University of Warwick

BEng Individual Project
School of Engineering

Distanced-Based Approaches for Classification of Neocortical Neurons

Author:
Lucia Barrios Vidal

Supervisor:
Prof. Adam Noel

Acknowledgments

I would like to show my appreciation and gratitude to Professor Adam Noel for his enthusiasm in supervising this project. He has offered me guidance and feedback throughout the academic year that has been of great value to the end result.
I wish to extend my special thanks to my family and friends for their exceptional support and encouragement in all my efforts and endeavours.

Abstract

Neocortical network behaviours and functions arise from firing patterns defined by the electrophysiological properties of individual neurons. Identification of neuronal cell types according to their firing behaviour is therefore critical for understanding functional properties within a network. Taking inspiration from the fast development of machine learning tools, we explore the use of a range of supervised classification methods to identify five of the most fundamental classes of firing patterns observed in the mammalian neocortex. We take this one step further to explore if recent advancements in time series classification can enhance the performance of traditional classifiers. We approach this analysis by generating synthetic data representative of the dynamics of real neocortical neurons in order to train k-nearest neighbour classifiers, benchmarking their performance against selected metrics.

We build a framework that allows for the automatic classification of five neocortical electrophysiological classes with 90% accuracy, using Dynamic Time Warping as the distance metric to a k-nearest neighbour classifier. We further explore the performance of the models on a class-wise basis, making further suggestions on the most appropriate techniques for each task. We find that for the classification of regular spiking neurons, FastDTW proves to be as effective and more efficient than DTW, and that Euclidean distance is the most appropriate metric for identifying chattering cells when the number of recordings to be classified is large.

Contents

Acknowledgments .... **i**

Abstract .... **ii**

List of Figures .... **v**

List of Tables .... **vi**

1. Introduction **1**

1.1 Motivation 1

1.2 Objectives 2

1.3 Challenges 3

1.4 Report Structure 4

2. Background **4**

2.1 Relevant Terminology 4

2.1.1 Time Series Data 4

2.1.2 Synthetic Data 5

2.1.3 Imbalanced Classification Problem 5

2.1.4 Multivariate Normal Distribution 5

2.1.5 Time Complexity and Big O Notation 6

2.2 Supervised Learning 6

2.2.1 K-Nearest Neighbours 7

2.2.2 Dynamic Time Warping 7

2.2.3 FastDTW 8

2.2.4 Convolutional Neural Network 8

2.2.5 Decision Tree Classifier 9

2.2.6 Random Forest Classifier 9

2.3 Izhikevich Model Introduction 10

2.2 Recent Literature Review 12

2.2.1 Existing Frameworks for Neural Classification 12

2.2.2 Existing Frameworks for Time Series Classification 13

3. System Design **15**

3.1 Language Selection 15

3.1.1 IDE Selection 16

3.1.2 Data Libraries 17

3.1.3 Machine Learning Libraries 17

3.2 Data Source 17

3.2 Model Selection 18

4. Implementation............ 19
4.1 Parameter Search............ 20
4.1.1 Variance of Multivariate Distribution............ 20
4.1.2 Size of Training Data............ 23
4.2 Data............ 24
4.2.1 Data Generation............ 24
4.2.2 Dataset Structure............ 25
4.2.3 Data Pre-Processing............ 25
4.2.4 Data Splits............ 26
4.3 Model Training and Testing............ 26
4.3.1 k-NN with Euclidean Distance............ 27
4.3.2 1-NN with DTW............ 27
4.3.3 1-NN with FastDTW............ 28
5. Performance Evaluation and Conclusions............ 28
5.1 Performance Evaluation............ 29
5.1.1 Selection of Performance Metrics............ 29
5.1.2 General Performance Comparison............ 30
5.1.3 Class-Wise Performance Comparison............ 31
5.2 Conclusions............ 32
6. Conclusions and Future Work............ 33
6.1 Summary of Contributions............ 33
6.2 Future Development............ 34
6.2.1 Potential Sources of Bias............ 34
6.2.2 Broader Applicability............ 34
6.2.3 Efficiency............ 35
6.3 Final Thoughts............ 36
7. References............ 37
8. Appendices............ 41
Appendix A. Overview of the Source Code Used for Training and Testing............ 41
Appendix B. User-Defined Class............ 46

List of Figures

Figure 1: The surface plot of a standard bivariate normal distribution. 5
Figure 2: A comparison of the relationship between execution time and input size for different types of time complexity. 6
Figure 3: A diagram of the class prediction of an unseen data point (red star) depending on the k-value. 7
Figure 4: A comparison of the mapping of data points without warping (Euclidean) and with warping (DTW). 8
Figure 5: The general architecture of a CNN. 9
Figure 6: A diagram of a decision tree classifier with four features. 9
Figure 7: A comparison of the accuracy obtained using different datasets when applying Resnet (CNN) vs DTW. 14
Figure 8: A flow chart highlighting the order of steps taken throughout the implementation process. 19
Figure 13: A demonstration of the mean and covariance matrices for an RS neuron with five percent variance. 20
Figure 9: The graphs of the simulated spiking patterns of two RS (top), two IB (middle) and two CH (bottom) neurons, over 1000 milliseconds and with a sampling frequency of two milliseconds, when the parameters are drawn from a multivariate distribution with one percent variance. 21
Figure 10: The graphs of the simulated spiking pattern of a FS (top) and LTS (bottom) neuron, over 1000 milliseconds and with a sampling frequency of two milliseconds, when the parameters are drawn from a multivariate distribution with one percent variance. 21
Figure 11: The graphs of the simulated spiking patterns of two RS (top), two IB (middle) and two CH (bottom) neurons, over 1000 milliseconds and with a sampling frequency of two milliseconds when the parameters are drawn from a multivariate distribution with five percent variance. 22
Figure 12: The graphs of the simulated spiking pattern of a FS (top) and LTS (bottom) neuron, over 1000 milliseconds and with a sampling frequency of two milliseconds, when the parameters are drawn from a multivariate distribution with five percent variance. 22
Figure 14: The plots of the mean error as the number of examples for each class grows, when k=1 (left) and k=3 (right). 23
Figure 15: The plots of the mean error as the number of examples for each class grows, when k=5 (left) and k=7 (right). 23
Figure 16: A display of the dataset format used for training and testing. 25

Figure 17: A demonstration of the use of the train_test_split function for creating balanced training and testing sets. 26
Figure 18: A display of the composition of the training and testing sets........ 26
Figure 19: A plot of the mean error as the value of k increases. 27
Figure 20: A display of the classification report for k-NN when k=5 and with Euclidean as the distance metric. 27
Figure 21: A display of the classification report with DTW as the distance metric........ 28
Figure 22: A display of the classification report with FastDTW as the distance metric. 28
Figure 23: A comparison of the precision of the three distance metrics for each neuron type. 31
Figure 24: A comparison of the recall of the three distance metrics for each neuron type. 32

List of Tables

Table 1: The characteristic parameters and firing patters of the different classes. 11
Table 2: A comparison of the performance of common programming languages against selected criteria. 15
Table 3: A comparison of the average mean error obtained for each example size. 24
Table 4: The corresponding labels of each neuron class........ 25
Table 5: A comparison of the performance of each distance metric in terms of macro precision, macro recall and accuracy........ 30

Chapter 1

1. Introduction

In this first section, we will discuss the motivation behind the development of this report, its main objectives and the challenges encountered throughout. Following this, we will discuss the content and structure of the subsequent chapters.

1.1 Motivation

Traditionally, most theoretical neuroscience research has concerned the analysis of neuronal circuits and synaptic organization. Neurons were classified into excitatory and inhibitory types, but their electrophysiological characteristics were rarely addressed. In recent years however, the scientific community has become aware of the importance of these properties towards information processing [1].

Although it is very difficult to know the precise electrophysiological properties that give rise to the dynamics of each neuron [1], these parameters result in different patterns of action potentials that neurons generate in response to input currents, and a relation has been found in literature between these firing patterns and cortical function [2]. These patterns are uniform enough to allow for the definition of electrophysiological classes that capture the dynamics found in extracellular recordings of neocortical neurons, and they can be simulated through simple models that are computationally efficient and biologically plausible [3].

These electrophysiological distinctions have also morphological correlates [4], so accurate classification could be further used to gain an insight into the shape and structure of neocortical neurons.

Furthermore, large-scale spiking models of entire brain regions offer a means to investigate theories of cognitive function and allow us to gain insights on how information processing occurs in the nervous system [5]. The accurate classification of neurons found in different regions of the neocortex is critical for the development of large-scale models that are representative of real-life systems, and that consider the different dynamics of individual cells in that network.

Accurate classification of neurons according to their spiking patterns is therefore not only essential for understanding the functions of local networks, but also in order to gain an insight into neuronal morphology and building accurate large-scale spiking network models that capture the intrinsic diversity of firing patterns of individual neurons at a sufficient level of accuracy.

Given that our aim is the classification of neocortical neurons according to the similarities/ dissimilarities in their dynamics, we can see that this is a problem related to the one that supervised classification attempts to resolve by forming predictions based on the similarities between new and previously seen data points [6].

The implementation of an automatic approach to neural classification as opposed to traditional methods provides several advantages. As currently, purely manual classification is still commonly applied, which is time consuming and error and bias prone [7].

Research papers can be found that explore the efficiency and accuracy of different techniques for time series or sequence data classification [8] and others that tackle neural classification by analysing different neuron properties [9], but we have not found any that operate at the intersection of the research areas, which is what we set out to study.

1.2 Objectives

The first objective is to provide a comparison of the different techniques available in literature for classification tasks and the criteria that has influenced the selection of one over another. Considering the constant development of new classifiers, we will first evaluate the performance of state-of-the-art models against key performance indicators. Whilst we will be using a traditional classifier initially for performance benchmarking, we will later extend and optimise the models through dedicated time series algorithms for increased accuracy and efficiency.

Our second objective is that, although we will be focusing on neural classification under controlled conditions of input current and the classification of only five classes, we will try to ensure the framework is easily adaptable for further implementation. The report will focus on the application of recent developments in time series classification for this task and will look at specifically distance-based methods [10].

More specifically, our aim is to use the Izhikevich model [3] to create a dataset of time series objects containing the waveform of simulated spikes, as a measure of membrane potential over time, and a fixed set of different examples for each class. The dataset in turn, will be used for the training and testing of models that will ultimately be able to associate each time series object with the probability of belonging to each electrophysiological class.

The objective is not the development of a model that is ready to be made available to users, but the exploration, implementation, and comparison of the performance of different techniques found in literature, in order to bring forward an appropriate framework for the automatic classification of extracellular recordings of neocortical neurons based on their spiking patterns.

1.3 Challenges

There are several assumptions that must be worked with in the field of neural classification due to the complex dynamics of the brain, one of the few is the definition of the classes identified in literature and reproduced in the Izhikevich model. Although most biologists agree with the classification, many would point out it is oversimplistic, that the differences between the classes are less defined, that within each class further subclasses could be identified, and that a neuron's firing class can change depending on the state of the brain [1] . For the purposes of this research this is an assumption we must work with, and further implementation could include the extension of the identification task to other classes.

In this report, we simulate the spiking patterns of neocortical neurons in response to a step of DC current. In practice, we are not limited to this form of input, and different behaviours can be observed for other forms of input current [1].

Another assumption is that the spiking patterns to be classified have been isolated and are noise-free. In practice, most extracellular neuronal recordings are obtained through electrodes, each of which records the dynamics of various neurons at a time, and computational analysis is needed to separate their activity. Furthermore, the amplitude of the spikes decreases with the distance from the neuron to the electrode. While at an appropriate distance the spikes are large enough to be detected, further away spikes will be masked by background noise [7]. For simplicity purposes, the model developed assumes that the extracellular recordings to be predicted have been sufficiently pre-processed to be considered the spiking pattern of a well-isolated noise-free cell. Further work is suggested in section 6.2.2 to address this assumption.

Careful consideration must also be taken regarding the techniques selected to train the classifier. A commonly used, however problematic approach to time series classification, is the application of standard classifiers that treat time points as independent features. Through this approach, the algorithm will not benefit from the information contained in the time order of the data [11]. Despite this, there are algorithms available for dedicated for time series classification that have been identified and discussed throughout the report.

An additional challenge that we faced is that, in order for the classification to be accurate, the model needs to be provided with enough training data to learn from. However, this provides a trade off with the time of execution, so the implementation of further techniques is explored in order to address this.

1.4 Report Structure

Chapter 2 will provide a background for this project, covering terminology, models and techniques discussed throughout the report. Following this, we explore state-of-the-art research in the respective fields in order to be able to make an informed selection of the techniques to be implemented.

In Chapter 3, the system design is explained, describing the considerations taken for the selection of the programming language, main libraries used and models to be implemented.

In Chapter 4, an overview of the process followed for training and testing is explained, including the approach to parameter selection, data generation, visualization, and pre-processing.

Following this, in Chapter 5 we give an overview of different performance metrics and discuss which ones will be selected in order to compare the models, the findings and explanations of the results are presented, and the main conclusions presented.

Finally, in Chapter 6, we summarise the main findings and identify opportunities for further work, including the steps that can be taken to improve the quality of data, increased efficiency and broader functionality.

Chapter 2

2. Background

In this chapter, we will cover some terminology that will be used frequently throughout the report, as well as the neuron model used to generate the data.

Following this, we will cover some background about supervised learning, touching on the principles behind the algorithms and techniques that will be discussed in the literature review.

2.1 Relevant Terminology

2.1.1 Time Series Data

As a result of different combinations of parameters in the Izhikevich model [3], different spiking patterns and therefore series of membrane potential will be generated. This is an example of time series data, which is defined as "A collection of data points that are gathered at successive intervals and recorded in time order " [12] . The characteristics of time series data calls for dedicated classification algorithms that exploit the information contained in the time order of the data.

2.1.2 Synthetic Data

Synthetic data is created digitally as opposed to being obtained by direct measurement. It allows us to create equal length time series data that shares a common number of time points at regularly spaced intervals. It was chosen for the objectives of our project as the Izhikevich model is effective in reproducing spiking and bursting behaviour of known types of cortical neurons, with four parameters as variables. The choice of synthetic data also allows us to avoid the imbalanced classification problem outlined in section 2.1.3.

2.1.3 Imbalanced Classification Problem

Most machine learning models are built around the assumption of an equal number of examples for each class. Imbalanced classification occurs when there is an unequal distribution of classes in the training dataset. Which results in models that have poor predictive performance for the class with less examples [13].
Synthetic data allows us to obtain balanced dataset, with a fixed number of instances for each class, from which we can obtain our training and testing sets, minimising the risk of our model being less trained in any one class.

2.1.4 Multivariate Normal Distribution

It is a generalization of the normal distribution to higher dimensions. It is specified by a mean and covariance matrix, which indicate the values around which the distribution is centred and the spread of the distribution from these values respectively. An example of a standard bivariate normal distribution, with a mean of [0,0] and a covariance matrix of [1,0; 0,1], is shown in Figure 1 [14].
Where the covariance matrix represents a variance of 1 for two variables that are independently correlated. In this project, we will be creating a multivariate normal distribution of 4 parameters, which is more difficult to visualize but operates on the same principle.

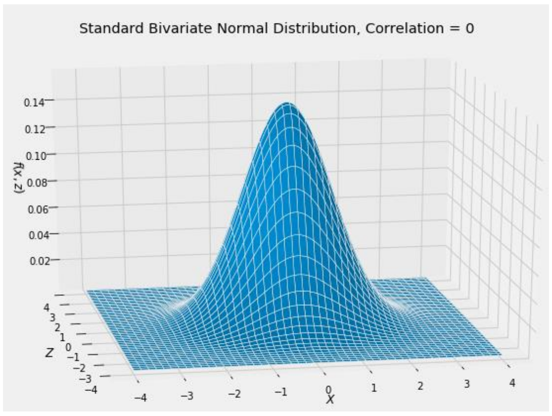


Figure 1: The surface plot of a standard bivariate normal distribution.

2.1.5 Time Complexity and Big O Notation

Time complexity describes the execution time of an algorithm as a function of the length of the input. This relation is denoted as Big O Notation O[n], where O is the order of growth and n is the length of the input. It is a measure of how efficient the algorithm is, and it will be used as a criterion for performance comparison between models throughout our research.

The main types of time complexities that will be mentioned are:

Linear time (O(n)): occurs when the time taken to run a function increases linearly with the size of the input. Seen in blue in Figure 2 [15].

Quadratic time (O(n^2)): when the time take taken to run a function increases quadratically as input grows. Seen in red in Figure 2.

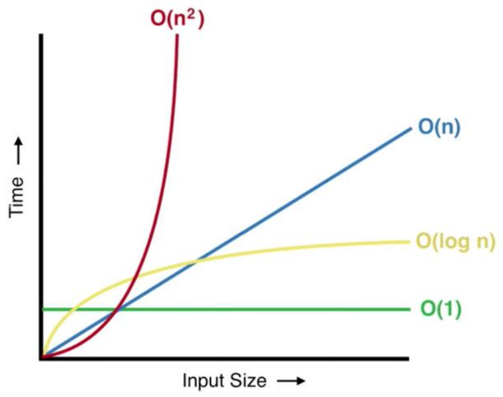


Figure 2: A comparison of the relationship between execution time and input size for different types of time complexity.

2.2 Supervised Learning

The main technique that will be explored throughout this report is supervised learning.
A supervised algorithm, as opposed to an unsupervised machine learning algorithm, relies on labelled input data to learn a mapping function that produces an appropriate output when given new unlabelled data.
Training data: Portion of the dataset used as input to the machine learning model to teach it how to recognise patterns and make predictions of unseen inputs.
Testing data: Collection of examples that do not form part of the training process and used to test the performance of the model. It allows for the comparison between different models and gives an indication of the accuracy of the model.

2.2.1 K-Nearest Neighbours

Also known as k-NN, it is a supervised learning algorithm used for classification tasks [16].
The traditional k-NN algorithm works by calculating the Pythagorean or Euclidian distance (although other metrics can be used) between the unseen data point to all other data points in the training set. It then selects the labels of the k-nearest points and assigns the label corresponding to the majority of k points to the unseen data point.

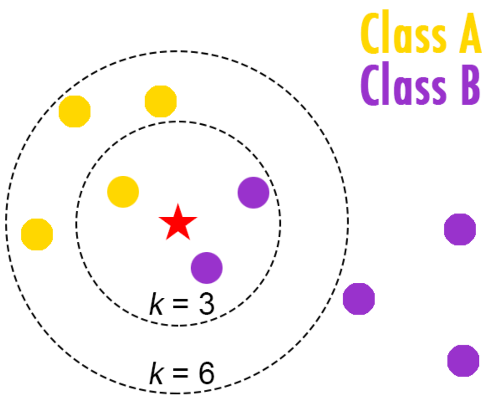
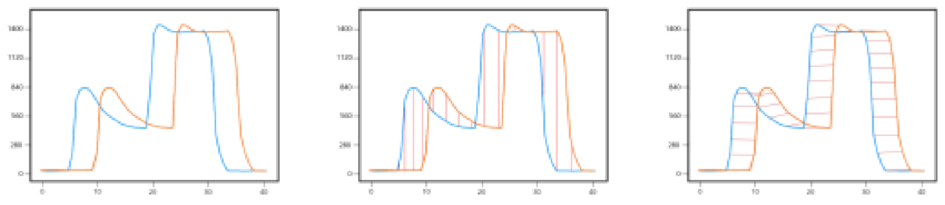


Figure 3: A diagram of the class prediction of an unseen data point (red star) depending on the k-value.

In Figure 3 [17] , the star, representing the unseen data point, would be predicted to belong to class B if k were equal to 3, and class A if k were equal to 6. In our research, the unseen data points will be vectors containing the simulated membrane potential values for each neuron. k-NN is appropriate when the data is labelled and noise-free and when we have a relatively small dataset.

2.2.2 Dynamic Time Warping

Also referred to as DTW, is a distance metric used for applying k-NN to the classification of time series data, it is normally used with a k value of 1 and referred to as 1-NN with DTW [18].
When dealing with time series data where the sequences may be displaced in time, the Euclidean distance calculated can be large when the two series have similar patterns but different time lengths or are shifted by a certain amount in the time axis, see Figure 4(a) [19]. To address this, Dynamic Time Warping allows for the comparison of data points between two time series in a non-linear fashion by warping the time axis. It fills a cost matrix, where the distance value of the corresponding data-points is contained in each cell, and then finds the shortest path through the matrix, in order to find the path that offers the minimum distance between the time series. See the user-defined class in figure B.1 for more detail.



(a) Two time series displaced in time. (b) Mapping of data-points without warping. (c) Mapping of data-points with warping.

Figure 4: A comparison of the mapping of data points without warping (Euclidean) and with warping (DTW).

When no warping is allowed (Euclidean distance), the data points between the two data series are directly compared based on the common time axis value. As seen in Figure 4(b), the method performs poorly when the time series are slightly shifted. By allowing the distance to be calculated with the flexibility of considering a warping window, DTW performs well at mapping the data-points that follow similar patterns, as observed in Figure 4(c).

2.2.3 FastDTW

An approximate algorithm to DTW which achieves linear time complexity through 3 main techniques [20]:

Coarsening: reduces the time series length while preserving the general trend of the data. This is done by averaging adjacent points in each sequence.

Projection: Finds minimum distance at a lower resolution and predicts the optimal solution for higher resolutions.

Refinement: Refines the path projected by lower resolutions.

2.2.4 Convolutional Neural Network

Also known as CNN, it is a deep learning algorithm that is able to take as input a time series and capture patterns in the data along with their relative importance for classification through a range of filters [21] . Its composed mainly of an alternation of Convolutional and Pooling layers before the Fully Connected Layer as shown in Figure 5 [21] .

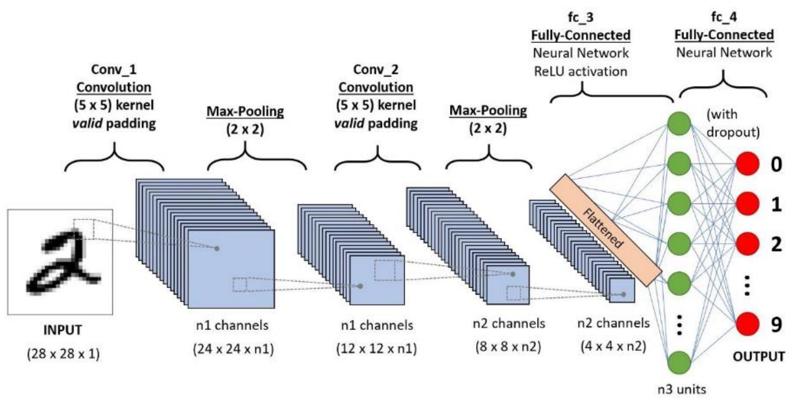


Figure 5: The general architecture of a CNN.

2.2.5 Decision Tree Classifier

A binary tree that recursively splits the dataset until it is left with data of only one type of class. The model learns what "splits" or condition rules to use in order to maximise the information gain, which is a measure of how well a given attribute separates the training examples according to their target classification. In figure 6 a decision tree [22] with 4 features or "splits" is observed.

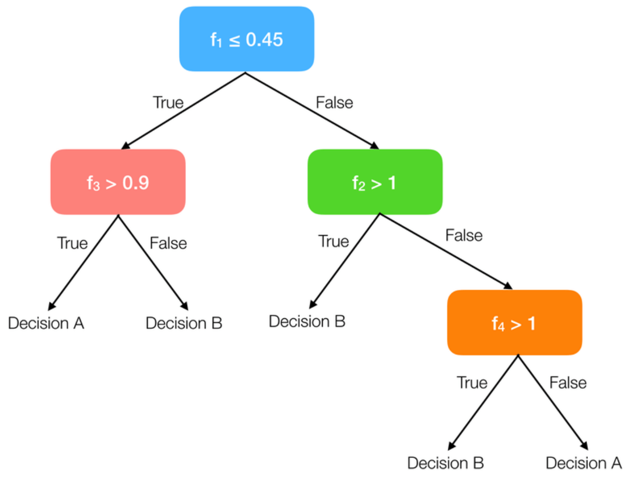


Figure 6: A diagram of a decision tree classifier with four features.

2.2.6 Random Forest Classifier

Collection of multiple random decision trees. Random Forest Classifiers [22] are less sensitive to the order of the training data because they use different portions of the dataset to independently create each tree. New data points will be passed through the trees, each of which will provide a prediction independently, and the most popular prediction will be assigned to the new data point.

2.3 Izhikevich Model Introduction

In order to select a model for generating the synthetic data, we had to consider two main requirements.

The model must be computationally efficient as we are generating our data on a device with very limited resources.The model must also be capable of reproducing the rich firing patterns observed in real biological neurons.

The model proposed by Eugene Izhikevich [3] uses mathematical equations to compute a wide range of firing patterns displayed by cortical neurons. The output is incredibly realistic and biologically plausible. The value of the four parameters *a, b, c,* and *d* used in the Izhikevich model, determine the spiking and bursting behaviour of the known types of cortical neurons. The time evolution of the membrane potential *v* in Izhikevich's paper is described by ordinary differential equations of the form:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I$$

$$\frac{du}{dt} = a(bv - u)$$

Where the after-spike setting condition is:

If $v \geq 30$ mV, then $v = c$ and $u = u + d$

And where the units and meaning of the model parameters are:

v = membrane potential (mV)

t = time (ms)

$\frac{dv}{dt}$ = *time rate of change in membrane potential (mV/ms)*

$u = recovery\ variable$ (mV)

I = external current input to cell (Injected DC currents)

And where the membrane recovery variable *u* accounts the activation of K+ ionic currents and inactivation of Na+ ionic currents, and its function is to provide negative feedback to *v*. The model does not have a fixed threshold for neuron firing, which most real neurons have. Instead, the firing depends on the history of the membrane potential prior to a spike. The threshold potential may be in the range of -55 mV to -40 mV. The resting potential in the model is between -70 mV and -60 mV depending on the value of the parameter *b.*

Different combinations of the parameters *a*, *b*, *c* and *d* result in various intrinsic firing patterns. Below an explanation of each parameter is provided:

The parameter a represents the time scale of the recovery value *u*, and smaller values result in slower recovery.

The parameter *b* describes the sensitivity of the recovery variable *u* to the subthreshold fluctuations of the membrane potential *v*. Larger values of *b* couple and *v* more strongly, resulting in possible sub-threshold oscillations and low-threshold spiking dynamics.

The parameter *c* describes the after-spike reset value of the membrane potential *v*.

The parameter *d* describes the after-spike reset of the recovery variable *u*.

Although Izhikevich's model discusses 7 different types of dynamics, for this project we will be focusing on 5 fundamental classes of firing patterns for neurons in in the cortex, and ignore thalamocortical neurons (TC) as they are found in the thalamus and the dynamics of resonators (RZ) as they refer to neurons with damped subthreshold oscillations rather than a neocortical class [1] .

We now provide an overview of the characteristics of each of the types of dynamics produced in response to an injected step of DC current.

RS (regular spiking) neurons are the most commonly found neurons in the cortex. They initially fire a few spikes at high frequency and then the period increases. Increasing the strength of the injected DC current decreases the time between spikes.

IB (intrinsically bursting) neurons fire a burst of spikes followed by repetitive single spikes.

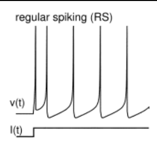
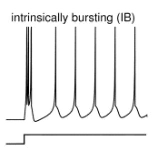
CH (chattering) neurons can fire stereotypical bursts of closely spaced spikes.

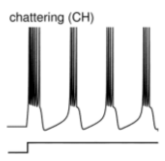
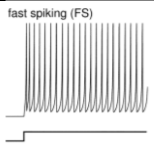
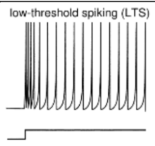
FS (fast spiking) neurons can fire periodic trains of spikes with very high frequency virtually without any adaptation.

LTS (low-threshold spiking) neurons can also fire high-frequency trains of spikes, but with a noticeable action potential frequency adaptation. These neurons have low firing thresholds.

In Table 1 [3] , a summary of characteristic parameters and intrinsic firing patterns can be visualized.

Table 1: The characteristic parameters and firing patters of the different classes.

Class Name	**a**	**b**	**c**	**d**	**Voltage Response to a step of dc current I =10**
Regular Spiking	0.02	0.2	-65	8	
Intrinsically Bursting	0.02	0.2	-55	4	

Chattering	0.02	0.2	-50	2	
Fast Spiking	0.1	0.2	-65	2	
Low-Threshold Spiking	0.02	0.25	-65	2	

2.2 Recent Literature Review

2.2.1 Existing Frameworks for Neural Classification

Within the field of neural classification, there have been research papers that have evaluated and provided improvements on manual methods of spike identification. Recorded extracellular data from real neurons provide some challenges to the advancement of better techniques, as the data does not directly provide the isolated activity of single neurons, but a collection of the dynamics of several neurons that are additionally corrupted by background noise. There are however spike sorting algorithms that have been identified in literature that are able to reconstruct the single neuron dynamics from these recordings [23].

Of particular relevance, is the implementation of a semi-automatic clustering approach proposed by Harris et al. [7] as a method for spike sorting. Compared to purely manual spike sorting, the semi-automatic approach introduced proved to have two major advantages:

An improvement in efficiency as it is substantially less time consuming to confirm the output of an automatic algorithm than to perform a fully manual sort. The second advantage is that this approach achieves considerably lower errors than purely manual sorting. Whilst this approach indicates that automatic classification methods perform well, it concerns the use of machine learning for the extraction of single neurons spikes from noisy recorded data, and it does not offer a framework through which to classify the obtained single cell dynamics.

2.2.2 Existing Frameworks for Time Series Classification

The objective of this project is to classify neurons according to their spiking dynamics. We are therefore presented with a classification task, now there are two types of classification procedures in machine learning, supervised and unsupervised.

Unsupervised classifiers divide/cluster input data points based on properties and attributes inherent to the data, and it is used to learn information about raw data when it has not been provided. This is an approach that has been taken for neural classification previously [7] . These techniques, however, require larger amounts of data and powerful tools, there's also a lack of transparency into how the data is clustered and therefore a higher risk of being unable to identify errors. On the other hand, supervised methods tend to outperform unsupervised on classification tasks, with higher accuracy and efficiency [24] . Model performance is also easily evaluated by comparing the predictions against the correct labels, which is not possible for unsupervised techniques.

The technique ultimately chosen was heavily influenced by the format of the training data and how it was acquired, which as outlined in section 2.3, was generated using a mathematical model and whose attributes and classes are already known. Therefore, as the data is labelled, supervised learning is the most appropriated technique.

Within supervised learning there are several types of classification algorithms available depending on whether the classification task is binary (two class labels) or multi-class (more than two classes). As we will be differentiating between 5 different classes, we must select techniques appropriate to multi-class classification tasks. These include Naïve Bayes, Random Forest, Decision Trees, Gradient Boosting and K-Nearest Neighbours [25] . An important factor to consider is the format of our input data, a collection of time series. For this we need dedicated algorithms, as the data differs from a regular classification problem since attributes or time points will have an ordered sequence. Typical classifiers like the ones outlined above could be used but would be inaccurate. As each time point would be treated as a separate feature and not consider the time order of the data, which contains valuable information. In literature, there are a few algorithms dedicated to time series classification. Amongst which a few have shown to perform better than a baseline classifier over a large number of different datasets [11].

One of such is distance-based algorithms, which work on the principle of distance metrics to determine a class membership. With some pre-defined similarity measures, such as Euclidean distance or Dynamic Time Warping, k-NN has been widely applied in time series classification problems, as it achieves, in conjunction with the DTW distance, the best accuracy scores against a range of benchmark datasets [26]. Furthermore, a throughout analysis completed by Mitsa [27] evaluating the performance of 1NN with DTW against a number of other techniques like Hidden Markov Models and multilayer perceptron neutral networks, suggests that when it comes to time series classification, 1 Nearest Neighbour (K=1) and Dynamic Time Warping (DTW) is very difficult to beat. This is also highlighted by the study of Mahato et al. [19], where state-of-the-art distance measures used for time series analysis are evaluated in their effectiveness for classification tasks. The study concludes that when it comes to time-series classification, k-NN with DTW has proven to be one of the most robust and well-performing methods on a wide range of datasets.

Another technique is interval- based approaches such as the time series forest, which adapts the random forest classifier discussed in section 2.6.6 to time series data. Deng et al. [28] propose a time series forest classification method, which uses a combination of information gain and distance measures for classification. They find that the time series forest is an accurate and efficient time series classifier that outperforms techniques like 1-NN with DTW in efficiency due to its linear time complexity.
Deep learning could also be used to classify time series, in particular LTSMS and CNNs can be successful. Weiwei Jiang [29] offers a comprehensive comparison between 1NN classifiers with different distance measures and deep learning models. Whilst results show that CNN achieves better performance over distanced based approaches, this is only the case when the dataset is large. See Figure 7 for a comparison in performance between Resnet (A type of CNN) against 1-NN with DTW [29].

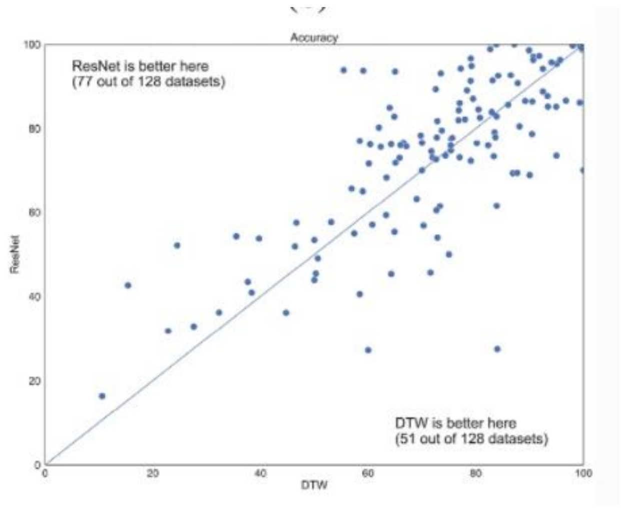


Figure 7: A comparison of the accuracy obtained using different datasets when applying Resnet (CNN) vs DTW.

Looking to address the quadratic time complexity of 1-NN with DTW Salvador et al. [30] explore the use of FastDTW, and compare its accuracy to other existing approximate DTW algorithms. They find that FastDTW its applicable to much larger datasets compared to DTW due to its linear time complexity. Whilst they conclude that it is an order to magnitude more efficient than standard DTW and that the algorithm tends to be more accurate compared to two other existing approximate DTW methods, it does not offer an accuracy comparison to the standard DTW algorithm.

Chapter 3

3. System Design

Before we dive into the implementation, it is important to consider the different technologies and tools currently available. In this chapter we will explore the different criteria considered for the selection of the programming language, Integrated Development Environment, libraries, and data sources. Following this, we evaluate the literature review outlined in section 2.2.2 in order to identify and select the models to be trained and benchmarked.

3.1 Language Selection

Table 2: A comparison of the performance of common programming languages against selected criteria.

Criteria	Execution Speed	Experience	Support in writing to CSV	Data Libraries	Machine Learning Libraries
Python3 (pypy)	Medium	Medium	High	High	High
MATLAB	Good	High	Medium	Good	Low
C++	High	Low	Low	Medium	Good
R	Low	Low	Medium	High	High

Python was ultimately chosen due to familiarity and previous experience, the extensive range of visualization packages, documentation, and the advantage of having all the requirements for an end-to-end process within one programming language. For the objective of this project, core libraries like NumPy [31] , Matplotlib [32] , pandas[33] and Scikit-Learn [34] will be useful for the visualization and manipulation of the data as well as the training and testing of the models. It is also relatively fast and appropriate for the size of the generated dataset. Due to its widespread use in industry, Python will also allow for the continued development and implementation of further functionality. Table 2 compares the most used programming languages for machine learning problems against criteria further explained below.

Execution Speed: The execution speed of the programming language is important as the program will be dealing with a large amount of data, and the execution speed will affect the efficiency of the model. Compiled languages like C++ offer an advantage over interpreted languages like Python, MATLAB and R as they are converted directly into machine code that the processor can execute when they are compiled. Whereas interpreted languages need an interpreter at run time as an intermediary step between compilation and execution. Python, R and MATLAB are also dynamically typed whereas C++ is statically typed, which means that the data type of each variable is known at run time whereas for dynamically typed it must be checked at run time, which slows the program down.

Experience: Due to time constraints and available documentation, previous experience with each programming language was an influencing factor in selection, although we have been more exposed to MATLAB, the available documentation, open-source code and the more widespread use of Python for machine learning applications, the latter was selected.

Support in Writing to CSV: As CSV was the chosen file format to store the synthetic data, the support of each programming language was considered part of the criteria. For time series data, NumPy and pandas provide to be powerful tools, as they offer a range of functions for storing data to CSV files in different formats. C++, although it has a faster time of execution, does not provide a reader/ writer built in function like Python, MATLAB or R and we must instead make use of File I/O and low-level logic.

Data Manipulation and Visualization Libraries: As the training data for the model will be computationally generated, an important factor for language selection is the range of data manipulation and visualization libraries available. Libraries like NumPy, Pandas and Matplotlib in Python are very flexible, feature-rich and can produce a large variety of plots that will be useful throughout the project to visualize the data generated (see Figure A.3) and identify errors against the expected results.

Machine Learning Libraries: Another important criterion is selecting a language that offers a range of machine learning libraries, as the appropriate use of libraries will allow us to save time writing redundant code and use readily available functions in order to meet the objectives of the research.

3.1.1 IDE Selection

JupyterLab is a browser-based IDE interface that allows for the editing and visualization of code within the same file [35]. JupyterLab was selected over fully developed IDES like PyCharm due to its great interface, segmentation of code and previous experience. Furthermore, it supports easy visualization of large data files, which will be useful for checking the generated data in the CSV files and identifying errors. Its main drawbacks are that the kernel must be restarted after a new library is downloaded and that the compiler runtime it's not as powerful as PyCharm's, it also does not provide code assistance or built-in debugger as PyCharm does. However, as the main objective of this project is the prototyping of a solution rather than its deployment, the functionality of JupyterLab far surpasses our requirements.

3.1.2 Data Libraries

The main data libraries used were Pandas and NumPy. The maximum dataset size generated was under 3.5 KB which is sufficiently small to be loaded, processed, and visualized by the Pandas, NumPy, MatplotLib and CSV libraries, that also offer the advantage of being compatible with Scikit-Learn. Furthermore, since Pandas and NumPy are libraries mainly written using C, the time of execution was not compromised due to their use.

The generated data was stored primarily as CSVs which are popular for machine learning as they are easy to view/debug and easy to read or write from programs, with the main drawback being the lack of formatting preservation, as lists of floats when written to a CSV file are converted to strings. Thus, the data must be processed and converted back to the correct format before training.

3.1.3 Machine Learning Libraries

The choices between libraries for the training and testing of the models were Scikit-Learn and Keras [36] due to the wide availability of documentation, user-friendly nature, use within the industry, as well as the extensiveness of the range of the functionality they provide. Scikit-Learn was selected throughout the project as the main library, mainly as it is the most used for the implementation of traditional Machine Learning Algorithms like SVM, K-Means, Decision Trees and k-NN. Whereas Keras is a higher-level framework used for creating deep learning models, given that the primary technique used throughout the project was k-NN. Scikit-Learn was selected as it provides sufficient functionality relating to data classification and pre-processing.

3.2 Data Source

In order to attempt to solve any problem using supervised learning, the first challenge is to obtain high quality data that is correctly labelled.
Due to a lack of comparative data and availability in literature of intracellular recordings from the neocortex of the different types of cells [7], it was decided that the membrane potential data of neurons would be obtained through the Izhikevich model [3], which provides a biologically plausible

mathematical model for obtaining the spiking behaviours of neurons in the neocortex. Generating synthetic data in the aforementioned manner proved to be advantageous due to the reduced pre-processing that had to be performed on the data. Modelling the electrical behaviour of neurons in that manner allowed us to easily create a noise-free, balanced dataset with a fixed number of instances and time points for each class.

There were a number of approaches that were considered in terms of the format of the data to be used as input.The parameters *a,b,c and d* of the Izhikevich model could have been used as features for each instance neuron, and the application of a traditional classifier used to obtain predictions from different combinations of parameters measured in neurons instead of the time series data containing the recorded membrane potential. The parameters in the Izhikevich model are however manually tuned to reproduce the voltage behaviour observed in neocortical recordings and are therefore not a representative of the biophysics of neurons. Thus, while the simulated waveforms are biologically plausible, the parameters are not biophysically meaningful and measurable experimentally in real neurons [1] , therefore the voltage data was selected in a time series format as it is incredibly realistic and biologically plausible.

3.2 Model Selection

We will now consider the advantages and disadvantages of the algorithms identified in section 2.2.2 in order to select the ones that will be implemented. These include Time Series Forest, CNNs and k-NN with DTW or FastDTW as the distance metric.

The major advantage of the Time Series Forest is that it has linear time complexity, and in some cases outperforms competitors such as 1-NN with Dynamic Time Warping [28]. Furthermore, it also allows for feature importance extraction that allows the user to understand which particularly features in the time series results in the predicted classification, it is however less commonly used for solving time series classification problems compared to 1-NN with DTW and tend to be more complex to implement due to a lack of documentation.

The main advantage of CNNs is that it learns the most important features for classification during training. Whereas algorithms like 1-NN with DTW are only looking at specific metrics like distance and disregard others that could be important like mean, standard deviation, or slope of the time series. Pre-processing required is also lower compared to other algorithms. Some the disadvantages and reasons why it was not ultimately selected is that results in literature have shown that although CNNs perform better than distance-based models, the performance difference is not significant [29]. Some of the other drawback include hyperparameter selection, as we must manually select the number of convolution filters and their size, pooling method, and number of epochs (number of times that the training set must pass through the model). Neural Networks also require an extremely large training set to provide an advantage over DTW, which is not feasible with the current processing power.

Lastly, distance-based classification, is simple, robust and does not require extensive hyperparameter tuning like CNN, and in literature 1-NN with DTW has been identified as a technique very difficult to beat for timeseries classification [19] [26] [27]. The main drawback it's the quadratic time complexity and time of execution compared to other methods, but this can be addressed through the implementation of techniques like FastDTW.

Chapter 4

4. Implementation

In this chapter we discuss the approach taken to train a k-NN model with a selection of three different distance metrics, Euclidean Distance, DTW and FastDTW, in order to compare their performance and select the most effective for our task. Below, a general flow chart highlighting the general steps that were taken to accomplish this can be observed.

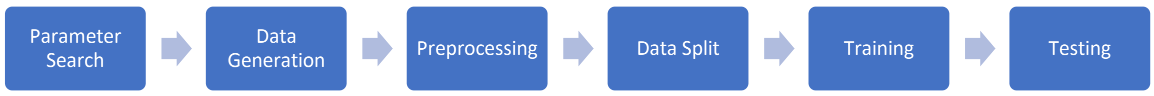


Figure 8: A flow chart highlighting the order of steps taken throughout the implementation process.

We will provide an overview of each process in further detail in the following sections, but the full code used in the implementation can be found in the appendices in the following order:

Data generation: Figure A.1

Pre-processing: Figure A.2

Visualization of the simulated spiking patterns: A.3

Parameter search: A.4 and A.5

Training and testing: A.6 and A.7

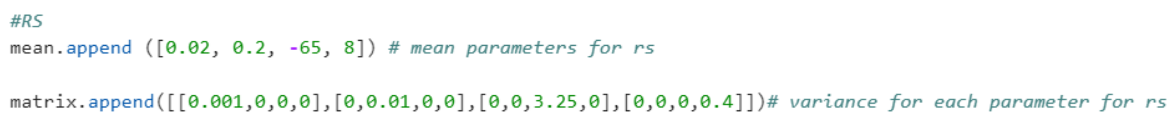
User-defined classes for DTW and FastDTW: B.1

4.1 Parameter Search

As discussed, we will be using the Izhikevich model to generate our training and testing data. Before we generate and store this data for training, it is important to tune the values of a number of parameters. Based on these parameters, we chose to maximise a metric, such as accuracy and efficiency, and we then use these parameters in the testing period. The parameters concerning our task are variance of the multivariate random distribution and the size of the training data.

4.1.1 Variance of Multivariate Distribution

The NumPy function Random.multivariate_normal [37] will be used to generate a multivariate normal distribution of the parameters that will be used to generate different spiking patterns. Five different distributions will be generated, from which we will draw the 4 parameter values for each of the 5 neuron classes. The function takes as parameters a mean and covariance matrix. The mean will be specified by the characteristic parameter values of each class, highlighted in Table 1, the covariance matrix however is analogous to the variance, or the spread of the parameter values generated, and this value will need to be careful tuned to ensure that the waveforms produced by these parameters are of the expected shape for its class. For our research, the appropriate variance was found through trial and error as there's a lack of information in literature of how much we can change the parameters in the Izhikevich model to create a waveform that it's still considered to be of that class.
This value will be found by comparing different percentage variances and visualizing the waveforms that are produced as a result.
The percentage variance is calculated as $\frac{percentage}{100} *$ mean value for each parameter, and this is done for every class. For example, the covariance matrix for an RS neuron with a 5% variance is defined as in Figure 13.



```
#RS
mean.append ([0.02, 0.2, -65, 8]) # mean parameters for rs

matrix.append([[0.001,0,0,0],[0,0.01,0,0],[0,0,3.25,0],[0,0,0,0.4]])# variance for each parameter for rs
```

Figure 9: A demonstration of the mean and covariance matrices for an RS neuron with five percent variance.

For easier visualization, we will be comparing two waveforms produced for each class when the parameters are drawn from a distribution with a variance of 1% and 5%, results for which are shown in Figures 9-10 and Figures 11-12 respectively. For the faster spiking neurons only one waveform was modelled due to their high frequency and for clearer visualization.

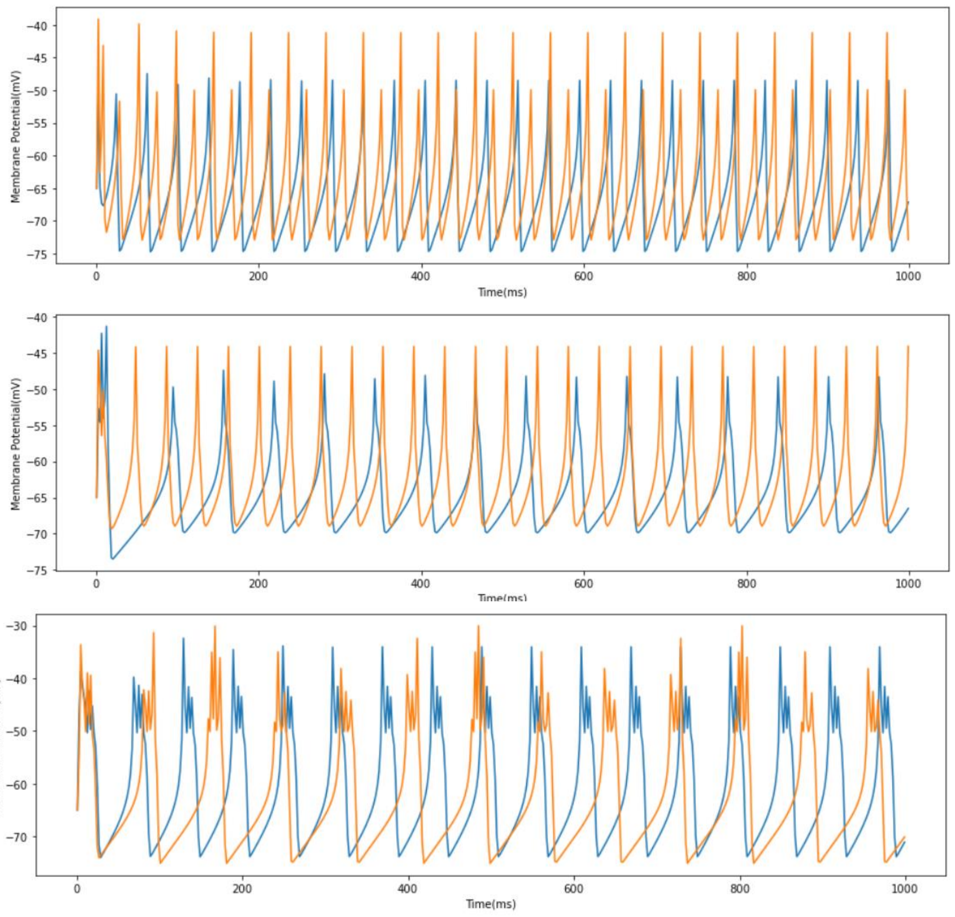


Figure 10: The graphs of the simulated spiking patterns of two RS (top), two IB (middle) and two CH (bottom) neurons, over 1000 milliseconds and with a sampling frequency of two milliseconds, when the parameters are drawn from a multivariate distribution with one percent variance.

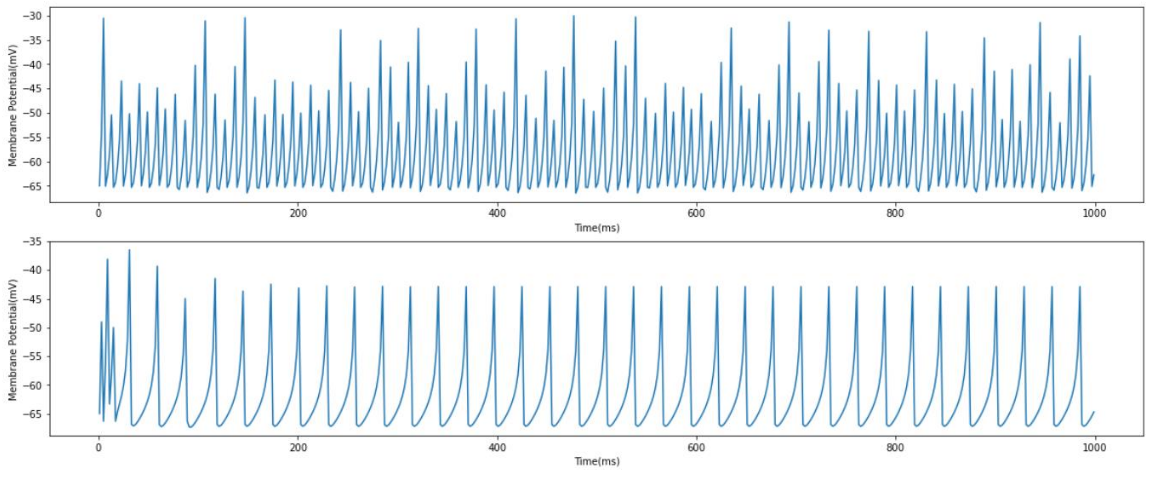


Figure 11: The graphs of the simulated spiking pattern of a FS (top) and LTS (bottom) neuron, over 1000 milliseconds and with a sampling frequency of two milliseconds, when the parameters are drawn from a multivariate distribution with one percent variance.

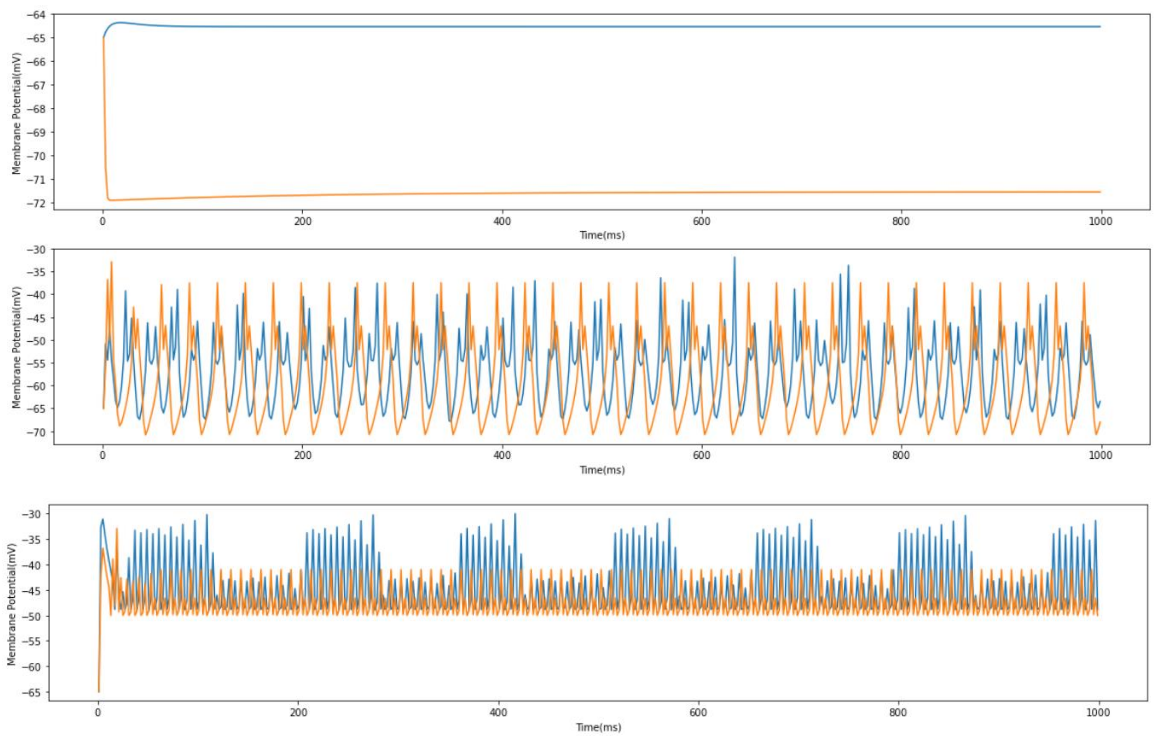


Figure 12: The graphs of the simulated spiking patterns of two RS (top), two IB (middle) and two CH (bottom) neurons, over 1000 milliseconds and with a sampling frequency of two milliseconds when the parameters are drawn from a multivariate distribution with five percent variance.

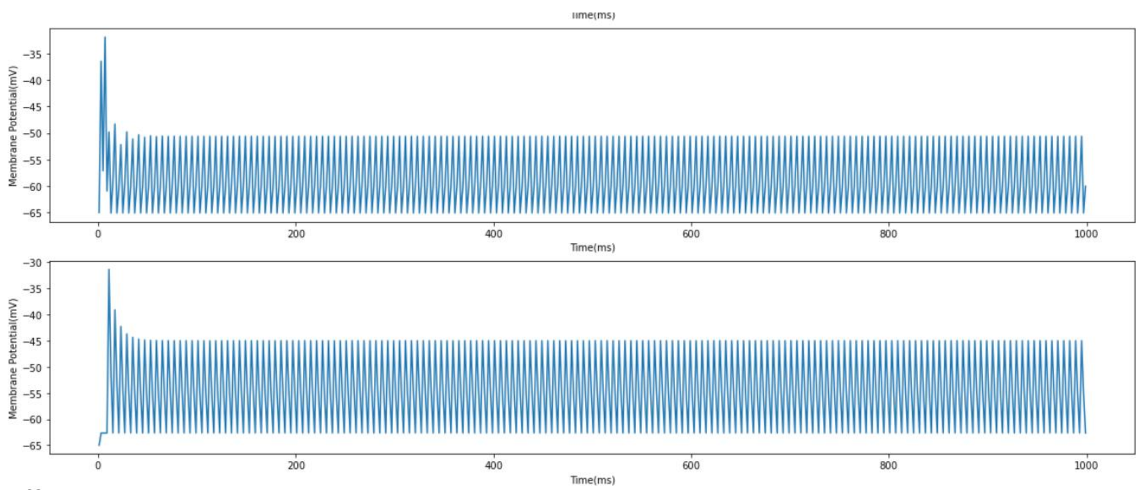


Figure 13: The graphs of the simulated spiking pattern of a FS (top) and LTS (bottom) neuron, over 1000 milliseconds and with a sampling frequency of two milliseconds, when the parameters are drawn from a multivariate distribution with five percent variance.

From the figures, we can observe that when variance is too large the waveforms generate are no longer representative of the intrinsic dynamics of each class highlighted in Table 1. For a variance of 1% however the spike patterns closely match the shape of those outlined in the Izhikevich model for RS, IB, CH, FS and LTS, so this value was selected.

4.1.2 Size of Training Data

For machine learning problems, the accuracy of the model tends to increase as the size of the training data becomes larger. However, due to the limitations in the performance of current hardware, a trade-off between dataset size and simulation time had to be made.

In order to make an informed decision of the optimal number of examples to use for each class, we generated 7 different datasets each containing a range of 10 to 70 examples per class each. We then used the different datasets to train and test the results on a k-NN classifier for different values of k. The values of k chosen were 1,3,5 and 7 as its best to choose a value of k that is an odd number to avoid a tie in selection of the majority class label [38]. Further details of the implementation of the k-NN classifier and finding the optimal value of k will be explored in section 4.3.1. Figures 14 and 15 show the results for the mean error obtained for different values of k as size of the example classes grow.

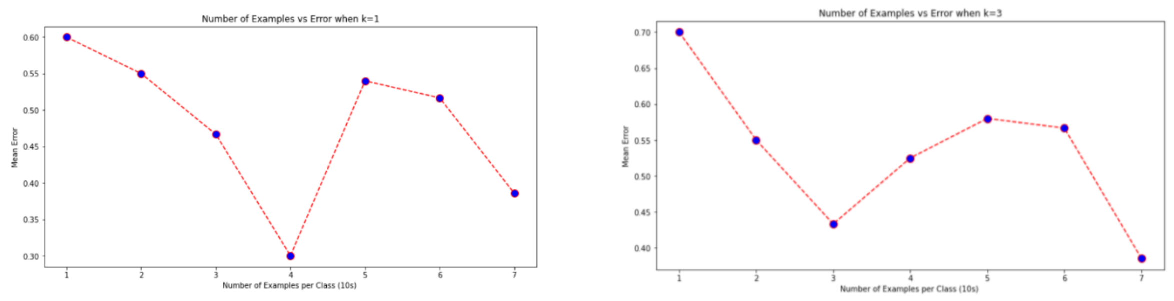


Figure 14: The plots of the mean error as the number of examples for each class grows, when k=1 (left) and k=3 (right).

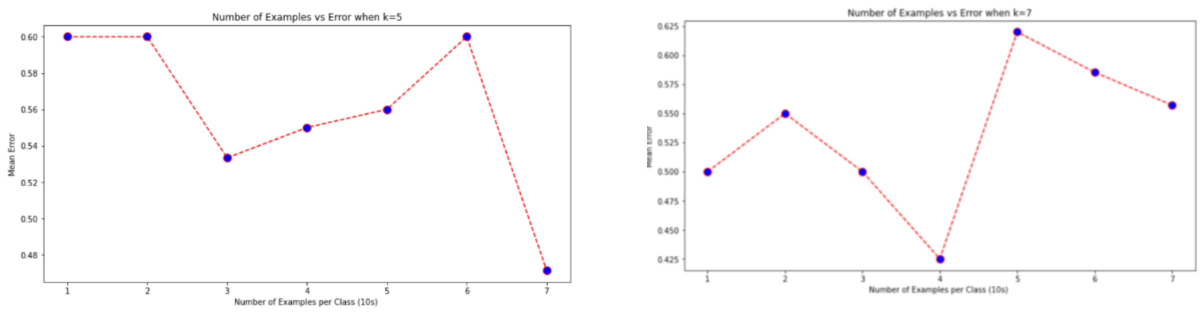


Figure 15: The plots of the mean error as the number of examples for each class grows, when k=5 (left) and k=7 (right).

Where mean error is defined as the average number of wrongly predicted classes over the number of samples in the testing set. The average mean error value for each dataset size is shown in Table 3.

Table 3: A comparison of the average mean error obtained for each example size.

Number of Examples per class	Average mean error
10	0.60
20	0.56
30	0.49
40	0.45
50	0.58
60	0.49
70	0.44

The results show that the two best performances are achieved with a dataset of 40 and 70 samples for each class. As the difference is not significant 40 was as our chosen as our selected sample size for improved simulation time.
We can observe that the mean error, even when the sample size is 40 is still quite large, further methods to address this are discussed in later sections.

4.2 Data

4.2.1 Data Generation

We now outline the processes through which the training and testing dataset were generated.
As discussed in the parameter selection section, a variance of 1% was selected to create the multivariate normal distribution. The mean and covariance matrices for all classes of neurons were stored in a list. We then generated *n* different sets of parameter values for each neuron type and assigned them to *a,b,c* and *d* accordingly. Where *n* represents the number of examples for each class, which was selected to be equal to 40 as a result of the analysis carried out in section 4.1.2.
We also found that even with a variance of 1%, in some cases we were obtaining negative values when the mean *a* value was equal to 0.02, which resulted in completely different waveforms from the ones expected. In order to account for this, we replaced the negative values with 0.01.
The parameter values were then used to generate a series of membrane potential values for each neuron according to the differential equations in the Izhikevich model outlined in section 2.3, which were stored in the variable *v*. The variable *I* was set to 10 and represents the injected DC-current step. Furthermore, the simulation length was set to 1000ms in order to capture enough information for appropriate classification.
The initial value of *v* was set to -65 to allow for all recordings to be on the same voltage scale.

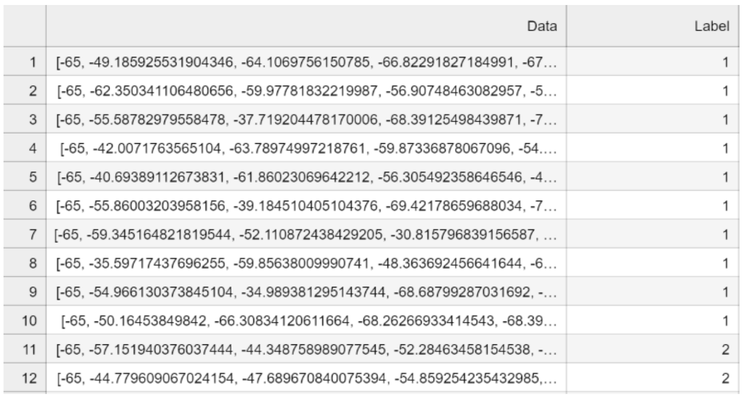
In order to avoid information loss while reducing the time taken for training and testing of models with larger datasets, the length of each time series was halved by only choosing to store every second value of *v* (sampling every 2ms) in the CSV file.
For simplicity, the class labels were stored in a numerical format in the manner highlighted in Table 4.

Table 4: The corresponding labels of each neuron class.

Neuron Class	**Corresponding Label**
Regular Spiking	**1**
Intrinsically Bursting	**2**
Chattering	**3**
Fast Spiking	**4**
Low-Threshold Spiking	**5**

4.2.2 Dataset Structure

The dataset to be used for training and testing will consist of time series data stored in a CSV file format as shown in Figure 16. Where every row displays the sequence of membrane potential values for each simulated neuron and the corresponding neuron class under the Data and Label columns respectively.



	Data	Label
1	[-65, -49.185925531904346, -64.1069756150785, -66.82291827184991, -67…	1
2	[-65, -62.350341106480656, -59.97781832219987, -56.90748463082957, -5…	1
3	[-65, -55.58782979558478, -37.719204478170006, -68.39125498439871, -7…	1
4	[-65, -42.0071763565104, -63.78974997218761, -59.87336878067096, -54.…	1
5	[-65, -40.69389112673831, -61.86023069642212, -56.305492358646546, -4…	1
6	[-65, -55.86003203958156, -39.184510405104376, -69.42178659688034, -7…	1
7	[-65, -59.345164821819544, -52.110872438429205, -30.815796839156587, …	1
8	[-65, -35.59717437696255, -59.85638009990741, -48.363692456641644, -6…	1
9	[-65, -54.966130373845104, -34.989381295143744, -68.68799287031692, -…	1
10	[-65, -50.16453849842, -66.30834120611664, -68.26266933414543, -68.39…	1
11	[-65, -57.151940376037444, -44.348758989077545, -52.28463458154538, -…	2
12	[-65, -44.779609067024154, -47.689670840075394, -54.859254235432985,…	2

Figure 16: A display of the dataset format used for training and testing.

4.2.3 Data Pre-Processing

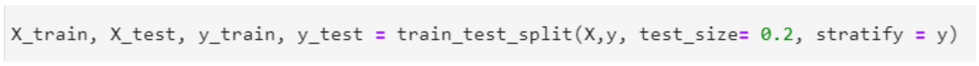
Although a common practice in machine learning as a pre-processing step, the effects of normalising and standardising the data were not explored, as it is predominantly used in machine learning problems where the time series data has input values with differing scales [39]. This is not the case for our data, as the membrane potential of the 5 neocortical classes we are concerned with display similar ranges of values.

As discussed in section 3.2.1, one of the drawbacks of using the CSV format is that writer function in the CSV library will store the time data lists as strings, so we must convert the data back into numerical format before we split the dataset into the training and testing sets. This is done by iterating through the rows in the data column, extracting the integers, which are separated by commas, and converting them to a float format. We then convert each time sequence from a list of floats to a NumPy array, which is the expected format of input data format for machine learning models in the Scikit-Learn library [40]. The class labels were stored as integers by the CSV file, so they are directly converted to a NumPy array.

4.2.4 Data Splits

In order to train and evaluate the models, the data was split into training and testing sets in the ratio of 80:20, as it is the most common split ratio for machine learning tasks [41] .

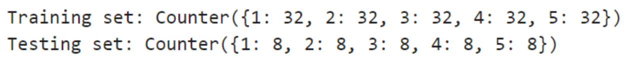
In order to do this the train_test_split function [42] from the Scikit-Learn library was used as shown in Figure 17. Where X represents the membrane potential values and y the corresponding labels of all elements in the dataset. The test_size parameter takes a float between 0 and 1 and represents the proportion of the dataset to include in the test split. The stratify argument was set to y, this ensures that the training and testing set contains the same percentage of samples of each target class as the complete set, addressing the imbalanced classification problem outlined in section 2.1.3.



```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.2, stratify = y)
```

Figure 17: A demonstration of the use of the train_test_split function for creating balanced training and testing sets.

This results in training and testing sets with distributions shown in Figure 18. Where the keys indicate the class, and the values indicate the number of examples for each.



```
Training set: Counter({1: 32, 2: 32, 3: 32, 4: 32, 5: 32})
Testing set: Counter({1: 8, 2: 8, 3: 8, 4: 8, 5: 8})
```

Figure 18: A display of the composition of the training and testing sets.

4.3 Model Training and Testing

Below, a brief description of the steps taken to train the models and the results obtained are highlighted. Further details can be found in the appendix containing the full code used for training and testing and the user defined classes for DTW and FastDTW.

4.3.1 k-NN with Euclidean Distance

As discussed in section 3.1.3 the Scikit-Learn library was used throughout the project. This library contains a function called KNeighboursClassifier which allows us to fit a k-NN classifier taking as parameter a value of k.

In order to identify the optimal value of k for our dataset. The mean error was plotted for values of k ranging from 1 to 40.

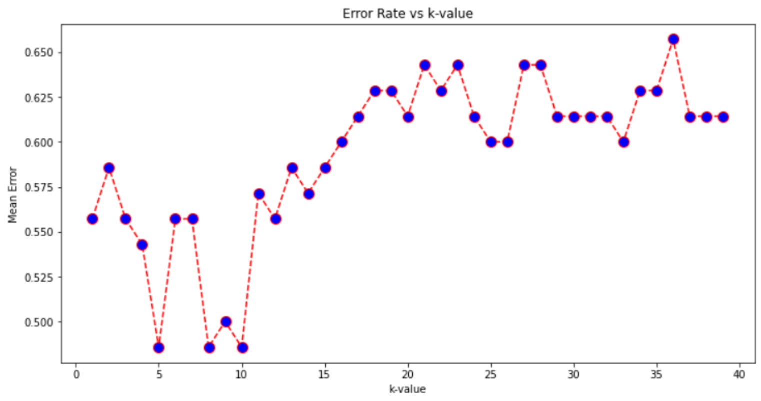
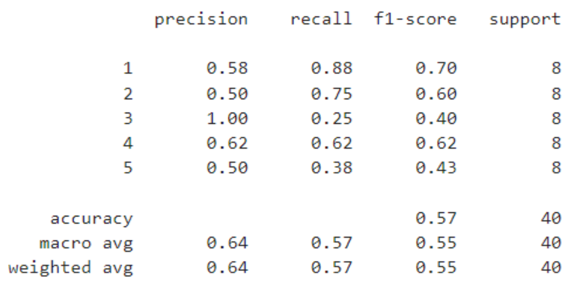


Figure 19: A plot of the mean error as the value of k increases.

In Figure 14, we can see that we obtain a minimum mean error when k = 5,7 or 10. As they all display similar values 5 will be used as the parameter value for our k-NN classifier to minimise the time of execution.

Following this, using the classification_report function provided by Scikit-Learn we obtain the performance metrics for the model shown in Figure 20.



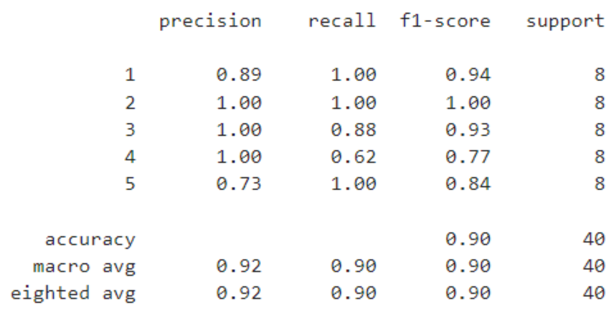
```
              precision    recall  f1-score   support

           1       0.58      0.88      0.70         8
           2       0.50      0.75      0.60         8
           3       1.00      0.25      0.40         8
           4       0.62      0.62      0.62         8
           5       0.50      0.38      0.43         8

    accuracy                           0.57        40
   macro avg       0.64      0.57      0.55        40
weighted avg       0.64      0.57      0.55        40
```

Figure 20: A display of the classification report for k-NN when k=5 and with Euclidean as the distance metric.

In addition to this, the timeit module was used to calculate the time of execution of the model. Which returned a value of 0.0135 seconds.

4.3.2 1-NN with DTW

The 1-NN with DTW algorithm is not defined in the Scikit-Learn library. Therefore, a user defined class had to be created, full code for which is available in appendix B. The value of k was set to 1, as this the most common approach when DTW is used as the distance metric[18]. After training and testing the model, we obtained the results shown in Figure 21.



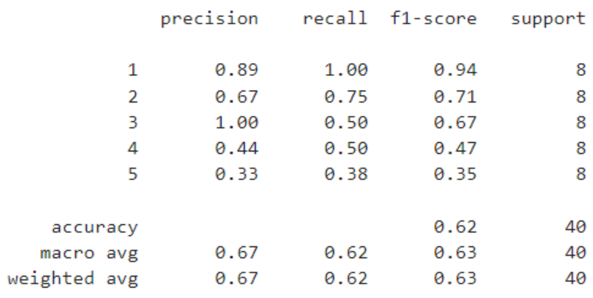
```
              precision    recall  f1-score   support

           1       0.89      1.00      0.94         8
           2       1.00      1.00      1.00         8
           3       1.00      0.88      0.93         8
           4       1.00      0.62      0.77         8
           5       0.73      1.00      0.84         8

    accuracy                           0.90        40
   macro avg       0.92      0.90      0.90        40
eighted avg       0.92      0.90      0.90        40
```

Figure 21: A display of the classification report with DTW as the distance metric.

With a time of execution of 7192 seconds.

4.3.3 1-NN with FastDTW

Similarly to 1-NN with DTW, the FastDTW algorithm is not defined in the Scikit-Learn library. But a library containing the function can be found in [43]. The library was installed and the k-NN class was modified with FastDTW as the distance metric. After training and testing the model, we obtain the results highlighted in Figure 22.



```
              precision    recall  f1-score   support

           1       0.89      1.00      0.94         8
           2       0.67      0.75      0.71         8
           3       1.00      0.50      0.67         8
           4       0.44      0.50      0.47         8
           5       0.33      0.38      0.35         8

    accuracy                           0.62        40
   macro avg       0.67      0.62      0.63        40
weighted avg       0.67      0.62      0.63        40
```

Figure 22: A display of the classification report with FastDTW as the distance metric.

With a time of execution of 906 seconds.

Chapter 5

5. Performance Evaluation and Conclusions

Analysing the findings from the previous chapter, we can now benchmark the results against each other to be able to critically evaluate the performance of the different distance-based classification approaches to the neural classification problem. The results will also be evaluated against previous research to gauge the applicability of their conclusions in the context of our task. In order to do this, we will first explain the metrics offered by the classification report and assess which ones are most relevant to assess performance. As a separate performance metric, we will also compare the time of execution of each model to assess efficiency.

5.1 Performance Evaluation

5.1.1 Selection of Performance Metrics

The classification report shows the main classification metrics of precision, recall and f1-score on a per-class basis. The metrics are calculated by using true and false positives and true and false negatives, which will be briefly described.

True Positive (TP): When an example was of that class and predicted to be of that class.
True Negative (TN): When an example was not of that class and predicted not to be of that class.
False Positive (FP): When an example was not of that class but predicted to be of that class.
False Negative (FN): When an example was of the that class but not predicted to be of that class.

Precision is a measure of the accuracy of positive predictions for a particular class, it can be seen as a measure of quality and it is used when our focus is to minimise False Positives. It is defined as:

$$\frac{TP}{TP + FP}$$

Recall is a measure of the fraction of positives that were correctly identified, it can be seen as a measure of quantity of predictions, and it is used when our focus is to minimise False Negatives. It is defined as:

$$\frac{TP}{TP + FN}$$

F1 Score is a weighted harmonic mean of precision and recall. It is a good metric when data is imbalanced [44] and thus will not be used.

Accuracy is the ratio of the total number of correct predictions divided by the size of the testing set. As outlined in [44] it is an average measure which is suitable for datasets that are balanced as it does not take into consideration the distribution of classes.

$$\frac{TP + TN}{TP + FP + FN + TN}$$

For multiclass classification, there are also 2 averaging techniques for precision, recall and F1 Score: Macro average refers to the mean of all metrics across all classes. It gives equal weights to all classes, making it a good option for balanced classification tasks [44].

Weighted average accounts for class imbalance calculated by multiplying each precision, recall and F1 score by the number of occurrences in each class and dividing by the total number of samples. In this case, large and small classes have a proportional effect on the result in relation to their size [44]. This is thus more suited for imbalanced datasets and will not be used.

Therefore, since we have an equal number of classes and hence a balanced dataset, the most appropriate metrics for the overall performance of our models will be accuracy, macro precision and macro recall. Whereas for class-wise performance, precision and recall will be used.

5.1.2 General Performance Comparison

The performance metric results for each model are summarised in Table 5.

Table 5: A comparison of the performance of each distance metric in terms of macro precision, macro recall and accuracy.

Distance Metric	Time of Execution (seconds)	Macro Precision	Macro Recall	Accuracy
Euclidean	0.0135	0.64	0.57	0.57
DTW	7192	0.92	0.9	0.9
FastDTW	906	0.67	0.62	0.62

We can see that the standard k-NN classifier, although extremely fast, displays a very poor performance compared to the other algorithms, this confirms the need of dedicated time series algorithms for the accurate classification of time series data as discussed in previous research [11].

The results obtained for 1-NN with DTW are also in agreement with what we would expect from the literature review conducted [19][26][27]. It displays the best overall predictive performance with an accuracy score of 90%. However, this algorithm takes a substantial length of time to run and further work will be necessary to extend its applicability to larger datasets.

The results obtained for FastDTW are both in agreement and disagreement with the outcome of previous research [20], which would suggest a better performance than the one we have obtained. Although FastDTW is 7.9 times faster than the DTW metric, it is not by an order of magnitude, and the accuracy scores are low, with only 62% overall accuracy. This is likely to be attributed to the relatively short length of the time series data used for our research compared with the ones that have been used for training in literature. It has been suggested that it is only once the length of the time series exceed 200-300 points that FastDTW becomes the more efficient algorithm [30]. For this study we dealt with 500 time points for each instance, this could account for a decreased time of execution that was however not by orders of magnitude. FastDTW is also an approximation algorithm unlike DTW and as such, may need more training data to learn from in order to be able to make better predictions. It would therefore we interesting to compare the performance of FastDTW with larger time series in a machine that benefits from the necessary computational power.

5.1.3 Class-Wise Performance Comparison

In order to gain further insight into the performance of the models for each class a breakdown of precision and recall for each of the 5 types of neurons is analysed.

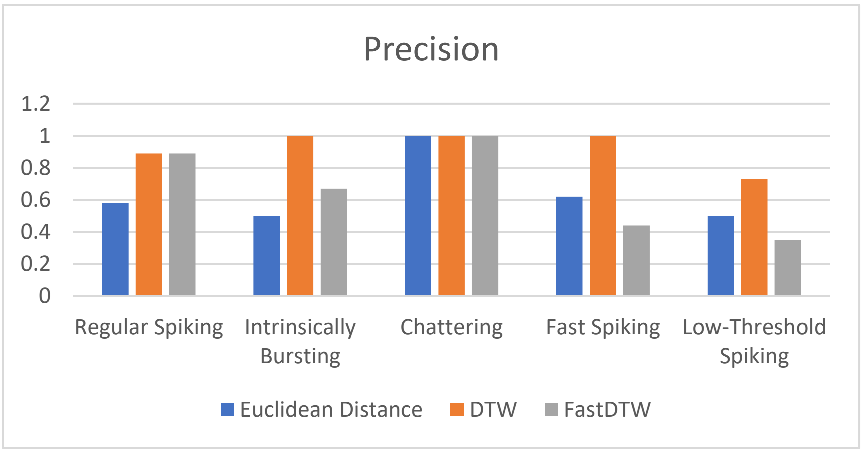


Figure 23: A comparison of the precision of the three distance metrics for each neuron type.

While as outlined in the previous section, DTW performs the best overall as a distance metric to the k-NN classifier, it can be observed in Figure 23 that predictions of regular spiking and chattering cells were as precise with DTW as with FastDTW. It can also be observed that, although the Euclidean Distance has low precision across most of the classes, it performs well at predicting chattering cells. This is interesting as research suggests that standard classifiers do not tend to perform well on time series data [19]. The high precision rate suggests that it does not tend to incorrectly assign other classes to this class (false positive), this can be attributed to the unique waveform of this neocortical neuron compared to the rest of classes, which do not differ as dramatically from each other, as seen in Figures 9 and 10.

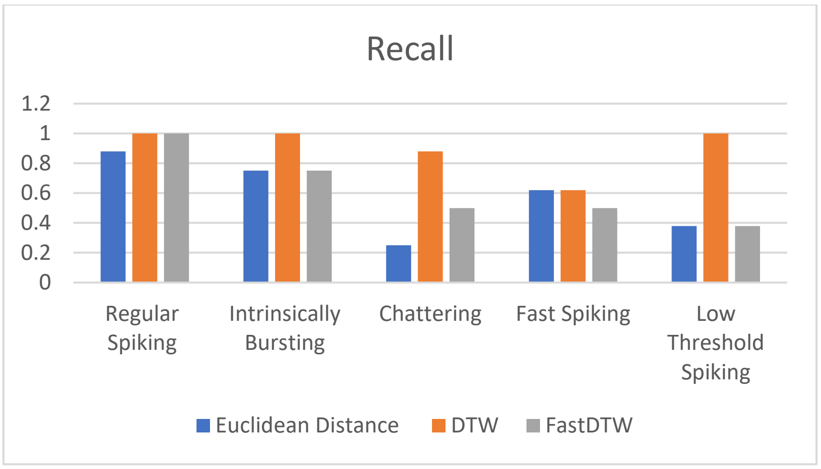


Figure 24: A comparison of the recall of the three distance metrics for each neuron type.

When observing recall in Figure 24 we find that, except for regular spiking cells predictions, DTW outperforms both Euclidean and FastDTW in predicting all other types of patterns. The low recall for the Euclidean classifier suggests a large ratio of false negatives compared to other techniques, which shows that while it performs well at being able to not incorrectly assign other types of neurons to be of the chattering type, it performs poorly at detecting a good portion of the chattering cells.

5.2 Conclusions

In summary, the results show that for a generalised framework that performs well at identifying all of the 5 types of neocortical patterns discussed throughout the paper, DTW proves to be the most accurate approach, however, it is only applicable to relatively small datasets. On the other hand, if one were interested in identifying only regular spiking cells, the results indicate that FastDTW performs as well as DTW both in terms of recall and precision and is therefore a more suitable approach due to its linear time complexity.

If our objective is to identify chattering cells and we are concerned with the quality of predictions rather than the quantity (precision over recall) then Euclidean should be used as a distance metric to k-NN due to its reduced time of execution compared to both FastDTW and DTW. The low recall rates, which as outlined in section 5.1.1 are related to a larger ratio of false negatives, could be addressed by the ability to predict a larger number of spiking patterns at any one time compared to DTW and FastDTW thanks to the efficiency of the Euclidean metric.

Chapter 6

6. Conclusions and Future Work

In this chapter we will summarise the key findings and suggest areas for further work, including the steps that can be taken to enhance the quality of the training data, increase efficiency and broader functionality.

6.1 Summary of Contributions

The aim of this project was to investigate the use and applicability of machine learning to the problem of neural classification in order to create a framework that improves upon traditional methods of spike pattern classification from extracellular recordings of neocortical neurons.
In chapter 3, we identified state-of-the algorithms that have shown superior performance for time series classification. We used this information to implement a selection of approaches in order to benchmark the results and select the best performing model for this application.
In chapter 4, we experimented with different distanced based approaches and variations in the spread of the model parameters, the number of examples per class, and k values before finding a model that performs successfully on unseen data across all neocortical classes, while offering some insights as to why some models failed and the predictive capabilities of the models on a class-wise basis.
We did this by using synthetically generated data representative of the dynamics of real neocortical neurons, modelled from the differential equations provided by the Izhikevich model, to train k-NN algorithms with different distance metrics, and benchmarking them against selected performance metrics. We found that for a generalised framework appropriate for the prediction of the five neocortical patterns discussed throughout the paper, DTW proves to be the most accurate distance metric with a 90% accuracy score. This approach however, proves to be inefficient for large datasets, so the number of spiking patterns that could be predicted at any one time would be limited. Further inspecting that performance of the models, we found that for the classification of regular spiking cells, FastDTW proved to be as effective and more efficient at predicting the spiking patterns than DTW. Whereas for chattering cells, Euclidean distance proved to be as precise but displayed low levels of recall compared to DTW, however due to its significantly lower time of execution, it proves to be an appropriate method of identifying chattering cells when the number of examples to be predicted is large.

6.2 Future Development

In this section, we will focus on the quality of training data, including potential sources of bias and how to address them. Following this, we discuss extending the models functionaly by the consideration of generating training data that is more representative of the dynamics of the neocortex, which would result in a reduction of the pre-processing needed for extracellular recordings before they are evaluated by the classifier. Finally, we explore additional techniques that could be applied for increased efficiency of FastDTW and DTW.

6.2.1 Potential Sources of Bias

During the generation of synthetic data, it was observed that some of the *a* values drawn from the multivariate distribution were below 0, this resulted in anomalous dynamics not characteristic of the corresponding classes. In order to account for this, these values were identified and set to 0.01. This however, biases the distribution towards that specific value of *a* which may lead to the over-representation of particular waveforms within each class. In order to account for this, the negative values of *a* could instead be converted to their positive equivalents. This could have also been avoided by the selection of a smaller variance percentage in the distribution, but the range of dynamics generated for each class would have been too small for accurate classification of real extracellular recordings. Following on this topic, the appropriate variance was found through trial and error as there's a lack of information in literature of how much we can change the parameters in the Izhikevich model to create a waveform that it's still considered to be of that class and further work is needed to identify this.

6.2.2 Broader Applicability

The initial values of membrane potential for each example neuron were set to -65mV to allow for all recordings to be on the same voltage and time scale. In practice, the initial membrane potential recorder for real neurons is not limited to this value, which means that the waveforms could be shifted by a particular amount in the time scale, this should not affect the predicted performance of time series metrics like DTW or FastDTW, which allow for time warping as mentioned in section 2.2 but must be accounted for in the use of Euclidean distance in order to identify chattering cells. This could be addressed either through sufficient pre-processing of extracellular recordings, or through further training of the Euclidean model, which would include generating training data containing a range of initial values of membrane potential.

As discussed in section 1.3 the extracellular recordings to be classified are assumed to be isolated and noise-free. In practice these recordings are subjected to background noise arising from neurons that superimpose the firing patterns of the neuron to be observed [7]. Further work could include the investigation of the distribution of background noise and integrate this in the generation of the synthetic data. This would allow for the training of a model that is able to account for noise and would reduce the amount of pre-processing that must be done on the recordings before the model is used for predictions.

The model also assumes that the membrane potential data to be classified is recorded under controlled conditions as a response to an injected DC current step of size ten as input. In practice, the neuron will be subject to a range of input signals, the strength of which will cause a delay in the firing of spikes for most cortical neurons [45]. The model could therefore be extended to generate example data for each class under a range of different input signals, such as oscillatory or ramped currents, in order to train the model to predict the dynamics of extracellular recording of neurons subjected to a variety of inputs, this could be easily done by changing the value of the variable I (see Figure A.1 for the full code).

The membrane potential ranges of the recordings will also be dependent on the distance of the neuron being measured to the electrode [7], which could result in values of voltage smaller from the ones simulated in the Izhikevich model. As a result, the model could be further extended to account for this through the normalization and standardisation [46] of the training data and the extracellular recordings, so that the values of the time series are on a common scale and our model is able to identify the underlying dynamics of the neuron without being affected by the strength of the signal.

6.2.3 Efficiency

We observed that although efficient, FastDTW failed to provide an accurate performance for classification. Therefore, further work is needed to identify techniques that speed up DTW. To date, there are a number of state-of-the-art methods that have been explored in literature, one of them being SparseDTW. Al-Naymat et al. [47] found that the algorithm outperforms DTW both in efficiency and accuracy. Therefore, an appropriate next step would be to implement and benchmark this method against the performance obtained for DTW to analyse the applicability of this statement to our classification problem.

Lower frequencies of sampling and their effect on performance could also be explored. Currently, the training data contains membrane potential sampled at intervals of 2 milliseconds in order to minimise information loss and preserve the shape of the time series, whilst reducing the length of the time series data and thus execution time. Further experiments with larger interval values could be carried out to reach a comprise between accuracy and efficiency.

Due to computational constraints, the optimal size of the training data and number of examples per class that yield the best compromise between time of execution and mean error for the DTW and

FastDTW metrics was not identified. Instead, the value discovered through empirical investigation for the Euclidean metric was used in the training of the latter two models. Further work could include carrying out the same procedure as in section 4.1.2 to identify these parameters for DTW and FastDTW, which could result in increased accuracy and efficiency. In order to do this, a more efficient language like C++ and/or a device with increased processing power should be used.

6.3 Final Thoughts

Modern recording tools allow us to simultaneously record the activity of hundreds of neurons, and developments will allow for the recording of thousands or tens of thousands in the near future. These advancements will pose new challenges for the data processing of neural data, as manual classification will not be feasible for such large amounts. These challenges can be tackled through the development of automatic classification algorithms, through approaches such as the ones explored in this paper. We hope that this analysis will encourage and accelerate further research into the application of time series classification techniques for spike pattern detection. Finally, although this paper focused on the classification of neocortical neurons, the dynamics of other neuronal types in areas such as the hippocampus, brainstem, basal ganglia, and olfactory bulb can also be described by the Izhikevich model, so it would be interesting to see an extension of the model for the classification of other types of neurons.

7. References

[1] Izhikevich, E., 2014. *Dynamical Systems in Neuroscience*. Cambridge: MIT Press, pp.-xvi, p.13, p.267, p.278, pp.280-281,

[2] Shinomoto, S., Kim, H., Shimokawa, T., Matsuno, N., Funahashi, S., Shima, K., Fujita, I., Tamura, H., Doi, T., Kawano, K., Inaba, N., Fukushima, K., Kurkin, S., Kurata, K., Taira, M., Tsutsui, K., Komatsu, H., Ogawa, T., Koida, K., Tanji, J. and Toyama, K., 2009. Relating Neuronal Firing Patterns to Functional Differentiation of Cerebral Cortex. PLoS Computational Biology, 5(7), p.9.

[3] Izhikevich, E., 2003. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6).

[4] Foehring, R., Lorenzon, N., Herron, P. and Wilson, C., 1991. Correlation of physiologically and morphologically identified neuronal types in human association cortex in vitro. Journal of Neurophysiology, 66(6).

[5] Venkadesh, S., Komendantov, A., Listopad, S., Scott, E., De Jong, K., Krichmar, J. and Ascoli, G., 2018. Evolving Simple Models of Diverse Intrinsic Dynamics in Hippocampal Neuron Types. Frontiers in Neuroinformatics, 12.

[6] Shetty, B., 2022. An in-depth guide to supervised machine learning classification. [online] Built In. Available at: <https://builtin.com/data-science/supervised-machine-learning-classification> [Accessed 30 March 2022].

[7] Harris, K., Quiroga, R., Freeman, J. and Smith, S., 2016. Improving data quality in neuronal population recordings. *Nature Neuroscience*, 19(9).

[8] Bagnall, A., Lines, J., Bostrom, A., Large, J. and Keogh, E., 2016. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery, 31(3).

[9] Nelson, S., Sugino, K. and Hempel, C., 2006. The problem of neuronal cell types: a physiological genomics approach. *Trends in Neurosciences*, 29(6).

[10] Berry, M. and Browne, M., 2006. *Lecture notes in data mining*. Singapore: World Scientific, pp.67-77.

[11] Amidon, A., 2022. A Brief Survey of Time Series Classification Algorithms. [online] Medium. Available at: <https://towardsdatascience.com/a-brief-introduction-to-time-series-classification-algorithms-7b4284d31b97> [Accessed 30 March 2022].

[12] Fitzgibbons, L., 2022. What is a Time Series Database? TSDB explained. [online] IoT Agenda. Available at: <https://www.techtarget.com/iotagenda/definition/time-series-database-TSDB?msclkid=5182dcb7aff111eca0d74fea86558ce0> [Accessed 30 March 2022].

[13] Brownlee, J., 2022. A Gentle Introduction to Imbalanced Classification. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/what-is-imbalanced-classification/> [Accessed 30 March 2022].
[14] All Things Statistics. 2022. *Bivariate Normal Distribution - PDF & Examples - All Things Statistics*. [online] Available at: <https://allthingsstatistics.com/distributions/bivariate-normal-distribution/> [Accessed 30 March 2022].
[15] learn2torials. 2022. *O(n2): Quadratic Time Complexity*. [online] Available at: <https://www.learn2torials.com/a/quadratic-time-complexity> [Accessed 30 March 2022].
[16] En.wikipedia.org. 2022. k-nearest neighbors algorithm - Wikipedia. [online] Available at: <https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm> [Accessed 30 March 2022].
[17] Medium. 2022. *K-Nearest Neighbors Algorithm for Machine Learning*. [online] Available at: <https://medium.com/@pranav3nov/k-nearest-neighbors-algorithm-for-machine-learning-3f2aaece4c2c> [Accessed 30 March 2022].
[18] Zhang, J., 2022. *Dynamic Time Warping*. [online] Medium. Available at: <https://towardsdatascience.com/dynamic-time-warping-3933f25fcdd> [Accessed 30 March 2022].
[19] Mahato, V., O'Reilly, M. and Cunningham, P., 2018. A Comparison of k-NN Methods for Time Series Classification and Regression. In: *AIAI Irish Conference on Artificial Intelligence and Cognitive Science*.

[20] Salvador, S. and Chan, P., 2007. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5).

[21] Saha, S., 2022. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. [online] Medium. Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> [Accessed 30 March 2022].
[22] Mollas, I., Tsoumakas, G. and Bassiliades, N., 2019. *LionForests: Local Interpretation of Random Forests through Path Selection*,.
[23] Franke, F., Natora, M., Boucsein, C., Munk, M. and Obermayer, K., 2009. An online spike detection and spike classification algorithm capable of instantaneous resolution of overlapping spikes. *Journal of Computational Neuroscience*, 29(1-2), pp.127-148.
[24] Ibm.com. 2022. *Supervised vs. Unsupervised Learning: What's the Difference?*. [online] Available at: <https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning?msclkid=d5ce1624b03611ecbbc77f10788fb6e0> [Accessed 30 March 2022].

[25] Brownlee, J., 2022. *4 Types of Classification Tasks in Machine Learning*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/types-of-classification-in-machine-learning/> [Accessed 30 March 2022].

[26] Petitjean, F., Forestier, G., Webb, G., Nicholson, A., Chen, Y. and Keogh, E., 2015. Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm. *Knowledge and Information Systems*, 47(1), pp.1-26.

[27] MITSA, T., 2019. *TEMPORAL DATA MINING*. [S.l.]: CRC PRESS, pp.99-100.

[28] Deng, H., Runger, G., Tuv, E. and Vladimir, M., 2013. A time series forest for classification and feature extraction. *Information Sciences*, 239, pp.142-153.

[29] Jiang, W., 2020. Time series classification: nearest neighbor versus deep learning models. *SN Applied Sciences*, 2(4).
[30] Salvador, S. and Chan, P., 2007. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5), pp.561-580.
[31] Numpy.org. 2022. *NumPy*. [online] Available at: <https://numpy.org/> [Accessed 30 March 2022].

[32] Matplotlib.org. 2022. *Matplotlib — Visualization with Python*. [online] Available at: <https://matplotlib.org/> [Accessed 30 March 2022].

[33] Pandas.pydata.org. 2022. *pandas - Python Data Analysis Library*. [online] Available at: <https://pandas.pydata.org/> [Accessed 30 March 2022].

[34] Scikit-learn.org. 2022. *scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation*. [online] Available at: <https://scikit-learn.org/stable/> [Accessed 30 March 2022].

[35] Jupyterlab.readthedocs.io. 2022. *Overview — JupyterLab 3.3.2 documentation*. [online] Available at: <https://jupyterlab.readthedocs.io/en/stable/getting_started/overview.html> [Accessed 30 March 2022].

[36] Team, K., 2022. *Keras: the Python deep learning API*. [online] Keras.io. Available at: <https://keras.io/> [Accessed 30 March 2022].

[37] Numpy.org. 2022. *numpy.random.multivariate_normal — NumPy v1.22 Manual*. [online] Available at: <https://numpy.org/doc/stable/reference/random/generated/numpy.random.multivariate_normal.html> [Accessed 30 March 2022].

[38] Brownlee, J., 2022. *K-Nearest Neighbors for Machine Learning*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/> [Accessed 30 March 2022].

[39] Brownlee, J., 2022. *How to Normalize and Standardize Time Series Data in Python*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/normalize-standardize-time-series-data-python/> [Accessed 30 March 2022].

[40] Brownlee, J., 2022. *How to Save a NumPy Array to File for Machine Learning*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/how-to-save-a-numpy-array-to-file-for-machine-learning/> [Accessed 30 March 2022].

[41] AskPython. 2022. *Split Training and Testing Data Sets in Python - AskPython*. [online] Available at: <https://www.askpython.com/python/examples/split-data-training-and-testing-set?msclkid=76296c8cad3e11ec83d935d3543e8a85> [Accessed 30 March 2022].

[42] scikit-learn. 2022. *sklearn.model_selection.train_test_split*. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html?msclkid=eb38ce7cad3d11ec93434b911fdc58bd> [Accessed 30 March 2022].

[43] PyPI. 2022. *fastdtw*. [online] Available at: <https://pypi.org/project/fastdtw/> [Accessed 30 March 2022].

[44] Grandini, M., Bagli, E. and Visani, G., 2020. Metrics for Multi-Class Classification: an Overview.

[45] Izhikevich, E., 2004. Which Model to Use for Cortical Spiking Neurons?. *IEEE Transactions on Neural Networks*, 15(5), pp.1063-1070.

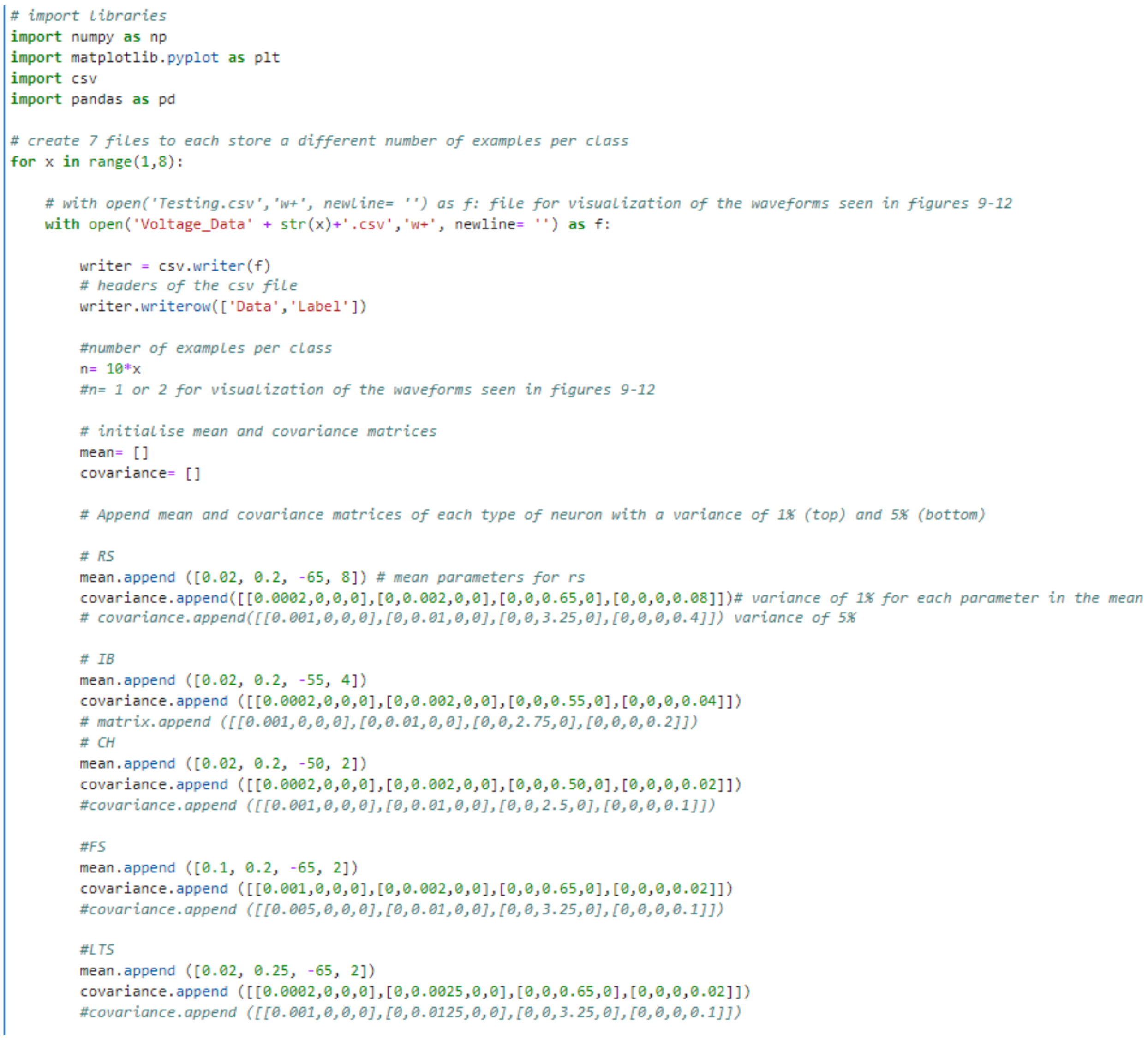
[46] JournalDev. 2022. *2 Easy Ways to Normalize data in Python - JournalDev*. [online] Available at: <https://www.journaldev.com/45109/normalize-data-in-python#:~:text=%20Steps%20to%20Normalize%20Data%20in%20Python%20,when%20it%20comes%20to%20normalizing%20data%3A...%20More%20?msclkid=a7754057ad0d11ecb6423d6bb8cfd9aa> [Accessed 30 March 2022].

[47] Al-Naymat, G., Chawla, S. and Taheri, J., 2012. SparseDTW: A Novel Approach to Speed up Dynamic Time Warping.

8. Appendices

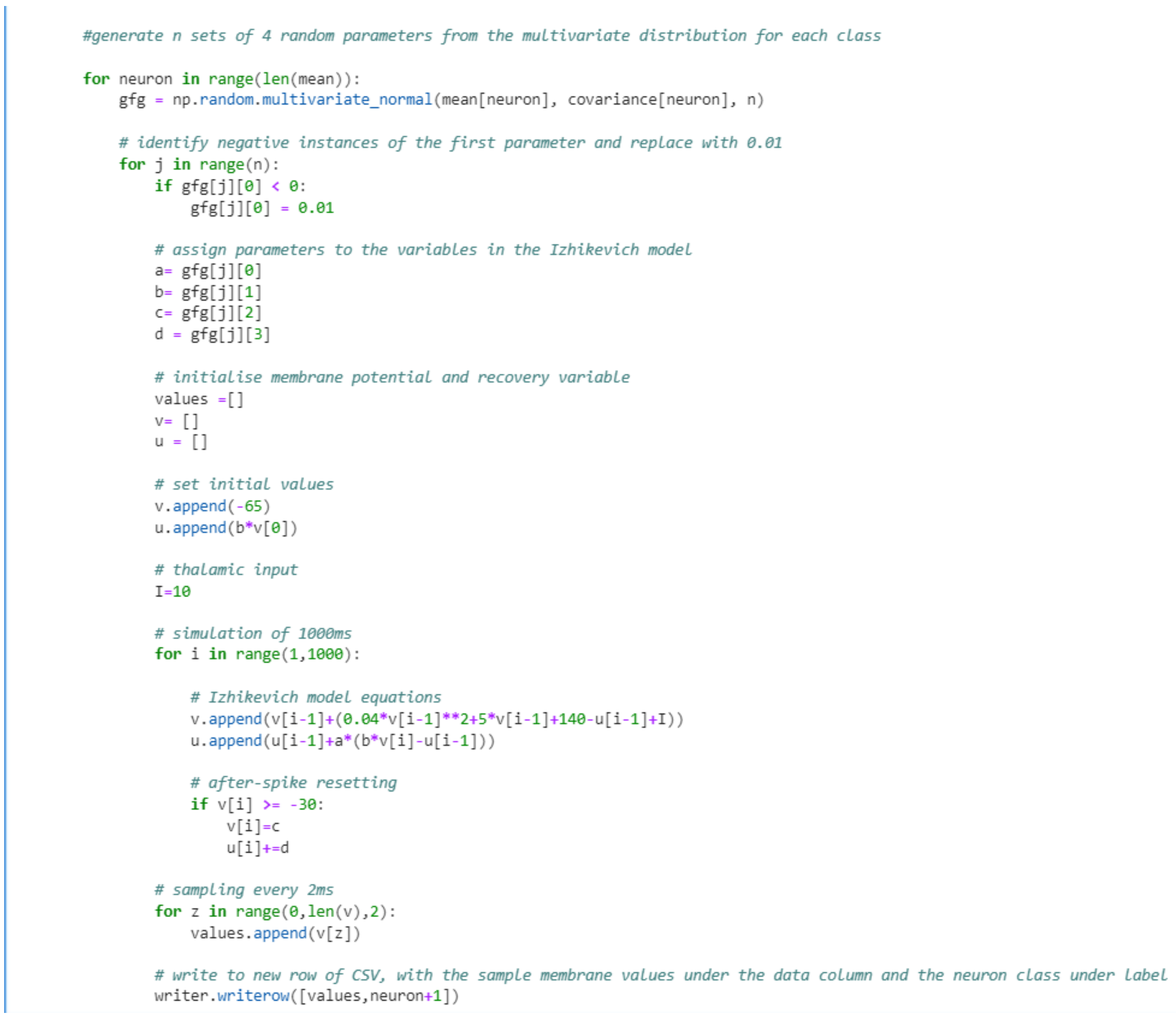
Appendix A. Overview of the Source Code Used for Training and Testing

This section contains screenshots of the full code used for data generation, data pre-processing, waveform visualization, parameter search, model training and testing of the 3 models and the user-defined classes for DTW and FastDTW.



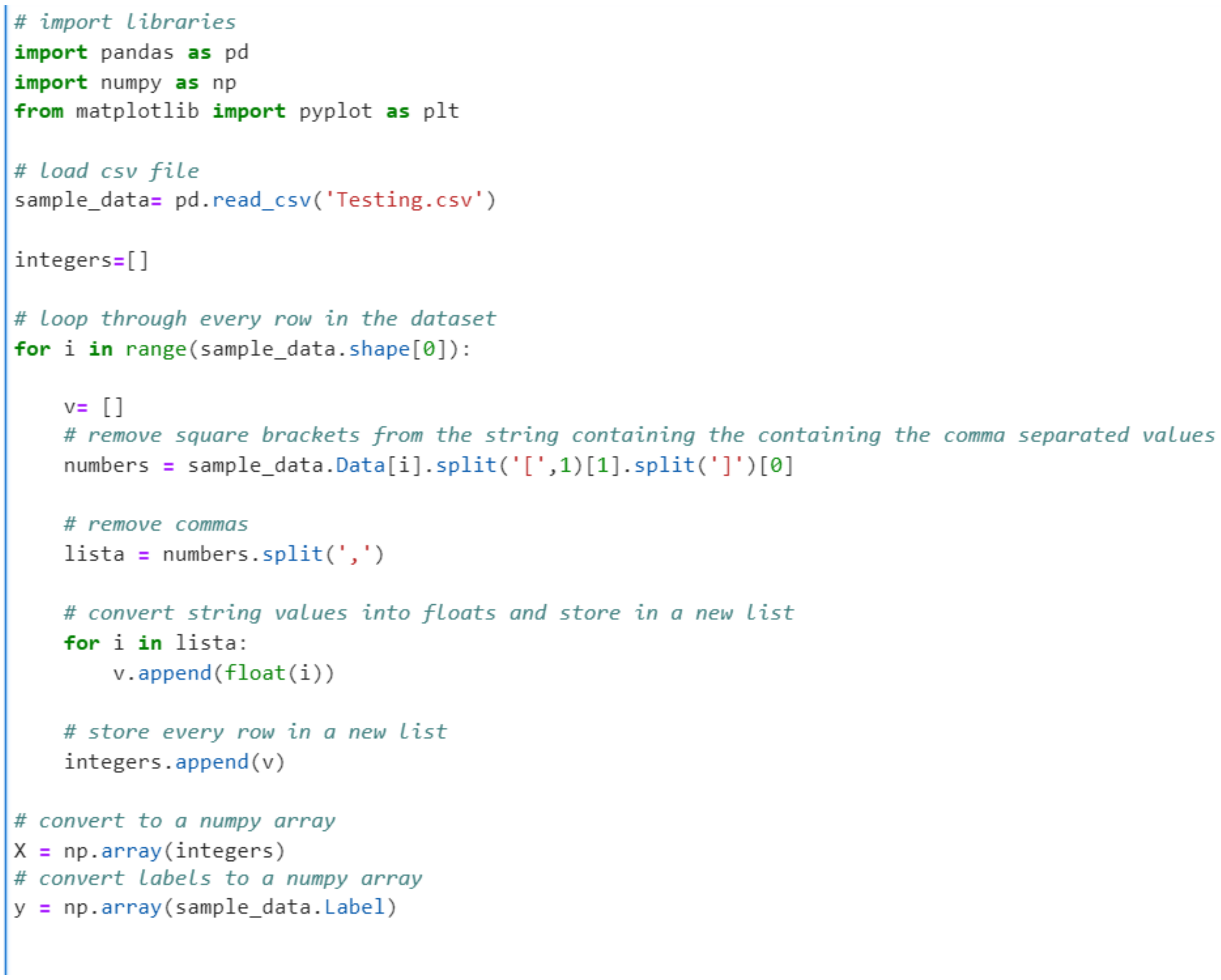
```python
# import libraries
import numpy as np
import matplotlib.pyplot as plt
import csv
import pandas as pd

# create 7 files to each store a different number of examples per class
for x in range(1,8):

    # with open('Testing.csv','w+', newline= '') as f: file for visualization of the waveforms seen in figures 9-12
    with open('Voltage_Data' + str(x)+'.csv','w+', newline= '') as f:

        writer = csv.writer(f)
        # headers of the csv file
        writer.writerow(['Data','Label'])

        #number of examples per class
        n= 10*x
        #n= 1 or 2 for visualization of the waveforms seen in figures 9-12

        # initialise mean and covariance matrices
        mean= []
        covariance= []

        # Append mean and covariance matrices of each type of neuron with a variance of 1% (top) and 5% (bottom)

        # RS
        mean.append ([0.02, 0.2, -65, 8]) # mean parameters for rs
        covariance.append([[0.0002,0,0,0],[0,0.002,0,0],[0,0,0.65,0],[0,0,0,0.08]])# variance of 1% for each parameter in the mean
        # covariance.append([[0.001,0,0,0],[0,0.01,0,0],[0,0,3.25,0],[0,0,0,0.4]]) variance of 5%

        # IB
        mean.append ([0.02, 0.2, -55, 4])
        covariance.append ([[0.0002,0,0,0],[0,0.002,0,0],[0,0,0.55,0],[0,0,0,0.04]])
        # matrix.append ([[0.001,0,0,0],[0,0.01,0,0],[0,0,2.75,0],[0,0,0,0.2]])
        # CH
        mean.append ([0.02, 0.2, -50, 2])
        covariance.append ([[0.0002,0,0,0],[0,0.002,0,0],[0,0,0.50,0],[0,0,0,0.02]])
        #covariance.append ([[0.001,0,0,0],[0,0.01,0,0],[0,0,2.5,0],[0,0,0,0.1]])

        #FS
        mean.append ([0.1, 0.2, -65, 2])
        covariance.append ([[0.001,0,0,0],[0,0.002,0,0],[0,0,0.65,0],[0,0,0,0.02]])
        #covariance.append ([[0.005,0,0,0],[0,0.01,0,0],[0,0,3.25,0],[0,0,0,0.1]])

        #LTS
        mean.append ([0.02, 0.25, -65, 2])
        covariance.append ([[0.0002,0,0,0],[0,0.0025,0,0],[0,0,0.65,0],[0,0,0,0.02]])
        #covariance.append ([[0.001,0,0,0],[0,0.0125,0,0],[0,0,3.25,0],[0,0,0,0.1]])
```

Figure A.1: Data generation 1



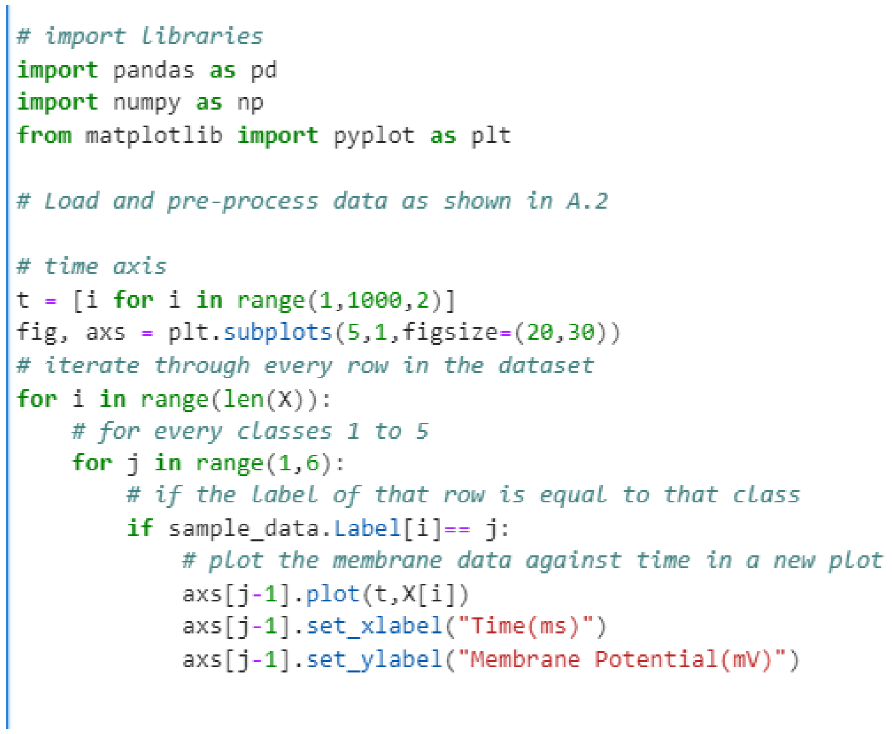
```python
#generate n sets of 4 random parameters from the multivariate distribution for each class

for neuron in range(len(mean)):
    gfg = np.random.multivariate_normal(mean[neuron], covariance[neuron], n)

    # identify negative instances of the first parameter and replace with 0.01
    for j in range(n):
        if gfg[j][0] < 0:
            gfg[j][0] = 0.01

        # assign parameters to the variables in the Izhikevich model
        a= gfg[j][0]
        b= gfg[j][1]
        c= gfg[j][2]
        d = gfg[j][3]

        # initialise membrane potential and recovery variable
        values =[]
        v= []
        u = []

        # set initial values
        v.append(-65)
        u.append(b*v[0])

        # thalamic input
        I=10

        # simulation of 1000ms
        for i in range(1,1000):

            # Izhikevich model equations
            v.append(v[i-1]+(0.04*v[i-1]**2+5*v[i-1]+140-u[i-1]+I))
            u.append(u[i-1]+a*(b*v[i]-u[i-1]))

            # after-spike resetting
            if v[i] >= -30:
                v[i]=c
                u[i]+=d

        # sampling every 2ms
        for z in range(0,len(v),2):
            values.append(v[z])

        # write to new row of CSV, with the sample membrane values under the data column and the neuron class under label
        writer.writerow([values,neuron+1])
```

Figure A.1: Data generation 2



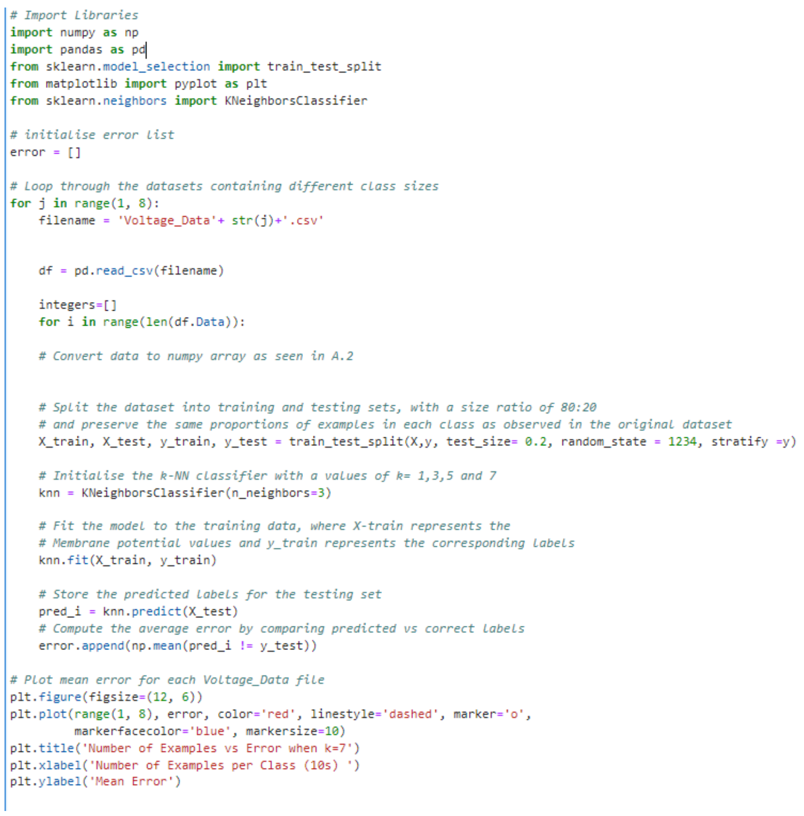
```python
# import libraries
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

# load csv file
sample_data= pd.read_csv('Testing.csv')

integers=[]

# loop through every row in the dataset
for i in range(sample_data.shape[0]):

    v= []
    # remove square brackets from the string containing the containing the comma separated values
    numbers = sample_data.Data[i].split('[',1)[1].split(']')[0]

    # remove commas
    lista = numbers.split(',')

    # convert string values into floats and store in a new list
    for i in lista:
        v.append(float(i))

    # store every row in a new list
    integers.append(v)

# convert to a numpy array
X = np.array(integers)
# convert labels to a numpy array
y = np.array(sample_data.Label)
```

Figure A.2: Data pre-processing



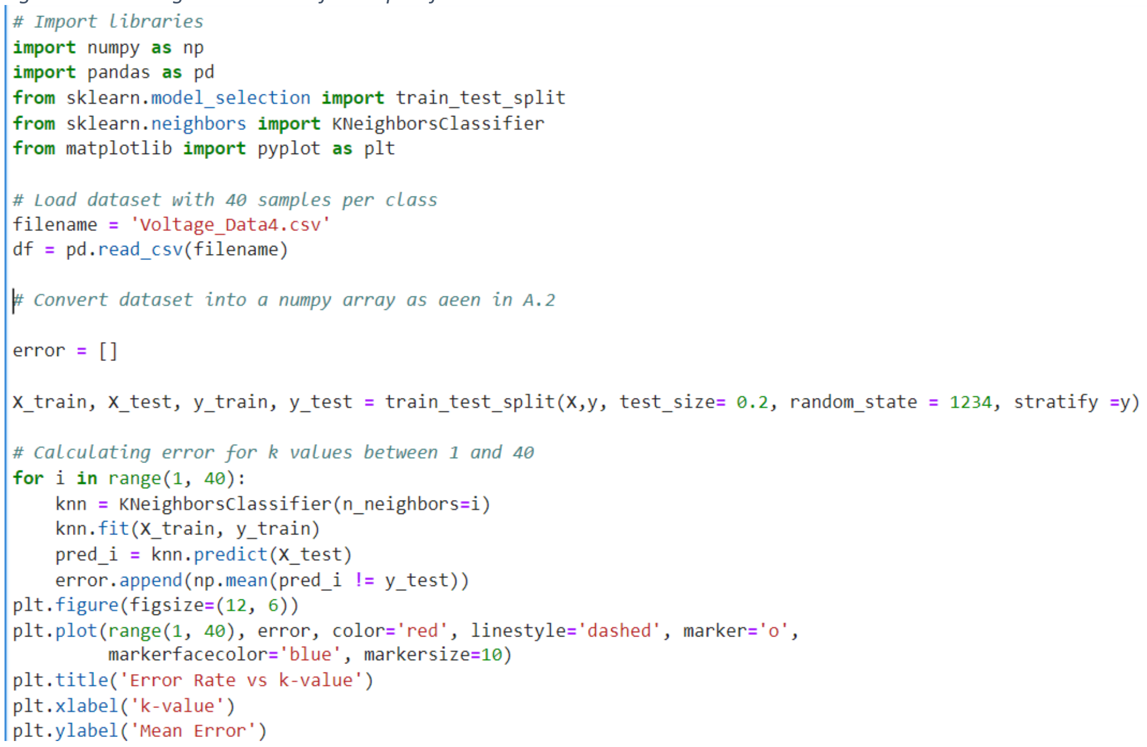
```python
# import libraries
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

# Load and pre-process data as shown in A.2

# time axis
t = [i for i in range(1,1000,2)]
fig, axs = plt.subplots(5,1,figsize=(20,30))
# iterate through every row in the dataset
for i in range(len(X)):
    # for every classes 1 to 5
    for j in range(1,6):
        # if the label of that row is equal to that class
        if sample_data.Label[i]== j:
            # plot the membrane data against time in a new plot
            axs[j-1].plot(t,X[i])
            axs[j-1].set_xlabel("Time(ms)")
            axs[j-1].set_ylabel("Membrane Potential(mV)")
```

Figure A.3: Visualization of the simulated spiking patterns



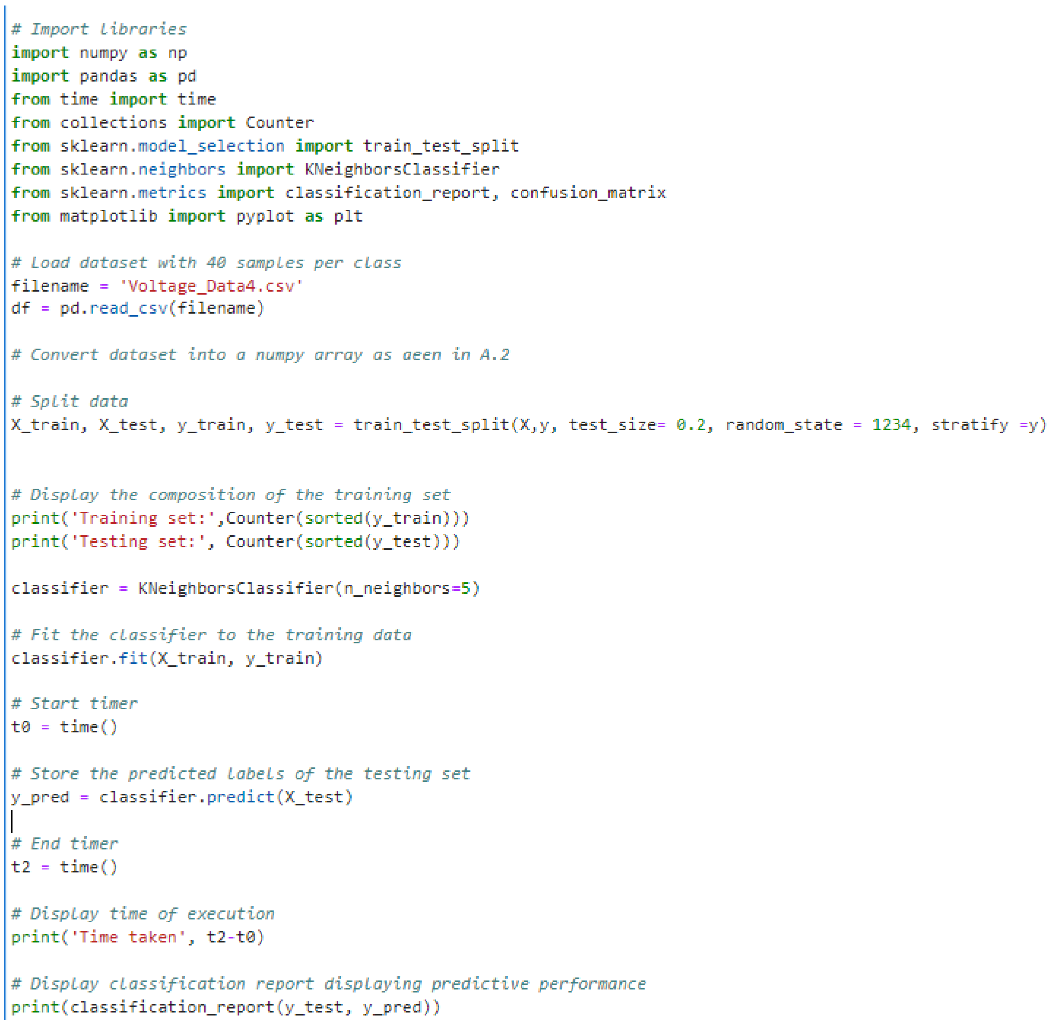
```python
# Import Libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

# initialise error list
error = []

# Loop through the datasets containing different class sizes
for j in range(1, 8):
    filename = 'Voltage_Data'+ str(j)+'.csv'


    df = pd.read_csv(filename)

    integers=[]
    for i in range(len(df.Data)):

    # Convert data to numpy array as seen in A.2


    # Split the dataset into training and testing sets, with a size ratio of 80:20
    # and preserve the same proportions of examples in each class as observed in the original dataset
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.2, random_state = 1234, stratify =y)

    # Initialise the k-NN classifier with a values of k= 1,3,5 and 7
    knn = KNeighborsClassifier(n_neighbors=3)

    # Fit the model to the training data, where X-train represents the
    # Membrane potential values and y_train represents the corresponding labels
    knn.fit(X_train, y_train)

    # Store the predicted labels for the testing set
    pred_i = knn.predict(X_test)
    # Compute the average error by comparing predicted vs correct labels
    error.append(np.mean(pred_i != y_test))

# Plot mean error for each Voltage_Data file
plt.figure(figsize=(12, 6))
plt.plot(range(1, 8), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Number of Examples vs Error when k=7')
plt.xlabel('Number of Examples per Class (10s) ')
plt.ylabel('Mean Error')
```

Figure A.4: Finding the number of examples for each class



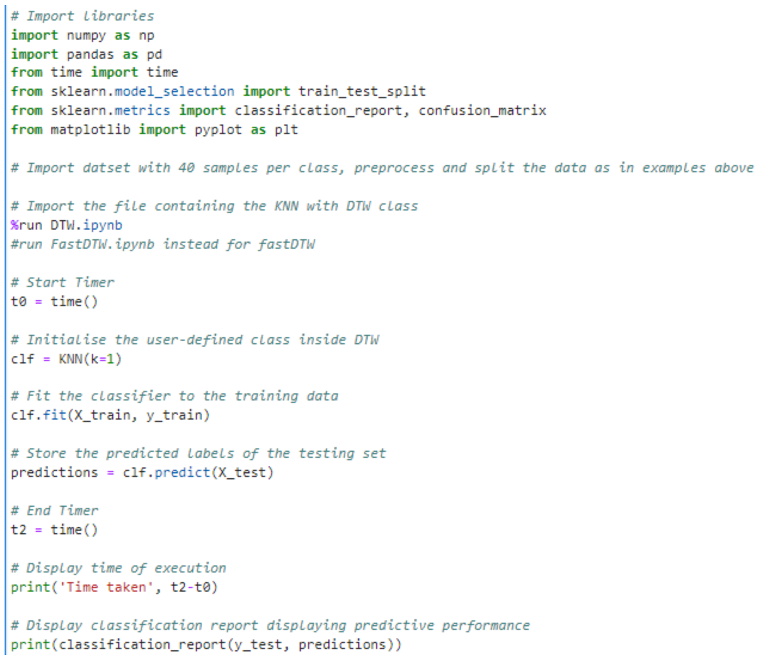
```python
# Import libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from matplotlib import pyplot as plt

# Load dataset with 40 samples per class
filename = 'Voltage_Data4.csv'
df = pd.read_csv(filename)

# Convert dataset into a numpy array as aeen in A.2

error = []

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.2, random_state = 1234, stratify =y)

# Calculating error for k values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate vs k-value')
plt.xlabel('k-value')
plt.ylabel('Mean Error')
```

Figure A.4: Finding the k-value



```
# Import libraries
import numpy as np
import pandas as pd
from time import time
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from matplotlib import pyplot as plt

# Load dataset with 40 samples per class
filename = 'Voltage_Data4.csv'
df = pd.read_csv(filename)

# Convert dataset into a numpy array as aeen in A.2

# Split data
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.2, random_state = 1234, stratify =y)


# Display the composition of the training set
print('Training set:',Counter(sorted(y_train)))
print('Testing set:', Counter(sorted(y_test)))

classifier = KNeighborsClassifier(n_neighbors=5)

# Fit the classifier to the training data
classifier.fit(X_train, y_train)

# Start timer
t0 = time()

# Store the predicted labels of the testing set
y_pred = classifier.predict(X_test)

# End timer
t2 = time()

# Display time of execution
print('Time taken', t2-t0)

# Display classification report displaying predictive performance
print(classification_report(y_test, y_pred))
```

Figure A.6 Visualizing the composition of the training set and training/ testing of the k-NN with Euclidean distance

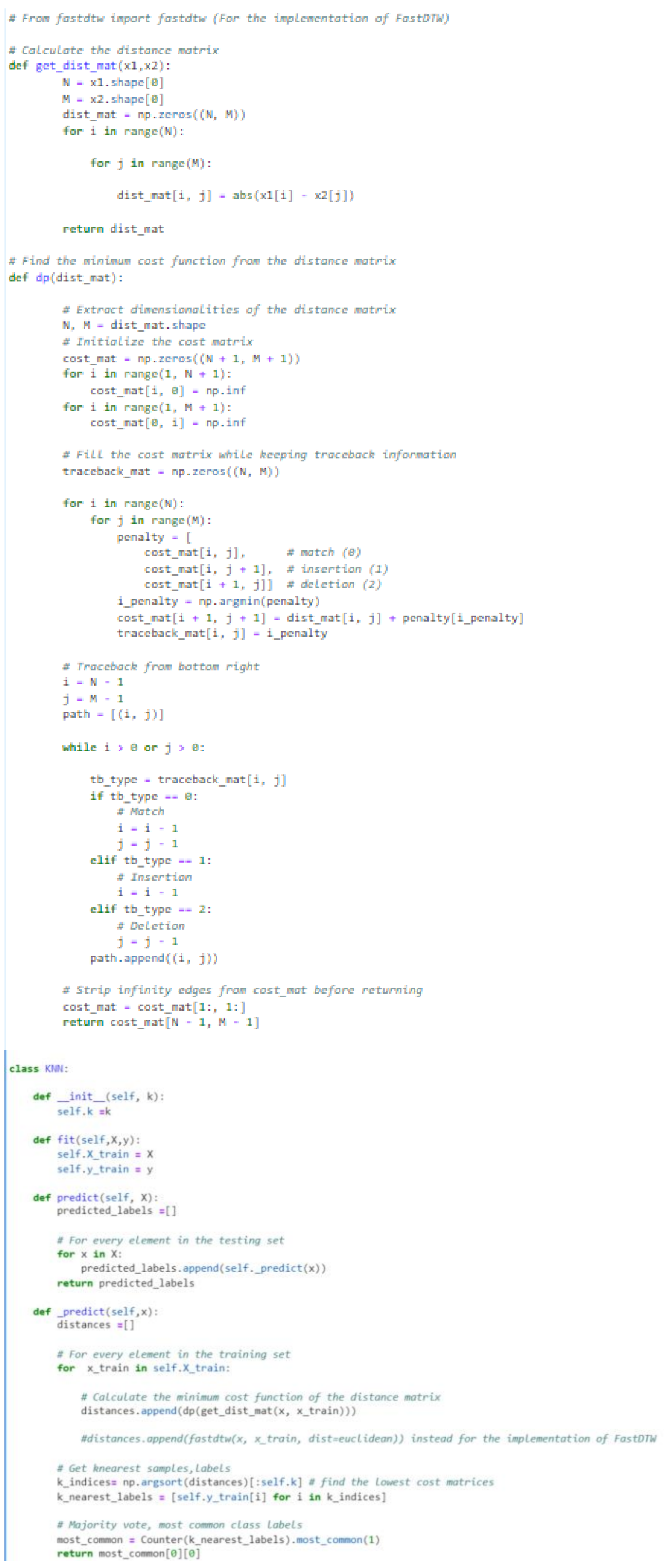


```
# Import libraries
import numpy as np
import pandas as pd
from time import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from matplotlib import pyplot as plt

# Import datset with 40 samples per class, preprocess and split the data as in examples above

# Import the file containing the KNN with DTW class
%run DTW.ipynb
#run FastDTW.ipynb instead for fastDTW

# Start Timer
t0 = time()

# Initialise the user-defined class inside DTW
clf = KNN(k=1)

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Store the predicted labels of the testing set
predictions = clf.predict(X_test)

# End Timer
t2 = time()

# Display time of execution
print('Time taken', t2-t0)

# Display classification report displaying predictive performance
print(classification_report(y_test, predictions))
```

Figure A.7: Training and testing of k-NN with DTW and FastDTW

Appendix B. User-Defined Class

This appendix contains the user-defined class for DTW and FastDTW.



```python
# From fastdtw import fastdtw (For the implementation of FastDTW)

# Calculate the distance matrix
def get_dist_mat(x1,x2):
        N = x1.shape[0]
        M = x2.shape[0]
        dist_mat = np.zeros((N, M))
        for i in range(N):

            for j in range(M):

                dist_mat[i, j] = abs(x1[i] - x2[j])

        return dist_mat

# Find the minimum cost function from the distance matrix
def dp(dist_mat):

        # Extract dimensionalities of the distance matrix
        N, M = dist_mat.shape
        # Initialize the cost matrix
        cost_mat = np.zeros((N + 1, M + 1))
        for i in range(1, N + 1):
            cost_mat[i, 0] = np.inf
        for i in range(1, M + 1):
            cost_mat[0, i] = np.inf

        # Fill the cost matrix while keeping traceback information
        traceback_mat = np.zeros((N, M))

        for i in range(N):
            for j in range(M):
                penalty = [
                    cost_mat[i, j],      # match (0)
                    cost_mat[i, j + 1],  # insertion (1)
                    cost_mat[i + 1, j]]  # deletion (2)
                i_penalty = np.argmin(penalty)
                cost_mat[i + 1, j + 1] = dist_mat[i, j] + penalty[i_penalty]
                traceback_mat[i, j] = i_penalty

        # Traceback from bottom right
        i = N - 1
        j = M - 1
        path = [(i, j)]

        while i > 0 or j > 0:

            tb_type = traceback_mat[i, j]
            if tb_type == 0:
                # Match
                i = i - 1
                j = j - 1
            elif tb_type == 1:
                # Insertion
                i = i - 1
            elif tb_type == 2:
                # Deletion
                j = j - 1
            path.append((i, j))

        # Strip infinity edges from cost_mat before returning
        cost_mat = cost_mat[1:, 1:]
        return cost_mat[N - 1, M - 1]


class KNN:

    def __init__(self, k):
        self.k =k

    def fit(self,X,y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predicted_labels =[]

        # For every element in the testing set
        for x in X:
            predicted_labels.append(self._predict(x))
        return predicted_labels

    def _predict(self,x):
        distances =[]

        # For every element in the training set
        for  x_train in self.X_train:

            # Calculate the minimum cost function of the distance matrix
            distances.append(dp(get_dist_mat(x, x_train)))

            #distances.append(fastdtw(x, x_train, dist=euclidean)) instead for the implementation of FastDTW

        # Get knearest samples,labels
        k_indices= np.argsort(distances)[:self.k] # find the lowest cost matrices
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        # Majority vote, most common class labels
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]
```

Figure B.1: User-defined DTW and FastDTW classes