convolutions for deep learning". "https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1, 2018. Accessed 20 May 2021.

[105] "kernel size - an overview". https://www.sciencedirect.com/topics/engineering/kernel-size, 2021. Accessed 20 May 2021. Short quote on kernel size.

[106] Francesco Lässig. "temporal convolutional networks and forecasting". https://medium.com/unit8-machine-learning-publication/temporal-convolutional-networks-and-forecasting-5ce1b6e97ce4, 2020. Accessed 14 Apr 2021.

[107] Min Xia, Yiqing Xu, Ke Wang, Xu Zhang, et al. Dilated residual attention network for load disaggregation. *Neural Computing and Applications*, 31(12):8931–8953, 2019.

[108] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[109] Karl Øyvind Mikalsen, Cristina Soguero-Ruiz, Filippo Maria Bianchi, Arthur Revhaug, and Robert Jenssen. An unsupervised multivariate time series kernel approach for identifying patients with surgical site infection from blood samples. *arXiv preprint arXiv:1803.07879*, 2018.

[110] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.

[111] scikit-learn developers. Support vector machines - scikit-learn 0.23.1 documentation. https://scikit-learn.org/stable/modules/svm.html#svm-kernels, 2020. [Online; accessed 15 June 2020].

[112] Wikipedia contributors. Hinge loss — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Hinge_loss&oldid=1010383095, 2021. [Online; accessed 25-May-2021].

[113] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[114] Wikipedia contributors. Random search — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Random_search&oldid=1002913588, 2021. [Online; accessed 16-June-2021].

[115] Wikipedia contributors. Digestive system surgery — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Digestive_system_surgery&oldid=928290845, 2019. [Online; accessed 29-May-2021].

[116] Minoti V Apte, Jeremy S Wilson, and Mark A Korsten. Alcohol-related pancreatic damage: mechanisms and treatment. *Alcohol health and research world*, 21(1):13, 1997.

[117] Henny H Billett. Hemoglobin and hematocrit. *Clinical Methods: The History, Physical, and Laboratory Examinations. 3rd edition*, 1990.

[118] Wikipedia contributors. Definite symmetric matrix — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Definite_symmetric_matrix&oldid=985186753, 2020. [Online; accessed 25-October-2020].

[119] Michael "Moor, Max Horn, Bastian Rieck, Damian Roqueiro, and Karsten" Borgwardt. "temporal convolutional networks and dynamic time warping can drastically improve the early prediction of sepsis". In *Proceedings of the 4th Machine Learning for Healthcare Conference*, 2019.

[120] Lei Lin, Beilei Xu, Wencheng Wu, Trevor W Richardson, and Edgar A Bernal. Medical time series classification with hierarchical attention-based temporal convolutional networks: A case study of myotonic dystrophy diagnosis. In *CVPR Workshops*, pages 83–86, 2019.

[121] Wikipedia contributors. Transfer learning — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Transfer_learning&oldid=1022394104, 2021. [Online; accessed 16-June-2021].

[122] Ananth Bhimireddy, Priyanshu Sinha, Bolu Oluwalade, Judy Wawira Gichoya, and Saptarshi Purkayastha. Blood glucose level prediction as time-series modeling using sequence-to-sequence neural networks. In *CEUR Workshop Proceedings*, 2020.

[123] Rui Ye and Qun Dai. Implementing transfer learning across different datasets for time series forecasting. *Pattern Recognition*, 109:107617, 2021.

[124] Mohammad Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. In *International Conference on Machine Learning*, pages 2611–2620. PMLR, 2018.

[125] Olav Sand and Øystein V. Sjaastad and Egil Haug and Jan G. Bjålie. *Menneskekroppen - fysiologi og anatomi*. Gyldendal Akademisk, 2016.