UNIVERSITY OF SOUTHAMPTON

Faculty of Social Sciences
School of Mathematical Sciences

Optimizing supervised machine learning algorithms with practical applications

DOI: 10.1002/0470841559.ch1

by

Anthony James Dunn
BSc Mathematics
ORCiD: 0009-0006-1179-117X

A thesis for the degree of
Doctor of Philosophy

11/06/2023

University of Southampton

Abstract

Faculty of Social Sciences
School of Mathematical Sciences

Doctor of Philosophy

Optimizing supervised machine learning algorithms with practical applications

by Anthony James Dunn

Machine learning (ML) and artificial intelligence (AI) are rapidly growing fields with applications in many scientific domains. In this work we focus on supervised learning algorithms for prediction tasks in practical applications. Specifically we focus on two applications, predicting adverse events in low-carbon energy production and screening patients' eligibility for implantable defibrillators. We present optimised preprocessing methods, optimise hyperparameter selections and present our own bilevel optimisation model for simultaneous training and hyperparameter tuning. We first consider the problem of predicting infrequently occurring adverse events from time series data (Provided by Andigestion Ltd, a UK-based anaerobic digestion company, and a civil nuclear power plant in the UK) for which we propose a framework for modeling this problem as an imbalanced classification task and we construct and compare numerous models and sampling techniques. The models developed here could be integrated into a decision support tool for providing advanced warning of adverse events which can lead to significant commercial benefits. We then propose an AI tool for automated, prolonged screening of patients' implantable defibrillator eligibility which is created using real ECG data provided by our partners at the University Hospital Southampton. As we demonstrate in our experiments, this tool is capable of predicting patients' T:R ratios (a major indication of implantation eligibility) to within 0.0461 of their true values. This level of accuracy is sufficient to facilitate the automation of the measurement process. We show how this tool can enable cardiologists to perform 24-hour automated screenings, thus allowing them to better determine patients' eligibility for implantation. Finally, we formulate the bilevel problem of hyperparameter selection for non-linear kernel support vector machines via k-fold cross validation and propose an algorithm for solving it, for which we demonstrate some convergence properties. We provide a number of examples of this algorithm in use on a number of real data sets from the UCI repository.

Contents

List of Figures

List of Tables

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as:

   Coniglio, S., Dunn, A. J., and Zemkoho, A. B. (2020). Infrequent adverse event prediction in low carbon energy production using machine learning. Preprint available at arXiv:2001.06916. *Under review in* Machine Learning.

   Dunn, A. J., ElRefai, M. H., Roberts, P. R., Coniglio, S., Wiles, B. M., and Zemkoho, A. B. (2021). Deep learning methods for screening patients' S-ICD implantation eligibility. Artificial Intelligence in Medicine, *119:102139.*

   ElRefai, M., Abouelasaad, M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., Wiles, B. M., and Roberts, P. R. (2022b). Eligibility for subcutaneous implantable cardiac defibrillator utilising artificial intelligence and deep learning methods for prolonged screening: where is the cut-off? Europace, *24 (Supplement_1):euac053–447.*

   ElRefai, M., Abouelasaad, M., Wiles, B. M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., and Roberts, P. R. (2022c). Deep learning-based insights on T:R ratio behaviour during prolonged screening for S-ICD eligibility. Journal of Interventional Cardiac Electrophysiology, doi.org/10.1007/s10840-022-01245-6.

   ElRefai, M., Abouelasaad, M., Conibear, I., Wiles, B. M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., and Roberts, P. R. (2022a). The use of artificial intelligence and deep *learning methods in subcutaneous implantable cardioverter defibrillator screening to optimise selection in special patient populations.* Europace, *24(Supplement_1):euac053–448.*

Dunn, A. J., ElRefai, M. H., Roberts, P. R., Coniglio, S., Wiles, B. M., and Zemkoho, A. B. (2022). Deep learning and hyperparameter optimization for assessing one's eligibility for a subcutaneous implantable cardioverter-defibrillator. Preprint available at https://optimization-online.org/?p=21066. *Under review in* Annals of Operations Research.

Signed: Anthony Dunn Date: 11/06/2023

Acknowledgements

I would like to thank my supervisors Alain Zemkoho and Stefano Coniglio for their advice, guidance and contributions to my research as well as Decision Analysis Services Ltd, who co-funded by PhD. In particular, I would like to thank Siôn Cave for supporting me during my studies. I would also like to thank all of the collaborators I have had on the various projects I have worked on, including our partners Mohamed ElRefai and Paul R. Roberts of University Hospital Southampton and Benedict W. Wiles of St George's University Hospitals NHS Foundation Trust, for working with me on patient screening for S-ICD implantation as well as Qingna Li of Beijing Institute of Technology for her contributions to my work on bilevel optimisation.

Definitions and Abbreviations

X	The data points of a data set.
y	The labels corresponding to X.
n	The number of data points in X.
d	The number of features comprising each data point
$[n]$	$= \{1, \quad 2, \quad \ldots, \quad n\}$.
$\widehat{X}$	The data points of the training set.
$\hat{y}$	The labels of the training set.
$\hat{n}$	The number of data points in the training set.
$\bar{X}$	The data points of the validation set.
$\bar{y}$	The labels of the validation set.
$\bar{n}$	The number of data points in the validation set.
k	The number of folds used for k-fold CV.
$\mathcal{L}$	A generic loss function.
$\bar{X}^i$	The data points belonging to the i^{th} fold in k-fold CV.
$\bar{y}^i$	The labels belonging to the i^{th} fold in k-fold CV.
$\bar{n}^i$	The number of data points belonging to the i^{th} fold in k-fold CV.
$\widehat{X}^i$	All of the data points which do not belong to the i^{th} fold in k-fold CV.
$\hat{y}^i$	All of the labels which do not belong to the i^{th} fold in k-fold CV.
$\hat{n}^i$	The number of data points which do not belong to the i^{th} fold in k-fold CV.
λ	A hyperparameter vector.
Λ	The set of all possible hyperparameter combinations.

Chapter 2

f	True, unknown, labeling function where $f(X) = y$.
h	The prediction rule of a classifier where $h(X_i)$ is the predicted label for X_i for all $i \in [n]$.
τ	The delay interval in PSR transformations.

P	The normalised phase space matrix.
ξ	Slack variables in SVMs.
ω	Vector defining the slope of the hyperplane constituting an SVM's decision rule.
b	Constant defining the intercept of the hyperplane constituting an SVM's decision rule.
C	A hyperparameter of the SVM training algorithm determining the relative importance of maximising the separation margin and minimising the sum of the slack variables.
ϕ	The feature space embedding function.
K	The kernel function.
γ	A hyperparameter of the SVM training algorithm determining how strongly related two points are related to one another based on the distance between them.
α	The model parameters of the dual formulation of the SVM training algorithm.
θ	The model parameters of a neural network.
g	The gradient of a loss function with respect to the model parameters θ.
m	The size of a minibatch in the minibatch SGD.
$\check{X}$	The m data points in a minibatch.
$\check{y}$	The m labels in a minibatch.
ϵ	The learning rate of SGD.
$\hat{g}$	An estimate of the gradient of a loss function with respect to the model parameters θ, calculated using a minibatch.

Chapter 3

$\tilde{X}$	The patterns, formed by leading the time series data points.
τ	The number of lagged data points included with each data point to create a pattern.
ω	The maximum amount of time before an event at which the system begins to show signs that an adverse event is imminent.
$\tilde{y}$	The warning labels, which have value 1 if an adverse event occurs within ω time intervals, and 0 otherwise.
m	The minimum number of time intervals required to carry out the necessary maintenance to prevent an impending adverse event.
β	The number of points included in each block.
T	The set of indexes defining a block when using our proposed block sampling.
T'	The set of indexes defining a block excluding the patterns for which the event is already occurring.

B	A more general notation to refer to either T or T' depending on whether the event patterns are being included in our model training.

Chapter 4

T_i	The index of the i^{th} T wave in an ECG segment.
R_i	The index of the i^{th} R wave in an ECG segment.

Chapter 5

k	The number of folds used for k-fold CV.
α	The model parameters of the dual formulation of the SVM training problem.
ω	The parameter of the primal formulation of the SVM training problem defining the slope of the hyperplane constituting the SVMs decision rule.
b	The parameter of the primal formulation of the SVM training problem defining the intercept of the hyperplane constituting the SVMs decision rule.
C	A hyperparameter of the SVM training problem determining the relative importance of maximising the separation margin and minimising the sum of the slack variables.
γ	A hyperparameter of the SVM training algorithm determining how strongly related two points are related to one another based on the distance between them.
ξ	Slack variables in SVMs.
ϕ	The feature space embedding function.
K	The kernel function.
ζ	A small perturbation used to ensure that our max approximation is smooth.
Ω	A function used to control the contribution of each point to the calculation of b.
τ	A small perturbation used to avoid dividing by 0 in the calculation of b.
e^i	A vector with elements all ones in $\mathbb{R}^{\hat{n}^i}$.
m	The sum of the number of data points in each of the i training sets.
v	A stacked vector containing all of the parameters of Problem MPEC.
$\dot{m}$	$= 2 + k + 2m$, the number of elements in v.
t	A parameter controlling how relaxed the compilmentarity constraints are in Problem 5.23.
g	The number of hyperparameter values on each range of values

defining the $g \times g$ grid of hyperparameter combinations used in grid-search.

Initialisms

AB	AdaBoost
AD	Anaerobic Digestion
AI	Artificial Intelligence
BCI	Brain Computer Interfaces
BRF	Balanced Random Forest
CNN	Convolutional Neural Network
CPR	Cardiopulmonary Resuscitation
CV	Cross Validation
CVD	Cardiovascular Disease
DT	Decision Tree
ECG	Electrocardiogram
FN	False Negatives
FP	False Positives
FPR	False Positive Rate
GB	Gradient Boosting
GNB	Gaussian Naive Bayes
ICD	Implantable Cardioverter-Defibrillator
KBS	Knowledge-Based System
KNN	K Nearest Neighbors
LR	Logistic Regression
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NLP	Natural Language Processing
NPP	Nuclear Power Production
PSR	Phase Space Reconstructions
QDA	Quadratic Discriminant Analysis
RBF	Radial Basis Function
RF	Random Forest
RMSE	Root Mean Squared Error
ROC	Receiver Operating Characteristic
ROS	Random Oversampling
RUS	Random Undersampling
SCD	Sudden Cardiac Death
SGD	Stochastic Gradient Descent

S-ICD	Subcutaneous Implantable Cardioverter-Defibrillator
SQP	Sequential Quadratic Programming
SVC	Support Vector Classification
SVM	Support Vector Machine
SVR	Support Vector Regression
TN	True Negatives
TP	True Positives
TPR	True Positive Rate
TV-ICDs	Transvenous ICDs
TWOS	T Wave Over Sensing
VA	Ventricular Arrhythmias

Chapter 1

Introduction

Artificial Intelligence (AI) is a rapidly developing field with applications in almost every domain, some examples of which are Natural Language Processing (NLP), time series forecasting, anomaly detection and image recognition. With more data being collected than ever before and the development of a greater understanding of the capabilities of AI across all scientific disciplines, the number of problems for which AI solutions can be constructed is ever increasing. At the center of the majority of AI tools lies one or more Machine Learning (ML) models (Alpaydin, 2016). Machine learning algorithms are designed to construct models capable of making predictions without explicitly being programmed to do so.

Two of the main classes of ML problems are: *supervised* and *unsupervised* learning problems. In unsupervised learning, only data is provided to the learning algorithm, whose task is to provide some insight into the relationships and trends prevalent within the data. In supervised learning, each data point has a corresponding label, and the task of the learning algorithm is to construct a model capable of accurately predicting labels for unseen data points. For a more detailed overview of these ML problems we refer the reader to Alpaydin (2020). In this work we focus on prediction algorithms and applications in which we construct predictive models. For this reason, we focus on supervised learning models. For practical use, these predictive models must be constructed and evaluated to provide optimal performance in the specific domain to which they will be applied and, ultimately, must be integrated into an AI tool to be used by experts in the relative domain.

There are numerous algorithms for constructing ML models to complete supervised learning tasks. The construction of a ML model using any of these algorithms is an optimisation problem wherein the parameters of a model must be selected optimally such that some loss function (measuring the errors in the model's predictions of the labels) is minimised. For a recent review paper detailing numerous popular ML algorithms, we refer the reader to Mahesh (2020). The objective function and constraints of these optimisation problems often depend on *hyperparameters* (parameters of the model construction algorithm which are unchanged during the construction process). The careful selection of these hyperparameters is therefore crucial for optimal performance (Probst et al., 2019). A detailed overview of how model construction algorithms are typically chosen and their hyperparameters tuned to optimise model performance can be found in Chapter 2.

In this work, we optimise numerous aspects of developing AI tools for predictive tasks (both classification and regression) including optimising the preprocessing methods, algorithm selection, hyperparameter optimisation and proposing novel algorithms for model construction. In Chapter 3 we address the problem of predicting adverse events in low carbon energy production. This involves modeling the problem as an imbalanced classification task, optimising the preprocessing, model selection, hyperparameter selection and techniques use to address class imbalance. In Chapter 4 we create an AI tool to screen patients' eligibility for Subcutaneous Implantable Cardioverter-Defibrillator (S-ICD) implantation. We formulate a regression problem of predicting a key indicator of S-ICD implantation eligibility from relevant patient data and construct a deep learning model to solve this task. Prior to model construction, we develop a sophisticated preprocessing scheme. We then optimise the architecture of the deep learning model and compared the use of a number of stochastic gradient descent based algorithms for solving the model construction problem discussed previously (see Subsection 2.2.2 for an overview of neural network architectures and the use of gradient descent for model training) before ultimately integrating this model into an AI tool for use by clinicians. Finally in Chapter 5 we model the problem of tuning the hyperparameters of Radial Basis Function (RBF) kernel Support Vector Machines (SVMs)—a variant of SVM which is better suited to practical applications—via Cross Validation (CV) as a bilevel optimisation problem and propose an algorithm for solving this problem. We demonstrate the effectiveness of this method and algorithm on a number of real data sets.

1.1 Adverse event prediction in low carbon energy production

The work in Chapter 3 is motivated by two practical problems that occur in the production process of bio and nuclear energy. The first one is foam formation (or *foaming*) in Anaerobic Digestion (AD). Foaming arises in the context of bioenergy production (i.e., biogas, more precisely) within the AD process, where sludge, comprising of food waste, agricultural waste, or crop feed (Ganidi et al., 2009), is fed into a digester in which it is broken down by microorganisms, releasing biogas which is then collected from the top of the digester and burned to produce energy. Under normal operations, gas bubbles rise from feed sludge as it is digested and then collapse releasing biogas.

This biogas is then burned in order to power a turbine which produces electricity. Under certain conditions, the gas bubbles may take longer to collapse than it takes for new bubbles to form, resulting in the formation of *foam* (Ganidi et al., 2009).

Foaming is often a major issue in AD as it can block the gas outlet, resulting in the digester having to be shut down for cleaning operations to take place. As cleaning can take days (during which the digester cannot be used for biogas production), foam formation can have a considerable impact on energy output. In some extreme cases, the pressure build-up caused by the gas outlet and pressure release valve becoming blocked can even lead to the roof of the digester being blown off. Foaming can be treated using an *anti-foaming agent* (see, e.g., Yang et al. (2021)), which, upon introduction into the digester, causes the bubbles to collapse. This, however, requires the foaming phenomenon to be detected early enough to prevent it from causing some of the aforementioned serious problems.

The second problem is that of *condenser tube leakage* in civil Nuclear Power Production (NPP). In the nuclear power production process, condenser tube leakage occurs in the operation of steam turbines for converting thermal energy from steam to electrical power. Under normal operations, steam is generated in an array of boilers and passed through the turbine rotors causing them to rotate, thus generating power—a turbine is said to be *on-load* if it is rotating and hence producing power, and *off-load* if it is not. The steam is then passed through a condenser, where it is cooled down to liquid water and recirculated. The condenser consists of thousands of titanium tubes containing sea water, which acts as the primary coolant. In normal operating conditions, the sea water circuit remains isolated from the steam circuit. It is, however, possible for a leak to form in one of the condenser tubes, causing sea water to contaminate the steam circuit. Such a leak is typically caused by the formation of deposits on heat-transfer surfaces (*fouling*)—for more details on condenser fouling within nuclear power production, see, e.g., Müller-Steinhagen (1999).

In the event of a leak in one of the tubes of the condenser, the steam turbine is automatically tripped, and remedial actions are taken to fix the leak off-load. Besides maintenance costs, an unplanned trip causes a loss of revenue proportional to the time needed to fix the leak and restart the power plant, which can be quite substantial. If warning of a tube leak were provided prior to its advent, the power output could be reduced and the turbine in question isolated and repaired on-load before the plant is brought back to full power, resulting in much smaller generation losses. Based on discussion with (Lewis, 2019), the lead consultant at DAS Ltd who provided us with the data sets used in this study, we understand that a predictive functionality could yield savings of up to £350,000 per day of warning provided for a standard nuclear plant and per tube leak. To the best of our knowledge, no machine learning techniques have been developed so far to provide advanced warning of condenser tube leaks, a goal which we set out to achieve with the work in Chapter 3.

The disruption in the corresponding energy production process and the damages that can be caused by the aforementioned adverse events result in significant losses for the concerned plants (Ganidi et al., 2009; Kanu et al., 2018; Cattant et al., 2008; Coit, 1980). Being able to predict these events before they occur can give the plant operators time to take actions to mitigate their effects, which can include damages to the environment and significant financial losses. Hence, our main goal in this work is to use time-series operating conditions, which are incidentally recorded for use in normal operation of a plant, to develop a machine learning tool that can generate such predictions.

We can observe that the structure of the AD and NPP related time series is strikingly similar. For each of them, we have sporadic adverse events occurring infrequently at, currently, unpredictable time intervals (the occurrence of these events has been identified based on records from the plant operators and various other data indicators) and a large amount of time-series data consisting of regular readings of operating conditions. Figure 1.1 better illustrates this similarity between the two time series. The top graph shows occurrences of condenser tube leaks from 2009 to 2019 in an NPP plant. The bottom graph reports foaming events that occurred between December 2015 and July 2017. The figure zooms in on two sections of the data, showing the behaviour of the respective operating conditions in the run-up to an event. As one can see, the plots do not exhibit a clear trend or a clear pattern leading to the occurrence of an event, indicating that the prediction task at hand is a non-trivial one. Furthermore, due to this infrequent nature of their occurrence, the data sets of the two applications that we consider suffer from a large *class imbalance*, i.e, they contain a far greater amount of data which is not associated to the event than that which is. Class imbalance is known to be a problem for most classification algorithms, as their solutions are likely to over-fit the majority class (Akbani et al., 2004a; Japkowicz and Stephen, 2002). The class imbalance problem is further discussed in Subsection 3.1. In our case, this would result in predicting the non-occurrence of the adverse event in almost all the cases, rendering the prediction *de facto* useless. These similarities motivate our choice to model these two prediction problems using the single framework proposed in Chapter 3.

1.2 Screening for S-ICD implantation

Sudden Cardiac Death (SCD) is one of the leading causes of death in the modern world. Most of these deaths can be attributed to Ventricular Arrhythmias (VA) (Adabag et al., 2010). The key to survival in patients affected by VA is adequate Cardiopulmonary Resuscitation (CPR) and early defibrillation (Hazinski et al., 2005). Medical guidelines recommend the use of Implantable Cardioverter-Defibrillators (ICDs) for prevention of SCD triggered by VA in high risk populations (Kusumoto et al., 2018; Members et al., 2015; Priori et al., 2016). Conventional transvenous ICDs (TV-ICDs) consist of a *can*

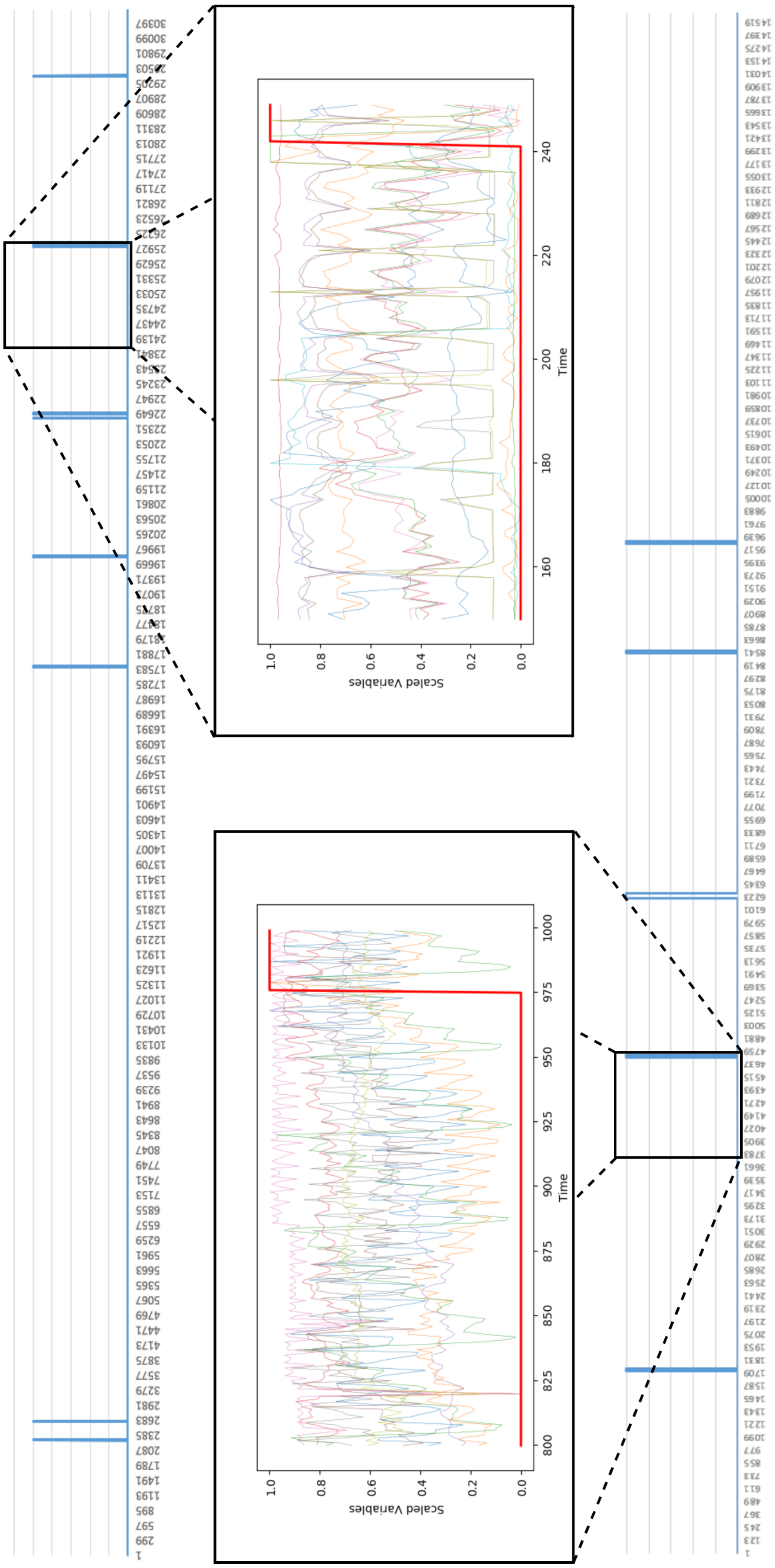

FIGURE 1.1: Scaled condenser variables (top) and anaerobic digester variables (bottom).

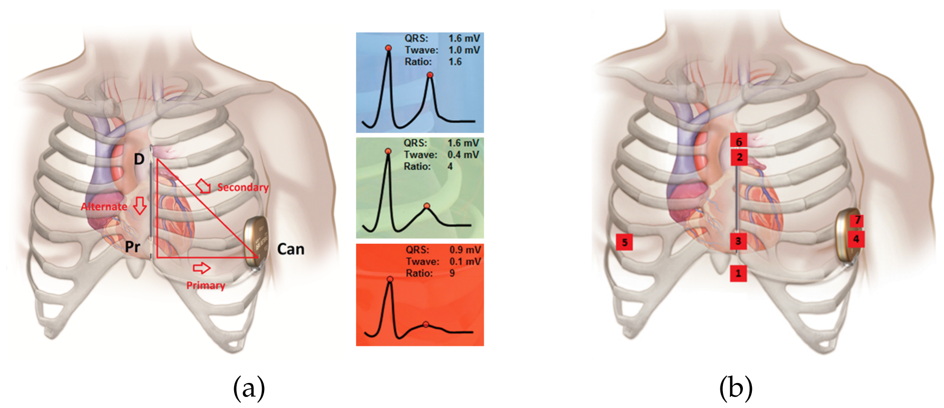

FIGURE 1.2: (a) S-ICD sensing electrodes and vectors between them. An implanted S-ICD with underlying anatomical features showing the location of the can (pulse generator), the proximal (Pr) and distal (D) sensing electrodes and the shocking coil (located between the electrodes). (b) Holter recorder surface ECG positions. Images (prior to annotation) © Boston Scientific Corporation or its affiliates. Reproduced with permission.

and transvenous leads implanted into the right ventricle to treat the arrhythmia by delivering a voltage shock. TV-ICDs are associated with the risk of complications with potentially fatal consequences. The Subcutaneous ICD (S-ICD), which comprises an electrically active can and a single subcutaneous *lead*[1] (see Figure 1.2a), was designed to avoid complications of the TV-ICD by utilising a totally avascular approach. The sensing mechanism of the S-ICD has been shown to be equally effective to that of the TV-ICD (Boersma et al., 2017) and demonstrated less incidence of device-related complications when compared with conventional ICDs (Knops et al., 2020). However, a consequence of the algorithm used by the S-ICD to detect VA is an inherent risk of T Wave Over Sensing (TWOS). TWOS can occur when the amplitudes of the T and R waves are similar causing the T wave to be misinterpreted as a second R wave. Figure 1.3(b) shows a diagram of the PQRST complex—the Electrocardiogram (ECG) of a single heartbeat—comprised of 5 main waveforms, i.e., the P, Q, R, S, and T waves. This apparent doubling in heart rate can be incorrectly identified by the S-ICD as an episode of ventricular arrhythmia. Such an error can cause the S-ICD to deliver an inappropriate shock. Inappropriate shocks are associated with increased morbidity and mortality (van Rees et al., 2011).

Not all patients are eligible for S-ICD therapy and eligibility is identified during a pre-implant screening process that is undertaken in potential S-ICD recipients. Surface ECGs are used as a surrogate marker of future S-ICD vectors (as shown in Figure 1.2a) to non-invasively assess S-ICD eligibility. These short, three-lead ECG recordings, are

[1]Using a single-lead, the S-ICD is able to record ECG on 3 vectors—see Figure 1.2(a). When using a Holter ECG recording to assess patients' S-ICD implantation eligibility, numerous leads are used. We refer to a single-lead of a Holter recording as the ECG recording for a single vector.

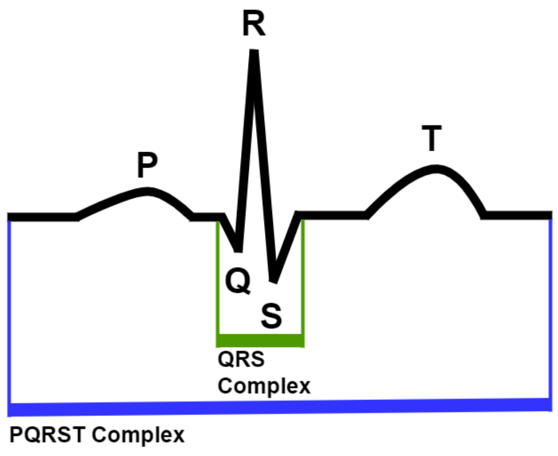

FIGURE 1.3: Diagram of a single PQRST complex, comprised of the P, Q, R, S, and T waves. The figure also labels the QRS complex (a section of the PQRST complex featuring the Q, R, and S waves only).

analyzed to assess the patient's risk of TWOS. Vectors with a lower *T:R ratio* (the ratio between the amplitude of the T wave and that of the R wave) are more likely to pass the screening, while patients with an ECG morphology that does not meet this screening criteria are deemed at high risk of TWOS and are ineligible for an S-ICD. Despite the current screening practice, the most common cause of inappropriate shocks in patients implanted with S-ICDs remains TWOS (Knops et al., 2020). The T:R ratio—a major predictor of S-ICD eligibility—is not fixed in the same individual because of frequently observed temporal variations in the amplitudes of the R and T waves which are influenced by multiple factors (Madias et al., 2001; Madias, 2005; Fosbøl et al., 2008; Assanelli et al., 2013). In patients with S-ICDs, variations in the T:R ratio often go undetected as "silent" episodes of TWOS in spite of carrying a considerable risk of leading to the development of clinically relevant oversensing, which manifests in inappropriate shocks.

Under the current screening process, ECG electrodes are placed on the chest wall using the anatomical landmarks which correspond to the location of the sensing electrodes of the potential future S-ICD implantation shown in Figure 1.2b. A roughly 10 second three-lead ECG recording (comprised of several PQRST complexes), corresponding to the three vectors utilised by the S-ICD, is evaluated using the manual S-ICD screening tool shown in Figure 1.4. The patient is deemed eligible for S-ICD implantation if, for at least one of the leads (representing one of the S-ICD vectors), the patient's QRST complex (the PQRST complex excluding the P wave) sits entirely within the boundary of the template. These templates correspond roughly to a maximum acceptable T:R ratio of 1:3. As mentioned previously, the T:R ratio could fluctuate, as the amplitudes of both R waves and T waves may vary according to other factors (e.g, electrolyte levels). Due to the short duration of the current screening, it is possible that a patient with a

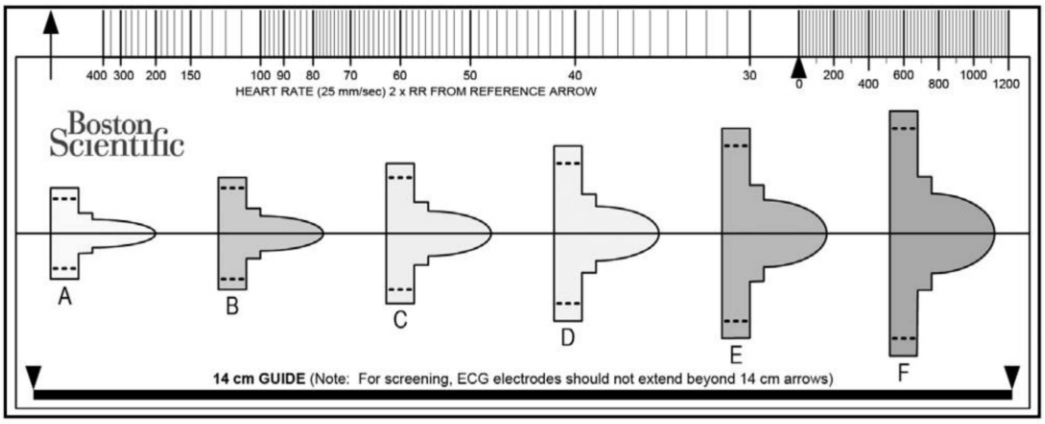
400 300 200 150 100 90 80 70 60 50 40 30 0 200 400 600 800 1000 1200
HEART RATE (25 mm/sec) 2 x RR FROM REFERENCE ARROW
Boston Scientific
A
B
C
D
E
F
14 cm GUIDE (Note: For screening, ECG electrodes should not extend beyond 14 cm arrows)

FIGURE 1.4: S-ICD screening tool. The recorded QRST morphology in every vector is then compared to the acceptable templates. The template is aligned to the isoelectric line of the ECG, and the QRST complexes are viewed through the appropriately sized template. The R wave peak of the ECG must be placed within either hashed box (positive or negative) of any template. A vector passes screening if the remainder of the QRST complex sits entirely within the boundary of the template. (This manual screening method is now largely replaced by the manufacturer with an automatic screening tool following the same principals)

typically high T:R ratio could pass this screening and likewise a patient with a typically low T:R ratio could fail it.

It is our goal in Chapter4 to propose an automated tool for measuring T:R ratios which can be used to screen patients over a much longer period of time, reducing the likelihood of the recording during the screening period not being representative of the patients normal ECG morphology, thus enabling clinicians to better scrutinise patients S-ICD implantation eligibility.

1.3 Bilevel hyperparameter optimisation for non-linear support vector machines

Support Vector Machines (SVMs) (Hearst et al., 1998) are one of the models used for both regression and classification. While the most basic implementation of an SVM has no hyperparameters, many common adaptations of the model such as soft-margin SVMs and Kernel SVMs introduce hyperparameters - variables of the model which must be selected by the user before fitting the model to data (also referred to as *training*). The standard approach to selecting values for these hyperparameters, while avoiding over-fitting them to the training data, is to perform Cross Validation (CV) (Vanwinckelen and Blockeel, 2012), wherein a portion of the total data set (the *validation set*) is held out from training and the hyperparameter values are selected such that when the model is fit to the training data the model's predictions for the hereto unseen validation set minimise some loss function. For classification, the loss function typically used here is a count of the number of miss-classified points.

Clearly, this is a bilevel optimisation problem. We are minimising a loss function of the model's predictions for the validation set subject to the constraint that the model parameters correspond to a solution to the training problem. While there have been multiple works which have approached hyperparameter tuning for SVMs via CV as a bilevel problem (Bennett et al., 2008, 2006; Moore et al., 2009; Kunapuli et al., 2008; Li et al., 2021), this bilevel problem is complex to formulate and solve. For this reason, many popular methods for hyperparameter selection (such as grid-search, random-search and Bayesian optimisation) use various schemes for sampling hyperparameter combinations and then for each of these hyperparameter combinations, they solve the training problem and evaluate the value of the loss function of the resulting trained model's predictions for the validation set. For an overview of how other methods attempt to approximate solutions to the hyperparameter tuning via CV problem, see Subsection 2.3. While such methods do a good job of finding hyperparameters which work well, it is not possible for these methods to find an actual solution to the problem as the solution will always lie in the space in between the sampled hyperparameter combinations. Additionally, the number of dimensions of the hyperparameter space which these methods endeavor to search increases with each additional hyperparameter, resulting in either very large run-times or poor quality solutions.

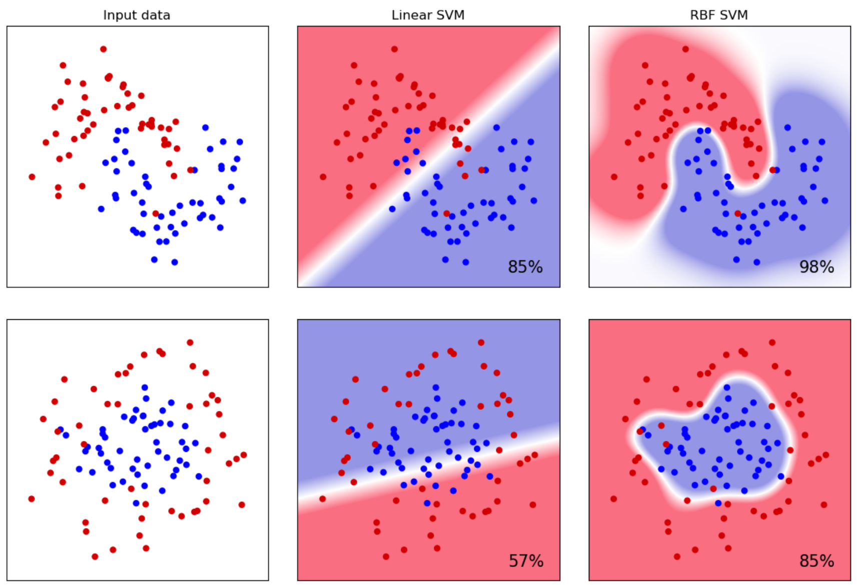

FIGURE 1.5: Plots showing the prediction rules of linear kernel and RBF kernel SVMs trained on two synthetic 2D data sets. Each 2D data point of the training set is plotted in the feature space (left) with the point's class represented by its colour. The label which would be assigned by linear kernel SVMs (centre) and RBF kernel SVMs (Right) for each area of the feature space is shown in red and blue. The accuracy of each model's predictions for an unseen validation set is then seen in the bottom right corner of these plots.

In Subsection 2.3 we overview the literature on hyperparameter tuning via CV as a bilevel optimisation problem. All of the works which tune hyperparameters for the soft-margin SVM focus on the linear kernel SVM. In Chapter 5 we focus on the Radial Basis Function (RBF) kernel, also known as the Gaussian Kernel. As we detail in Section 5.1, non-linear kernel embedding (such as the RBF kernel) allows SVMs to capture non linear behavior in the data and the RBF kernel has been shown to outperform other kernel functions in numerous applications (Nanda et al., 2018; Yekkehkhany et al., 2014; Feizizadeh et al., 2017; Hong et al., 2017; Tbarki et al., 2016; Garrett et al., 2003) and is generally recommended for practical applications (Prajapati and Patle, 2010). This is illustrated in Figure 1.5, where linear kernel SVMs and RBF kernel SVMs are trained on two synthetic data sets each with just two features. However, the use of the BRF kernel does not come without its drawbacks. Utilising this kernel embedding causes the primal form of the training problem to contain a function which maps into infinite dimensions. Our goal in Chapter 5 is to model this problem in such a way that we can perform a single-level reformulation of the bilevel problem and develop an algorithm to solve the resulting problem. Finally, we demonstrate the capabilities of this algorithm by presenting its performance on a number of real data sets.

1.4 Contributions of the research

In this subsection we overview the main contributions made in each chapter, as well as the publications that each chapter has led to.

1.4.1 Imbalanced data classification for infrequent adverse event prediction in low-carbon energy production

In Chapter 3, we consider the problem of predicting the occurrence of infrequent adverse events in low-carbon energy production systems. We cast the corresponding learning task as an imbalanced classification problem (a specific type of classification problem for which specialised techniques and models are required) and propose a framework for solving it that is capable of leveraging different classifiers in order to predict the occurrence of adverse events before they take place. Central to this framework is a bespoke informed undersampling method designed to address class imbalance as well as a number of other issues arising in infrequent adverse event prediction problems. This work is motivated by two practical engineering applications arising in the production processes of low-carbon energy, i.e., foam formation in AD and condenser tube leakage in the steam condenser in NPP. The results of an extensive set of computational experiments show the effectiveness of the techniques we propose.

The vision that we have for the research conducted in Chapter 3 is that the resulting tool could be installed in a plant to be used as a decision support system, receiving live data and enhancing the monitoring process of a plant in order to alert plant operators to the potential occurrence of the aforementioned adverse events. While a lot of work has been done in the literature around the use of times-series forecasting for decision support; see e.g., Jabeur et al. (2020); Hansen et al. (2006); Watson et al. (2020); Shi et al. (2019); Zhu et al. (2018), to the best of our knowledge the two applications considered here have not been addressed in the decision support literature.

The impact of the work in Chapter 3 goes beyond the two applications mentioned above as foaming and condenser leakage occur in various other types of chemical processes and power plants. For instance, it is well-known that foaming is a common operational problem in many wastewater treatment plants; see, e.g., De Los Reyes (2010); Müller-Steinhagen (1999). Similarly, it has been well-documented that condenser tube leakage can occur in any power plant using a condenser to cool water in its processing system (Daniels, 2010; Coit, 1980; Lawrence et al., 1977). For example, steam condensers are also used in thermoelectric power generation (Walker et al., 2012). The framework we propose is not limited to applications involving AD or condensers but can be applied to any system for which time-series data is recorded and in which infrequent adverse events occur.

In summary, the main contributions of Chapter 3 are three-fold:

1. We model the problem of predicting infrequent adverse events occurring in low-carbon energy production processes as a classification problem in such a way that some issues caused by the inherent class imbalance are mitigated.

2. We optimise the performance of existing machine learning algorithms by introducing block sampling, a new technique for undersampling blocks of relevant data for use in model construction and evaluation.

3. We illustrate the effectiveness of our approach by applying the work from points 1 and 2 above to practical data sets from some AD and NPP plants in the United Kingdom (UK).

The work in Chapter 3 has led to a paper:

Coniglio, S., Dunn, A. J., and Zemkoho, A. B. (2020). Infrequent adverse event prediction in low carbon energy production using machine learning. Preprint available at arXiv:2001.06916. *Under review in* Machine Learning.

1.4.2 Deep learning methods for screening a patient's Subcutaneous Implantable Cardioverter-Defibrillator (S-ICD) implantation eligibility

In Chapter 4, we propose an accurate, reliable and robust method that utilises the concept of prolonged screening for Subcutaneous Implantable Cardioverter-Defibrillator

(S-ICD) eligibility to better scrutinise the selection criteria in an attempt to find a cohort of patients with low probability of TWOS and inappropriate shocks. Central to this prolonged screening is a deep learning based method for predicting the T:R ratio—the main determinant of S-ICD eligibility—of a 10-second segment of a single-lead ECG. Solving this regression problem is at the core of our work in Chapter 4.

The aim of our proposed screening method is to use the model developed in this work to analyse the T:R ratio of each single-lead 10-second ECG segment within a three-lead 24-hour ECG recording to determine if any of the three leads have a suitably low risk of TWOS. For a given lead, should a patient have a T:R ratio above 1:3 for a continuous period of at least 20 seconds (the duration of TWOS at which the S-ICD would deliver an inappropriate shock), that lead would fail the screening. If all three leads fail the screening, the patient would be deemed not eligible for an S-ICD. Temporal variations in the T:R ratio make the current 10-second screening process unreliable. Our proposed method allows for a much more robust screening, as it would allow for the analysis of variations in the T:R ratio over a 24-hour period. Subsection 4.4 gives an example of how the model for predicting T:R ratios from ECG segments we develop can be used within a screening tool to access patients' eligibility for S-ICD implantation.

In order to ensure that our screening tool is capable of accurately predicting T:R ratios in patients suffering from heart conditions which make them likely candidates for S-ICD implantation, we require ECG data from those patient groups for the training and evaluation of our models. To this end, we have collected ECG data from a range of patient groups which potential S-ICD implantation candidates are likely to belong to. Further details of this data set are given in Section 4.3.1. This new data set, collected for the purpose of this project and related analysis, enables us to train models with a high degree of robustness across a range of different patient groups.

In the course of developing this tool we apply a range of preprocessing techniques including numerous signal processing methods, creating Phase Space Reconstructions (PSR) of the ECG data and image augmentation. We then evaluate a wide range of different deep learning model architectures constructed from a combination of fully connected and convolutional layers. Having determined the best performing model architectures we assess a range of stochastic gradient descent based optimisation methods for model training, perform hyperparameter tuning, and create ensembles of the best performing models. Finally, these deep learning models capable of predicting T:R ratios are situated within an AI framework allowing clinicians to assess patient's S-ICD implantation eligibility.

In contrast with most of the literature, in which, with only a few exceptions (see, e.g., Babu et al. (2016)), Convolutional Neural Networks (CNNs) are used for image classification rather than for regression, we propose using 2D PSR images of the filtered ECG signals as input to CNNs with 2D filters. In particular, rather than extracting features from PSR images which are then used as model inputs, as typically done in previous works, we use the entire PSR image as input to our models. While CNNs have been used to analyse ECGs, this has been only done using data in which each lead corresponds to a single 1D-ECG signal. To the best of our knowledge, this is the first work in ECG analysis where a PSR matrix is generated for each lead that serves as input to a deep-learning model. Further separating this work form the literature is our focus on predicting T:R ratios. This is a very unique goal in ECG analysis, where various other ECG factors are identified and examined. The intuitive approach to calculating T:R ratios by detecting and measuring the amplitude of the R and T waves individually is vulnerable to the same TWOS problem that the algorithm used within the S-ICDs suffers from. Our novel methodology for predicting the T:R ratio does not share this vulnerability as it calculates T:R ratios from PSR images without explicitly determining the locations of the T or R waves in the original ECG signal.

To summarise the key contributions of Chapter 4,

1. We outline a preprocessing scheme by which ECG signals are filtered and transformed into PSR images, ready for use as inputs for our training problem.
2. We formulate the computer vision task of predicting T:R ratios from 2D PSR images as a regression problem.
3. We construct and optimise the performance of a deep-learning based regression model by assessing a range of architectures and stochastic gradient descent based training algorithms, tuning the hyperparameters of said algorithms and creating ensembles of the best performing models.
4. We demonstrate how such models can be incorporated into an AI tool allowing clinicians to conduct prolonged, automated screenings to assess patient's S-ICD implantation eligibility with a greater degree of detail and reliability.

The work in Chapter 4 has led to an initial paper detailing the workings of our AI screening tool:

Dunn, A. J., ElRefai, M. H., Roberts, P. R., Coniglio, S., Wiles, B. M., and Zemkoho, A. B. (2021). Deep learning methods for screening patients' S-ICD implantation eligibility. Artificial Intelligence in Medicine, *119:102139.*

We have developed the following two papers which use our tool to assess what the optimal choice of the T:R ratio cut-off used in the screening should be:

ElRefai, M., Abouelasaad, M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., Wiles, B. M., and Roberts, P. R. (2022b). Eligibility for subcutaneous implantable cardiac defibrillator utilising *artificial intelligence and deep learning methods for prolonged screening: where is the cut-off?* Europace, *24 (Supplement_1):euac053–447.*

ElRefai, M., Abouelasaad, M., Wiles, B. M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., and Roberts, P. R. (2022c). Deep learning-based insights on T:R ratio behaviour during prolonged screening for S-ICD eligibility. Journal of Interventional Cardiac Electrophysiology, doi.org/10.1007/s10840-022-01245-6.

A paper using our tool to evaluate the variation of the T:R ratio of patients with various heart conditions:

ElRefai, M., Abouelasaad, M., Conibear, I., Wiles, B. M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., and Roberts, P. R. (2022a). The use of artificial intelligence and deep learning methods in subcutaneous implantable cardioverter defibrillator screening to optimise selection in special patient populations. Europace, *24(Supplement_1):euac053–448.*

A further paper detailing the hyperparameter optimisation and comparison of optimisation algorithms for model training:

Dunn, A. J., ElRefai, M. H., Roberts, P. R., Coniglio, S., Wiles, B. M., and Zemkoho, A. B. (2022). Deep learning and hyperparameter optimization for assessing one's eligibility for a subcutaneous implantable cardioverter-defibrillator. Preprint available at https://optimization-online.org/?p=21066. *Under review in* Annals of Operations Research.

1.4.3 Bilevel hyperparameter optimisation for non-linear support vector machines

In Chapter 5 we address the problem of tuning the hyperparameters of a Support Vector Machine (SVM) model via cross validation using bilevel optimisation. There have been numerous attempts to formulate and solve the bilevel problem of selecting hyperparameters. As mentioned previously, it is crucial that the lower-level objective (the objective of the training problem) be a convex function. For this reason, the majority of work on solving the bilevel problem of selecting hyperparameters use models such as SVM (detailed in Section 5.1). Many of these examples tackle Support Vector Regression (SVR) (Bennett et al., 2008, 2006; Moore et al., 2009). In Chapter 5 we will consider Support Vector Classification (SVC). For examples of works focusing on bilevel hyperparameter tuning of SVCs via CV, we refer the reader to Kunapuli et al. (2008); Li et al. (2021). Hyperparameter tuning via bilevel optimisation has also been used for classification models other than SVMs such as Lp regression (Okuno et al., 2021; Nguyen et al., 2021). Some of these works show that, especially when using variants of the SVM model which have additional hyperparameters (as done in (Kunapuli et al., 2008)), bilevel methods are able to outperform the aforementioned sampling based methods such as grid-search. However, all of these works consider only the linear kernel SVM.

The Radial Basis Function (RBF) kernel has been shown to outperform other kernels and is generally recommended for practical applications as it affords SVM models the ability to capture more complex relationships between the variables of the training data. We speculate that the reason non-linear kernels, specifically the RBF kernel, have not been used in these previous works is due to the fact that it introduces the feature space embedding function $\phi : \mathbb{R}^d \to \mathbb{R}^\infty$ to the primal SVM training problem. In order to avoid this function, we formulate the dual form of the problem, which instead includes the *kernel function* $K(X_i, X_j) = \phi(X_i)^T\phi(X_j)$ which provides the relationship between two points X_i, X_j which have been embedded into the feature space. More detail on kernel SVMs can be found in Subsection 5.1. To the best of our knowledge this is the first work which formulates a bilevel problem for SVM hyperparameter selection using the dual form of the training problem and also the first work using a non-linear kernel SVM.

Typically, when grid-search, random-search or Bayesian optimisation are used to approximate a solution for this bilevel problem, the upper level problem involves maximising the accuracy of the model's predictions for the testing set. This choice of upper level objective function is non-smooth and hence would render any gradient based approach to solving the bilevel problem useless. We instead construct a smooth loss function for use in the upper level. Typically this is done using a variation of the lower level training objective. However, having switched to the dual problem, formulating this loss function becomes significantly more difficult. We propose a loss function for use in the upper level which can be calculated using the dual form parameters and formalise the bilevel optimisation problem of RBF kernel SVM hyperparameter optimisation via k-fold CV. We then perform the single level KKT reformulation and propose an algorithm for iteratively applying the Scholtes relaxation to find solutions to the original problem. The relaxed problem is solved using fmincon in MATLAB. The relaxation of the original problem is then reduced and the solution reached by the solver on the previous iteration is used as a starting point. We provide an analysis of the MPEC MFCQs to show that the solution found by this process of iteratively solving the problems with a lesser relaxation converges to a c-stationary point. We provide a number of examples of this algorithm in action on real data sets from the UCI repository which indicate that it is capable of finding local minima, and even outperforming grid-search.

In summary, the main contributions of Chapter 5 are:

1. We construct a smooth loss function, based on the primal SVM training objective function, which can be calculated using the parameters of the dual form of the training problem.

2. We define hyperparameter selection via k-fold CV as a bilevel problem using our proposed loss function in the upper level objective and the dual SVM training problem in the lower level.

3. We propose an algorithm which utilises the Scholtes relaxation to solve the bilevel problem and we provide the necessary analysis of the MPEC MFCQs to guarantee the convergence of this algorithm to a C-stationary point.

4. We provide a number of illustrative examples of our algorithm in use on real data sets which demonstrate its effectiveness, which we compare with that of grid-search.

The work in Chapter 5 has led to a paper, which is under preparation (note that this paper will be a significant extension of the work contained in this chapter):

Stefano Coniglio, Anthony J Dunn, Qingna Li, and Alain B Zemkoho, Bilevel hyperparameter optimisation for non-linear support vector machines. To be submitted to Mathematical Programming *by the end of* 2022.

Chapter 2

Basic machine learning preliminaries

2.1 Model evaluation

In this thesis we principally focus on supervised learning problems. In supervised learning problems, we have n data points $(X_i)_{i\in[n]}$, where $[n] = \{1, 2, ..., n\}$, each with d features, together with their corresponding labels $(y_i)_{i\in[n]}$ and we have a true, unknown labelling function $f : X_i \mapsto y_i$, for $i \in [n]$, which maps each data point to its corresponding label. For a classification problem $f(X_i)$ is discrete and for a regression problem, $f(X_i)$ is continuous. The supervised learning problem is to find a function h which best approximates f. For a chosen supervised learning algorithm A, the function $h : X_i \mapsto \mathbb{R}$ (also commonly referred to as a prediction rule, a predictor, a hypothesis, a model, or a classifier for classification problems) must belong to H_A, the family of possible prediction rules corresponding to A (e.g., for linear-kernel SVM, H_A would be the set of all linear discriminations). The optimal choice of h is one which minimises the supervised learning algorithms specific training *loss function* $\mathcal{L}_A : h(X), y \mapsto \mathbb{R}$ which has a large value the more significant the differences between $h(X)$ and y are. This process of optimally selecting $h \in H_A$ is referred to as *training* and is typically done on a subset of the n data point, label pairs $(X_i, y_i)_{i\in[n]}$. The subset of $\hat{n}$ data points $\widehat{X}$ and labels $\hat{y}$ used during this process is referred to as the *training set*.

2.1.1 Cross validation

To determine which supervised learning algorithm A results in a model h which is most capable of approximating the true labelling function we perform *model-selection*. Where f is a complicated function which may comprise some random components, it is, in fact, not ideal to find a model for which $h(\widehat{X}_i) = f(\widehat{X}_i) \quad \forall i \in [\hat{n}]$. While such a model predicts the training labels perfectly it is unlikely to have correctly found the process by which the true labeling function f performs the labeling but rather has just found a process which works well specifically for the training data and as a result will not generalise well to new unseen data. This phenomena is referred to as *over-fitting*.

To avoid over-fitting, we do not judge a model's performance by how well it predicts the labels for training data $\widehat{X}$, but rather, by how well it predicts labels for a set of, hereto unseen, data points $\bar{X}$ which we refer to as *validation data*, or alternatively, *testing data*. This process is referred to as *Cross Validation* (CV). The idea of reserving a subset of data on which to assess model performance dates back to the 1930s (Larson, 1931) and since then it has become the standard approach to assessing model performance. For an overview of cross validation approaches we refer the reader to Vanwinckelen and Blockeel (2012); Refaeilzadeh et al. (2009). A typical approach to CV is to reserve a portion of the entire set of data, label pairs (X, y) to form the *validation set* $(\bar{X}, \bar{y})$ where $\|\bar{X}\| = \bar{n}$. The remaining data and labels form the *training set* $(\widehat{X}, \hat{y})$ where $\|\widehat{X}\| = \hat{n}$. k-fold CV is a popular way to perform CV in which the total data X and labels y are partitioned into k equally sized *folds*. Then iteratively, one fold is selected for use as the validation set, the $k-1$ remaining folds form the training set and are used to train a model, the performance of which is assessed on the validation set. This is repeated such that k models are trained and each is evaluated on a different fold. This enables us to have k assessments of how well models resulting from a particular supervised learning algorithm can approximate the true labelling function f as apposed to the single assessment provided by a single round of CV. The process of model selection via k-fold CV can be summarised in the following algorithm:

Algorithm 1 k-fold CV for model selection

Result: The classification algorithm A_{best} which best approximates the true labeling function.

```
$\text{loss}_{\text{best}} = \text{loss}_{\text{initial}}$
**for** $A \in$ Algorithms **do**
    $\text{loss} = 0$
    **for** $i \in [k]$ **do**
        $h \in \arg\min_h \; \mathcal{L}_A\left(\widehat{X}^i, \hat{y}^i\right)$
            s.t. $h \in H_A$
        $\text{loss}_{\text{fold}} = \mathcal{L}\left(h(\bar{X}^i, \bar{y}^i)\right)$
        $\text{loss} \leftarrow \text{loss} + \frac{1}{k}\text{loss}_{\text{fold}}$
    **end for**
    **if** $\text{loss} < \text{loss}_{\text{best}}$ **then**
        $\text{loss}_{\text{best}} \leftarrow \text{loss}$
        $A_{\text{best}} \leftarrow A$
    **end if**
**end for**
**Return:** $A_{\text{best}}$
```

$\text{loss}_{\text{initial}}$ is an arbitrary large value, $\bar{X}^i$ and $\bar{y}^i$ are the $\bar{n}^i$ data points and labels belonging to the i^{th} fold, $\widehat{X}^i$ and $\hat{y}^i$ are the $\hat{n}^i$ data points and labels belonging to all folds but the i^{th} fold and $\mathcal{L}$ is a generic loss function for assessing a model's performance on a validation set. A wide range of loss functions for assessing model performance have been developed in the literature, each suitable for a specific problem class—for more details, we refer the reader to Vapnik (2000); Bishop (2006).

2.1.2 Performance metrics

Up to this point, we have considered both supervised learning tasks together. However, we use different metrics to evaluate model's performance for regression and classification tasks. For regression tasks we are able to consider the difference between the predicted label $f(\bar{X}_i)$ and the true label $\bar{y}_i$ for each data point label pair in the validation set $(\bar{X}_i, \bar{y}_i)$. We primarily focus on three accuracy metrics for assessing the performance of our regression models; i.e., the Mean Absolute Error (MAE), Mean Squared Error (MSE) and Root Mean Squared Error (RMSE):

$$\begin{aligned}
\text{MAE}(X,y) &= \frac{\sum_{i=1}^{n} |y_i - h(X_i)|}{n}, \\
\text{MSE}(X,y) &= \frac{\sum_{i=1}^{n} (y_i - h(X_i))^2}{n}, \\
\text{RMSE}(X,y) &= \sqrt{\frac{\sum_{i=1}^{n} (y_i - h(X_i))^2}{n}}, \\
\text{STD of errors}(X,y) &= \sqrt{\frac{\sum_{i=1}^{n} \left(y_i - h(X_i) - \frac{\sum_{j=1}^{n} y_j - h(X_j)}{n} \right)^2}{n-1}}.
\end{aligned} \tag{2.1}$$

These measures have historically been three of the most widely used accuracy metrics (Botchkarev, 2018). MAE is simply the average of the absolute values of the differences between each predicted label and its true label. This metric is easily interpretable as it represents how far one can expect your model's prediction to be from the true label of a single additional unseen data point. MSE is the average of the square of the values of the differences between each predicted label and its corresponding true label. This metric penalises large errors much more significantly than MAE thanks to squaring the errors, which is typically preferable. RMSE is the square root of the MSE. This metric is therefore very similar to MSE except for that by taking the square root of the sum of the squared errors we return to using the original units of the label. This makes RMSE more interpretable than MSE. So, while many models will be trained to minimise the MSE, the performance of the model will often then be assessed using RMSE. Finally, we consider the standard deviation of errors. While this metric is not as commonly used it is of interest to be able to observe the distribution of the errors.

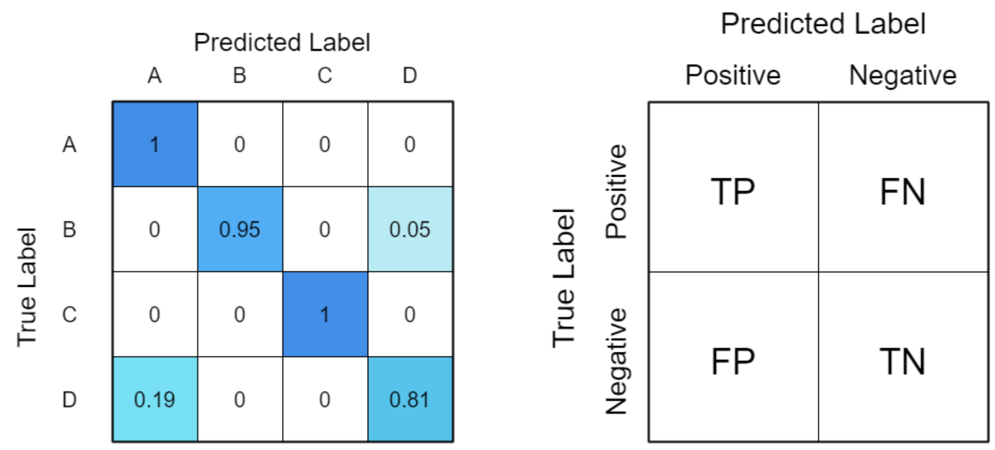

FIGURE 2.1: Left: A confusion matrix for a classification problem with four classes. Right: A matrix showing the number of data points with given true labels and predicted labels for a binary classification problem.

For classification we can not consider how close the predicted label was to the true label, as predictions are either for the correct class or the wrong class. For and overview of methods for assessing classification performance, see e.g., Davis and Goadrich (2006). One tool for analysing a model's performance is using a confusion matrix, shown in Figure 2.1. In this work we primarily focus on binary classification. Here, we refer to one class as *positive* and the other as *negative*. We therefore refer to the number of data points $\bar{X}_i$ with positive labels $\bar{y}_i$ whose predicted labels $h(\bar{X}_i)$ are also positive as the number of *true positives* (TP). Likewise, the number of data points with negative labels whose predicted labels are also negative is referred to as the number of *true negatives* (TN). The number of positive data points which are predicted to have negative labels is referred to as the number of *false negatives* (FN) and likewise the number of negative data points which are predicted to have positive labels is referred to as the number of *false positives* (FP).

The most straightforward way to assess the performance of a classification model on a binary classification task is to calculate its accuracy: the proportion of data points which are correctly labelled, as defined in (2.2). There are numerous other accuracy metrics which can be used to assess performance. For example, balanced accuracy, defined in (2.2), provides the average prediction accuracy for each class. If there were to be many more positive data points than negative (i.e. 10 times as many) then the standard accuracy metric would be heavily skewed toward accurate prediction for the positive data points. By using balanced accuracy, the model's prediction accuracy for each class is equally weighted regardless of how many data points belong to each. For this reason balanced accuracy is a fairer measure of accuracy when using *imbalanced* data sets, discussed in Subsection 3.1.

$$\begin{aligned}
\text{Accuracy} &= \frac{TP+TN}{TP+TN+FP+FN}, \\
\text{Sensitivity}(TPR) &= \frac{TP}{TP+FN}, \\
\text{Specificity}(TNR) &= \frac{TN}{TN+FP}, \\
FPR &= \frac{FP}{TN+FP}, \\
TPR &= \frac{TP}{TP+FN}, \\
\text{Balanced Accuracy} &= \frac{\text{Sensitivity}+\text{Specificity}}{2}.
\end{aligned} \tag{2.2}$$

Some models have a probabilistic output. Given a data point they output a predicted probability of that point having a positive label. If this probability exceeds 0.5 then the point's predicted label is positive. However, a common *post-processing* technique is to adjust this threshold to achieve a preferable True Positive Rate (TPR) and False Positive Rate (FPR), as defined in (2.2). Decreasing the threshold will increase both the TPR and FPR. We can compare all possible thresholds by plotting the TPR and FPR of each threshold as a point in a 2D plot. Such a plot is referred to as a Receiver Operating Characteristic (ROC) curve. ROC curve analysis has been used extensively in the literature for non-ML classification, see e.g., Hajian-Tilaki (2013); Park et al. (2004). In ML based classification, analysis of the ROC curve of a model's predictions has been used extensively in the context of imbalanced classification (Guo et al., 2008; Abd Elrahman and Abraham, 2013). An example of an ROC curve can be seen in Figure 2.2. The further the curve tucks into the top left corner the more thresholds exist which provide a high TPR and low FPR. The further the curve is from the top left corner the worse the model, as it shows that it is not possible to have a high TPR while having a low FPR. For this reason the area under the ROC curve can be a very good measure of how well a classification model can perform when using this post-processing step of adjusting the decision threshold.

However, a large area under a classifier's ROC curve indicates only that there are a range of thresholds which provide good TPR and FPR (Sokolova et al., 2006), and as such to actually determine the optimal value of these thresholds one should perform an additional round of CV. In other words, a high area under an ROC curve indicates that, were you to perform this post processing step, the resulting model would have good performance. The proper process should be to perform a nested k-fold CV where at each iteration, after reserving a validation set to evaluate the model on, an interior k-fold CV is done to train models, and find the optimal threshold for those models before the model (with its optimally adjusted threshold) is evaluated on the reserved validation set. For this reason, if one does not have a sufficient amount of data to

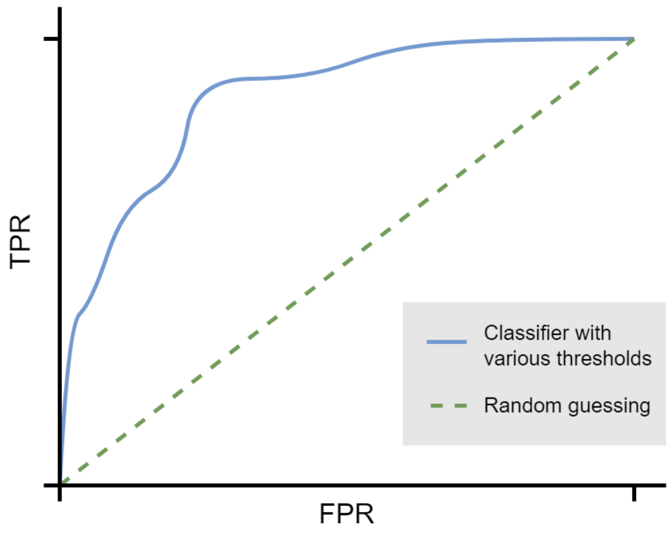

FIGURE 2.2: Graph of an ROC curve. Given a trained classification model, each decision threshold results in a given TPR and FPR, and so plots a single point. Plotting the TPR and FPR for each threshold results in the line shown in blue referred to as the model's ROC curve. The dotted green line shows the ROC curve which would be produced by a classifier which predicts a random value for the probability of each data point having a positive label.

perform the additional round of CV to choose the optimal threshold, balanced accuracy may be a preferable accuracy metric.

2.2 Supervised learning algorithms

In this subsection we give an in depth overview of neural networks as we will use them extensively in Chapter 4. We also provide a brief overview of a number of other supervised learning algorithms which we will evaluate in Chapter 3. For ease of notation, in this subsection we will assume that models are being trained on the entire data set (i.e. $(\widehat{X}_i, \hat{y}_i)_{i \in \hat{n}}$ is the same as $(X_i, y_i)_{i \in n}$), as in this section we do not require notation for the validation set. In this subsection, when we do mention the validation set, it can be considered as a separate data set to $(X_i, y_i)_{i \in n}$ rather than a subset of it.

2.2.1 Scaling

Scaling is an essential preprocessing step before the model training process. Many machine learning algorithms (e.g. KNN, SVM and Lp regression) take into consideration the distance between data points, i.e. $\sqrt{\sum_{l=1}^{d}(X_{il} - X_{jl})^2}$ where X_{il} is the l^{th} feature of the i^{th} data point in X. For these algorithms, were one feature to have a much larger variance than the other features, then the distance between any two data points would be heavily skewed by the feature with the large variance. This would result in features with comparatively small variances having a greatly reduced impact on the distances between points and therefore the predictions made by the model. Similarly, for models which train by calculating the gradient of a loss function with respect to the model parameters which relate to particular features (e.g. Neural Networks), this gradient would be skewed by the model parameters relating to features with a high variance. While decision tree based models such as random forest do not require the data to be scaled, doing so does not degrade their performance.

Normalisation, also known as min-max scaling (Patro and Sahu, 2015), is one method for scaling features whereby features are scaled to be bounded between 0 and 1. This is done using the following formula:

$$X_{il} \leftarrow \frac{X_{il} - \min(x_l)}{\max(x_l) - \min(x_l)} \qquad \forall l \in [d] \text{ and } \forall i \in [n],$$

where $x_l = \{X_{il} : \forall i \in [n]\}$. Standard scaling, also referred to as Z-score normalisation (Patro and Sahu, 2015), is another popular method where features are scaled to fit a standard normal distribution. This can be done using

$$X_{il} \leftarrow \frac{X_{il} - \mu_l}{\sigma_l} \qquad \forall l \in [d] \text{ and } \forall i \in [n],$$

where μ_l and σ_l are the mean and standard deviation of x_l respectively. Both of these methods achieve the goal of equalising the variance of each feature. Large outliers can cause normalisation to adjust all of the normal data values to be very close to each other. This can lead to the variance of the non-outlier data points to be very small. Standard scaling does not have this drawback.

2.2.2 Neural networks

In Chapter 4 we conduct an extensive set of experiments to optimise the performance of neural network models for use in screening patients for S-ICD implantation. In this section we outline the process by which neural networks are trained and detail some of the commonly used architectures.

The concept of artificial neural networks was introduced in 1943 (McCulloch and Pitts, 1943) when the process by which neurons function in the brain was modeled as a simple electrical circuit. In 1958, the notion of a perceptron was intoduced (Rosenblatt, 1958), a simple model which calculated a weighted sum of inputs which was then discretised using an activation function to generate a categorical output. The weights of this model could be learned via feeding the model successive inputs and calculating how the weights should be adjusted to attain the desired outputs. In 1960 neural networks were first applied to a real world problem: removing noise from phone lines (Widrow and Hoff, 1960). Training a neural network is an empirical loss minimisation problem; specifically, we aim to select the model parameters which minimise some loss function which is larger for models which are less accurately able to predict the labels of the data points in the training set. This is achieved by calculating the gradient of a loss function with respect to the model parameters using backpropagation, then updating the model parameters with a step, whose size is proportional to a hyperparameter called the learning rate, in the direction of steepest descent. In this section, we detail the various gradient descent based optimisation methods we employ to train our deep learning models.

Backward propagation

In neural networks, the process by which an input X is sequentially transformed as it passes through the layers of the network until the output $h(X)$ is finally produced is called *forward propagation*. This output is then input into a loss function. As is typical for regression problems, we use Mean Squared Error (MSE), defined in Equation (2.1). The gradient of the loss function with respect to each of the model parameters is then used to update the model parameters. The input is sequentially transformed in each layer and, as such, the total transformation $h(X)$ is a composition of the functions at each layer. For this reason, the chain rule is used to calculate the gradient of the loss function with respect to the model parameters in each layer. *Backward propagation* or *Back propagation* (Rumelhart et al., 1986) is an efficient algorithm for using the chain rule to compute these gradients:

$$g = \nabla_\theta L(h(X;\theta), y).$$

Early stopping

We employ *early stopping* (Prechelt, 1998) for regularisation in model training. It is often observed during training that both the training and validation errors initially decrease and, at some point, the validation error begins to increase while the training error continues to decrease. The error on the validation set serves as a proxy for the generalisation error which is used to determine when over-fitting has begun. Therefore, in order to achieve the lowest possible validation error, we reset the model parameters to their values before over-fitting began and the validation error started increasing. This means that we continue training our model until the validation error has not improved for a given number of epochs, referred to as the *patience* of the optimisation algorithm, instead of continuing to train until we reach a local minimum of the training error (Bengio et al., 2017).

Stochastic gradient descent

Stochastic Gradient Descent (SGD) is similar to typical batch gradient descent. However, rather than using all data points to compute the gradient of the loss function $\mathcal{L}$, an unbiased estimator of the gradient $\hat{g}$ (computed using a single randomly selected data point X_i and its associated label y_i) is used together with the learning rate ϵ to calculate the step direction and the step size and, thereby, update the model parameters θ. At each step, a different data point X_i and its associated label y_i are chosen. The formula by which $\hat{g}$ and θ are updated is

$$\begin{aligned} \hat{g} &:= \nabla_\theta L(f(X_i;\theta), y_i), \\ \theta &:= \theta - \epsilon \hat{g}. \end{aligned} \tag{2.3}$$

In practice, we in fact use minibatch SGD. Wherein, the motivation behind SGD that the gradient on a single data point is an unbiased estimator for the gradient on all data points is extended to: the average gradient on a minibatch of m randomly sampled data points is also an unbiased estimator for the gradient on all data points. The formula by which $\hat{g}$ updated using minibatch SGD is

$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(f(\check{X}_i;\theta), \check{y}_i),$$

where $\check{X}$ is the set of m data points in the minibatch with corrosponding labels $\check{y}$. The model parameters θ are then updated as before in Equation (2.3).

Both the learning rate ϵ and the batch size m are hyperparameters of the minibatch SGD algorithm. The advantage of the minibatch SGD is that the batch size m can be tuned such that minibatch SGD with smaller values of m enjoys the benefits of lower computational costs than batch gradient descent, thanks to not calculating the gradient on every data point at each step, and higher values of m leading to a reduction in the fluctuations in the value of the estimates of the gradient of the loss function observed while training using SGD.

Deep learning models are typically trained for a number of epochs. An epoch is comprised of a number of descent steps such that the gradient is calculated on each data point in the training set at least once. For example, in batch gradient descent there is one descent step per epoch and, in SGD with a batch size of 1, there are n descent steps, where n is the number of data points in the training set. When using minibatch SGD, the number of descent steps in a single epoch is roughly equal to n/m, where m is the batch size. Therefore, when training a model with a small batch size for a given number of epochs, one would expect the model parameters to be updated many more times than they would if one were to use a larger batch size.

SGD with Nesterov momentum

In cases where the gradient is small, SGD can be slow. Momentum is designed to leverage the fact that, when the gradient is small but consistent across multiple descent steps, we can afford to take larger steps in its direction, thus accelerating learning (Bishop et al., 1995). When implementing the classical momentum method (Polyak, 1964), at each descent step, as before, the new step is calculated by multiplying the gradient of the loss function by the learning rate. Now, however, an exponentially decaying history of the previous descent steps is added to new descent step. We refer to this summation as the velocity v. The formula for updating the model parameters at a single step of minibatch SGD is given by

$$\begin{aligned} v &\leftarrow \alpha v - \epsilon \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(f(\breve{X}_i;\theta), \breve{y}_i), \\ \theta &\leftarrow \theta + v, \end{aligned} \tag{2.4}$$

where α is the momentum constant, (an additional hyperparameter for determining how strong the contribution made by previous descent steps should be relative to the new velocity).

Nesterov momentum is a variant on the standard momentum algorithm whereby, rather than calculating the gradient using the parameters θ, the gradient is calculated using the parameters after the momentum has been applied, i.e., $\theta + \alpha v$ (Sutskever et al., 2013). This is often interpreted as estimating the descent step using only the momentum and then correcting that estimate by calculating the gradient at the location of the estimated parameters. The process for updating the velocity in a single gradient descent step of minibatch SGD with Nesterov Momentum is given by

$$v \leftarrow \alpha v - \epsilon \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(f(\breve{X}_i;\theta + \alpha v), \breve{y}_i).$$

The model parameters θ are then updated as before using Equation (2.4).

AdaGrad

Selecting an appropriate learning rate is an important yet difficult task. Typically, the loss function is much more sensitive to a subset of the model parameters than the rest. The descent step (the product of the learning rate and gradient) will be largest in the more sensitive directions. Consequently, progress in the less sensitive directions of the parameter space will be slow (Reed and MarksII, 1999). Momentum can mitigate these issues by muting the oscillations in the more sensitive directions while amplifying the consistent movement in the sensitive ones. However, momentum introduces an additional hyperparameter α, which can be just as difficult to select as the learning rate. Another approach to mitigating these issues would be to use an adaptive learning rate where the learning rate is scaled for each individual model parameter.

AdaGrad is one such adaptive learning rate algorithm where the learning rates corresponding to each model parameter are scaled by the inverse of the sum of the squared partial derivatives over all training iterations (Duchi et al., 2011). The process for updating the model parameters in a single descent step in AdaGrad is given by

$$\begin{aligned}
\hat{g} &= \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(f(\breve{X}_i;\theta),\breve{y}_i), \\
r &\leftarrow r + \hat{g}^2 && \text{(square is applied element-wise)}, \\
\Delta\theta &= -\frac{\epsilon}{\delta + \sqrt{r}}\hat{g} && \left(\frac{1}{\sqrt{r}} \text{ is applied element-wise}\right), \\
\theta &\leftarrow \theta + \Delta\theta,
\end{aligned} \tag{2.5}$$

where δ is a small constant included to avoid dividing by zero. In practice, this means that the learning rates corresponding to parameters with consistently large partial derivatives diminish rapidly, while those corresponding to parameters with small partial derivatives increase.

RMSprop

RMSprop is an adaptation of AdaGrad to converge more effectively in non-convex optimisation problems such as the regression problem of training multi-layer neural networks. In AdaGrad, the learning rate diminishes proportionally to the inverse of its entire history of squared gradients. This may cause the learning rate to diminish before reaching a convex bowl leading to a poor solution. RMSprop diminishes the learning rate proportionally to the inverse of an exponentially decaying window of squared gradients, thus allowing the learning rate to only diminish significantly upon settling in a convex region. The process for updating the model parameters in a single descent step in RMSprop is the same as that of Adagrad (shown in Equation (2.5)), except for r, which is instead updated as

$$r \leftarrow \rho r + (1-\rho)\hat{g}^2 \quad \text{(square is applied element-wise)},$$

where ρ is a hyperparameter determining the size of the exponentially decaying window.

ADAM

The final adaptive learning rate algorithm we consider is ADAM (Kingma and Ba, 2014). In ADAM, an exponentially decaying history is kept of both the first and second order moments of the gradient. These histories are then corrected to account for bias in their initialisation before being used to calculate the parameter update. The process for calculating the parameter updates $\Delta\theta$ at each descent step is as follows:

$$\begin{aligned} s &\leftarrow \rho_1 r + (1-\rho_1)\hat{g}, & \hat{s} &\leftarrow \frac{s}{1-{p_1}^t}, \\ r &\leftarrow \rho_2 r + (1-\rho_2)\hat{g}\odot\hat{g}, & \hat{r} &\leftarrow \frac{r}{1-{p_2}^t}, \\ \Delta\theta &= -\epsilon\frac{\hat{s}}{\delta+\sqrt{\hat{r}}}, \end{aligned}$$

where $\hat{g}$ is the estimate of the gradient as calculated in Equation (2.5), and t is the number of descent steps that the algorithm has been running for. The model parameters θ are then updated in the same was as in Equation (2.5). ADAM can be thought of as similar to RMSprop. However, the inclusion of the decaying history of first order moments of the gradient $\hat{s}$ is similar to using momentum. The other notable difference is the bias correction, which leads to lower bias in the early stages of training. ADAM introduces two hyperparameters, ρ_1 and ρ_2. However, ADAM is considered to be robust to the choice of these hyperparameters and as such, only the global learning rate ϵ requires tuning (Bengio et al., 2017).

Initialisation

SGD and its variants reach solutions in the limit by building a sequence of iterates and, as such, require a starting point. In the context of neural networks, this starting point is the initialisation of the model parameters θ, i.e., the initialisation of the layer weights w and biases b. The principal requirement when initialising the model parameters is to *break symmetry* (Goodfellow et al., 2016). If multiple nodes connected to the same input had the same weights and biases, then the gradient of the loss function with respect to the weights corresponding to those nodes would be the same and, thus, the updates to these parameters would be the same throughout training, leading to these nodes being duplicates of each other. To avoid this, the weights are initialised randomly. The distribution used for this must be selected carefully, as large initial weights break symmetry well leading to a low number of redundant nodes, but, should they be too large, they may cause exploding values.

A widely used approach is the Glorot normalised initialisation method (Glorot and Bengio, 2010). In their work, the authors suggested that training becomes difficult when layer-to-layer transformations do not allow the gradients to “flow well” (i.e., the elements of the Jacobian associated with each layer are far from 1). They suggest that the variance of the weights should be seen as in Equation (2.6) (see further), as this allows the gradients to 'flow well' in both backward and forward propagation. The authors then suggest that the weights in each layer be distributed uniformly according to Equation (2.7). We instead use another popular initialisation strategy of distributing the weights normally as in Equation (2.8)[1]. This distribution's variance is also given in Equation (2.6) and, as such, we retain the nice properties of the Glorot normalised initialisation method. The equations are

$$\mathrm{Var}\left[W\right] = \frac{2}{n_{\text{in}} + n_{\text{out}}}, \tag{2.6}$$

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right], \tag{2.7}$$

$$W \sim N\left[0, \frac{2}{n_{\text{in}} + n_{\text{out}}}\right], \tag{2.8}$$

where W is the matrix of weights in a given layer, n_{in} is the number of nodes in the previous (input) layer and n_{out} is the number of nodes in this layers output. We define the operator $\sim$ such that $X \sim F$ implies that the random variable X follows probability distribution F.

As is standard, we initialise the biases b with value 0 as, by randomly distributing the weights, we have already ensured that symmetry is broken.

Neural network layers

In this subsection, we discuss the architecture of the models we will use in Chapter 4. There are 5 types of layers that are used within the architectures of our neural networks. The first and most fundamental of these layers is the fully connected, or *dense*, layer. A dense layer consists of a number of neurons, each of which is connected to all neurons in the previous layer. Each neuron outputs a weighted and shifted sum z_i of its input x, a vector containing the outputs from the previous layer. z_i is given by

$$z_i = b_i + x \odot w_i,$$

where b is a vector containing a bias for each neuron in the layer and w_i is a vector of weights for the i^{th} neuron containing a weight associated to each element in the input. The Hadamard product $\odot$, also referred to as element-wise product, is used in this calculation. This type of layer can only take one dimensional inputs and so any multi-dimensional inputs (such as the PSR matrices we use as input in Chapter 4) must be flattened before being passed to a dense layer. Models comprised of dense layers are referred to as Multi-Layer Perceptrons (MLPs).

Next, we consider convolutional layers. Convolutional Neural Networks (CNNs), neural networks comprised of convolutional layers, are commonly used for image analysis and exploit the multi-dimensional nature of their inputs. In a convolutional layer

[1]This is the default initialisation scheme in the popular deep learning Python package Keras.

3-dimensional filters, also referred to as *kernels*, are applied to the input at regular intervals, determined by the layer's *stride*, to transform it into a 3-dimensional output comprised of a number of 2-dimensional feature maps. The output for a neuron in row i and column j of the feature map k in a convolutional layer with 3-dimensional input x is given by

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1}\sum_{v=0}^{f_w-1}\sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k}, \qquad \text{where } \begin{cases} i' = i \times s_h + u, \\ j' = j \times s_w + v, \end{cases}$$

where s_h and s_w are the vertical and horizontal strides respectively, f_h, f_w and $f_{n'}$ are the height, width and depth of this layer's filters, where $f_{n'}$ is equal to the number of feature maps in the previous layer, b_k is the bias applied to the feature map k and w_k is the matrix of weights defining the 3-dimensional filter used to generate the k^{th} feature map and hence $w_{u,v,k',k}$ is the weight in row u and column v of the two dimensional slice of the filter which connects to feature map k' of the input. For a detailed introduction to CNNs we refer the reader to Wu (2017).

Batch normalisation, or *batchnorm*, is a tool for reparameterising neural networks to significantly speed up training. The gradient of the loss function with respect to each model parameter is used to update the model parameters. For a given parameter in a layer, these updates assume that the parameters in other layers do not change and therefore neither does the distribution of the inputs to this layer across each minibatch (for further details on minibatch gradient descent see Section 2.3). However, all of the model parameters are updated simultaneously meaning that the distribution of the inputs to this layer across each mini-batch which the model parameter was adjusted to fit better is no longer being output from the previous layer. Batch normalisation can be applied to any layer and normalises the distribution across the minibatch for each output in a layer, ensuring that throughout training the inputs to the following layer always have the same distribution (Garbin et al., 2020).

Batch normalisation is applied within its own layer. The minibatch outputs $z = \{z^{(1)}, \ldots, z^{(m)}\}$ of a batch normalisation layer with minibatch inputs $x = \{x^{(1)}, \ldots, x^{(m)}\}$ are given by

$$\begin{aligned}
\mu^B &= \frac{1}{m}\sum_{i=1}^{m} x^{(i)}, \\
\sigma_B \odot \sigma_B &= \frac{1}{m}\sum_{i=1}^{m} \left(x^{(i)} - \mu^B\right) \odot \left(x^{(i)} - \mu^B\right), \\
\hat{x}^{(i)} &= \left(x^{(i)} - \mu^B\right) \oslash \left(\sigma^B + \epsilon\right), \\
z^{(i)} &= \gamma \odot \hat{x}^{(i)} + \beta,
\end{aligned}$$

where m is the size of the minibatch B and $x^{(i)}$ is the matrix containing the output from the previous layer for the i^{th} data point in the minibatch. The outputs $z^{(i)}$, the minibatch mean μ^B and standard deviation σ^B, and the scaling and shifting parameters γ and β

are all matrices with the same shape as the inputs $x^{(i)}$. The only learnable prameters in this layer are γ and β dimensional vectors which are learnable model parameters. The Hadamard product $\odot$ and Hadamard division $\oslash$, also referred to as element-wise product and division respectively, are used in these calculations. Batch normalisation also reduces generalisation error, allowing us to forgo the use of dropout (Goodfellow et al., 2016) and has been shown to be superior to dropout for regularisation in CNNs (Garbin et al., 2020).

The activation function is often applied to the output of each neuron within each layer. However, we perform batch normalisation before applying the activation function as proposed in the original paper on batch normalisation (Ioffe and Szegedy, 2015). For this reason we apply the activation function within its own layer after batch normalisation has been applied to the output of each convolutional or dense layer. We use the generally recommended rectified linear unit or ReLU (Nair and Hinton, 2010; Glorot et al., 2011) given by

$$R(z) = \max(0, z).$$

This function is applied elementwise to the previous layer's output.

Finally addition skip connections, as utilised in ResNet (He et al., 2016), are used within our deeper CNNs. The gradient by which parameters of a neural network are updated is calculated using the chain rule and, as such, as we consider layers which are further back in the network, more gradients are included in the product defining the gradient of the loss function with respect to that layers parameters. As it is often the case that these gradients are less than one, the gradients relating to parameters in early layers become smaller the deeper the network becomes, posing a serious problem in deep networks. In an addition skip connection the output of a layer is added to the input of a layer located significantly deeper into the network. The inclusion of addition skip connections allows better flow of gradients from the first layer to the last allowing the early layer's parameters to have larger associated gradients, allowing them to train more quickly. Models with these skip connections have a smoother loss surface (Balduzzi et al., 2017; Li et al., 2017) which makes optimising the model parameters easier and thus speeds up training.

2.2.3 Other learning algorithms

In Chapter 3 we provide a broad comparison of a range of classification algorithms. In this Subsection we provide a brief overview of the working of these algorithms.

K nearest neighbours

The K Nearest Neighbors (KNN) algorithm is one of the most simple supervised learning algorithms. It was introduced in 1951 (Fix and Hodges, 1951) and has been a staple model for both classification and regression. For a more in-depth overview of this algorithm we refer the reader to Kramer (2013). Unlike with other methods where model parameters must be tuned to minimise a loss function of the model's predictions on the training set, the k nearest neighbours algorithm has no model parameters but rather stores a record of training data points and labels, and has only a single hyperparameter k which must be determined before training them model. Given a new, unlabelled, data point the model locates the K nearest training points. Then for regression, the predicted label (dependant variable) is given by averaging the labels of the K nearest training points. For classification, the class is assigned as the most prevalent class among its K-nearest neighbors.

Logistic regression

Logistic regression is a classification model defined by the composite of a standard linear regression function and a particular sigmoid function referred to as a logistic function:

$$f(X_i) = \text{sigmoid}(\alpha X_i + \beta) = \frac{1}{1 + e^{-(\alpha X_i + \beta)}} \qquad \forall i \in [n],$$

where $\alpha \in \mathbb{R}^d$ and $\beta \in \mathbb{R}$ are the model parameters. This particular formulation requires the labels $y \in \{0, 1\}$ These model parameters are optimised during training such that a loss function comparing $f(X_i)$ (the predicted probability that X_i has a corresponding label of 1) with y_i for each $i \in [n]$. If one was to use MSE as this loss function then the problem of selecting α and β would be non convex (Kleinbaum et al., 2002). Instead the log loss function is used:

$$\text{LogLoss}(X, y, \alpha, \beta) = \frac{1}{n} \sum_{i=1}^{n} y_i \log(f(X_i)) + (1 - y_i) \log(1 - f(X_i)).$$

Decision trees

The decision tree classifier is in essence a set of if statements leading ultimately to a classification. This tree of checks (also referred to as (nodes)) is constructed and the conditions of each node is determined during training. In order to determine which variable should be checked at a given node, the *Gini index* (the average of the proportion of data points which miss-classified on each side of the proposed split) (Myles et al., 2004) for each variable, with the variable which results in the lowest error on each branch of the split being selected. Too deep of a tree can lead to over-fitting as a deep enough tree can correctly classify virtually every data point in the training set. There are numerous criteria which can be used to determine when the decision tree should stop branching. One is a maximum depth, a hyperparameter limiting the number of times the decision tree can branch. Another is a minimum number of samples for which a new split can be made, this hyperparameter stops *leaf nodes* (nodes at the deepest layer of the tree) which only assign a label to a few samples splitting further and over-fitting to those points. Small changes in the training data can lead to significantly different decisions rules found by the decision tree. For this reason, given an unfortunate draw when splitting data into training and validation sets, a single decision tree can have quite unreliable performance.

To overcome this problem we can instead create *ensembles* of decision trees called a *Random Forest* (RF) (Biau and Scornet, 2016), where each tree is trained on a slightly different training set. An ensemble model is simply a collection of models which, when given a data point to predict a label for, outputs the average of the predicted label of each model in the ensemble. To ensure that each decision tree is trained on a different data set bootstrap aggregated (*bagging*) is used, whereby, in order to generate a training set for each decision tree, the original training set is sampled with replacement to create a new set of data point, label pairs which is the same size as the original training set. In balanced random forest, a variation on random forest designed to work well on imbalanced data, the bagging utilises a sampling scheme which ensures that each newly generated training set is balanced.

Another method for creating more complex models from numerous decision trees is *boosting*, wherein decision trees are chained sequentially, with each subsequent tree being tasked with correcting the residuals (the errors) of the previous tree (Drucker and Cortes, 1995). Similarly to random forest, this method combines multiple weak learners to formulate a strong model. However, unlike with random forest, adding more decision trees to a boosting model can result in over-fitting to the training set. As a result, the number of decision trees used in a boosting model is an important hyperparameter.

Lp regression

Bilevel optimisation has only been used to tune the hyperparameters of a small number of algorithms. One of those algorithms is Lp regression, which we overview in this subsection. p refers to the power of the norm used to regularise the regression model. The Lp regression algorithm is an extension of the standard least squares regression method which includes a regularisation term to avoid over-fitting. The Lp regression model trains by selecting the coefficients β which minimise the following equation:

$$\sum_{i=1}^{n} (y_i - X_i\beta))^2 + \lambda \sum_{j=1}^{d} \|\beta_j\|^p.$$

λ and p are hyperparameters of the model. λ directly controls the importance of the penalty caused by the coefficients being non-zero, while p controls the (possibly non-linear) nature in which larger coefficients contribute towards this penalty. When p is even, there is of course no need to calculate the absolute values of the coefficients. When $p = 1$ this model is called *lasso regression* and when $p = 2$ this model is called *ridge regression*. For a more detailed overview of lasso and ridge regression as well as a comparison of the two with Lp regression models with $p < 1$ we refer the reader to Fu (1998).

2.3 Hyperparameter tuning

Supervised learning algorithms often depend on one or more *hyperparameters* whose tuning is crucial to obtain a high prediction accuracy (Probst et al., 2019). Hyperparameter selection is typically performed using k-fold CV. The goal is to find the hyperparameter combination λ belonging to the set of all possible hyperparameters Λ which leads to the k models which have the lowest possible error on the k validation sets.

Because this problem is very difficult to solve, there exists a range of techniques to approximate its solutions. These techniques typically revolve around sampling a hyperparameter combination, solving the k training problems, and evaluating the error on the k validation sets. This is repeated for numerous hyperparameter combinations. The scheme by which these hyperparameter combinations are selected is the principal difference between these techniques (Del Buono et al., 2020).

Among different options, *grid-search* is, arguably, the most widely applied technique. When performing grid-search the hyperparameter space Λ is discreteised into a grid of hyperparameter combinations G (typically the ranges of hyperparameter values defining these grids are logarithmic), for each hyperparameter combination the k training problems are solved resulting in k models which are then used to evaluare the error of each model on its validation set. The hyperparameter combination which leads to models with the best performance over all k rounds of CV is deemed to be the approximate solution to the hyperparameter optimisation problem. Algorithm 2 details the process of performing gridsearch via k-fold CV. As in Subsection 2.1.1, $\bar{X}^i$ and $\bar{y}^i$ are the $\bar{n}^i$ data points and labels belonging to the i^{th} fold, $\widehat{X}^i$ and $\hat{y}^i$ are the $\hat{n}^i$ data points and labels belonging to all folds but the i^{th}, A denotes a supervised learning algorithm, h is a model or decision rule and H_A is the set of all possible decision rules for A.

Algorithm 2 Grid-search for k-fold CV

Result: The hyperparameter combination λ_{best} for algorithm A which leads to the best CV performance.

```
  $\text{loss}_{\text{best}} = \text{loss}_{\text{initial}}$
  for $\lambda \in G$ do
      $\text{loss} = 0$
      for $i \in [k]$ do
          $h \in \arg\min_h \quad \mathcal{L}_A\left(\widehat{X}^i, \hat{y}^i, \lambda\right)$
                s.t. $\quad h \in H_A(\lambda)$
          $\text{loss}_{\text{fold}} = \mathcal{L}\left(h(\bar{X}^i, \bar{y}^i), \lambda\right)$
          $\text{loss} \leftarrow \text{loss} + \frac{1}{k}\text{loss}_{\text{fold}}$
      end for
      if $\text{loss} < \text{loss}_{\text{best}}$ then
          $\text{loss}_{\text{best}} \leftarrow \text{loss}$
          $\lambda_{\text{best}} \leftarrow \lambda$
      end if
  end for
  Return: $\lambda_{\text{best}}$
```

Random-search is another method for hyperparameter selection whereby, rather than sampling hyperparameter combinations from a uniform grid, hyperparameter combinations are randomly sampled (this sampling will typically also be done on a logarithmic scale). Bayesian optimisation is another sampling based method, which aims to construct a map of the value of the average validation error with respect to the hyperparameters by iteratively sampling hyperparameter combinations which provide the most information to fill in gaps in this map. For details on these techniques, we refer the reader to the following overview of hyperparameter tuning techniques Del Buono et al. (2020).

Chapter 3

Imbalanced data classification for infrequent adverse event prediction in low-carbon energy production

In this chapter we address the problem of Adverse Event prediction in low carbon energy production introduced in Subsection 1.1. We develop a framework to model any problem involving the prediction of infrequently occurring adverse events from time-series data as an imbalanced classification problem, which we then solve using a range of machine learning techniques that we adapt in order to better leverage the unique features of this application. As part of the algorithmic process proposed in this chapter, we introduce *block sampling*, a novel approach for undersampling by removing problematic data points and retaining only the most useful data for use in model construction and evaluation. The use of this framework for the Anaerobic Digestion (AD) application leads to models capable of discriminating between normal operating conditions and operating conditions which are indicative of an impending foaming event with very high accuracy. This allows us to provide advanced warning for all foaming events in our data set. Also for the Nuclear Power Production (NPP) application, we are able to construct models capable of providing advanced warning for the majority of the condenser tube leaks in our data set.

In Section 3.1 we provide a brief recap of the problem and provide a review of some related works. In Section 3.2 we outline the process by which we model the prediction of adverse events from time series data while tackling the class imbalanced problem, before proposing a framework for creating models capable of capturing the hidden phenomena which lead to adverse events and, as such, are able to provide advanced warning of the occurrence of adverse events sufficiently ahead of time as to allow for effective remedial actions to be taken. This framework leverages various machine learning techniques for solving the resulting learning problem and incorporates a set of appropriate modifications to common methods designed for our specific context. In Section 3.3, we assess the effectiveness of our framework in the two applications under consideration via a set of extensive computational experiments in which 11 popular classification algorithms, imbalanced classification techniques and hyperparmeter tuning methods are evaluated and compared. We also make final recommendations for which models and enhancement techniques are advisable for use in each of the two applications based on their performance in our experiments. The chapter is concluded with Section 3.4, where some final remarks are provided. This work has led to the following publication:

Coniglio, S., Dunn, A. J., and Zemkoho, A. B. (2020). Infrequent adverse event prediction in low carbon energy production using machine learning. Preprint available at arXiv:2001.06916. *Under review in* Machine Learning.

3.1 Problem description and related works

For the Anaerobic Digestion (AD) application, our aim is to create a tool capable of reliably predicting foaming with enough warning for the plant operators to administer the anti-foaming agent and subdue the foaming before it can damage the digester. While foaming in AD is a well-researched area, most of the available works (see, e.g., Kanu et al. (2015, 2018)) analyse the phenomenon from a chemical or engineering perspective. To the best of our knowledge, only a few attempts have been made at modelling it using machine learning methods; see, e.g., Dalmau et al. (2010a); Fernandes (2014). More broadly, most data driven approaches rely on the design of a Knowledge-Based System (KBS) and, thus, require in-depth knowledge of the specific digester in question; see, e.g., Dalmau et al. (2010b); Gaida et al. (2012); Kanu et al. (2018). Often, they also rely on a deep knowledge of the chemical composition of the feed stock, also known as *feed sludge characteristics*, as well as on time-series sensor readings of the conditions within the digester, typically referred to as *operating characteristics*.

While it has been shown that relying on feed sludge characteristics can be more effective than simply considering operating characteristics, monitoring them is not feasible in many industrial applications. More precisely, in the context of foam-formation in AD, machine learning models have only been used to a limited extent such as for state estimation using neural networks to predict methane production, e.g., (Gaida et al., 2012; Fernandes, 2014), and for gaining insights into which variables could be the best indicators of foaming (Dalmau et al., 2010a). Crucially, such approaches are often unable to predict foaming before it occurs, as typically they provide a warning only when the foaming is already in an advanced stage and it is too late to take any mitigating actions (Dalmau et al., 2010b).

On the contrary, the methods we propose in Chapter 3 allow for predicting foaming within a user-specified warning window from operating characteristics alone. Regular and frequent injections of anti-foaming agent into the digester appears to be the most popular approach to prevent foam from building up. As we will illustrate, the models we propose are able to provide advanced warning of the occurrence of foaming. This advanced warning can enable plant operators to inject anti-foaming only when foaming is likely to occur, resulting in a significant reduction in the amount of anti-foaming agent required to avoid foaming compared to routinely injecting anti-foaming agent, thus leading to a considerable cost reduction. More details on foam formation in AD can be found in Dalmau et al. (2010b); Yang et al. (2021); Gaida et al. (2012); Kanu et al. (2018); Ganidi et al. (2009) To the best of our knowledge, there are no works attempting to predict condenser tube leaks. However, for some background on tube leakage in NPP we refer the reader to Daniels (2010); Coit (1980); Cattant et al. (2008); Müller-Steinhagen (1999); Lawrence et al. (1977).

In this work, we model the problem of predicting infrequent adverse events in the context of our two applications as an imbalanced classification problem. The class imbalance problem was discovered close to three decades ago and has been at the center of attentions since then, with various practical mitigation strategies to improve the performance of existing prediction algorithms; see, e.g., Japkowicz and Stephen (2002); Akbani et al. (2004a); Chawla et al. (2004); Sáez et al. (2016) for some related papers and surveys. However, to the best of our knowledge, this is the first work studying this issue in the context of foaming and condenser tube leakage in AD and NPP, respectively. Unlike in previous studies, we use only operating conditions which are recorded as part of the standard monitoring activities of a plant, and, as such, no additional data is required for our analysis.

Class imbalance occurs when a data set has many more data points belonging to one class (the *majority class*) than the others. In the case of binary classification the class containing fewer data points is referred to as the *minority class*. This can cause issues during training as models tend to over-fit to the majority class. This leads to models which, given an unlabelled data point, are predisposed to predicting that it belongs to the majority class and conservative in predicting that it belongs to the minority class.

There are numerous strategies used to mitigate the impact of class imbalance; for a recent survey, see, e.g., Leevy et al. (2018); Zhu et al. (2018); Japkowicz (2000). These strategies generally fall into three categories: data pre-processing, special-purpose learning methods and post-processing (Branco et al., 2016).

The most commonly used pre-processing technique to address class imbalance is sampling (Estabrooks et al., 2004), whereby the data is resampled such that the new data set has an *imbalance ratio* (the ratio of the number of samples belonging to the minority class and majority class, respectively) which is closer to 1. The most popular of these sampling methods are: *random undersampling* (RUS), where the cardinality of the majority class is reduced by randomly sampling (with replacement) a subset of the majority class; *random oversampling* (ROS), where the cardinality of the minority class is increased by randomly sampling from it with replacement; and synthetic oversampling, where new, artificial, minority class samples are created. Optimal results may be obtained through a combination of undersampling and oversampling methods, see, Estabrooks et al. (2004). For an example of combining sampling techniques for classifying time-series data, we refer the reader to Moniz et al. (2016), in which undersampling is used in concert with synthetic oversampling.

Special-purpose learning methods refers to a range or classification algorithms either shown to perform well on imbalanced data sets or which are specifically designed for this problem class. Classification algorithms which utilise bagging and boosting have been shown to provide good results in imbalanced domains (Singh and Purohit, 2015). In Section 3.3, we evaluate the performance of numerous popular classification algorithms, including: random forest classifiers, which utilise bagging; and adaptive boosting and gradient boosting classifiers, which utilise boosting methods. There also exists variation of common algorithms to better suit imbalanced classification problems. For example balanced random forest is a variation of random forest which is designed to perform better for imbalanced classification problems. For an example of how existing algorithms can be altered to better suit the imbalanced domain, we refer the reader to Akbani et al. (2004b). Generative models such as the Gaussian naive Bayes classification algorithm (which we evaluate in Section 3.3) also show a great deal of robustness to class imbalance, as the distribution for each class is calculated independently and, therefore, they are not biased to over predicting the majority class.

A popular form of post processing is *thresholding*. Where a classifier gives a probabilistic output, the threshold at which a label is assigned can be adjusted such that the classifier more readily assigns data points to the minority class. As discussed in Subsection 2.1.2, analysis of a model's ROC curve can enable the optimal selection of this threshold. Therefore, when possible, an additional validation set should be partitioned from the training set with which to assess a range of thresholds, the best of which can then be used to evaluate the model's new adjusted performance on the original validation set.

3.2 Modeling and solving the infrequent adverse event prediction problem

3.2.1 Machine learning problem set-up

Intuitively, adverse event prediction from time series data is an anomaly detection problem. In this subsection, we suggest a modeling framework to label the time series operating condition data, hence constructing a supervised anomaly detection problem (Omar et al., 2013; Laptev et al., 2015). Anomaly detection models are typically trained either offline (in *batches*), whereby historical data sets are used to periodically update the model, or online, where models are continuously updated as live data is fed into the model (Srikanth et al., 2020). Due to the small number of adverse events in our data sets, it is impractical to train our models online as, by the time the model has seen enough training data to give reasonable predictions, we would only have one or two adverse events left to evaluate the model's performance on. For this reason, we batch train our models. Treating adverse events in our applications each as a single anomaly would lead to a trivial data set, containing only 5 anomalies for the AD application and 7 for the NPP application. Further details on the datasets are provided in Section 3.3.1. Training a model using such a small number of data points in the minority class would be infeasible. For this reason, we introduce a modeling framework which derives numerous data points of anomalous operating conditions relating to each adverse event. To the best of our knowledge, the following modeling framework has not been applied to any applications of supervised anomaly detection from time series data.

Given a time horizon, let $(X_i,\, y_i)_{i\in[n]}$ be a time series where each data point $X_i \in \mathbb{R}^d$ consists of an observation of d features at time i and y_i is a binary label denoting whether the adverse event occurs at time i or not. More formally, the pair $(X_i, y_i)_{i\in[n]}$ is such that

$$X_i \in \mathbb{R}^d \quad \text{and} \quad y_i = \begin{cases} 1 & \text{if the event occurs at time } i, \\ 0 & \text{otherwise.} \end{cases}$$

We adopt a common pre-processing technique often used in time-series forecasting and referred to as *leading* or *lag inclusion*, where we augment each data point X_i by creating the vector $\tilde{X}_i \in \mathbb{R}^{d(\tau+1)}$ defined as follows:

$$\tilde{X}_i := (X_{i-\tau}; \ldots;\; X_i),$$

where τ is the number of lags that we wish to include. We refer to each $\tilde{X}_i$ as a *pattern*.

Let ω be the maximum amount of time before an event at which the system begins to show signs that an event is imminent. To be able to capture an adverse event at most ω steps before its occurrence, we introduce a new set of labels $\tilde{y}_i \in \{0,1\}$, $i \in [n]$, defined in such a way that, for a given $\omega \in \mathbb{N}$, it holds that

$$\tilde{y}_i = 1 \quad \iff \quad \exists i' \in \{i, \ldots, i+\omega-1\} : \; y_{i'} = 1.$$

We refer to such labels as *warning labels*. According to this definition, $\tilde{y}_i$ is equal to 1 if and only if an adverse event occurs in the time window starting at time i and ending at time $i+\omega-1$, whereas it is equal to 0 otherwise. Relabeling the data in this manner is crucial for constructing models capable of predicting adverse events before their occurrence. By generating a number of additional data points related to each adverse event, this method also reduces the class imbalance present in our data sets. To the best of our knowledge, this relabeling has not been used previously in the literature. The problem of training a classifier to determine whether an event will occur in the next ω steps can now be formalised as follows

Problem 1. Given $\left(\tilde{X}_i, \tilde{y}_i\right)_{i\in[n]}$, find h such that:

$$\begin{aligned} h \in \arg\min_{h} \quad & \mathcal{L}_A\left(\tilde{X}, \tilde{y}\right) \\ \text{s.t.} \quad & h \in H_A, \end{aligned}$$

where H_A is the family of possible prediction rules which can be output from classification algorithm A and $\mathcal{L}_A$ is the loss function (which takes, as input, pairs of corresponding predictions $h(\tilde{X}_i)$ and true labels $\tilde{y}_i$) used by algorithm A to determine the optimal prediction rule h (Shalev-Shwartz and Ben-David, 2014).

Solving this problem for a given classifier is just a step of the approach that we propose in the next section. In our method, Problem 1 can be solved by adopting, virtually, any classifier (we will consider 11 such classifiers in Section 3.3). In the next subsection, we outline the main challenges that one faces when tackling this problem for our specific applications with off-the-shelf methods, and the techniques that we propose to avoid them.

Note that in order to predict the adverse event before it actually occurs with enough warning such that maintenance can be carried out to prevent or minimise the damage caused by it, we introduce m, the minimum length of the warning window that is required for the application at hand. The value of m should be large enough to allow sufficient time for the required maintenance to be carried out in order to avert the impending adverse event. ω, on the other hand, is a hyperparameter for Problem 1, the optimal choice of which is dependant on each specific system and application. The careful selection of ω is crucial, as this number must be significantly larger than m to allow time for maintenance, yet too large an ω makes the assumption that the behavior of the system long before the event is indicative of an event's imminent occurrence which may not be true (very large values of ω would lead to $\tilde{y}_i = 1$ for almost all $i \in [n]$, rendering the prediction, *de facto*, useless).

Of course, there is no value in predicting events either within m steps of their occurrence or when they are already occurring as, at this point, it would be too late for the event to be prevented. For this reason, we are not interested in evaluating the performance of our models when the original label y_i equals 1, or fewer than m steps before y_i equals 1.

In both of our applications, the maintenance required to prevent an adverse event from occurring can be completed almost immediately. Anti-foaming agent can be administered, which very quickly subdues severe foaming and the load on a condenser's corresponding turbines can be immediately reduced, allowing time for repairs to be carried out without the risk of a large leak developing. For this reason, we can consider m to be 0, meaning that we need only disregard our model's performance on the patterns whose original label y_i equals 1. In Subsection 3.2.3, we detail how we handle such patterns in our experiments.

In our experiments we adopt the customary method of constructing a prediction rule by fitting a classifier to a training set and then testing said rule on a reserved, unseen testing set. As is typical when assessing classification accuracy, we repeat our training and testing via k-fold CV (detailed in Subsection 2.1.1). To evaluate each classifier's ability to predict warning labels for a given testing set we use an accuracy metric, such as balanced accuracy or the area under an ROC curve (each detailed in Subsection 2.1.2). We weigh the pros and cons of these metrics for use in this application in the following subsection. In Subsection 3.2.3, we introduce a novel method for sampling folds for use in CV which, as we show in Subsection 3.3.2, results in a more accurate prediction of adverse events than standard approaches. When using our proposed method, the number of folds we can sample from our data into is limited by the number of events we have. The number of adverse events in either of the data sets corresponding to our two applications is very low. Due to the nature of the learning tasks we tackle, a number of crucial issues are likely to arise when resorting to the classical method of k-fold CV for training and testing the classification algorithm we propose. We address such issues in the following subsection, presenting the appropriate techniques we adopt for facing them.

3.2.2 Standard approaches and their drawbacks

We begin by establishing the need for chronology. It is common to avoid shuffling time series data. This is especially true when creating patterns via lag inclusion as, by randomly sampling patterns into training and testing sets, it is likely that for a given pattern $\tilde{X}_i$ in the testing set that either $\tilde{X}_{i-1}$ or $\tilde{X}_{i+1}$ will be contained in the training set. As the features that are observed on both the AD and the NPP applications do not vary drastically over a single time interval, it is likely for the pattern $\tilde{X}_i$ to be very similar to both $\tilde{X}_{i-1}$ and $\tilde{X}_{i+1}$. This would result in the testing set containing patterns which were "almost seen" in the training phase, resulting in solutions which, while exhibiting a very high predictive accuracy on the testing set, are likely to have poor generalisation capabilities. This occurrence falls within the phenomena of *data leakage*, wherein data from our testing sets "leaks" into our training sets.

As mentioned in the previous section, we train our models in batch as opposed to online. We do so using k-fold CV, shuffling folds (within each fold patterns are ordered chronologically) while iteratively selecting each one to serve as a testing set. Both online and batch trained classifiers have been shown to be effective for predicting adverse events (Shi et al., 2019). If implemented within a predictive maintenance tool, this batch-trained model would be retrained whenever new, relevant data becomes available (i.e., after each new adverse event occurs). It is worth noting that our goal in preserving chronology is to ensure that patterns which occur within a short duration of our testing data are not included in our training sets. As such, we only require chronology within each fold. After that point chronology is no longer a requirement. I.e. having ensured that there is no data leakage in our fold selection and subsequent train test split we can freely shuffle the training data.

Shuffling folds of chronological data also presents an issue, as the system in question may change slightly over time as events and maintenance occur. However, this actually is an advantage as, should we find a model capable of accurately predicting events using this method, then such a model would have captured the behaviours of the system that are pervasive through the minor changes to the system caused by temporal variation, the occurrence of adverse events and maintenance. In light of this, an intuitive fold sampling method would be to simply partition the data into equal sized folds, preserving chronology in each of them. There are, however, significant issues with the application of this method to our problem, which we now outline.

Due to the infrequent occurrence of adverse events, the data sets of the two applications we consider are likely to suffer from a large *class imbalance*, detailed in Subsection 3.1. Class imbalance is known to be a problem in supervised anomaly detection (Luo et al., 2019; Maurya et al., 2015; Kozik and Choraś, 2016), as classifiers trained for these problems are likely to over-fit the majority class (the non-anomalous data). In our application, this would lead to models which over-predict the non-occurrence of adverse events, increasing the likelihood of the model failing to predict the occurrence of adverse events. As detailed in Subsection 3.1 the three main strategies for addressing class imbalance are data pre-processing, special-purpose learning methods and post-processing (Branco et al., 2016). In the next subsection, we propose *block sampling*, a method of informed undersampling specifically designed for the problem of predicting infrequent adverse events in systems from time-series data. In Section 3.3 we compare block sampling with other sampling techniques and evaluate the performance of a number of special-purpose learning algorithms. As discussed in Subsection 2.1.2, the common post-processing technique of thresholding the outputs of probabilistic classifiers requires an additional validation set with which to determine the optimal threshold using analysis of ROC curves. As mentioned in the previous subsection, there is a very low number of adverse events in our two data sets and, as such, there is only a small amount of data for which the warning label $\tilde{y}_i$ is 1. For this reason, it is not practical for us to perform thresholding as reserving an additional validation set for determining the optimal threshold would leave us with too little data for model training.

As the adverse events occur infrequently and irregularly in the two applications we consider, ideally we would partition the data set into the k folds in such a way that at least one adverse event occurs in each of them. Failing to guarantee this would lead to us evaluating model performance on testing sets which do not contain any adverse events. Any classifier which almost exclusively predicts the non-occurrence of adverse events would score extremely well on any such testing set, regardless of how well it predicts the event itself. This would lead to a further bias in the model selection process towards models which accurately predict the majority class.

Our binary warning labels simplify the state of the system to two states: the system being at risk of the adverse event occurring within the next ω time intervals if $\tilde{y}_i = 1$ and the system being in normal operations if $\tilde{y}_i = 0$. In the time period immediately following an event, we would not expect the system to have returned to normal operations typically due to changes caused either directly by the event or indirectly by the maintenance undertaken following the event. For this reason, while we would not label patterns corresponding to a point in time immediately following an event $\tilde{y}_i = 1$, they are not indicators that the system is in a stable state and so should not be labelled $\tilde{y}_i = 0$ either. For this reason, retaining such patterns in our data set with either label may negatively affect the performance.

3.2.3 Block sampling

In this subsection, we propose *block sampling*: a method of informed undersampling designed to address all of the previously described drawbacks to fold sampling by partition. We identify a set of k non-overlapping and contiguous *blocks* of equal size β within the data set, where k is the number of events that occur in the entire data set. We use k here as we will later use these blocks as folds in k-fold CV. Each block $(\tilde{X}_j, \tilde{y}_j)_{j \in T_i}$ is defined by a set of indexes T_i. The index set of each block T_i for $i \in \{1, \ldots, k\}$, consists of the end of an event (i.e., the last time step in an event with $\tilde{y}_j = 1$) together with the preceding β time intervals. For this reason, β should be selected to be small enough that the blocks do not overlap. More formally, for each label y_j where

$$\tilde{y}_j = 1 \wedge \tilde{y}_{j+1} = 0,$$

there exists a block defined as

$$T_i := \{j - \beta, \ldots, j\}.$$

We call all patterns whose original label y_j is 1 *event patterns*. Including event patterns in the data used to train our models makes the implicit assumption that the behaviour of the system during the event itself is indicative of its behaviour leading up to the event. Including these patterns would increase the number of patterns associated to the event, helping to reduce the large class imbalance and improving the strength of the classifiers. On the contrary, excluding the event patterns from model training makes the assumption that the behaviour of the system during the event is independent from its behaviour leading up to the event. If this is the case, including these patters would only add noise to the minority class, ultimately resulting in weaker classifiers. For these reasons, the decision to include or exclude the event patterns from model training is specific to the system and application at hand. Preliminary experiments or in-depth knowledge of the system and of the nature of the adverse event is required to make this decision. A block which does not contain event patterns $(\tilde{X}_j, \tilde{y}_j)_{j \in T_i'}$ is defined by a set of indexes $T_i' = \{j \in T_i | y_j = 0\}$. We generally refer to the index set defining a sampled block as

$$B_i = \begin{cases} T_i & \text{if} \quad \text{include event} \; = \; \textbf{True}, \\ T_i' & \text{if} \quad \text{include event} \; = \; \textbf{False}, \end{cases}$$

and therefore a sampled block is given by $(\tilde{X}_j, \tilde{y}_j)_{j \in B_i}$. In Subsection 3.3.2 we will demonstrate how, when using a standard method of equally partitioning the entire data set into equally sized folds, training models on blocks sampled in this manner from the training folds leads to better performance than using the entirety of the training folds and outperforms other sampling methods.

3.2.4 Block sampling for *k*-fold CV

Up to this point block sampling has simply been a sampling method. In this subsection we outline the process by which block sampling can be used to generate folds for *k*-fold CV which we will use in the experiments in Subsections 3.3.3 and 3.3.4 and we will detail the benefits of doing so.

When using block sampling to generate folds for *k*-fold CV, we do not partition the entire data set into folds, but instead, consider only the blocks sampled from entire data set which we use as folds in *k*-fold CV. Therefore, the set of folds we will use is,

$$\left\{ \left(\tilde{X}_j, \tilde{y}_j\right)_{j \in B_1}, \ldots, \left(\tilde{X}_j, \tilde{y}_j\right)_{j \in B_k} \right\}.$$

We define $\mathbf{B} = B_1, \ldots, B_k$ and $\widehat{X}^i = (\tilde{X}_j)_{j \in \mathbf{B} \setminus B_i}$ as all of the patterns within the blocks other than the i^{th} block. We similarly define $\hat{y}^i = (\tilde{y}_j)_{j \in \mathbf{B} \setminus B_i}$. At the i^{th} iteration of our *k*-fold CV, $(\widehat{X}^i, \hat{y}^i)$ will be our training set. As we aim to build models capable of predicting if an event is likely to occur in the next ω time intervals, we are not interested in predicting if an event is currently occurring. For this reason, we remove all event patterns from our blocks/folds when we use them for testing. We therefore define $\bar{X}^i = (\tilde{X}_j)_{j \in T'_i}$ and $\bar{y}^i = (\tilde{y}_j)_{j \in T'_i}$ as the pattern label pairs belonging to the i^{th} block excluding any event patterns. At the i^{th} iteration of our k-fold CV, $(\bar{X}^i, \bar{y}^i)$ will be our testing set. We therefore define the CV accuracy for a given classifier as

$$\frac{1}{k}\sum_{i=1}^{k} accuracy\left(h_{(\widehat{X}^i,\hat{y}^i)}(\bar{X}^i),\ \bar{y}^i\right).$$

In this formula $h_{(\widehat{X}^i,\hat{y}^i)}$ denotes the decision rule of a classifier which has been trained on the data and labels $(\widehat{X}^i, \hat{y}^i)$ and "*accuracy*" can be any function which maps a set of prediction, true label pairs to a number between 0 and 1. In Subsection 3.3 we will use balanced accuracy (defined in Subsection 2.1.2) for this accuracy metric.

Remark 3.1. The benefits to this method over the straightforward method of partitioning the entire data set into equally sized folds are as follows:

1. Using block sampling for doing this we guarantee that each testing set contains exactly one event; in the experiments in Subsection 3.3.2, where we partition the entire data set into equally sized folds, an issue we have to address is that some of these folds do not contain any adverse events rendering testing using them pointless.

2. Unlike when equally partitioning the entire data set, we do not have any patterns corresponding to the unstable period immediately following adverse events. As we are not interested in our model's performances on these patterns, their inclusion would only serve to disrupt training and bias our evaluation of model performance.

3. Sampling blocks to use as folds in CV dramatically reduces the class imbalance present in the original data set.

4. A drop in model performance due to dropping a large amount of the data when using only blocks for training is not a concern. As we will show in the experiments in Subsection 3.3.2, models trained on sampled blocks outperform models trained on folds generated by partitioning the entire data set for k-fold CV, even when other sampling techniques are applied.

5. As the data in each block is not shuffled, each block contains patterns that are *contiguous* in time. By sampling blocks in this manner we preserve in-fold chronology and avoid data-leakage.

6. It is common to use undersampling in concert with oversampling (Moniz et al., 2016; Estabrooks et al., 2004). Block sampling is no different in this regard and oversampling methods can be used to further reduce the class imbalance within each block.

3.2.5 Hyperparameter selection

In the experiments in Subsections 3.3.3 and 3.3.4 we perform *k*-fold CV using folds sampled using our block sampling method. In this section we outline the process by which we perform hyperparameter selection using block sampled folds. At each iteration of our *k*-fold CV, before evaluating our models on the testing set, we will perform a (*k*-1)-fold CV in order to tune the hyperparameters. The folds used in this (*k*-1)-fold CV will be the same *k*-1 folds originally assigned for training. This means that all of the desirable features of our block sampled folds, such as chronology and ensuring that each fold contains exactly one event, will be preserved in the (*k*-1)-fold CV. As with the basic *k*-fold CV method, we will have to modify this method to allow for including event patterns during training while excluding them during testing.

Here we define $\widehat{X}^{i,j} = (\tilde{X}_l)_{l\in\mathbf{B}\setminus\{B_i,B_j\}}$ and $\hat{y}^{i,j} = (\tilde{y}_l)_{l\in\mathbf{B}\setminus\{B_i,B_j\}}$, for $i,j \in [k]$, as all of the pattern label pairs within the blocks other than the i^{th} and j^{th} block. Let $\widehat{X}^i$, $\hat{y}^i$, $\bar{X}^i$, $\bar{y}^i$ be as they were defined in the previous subsection. Let Λ be the set of all combinations of values for the hyperparameter(s) which are being tested. As the decision rule found by a classifier is dependant on its hyperparameter selection, we now redefine $h^{\lambda}_{(\widehat{X},\hat{y})}$ as the model which has been trained on $(\widehat{X},\hat{y})$ using hyperparameter values $\lambda \in \Lambda$. Let $\lambda^{(\widehat{X},\hat{y})}$ be the set of hyperparameter values which gives the best performance on the pattern label pairs $(\widehat{X},\hat{y})$. Similarly, let $\alpha^{(\widehat{X},\hat{y})}$ be the average CV accuracy of models on $(\widehat{X},\hat{y})$. Algorithm 3 shows the modified version of (*k*-1)-fold CV for hyperparameter selection nested within *k*-fold CV for model selection.

Algorithm 3 Nested *k*-fold CV for model selection using grid-search for hyperparameter selection

```
Result: Average $k$-fold CV classification accuracy with tuned hyperparameters
for $i \in [k]$ do
    $\alpha^{(\widehat{X}^i,\hat{y}^i)} = 0$
    for $\lambda \in \Lambda$ do
        $\alpha \leftarrow \frac{1}{k-1}\sum_{j=1}^{k-1} \text{accuracy}\left(h^{\lambda}_{(\widehat{X}^{i,j},\hat{y}^{i,j})}(\bar{X}^j), \bar{y}^j\right)$
        if $\alpha \geq \alpha^{(\widehat{X}^i,\hat{y}^i)}$ then
            $\alpha^{(\widehat{X}^i,\hat{y}^i)} \leftarrow \alpha$
            $\lambda^{(\widehat{X}^i,\hat{y}^i)} \leftarrow \lambda$
        end if
    end for
end for
Return: $\frac{1}{k}\sum_{i=1}^{k} \text{accuracy}\left(h^{\lambda^{(\widehat{X}^i,\hat{y}^i)}}_{(\widehat{X}^i,\hat{y}^i)}(\bar{X}^i), \bar{y}^i\right)$
```

As mentioned previously, other sampling methods can still be used to further address class imbalance when using block sampling to generate folds for *k*-fold CV. To do so, we simply apply the sampling method to the data which will be used to train the model. We note that the shuffling of the training set which is performed as part of methods such as ROS and RUS (detailed in Subsection 3.1) is not a problem because, as mentioned in Subsection 3.2.2, we only require chronology within each fold in order to avoid data-leakage between our training and testing sets and as such, having assigned our training folds, we are free to shuffle the data within them.

3.3 Numerical experiments

In this section, we assess the predictive power of the modeling method we have proposed by experimenting on a data set corresponding to each of our two applications: foaming formation in AD and condenser fouling in NPP.

3.3.1 Setup

For the foam formation in AD problem, we rely on a data set provided to us by DAS Ltd in collaboration with Andigestion Ltd, a UK-based AD company producing renewable energy for the national grid through the combustion of biogas obtained from natural waste. The AD data we use consists in a time series collected from sensors at Andigestion Ltd's plant based in Holsworthy, UK. The data set consists of 14,617 hourly readings of 9 numeric variables. This amounts to 20 months of runtime from December 1st 2015 to July 31st 2017. Over this period, no anti-foaming agent was being used and 5 distinct foaming events occurred.

For the problem of predicting condenser tube leaks in civil NPP, we consider a data set collected by DAS Ltd during work carried out for a UK civil nuclear plant. This data consists of 30,664 readings taken at 3 hour intervals of 14 numeric variables, amounting to 10 years of data from 2009 to 2019. The 3 hour intervals between readings means that $\omega = 8$, for example, corresponds to 24 hours of warning. Over this period, 10 recorded tube leaks occurred. 4 of these leaks occurred shortly after another tube leak, and, as they cannot be considered independent events. Including the data corresponding to such leaks led to poor model performance, as, in the brief period between the two non-independent events, our assumption that the system was in a stable state did not hold. For this reason, the data corresponding to such leaks is discarded. This leaves us with a total of 6 fouling events.

We evaluate 11 classification algorithms used for solving Problem (1), selected among the most used algorithms in the literature for classification tasks: Support Vector Machine with a radial-basis-function kernel (SVM), Random Forest (RF), Balanced Random Forest (BRF), Multilayer Perceptron (MLP), Logistic Regression (LR), AdaBoost (AB), k-Nearest Neighbours (KNN), Decision Tree (DT), Gaussian Naive Bayes (GNB), Quadratic Discriminant Analysis (QDA), and Gradient Boosting (GB). For each of them, we rely on the corresponding Python implementation available in the Python package `scikit-learn` (Pedregosa et al., 2011). Notice that, due to the potential need to include event patterns in the training phase while excluding them from testing, we cannot use the `Scikit-learn` function `cross_val_score` for *k*-fold CV but, rather, we employ the custom methods presented in Section 3.2.

The only preprocessing that we apply to the data is a simple rescaling by which we restrict each variable to a range between 0 and 1. In our experiments, a scaler is fit to the training set and then used to scale the data from the training and testing sets. This is implemented using the function `MinMaxScaler` from `Scikit-learn`. Also note that all experiments are run using a consumer-grade personal computer, equipped with a quad-core i5 CPU and 8 GB of RAM.

3.3.2 Comparing block sampling to other sampling methods

In this experiment, we compare our proposed block sampling technique with other popular sampling techniques. In order to quantify the advantages of selecting folds using the block sampling method proposed in Section 3.2, we begin by adopting the standard approach of partitioning the entire data set into k folds, where k is equal to the number of events. We go on to perform k-fold CV by iteratively selecting one fold for testing and using the remaining folds for training. At each iteration of this k-fold CV, models are trained on 5 different training sets, each using a different sampling scheme to generate a training set from the $k-1$ folds assigned for training.

The first of these training sets is the *basic partitioning* training set, comprised of the entirety of the $k-1$ folds reserved for training. The second is the *oversampling* training set, created by oversampling the folds reserved for training to have an imbalance ratio of 1. The third is the *SMOTE* training set, created by using the SMOTE algorithm (a popular synthetic oversampling algorithm (Chawla et al., 2002)) to synthetically oversample the folds reserved for training to have an imbalance ratio of 1. Fourth, we have the *block sampling* training set, created using our proposed block sampling method to create a block for each event contained within the folds reserved for training. Finally we have the *undersampling* training set. Undersampling to an imbalance ratio of 1 would lead to poor results as the total size of the data set would become very small (containing as few as 104 total patterns). For this reason the undersampling training set is constructed by undersampling the folds reserved for training to have an imbalance ratio of 0.05 (the same imbalance ratio achieved by block sampling). We then train classification models on these 5 training sets and evaluate the performance of all of them on the same testing set, the entire fold reserved for testing. This process is repeated using each of the k folds as a testing set and the average CV accuracy of each model is compared.

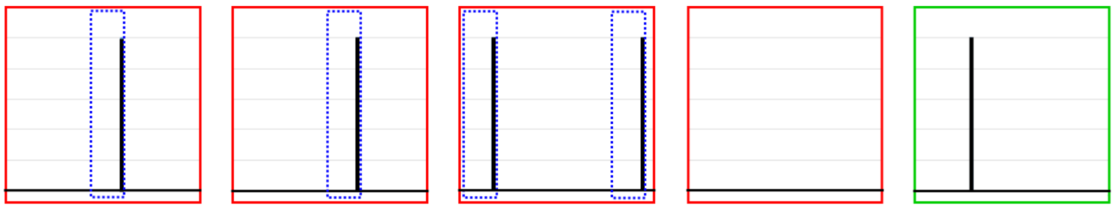

FIGURE 3.1: The entire time series for the AD application is partitioned into $k = 5$ equally sized folds. Foaming events are shown in black. One fold, shown in green, is reserved for testing. The remaining 4 folds, shown in red, are reserved for training. These training folds will be used to generate the basic partitioning, oversampling, SMOTE, block sampling and undersampling training sets. The blue boxes show the blocks which are sampled from the folds reserved for training to create the block sampled training set. At the next iteration of CV a different fold will be selected for testing.

Figure 3.1 gives an example of one iteration of this process on the AD data. As there are 5 foaming events, the data is partitioned chronologically into 5 folds. One of these folds is selected for testing and the remaining 4 are assigned for training. The basic partitioning, undersampling, oversampling, SMOTE and block-sampling training sets are created as described previously, and 11 classifiers are trained on each of them for 55 total models. All of these classifiers are then tested on the entire testing fold, and their results are compared. The process is repeated 5 times and the average accuracy of the classifiers trained with each type of training set is compared.

As stated in Section 3.2, due to the infrequent and irregular occurrence of events, it is likely that some of the partitioned folds in this experiment will contain no events. An example of this can be clearly seen in Figure 3.1. The inclusion of iterations in which these folds are used for testing in the calculation of the average testing accuracy would introduce a bias towards models which correctly predict that the adverse event will not occur regardless of how well they would predict that it will. For this reason, we do not include iterations where we test folds containing no events in the calculation of the average testing accuracy.

Through preliminary experiments, we have found that, for both applications, the inclusion of 4 lags in each data point (i.e., the adoption of $\tau = 4$) resulted in stronger classifiers. In this initial experiment, we set ω equal to 24 and 8 for the foaming in AD and tube leakage in NPP problems respectively, corresponding to a warning of 24 hours. This means that, when including the event patterns in the training sets, the partitioned training sets have an imbalance ratio of 0.016 for the AD application and of 0.003 for the NPP application. When using the block sampling method shown in Section 3.2, we sample 1000-hour blocks from the AD data set and 750-hour blocks from the NPP data set. This corresponds to slightly over a month of runtime in each block and results in the block sampled training sets having an imbalance ratio of around 0.05. As the AD data is recorded at hourly intervals and the NPP data is recorded at 3 hourly intervals, the sampled blocks contain 1000 and 250 patterns, respectively.

While patterns occurring within the event itself are always excluded from the testing set, we will experiment with including such patterns in the training set using the methodology laid out in Section 3.2.

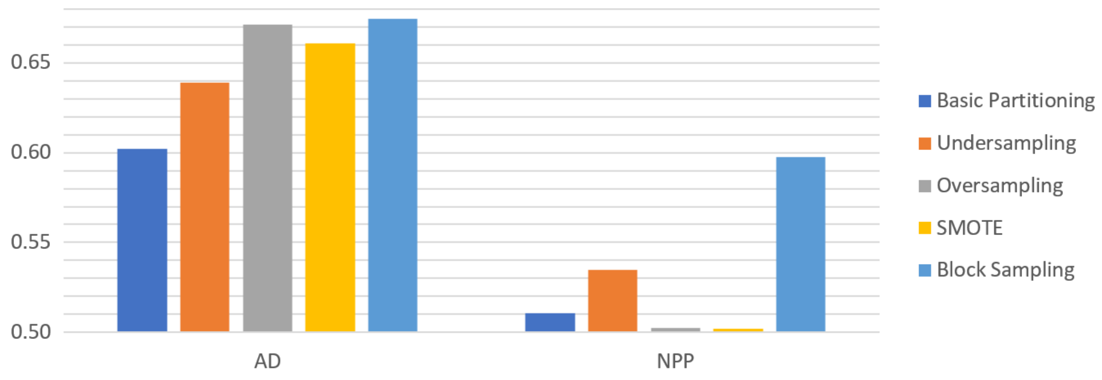

FIGURE 3.2: Average CV accuracies across all 11 classifiers for each sampling strategy in each application when including event patterns in the training data. Full experimental results can be found in Tables A.2 and A.2 in Appendix A.3. A baseline of 0.5 is used for this graph as this is the balanced accuracy that randomly guessing would achieve.

Tables A.1 and A.2 in Appendix A.3 show the full results of this experiment. From inspection of these results, we can see that, in both applications, models which include event patterns in their training outperform those which do not. As shown in Figure 3.2, when including event patterns in training, constructing models using the block sampling training sets outperformed models trained using any other sampling techniques. As can be seen in Tables A.1 and A.2 in Appendix A.3, when including the event patterns in the training sets, using the block sampling training sets leads to better accuracy than any other sampling method for 8 of the 11 classifiers in the AD application and 10 of the 11 classifiers in the NPP application. This result shows that, while not all of the 11 classification algorithms achieved their best accuracy using block sampling, block sampling was the most robust sampling method for improving model performance in these applications as it provided the most consistent improvement in performance over the 11 classifiers. Training models on the block sampled training sets instead of the basic partitioning training sets folds improved the average testing accuracy of the 11 classifiers from 60.2% to 67.5% for the AD application and from 51.1% to 59.7% for the NPP application. In subsequent experiments we investigate using further sampling methods within each sampled block to further improve the performance of models.

3.3.3 AD experiments

In the previous section, it was established that models for predicting foaming which were trained on sampled blocks significantly outperform those trained on the entirety of the partitioned folds and models which included event patterns in training outperformed those which did not. For this reason, together with the other reasons given in

Subsection 3.2.4, in the following experiments we perform k-fold CV where each fold is a block containing one foaming event and event patterns are excluded from testing but included in training. For further details on block sampling and event inclusion versus exclusion see Subsection 3.2. This means that, unlike in the previous experiment, there will be an event in the testing set in every iteration, and so, no iteration's results will need to be disregarded.

Firstly, we evaluate the performances of the classifiers with $\omega \in \{12, 24, 36, 48, 96\}$ resulting in models which provide up to 12, 24, 36, 48 and 96 hours of warning. Table A.3 in the Appendix shows the average balanced accuracy of the 11 classifiers over 5-fold CV with a range of values of ω. The GNB classifier outperformed all others for all values of ω, with CV accuracies ranging from 0.671 to 0.875.

These results show that the three classifiers achieving the best performance for predicting foaming for a range of values of ω are SVM, BRF and GNB. In hopes of mitigating the negative effects that class imbalance may have on these three classifiers, we assess the effects of undersampling and oversampling. For details on how these sampling techniques were implemented, we refer the reader to Section A.1 in the Appendix. Table A.4 in Appendix A.3 shows the results of 5-fold CV, comparing the results of using different sampling techniques. As expected, sampling techniques generally do not improve the performance of the BRF classifier due to the fact that it already integrates them. The GNB classifiers only see minor increases in performance for all values of ω when the training set is under or oversampled. This is to be expected as, unlike BRF and SVM which are discriminative classifiers, GNB is a generative classifier and, as such, is less vulnerable to class imbalance. SVM classifiers show a significant increase in average CV accuracy across all tested values of ω with an average increase of 6.7% when using undersampling and 3.4% when using oversampling.

Finally, whereas in previous experiments we adopted the default hyperparameter selections, here we use a modified version of grid-search detailed in Algorithm 3 to tune the hyperparameters of the three best-performing classification algorithms using a grid of roughly 100 combinations of hyperparameter values. Details on default hyperparameter selections and the grids of hyperparameter values used for each classification algorithm can be found in Section A.2 in the Appendix. Table A.5 in the appendix compares the performances of the three best performing models with tuned hyperparameters and with default hyperparameter values for a range of values of ω. We observe that GNB and SVM classifiers with tuned hyperparameters see improved average accuracy over those with preselected hyperparameters values across all tested values of ω, with an average increase of 2.1% and 5.0% respectively. The performance of the BRF classifiers is, in general, not improved by hyperparameter tuning. A possible explanation for this is that the small size of our training set has caused over-fitting to occur during the hyperparameter tuning phase, leading to poor generalisation on the testing set.

Figure 3.3 shows the average testing accuracy of the best classifiers for various values of ω over 5-fold CV of the AD data. Each classifier included in this graph achieved the highest average testing accuracy of all classifiers tested for at least one value of ω. Though sampling techniques and hyperparameter tuning both improved the performance of our best-performing classifiers (SVM and GNB), tuning hyperparameters in combination with sampling the training set did not lead to strong classifiers. In light of these results, $\omega = 24$, corresponding to 24 hours of warning, results in the models which offered similar or improved accuracy compared to those trained with $\omega = 12$ whilst providing up to double the amount of warning. The best performing model for any value of ω was a GNB classifier using oversampling with $\omega = 24$, which achieved an average testing accuracy of 88.6%.

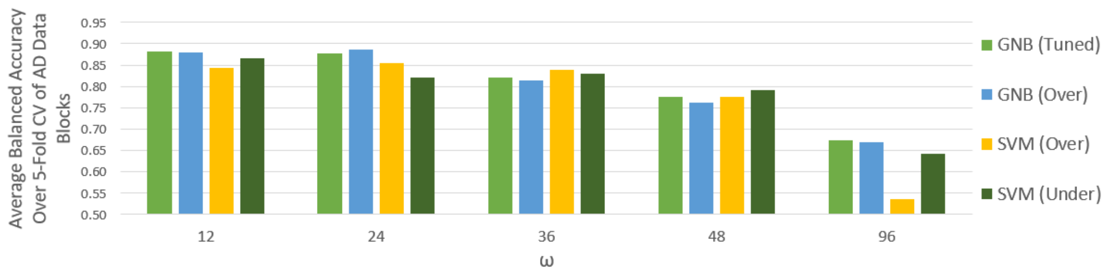

FIGURE 3.3: Graph of the average testing accuracies of the classifiers which perform best at one or more value of ω in predicting foaming

Figure 3.4 shows the testing predictions of GNB trained on an oversampled training set with a 24-hour event association. Each plot shows the last 250 hours of a block as well as the predictions of a classifier which has been trained on the remaining 4 blocks. The blue line shows the predicted probability of foaming and the green dashed line shows the corresponding warning labels $\tilde{y}$ which go from 0 to 1 24 hours before the foaming would occur. As can be seen, this classifier is able to accurately preempt each one of the foaming events given that it has been trained on the remaining four events. While there are periods of false positives (i.e. predicting foaming when it is not imminent) in these predictions, these periods account for very little cost as, should this classifier be implemented into a predictive maintenance model, false positives such as these would result in anti-foaming agent being needlessly injected into the digester. Despite some wastage, this would still result in a massive reduction in the amount of anti-foaming agent used when compared to the current method for guaranteeing the non-occurrence of foaming, which is to routinely inject anti-foaming agent into the digester at regular intervals.

3.3.4 NPP experiments

As in Subsection 3.3.3, in the following experiments we perform k-fold CV where each fold is a sampled block (as detailed in Subsection 3.2.4) containing one condenser tube

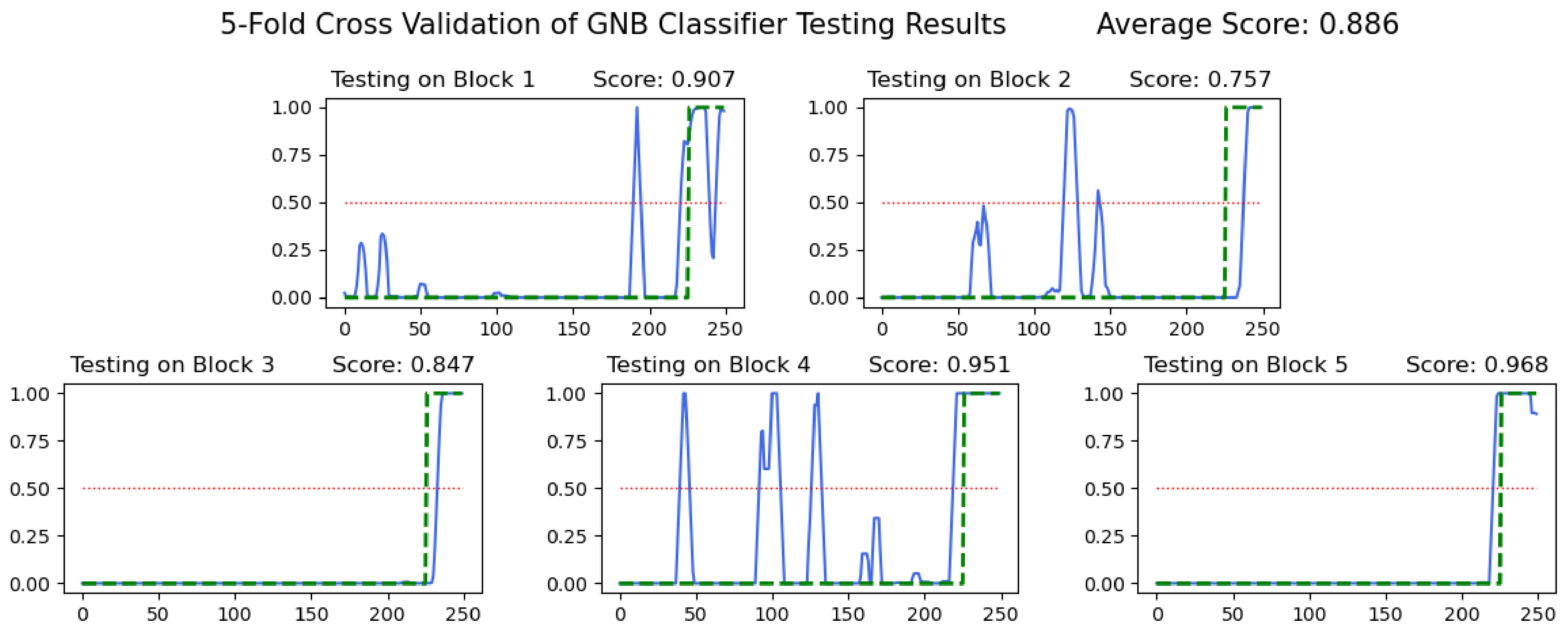

FIGURE 3.4: Examples of five GNB classifiers, trained on oversampled data from 4 of the AD blocks, being tested on the block which they had not seen during training. The model's predicted likelihood of foaming is shown in blue and the warning label is shown in green. For readability, the model's prediction has been smoothed using a rolling average of the latest 5 predictions.

leak event sampled from the NPP data as described in the Subsection 3.3.2 and event patterns were excluded from testing but included in training.

Firstly, we evaluate the performances of the classifiers with $\omega \in \{4, 8, 12, 16, 32\}$ which, due to the data points being recorded at 3-hour intervals, results in models which provide up to 12, 24, 36, 48 and 96 hours of warning. A comparison of the average balanced accuracy of the 11 classifiers over 6-fold CV with a range of values of ω can be found in Table A.6 in Appendix A.3. The results suggest the three classifiers achieving the best performance for predicting condenser tube leaks for a range of values of ω are SVM, AB, and GNB, with best CV accuracies of 0.634, 0.644, and 0.609 respectively. In particular, AB performs best for small values of ω, while GNB performs best for large values of ω, and SVM outperforms both for $\omega = 12$.

Table A.7 in the Appendix shows the results of 6-fold CV, comparing the results of using undersampling and oversampling before training the three best performing classifiers. Details on the implementation of these sampling techniques can be found in Section A.1 in the Appendix. SVM and GNB classifiers both benefit from oversampling, with an average increase in accuracy, across all tested values of ω, of 10.0% and 5.5% respectively. The use of sampling techniques results in little to no improvement in the accuracy of the AB classifiers while making the performance quite unreliable and, upon rerunning the 6-fold CV with a different random seed, the results from this classifier varied considerably.

Finally, we use a modified version of grid-search detailed in Algorithm 3 to tune the hyperparameters of the three best-performing classification algorithms using a grid of roughly 100 combinations of hyperparameter values. Details on the implementation of this hyperparameter tuning can be found in Section A.2 in the Appendix. A comparison of the performances of the three best performing models with and without tuned hyperparameters for a range of values of ω can be seen in Table A.8 in the Appendix. As the table indicates, for this dataset, hyperparameter tuning does not consistently improve performance and, where it does, the improvement is typically very minor. As in the AD application, this could be due to over-fitting the hyperparameter selection to the relatively small amount of data.

Figure 3.5 shows the average balanced accuracy of the best models for various values of ω over 6-fold CV of the NPP data blocks. Each classifier included in this graph had the highest average testing accuracy of all classifiers tested for at least one value of ω. Unlike with the AD data set, where a range of classifiers performed well at different values of ω, here we have one classifier outperforming all others for the majority of tested values of ω. In particular, the SVM classifier using oversampling outperforms all other classifiers for ω less than or equal to 36. For values of ω larger than 36, the GNB classifier using oversampling gives the best performance; however, all classifiers have relatively poor accuracy for large values of ω. For this reason, we recommend adopting the smallest value of ω which would still allow time for preventative measures to be undertaken. An SVM classifier trained using oversampling with $\omega = 4$ (corresponding to 12 hours of warning) is able to achieve an average testing accuracy of 76.2%.

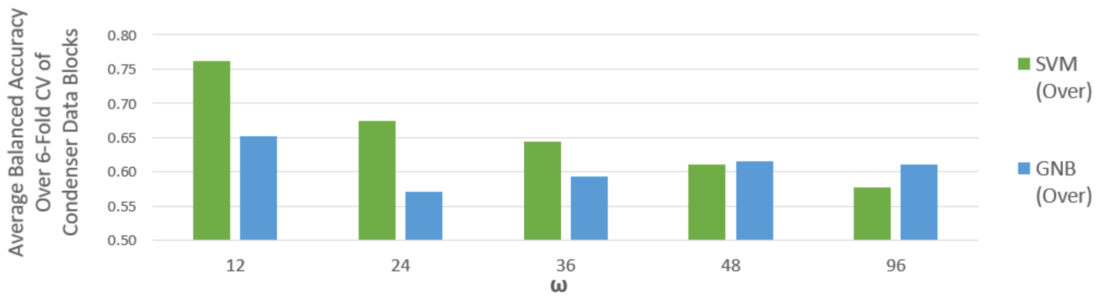

FIGURE 3.5: Graph of average testing accuracies of the classifiers which perform best at one or more value of ω in predicting tube leaks

When predicting condenser tube leaks in the NPP application, our best performing model, SVM trained on an oversampled training set with a 12 hour event association, achieves an average accuracy of 0.762 across the 6 blocks (corresponding to the 6 tube leaks). This classifier is able to accurately preempt 4 of the 6 tube leakage events given that it has been trained on all of the other 5 events, as can be seen in Figure 3.6. Each plot shows the last 100 3-hour intervals of a block as well as the predictions of a classifier which has been trained on the remaining 5 blocks. The blue line shows the predicted probability of tube leakage and the green dashed line shows the corresponding warning labels $\tilde{y}$ which go from 0 to 1 12 hours before the tube leak would occur. Whilst this result is not as strong as those for the AD application, the fact that with as few as 5 tube leaks to learn from it is possible to correctly preempt most condenser tube leaks with a low number of false positives is encouraging.

3.4 Conclusions

In this chapter, we have proposed a framework for modeling the problem of predicting infrequently occurring adverse events from time series data as a classification problem. In particular, we have focused on two applications: predicting foaming in AD and predicting condenser tube leaks in civil NPP. We have discussed the drawbacks of standard approaches when working with small, highly imbalanced data sets and proposed variations on these techniques which best leverage the structure of our classification problems and mitigate many of the difficulties they pose. We have observed that, using our proposed block sampling method as opposed to standard methods for constructing folds, leads to an increase in the average testing accuracy across all 11 classifiers from 60.2% to 67.5% for the AD application and from 51.1% to 59.7% for the NPP application and consistently results in better performance than alternative sampling methods.

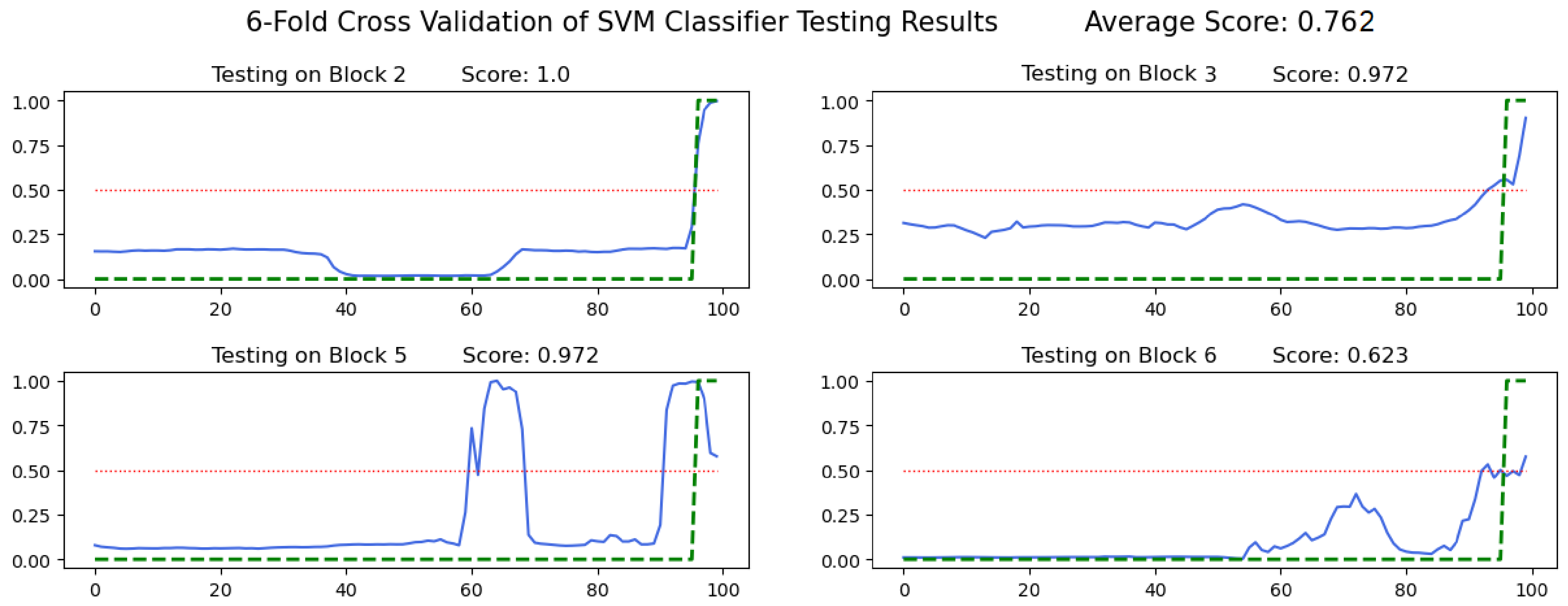

FIGURE 3.6: Examples of 4 SVM classifiers, trained on oversampled data from 5 of the NPP blocks, being tested on the block which they had not seen during training. The model's predicted likelihood of tube leakage is shown in blue and the warning label is shown in green.

In this experiment we also observe that, while event patterns were excluded from all test sets, including them in the training of models results in a significant increase in accuracy. Our extensive set of experiments have revealed that our proposed framework for building and comparing a range of classification algorithms, sampling techniques and hyperparameter selections results in models which achieve a good performance in predicting adverse events in both of our applications. Our results have shown the importance of evaluating a variety of classification algorithms, sampling techniques and hyperparameter tuning methods across a range of values of ω as these differences lead to significant variation in classifier performance.

Using our proposed framework for the AD application, we were able to train models capable of accurately preempting each of the 5 foaming events, having been trained on the remaining 4, with up to 24 hours warning and an average balanced accuracy of 88.6%. Models such as these could be very easily integrated into a decision support tool for instructing plant operators when to inject anti-foaming agent to avert an imminent foaming event. This method of only injecting anti-foaming agent when the digester is at high risk of foaming would result in a great reduction in the amount of expensive anti-foaming agent required to avoid foaming when compared with the current method of avoiding foaming by routinely injecting anti-foaming agent.

For the NPP application, using our proposed framework of model training and evaluation we were able to produce models capable of accurately preempting 4 of the 6 tube leakage events, having been trained on the remaining 5, with up to 12 hours warning and an average balanced accuracy of 76.2%. Recall that, in the AD application, there exists a scheme which can guarantee the non-occurrence of adverse foaming events: routinely injecting anti-foaming agent into the digester. As such, it is important that any alternative method we propose is also capable of preventing all foaming events. In the NPP application, no such scheme exists and, as such, any tube leakage events which we can enable plant operators to repair off load by providing them with advanced warning of the leak's occurrence would result in fewer unplanned turbine trips and, as such, a reduced loss of revenue. Our experiments suggest that integrating the models we have shown, trained on only 5 tube leakage events, into a decision support tool would lead to a significant reduction in the number of unplanned turbine trips. These results are encouraging and, should additional data be collected, we would expect to be able to predict these tube leakage events with even greater accuracy. In future works, we hope to apply this method to predicting foaming in other systems such as waste water treatment plants which utilise AD, as well as other systems with a steam circuit which utilise a condenser.

Chapter 4

Deep learning methods for screening patients' S-ICD implantation eligibility

The chapter is organised as follows. In Section 4.1 we provide a brief recap of the problem and provide a review of some related works. In Section 4.2, we first introduce our methodology and the preprocessing techniques we use to de-noise the ECG signals. We then detail the architectures of the deep learning models we propose for the task. We then outline the way our models are evaluated, the way we build ensemble models, and the way hyperparameter tuning is carried out (which is key to the performances we achieve). In Section 4.3, we assess the accuracy of our models and methods by selecting the best training method, tuning the corresponding hyperparameters, and creating ensemble models. In Section 4.4, we demonstrate how our best performing models can be incorporated into a clinical screening process for predicting the T:R ratio. Finally, in Section 4.5, we conclude this chapter and outline directions for future work.

Besides the author and his supervisors, this work is done in collaboration with our partners, Dr Mohamed ElRefai and Dr Paul R. Roberts, who are clinical cardiologists at the University Hospital Southampton, as well as Dr Benedict W. Wiles, a cardiologist from St George's University Hospitals NHS Foundation Trust. The work presented in this Chapter has been published in the two following papers:

Dunn, A. J., ElRefai, M. H., Roberts, P. R., Coniglio, S., Wiles, B. M., and Zemkoho, A. B. (2021). Deep learning methods for screening patients' S-ICD implantation eligibility. Artificial Intelligence in Medicine, *119:102139.*

Dunn, A. J., ElRefai, M. H., Roberts, P. R., Coniglio, S., Wiles, B. M., and Zemkoho, A. B. (2022). Deep learning and hyperparameter optimization for assessing one's eligibility for a subcutaneous implantable cardioverter-defibrillator. Preprint available at https://optimization-online.org/?p=21066. *Under review in* Annals of Operations Research.

A further three papers have been published in which the AI screening tool we have developed is applied in a clinical setting:

ElRefai, M., Abouelasaad, M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., Wiles, B. M., and Roberts, P. R. (2022b). Eligibility for subcutaneous implantable cardiac defibrillator utilising artificial intelligence and deep learning methods for prolonged screening: where is the cut-off? Europace, *24 (Supplement_1):euac053–447.*

ElRefai, M., Abouelasaad, M., Wiles, B. M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., and Roberts, P. R. (2022c). Deep learning-based insights on T:R ratio behaviour during prolonged screening for S-ICD eligibility. Journal of Interventional Cardiac Electrophysiology, doi.org/10.1007/s10840-022-01245-6.

ElRefai, M., Abouelasaad, M., Conibear, I., Wiles, B. M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., and Roberts, P. R. (2022a). The use of artificial intelligence and deep learning methods in subcutaneous implantable cardioverter defibrillator screening to optimise selection in special patient populations. Europace, *24(Supplement_1):euac053–448.*

In two of these papers, our tool is used to assess what the optimal choice of the T:R ratio cut-off used in the screening should be. In the third, it is used to evaluate the variation of the T:R ratio of patients with various heart conditions.

4.1 Problem description and related works

As detailed in Subsection 1.2, patients are screened before they can be implanted with Subcutaneous Implantable Cardioverter-Defibrillators (S-ICDs) to treat ventricular arrhythmias (VA). The current practice is to examine a patient's T:R ratio—a major predictor of a patients likelihood of T Wave Oversensing (TWOS) leading to inappropriate shocks from an S-ICD—over 10 seconds, with patients who have a suitably low T:R ratio (below 1/3) being deemed eligible for S-ICD implantation. However, due to the fact that a patient's T:R ratio can fluctuate significantly over time, this 10-second snapshot is insufficient to reliably reflect the normal behaviours of the patients T:R ratio. In this chapter we propose a method for automated T:R ratio prediction and demonstrate how such a method can be integrated into an AI tool, allowing clinicians to perform significantly longer screenings, providing a far greater level of detail of the behaviour of the patient's T:R ratio and thus, allowing clinicians to much more reliably assess their implantation eligibility.

Machine learning methods have been used for ECG analysis in a variety of applications. There has been a wealth of work in the classification of various Cardiovascular Diseases (CVDs) from ECG data (Vemishetty et al., 2019, 2016; Rocha et al., 2008; Roberts et al., 2001; Zhang et al., 2020; Pourbabaee et al., 2018; Kiranyaz et al., 2015; Fan et al., 2018; Lih et al., 2020). Other applications of machine learning in ECG analysis include detecting seizures and heart attacks (Lee et al., 2014; Liu et al., 2017), predicting patients' blood pressure (Miao et al., 2020), detecting patients' facial expressions (Dawid, 2019) and analysis of ECG of the brain has been used for creating Brain Computer Interfaces (BCI) capable of detecting which body part the subject was completing a task with (Djemal et al., 2016; Chen et al., 2014).

A popular technique for preprocessing ECG data is to create its Phase Space Reconstruction (PSR) matrix. Typically, features are extracted from the PSR matrix of ECG data which can then be used as inputs for a classification model. Box counting as well as column and row statistics are features often extracted from the PSR matrix of ECG data. These methods have been used in the prediction of CVD (Vemishetty et al., 2019, 2016; Rocha et al., 2008; Roberts et al., 2001), creating BCIs (Chen et al., 2014; Djemal et al., 2016) and detecting facial expressions (Dawid, 2019). These approaches are all centred around manually selecting features to extract from the PSR matrix. Our proposed method diverges from this by using the whole PSR matrix as the input to a model which is itself capable of extracting features.

The method we propose is based on deep learning and, in particular, CNNs and MLPs. Deep neural networks (such as CNNs and MLPs) are examples of models which can extract features without supervision. During training, the many layers of the network extract features of the input image which are most impactful in accurately determining the model's output. As such, these models can replace the need for time consuming feature extraction and engineering and can derive much more descriptive features. CNNs have been used in ECG analysis for classifying heart attacks (Liu et al., 2017; Cho et al., 2020), CVDs (Siontis et al., 2021), atrial fibrillation (Fan et al., 2018; Pourbabaee et al., 2018; Jo et al., 2021) and other arrhythmias and rhythm abnormalities (Kiranyaz et al., 2015; Zhang et al., 2020; Sangaiah et al., 2020; Lih et al., 2020; Zhu et al., 2020) as well as for predicting blood pressure (Miao et al., 2020), locating cardiac accessory pathways (Nishimori et al., 2021), recognising early signs of heart failure with reduced ejection fraction (Cho et al., 2021) and detecting digoxin toxicity (Chang et al., 2021), electrolyte imbalance (Kwon et al., 2021) and anaemia (Kwon et al., 2020). All of these methods use the filtered ECG as the model input, where each lead corresponds to a single 1D signal. Differently from these works, in our model we include the additional step of transforming the 1D ECG signal of each lead into a 2D phase-space reconstruction (PSR) image which is then fed as input to a deep learning model. To the best of our knowledge PSR matrices have never before been used directly as input to a deep learning model. As a second element of novelty and differently from most of the literature

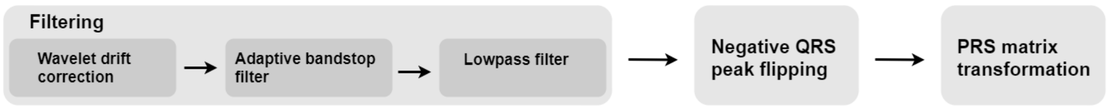

FIGURE 4.1: Flowchart of the methods in our ECG data preprocessing step.

where CNNs are used for classification (see Babu et al. (2016) for an exception), in this work we use them to perform a regression task.

4.2 Methodology

We propose a new screening process where a Holter®—a portable ECG recording device—is used to record 24 hours of ECG data from three leads corresponding to the three vectors utilised by the S-ICD. This data is then split into 10-second segments and an artificial neural-network based model is used to predict the T:R ratio for each 10-second segment. The cardiologist would then be able to review the behavior of the patient's T:R ratio over the 24-hour period and evaluate their eligibility. We rely on a number of filtering techniques for removing noise from the ECG signals. We then use PSR, a popular technique in waveform analysis, to convert the ECG signal into an image of its PSR matrix with which we then train a range of deep learning models to predict the T:R ratio from these images.

The most straightforward approach to measuring the T:R ratio is to locate the R and T waves and measure their amplitudes. However, by explicitly detecting and measuring the peaks of the R and T waves, we run the risk of TWOS when the characteristics of the T wave becomes similar to those of the R wave. To avoid this, by analysing the PSR matrix of multiple PQRST complexes at once, our model aims to predict the T:R ratio without ever explicitly locating or measuring the R or T waves within the original ECG recording.

4.2.1 Preprocessing

Preprocessing involves filtering the ECG data, performing transformations to emulate the methods used within S-ICDs and creating images by creating PSR matrices from the filtered ECG signals. Figure 4.1 gives an overview of the preprocessing techniques used to prepare our data for the training of the regression models.

4.2.1.1 Data structure

We consider data in the form of 10-second segments of three-lead ECG recordings from Holter leads corresponding to the vectors used by the S-ICD. These three-lead ECG recordings are then broken out into three single-lead ECG signals. Each single-lead 10-second ECG segment is annotated with the positions of the peaks of the T and R waves occurring in this period.

From these annotations, we are able to calculate the dependant variable for our regression problem: the average T:R ratio. As mentioned previously, the T:R ratio of a single PQRST complex is simply the ratio between the amplitudes of the T and R waves. For the purposes of this work, we will consider this ratio in the form of a fraction. The average T:R ratio y_i for a 10-second ECG segment $X_i = \{x_1, \ldots, x_{10 \cdot f}\}$, where f is the sampling frequency of the signal, with T-peak annotations at indexes $\{T_1, \ldots, T_n\}$ and R-peak annotations at indexes $\{R_1, \ldots, R_n\}$, is given by

$$\frac{\sum_{i=1}^{n} x_{T_i}}{\sum_{i=1}^{n} x_{R_i}}.$$

When the T wave has negative amplitude, this fraction is negative. From a clinical perspective, we are only interested in the magnitude of the T:R ratio. However, from a signal processing perspective, there is a great difference between a PQRST complex with a negative T wave and one with a positive T wave. For this reason, we will preserve the sign of the T:R ratio, as the loss of information resulting from considering only the magnitude of the ratio would lead to a reduction in the accuracy of our models. Having built a model capable of predicting T:R ratios from 10-second ECG segments, we take the magnitude of this model's outputs for use in a clinical tool. Our choice to consider the T:R ratio rather than the R:T ratio, which is more common in the literature, is well motivated. As the T wave amplitude approaches 0, very small changes in the T wave amplitude can result in extreme changes in the R:T ratio. This massive variation in R:T ratio for very similar ECG signals makes it inappropriate for use as a label in our regression problem. Typically, the R wave is of greater amplitude than the T wave. Because of this, the T:R ratios of a set of ECG segments are well distributed between 0 and 1. For this reason, we use the T:R ratio as our dependent variable in our regression problem. If the situation requires it, the R:T ratio can of course be derived from our model by simply taking the inverse of the model's output.

4.2.1.2 Filtering

Figure 4.1 gives an overview of the filtering methods we will use to remove noise from our ECG signal (Lugovaya, 2005). Firstly, baseline drift correction is implemented using one-dimensional Discrete Wavelet Transformation (DWT). The ECG signal is decomposed at 9 levels, using the Daubechies 8 (db8) wavelet, then reconstructed using only level 9 coefficients. This reconstructed signal is the low frequency component for the ECG signal which is assumed to be the drifting baseline. Subtracting this from the

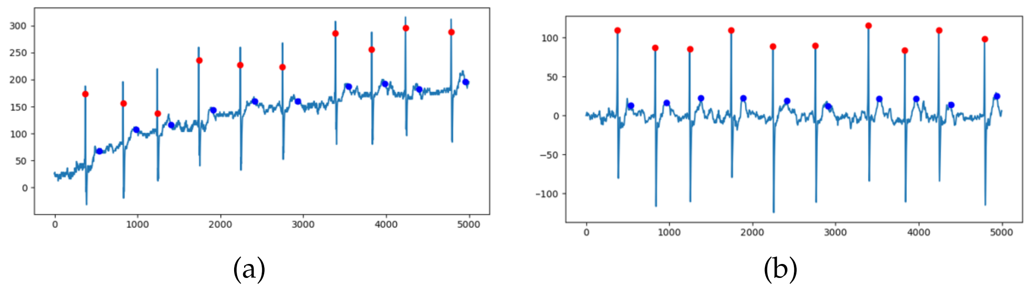

FIGURE 4.2: (a) Example of a 10-second ECG segment pre-filtering. R and T peaks shown in red and blue, respectively. (b) The same 10-second ECG segment post-filtering.

original signal leaves us with an ECG signal with a stable baseline of value 0. Adaptive bandstop filtering is used to suppress power-line noise with a frequency of 50 Hz, while a lowpass filter is used to remove the remaining high-frequency noise. Having applied these filters, the locations of the R and T peak markers may no longer be correct. To account for this, a small region around the R peak is searched for a maximum and this maximum is taken as the new R peak. Similarly, a small region around the T peak is searched and the maximum or minimum value in this region is taken as the new T peak depending on whether the T peak was positive or negative, respectively. Figure 4.2 gives an example of an unfiltered 10-second ECG segment as well as the same segment after filtering has been applied.

4.2.1.3 Negative QRS peak flipping

While R waves are strictly positive, a PQRST complex with a small R wave could be prone to the T wave being labeled as an R wave, leading to double counting. To address this, the standard algorithm employed within S-ICDs will search the QRS complex (shown in Figure 1.3) for negative Q or S waves with greater amplitude than that of the R wave. Provided that the amplitude of the largest wave in the QRS complex is significantly larger than that of the T wave, the S-ICD will not deliver a shock. For this reason, we are not strictly interested in predicting the T:R ratio but, rather, the ratio between the amplitudes of the wave of greatest amplitude in the QRS complex and the T wave. To account for this in our analysis, after filtering (including baseline drift correction), we search for the peak of greatest magnitude within a narrow region surrounding each R peak label. For simplicity, in the rare case that the largest peak found is not the R peak, these new peaks are assigned as the new R peak despite, in fact, being a Q or an S wave. As R waves are always positive, to ensure that all signals handed to our models have a positive wave labeled as the R wave, when the R peak label is moved to a Q or an S wave, the whole ECG signal is flipped, making these waves positive. Figure 4.3 shows some possible QRS complexes where the amplitude of the Q or the S wave is greater than that of the R wave.

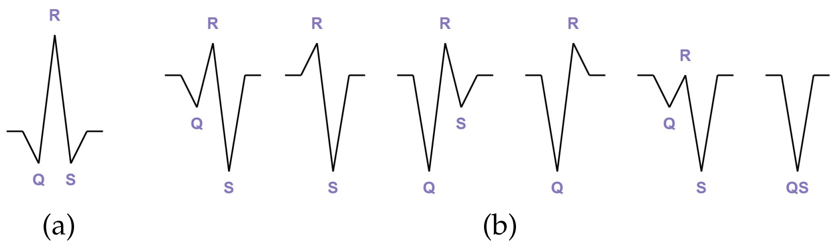

FIGURE 4.3: (a) An example of a normal QRS complex. (b) Examples of QRS complexes in which the R wave is not the wave of greatest amplitude.

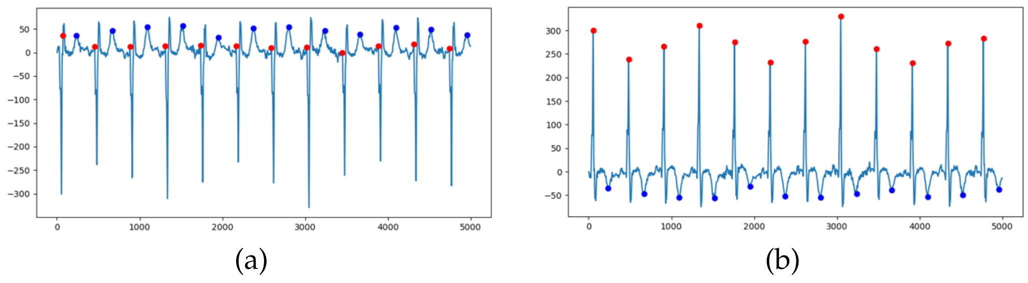

FIGURE 4.4: (a) Example of a filtered 10-second ECG segment. R and T peaks shown in red and blue, respectively. (b) The same 10-second ECG segment post-flipping.

Figure 4.4a gives an example of a filtered 10-second ECG segment while Figure 4.4b shows the same segment after the negative QRS peaks have been detected, the signal has been flipped and the peaks have been reassigned.

4.2.1.4 Phase space reconstruction

Phase Space Reconstruction (PSR) is a method for feature engineering which we use in Chapter 4. Analysing the PSR of a signal can allow one to capture the repeating behaviours of the signal and as such deriving features from the PSR can lead to drastically improved performance for prediction models using time series data. When creating a phase space reconstruction, a one-dimensional time series $x_1, x_2, \ldots, x_n$ is transformed to an $(n-(d-1)\tau) \times d$-dimensional Phase Space Reconstruction (PSR) matrix. The transformed time-series is given as

$$\boldsymbol{x} = \begin{bmatrix} \frac{1}{q}x_1 & \frac{1}{q}x_{1+\tau} & \frac{1}{q}x_{1+2\tau} & \cdots & \frac{1}{q}x_{1+(d-1)\tau} \\ \frac{1}{q}x_2 & \frac{1}{q}x_{2+\tau} & \frac{1}{q}x_{2+2\tau} & \cdots & \frac{1}{q}x_{2+(d-1)\tau} \\ \vdots & \vdots & & & \\ \frac{1}{q}x_{n-(d-1)\tau} & \frac{1}{q}x_{n-(d-2)\tau} & \frac{1}{q}x_{n-(d-3)\tau} & \cdots & \frac{1}{q}x_n \end{bmatrix},$$

where $q = \max\{|x_i| : i \in [n]\}$. Therefore, each scaled point $\frac{1}{q}x_i$ in the time-series signal is mapped to a *phase space vector* $\boldsymbol{x}_i$ comprised of the original scaled point and the scaled points previous at intervals of τ. When using $d = 2$ we can create the *phase space plot* (the plot of all phase space vectors in $\mathbb{R}^2$, ranging from -1 to $+1$ on each axis) which we then divide into N^2 squares which we denote by g_{ij} for all $i, j \in \{1, \dots, N\}$, of size $\frac{2}{N} \times \frac{2}{N}$, with $N \in \mathbb{Z}^+$. C is the phase space matrix with dimension $N \times N$ and is constructed in such a way that, for each $i, j \in N$, the entry c_{ij} is equal to the number of phase space vectors in X that fall within the square area g_{ij}. We construct P by normalising C such that, for each $i, j \in N$, the element p_{ij} corresponds to the proportion of all of the phase points which fall within g_{ij}. We calculate P as follows:

$$P = \frac{1}{M}C, \quad M = \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}.$$

A typical approach is to extract features from these PSR by either box-counting (Vemishetty et al., 2019), calculating the spatial filling index (Krishnan et al., 2007) or calculating statistics of the distributions of values within each column of C (Rocha et al., 2008). These features can then be used as input to predictive models.

The one-dimensional time series representing the filtered and potentially flipped single-lead ECG signal is then transformed to a two-dimensional Phase Space Reconstruction (PSR) matrix. While high-dimension PSR transformations have been used in the field of BCI (Chen et al., 2014; Djemal et al., 2016), two-dimensional PSR transformations are more commonly used in the literature on ECG analysis (Rocha et al., 2008; Krishnan et al., 2007; Vemishetty et al., 2019; Roberts et al., 2001). As mentioned previously, features are typically extracted from the PSR matrices by either box-counting (Vemishetty et al., 2019), calculating the spatial filling index (Krishnan et al., 2007) or calculating statistics of the distributions of values within each column of C (Rocha et al., 2008). These features are then used as model inputs for classifying various different categories of ECG. Differently, in this work we consider these matrices as $N \times N$ (with $N = 32$) pixel images and rely on deep learning model architectures typically used for computer vision to autonomously perform feature selection.

Figure 4.5 gives an example of a PSR image with $N = 32$ as well as a darkened version of the same image for readability. As one can see in the darkened image, there are some connected bands of higher probability. With values of N greater than 32, these bands become disconnected as no phase space vectors land within that portion of the grid.

4.2.2 MLP and CNN for regression

In this subsection, we discuss the architecture of the deep learning models used to predict T:R ratios from 32x32 pixel PSR images. In this subsection we assume the reader has an understanding of neural network layers. Details on the neural network layers used in this subsection can be found in Subsection 2.2.2. The general structure of each

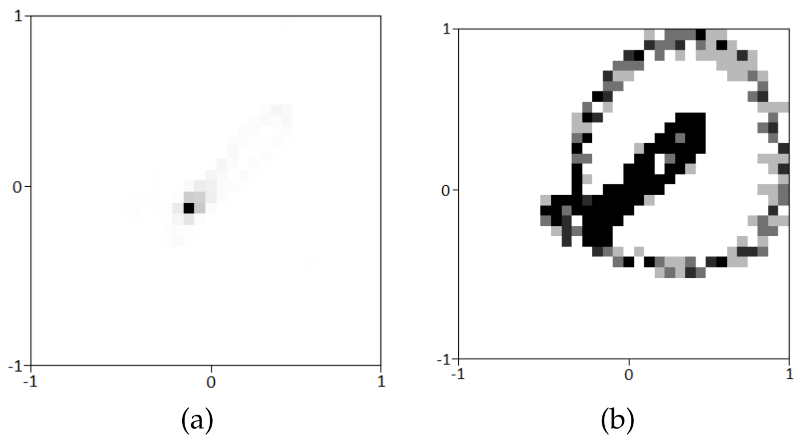

FIGURE 4.5: (a) Image of the PSR matrix formed from a 10-second segment ECG signal with $N = 32$. (b) Darkened image of the same PSR matrix.

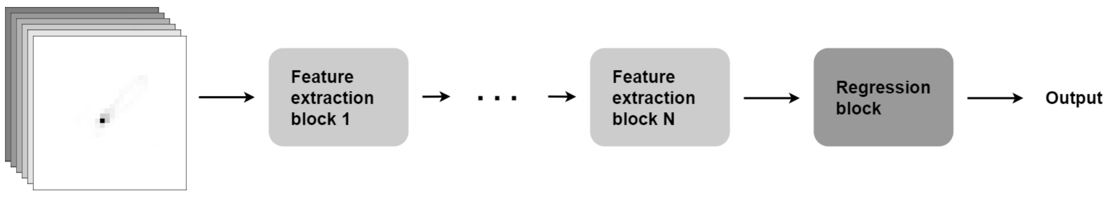

FIGURE 4.6: The general structure of a model which takes PSR images representing 10-second ECG segments as input and outputs predicted T:R ratios.

model is laid out in Figure 4.6. Each model is made up of N feature-extraction blocks which extract abstract features from the PSR images, and pass them on to the regression block.

The regression block is used by all models to derive the T:R ratio from the extracted features. The outputs from the preceding feature-extraction blocks are flattened to a 1D vector and fed into a series of fully connected (dense) layers of neurons to arrive at the final regression output: the T:R ratio. Table 4.1 gives an overview of the layers comprising this block. We use batch normalisation for regularisation as it has been shown to be superior to dropout for use in CNNs (Garbin et al., 2020). We perform batch normalisation before applying the activation function as proposed in the original paper on batch normalisation (Ioffe and Szegedy, 2015).

TABLE 4.1: Regression block

Layer	Type	Output size
1	Dense	256
2	BatchNorm	256
3	Activation(Relu)	256
4	Dense	64
5	BatchNorm	64
6	Activation(Relu)	64
7	Dense	1

The first and most basic of our feature-extraction blocks is the MLP feature-extraction block, comprised of a single layer of fully connected neurons followed by a batch normalisation and activation layer. The input and output of these blocks are 1D. As such, when using these blocks, we flatten the images before the first feature-extraction block rather than before the regression block.

TABLE 4.2: The MLP feature extraction block

Layer	Type	Output size
1	Dense	1024
2	BatchNorm	1024
3	Activation(Relu)	1024

The basic CNN feature-extraction blocks utilise convolutional layers, which exploit the 2D structure of the PSR images, as opposed to fully connected layers. As convolutional layers take three-dimensional (3D) inputs, the original PSR images are considered to have shape $32 \times 32 \times 1$. These layers are followed by the batch normalisation and activation layers mentioned previously and finally a maximum pooling layer to reduce the size of the output images. As shown in Table 4.3, the output size of each layer and the kernel size of the convolutional layer varies across the feature-extraction blocks depending on where in the overall model they lie.

TABLE 4.3: The i^{th} basic CNN feature extraction block

Layer	Type	Output size	Kernel size	Stride
1	Convolutional	$2^{i+4} \times 2^{6-i} \times 2^{6-i}$	$(7-i) \times (7-i)$	1
2	BatchNorm	$2^{i+4} \times 2^{6-i} \times 2^{6-i}$		
3	Activation(Relu)	$2^{i+4} \times 2^{6-i} \times 2^{6-i}$		
4	MaxPooling	$2^{i+4} \times 2^{5-i} \times 2^{5-i}$	2×2	2

The Complex CNN feature extraction block contains two convolutional layers (each followed by batchnorm and activation layers) with relatively small kernels which feed into a convolutional layer with stride 2 and a larger kernel (followed by batchnorm and activation layers). The first two convolutional layers extract features from the PSR images, while the third reduces the size of the feature maps and increases their number. While pooling layers are typically used to decrease the size of the feature maps, using a convolutional layer to do this allows an additional opportunity to extract more complex features from the outputs of the previous layers. Finally, *addition skip connections*, as popularised in ResNet (He et al., 2016) and utilised in other works using CNNs for ECG analysis (Kwon et al., 2021; Zhu et al., 2020; Jo et al., 2021), are added over the first two convolutional layers in order to speed up training.

Table 4.4 gives an overview of the layers of this block and Figure 4.7 shows the first feature extraction block of the Complex CNN5 model. As can be seen in both Table 4.4 and Figure 4.7 each subsequent Complex CNN feature extraction block has twice as many feature maps as its predecessor had and the height and width of these feature maps are half the size of those in the previous Complex CNN feature extraction block.

TABLE 4.4: The i^{th} complex CNN feature extraction block

Layer	Type	Output size	Kernel size	Stride
1	Convolutional	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$	$(6-i) \times (6-i)$	1
2	BatchNorm	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
3	Activation(Relu)	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
4	Convolutional	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$	$(6-i) \times (6-i)$	1
5	BatchNorm	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
6	Activation(Relu)	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
7	Skip(Input)	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
8	Convolutional	$2^{i+3} \times 2^{5-i} \times 2^{5-i}$	$(7-i) \times (7-i)$	2
9	BatchNorm	$2^{i+3} \times 2^{5-i} \times 2^{5-i}$		
10	Activation(Relu)	$2^{i+3} \times 2^{5-i} \times 2^{5-i}$		

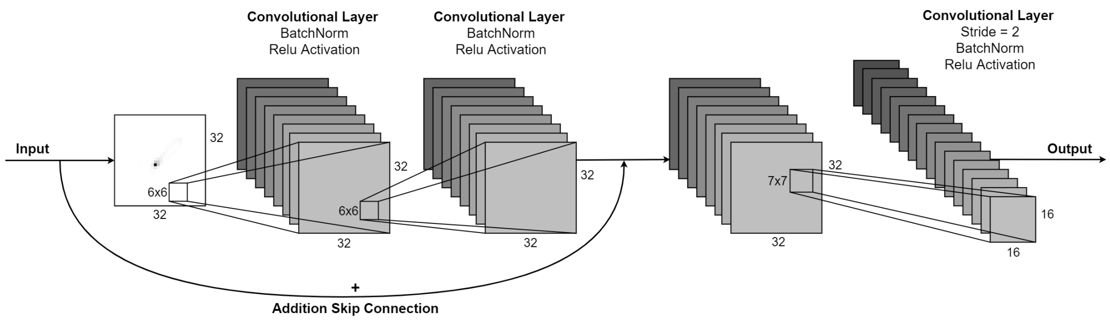

FIGURE 4.7: A diagram of the first feature extraction block of the Complex CNN5 model. The batchnorm and activation layers are not visually represented.

Because of this halving in height and width of the feature maps, the kernel size of the convolutional layers must also decrease.

The Deep CNN feature-extraction block is very similar to the complex CNN feature-extraction block. Before the convolutional layer with stride equal to 2 which is used for pooling, we include an additional pair of convolutional layers with smaller kernels as well as a second skip connection. Table 4.5 gives an overview of the layers of this block. As with the Complex CNN5 feature extraction block, when referring to the model, we give the type of feature-extraction block used followed by the number of feature-extraction blocks. For example, a model comprised of 5 MLP feature-extraction blocks followed by the regression block would be referred to as MLP5 (a diagram of this model architecture is shown in Figure 4.8), while a model comprised of 3 Complex CNN feature-extraction blocks followed by the regression block would be referred to as Complex CNN3.

In our experiments we evaluate the 6 minibatch Stochastic Gradient Descent (SGD) variants laid out in Section 2.2.2. We consider the choice of SGD based algorithm used to train the model parameters as a hyperparameter of the model itself which should be selected optimally. We then look to tune the hyperparameters of these SGD based algorithms themselves, namely, batch size and global learning rate. As the rounds of

TABLE 4.5: The i^{th} deep CNN feature extraction block

Layer	Type	Output size	Kernel size	Stride
1	Convolutional	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$	$(6-i) \times (6-i)$	1
2	BatchNorm	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
3	Activation(Relu)	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
4	Convolutional	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$	$(6-i) \times (6-i)$	1
5	BatchNorm	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
6	Activation(Relu)	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
7	Skip 1(Input)	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
8	Convolutional	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$	$(6-i) \times (6-i)$	1
9	BatchNorm	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
10	Activation(Relu)	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
11	Convolutional	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$	$(6-i) \times (6-i)$	1
12	BatchNorm	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
13	Activation(Relu)	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
14	Skip 2(Skip 1)	$2^{i+2} \times 2^{6-i} \times 2^{6-i}$		
15	Convolutional	$2^{i+3} \times 2^{5-i} \times 2^{5-i}$	$(7-i) \times (7-i)$	2
16	BatchNorm	$2^{i+3} \times 2^{5-i} \times 2^{5-i}$		
17	Activation(Relu)	$2^{i+3} \times 2^{5-i} \times 2^{5-i}$		

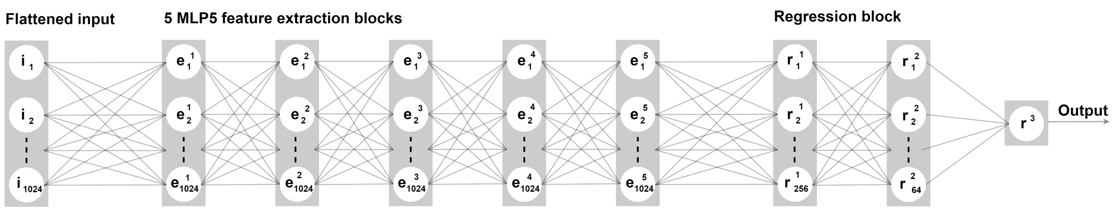

FIGURE 4.8: A diagram of the MLP5 model, consisting of 5 MLP feature extraction blocks and the regression block. The batchnorm and activation layers are not visually represented. The neurons in the flattened input, feature extraction blocks, and regression block are labeled i, e, and r, respectively. Their superscript denotes which layer they belong to, while the subscript refers to their position in that layer.

10-fold CV which we use to asses model performance are very computationally expensive, using grid-search to search a 3D grid is infeasible. We instead tune the hyperparameters sequentially, first evaluating SGD, SGD with Nesterov Momentum, AdaGrad, RMSProp, RMSprop with momentum, and ADAM to determine the SGD based algorithm which gives the best accuracy with default hyperparameter selections. We then tune its batch size, and finally tune its global learning rate while using the batch size found in the previous step.

4.2.3 Model evaluation and enhancement

To evaluate these models, we use 10-fold cross validation. The data is shuffled before being split into 10 equally sized folds. Then, at each iteration of the 10-fold CV, one fold is reserved as the testing set. A randomly sampled 20% of the remaining 9 folds is used to form the validation which is used to determine when early stopping should occur to prevent over-fitting while training. The other 80% of the non-testing data is used as training set [1]. In this chapter, our models are trained for at most 1000 epochs; however, we employ early stopping (Prechelt, 1998) for regularisation in model training using a patience of 200 epochs. During the training of our models we look to minimise the Mean Squared Error (MSE) however, when we come to evaluate the performance of our models in predicting T:R ratios of the testing PSR images there are additional metrics we are interested in. We assess the accuracy of each model using mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE) and STD of errors. Details of these accuracy metrics, as well as early stopping can be found in Subsection 2.2.2 of the preliminaries which focuses on neural networks.

In order to avoid over-fitting, 20% of our training data is randomly reserved for validation. Models are trained until their accuracy on the validation set is no longer increasing. This means that 20% of our original training data is not being used for training. To avoid this, we can iteratively reserve a different 20% of our training data for validation, training 5 *sub-models* with the same architecture but which have all used a different, non-overlapping set of data for validation. In doing so, while some data may be used for validation by one model, it will be used for training by the remaining 4. In contrast, when training a single model, the data points in the validation set are never used for training. By taking the average of these sub-models' predictions, we aim to obtain a higher prediction accuracy than would be archived by any single model. This is illustrated in Figure 4.10. When referring to an ensemble model comprised of 5 sub-models, we simply append the word ensemble onto the name of the sub-model.

Image augmentation can strengthen the training of models by randomly introducing small distortions to the training data, thus resulting in models which are more robust to distortions which may occur naturally in our unaltered data. While there are many image augmentation strategies, only a small number of them are appropriate for the sort of images we are handling. We test 4 image augmentation strategies: shifting, zooming, rotating and shearing (Shorten and Khoshgoftaar, 2019). Figure 4.9 gives an example of how each of these augmentations could affect a PSR image. When implementing image augmentation, a model may be trained on data which, at each mini-batch, is either shifted randomly within a range of $[1 - \phi/100, 1 + \phi/100]$, zoomed randomly within a range of $[1 - \phi/100, 1 + \phi/100]$, rotated randomly by an angle within a range of $[-\phi^\circ, +\phi^\circ]$ or sheared randomly with a shear angle within a range of $[-\phi^\circ, +\phi^\circ]$. We denote the augmentation as either tiny ($\phi = 2.5$), small ($\phi = 5$) or moderate ($\phi = 10$). For instance, MLP5 Small Rotation would refer to a model consisting of 5 MLP feature-extraction blocks followed by a regression block which has been trained on images which have been, at each mini-batch, rotated randomly by an angle within a range of $[-5^\circ, 5^\circ]$.

[1]Figure 4.10 illustrates how ensemble models are trained within the 10-fold CV process. However if we were to consider an ensemble model with but a single sub-model then this figure would also outline the process for training non-ensemble models within the 10-fold CV process.

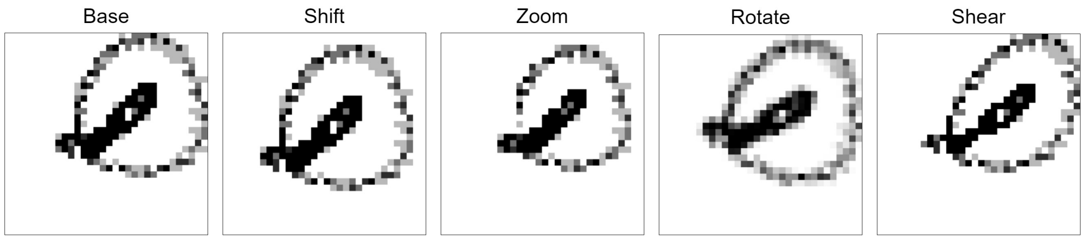

FIGURE 4.9: Examples of how shifting down and to the left by 12.5%, zooming by -12.5%, shearing with an angle of 15° and rotating by 15° effect a PSR image.

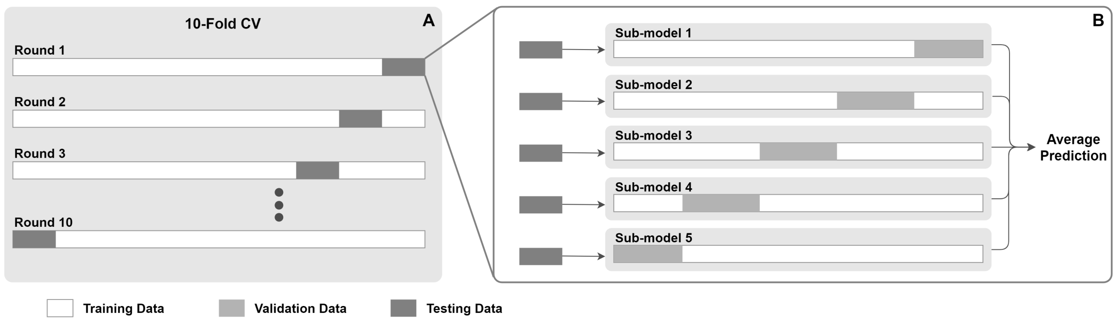

FIGURE 4.10: Part A gives an overview of how training and testing sets are iteratively sampled within a single 10-fold CV. Part B illustrates how within each round of a single 10-fold CV, having reserved a testing set, 5 sub-models are trained with different training and validation sets and their predictions for the testing set are averaged giving the ensemble model's prediction. If we consider an ensemble model with only one sub-model, this becomes exactly the same as the 10-fold CV process for training non-ensemble models, described in Section 4.2.3.

4.3 Experiments

4.3.1 SGH-ECG and ECG-ID data sets

Our aim in this study is to show that sophisticated ML tools can be used to better scrutinise patients' eligibility for S-ICD implantation. We achieve this by constructing deep learning models capable of accurately predicting T:R ratios of patients belonging to a range of patient groups; namely, patients suffering from congestive heart failure, underlying congenital heart disease and "hypertrophic cardiomyopathy", as well as patients with structurally normal hearts. In order to develop such models and ensure their accuracy across all of these patient groups, we have collected our own data set. We collected this data set at Southampton General Hospital as part of a study on S-ICD eligibility and as such we refer to it as the Southampton General Hospital ECG (SGH-ECG) data set. The SGH-ECG data set consists of 390 10-second ECG segments, sampled at 500 Hz, annotated with R and T peaks. These signals were obtained at random intervals from the 24-hours ECG recordings of 18 different participants and were collected as a part of a clinical research study – HEART TWO[2] conducted by the Cardiac rhythm management research department at the University Hospital of Southampton. A detailed break down of the participants demographics and heart conditions can be found in Table 4.6. The participants ages range from 20 to 80 years old with a mean of 53.16 years. Using the preprocessing methods laid out in Subsection 4.2.1, 32x32-pixel PSR images and their corresponding T:R ratios are derived from these 10-second ECG segments. Our aim is to build a model capable of accurately predicting T:R ratios from these PSR images.

Total Number of Participants		n=18	
Demographics:	Mean age [years]	53.16	
	Male	9	(50.00%)
Heart condition:	Structurally normal heart	4	(22.22%)
	Adult congenital heart disease	3	(16.67%)
	Hypertrophic cardiomyopathy	3	(16.67%)
	Congestive heart failure	8	(44.44%)

TABLE 4.6: Detailed breakdown of the demographics and underlying aetiology of patients in the SGH-ECG data set.

In order to increase the amount of training data, at each round of cross validation, after testing and validation sets have been reserved, we bolster our training set by combining it with the 32x32 pixel PSR images and T:R ratios derived from the ECG Identification (ECG-ID) Database. The ECG-ID data set, collected by Lugovaya (Lugovaya, 2005), consists of 310 20-second ECG segments sampled at 500Hz, with R and T peak annotations for the first 10 heartbeats. These signals are obtained from 90 participants, 44 men and 46 women, with ages ranging from 13 to 75. We do not add this data to the testing and validation sets as we are only interested in evaluating the performance of our models for patients with specific heart conditions which the participants of the ECG-ID study did not have.

4.3.2 Experiment 1: broard architecture comparisson

Our first experiment is to assess which model architectures, as detailed in Subsection 4.2.2, most accurately predict T:R ratios from 32x32 pixel PSR images. Figure 4.11 shows the average 10-fold CV accuracy of some of the best performing models using the accuracy metrics laid out in Subsection 4.2.3 and the model naming convention laid out in Subsection 4.2.2. As can be seen from the results in Figure 4.11, the MLP5 model is capable of accurately predicting T:R ratios with a mean absolute prediction error of only 0.0558. We recall that T:R ratios range between 0 and 1 and the threshold for failing a screening is 0.33. In switching to a Basic CNN structure, we see a considerable

[2]This study was performed with favourable opinion from the REC (17/SC/0623) and with R&D (RHM-CAR0528) approval. This study was conducted in accordance with the Research Governance Framework for Health and Social Care (2005), Good Clinical Practice and their relevant updates.

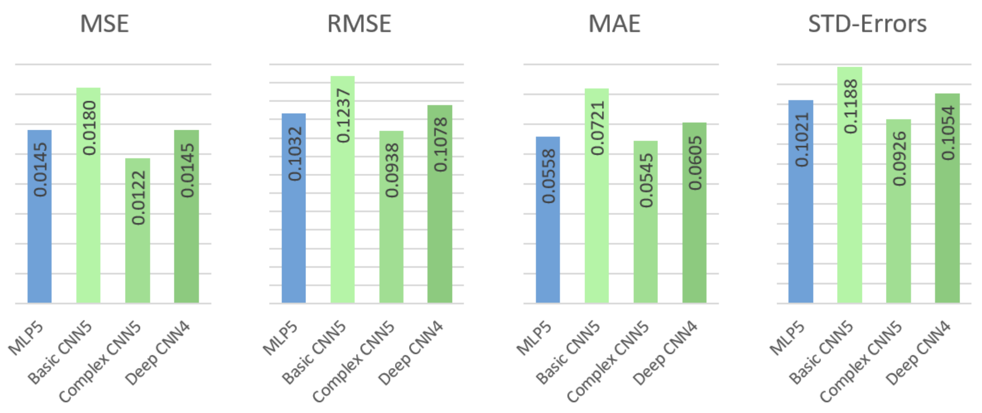

FIGURE 4.11: Performances of the best models using each type of feature extraction block

drop in accuracy. Adding more convolutional layers by using Complex CNN feature-extraction blocks allows us to recoup this loss. The Complex CNN5 model outperforms the MLP5 model for all tested metrics. However, continuing to add convolutional layers by using Deep CNN feature-extraction blocks results in a drop in accuracy for all tested metrics. This is likely due to the fact that our data set is not sufficiently large to enable the training of such deep neural-networks. Figure 4.11 shows the average cross validation accuracies of only the best performing model for each type of feature-extraction block. There are several models with architectures utilising the MLP and Complex CNN feature-extraction blocks with a MAE under 0.06 indicating that, on average, these models are able to predict T:R ratios within 0.06 of their true value.

4.3.3 Experiment 2: evaluation of enhancement methods

In this experiment, we test the effect of image augmentation and creating ensemble models on our two best performing models from the previous experiment: MLP5 and Complex CNN5. We test 4 different image augmentation schemes (shifting, zooming, rotating and shearing) at three different magnitudes (tiny, small and moderate). We also evaluate ensemble models, created by averaging the prediction of 5 sub-models which each use a different portion of the non-testing data and labels for validation. The details of both of these methods are given in Subsection 4.2.3. Neither method for model enhancement had a significant positive impact on the accuracies of the Complex CNN5 model.

For this reason, Figure 4.12 compares the average accuracy across the 10 rounds of cross validation of the enhanced MLP5 models which were able to outperform the base MLP5 model with that of the MLP5 and the Complex CNN5 models. The only two image augmentation schemes which improve the performance of the MLP5 model are small rotations and small shears, which both only result in small increases in some accuracy metrics, along with small decreases in others. Although, individually, rotating

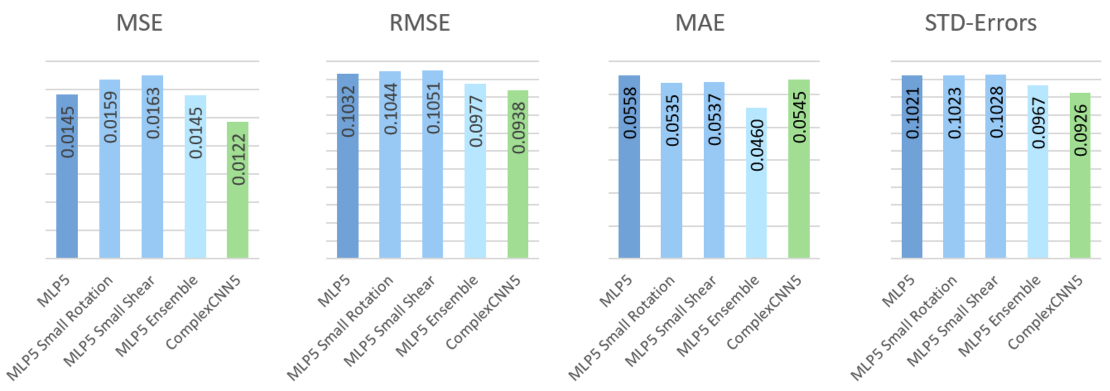

FIGURE 4.12: Comparison between MLP5 model variants and the Complex CNN5 model

and shearing results in improved performance for the MLP5 model, we find that using both of these image augmentation schemes in concert does not result in improved performance. The MLP5 Ensemble model gives modest improvements in performance, outperforming all other variants of the MLP5 model on each metric. For these reasons, in later experiments, having finely tuned the training of our models we will re-evaluate the benefits of creating ensemble models but we will not evaluate image augmentation schemes.

4.3.4 Experiment 3: narrow architecture comparison

Following the results of experiments 1 and 2, we conclude that the the MLP5 and Complex CNN5 model architectures show the most promise and that while image augmentation did not result much improvement in accuracy, creating ensemble models did improve model performance. In this section we perform a more reliable and detailed set of experiments to determine which of our two best performing model architectures results in better models. For each model architecture, we perform 10 rounds of 10-fold CV between each of which the data set is shuffled to ensure that the folds used are different in all 10 rounds of the 10-fold CVs. For the ease of comparison, in these experiments we only consider two of the four accuracy metrics used in the previous experiments, RMSE and MAE. RMSE, while similar to its squared counterpart the MSE, does not see such a broad distribution across the various rounds of CV and, thus, lends itself to a graphical comparison of errors over multiple rounds of CV. MAE is the most intuitive of these errors and, therefore, it is important when discussing the impact of the predictions of a model.

We record the *training time* for each model as, should two models give comparable accuracy, the one with a shorter training time is preferable as it would be less computationally expensive to retrain it if and when new data became available. Similarly, we record the number of epochs the model trained for before early stopping kicked in. The *average prediction time*, i.e., the average amount of time taken for the model to predict each testing PSR image's T:R ratio, is also recorded. A lower average prediction time indicates that predicting T:R ratios in real time would be less computationally expensive and, therefore, the model would be more suitable for incorporation into a device with limited computational resources[3].

In this experiment, we use ADAM as the minibatch SGD variant when training, as "...ADAM works well in practice and compares favorably to other stochastic optimisation methods"(Kingma and Ba, 2014). We use the generally accepted default selections of batch size 32 (Bengio, 2012; Masters and Luschi, 2018) and learning rate 0.001 as suggested in the original paper (Kingma and Ba, 2014). These experiments are conducted using Tensorflow in Python 3.10.

Figure 4.13 comprises 5 violin plots of the distributions of 5 performance metrics (detailed in Subsection 4.2.3) over the 100 total rounds of CV for the MLP5 architecture (shown in blue) and the Complex CNN5 model (shown in red). We use violin plots are to visualise the distributions of the various metrics over the 100 rounds of CV. The width of the violin plot corresponds approximately with the number of observations at that point. These violin plots also contain a box plot displaying some summary statistics of the distributions. The MLP5 model completes its training in, on average, 493 seconds. This is significantly quicker than the time of the Complex CNN5, whose average training time is 1262 seconds despite taking roughly the same number of epochs to complete training. This shorter training time is preferable as it would enable the model to be updated more regularly as new data becomes available.

The MLP5 models are found to be able to predict the testing PSR images' T:R ratios much quicker than the Complex CNN5 models, with an average prediction time per PSR image of 0.0075 seconds compared to an average of 0.0119 seconds for the Complex CNN5 one. This shorter prediction time is valuable, as it indicates that the MLP5 models would require fewer computational resources to make predictions of the T:R ratio in real time than the Complex CNN5 models would.

In our analysis, our primary concern is accuracy. Here, we consider RMSE and MAE as defined in Equation (2.1). The MLP5 models achieve an average RMSE of 0.105 and an average MAE of 0.0567 compared to the Complex CNN5 models' average RMSE of 0.111 and MAE of 0.0624. For these reasons, we conclude that the MLP5 architecture leads to better performance as well as to shorter training and prediction times. In the subsequent experiments, we will look to select the best SGD variant for training these MLP5 models. We will not consider the prediction times in the subsequent analysis, as they should be the same for all models with the same architecture.

[3]At the time of this work, the incorporation of this model into a device with limited computational resources is not of primary concern. However, our work with our partners at the University Hospital of Southampton is ongoing and one area of discussion is using a model, such as this one, within the S-ICD to supplement the algorithm already in use.

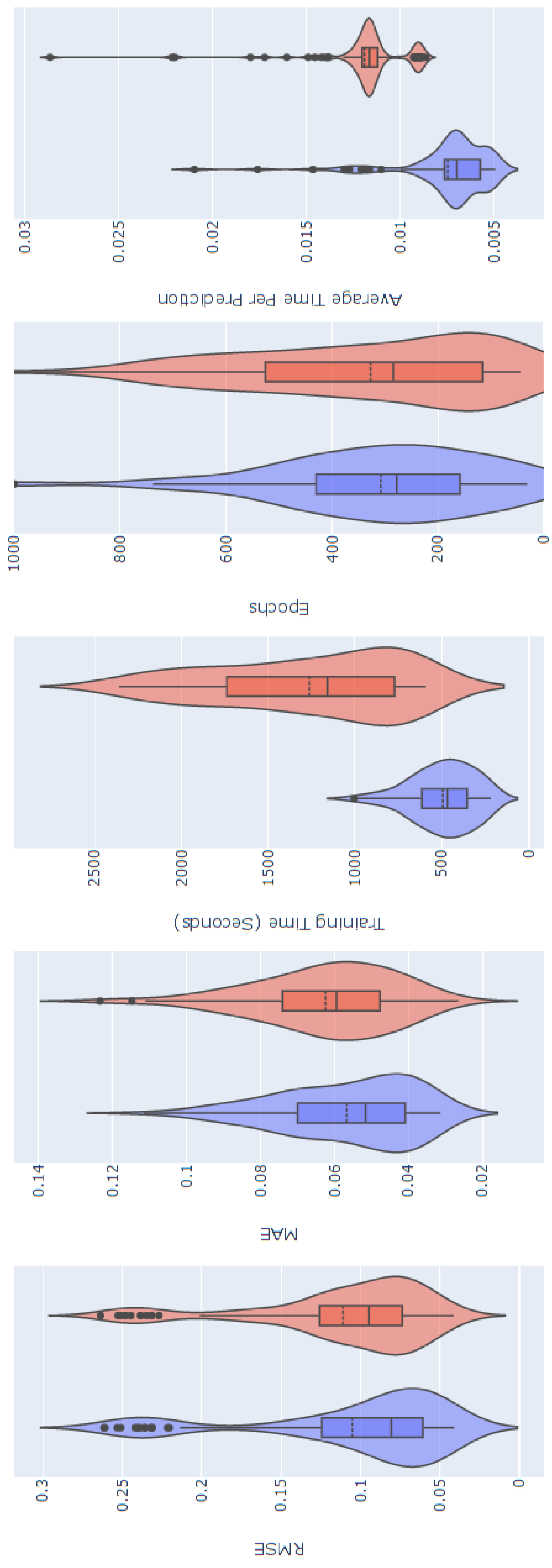

FIGURE 4.13: Violin plots showing the distribution of the RMSE, MAE, training time, epochs, and average prediction time of the 100 MLP5 models (blue) and Complex CNN5 models (red) evaluated in the 10 rounds of 10-fold CV.

4.3.5 Experiment 4: optimiser comparison

In the previous experiment, we assumed ADAM was the preferred minibatch SGD variant. In this experiment, we test this assumption by evaluating 6 minibatch SGD variants: SGD, SGD with Nesterov Momentum, AdaGrad, RMSProp, RMSprop with momentum, and ADAM. At this stage, we still use the default batch size and learning rate of, respectively, 32 and 0.001. We perform 10 rounds of 10-fold CV, in each fold training an MLP5 model with each of the 6 minibatch SGD variants for a total of 100 models trained using each minibatch SGD variant and evaluated on a distinct testing set. It is worth noting here that we do not compare the average prediction time of models with different architectures as, after training, the average prediction time for models with the same architecture should be consistent.

Figure 4.14 shows the distribution of the RMSEs across the 100 models for each minibatch SGD variant (left) as well as an expanded version of the same graph (right). The trend shown in these results is that the prediction error is lower for SGD variants which are better suited to the non-convex setting (such is the case when training a deep neural network for T:R ratio prediction from PSR images). MLP5 models trained using ADAM exhibit an average RMSE of 0.105 compared to those trained using basic minibatch SGD, which have an average RMSE of 0.121. We observe that the training time and the number of epochs before early stopping is generally greater for more complex versions of minibatch SGD such as for ADAM and RMSprop. This can be seen in Figure B.1 in Section B of the Appendix. With this experiment, we have confirmed that ADAM does indeed give the best accuracy (with a default hyperparameter selection) than a number of popular minibatch SGD algorithms and, hence, is the optimisation algorithm we will use going forward.

4.3.6 Experiment 5: tuning batch size

Batch size is a hyperparameter of all minibatch SGD based algorithms. In previous experiments, we used the generally recommended batch size of 32 (Bengio, 2012; Masters and Luschi, 2018). In this experiment, we evaluate batch sizes of 2^i for $i \in \{3, 4, 5, 6, 7, 8\}$ (while still using a learning rate of 0.001) by performing 10 rounds of 10-fold CV, in each fold training an MLP5 model with each of the 6 different batch sizes for a total of 100 models trained with each batch size and evaluated on a distinct testing set.

Figure 4.15 shows the distribution of the RMSEs across the 100 models trained with each batch size (left) as well as an expanded version of the same graph (right). As can be seen, increasing the batch size steadily decreases the RMSE up to a batch size of 128, which achieves an average RMSE of 0.1029, compared to the previously used batch size of 32 which leads to an average RMSE of 0.1053.

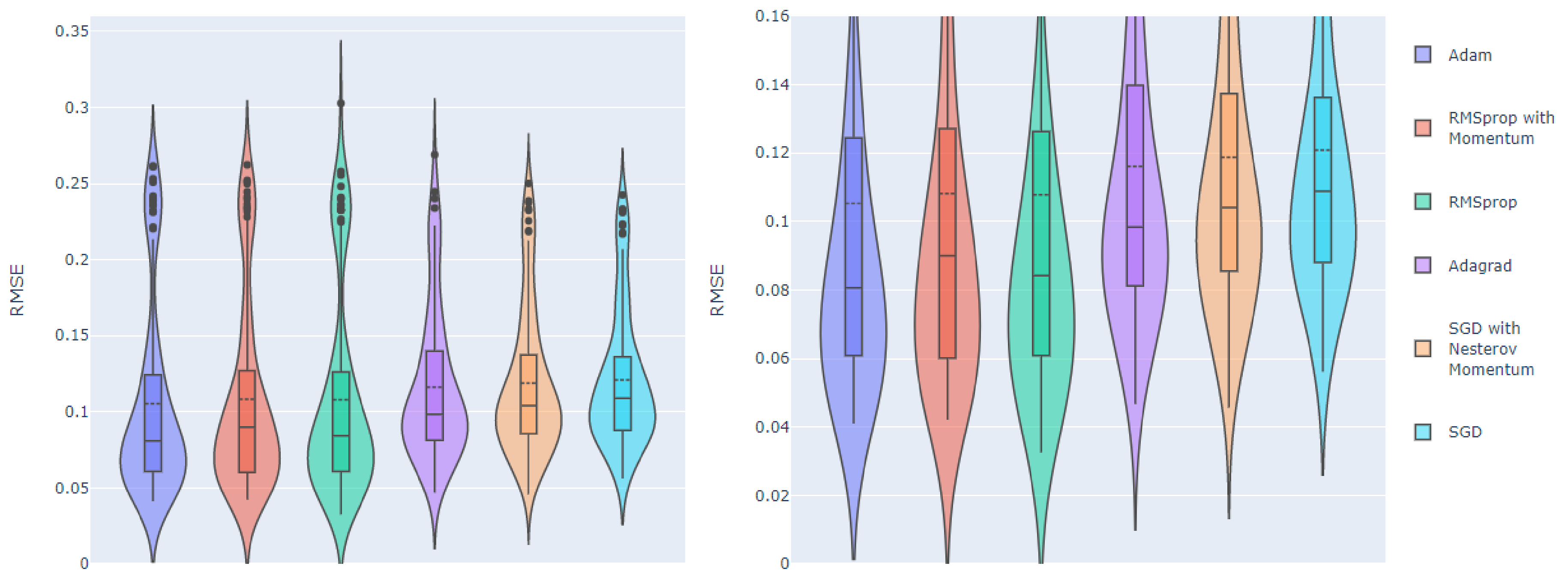

FIGURE 4.14: Left: Violin plots showing the distribution of the RMSEs of the 100 MLP5 models trained with each SGD variant evaluated in the 10 rounds of 10-fold CV. Right: The same graph resized for readability.

Figure B.2 in Section B of the Appendix shows that, while the number of epochs before earlystopping increases as the batch size increases, the training time decreases. In one epoch, minibatches are propagated forward though the network and their errors are used to calculate gradients and update the parameters. This is repeated until all data has been seen at least once. This means that a large batch size results in the parameters being updated far fewer times per epoch. This is the reason why, in Figure B.2, we see that the models with a large batch size typically take more epochs to train as each epoch represents fewer descent steps. However, despite training for more epochs, the large reduction in the number of times the model parameters are updated results in a drastic reduction in training time for models trained with larger batch sizes.

Using a batch size of 128 leads to the best average RMSE of 0.1029 while also resulting in a significant reduction in average training time compared to using a batch size 32 and, as such, 128 is the batch size we will use going forward.

4.3.7 Experiment 6: tuning learning rate

ADAM is considered robust to the choice of hyperparameters ρ_1 and ρ_2 and, as such, we only tune the global learning rate ϵ by searching a logarithmic range 10^{-i} for $i \in \{1, 2, 3, 4, 5, 6\}$ while using the best-performing batch size of 128 found in the previous experiment (Bengio et al., 2017). We evaluate these learning rates by carrying out 10 rounds of 10-fold CV, in each fold training an MLP5 model with each of the different learning rates for a total of 100 models trained with each learning rate and evaluated on a distinct testing set.

Figure 4.15 shows the distribution of the RMSEs across the 100 models trained with each learning rate (left) as well as an expanded version of the same graph (right). Figure 4.15 does not show the results for models trained with a learning rate of 10^{-5} and 10^{-6}. However, the trend demonstrated in Figure 4.15 (that, for learning rates less than 0.01, the average RMSE increases as the learning rate decreases) continues for these values. We observe that using a learning rate of 0.01 gives an average RMSE of 0.1020 as opposed to the RMSE of 0.1029 given by the previously used default of 0.001. Figure B.3 in Section B of the Appendix shows that, as we would expect, the number of epochs increases for smaller learning rates and, as the batch size is fixed at 128 in this experiment, the training time is proportional to the number of epochs and so it also increases as the learning rate becomes smaller.

We conclude from this experiment that a MLP5 model trained using ADAM with a batch size of 128 and a learning rate of 0.01 offers improved accuracy while taking less time to train. Figure 4.17 shows the average training and validation error at each epoch of each of the 100 models trained using ADAM with a batch size of 128 and a learning rate of 0.01 in each of the 100 rounds of CV performed in this experiment. As can be

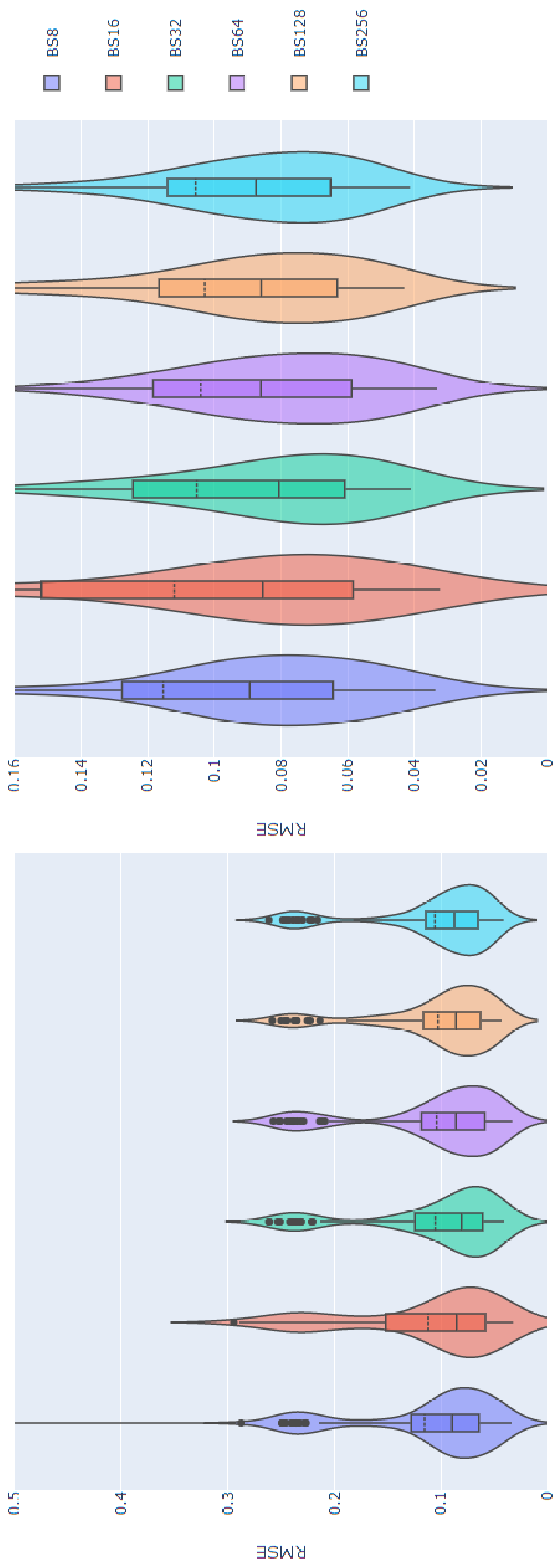

FIGURE 4.15: Left: Violin plots showing the distribution of the RMSEs of the 100 MLP5 models trained with each batch size evaluated in the 10 rounds of 10-fold CV. Right: The same graph resized for readability.

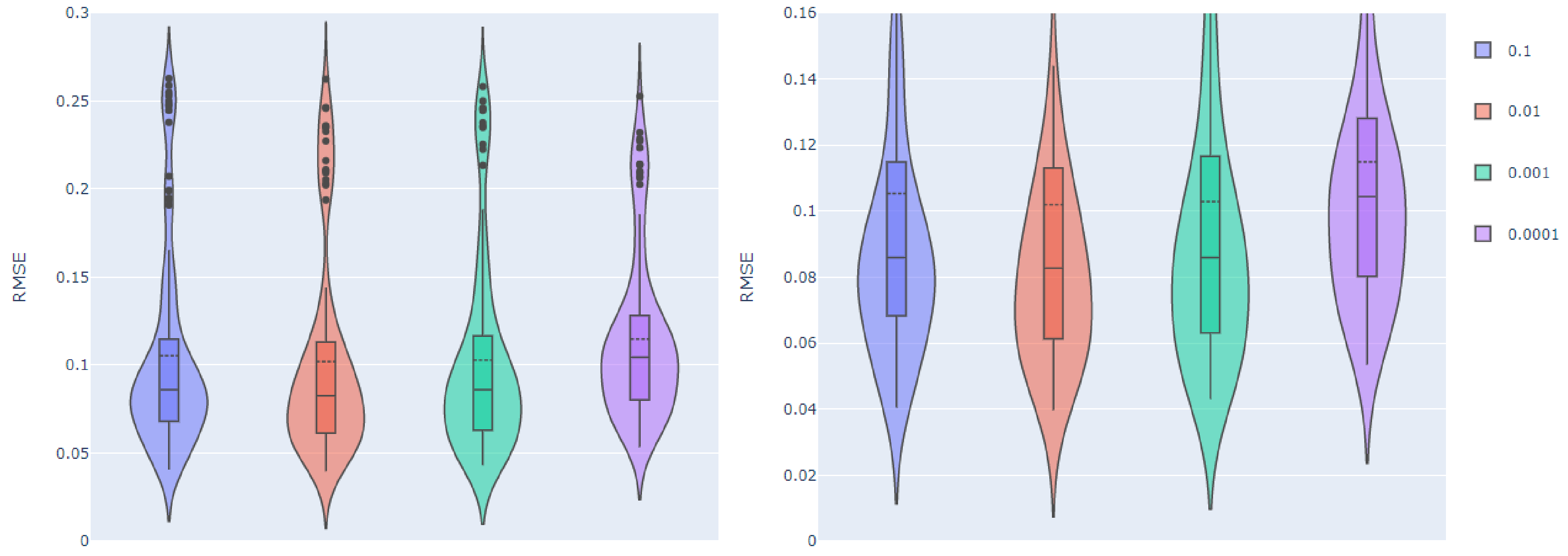

FIGURE 4.16: Left: Violin plots showing the distribution of the RMSEs of the 100 MLP5 models trained with each global learning rate evaluated in the 10 rounds of 10-fold CV. Right: The same graph resized for readability.

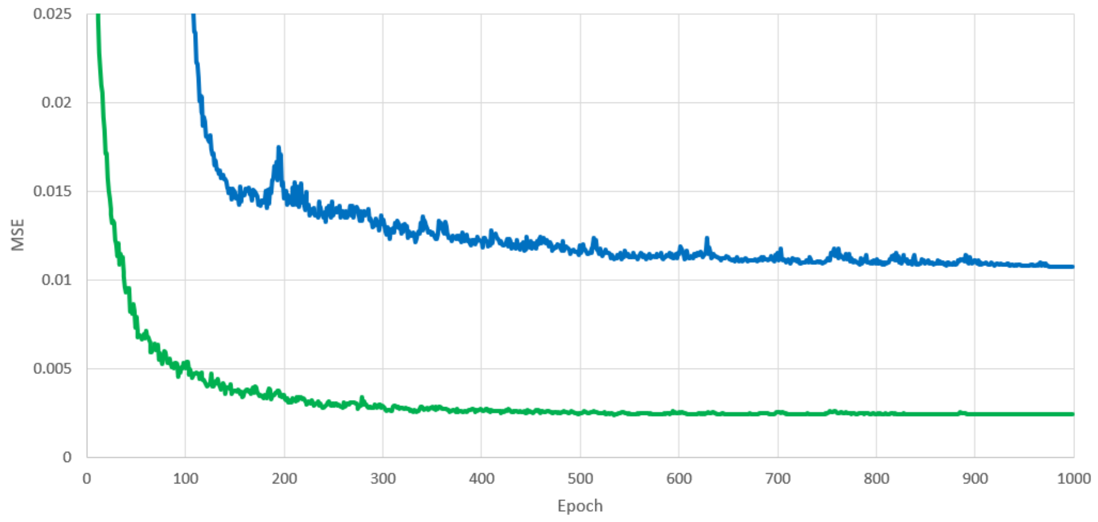

FIGURE 4.17: The average training and validation loss over the 100 CV rounds over 1000 epochs.

seen, while the validation error is higher than the training error during training, both decrease rapidly before stabilising and reduce slowly over the rest of the training. For the purposes of this plot where early stopping occurs, the errors are kept constant for the remainder of the 1000 epochs.

4.3.8 Experiment 7: creating ensemble models

In the previous experiments, we found that a *tuned* MLP5 model trained using ADAM with a batch size of 128 and a global learning rate of 0.01 gave the best accuracy. In this experiment, we assess the benefits of ensemble models by performing 10 rounds of 10-fold CV and, in each fold, training 5 MLP5 submodels, for a total of 100 ensemble models each containing 5 tuned MLP5 submodels. We compare each of these MLP5 ensemble models to a tuned MLP5 model and to a *default* MLP5 model trained using ADAM with a batch size of 32 and a learning rate of 0.001 (the naive hyperparameter selections we initially considered to be good defaults).

Figure 4.15 shows the distribution of the RMSEs of the 100 MLP5 ensemble models on their respective testing sets, compared with those of the tuned MLP5 models and the default MLP5 models. As can be seen, the default MLP5 models achieve an average RMSE of 0.1052 while the tuned MLP5 models average 0.1024 and the ensemble MLP5 models average 0.0928. This represents a reduction in average RMSE of 12% for ensemble MLP5 models but only 2% for the tuned MLP5 models. If, however, we examine the upper quartile of these distributions, we see that both the ensemble MLP5 models and the tuned MLP5 models offer a more significant improvement over the base MLP5 models. The ensemble MLP5 models' RMSE distribution has an upper quartile of 0.1002 compared to the tuned MLP5 models' 0.1147 and to the base MLP5 models'

0.1259, which is a reduction of 9% for the tuned models and 20% for the ensemble models. We do not examine the number of epochs or the training duration for this experiment as the number of epochs used to train each submodel is expected to be the same as for the tuned MLP5 models and, likewise, the training time for the whole ensemble MLP5 model is expected to be simply 5 times that of a single tuned MLP5 model.

To this point, we have focused on RMSE because it makes the violin plots we have used more easily interpretable than MSE, and it provides a stronger penalisation to large prediction errors than MAE. However, in order to put the performance of our model in context, we will once again examine MAE, as it as easy to interpret as the average absolute error across each prediction. The default MLP5 models achieve an average MAE of 0.0567 compared to our best performing model, the ensemble MLP5 model, which achieves an average MAE of 0.0461. This represents a substantial reduction in average absolute prediction error (MAE) of 19%.

4.4 Screening tool

In the previous section we showed that our optimised models are capable of predicting T:R ratio with an average error of only 0.0461 and with an average RMSE of 0.0928. In this section, we discuss the clinical impact of our model's ability to predict the T:R ratios of 10-second, single-lead, ECG segments. Having established that this model can be used to accurately and autonomously predict T:R ratios from 10-second ECG segments, we can now use it to efficiently perform screening to assess the normal variations in the patients' T:R ratios across multiple leads over a 24-hour period. Each single-lead ECG recording, created by breaking down the patient's 24-hour, three-lead ECG recording, would be further broken into 8640 non-overlapping 10-second segments which would then be processed using the methodology detailed in Subsection 4.2.1, resulting in 8640 PSR images for each lead, which are ordered chronologically.

For each lead, a time series of T:R ratios can be generated by inputting the PSR images, representing each 10-second segment of the 24-hour, single-lead recording, into the model. As described in Subsection 4.2.1, from a clinical perspective the sign of the T:R ratio is irrelevant. Depending on the sign of the T wave, our model is expected to output both positive and negative predicted T:R ratios. For use in clinical analysis, we simply compute the absolute value of the model outputs. Figure 4.19 shows the predicted absolute T:R ratios for each PSR image (notice that, during each round of 10-fold CV, each PSR image is used for testing exactly once) generated using the SGH-ECG data set when it was reserved for testing in 10-fold CV, plotted against their true absolute T:R ratios. The green line represents a perfect labeling. As can be seen, the absolute value of the outputs of the model predict the true absolute T:R ratios with very low error.

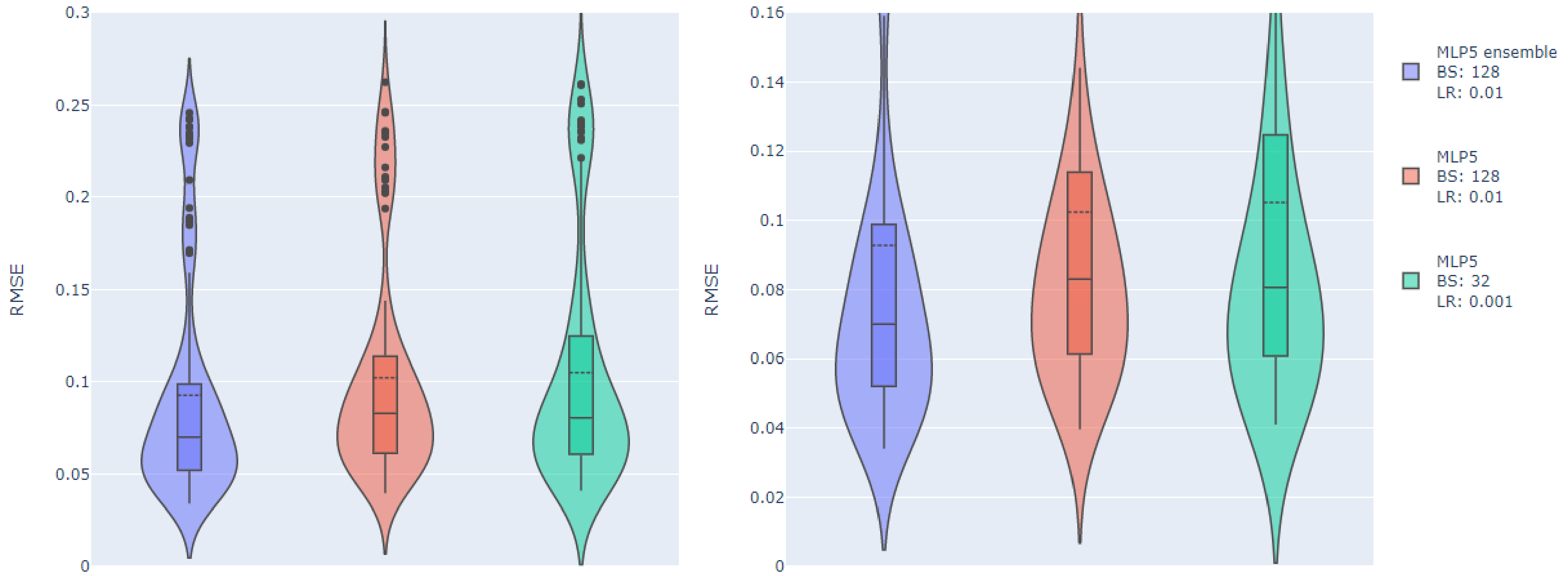

FIGURE 4.18: Left: Violin plots showing the distribution of the RMSEs of the 100 ensemble MLP5 models (blue), tuned MLP5 models (red), and default MLP5 models (red) evaluated in the 10 rounds of 10-fold CV. Right: The same graph resealed for readability.

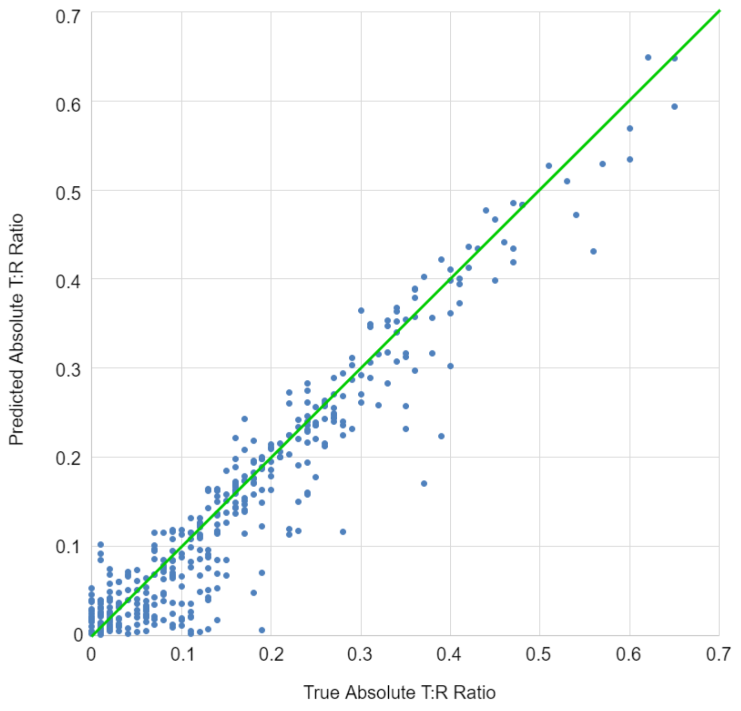

FIGURE 4.19: Graph of the absolute value of true T:R ratios for the SGH-ECG data set plotted against our model's predicted T:R ratios during testing. For readability, 5 outliers have been removed.

The primary aim of the screening is to determine if any of the leads are at a low enough risk of TWOS to be used by an S-ICD. As stated in Section 1.4, for a given lead, should a patient have a T:R ratio above 1:3 for a continuous period of at least 20 seconds, that lead would fail the screening. This means that, for each lead, if our model predicts a T:R ratio of over 0.33 for two or more consecutive 10-second segments, then that lead has failed the screening.

In the event that multiple leads pass the screening, the secondary aim of the screening is to determine which of the leads that pass are at the lowest risk of TWOS. To examine how the behavior of the T:R ratio differs between each lead, we may wish to plot a histogram of what proportion of the 24-hour screening period the T:R ratio of a particular lead spent in each range of T:R ratios. To give an example of how our tool is capable of performing this task, Figure 4.20 shows the histogram of the T:R ratios predicted by the model as well as the histogram of true T:R ratios for PSR images reserved for testing in one of the rounds of cross validation. As one can see, our model is able to predict the proportion of PSR images in the testing set whose T:R ratio belongs to each range to within 2.5% of the true value. Figure 4.21 contains three histograms showing the proportion of the 24-hour screening that each lead spent with a predicted T:R ratio in each range. This would enable the cardiologist to assess which of the leads that passed the screening spends the smallest proportion of the 24-hour screening with high values of T:R ratio and, as such, is at the lowest risk of TWOS.

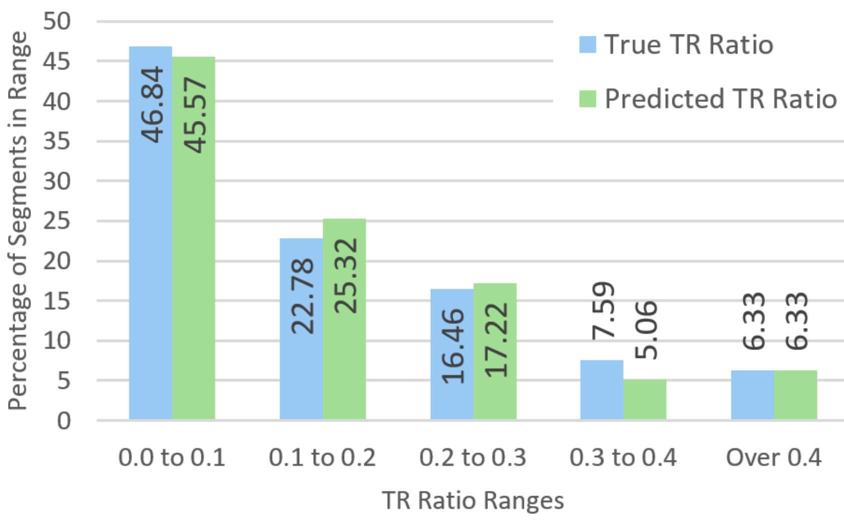

FIGURE 4.20: Histogram of true and predicted TR ratios of 10-second ECG segments in the testing set.

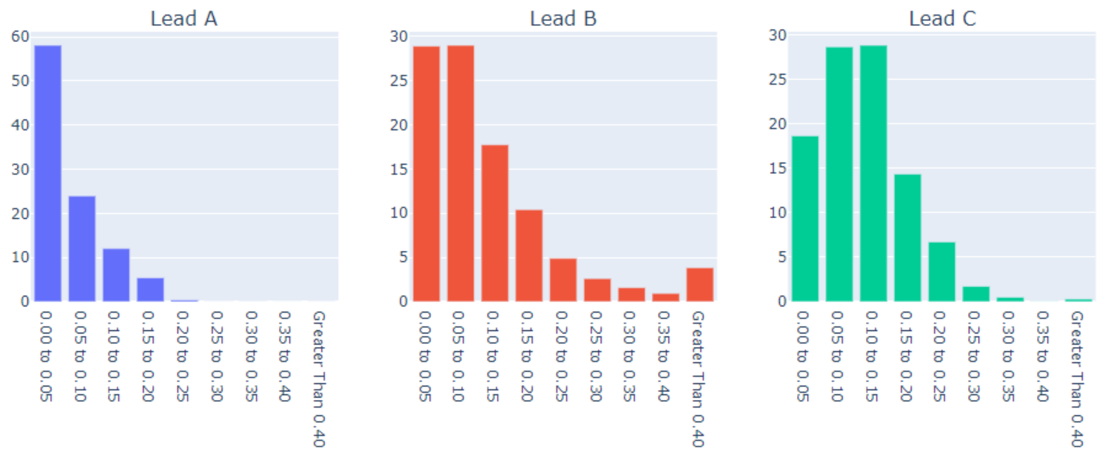

FIGURE 4.21: Histogram of the predicted TR ratio over the 24-hour screening preriod for each lead

Additionally, while less directly applicable to the screening, our model allows for further analysis of the variation of the T:R ratio over the 24-hour screening. Figure 4.22 shows one tool that our model facilitates. The variation of the T:R ratio is plotted for each lead, over the 24-hour period. For readability, the lines in this graph are smoothed in such a way that each point gives the average T:R ratio for the preceding half hour. This could enable a cardiologist to detect any period during the 24 hours where the T:R ratio was consistently high and, as such, the patient was at greater risk of TWOS. Our model could also allow cardiologists to further examine any single 10-second segment from within the 24-hour screening period across all 3 leads and view the ECG signal alongside its predicted T:R ratio. This is shown in Figure 4.23, where, for a single 10-second segment, for each lead, the predicted T:R ratio is given alongside a plot of the original ECG signal, a plot of the filtered ECG signal and the PSR image of that filtered ECG signal.

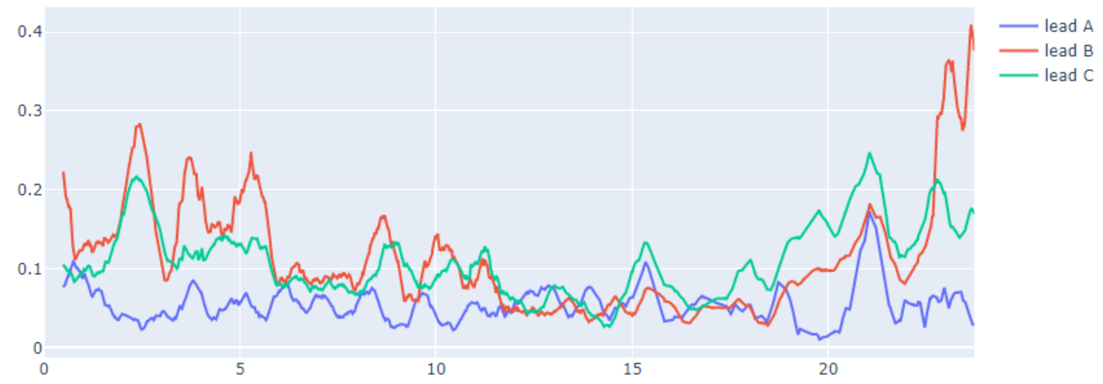

FIGURE 4.22: Graph of the variation of the predicted T:R ratio over the 24-hour screening preriod for each lead

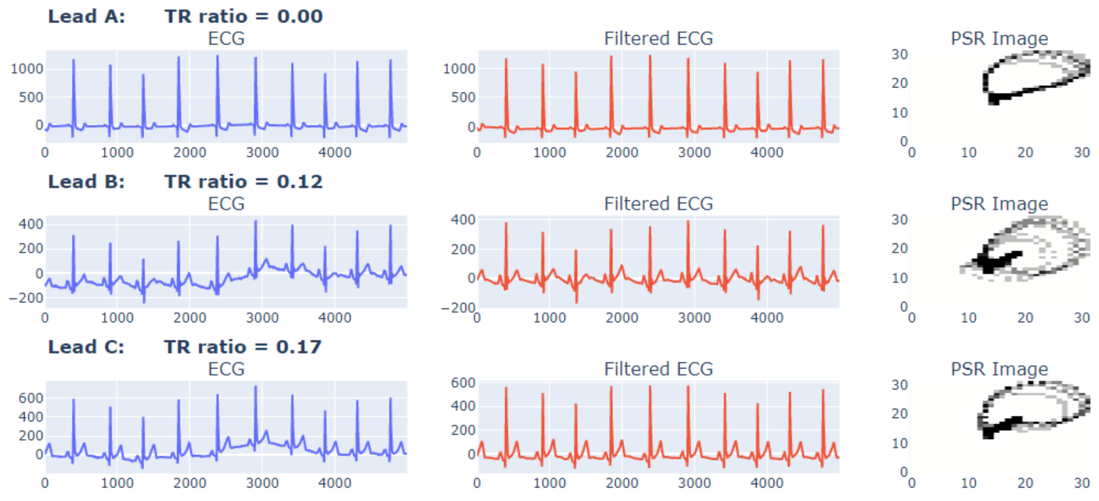

FIGURE 4.23: Visualisations of data from a single 10-second segment of the 24 hour ECG recording

4.5 Conclusions

TWOS is an inherent risk with S-ICDs and can lead to inappropriate shocks. The current method for determining if a patient is likely to suffer from TWOS is to perform a 10-second screening of the patient's ECG, where the T:R ratio, a major predictor of TWOS, is examined. Due to the fact that the T:R ratio can vary significantly over time, the current screening process of using a 10-second ECG recording to estimate a patient's typical T:R ratio does not reliably capture the normal behavior of the patient's T:R ratio.

In this chapter, we have collected the SGH-ECG data set (detailed in section 4.3.1) consisting of 10-second ECG segments from patients belonging to a range of patient groups which S-ICD implantation candidates are likely to belong to. As shown in Section

4.3, we have used this data set to train and evaluate a range of deep learning models capable of predicting T:R ratios from 10-second ECG segments with a high degree of accuracy. In our experiments we have found the MLP5 architecture to lead to the best performing models for predicting T:R ratios from 32x32 pixel PSR images derived from 10-second ECG segments. Creating ensemble models comprised of MLP5 models, trained using the best-performing SGD based optimisation method and with a tuned batch size and global learning rate, we were able to significantly reduce the prediction error compared to the models which used default selections for these hyperparameters. We were able to reduce the average MAE across the 100 CV rounds to 0.0461, a reduction of 19%, and the mean and upper quartile of the distribution of RMSEs over the 100 CV rounds by 12% and 20% respectively.

Ours is a novel approach to ECG analysis. While deep learning and PSR transformations have both individually been used in ECG analysis, to the best of our knowledge, this is the first work using them both in conjunction. We have also shown that this model can be integrated into a clinical tool for performing automated screening over much longer periods than the 10 second window currently used. For example, this tool would enable clinicians to perform 24-hours screenings. This prolonged screening period would enable one to much more reliably determine the normal range of a potential implantation candidate's T:R ratio than the current 10-second screening, as well as providing insight into the variation of the T:R ratio over the screening period. The increased reliability and descriptiveness of this tool can allow cardiologists to better assess patients' risk of TWOS and hence their eligibility for S-ICD implantation.

Currently, we are using our proposed screening tool to assess the temporal variations of the T:R ratio in patients belonging to patient groups who are likely to be implanted an S-ICDs. We are also investigating whether the increased detail provided by our tool can allow the cut-off threshold for the screening to be increased above the current value of 1/3, allowing patients who had, under the previous screening process, been determined to be at too high a risk of TWOS to be safely implanted with S-ICDs. We hope to investigate in future research if fluctuations in a patients' T:R ratios can provide indication that a cardiac episode is impending. Finally, we are looking to formally conduct clinical trials to determine if the methods presented in this chapter can be used in place of the current screening procedure.

Chapter 5

Bilevel hyperparameter optimisation for non-linear support vector machines

In this chapter we address the problem of hyperparameter selection for radial basis function (RBF) kernel support vector machines (SVMs) via cross validation using bilevel optimisation which is set out in Subsection 1.3. Solutions to this problem are typically approximated using methods such as grid-search. However, there have been numerous attempts to solve the problem as a bilevel optimisation problem. An overview of works in bilevel optimisation for hyperparameter selection can be found in Subsection 2.3. All of these works however have focused on linear kernel SVMs and, as such, have all used the primal form of the training problem. In this chapter we focus on the use of the RBF kernel to be able to address problems where linear kernels are not applicable as it will be clear in the next section.

In Section 5.2 we formalise the dual form of the RBF kernel SVM training problem which constitutes our lower level problem, construct the upper level loss function using dual parameters and formulate the bilevel optimisation problem of RBF kernel SVM hyperparameter optimisation via k-fold CV. In Section 5.3 we provide the Karush-Kuhn-Tucker (KKT) reformulation of this bilevel optimisation problem and introduce the C-stationary concept, representing the optimality conditions-type that the Scholtes relaxation algorithm (to be studied in this chapter) converges to. In fact, note that the Scholtes relaxation method will be introduced in Subsection 5.4, as well as some illustrative examples of this algorithm in action on real data sets. The work in this Chapter has led to a paper, which is under preparation (note that this paper will be a significant extension of the work contained in this chapter):

Stefano Coniglio, Anthony J Dunn, Qingna Li, and Alain B Zemkoho, Bilevel hyperparameter optimisation for non-linear support vector machines. To be submitted to Mathematical Programming *by the end of* 2022.

5.1 Support vector machines

Support Vector Machines (SVMs) (Cortes and Vapnik, 1995) are a maximum margin classifier for binary classification. The most simple version is the hard-margin linear SVM, for which, the training problem is to find the hyperplane which separates the two classes and has the largest margin around the hyperplane with no data points within it. However, in practice, it is rarely the case that the two classes are lineally separable. To accommodate for this the soft-margin SVM is used.

The goal of a soft-margin SVM is to find the hyperplane to separate the data, however here, points are allowed to be on the wrong side of the hyperplane. Each data point X_i has a corresponding *slack variable* ξ_i which has value 0 if the point is on the correct side of the hyperplane and outside of the margin, has value 0.5 if the point lies on the hyperplane, and increases lineally the further from being on the correct side of the hyperplane and outside of the margin it is. The optimal hyperplane therefore is one which maximises the margin size while minimising the sum of the slack variables. The optimisation problem for training a soft-margin SVM is given by

$$\begin{aligned} \min_{\omega,b,\xi} \quad & \frac{1}{2}\omega^T\omega + C\sum_{i\in[n]}\xi_i \\ \text{s.t.} \quad & y_i(\omega^T X_i + b) \geq 1 - \xi_i \qquad \forall i \in [n], \\ & \xi_i \geq 0 \qquad \forall i \in [n]. \end{aligned} \tag{5.1}$$

The hyperplane is defined by a vector ω and intercept b. The objective function is comprised of two terms: the inverse of the size of the margin $\frac{1}{2}\omega^T\omega$ and the sum of the slack variables. The hyperparameter C controls the relative contribution of each. The linear SVM can only find linear relationships between features of the data. In order to allow SVMs to find non-linear hyperplanes the kernel trick was proposed (Boser et al., 1992), wherein each data point X_i is embedded into a higher dimensional *feature space* using $\phi : \mathbb{R}^d \to \mathbb{R}^{d^*}$ where $d^* > d$. The optimisation problem of training a SVM using the kernel trick is as follows:

$$\begin{aligned} \min_{\omega,b,\xi} \quad & \frac{1}{2}\omega^T\omega + C\sum_{i\in[n]}\xi_i \\ \text{s.t.} \quad & y_i(\omega^T \phi(X_i) + b) \geq 1 - \xi_i \qquad \forall i \in [n], \\ & \xi_i \geq 0 \qquad \forall i \in [n]. \end{aligned} \tag{5.2}$$

Here, ϕ is defined by the *kernel function*

$$K(X_i, X_j) = \phi(X_i)^T \phi(X_j), \tag{5.3}$$

which outputs the relationship between two points which have been embedded into the feature space. We consider the Radial Basis Function (RBF) kernel defined by

$$K(X_i, X_j) = \exp(-\gamma \|X_i - X_j\|^2).$$

The RBF kernel is recommended for practical applications (Prajapati and Patle, 2010) and has been shown to outperform other kernels such as the polynomial kernel and sigmoid kernel in numerous problem classes (Nanda et al., 2018; Yekkehkhany et al., 2014; Feizizadeh et al., 2017; Hong et al., 2017; Tbarki et al., 2016; Garrett et al., 2003).

Previous works on solving the bilevel problem of selecting the hyperparameters of SVMs (Bennett et al., 2008, 2006; Moore et al., 2009; Kunapuli et al., 2008; Li et al., 2021) all consider only the linear kernel SVM. We will see later that this is not suitable for many applications. We propose a more general formulation of the problem, which will allow us to consider nonlinear kernels. However, the RBF kernel function ϕ has infinite dimensionality. Hence, we consider the dual problem, which does not contain ϕ. To proceed, we introduce $\vartheta = (C, \gamma, w, b, \xi, \alpha, \eta, X, y)$ and hence, the Lagrangian function of problem (5.2) as follows:

$$L(\vartheta) := \frac{1}{2}\omega^T\omega + C\sum_{i\in[n]}\xi_i + \sum_{i\in[n]}\alpha_i(1 - \xi_i - y_i(\omega^\top\phi(X_i) + b)) - \sum_{i\in[n]}\eta_i\xi_i,$$

where α and η are vectors of Lagrangian multipliers. Hence, the Lagrangian dual of problem (5.2) can be written as

$$\max_{\alpha,\,\eta\geq 0}\left\{\min_{\omega,\,b,\,\xi\geq 0} L(\vartheta)\right\}. \tag{5.4}$$

Considering a stationary point ϑ of the Lagrangian function, we have

$$\frac{\partial L}{\partial w}(\vartheta) = 0 \iff \omega - \sum_{i\in[n]}\alpha_i y_i \phi(X_i) = 0, \tag{5.5}$$

$$\frac{\partial L}{\partial b}(\vartheta) = 0 \iff \sum_{i\in[n]}\alpha_i y_i = 0, \tag{5.6}$$

$$\frac{\partial L}{\partial \xi_i}(\vartheta) = 0 \iff C - \alpha_i - \eta_i = 0 \quad \forall i \in [n]. \tag{5.7}$$

Substituting ω out via (5.5) and factoring by ξ_i, the Lagrangian becomes

$$L(\vartheta) = \frac{1}{2}\sum_{i\in[n]}\sum_{j\in[n]} \alpha_i\alpha_j y_i y_j \phi(X_i)^\top \phi(X_j) + \sum_{i\in[n]}(C-\alpha_i-\eta_i)\xi_i + \sum_{i\in[n]}\alpha_i +$$
$$-\sum_{i\in[n]}\left(\alpha_i y_i \Big(\sum_{j\in[n]}\alpha_j y_j \phi(X_j)^\top\Big)\phi(X_i)\right) + \sum_{i\in[n]}\alpha_i y_i b.$$

The term $\sum_{i\in[n]}(C-\alpha_i-\eta_i)\xi_i$ vanishes thanks to Equation (5.7). The term $\sum_{i\in[n]}\alpha_i y_i b$ vanishes due to Equation (5.6). Considering these substitutions, the expression of the Lagrangian function can simply be written as

$$L(C,\gamma,\alpha,X,y) = \sum_{i\in[n]}\alpha_i - \frac{1}{2}\sum_{i\in[n]}\sum_{j\in[n]}\alpha_i\alpha_j y_i y_j \phi(X_i)^\top\phi(X_j).$$

As $\eta \geq 0$, Equation (5.7) implies $C-\alpha_i = \eta_i \geq 0$, i.e., $\alpha_i \leq C$ for all $i \in [n]$, changing the sign of L and rewriting $\phi(X_i)^\top\phi(X_j)$ as $K(X_i, X_j)$, the Lagrangian dual problem (5.4) can be equivalently written as

$$\begin{aligned}
\min_{\alpha}\quad & \frac{1}{2}\sum_{i\in[n]}\sum_{j\in[n]}\alpha_i\alpha_j y_i y_j \exp(-\gamma\|X_i - X_j\|^2) - \sum_{i\in[n]}\alpha_i \\
\text{s.t.}\quad & 0 \leq \alpha_i \leq C \quad \forall i \in [n], \\
& \sum_{i=1}^{n}\alpha_i y_i = 0.
\end{aligned}$$

It is important to note that in this problem, C and γ are hyperparameters that are necessary before the problem can be solved. In the context of linear SVMs, we typically have the regularisation parameter C as the only hyperparameter. However, as it can be seen from (5.3), the RBF function introduces the second hyperparameter γ.

Recall that it is common in machine learning to use grid-search or Bayesian optimisation to compute these hyperparameters. But with the limitations listed in the introduction of this dissertation, bilevel optimisation has been used in the case of linear SVMs (5.1). Below, we present the first formulation of hyperparameter optimisation tailored to non-linear SVMs.

5.2 Bilevel hyperparameter optimisation formulation for the non-linear kernel SVM problem

In this section we construct the bilevel optimisation problem of RBF kernel SVM hyperparameter optimisation via k-fold CV. Firstly, this involves formalising the dual form of the lower level training problem in order to incorporate the RBF kernel. Secondly we construct a smooth loss function for use in our upper level objective which utilises the dual model parameters. Importantly, we note here that the labels y have values of either 1 or -1. Based on the discussion in the previous section, observe that the dual from of the optimisation problem of training a RBF kernel SVM on a training set of $\hat{n}$ data points $\widehat{X}$ and corresponding labels $\hat{y}$ can be written as

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{i \in [\hat{n}]} \sum_{j \in [\hat{n}]} \alpha_i \alpha_j \hat{y}_i \hat{y}_j \exp(-\gamma \|\widehat{X}_i - \widehat{X}_j\|^2) - \sum_{i \in [\hat{n}]} \alpha_i \\ \text{s.t.} \quad & 0 \le \alpha_i \le C \quad \forall i \in [\hat{n}], \\ & \sum_{i=1}^{\hat{n}} \alpha_i \hat{y}_i = 0. \end{aligned} \tag{5.8}$$

We can now define a bilevel optimisation problem where, in the lower level the SVM model is trained to fit a training set of $\hat{n}$ data points $\widehat{X}$ and labels $\hat{y}$ and in the upper level hyperparameters C and γ are tuned such that the performance of the SVM model defined in the lower level on the validation set of $\bar{n}$ data points $\bar{X}$ and labels $\bar{y}$ is optimised. As such, our lever level problem will be the training problem defined in Equation (5.8) and our upper level problem will be to minimise a loss function $\mathcal{L}$ whose value decreases as the performance of the SVM model on the validation set increases.

The appropriate selection of $\mathcal{L}$ is important as minimising different loss functions may lead to a different selection of optimal hyperparameters. One commonly used loss function when tuning hyperparameters is counting the number of misclassified points. The sign of the expression

$$\omega^T \phi(X_i) + b \tag{5.9}$$

shows which side of the hyperplane defining the SVM's decision rule the point X_i lies on, and hence, the function

$$y_i \left(\omega^T \phi(X_i) + b \right),$$

which is positive if X_i lies on the correct side of the hyperplane and negative if it lies on the wrong side. Therefore, if we define

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \ge 0, \\ -1 & \text{otherwise,} \end{cases}$$

then the function

$$\frac{\text{sign}\left(-y_i(\omega^T \phi(X_i) + b)\right) + 1}{2}$$

has a value of 1 for incorrectly classified points and 0 for correctly classified points. Therefore averaging the result of this function, applied to all of the points in the validation set, can be used as a loss function to assess how accurately the model predicts labels for the validation set:

$$\mathcal{L}_{\text{count}}(\gamma, \omega, b, \bar{X}, \bar{y}) = \frac{1}{\bar{n}} \sum_{i \in [\bar{n}]} \frac{\text{sign}\left(-\bar{y}_i(\omega^T \phi(\bar{X}_i) + b)\right) + 1}{2}. \tag{5.10}$$

Minimising this loss function is analogous to maximising prediction accuracy and, as such, is the loss function typically used when approximating solutions to the bilevel problem using grid-search, random search or Bayesian optimisation. However this loss function is non-smooth as can be seen in the example in Figure 5.2. For this reason gradient based optimisation methods will not be able to travel on this loss surface as the gradient in all directions will be 0. For this reason we must consider another loss function for our upper level objective.

The primal form of the lower level training problem (5.2) is a bi-objective problem where two terms are minimised: $\omega^T\omega/2$; the reciprocal of the separation margin size, and $\sum_{i=1}^{n} \xi_i$; the sum of the slack variables. The hyperparameter C controls the relative importance of these two factors to the objective function. $\omega^T\omega/2$ is a regularisation term. By minimising it during training, we maximise the separation margin, ensuring that the decision rule defined by the SVM solution's hyperplane generalises well to unseen data. In the upper level objective we are assessing how well the SVM predicts labels for the hereto unseen data in the validation set. For this reason, maximising the separation margin in the upper level would be redundant. Furthermore, the size of the margin is unaffected by the validation points and, as such, should not be considered when evaluating validation performance. We can however calculate new slack variables for each of the validation points. As all ξ are positive in the objective function and this objective function is being minimised, the two constraints

$$\begin{aligned} \xi_i &\geq 1 - y_i(\omega^T \phi(X_i) + b) && \forall i \in [n], \\ \xi_i &\geq 0 && \forall i \in [n], \end{aligned}$$

are equivalent to

$$\xi_i = \max\left(0, 1 - y_i(\omega^T \phi(X_i) + b)\right) \quad \forall i \in [n].$$

As illustrated in Figure 5.1, the function $1 - y_i(\omega^T\phi(X_i) + b)$ is negative if the point X_i is on the correct side of the hyperplane and outside of the separation margin and has value 0 if the point point X_i lies on the correct side of the hyperplane and exactly on the separation margin. From there the value of $1 - y_i(\omega^T\phi(X_i) + b)$ increases lineally the further the point X_i is from being on the correct side of the hyperplane and outside of the separation margin, and has value 1 when X_i lies exactly on the hyperplane. We can therefore construct a loss function which minimises the average of the new slack

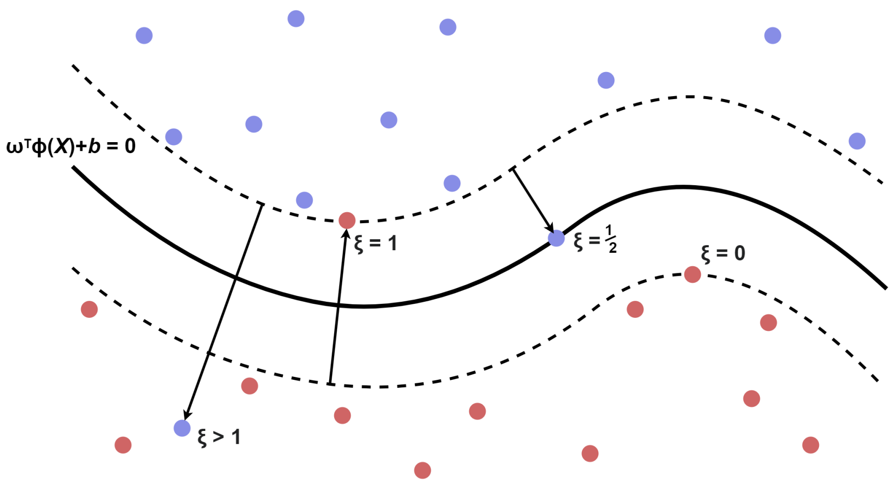

FIGURE 5.1: A diagram showing the value of the slack variables ξ for points in differing positions. The slack variable for all correctly classified points which lie outside of the separation margin is 0. The value of the slack variable then increases linearly the further the point is from being correctly classified and outside of the margin. When this distance is the size of the margin, the slack variable has value 1. As this distance continues to increase beyond the size of the margin, the size of the slack variable continues to increase lineally with it.

variables which are calculated using the validation points:

$$\mathcal{L}_{\text{SVM}}(\gamma, \omega, b, \bar{X}, \bar{y}) = \frac{1}{\bar{n}} \sum_{i \in [\bar{n}]} \max\left(0, 1 - \bar{y}_i(\omega^T \phi(\bar{X}_i) + b)\right). \tag{5.11}$$

This *SVM loss function* is larger the further validation points are from being both on the correct side of the hyperplane and outside of the separation margin. Importantly this loss function is significantly smoother than the counting loss function, detailed in Equation (5.10), as, provided at least one slack variable is non-zero (which is always the case unless the model is already achieving perfect accuracy), tiny changes in the position of the hyperplane correspond to changes in the size of the slack variables. An example of this improved smoothness can be seen in Figure 5.2. However, the function $\max(0, x)$, used within the SVM loss function is non-smooth when $x = 0$, and so we will use a smooth approximation (Biswas et al., 2021). We define

$$\text{SmoothMax}(x) = x + \sqrt{x^2 + \zeta}, \tag{5.12}$$

where ζ is a small perturbation. We therefore have,

$$\lim_{\zeta \to 0} \left(x + \sqrt{x^2 + \zeta}\right) = \max(0, x)$$

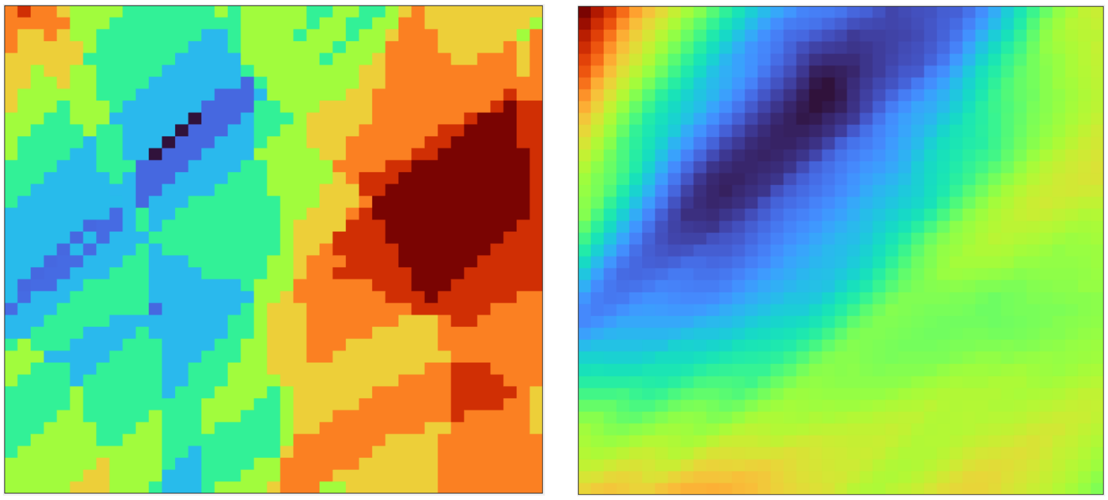

FIGURE 5.2: Heat maps generated by training, then evaluating $\mathcal{L}_{\text{count}}$ (left) and $\mathcal{L}_{\text{SVM}}$ (right) of SVMs trained with a grid of different values of γ (across the x-axis) and C (across the y-axis).

The next challenge is to calculate $\omega^T\phi(\bar{X}_i) + b$ in terms of the dual parameters which we use in our lower level. We can substitute ω out using (5.5):

$$\begin{aligned}\omega^T\phi(\bar{X}_i) + b \quad &= b + \sum_{j\in[\hat{n}]} \alpha_j\hat{y}_j\phi(\widehat{X}_j)\phi(\bar{X}_i),\\ &= b + \sum_{j\in[\hat{n}]} \alpha_j\hat{y}_jK(\widehat{X}_j, \bar{X}_i),\\ &= b + \sum_{j\in[\hat{n}]} \alpha_j\hat{y}_j exp(-\gamma\|\widehat{X}_j - \bar{X}_i\|^2).\end{aligned}$$

To recover the value of b from a dual solution, we consider the following primal and dual complementarity conditions:

$$(1 - \xi_i - \hat{y}_i(\omega^\top\phi(\widehat{X}_i) + b))\alpha_i = 0 \qquad \forall i \in [\hat{n}], \tag{5.13}$$

$$\xi_i\eta_i = 0 \qquad \forall i \in [\hat{n}]. \tag{5.14}$$

If $\alpha_i > 0$, then $1 - \xi_i - \hat{y}_i(\omega^\top\phi(\widehat{X}_i) + b) = 0$, implying $\xi_i = 1 - \hat{y}_i(\omega^\top\phi(\widehat{X}_i) + b)$. Since $\xi \geq 0$, we can have both $\xi_i = 1 - \hat{y}_i(\omega^\top\phi(\widehat{X}_i) + b) = 0$ and $\xi_i = 1 - \hat{y}_i(\omega^\top\phi(\widehat{X}_i) + b > 0$. If $\alpha_i < C$, then due to Equation (5.7) we have $\eta_i > 0$. This implies $\xi_i = 0$ due to Equation (5.14). Which in turn implies $1 - \hat{y}_i(\omega^\top\phi(\widehat{X}_i) + b) = 0$. Therefore, we have:

$$1 - \hat{y}_i(\omega^\top\phi(\widehat{X}_i) - b) = 0 \qquad \forall i \in [\hat{n}] : 0 < \alpha_i < C.$$

After multiplying the equation through by y_i and relying on $y_i^2 = 1$ (or, alternatively, dividing through by y_i and noting that $\frac{1}{y_i} = y_i$), we have:

$$b = \hat{y}_i - \omega^\top\phi(\widehat{X}_i) \qquad \forall i \in [\hat{n}] : 0 < \alpha_i < C.$$

Using Equation (5.5), we deduce

$$b = \hat{y}_i - \sum_{j \in [\hat{n}]} \alpha_j \hat{y}_j \phi(\widehat{X}_j)\phi(\widehat{X}_i) \qquad \forall i \in [\hat{n}] : 0 < \alpha_i < C.$$

Notice that the need to sum over the only points in $[\hat{n}]$ with $\alpha_j > 0$ never arises here. In our training problem α is constrained such that $0 \leq \alpha_i \leq C$ for $i \in [\hat{n}]$ and so b can not be calculated using all elements α. In our upper level objective function we can not check which α_i meet the conditions for being used to calculate b so we require a method which can take as input all of the alphas. In the following result, we propose a method for calculating b using all of the elements of α, which introduces a function Ω and defines the requirements for this function.

Proposition 5.1. Given a function $\Omega : \mathbb{R} \to \mathbb{R}$ for which $\Omega(0) = \Omega(C) = 0$ and $\Omega(x) > 0 \; \forall \; 0 < x < C$. Then,

$$b = \frac{1}{\sum_{i \in [\hat{n}]} \Omega(\alpha_i)} \sum_{i \in [\hat{n}]} \Omega(\alpha_i) \left(\hat{y}_i - \sum_{j \in [\hat{n}]} \alpha_j \hat{y}_j \exp\left(-\gamma \|\widehat{X}_i - \widehat{X}_j\|^2\right) \right).$$

Proof. Let $\bar{I} = \{i \in [\hat{n}] : 0 < \alpha_i < C\}$ and $\hat{I} = \{i \in [\hat{n}] : \alpha_i = 0 \text{ or } \alpha_i = C\}$. We have,

$$\begin{aligned}
&\frac{1}{\sum_{i \in [\hat{n}]} \Omega(\alpha_i)} \sum_{i \in [\hat{n}]} \Omega(\alpha_i) \left(\hat{y}_i - \sum_{j \in [\hat{n}]} \alpha_j \hat{y}_j \exp\left(-\gamma \|\widehat{X}_i - \widehat{X}_j\|^2\right) \right) \\
&= \frac{1}{\sum_{i \in \bar{I}} \Omega(\alpha_i) + \sum_{i \in \hat{I}} \Omega(\alpha_i)} \sum_{i \in \bar{I}} \Omega(\alpha_i) \left(\hat{y}_i - \sum_{j \in [\hat{n}]} \alpha_j \hat{y}_j \exp\left(-\gamma \|\widehat{X}_i - \widehat{X}_j\|^2\right) \right) \\
&\qquad + \frac{1}{\sum_{i \in \bar{I}} \Omega(\alpha_i) + \sum_{i \in \hat{I}} \Omega(\alpha_i)} \sum_{i \in \hat{I}} \Omega(\alpha_i) \left(\hat{y}_i - \sum_{j \in [\hat{n}]} \alpha_j \hat{y}_j \exp\left(-\gamma \|\widehat{X}_i - \widehat{X}_j\|^2\right)) \right) \\
&= \frac{1}{\sum_{i \in \bar{I}} \Omega(\alpha_i) + \sum_{i \in \hat{I}} 0} \sum_{i \in \bar{I}} \Omega(\alpha_i) b \\
&\qquad + \frac{1}{\sum_{i \in \bar{I}} \Omega(\alpha_i) + \sum_{i \in \hat{I}} 0} \sum_{i \in \hat{I}} 0 \left(\hat{y}_i - \sum_{j \in [\hat{n}]} \alpha_j \hat{y}_j \exp\left(-\gamma \|\widehat{X}_i - \widehat{X}_j\|^2\right) \right) \\
&= \frac{1}{\sum_{i \in \bar{I}} \Omega(\alpha_i)} \sum_{i \in \bar{I}} \Omega(\alpha_i) b \\
&= b
\end{aligned}$$

and this concludes the proof. □

Intuitively Ω should be an indicator function (a linear combination of Heaviside functions, which have a value of 0 up to a certain value of x where their value steps up to 1) whereby $\Omega(x) = 1$ for all $0 < x < C$. This function however would be non-smooth.

It could be approximated using a sum of sigmoid functions however it is worth noting that the only restrictions on our choice of function to use for Ω are those given in Proposition 5.1. Specifically, there is no requirement that the value $\Omega(x)$ should take for any particular value for $0 < x < C$ and there is no requirement for $\Omega(x)$ to have the same value for all $0 < x < C$.

Let, $\hat{b}(\widehat{X}_i, \hat{y}_i) = \hat{y}_i - \sum_{j\in[\hat{n}]} \alpha_j \hat{y}_j \exp\left(-\gamma\|\widehat{X}_i - \widehat{X}_j\|^2\right)$. To calculate b we take a weighted sum of each $\hat{b}(\widehat{X}_i, \hat{y}_i)$ with $0 < \alpha_i < C$. However, not every $\hat{b}(\widehat{X}_i, \hat{y}_i)$ with $0 < \alpha_i < C$ needs to have a significant contribution to the formula for calculating b. In fact, if for example, a given α_i is very close to 0 it is unclear whether the difference between α_i and 0 is a numerical error or an actual difference. To safeguard ourselves against this, we select an Ω which is near linear when x is close to 0 and C. This ensures that if the difference between α_i and 0 or C is orders of magnitude larger than the difference between α_j and 0 or C then the contribution of $\hat{b}(\widehat{X}_i, \hat{y}_i)$ to the calculation of b is orders of magnitude larger than that of $\hat{b}(\widehat{X}_j, \hat{y}_j)$. For example if the two elements of α which is furthest from 0 and C are $\alpha_i = C/2$ and $\alpha_j = 10^{-6}$ then the contribution of $\hat{b}(\widehat{X}_j, \hat{y}_j)$ to calculating b should be insignificant compared to the contribution of $\hat{b}(\widehat{X}_i, \hat{y}_i)$. Likewise, if the two elements of α which is furthest from 0 and C are $\alpha_i = 10^{-6}$ and $\alpha_j = 10^{-12}$ then the contribution of $\hat{b}(\widehat{X}_j, \hat{y}_j)$ to calculating b should be insignificant compared to the contribution of $\hat{b}(\widehat{X}_i, \hat{y}_i)$. To achieve this we propose the quadratic function

$$\bar{\Omega}(x, C) = 1 - \left(\frac{2x}{C} - 1\right)^2 + \tau. \tag{5.15}$$

It is sometimes the case that all elements of α are so close to 0 or C that the difference is numerically 0. In such cases all $\Omega(\alpha_i) = 0\ \forall \alpha_i \in \alpha$ and hence in the proof of Proposition 5.1, we would divide by 0. To avoid this we add a very small perturbation τ to Ω. Having calculated the primal model parameters using the dual form parameters we can now rewrite the SVM loss Function shown in (5.11) as

$$\mathcal{L}_{\text{SVM}}(C, \gamma, \alpha, \widehat{X}_i, \bar{X}, \hat{y}, \bar{y},) = \frac{1}{\bar{n}} \sum_{i\in[\bar{n}]} \text{SmoothMax}\Bigg(1 - \bar{y}_i(\omega^T\phi(\bar{X}_i) + b)\Bigg),$$

where

$$\begin{aligned} \omega^T\phi(\bar{X}_i) &= \sum_{j\in[\hat{n}]} \alpha_j \hat{y}_j exp(-\gamma\|\widehat{X}_j - \bar{X}_i\|^2), \\ b &= \frac{1}{\sum_{i\in[\hat{n}]} \bar{\Omega}(\alpha_i)} \sum_{i\in[\hat{n}]} \bar{\Omega}(\alpha_i) \left(\hat{y}_i - \sum_{j\in[\hat{n}]} \alpha_j \hat{y}_j \exp\left(-\gamma\|\widehat{X}_i - \widehat{X}_j\|^2\right)\right), \end{aligned}$$

and SmoothMax is defined in (5.12), $\bar{\Omega}$ is defined in (5.15). Figure 5.2 shows the loss surface of this function with respect to γ and C. As can be seen, this loss surface is smooth unlike that of counting loss function which is non-smooth and has gradient is 0 everywhere.

Having determined which loss function we intend to use to evaluate validation performance in the upper level objective we now look to adapt our bilevel problem to k-fold CV. In k-fold CV the n data points X with their cosponsoring labels y are split into k roughly equal sized folds. A fold is selected to be the validation set while remaining folds are used as the training set. CV is performed in which an SVM is trained on the training set and evaluated on the validation set. The next fold is then selected to form the validation set. The average validation performance across the k rounds of CV is used to assess overall validation performance.

We define $\bar{X}^i$ as the set of $\bar{n}^i$ data points in fold i with $\bar{y}^i$ as their corresponding labels and $\widehat{X}^i$ as the set of $\hat{n}^i$ data points which are not in fold i with $\hat{y}^i$ as their corresponding labels. We also define α^i as model parameters of the SVM trained on the data from folds other than fold i and $\alpha = \{\alpha^1; \dots; \alpha^k\}$ as the vector formed by stacking each vector α^i for all $i \in [k]$. k-fold CV for hyperparameter selection can be constructed as a bilevel problem where the lower level problem is a multi-objective problem and the upper level problem is maximising the average of the model performances on the validation sets (or more specifically minimising a loss function which is inversely proportional to the model performances). In the lower level there are k objective functions, each of which responsible for the training of one SVM. For a given SVM trained on $\widehat{X}^i$ and $\hat{y}^i$ fold i the only parameters the objective function responsible for its training are α^i. Therefore, as no model parameters are shared between the k lower level objective functions, minimising each of them independently is equivalent to minimising their sum. Our bilevel formulation of the hyperparameter tuning for a RBF kernel SVM problem is therefore given as

$$\begin{aligned}
\min_{C,\gamma,\alpha} \quad & \frac{1}{k}\sum_{i\in[k]} \frac{1}{\bar{n}^i} \sum_{j\in[\bar{n}^i]} \text{SmoothMax}\Bigg(1 - \bar{y}^i_j \Bigg(\Big(\sum_{l\in[\hat{n}^i]} \alpha^i_l \hat{y}^i_l exp(-\gamma \|\widehat{X}^i_l - \bar{X}^i_j\|^2) \Big) \\
& \qquad + \frac{1}{\sum_{l\in[\hat{n}^i]} \bar{\Omega}(\alpha^i_l)} \sum_{l\in[\hat{n}^i]} \bar{\Omega}(\alpha^i_l) \Big(\hat{y}^i_l - \sum_{m\in[\hat{n}^i]} \alpha^i_m \hat{y}^i_m \exp\Big(-\gamma \|\widehat{X}^i_m - \widehat{X}^i_j\|^2\Big) \Big) \Bigg) \Bigg) \\
\text{s.t.} \quad & C \geq 0, \\
& \gamma \geq 0, \\
& \alpha \in \arg\min_{\alpha} \quad \sum_{i\in[k]} \Bigg(\frac{1}{2} \sum_{j\in[\hat{n}^i]} \sum_{l\in[\hat{n}^i]} \alpha^i_j \alpha^i_l \hat{y}^i_j \hat{y}^i_l \exp\Big(-\gamma \|\widehat{X}^i_j - \widehat{X}^i_l\|^2\Big) - \sum_{j\in[\hat{n}^i]} \alpha^i_j \Bigg) \\
& \text{s.t.} \quad 0 \leq \alpha^i_j \leq C \qquad \forall i \in [k], \forall j \in [\hat{n}^i], \\
& \qquad \sum_{j\in[\hat{n}^i]} \hat{y}^i_j \alpha^i_j = 0 \qquad \forall i \in [k].
\end{aligned} \tag{5.16}$$

Empirically we see that, provided γ and C are initialised to be greater than 0, that they are never driven below 0 and so the upper level constraints can be dropped.

5.3 Single level reformulation and theoretical analysis

In this section, we construct an algorithm capable of solving the bilevel problem (5.16). The following subsections will focus on first reformulating the problem into a single level problem, then applying the Scholtes relaxation and finally detailing an algorithm for driving the solution of the relaxed problem to be that of the original problem. Here, we transform problem (5.16) into a single-level optimisation problem by means of the Karush-Kuhn-Tucker (KKT) reformulation.

Note that both criteria for deriving the KKT reformulation are met. We have only linear constraints in our lower level problem and the lower level objective is convex by virtue of being the Lagrangian dual of a convex optimisation problem.

To proceed, we introduce some notation to write the problem in a more compact form, which will facilitate the presentation of the theoretical elements of our analysis. We let e^i denote a vector with elements all ones in $\mathbb{R}^{\hat{n}^i}$. With this notation we can write the subproblem of the lower level problem (in the dual form), the problem of training a single RBF kernel SVM on the training set $(\widehat{X}^i, \hat{y}^i)$ as follows

$$\begin{array}{rll} \alpha^i \in & \arg\min_{\alpha^i \in \mathbb{R}^{n^i}} & \frac{1}{2}(\alpha^i)^\top Q^i(\gamma)\alpha^i - (\alpha^i)^\top e^i \\ & \text{s.t.} & \alpha^i \in [0, C], \\ & & (\alpha^i)^\top \hat{y}^i = 0. \end{array} \tag{5.17}$$

Here $Q^i(\gamma) \in \mathbb{R}^{n^i \times n^i}$ is defined by

$$(Q^i(\gamma))_{jl} = \hat{y}_j^{(i)} \hat{y}_l^{(i)} \exp(-\gamma \|\widehat{X}_j^{(i)} - \widehat{X}_l^{(i)}\|^2).$$

The Lagrange function of (5.17) is as follows

$$\hat{L}(\alpha^i, \bar{\lambda}^i, \underline{\lambda}^i, u^i) = \frac{1}{2}(\alpha^i)^\top Q^i(\gamma)\alpha^i - (\alpha^i)^\top e^i - \langle \alpha^i, \bar{\lambda}^i \rangle - \langle Ce^i - \alpha^i, \underline{\lambda}^i \rangle + u^i (\alpha^i)^\top \hat{y}^i,$$

where $\bar{\lambda}^i \in \mathbb{R}^{n^i}$, $\underline{\lambda}^i \in \mathbb{R}^{n^i}$ and $u^i \in \mathbb{R}$ are the Lagrange multipliers corresponding to the lower bound constraint, upper bound constraint and the equality constraint, respectively. The KKT conditions of (5.17) are

$$\left\{ \begin{array}{l} Q^i(\gamma)\alpha^i - e^i - \bar{\lambda}^i + \underline{\lambda}^i + u^i \hat{y}^i = 0, \\ 0 \leq \alpha^i \perp \bar{\lambda}^i \geq 0, \\ 0 \leq \underline{\lambda}^i \perp Ce^i - \alpha^i \geq 0, \\ (\alpha^i)^\top \hat{y}^i = 0. \end{array} \right. \quad i \in [k]. \tag{5.18}$$

In Equation (5.18), we use the orthogonal operator. Using this operator $0 \leq x \perp y \geq 0$ is equivalent to $0 \leq x,\ 0 \leq y,\ x \cdot y = 0$. By the first equation in (5.18), there is

$$\bar{\lambda}^i = Q^i(\gamma)\alpha^i - e^i + \underline{\lambda}^i + u^i \hat{y}^i := \theta^i.$$

We can eliminate the vector of Lagrange multipliers $\bar{\lambda}^i$ corresponding to the lower bound constraints by substituting this into (5.18):

$$\begin{cases} 0 \leq \alpha^i \perp \theta^i \geq 0, \\ 0 \leq \underline{\lambda}^i \perp Ce^i - \alpha^i \geq 0, \quad i \in [k]. \\ (\alpha^i)^\top \hat{y}^i = 0, \end{cases} \tag{5.19}$$

Let $m = \sum_{i=1}^{k} \hat{n}^i$. We can then define

$$\begin{array}{lcl} \alpha & = & (\alpha^1; \cdots; \alpha^k) \in \mathbb{R}^m, \\ \underline{\lambda} & = & (\underline{\lambda}^1; \cdots; \underline{\lambda}^k) \in \mathbb{R}^m, \\ e & = & (e^1; \cdots; e^k) \in \mathbb{R}^m, \\ u & = & (u^1; \cdots; u^k) \in \mathbb{R}^k, \\ \theta(\gamma, \alpha, \underline{\lambda}, u) & = & (\theta^1; \cdots; \theta^k) \in \mathbb{R}^m. \end{array}$$

The KKT conditions (5.19) for the entire lower level problem (5.17) can then be rewritten in the following form

$$\begin{cases} 0 \leq \alpha \perp \theta(\gamma, \alpha, \underline{\lambda}, u) \geq 0, \\ 0 \leq \underline{\lambda} \perp Ce - \alpha \geq 0, \\ \overline{Y}\alpha = 0, \end{cases} \tag{5.20}$$

where

$$\overline{Y} = \begin{bmatrix} (\check{y}^1)^\top \\ \vdots \\ (\check{y}^k)^\top \end{bmatrix} \in \mathbb{R}^{k \times m},\ \check{y}^i = (0_{n^1}, \cdots, 0_{n^{i-1}}, \quad \hat{y}^i, \quad 0_{n^{i+1}}, \cdots, 0_{n^k}) \in \mathbb{R}^m.$$

Let $\acute{m} = 2 + k + 2m$, we define

$$v = \begin{bmatrix} C \\ \gamma \\ \underline{\lambda} \\ u \\ \alpha \end{bmatrix} \in \mathbb{R}^{\acute{m}},\ H(v) = \begin{bmatrix} \alpha \\ \underline{\lambda} \end{bmatrix},\ G(v) = \begin{bmatrix} \theta(\alpha, \gamma, \underline{\lambda}, u) \\ Ce - \alpha \end{bmatrix},\ h(v) = \overline{Y}\alpha. \tag{5.21}$$

Notice that by the definition of v, $\theta(\gamma, \alpha, \underline{\lambda}, u)$ can be also denoted by $\theta(v)$, which we will use instead of $\theta(\gamma, \alpha, \underline{\lambda}, u)$ from this point on. Using these general notations the

KKT system (5.20) of the lower level problem can be written as

$$\begin{cases} 0 \leq H(v) \perp G(v) \geq 0, \\ h(v) = 0, \end{cases}$$

and hence, the KKT reformulation of problem (5.16) can be written in the compact form

$$\begin{array}{ll} \min_{v \in \mathbb{R}^{\tilde{m}}} & f(v) \\ \text{s.t.} & 0 \leq H(v) \perp G(v) \geq 0, \\ & h(v) = 0, \end{array} \tag{MPEC}$$

where $f(v)$ represents the objective function shown in problem (5.16).

Clearly, problem (MPEC) is a special class of the Mathematical Program with Equilibrium Constraints (MPEC); see, e.g., Flegel (2005) for more details. Note that the algorithm to be proposed in the next section to solve the problem will compute a stationary point; however, to formulate stationary points, or "KKT"-type necessary optimality conditions, a regularity condition is needed. The most commonly used regularity condition, or constraint qualification, to derive necessary optimality conditions for optimisation problems with non-linear constraints is the Mangasarian-Fromovitz Constraint Qualification (MFCQ); cf. Nocedal and Wright (1999).

Unfortunately, for MPECs, it is well-known that the MFCQ systematically fails (Ye et al., 1997). Hence, to derive necessary optimality conditions for problem (MPEC), a specifically tailored version of the MFCQ has been introduced in the literature; see Flegel (2005) and references therein for details. Next present the this regularity condition for our problem (MPEC). To proceed, let v be a feasible point of problem (MPEC), then we have the following standard decomposition of the indices associated to the complementarity constraints:

$$\begin{array}{lll} J_G & := & \{i \mid G_i(v) = 0, \ H_i(v) > 0\}, \\ J_{GH} & := & \{i \mid G_i(v) = 0, \ H_i(v) = 0\}, \\ J_H & := & \{i \mid G_i(v) > 0, \ H_i(v) = 0\}. \end{array}$$

Using this decomposition, we can now introduce the MPEC Mangasarian-Fromovitz (MPEC-MFCQ) as follows.

Definition 5.2. A feasible point v of problem (MPEC) satisfies the MPEC-MFCQ if and only if the set of gradients of active constraints

$$\{\nabla G_i(v) \mid i \in J_G \cup J_{GH}\} \cup \{\nabla H_i(v) \mid i \in J_H \cup J_{GH}\} \cup \{\nabla h_i(v) \mid i \in [k]\} \tag{5.22}$$

in (5.22) is positive-linearly independent.

The set of gradient vectors in (5.22) is said to be positive-linearly *dependent* if there exists non-zero non-negetive scalars $\{\delta_i\}_{i\in I_G\cup I_{GH}}$, $\{\beta_i\}_{i\in I_H\cup I_{GH}}$ and $\eta \in \mathbb{R}^k$ such that

$$\Sigma_{i\in I_G\cup I_{GH}}\delta_i\nabla G_i(v) + \Sigma_{i\in I_H\cup I_{GH}}\beta_i\nabla H_i(v) + \sum_{i=1}\eta_i\nabla h_i(v) = 0.$$

Otherwise, we say that this set of gradient vectors is positive-linearly *independent*.

It is also important to mention that because of the complementarity constraints in problem (MPEC), various stationarity concepts are possible. Our focus in this chapter will be on the C-stationarity concepts, as, in the next section, we will introduce an algorithm which we prove outputs solutions which are C-stationary.

Definition 5.3. A point v that is feasible for problem (MPEC) will be said to be C-stationary if there exist Lagrange multipliers μ^h, μ^G, and μ^H such that the following conditions are satisfied:

$$\begin{aligned} \nabla f(v) + \textstyle\sum_{i=1}^{k}\mu_i^h\nabla h_i(v) - \sum_{j=1}^{k+m}\left[\mu_j^G\nabla G_j(v) + \mu_j^H\nabla H_j(v)\right] &= 0,\\ \forall j\in J_G:\ \ \mu_j^G\ \ \text{free},\ \ \forall j\in J_{GH}:\ \ \mu_j^G\ \ \text{free},\ \ \forall j\in J_H:\ \ \mu_j^G &= 0,\\ \forall j\in J_H:\ \ \mu_j^H\ \ \text{free},\ \ \forall j\in J_{GH}:\ \ \mu_j^H\ \ \text{free},\ \ \forall j\in J_G:\ \ \mu_j^H &= 0,\\ \forall j\in J_{GH}:\ \ \mu_j^G\mu_j^H &\geq 0. \end{aligned}$$

Theorem 5.4 (see, e.g., Scholtes (2001))**.** *Let v be a local optimal solution of problem* (MPEC) *that satisfies the MPEC-MFCQ. Then v is C-stationary.*

To conclude this section, note that it can proven that the MPEC-MFCQ automatically holds for our problem (MPEC) under a mild assumption. This proof follows the same analysis of index sets as was performed in the paper (Li et al., 2021), where the result is established linear kernel SVM. A paper containing this additional analysis is currently being developed but such a work falls outside of the scope of this thesis.

5.4 Numerical method and illustrative examples

In this section we propose an algorithm for solving Problem (MPEC) which utilises the Scholtes relaxation and provide some illustrative examples of this algorithm in action on real data sets from the UCI repository.

5.4.1 The Scholtes relaxation

In order to solve problem (MPEC) more easily, we apply the Scholtes relaxation (Scholtes, 2001). The problem can be equivalently written as,

$$\begin{aligned} \min_{v\in\bar{n}} \quad & f(v) \\ \text{s.t.} \quad & 0 \leq H_i(v), \quad 0 \leq G_i(v), \quad H_i(v)G_i(v) = 0 \qquad \forall i \in [2m], \\ & h(v) = 0. \end{aligned}$$

Where f, G, H, h, v and m are as defined in the previous section. Here, as each element of $H(v)$ and $G(v)$ are strictly non-negative, $H_i(v)G_i(v) = 0$ is equivalent to $H_i(v)G_i(v) \leq 0$. To perform the Scholtes relaxation we relax this such to $H_i(v)G_i(v) \leq t$. Resulting in the following relaxed problem,

$$\begin{aligned} \min_{v\in\bar{n}} \quad & f(v) \\ \text{s.t.} \quad & 0 \leq H_i(v), \quad 0 \leq G_i(v), \quad H_i(v)G_i(v) \leq t \qquad \forall i \in [2m], \\ & h(v) = 0. \end{aligned} \tag{5.23}$$

In our non-relaxed problem we have $H_i(v)G_i(v) = 0$, $0 \leq H_i(v)$ and $0 \leq G_i(v)$ this means that for all $i \in 2m$, for the original complementarity condition to be satisfied, at least one of $H_i(v)$ and $G_i(v)$ must be equal to 0. When we apply the Scholtes relaxation we are allowing both $H_i(v)$ and $G_i(v)$ to both be non-zero for all $i \in 2n$. The largest violation of the unrelaxed complementarity conditions can therefore be written as

$$\text{compVio}(v) = || \min\left(G(v), H(v)\right) ||_\infty,$$

where the *min* function takes a pair of $(2m \times 1)$ vectors and returns a single $(2m \times 1)$ vector for which each element $\min_i(G(v), H(v))$ = $\min(G_i(v), H_i(v))$. This vector contains the violation of each of the complimentary conditions of the non-relaxed problem. To find the largest of these we apply the infinity norm (which simply outputs the largest element of its argument). This value can then be used as a stopping criteria by which we can determine when the value of t is sufficiently small as to render its solutions approximately equal to that of the non-relaxed problem. Hence, we propose the Algorithm 4, which is constructed using a framework borrowed from Hoheisel et al. (2013).

Subsequently, we have the following convergence result, which ensures that a sequence of stationary points of problem (5.23), computed by Algorithm 4, converges to a C-stationary point of problem (MPEC), as defined in the previous section.

Theorem 5.5. (Hoheisel et al., 2013) *Let $\{t_k\} \downarrow 0$ and let v^k be a stationary point of* (NLP-t_k) *with $v^k \to v$ such that MPEC-MFCQ holds at the feasible point v. Then v is a C-stationary point of problem* (MPEC).

Algorithm 4 Scholtes relaxation algorithm ($v_{\text{start}}, t_{\text{start}}, t_{\text{min}}, \epsilon$)

```
Require: starting values for the model parameters $v_{\text{start}}$, $t_{\text{start}} > t_{min} > 0$ and $\epsilon > 0$.
$k \leftarrow 0$
$t_k \leftarrow t_{\text{start}}$
$v \leftarrow v_{\text{start}}$

while $t_k \geq t_{\text{min}}$ and compVio($v$)$> \epsilon$ do
        $v \leftarrow$ the approximate solution to the relaxed problem (5.23)
                using $v$ as a starting point and $t = t_k$.
        $t_{k+1} \leftarrow \frac{t_k}{10}$
        $k \leftarrow k+1$
end while

Return: the value of $v$ after the final iterate.
```

It is well known that in non-convex continuous optimisation, choice of starting point can greatly affect the final solution found by the model. So as not to provide our model an unfair head start in Algorithm 4 we will simply use the naive hyperparameter starting point of $\gamma = 1$, and $C = 1$. At each iteration of Algorithm 4 we solve the optimisation problem using `fmincon` in MATLAB. In preliminary experiments we found that `fmincon`'s default interior point solver was not able to reduce the objective function so we instead use the Sequential Quadratic Programming (SQP) solver, which has been shown to perform well for solving the hyperparameter tuning for SVMs problem (Bennett et al., 2006). However, the SQP solver did not work well when initialised at an infeasible point.

For this reason, we aim to compute a feasible starting point v for Problem 5.23. To find this feasible starting point we consider the following infeasible point ($\underline{\lambda} = \mu = 0$, $\alpha = C/2$, $\gamma = 1$ and $C = 1$) and then assert feasibility by adjusting α, $\underline{\lambda}$ and μ. Our procedure for asserting feasibility by adjusting α, $\underline{\lambda}$ and μ is to set the objective function of Problem 5.23 to 0 and add two equality constraints fixing the values of γ and C. Solving this new problem simply finds a feasible point with $C = v_1$ and $\gamma = v_2$. As in this problem we do not need to decrease the objective function, we can use the interior point solver, which, in this context, is faster than the SQP solver. It is possible that at any iteration Algorithm 4 the SQP solver converges to an infeasible point. Where this is the case we once again use our procedure for asserting feasibility to find a feasible point (without moving γ or C) to start the SQP solver at for the next iteration of Algorithm 4.

When solving Algorithm 4 we use $t_{\text{start}} = 10^{-1}$ and $t_{\text{min}} = 10^{-15}$. In the function $\bar{\Omega}$, used to determine the contribution of each point to the calculation of b (shown in Equation (5.15)) we use a very small perturbation $\tau = 10^{-10}$. Likewise we use the perturbation $\zeta = 10^{-4}$ to smooth the max function in Equation (5.12).

5.4.2 Illustrative examples

In this section we will provide four illustrative examples of our bilevel hyperparameter tuning model in action on real data sets from the UCI Machine Learning Repository. The Cleveland Heart Disease data set consists of 303 data points comprised of 13 variables together with their associated labels (Detrano et al., 1989). The original labels had value 0 for patients with no heart disease and labels 1-4 for patients with heart disease depending on their condition. To formulate a binary classification problem we assign the label 1 to all patients with heart disease. The Glass Classification data set consists of 214 data points comprised of 10 variables together with their associated labels (Evett and Spiehler, 1987). This data set has 7 classes representing specific types of glass. To formulate a binary classification problem we consider only two, building glass which is produced using floating and building glass which is produced using other methods. The Pima Indians Diabetes data set consists of 768 data points comprised of 8 variables together with their associated labels of whether the patient has diabetes or not (Smith et al., 1988). Finally, the Sonar, Mines vs. Rocks data set consists of 208 data points comprised of 60 variables obtained by bouncing sonar signals off of either metal cylinders (representing mines) or rocks (Gorman and Sejnowski, 1988). The labels associated to each data point reflect whether the object was a metal cylinder or a rock.

In this work we perform k-fold CV using $k = 3$ as suggested in similar works on bilevel optimisation for SVM hyperparameter tuning (Kunapuli et al., 2008; Bennett et al., 2006, 2008). The model has $\dot{m}$ parameters where $\dot{m} = 2 + k + 2m$ and $m = n(k-1)$. In order to keep the size of the bilevel problem manageable (with 405 parameters) we sample $n = 100$ data point, label pairs from each data set. To ensure that class imbalance did not play a role in our analysis our new 100 point data sets contain 50 data point, label pairs from each class. As scaling can affect the performance of RBF kernel SVMs, we apply standard scaling to each variable.

We compare our bilevel hyperparameter tuning algorithm with grid-search. This grid-search will be performed by iteratively selecting values of γ and C from a grid of hyperparameter combinations, solving the lower level training problem 5.8 using `fmincon` in MATLAB and then using the output values for α together with the selected vaules for γ and C to compute the value of the objective function of Problem 5.23.

The hyperparameter grid will have shape $g \times g$ and will be comprised of a logarithmic range of values of $C \in \{10^{-4+8(i-1)/g-1} \quad \forall i \in [g]\}$ and values of $\gamma \in \{10^{-6+12(i-1)/g-1}$ $\forall i \in [g]\}$. This represents searching a range of values of C from 10^{-4} to 10^{4}, which is very similar to the range of values used in other studies on bilevel optimisation for SVM hyperparameter tuning (Moore et al., 2009; Bennett et al., 2006). As, to the best of our knowledge, this is the first work on bilevel optimisation for hyperparameter tuning of SVMs which uses the RBF kernel, we cannot base our range of values for γ off of that used in a similar work. The range of values we have chosen for γ of 10^{-6}

to 10^6 is derived from the range of values evaluated by MATLAB's implementation of grid search for optimising the hyperparameters of a RBF kernel SVM. Increasing the value of g will improve the quality of the best hyperparameter combination found by grid-search, however it will also increase the run time as more SVMs will have to be trained. Of course, this also means that decreasing g will lead to a shorter run time at the cost of best the hyperparameter combination having a larger objective value. As there are two hyperparameters we have: run time $\propto g^2$.

We use $g = 16$, as the time taken to complete this grid-search was comparable to the amount of time taken to run our bilevel hyperparameter tuning algorithm. For some data sets this gridsearch is slightly faster than our bilevel method and in others it is slower. It would be possible to select a grid size for this gridsearch such that the run times are even more similar. We chose not to do this and to instead select the grid which gave the closest run time across all data sets. It can be seen in Figures 5.3, 5.4, 5.5 and 5.6, that even if we had used a slightly finer mesh for the grid, this would not result in a dramatic improvement in performance for the gridsearch method. A comparison of these run times can be seen in Table 5.1.

Data Set	Tuning Method	Objective Value	Time(s)	γ	C
Cleveland	Bilevel	**1.0979**	5311.0	0.0147	9.6651
	Grid-Search	1.1110	5245.0	0.0100	857.70
Glass	Bilevel	**1.5215**	2252.3	0.0989	11.195
	Grid-Search	1.5666	4203.4	0.0631	21.544
Pima Diabetes	Bilevel	**2.0420**	6530.5	0.0211	5.7434
	Grid-Search	2.0851	4552.6	0.0016	73.564
Sonar	Bilevel	**1.8078**	5779.6	0.0066	11.691
	Grid-Search	1.8252	5211.5	0.0100	251.19

TABLE 5.1: Table detailing the solutions found by each hyperparameter tuning method.

Figures 5.3, 5.4, 5.5 and 5.6 show how our bilevel hyperparameter tuning algorithm navigates the hyperparameter space on our various data sets. The path taken by our bilevel hyperparameter tuning algorithm is shown in yellow with the yellow cross being its final solution. The black crosses represent the points tested by the grid-search described previously. Of course, most of the points evaluated by this grid-search do not lie in the region shown in these plots. The heatmap shown in the background was generated by conducting a very fine grid-search of this local region, the run time of which was orders or magnitude greater than that of either the previous grid-search or our bilevel algorithm.

As can be seen, in each instance the bilevel model appears to converge to a local minimum. As expected, we see that this approach is able to find solutions in-between the

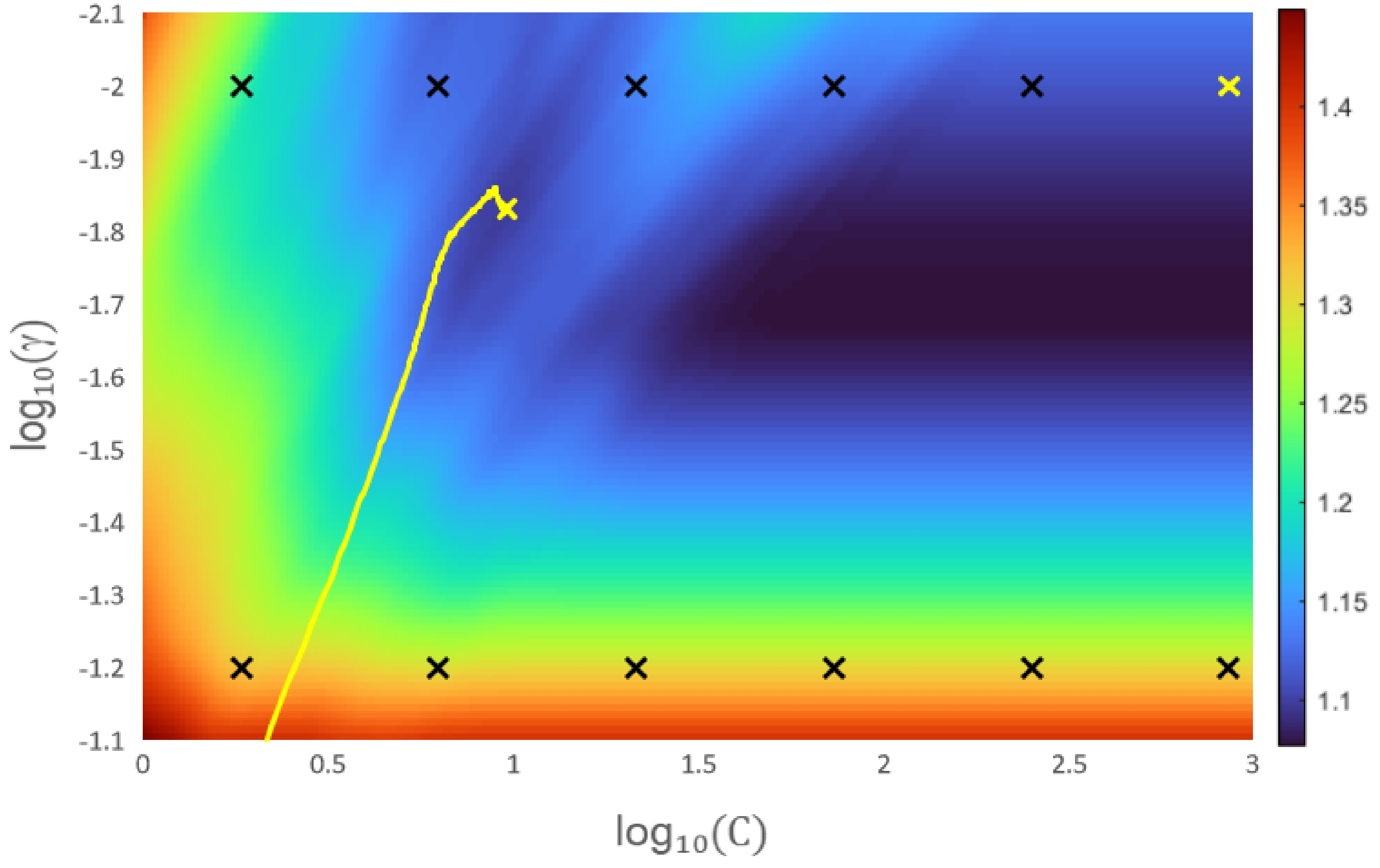

FIGURE 5.3: Figure comparing the performance of our method with that of grid-search for tuning hyperparameters for the Cleveland data set. The path taken through the hyperparameter space by `fmincon` as it attempts to solve the bilevel problem is shown in yellow. The yellow crosses show the solutions found by the bilevel method and grid search method. The black crosses show the points tested by a grid search. The background heatmap was created by performing a very fine grid search to evaluate the value of the upper level objective in this local region.

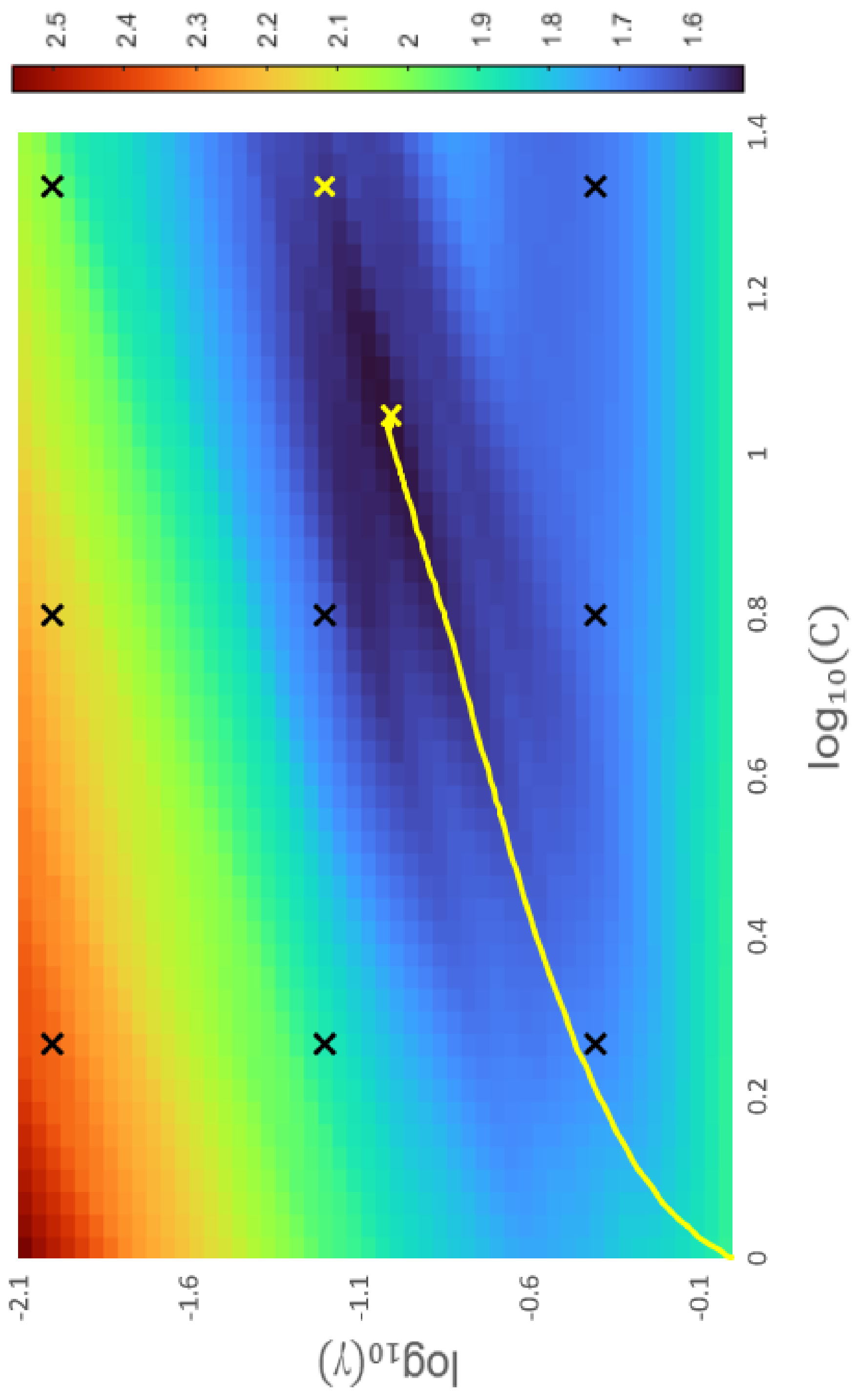

FIGURE 5.4: Figure comparing the performance of our method with that of grid-search for tuning hyperparameters for the Glass data set. See Figure 5.3 for a description of this graph.

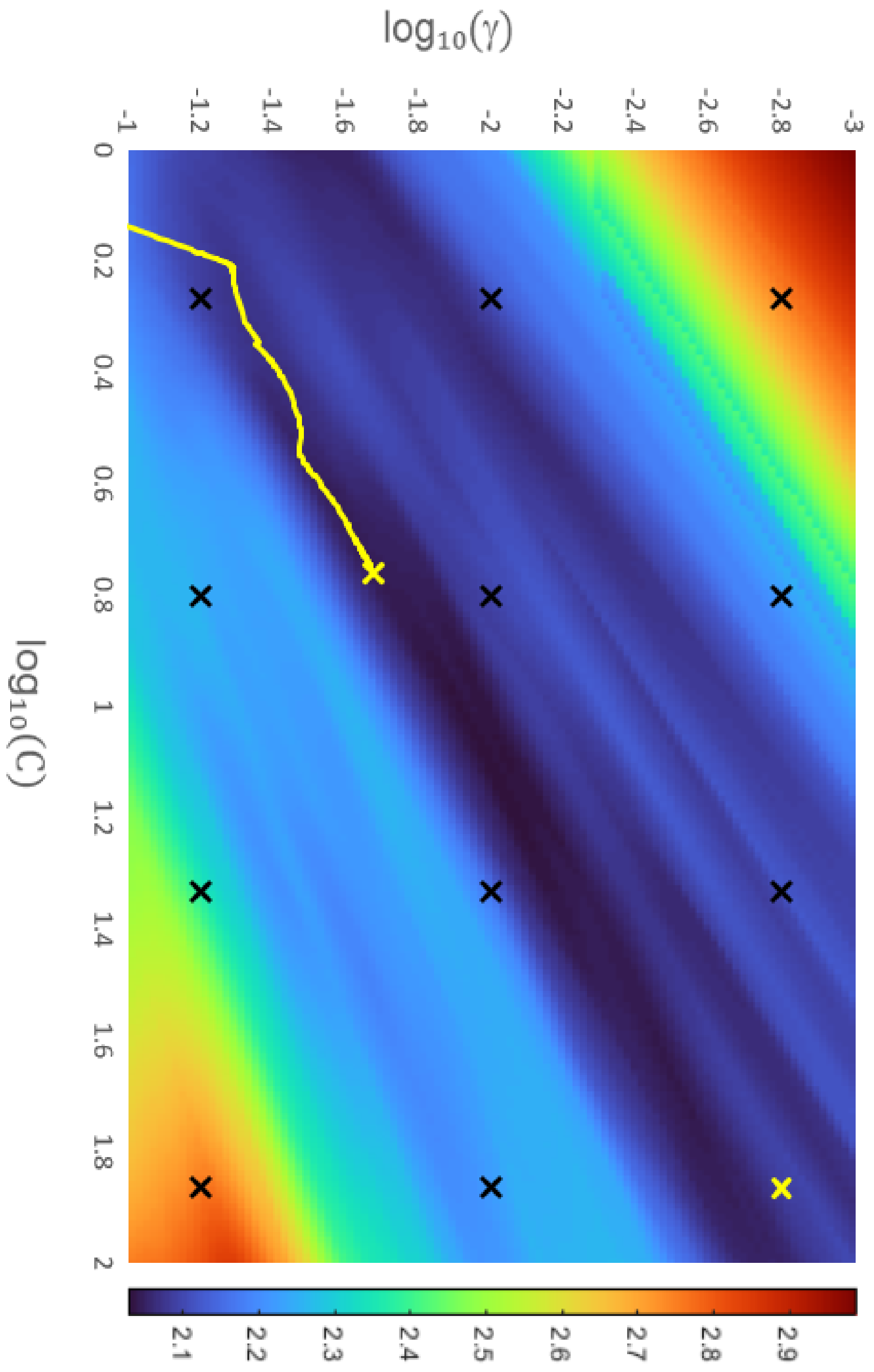

FIGURE 5.5: Figure comparing the performance of our method with that of grid-search for tuning hyperparameters for the Pima Indians Diabetes data set. See Figure 5.3 for a description of this graph.

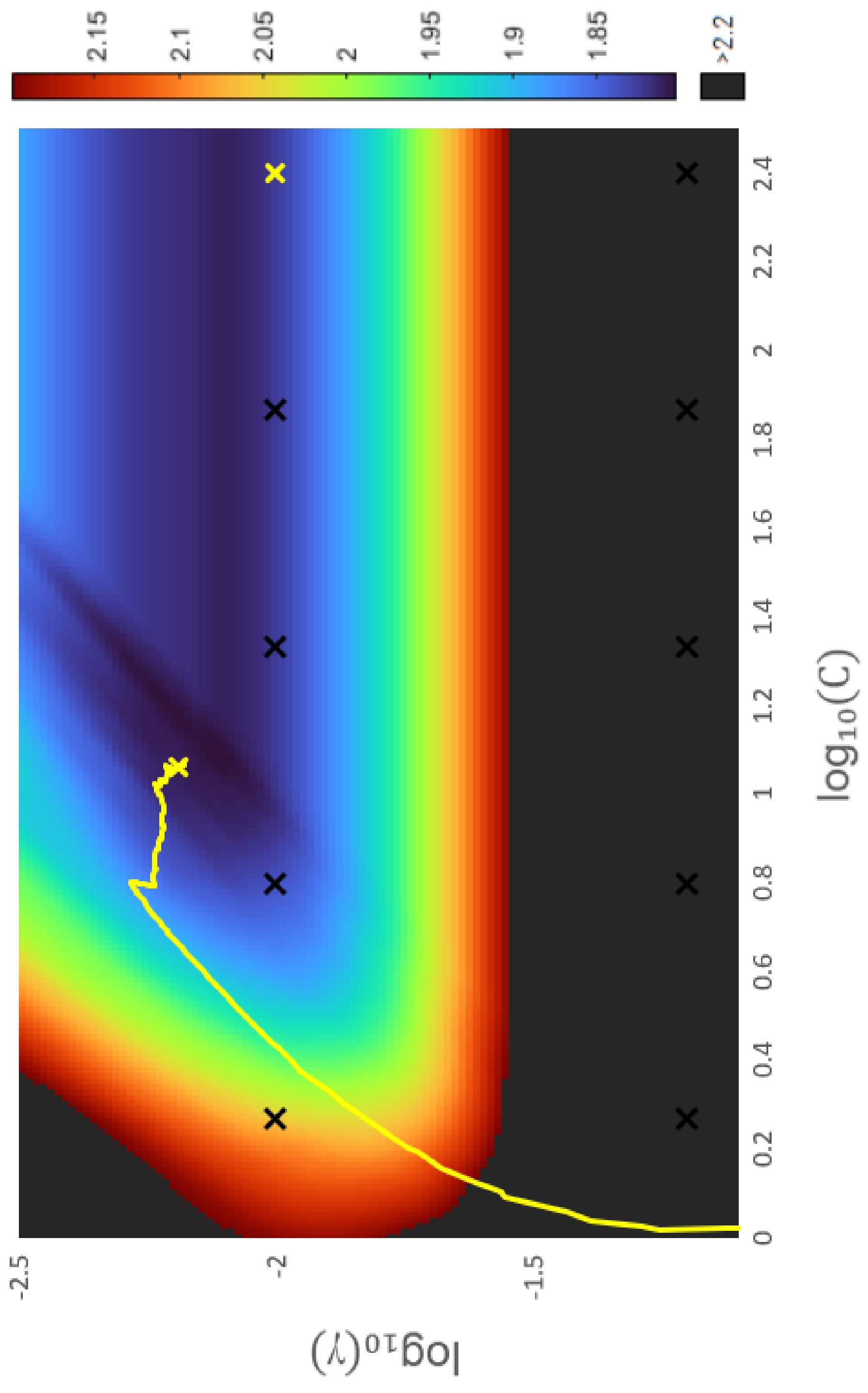

FIGURE 5.6: Figure comparing the performance of our method with that of grid-search for tuning hyperparameters for the Sonar data set. See Figure 5.3 for a description of this graph.

points tested by grid-search. It is worth noting that, with the exception of the glass data set, the best hyperparameter combination found by grid-search does not lie near the solution found by our algorithm but rather at more extreme values of γ and C. The Glass data set illustrates the exact behaviour which makes the bilevel approach conceptually preferable to grid-search. As shown in Table 5.1, the best hyperparameter combination found by grid-search for this data set is $\gamma = 10^{-1.2}$ and $C = 10^{1.3}$. As can be seen in Figure 5.4, this is the closest point tested by the grid-search to the apparent local minimum which the bilevel algorithm has correctly located. In other words, there exists a local minimum between the points tested by grid-search and the bilevel algorithm was able to navigate this space to approximately locate this local minimum.

The Banknote Authentication data set consists of 1372 data points comprised of 4 variables together with their associated labels (Lohweg, 2013) indicating whether the banknotes are genuine or fake. The same preprocessing was applied to this data set as was applied to the others. Figure 5.7 shows the path our bilevel hyperparameter tuning algorithm navigates the hyperparameter space on the banknote data set. As can be seen on the right, our model preforms as well as can be expected. It traverses the hyperparameter space to find what appears to be a local minimum. However, on the left we can see from the heatmap generated by the 16×16 grid-search that the solution found by grid-search here is better than that found by our algorithm. As can be seen the best hyperparameter combination grid-search was able to find is on the edge of the evaluated portion of the hyperparameter space far from the starting point of our algorithm. This outcome is to be expected for data sets where the best hyperparameter combination is very far from our methods starting point, as due to the non-convex nature of our problem, it is unlikely that our method can traverse the hyperparameter space to that point without finding a local minimum first.

5.5 Conclusions

In this chapter we have formulated the bilevel problem of hyperparameter selection for RBF kernel SVMs via k-fold CV. This has required the formulation of the dual form of the training problem and the construction of a smooth loss function for use in the upper level objective which uses only model parameters for the dual formulation of the training problem. We perform a KKT reformulation to derive a single level reformulation of the problem. We propose an algorithm for approximating the problem's solution using the Scholtes relaxation and we perform the required analysis of the MPEC MFCQs to show that this algorithm converges to a C-stationary point. Finally, we provide a number of illustrative examples of our algorithm in action, and demonstrate its apparent ability to locate local minima of the problem and outperform grid-search on some instances.

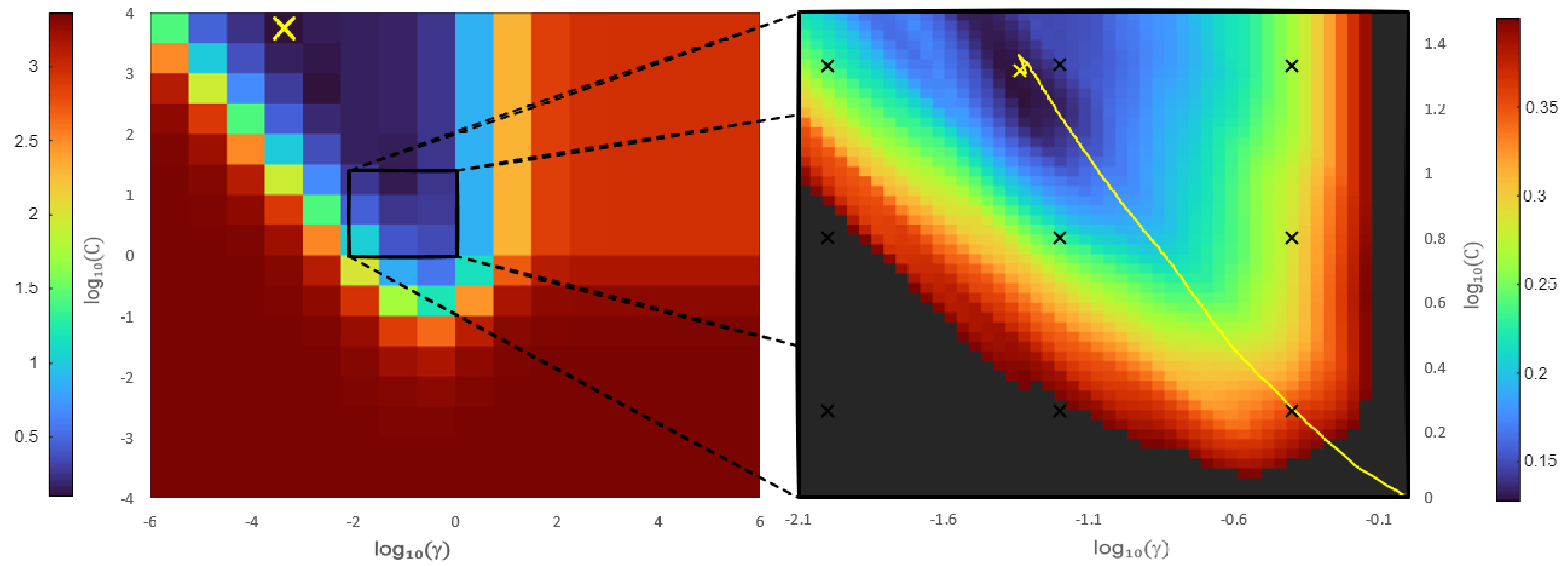

FIGURE 5.7: Figure comparing the performance of our method with that of grid-search. Left: The heatmap formed from the values of the objective function for the hyperparameter combination evaluated. Right: The path taken, shown in yellow, through the hyperparameter space by `fmincon` as it attempts to solve the bilevel problem. The yellow cross shows the solution found by the bilevel method. The black crosses show the points tested by a grid-search. The background heatmap was created by performing a very fine grid-search to evaluate the value of the upper level objective in this local region.

Our upper level objective function is non-covex and as such has numerous local minima. Locating a local minimum is all that can be expected from the methods we use to solve Problem 5.23. Figures 5.3, 5.4, 5.5 and 5.6 suggest that our proposed algorithm is capable of locating local minima. However, in some instances (such as for the banknote data set shown in Figure 5.7), while our method is seemingly able to successfully locate a local minimum, grid-search was able to find a hyperparameter combination far from the starting point used for our method which had a lower objective function value. In future works we intend to investigate potential solutions to this drawback, such as using different and multiple starting points and creating our own solver to leverage principals such as momentum.

Chapter 6

Final comments

In this work we have focused on supervised learning algorithms for prediction tasks. We optimise the processes by which these algorithms are used in particular applications. Specifically, we present optimised preprocessing methods and optimise hyperparameter selections. We also present our own optimisation model for simultaneously training and tuning the hyperparameters of non-linear kernel SVMs.

In Chapter 3, we have proposed a framework for modeling the problem of predicting infrequently occurring adverse events from time series data as an imbalanced classification problem. We have focused on two applications: predicting the occurrence foaming in Anaerobic Digestion (AD) and predicting condenser tube leaks in civil Nuclear Power Production (NPP). Central to this framework, we propose block sampling a method of informed undersampling which, as we demonstrate, outperforms other sampling techniques for addressing class imbalance in our two applications. Our extensive set of experiments have revealed that adopting our proposed framework results in models which achieve a good performance in predicting adverse events in both of our applications.

Using our proposed framework for comparing a range of classification algorithms, sampling techniques and hyperparameter selections for the AD application, the best performing models were capable of accurately preempting each of the 5 foaming events in our time series with up to 24 hours warning and an average balanced accuracy of 88.6%, having been trained using data relating to the remaining 4. For the NPP application, using our proposed framework we were able to produce models capable of accurately preempting 4 of the 6 tube leakage events, having been trained on the remaining 5, with up to 12 hours warning and an average balanced accuracy of 76.2%. The models developed in this chapter could be integrated into a decision support tool for providing advanced warning of the occurrence adverse events. In the AD application this warning would allow plant operators to administer anti-foaming agent, thus averting the foaming. In the NPP application this warning would allow the plant operators to reduce the load on the turbines and repair the leak without the turbine being tripped. In both of these applications the capabilities provided by such a tool would lead to significant commercial benefits. The main difficulty faced in these analyses is the lack of data relating to the adverse events. As such, were additional data be collected, we would expect to be able to predict these adverse events with even greater accuracy.

In Chapter 4, we propose an AI tool for automated, prolonged ECG screening to replace the current screening process where a patient's T:R ratio is measured over a 10-second period. We have collected the SGH-ECG data set (detailed in section 4.3.1) consisting of 10-second ECG segments from patients belonging to a range of relevant patient groups. We propose a detailed preprocessing scheme utilising signal preprocessing methods together with creating phase space reconstructions to enable our models to more accurately predict T:R ratios. We have conducted an extensive set of experiments wherein we evaluate a range of deep learning model architectures, hyperparameter combinations and Stochastic Gradient Descent (SGD) based optimisation algorithms for model training. We find that ensemble models comprised of MLP5 models, trained using the best-performing SGD based optimisation method and with a tuned batch size and global learning rate gave the best performance, being capable of predicting the T:R ratios of 10-second segments to within 0.0461 of their true values. We have shown how this model can be integrated into a clinical tool enabling clinicians to perform 24-hours automated screenings, thus providing a much more reliable indication of the normal behaviours of the patient's T:R ratio over time. This increased reliability and descriptiveness can allow cardiologists to better determine patients' risk of TWOS and hence their eligibility for S-ICD implantation.

In Chapter 5 we formulate the bilevel problem of hyperparameter selection for RBF kernel SVMs via k-fold CV and propose an algorithms for solving it. To do this we have constructed a smooth loss function for use in the upper level objective, based off of the lower level training objective and formulated using only parameters of the dual problem. We then perform a single level reformulation and propose an algorithm for approximating the problems solution by iteratively using the Scholtes relaxation. We perform the necessary analysis of the MPEC MFCQs to show that this algorithm converges to a C-stationary point. Lastly, we give a number of illustrative examples showing that our algorithm works on real data sets from the UCI repository. We observe our algorithm's apparent ability to locate local minima of the problem and outperform grid-search on some instances.

The work on predicting adverse events in low-carbon energy production, presented in Chapter 3, has led to the following paper:

Coniglio, S., Dunn, A. J., and Zemkoho, A. B. (2020). Infrequent adverse event prediction in low carbon energy production using machine learning. Preprint available at arXiv:2001.06916. *Under review in* Machine Learning.

The work in Chapter 4 in developing an AI tool for screening patients' S-ICD implantation eligibility has led to a paper presenting the development of the tool:

Dunn, A. J., ElRefai, M. H., Roberts, P. R., Coniglio, S., Wiles, B. M., and Zemkoho, A. B. (2021). Deep learning methods for screening patients' S-ICD implantation eligibility. Artificial Intelligence in Medicine*, 119:102139.*

Two papers which use our tool to assess what the optimal choice of the T:R ratio cut-off used in the screening should be:

ElRefai, M., Abouelasaad, M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., Wiles, B. M., and Roberts, P. R. (2022b). Eligibility for subcutaneous implantable cardiac defibrillator utilising artificial intelligence and deep learning methods for prolonged screening: where is the cut-off? Europace*, 24 (Supplement_1):euac053–447.*

ElRefai, M., Abouelasaad, M., Wiles, B. M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., and Roberts, P. R. (2022c). Deep learning-based insights on T:R ratio behaviour during prolonged screening for S-ICD eligibility. Journal of Interventional Cardiac Electrophysiology, doi.org/10.1007/s10840-022-01245-6.

A paper using our tool to evaluate the variation of the T:R ratio of patients with various heart conditions:

ElRefai, M., Abouelasaad, M., Conibear, I., Wiles, B. M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., and Roberts, P. R. (2022a). The use of artificial intelligence and deep learning methods in subcutaneous implantable cardioverter defibrillator screening to optimise selection in special patient populations. Europace*, 24(Supplement_1):euac053–448.*

A further paper detailing the hyperparameter optimisation and comparison of optimisation algorithms for model training

Dunn, A. J., ElRefai, M. H., Roberts, P. R., Coniglio, S., Wiles, B. M., and Zemkoho, A. B. (2022). Deep learning and hyperparameter optimization for assessing one's eligibility for a subcutaneous implantable cardioverter-defibrillator. Preprint available at https://optimization-online.org/?p=21066. *Under review in* Annals of Operations Research.

The work in Chapter 5 has led to a paper, which is under preparation (note that this paper will be a significant extension of the work contained in this chapter):

Stefano Coniglio, Anthony J Dunn, Qingna Li, and Alain B Zemkoho, Bilevel hyperparameter optimisation for non-linear support vector machines. To be submitted to Mathematical Programming *by the end of* 2022.

During the course of my PhD I have also supervised two masters students whose work is being developed into publications. Hanyu Wang's project:

Hanyu Wang, Emmanuel K. Tsinda, Anthony J. Dunn, Francis Chikweto, Nusreen Ahmed, Emanuela Pelosi, and Alain B. Zemkoho. Deep learning forward and reverse primer design to detect SARS-CoV-2 emerging variants. Under review in Bioinformatics.,

And Tianjie Gu's project:

Tianjie Gu, Emmanuel Kagning Tsinda, Anthony Dunn, Adithya Madhusoodanan, and Alain B. Zemkoho. Descriptive analysis of COVID-19 vaccine adopters and hesitant among social media users. Currently being prepared for publication.

Appendix A

Chapter 3 appendix

A.1 Undersampling and oversampling implementation

Sampling techniques were incorporated into the experiment process at each iteration of CV by undersampling or oversampling the already block sampled training folds before training the classifiers and testing them on the block sampled testing fold. When undersampling, we sample, with replacement, a set from our majority class of the same cardinality as our minority class, whereas, when oversampling, we sample, with replacement, a set from the minority class that has the same cardinality as the majority class. This is implemented using the functions `RandomUnderSampler` and `RandomOverSampler` from the Python package `Imbalanced-learn`.

A.2 Hyperparameter tuning details

In most experiments in Subsections 3.3.3 and 3.3.4, we trained classifiers using the default hyperparameter selections used by the classifier implementations in `scikit-learn` (Pedregosa et al., 2011). When we use grid-search to tune the hyperparameters of classification algorithms, we use a grid of roughly 100 combinations of hyperparameter values. As was shown in Subsection 3.3.2, in both of our application it is advantageous to include event patterns in the training of our models, while excluding them from testing. This same reasoning extends to performing grid-search via $(k-1)$-fold CV nested within each iteration of a k-fold CV for model selection. The models trained using any particular hyperparameter combination should not be assessed on event patterns. This need to include event patterns in folds when they are used for training yet exclude event patterns when the same fold is used for testing means that it is not possible to use the `scikit-learn` function `GridSearchCV` for hyperparameter tuning. Instead, we use the method laid out in Algorithm 3.

For the SVM classifier, we search over the values of two hyperparameters, the regularisation parameter (**C**) and the kernel coefficient (γ). We test values 10^n with $n \in \{-6, ..., 3\}$ for **C** and values $\frac{1}{d\,(\tau+1)}$ (where $d\,(\tau + 1)$ is the number of features of our data) and 10^n with $n \in \{-6, ..., 2\}$ for γ.

For the GNB classifier, we only tune the variance smoothing hyperparameter by searching over a logarithmic range of 100 values from 10^{-15} to 10^0.

For the BRF classifier, we tune three hyperparameters: the number of trees in the forest (n-estimators), the number of features to consider when looking for the best split at each node of the decision trees (max-features), and the sampling strategy used to balance the data for each tree. We test values of $\{10, 25, 50, 100, 200, 300, 400, 500, 750, 1000\}$ for n-estimators, and of $\{2, 4, 8, 16, 32\}$ for max-features, and we test under and oversampling for balancing the data for each tree.

For the AB classifier, we tune the maximum number of estimators in the ensemble, as well as the learning rate. We test values of $\{10, 20, 30, 40, 50, 75, 100, 150, 200, 500, 1000, 2000\}$ for the maximum number of estimators and learning rates of 10^n with $n \in \{-7, ..., 0\}$.

A.3 Full experimental results

In Tables A.1 and A.2, we refer to basic partitioning as BP, random undersampling as RUS, random oversampling as ROS, artificial oversampling (SMOTE) as AOS and block sampling as BS. In Tables A.4 and A.7, we use NS to denote that a classifier was trained without the use of undersampling or oversampling, and US or OS denote that undersampling or oversampling, respectively, was used to balance the training set. In Tables A.5 and A.8, classifiers with tuned hyperparameters are markded with an asterisk whereas classifiers using default hyperparmeters are not. See Subsection 3.3.1 for definitions of the classifier name acronyms.

TABLE A.1: Foaming in AD: Comparison of fold sampling techniques

Event Inclusion	Sampling	Classifier											
Inclusion	Technique	SVM	RF	BRF	MLP	LR	AB	KNN	DT	GNB	QDA	GB	Aver.
	BP	0.53	0.50	0.71	0.55	0.54	0.54	0.50	0.51	0.84	0.51	0.51	0.567
	RUS	0.78	0.54	0.68	**0.72**	0.60	**0.61**	0.51	0.52	**0.85**	0.51	0.51	0.621
Excluding	ROS	**0.90**	0.55	0.55	0.57	**0.88**	0.56	0.54	0.56	**0.85**	0.51	0.51	0.636
	AOS	**0.90**	**0.58**	0.58	0.58	0.87	0.59	**0.56**	0.67	**0.85**	0.51	0.51	**0.656**
	BS	0.71	0.55	**0.73**	0.68	0.64	0.58	0.52	**0.70**	0.83	**0.54**	**0.54**	0.637
	BP	0.65	0.54	0.68	0.56	0.57	0.56	0.50	0.55	0.84	0.59	0.59	0.602
	RUS	0.78	0.56	0.68	0.64	0.62	0.60	0.51	0.59	0.86	0.60	0.60	0.639
Including	ROS	**0.90**	0.55	0.56	0.57	**0.92**	**0.64**	0.53	0.60	0.86	0.62	0.62	0.671
	AOS	0.87	0.58	0.59	0.57	0.90	0.61	**0.55**	0.60	0.86	0.56	0.56	0.661
	BS	0.75	**0.62**	**0.74**	**0.67**	0.65	**0.64**	0.52	**0.70**	**0.87**	**0.63**	**0.63**	**0.675**

TABLE A.2: Condenser tube leakage: Comparison of fold sampling techniques

Event Inclusion	Sampling	Classifier											
or Exclusion	Technique	SVM	RF	BRF	MLP	LR	AB	KNN	DT	GNB	QDA	GB	Aver.
Excluding	BP	0.50	0.50	0.65	0.50	0.46	0.50	0.50	0.50	0.50	0.50	0.50	0.509
	RUS	0.51	0.50	0.66	0.51	0.46	0.51	0.50	**0.51**	0.50	0.50	0.50	0.514
	ROS	0.57	0.50	0.50	0.50	0.39	0.50	0.49	0.50	0.50	0.50	0.50	0.496
	AOS	**0.58**	0.50	0.50	0.50	0.40	0.51	0.49	**0.51**	0.51	0.50	0.50	0.499
	BS	0.57	**0.56**	**0.69**	**0.54**	**0.52**	**0.68**	**0.52**	0.48	**0.54**	**0.52**	**0.52**	**0.559**
Including	BP	0.50	0.50	0.65	0.50	0.46	0.50	0.50	0.51	0.50	0.50	0.50	0.511
	RUS	0.59	0.50	0.64	0.58	0.46	0.54	0.51	**0.57**	0.50	0.50	0.50	0.535
	ROS	0.56	0.50	0.50	0.51	0.41	0.54	0.52	0.50	0.50	0.50	0.50	0.502
	AOS	0.56	0.50	0.51	0.50	0.43	0.50	0.51	0.50	0.51	0.50	0.50	0.502
	BS	**0.62**	**0.67**	**0.71**	**0.68**	**0.54**	**0.67**	**0.56**	0.47	**0.52**	**0.56**	**0.56**	**0.597**

TABLE A.3: Foaming in AD: 5-fold CV for classifier comparison

ω	SVM	BRF	MLP	LR	AB	KNN	DT	GNB	QDA	GB
12	0.843	0.757	0.768	0.688	0.742	0.541	0.753	**0.875**	0.627	0.747
24	0.759	0.775	0.695	0.658	0.715	0.536	0.733	**0.873**	0.596	0.713
36	0.751	0.743	0.710	0.686	0.665	0.560	0.666	**0.784**	0.620	0.674
48	0.734	0.722	0.640	0.667	0.590	0.555	0.599	**0.746**	0.605	0.620
96	0.613	0.628	0.635	0.591	0.539	0.591	0.588	**0.671**	0.622	0.579

TABLE A.4: Foaming in AD: 5-fold CV sampling technique comparison

ω	SVM			BRF			GNB		
	NS	OS	US	NS	OS	US	NS	OS	US
12	0.843	0.844	**0.866**	**0.757**	0.742	0.754	0.875	**0.880**	0.876
24	0.759	**0.854**	0.821	**0.775**	0.629	0.772	0.873	**0.886**	0.876
36	0.751	**0.839**	0.829	**0.743**	0.634	0.732	0.784	**0.813**	0.811
48	0.734	0.776	**0.791**	**0.722**	0.606	0.693	0.746	**0.761**	0.756
96	0.613	0.536	**0.641**	0.628	0.593	**0.633**	**0.671**	0.670	0.663

TABLE A.5: Foaming in AD: Tuned vs not tuned classifiers. Classifiers with tuned hyperparameters are marked with an asterisk.

ω	SVM	SVM*	BRF	BRF*	GNB	GNB*
4	**0.843**	0.830	**0.757**	0.692	0.875	**0.882**
8	0.759	**0.795**	**0.775**	0.729	0.873	**0.877**
12	0.751	**0.806**	**0.743**	0.733	0.784	**0.822**
16	0.734	**0.778**	0.722	**0.735**	0.746	**0.776**
32	0.613	**0.667**	**0.628**	0.609	0.671	**0.674**

TABLE A.6: Condenser tube leakage: 6-fold CV for classifier comparison

ω	SVM	BRF	MLP	LR	AB	KNN	DT	GNB	QDA	GB
4	0.575	0.603	0.577	0.535	**0.608**	0.501	0.390	0.577	0.500	0.502
8	0.633	0.549	0.599	0.577	**0.644**	0.523	0.455	0.520	0.500	0.564
12	**0.634**	0.545	0.572	0.534	0.578	0.517	0.441	0.585	0.500	0.547
16	0.573	0.528	0.552	0.545	0.510	0.503	0.515	**0.609**	0.520	0.490
32	0.560	0.537	0.545	0.555	0.507	0.520	0.555	**0.597**	0.553	0.499

TABLE A.7: Condenser tube leakage: 6-fold CV sampling technique comparison.

ω	SVM			AB			GNB		
	NS	OS	US	NS	OS	US	NS	OS	US
4	0.575	**0.762**	0.532	**0.608**	0.556	0.605	0.577	**0.652**	0.601
8	0.633	**0.674**	0.607	0.644	**0.652**	0.605	0.520	**0.570**	0.485
12	0.634	**0.644**	0.580	**0.578**	0.566	0.550	0.585	**0.594**	0.568
16	0.573	**0.611**	0.535	0.510	**0.554**	0.512	0.609	**0.615**	0.597
32	0.560	**0.577**	0.554	0.507	**0.588**	0.497	0.597	**0.611**	0.599

TABLE A.8: Condenser tube leakage: Tuned vs not tuned classifiers. Classifiers with tuned hyperparameters are marked with an asterisk.

ω	SVM	SVM*	AB	AB*	GNB	GNB*
4	0.575	**0.576**	**0.608**	0.584	0.577	**0.588**
8	**0.633**	0.573	**0.644**	0.614	0.520	**0.587**
12	0.634	0.634	**0.578**	0.558	**0.585**	0.580
16	0.573	**0.577**	0.510	**0.519**	**0.609**	0.578
32	**0.560**	0.473	**0.507**	0.496	**0.597**	0.522

Appendix B

Chapter 4 appendix

B.1 Additional experiment figures

In this section, we provide additional figures from the experiments reported in Section 4.3. Figure B.1 shows the training time and number of epochs before early stopping for the various SGD variants evaluated in Subsection 4.3.5, Figure B.2 shows the training time and number of epochs before early stopping for the various batch sizes evaluated in Subsection 4.3.6, and Figure B.3 shows the training time and number of epochs before early stopping for the various global learning rates evaluated in Subsection 4.3.7.

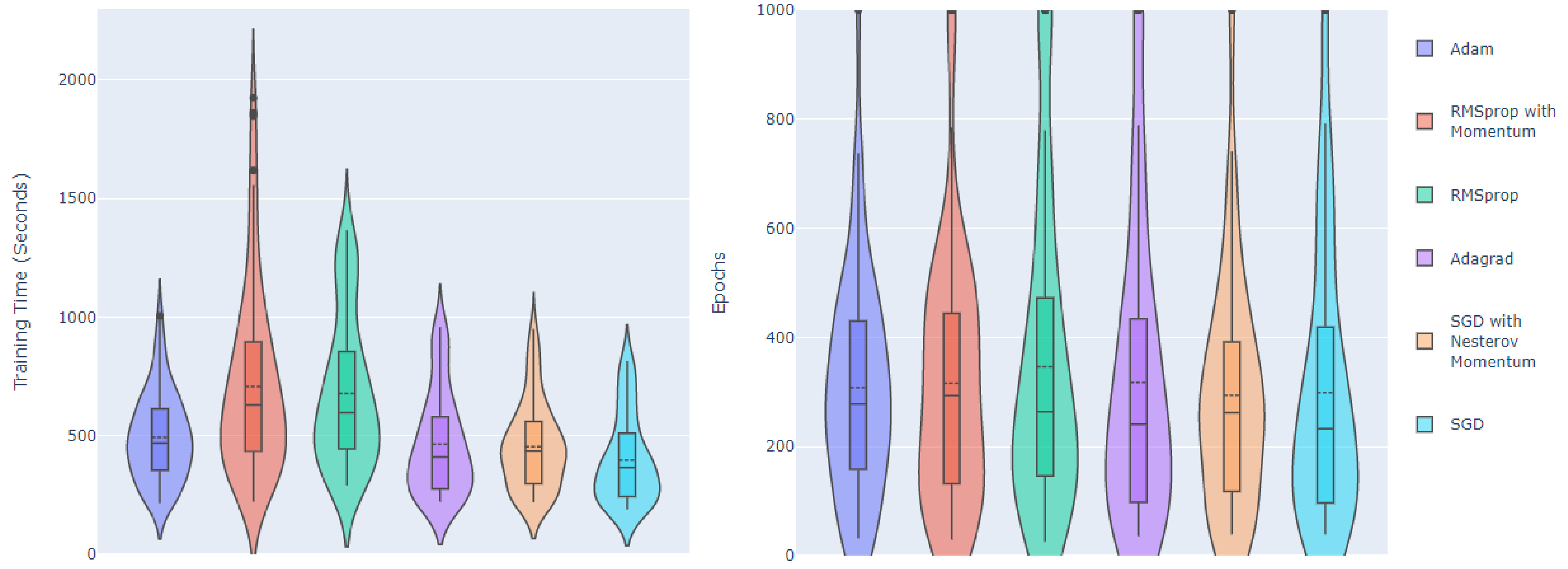

FIGURE B.1: Left: Violin plots showing the distribution of the training time of the 100 MLP5 models trained with each SGD variant evaluated in the 10 rounds of 10-fold CV. Right: Violin plots showing the distribution of the number of epochs required to train the 100 MLP5 models trained with each SGD variant evaluated in the 10 rounds of 10-fold CV.

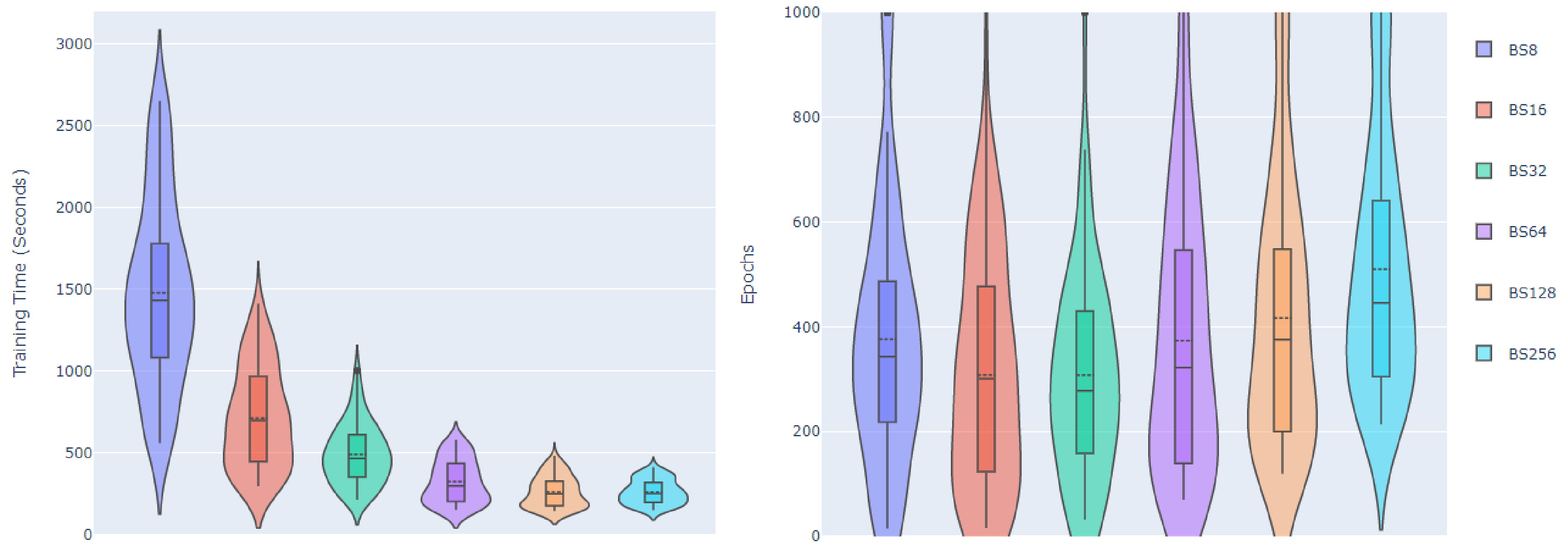

FIGURE B.2: Left: Violin plots showing the distribution of the training time of the 100 MLP5 models trained with each batch size evaluated in the 10 rounds of 10-fold CV. Right: Violin plots showing the distribution of the number of epochs required to train the 100 MLP5 models trained with each batch size evaluated in the 10 rounds of 10-fold CV.

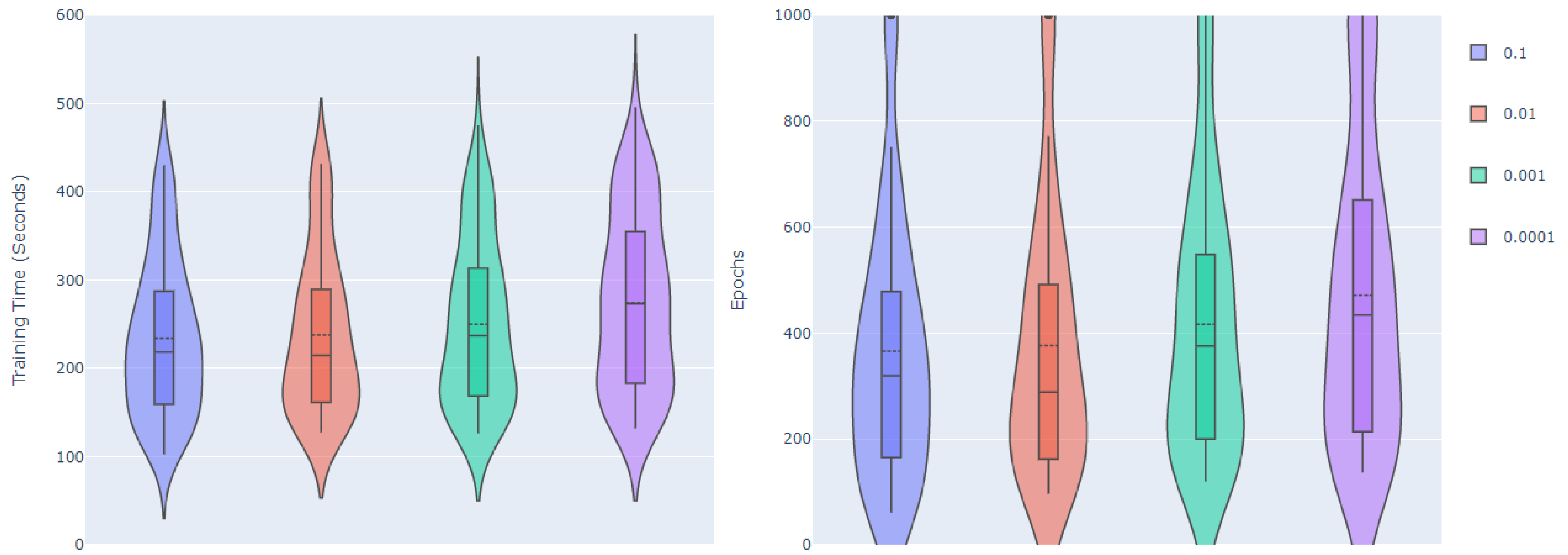

FIGURE B.3: Left: Violin plots showing the distribution of the training time of the 100 MLP5 models trained with each global learning rate evaluated in the 10 rounds of 10-fold CV. Right: Violin plots showing the distribution of the number of epochs required to train the 100 MLP5 models trained with each global learning rate evaluated in the 10 rounds of 10-fold CV.

References

Abd Elrahman, S. M. and Abraham, A. (2013). A review of class imbalance problem. *Journal of Network and Innovative Computing*, 1(2013):332–340.

Adabag, S. A., Luepker, R. V., Roger, V. L., and Gersh, B. J. (2010). Sudden cardiac death: epidemiology and risk factors. *Nature Reviews Cardiology*, 7(4):216–225.

Akbani, R., Kwek, S., and Japkowicz, N. (2004a). Applying support vector machines to imbalanced datasets. In *European Conference on Machine Learning*, pages 39–50. Springer.

Akbani, R., Kwek, S., and Japkowicz, N. (2004b). Applying support vector machines to imbalanced datasets. In *European Conference on Machine Learning*, pages 39–50. Springer.

Alpaydin, E. (2016). *Machine learning: the new AI*. MIT Press.

Alpaydin, E. (2020). *Introduction to machine learning*. MIT Press.

Assanelli, D., Di Castelnuovo, A., Rago, L., Badilini, F., Vinetti, G., Gianfagna, F., Salvetti, M., Zito, F., Donati, M. B., De Gaetano, G., et al. (2013). T-wave axis deviation and left ventricular hypertrophy interaction in diabetes and hypertension. *Journal of Electrocardiology*, 46(6):487–491.

Babu, G. S., Zhao, P., and Li, X.-L. (2016). Deep convolutional neural network based regression approach for estimation of remaining useful life. In *International Conference on Database Systems for Advanced Applications*, pages 214–228. Springer.

Balduzzi, D., Frean, M., Leary, L., Lewis, J., Ma, K. W.-D., and McWilliams, B. (2017). The shattered gradients problem: If resnets are the answer, then what is the question? In *International Conference on Machine Learning*, pages 342–350. PMLR.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, Berlin/Heidelberg, Germany.

Bengio, Y., Goodfellow, I., and Courville, A. (2017). *Deep learning*, volume 1. MIT Press Massachusetts, USA, Cambridge, Massachussets and London, England.

Bennett, K. P., Hu, J., Ji, X., Kunapuli, G., and Pang, J.-S. (2006). Model selection via bilevel optimization. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 1922–1929. IEEE.

Bennett, K. P., Kunapuli, G., Hu, J., and Pang, J.-S. (2008). Bilevel optimization and machine learning. In *IEEE World Congress on Computational Intelligence*, pages 25–47. Springer.

Biau, G. and Scornet, E. (2016). A random forest guided tour. *Test*, 25(2):197–227.

Bishop, C. M. (2006). Pattern recognition. *Machine learning*, 128(9).

Bishop, C. M. et al. (1995). *Neural networks for pattern recognition*. Oxford University Press, Cambridge, Massachussets and London, England.

Biswas, K., Kumar, S., Banerjee, S., and Pandey, A. K. (2021). Smu: Smooth activation function for deep networks using smoothing maximum technique. *arXiv preprint arXiv:2111.04682*.

Boersma, L., Barr, C., Knops, R., Theuns, D., Eckardt, L., Neuzil, P., Scholten, M., Hood, M., Kuschyk, J., Jones, P., et al. (2017). Implant and midterm outcomes of the subcutaneous implantable cardioverter-defibrillator registry: the effortless study. *Journal of the American College of Cardiology*, 70(7):830–841.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152.

Botchkarev, A. (2018). Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *arXiv preprint arXiv:1809.03006*.

Branco, P., Torgo, L., and Ribeiro, R. P. (2016). A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys (CSUR)*, 49(2):1–50.

Cattant, F., Crusset, D., and Féron, D. (2008). Corrosion issues in nuclear industry today. *Materials Today*, 11(10):32–37.

Chang, D.-W., Lin, C.-S., Tsao, T.-P., Lee, C.-C., Chen, J.-T., Tsai, C.-S., Lin, W.-S., and Lin, C. (2021). Detecting digoxin toxicity by artificial intelligence-assisted electrocardiography. *International Journal of Environmental Research and Public Health*, 18(7):3839.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357.

Chawla, N. V., Japkowicz, N., and Kotcz, A. (2004). Special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter*, 6(1):1–6.

Chen, M., Fang, Y., and Zheng, X. (2014). Phase space reconstruction for improving the classification of single trial eeg. *Biomedical Signal Processing and Control*, 11:10–16.

Cho, J., Lee, B., Kwon, J.-M., Lee, Y., Park, H., Oh, B.-H., Jeon, K.-H., Park, J., and Kim, K.-H. (2021). Artificial intelligence algorithm for screening heart failure with reduced ejection fraction using electrocardiography. *ASAIO Journal*, 67(3):314–321.

Cho, Y., Kwon, J.-m., Kim, K.-H., Medina-Inojosa, J. R., Jeon, K.-H., Cho, S., Lee, S. Y., Park, J., and Oh, B.-H. (2020). Artificial intelligence algorithm for detecting myocardial infarction using six-lead electrocardiography. *Scientific Reports*, 10(1):1–10.

Coit, R. L. (1980). Assessment of condenser leakage problems, technical planning study tps 79-729. Technical report, Electric Power Research Institute.

Coniglio, S., Dunn, A. J., and Zemkoho, A. B. (2020). Infrequent adverse event prediction in low carbon energy production using machine learning. *Preprint available at arXiv:2001.06916*.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Dalmau, J., Comas, J., Rodríguez-Roda, I., Latrille, E., and Steyer, J.-P. (2010a). Selecting the most relevant variables for anaerobic digestion imbalances: two case studies. *Water Environment Research*, 82(6):492–498.

Dalmau, J., Comas, J., Rodríguez-Roda, I., Pagilla, K., and Steyer, J.-P. (2010b). Model development and simulation for predicting risk of foaming in anaerobic digestion systems. *Bioresource Technology*, 101(12):4306–4314.

Daniels, D. G. (2010). Taming condenser tube leaks, part i. *Power Magazine*, 154(9):56–59.

Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine learning*, pages 233–240.

Dawid, A. (2019). Psr-based research of feature extraction from one-second eeg signals: a neural network study. *SN Applied Sciences*, 1(12):1–12.

De Los Reyes, F. L. (2010). Foam in wastewater treatment facilities. In *Handbook of Hydrocarbon and Lipid Microbiology*. Springer.

Del Buono, N., Esposito, F., and Selicato, L. (2020). Methods for hyperparameters optimization in learning approaches: an overview. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 100–112. Springer.

Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J.-J., Sandhu, S., Guppy, K. H., Lee, S., and Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. *The American Journal of Cardiology*, 64(5):304–310.

Djemal, R., Bazyed, A. G., Belwafi, K., Gannouni, S., and Kaaniche, W. (2016). Three-class eeg-based motor imagery classification using phase-space reconstruction technique. *Brain Sciences*, 6(3):36.

Drucker, H. and Cortes, C. (1995). Boosting decision trees. *Advances in Neural Information Processing Systems*, 8.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7).

Dunn, A. J., ElRefai, M. H., Roberts, P. R., Coniglio, S., Wiles, B. M., and Zemkoho, A. B. (2021). Deep learning methods for screening patients' S-ICD implantation eligibility. *Artificial Intelligence in Medicine*, 119:102139.

Dunn, A. J., ElRefai, M. H., Roberts, P. R., Coniglio, S., Wiles, B. M., and Zemkoho, A. B. (2022). Deep learning and hyperparameter optimization for assessing one's eligibility for a subcutaneous implantable cardioverter-defibrillator. *Preprint available at https://optimization-online.org/?p=21066*.

ElRefai, M., Abouelasaad, M., Conibear, I., Wiles, B. M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., and Roberts, P. R. (2022a). The use of artificial intelligence and deep learning methods in subcutaneous implantable cardioverter defibrillator screening to optimise selection in special patient populations. *Europace*, 24(Supplement_1):euac053–448.

ElRefai, M., Abouelasaad, M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., Wiles, B. M., and Roberts, P. R. (2022b). Eligibility for subcutaneous implantable cardiac defibrillator utilising artificial intelligence and deep learning methods for prolonged screening: where is the cut-off? *Europace*, 24 (Supplement_1):euac053–447.

ElRefai, M., Abouelasaad, M., Wiles, B. M., Dunn, A. J., Coniglio, S., Zemkoho, A. B., and Roberts, P. R. (2022c). Deep learning-based insights on T:R ratio behaviour during prolonged screening for S-ICD eligibility. *Journal of Interventional Cardiac Electrophysiology,* doi.org/10.1007/s10840-022-01245-6.

Estabrooks, A., Jo, T., and Japkowicz, N. (2004). A multiple resampling method for learning from imbalanced data sets. *Computational Intelligence*, 20(1):18–36.

Evett, I. W. and Spiehler, E. J. (1987). Rule induction in forensic science. In *KBS in Government*, pages 107–118. Online Publications.

Fan, X., Yao, Q., Cai, Y., Miao, F., Sun, F., and Li, Y. (2018). Multiscaled fusion of deep convolutional neural networks for screening atrial fibrillation from single lead short ECG recordings. *IEEE Journal of Biomedical and Health Informatics*, 22(6):1744–1753.

Feizizadeh, B., Roodposhti, M. S., Blaschke, T., and Aryal, J. (2017). Comparing gis-based support vector machine kernel functions for landslide susceptibility mapping. *Arabian Journal of Geosciences*, 10(5):1–13.

Fernandes, L. M. S. (2014). Modeling anaerobic digestion with artificial neural networks. Technical report, Instituto Superior Tecnico, Universidade de Lisboa.

Fix, E. and Hodges, J. L. (1951). Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247.

Flegel, M. L. (2005). *Constraint qualifications and stationarity concepts for mathematical programs with equilibrium constraints*. PhD thesis, Universität Würzburg.

Fosbøl, E. L., Seibæk, M., Brendorp, B., Torp-Pedersen, C., Køber, L., Investigations, D., et al. (2008). Prognostic importance of change in qrs duration over time associated with left ventricular dysfunction in patients with congestive heart failure: the diamond study. *Journal of Cardiac Failure*, 14(10):850–855.

Fu, W. J. (1998). Penalized regressions: the bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416.

Gaida, D., Wolf, C., Meyer, C., Stuhlsatz, A., Lippel, J., Bäck, T., Bongards, M., and McLoone, S. (2012). State estimation for anaerobic digesters using the adm1. *Water Science and Technology*, 66(5):1088–1095.

Ganidi, N., Tyrrel, S., and Cartmell, E. (2009). Anaerobic digestion foaming causes–a review. *Bioresource Technology*, 100(23):5546–5554.

Garbin, C., Zhu, X., and Marques, O. (2020). Dropout vs. batch normalization: an empirical study of their impact to deep learning. *Multimedia Tools and Applications*, pages 1–39.

Garrett, D., Peterson, D. A., Anderson, C. W., and Thaut, M. H. (2003). Comparison of linear, nonlinear, and feature selection methods for eeg signal classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 11(2):141–144.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323. JMLR Workshop and Conference Proceedings.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press, Cambridge, Massachusetts.

Gorman, P. R. and Sejnowski, T. J. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1(1):75–89.

Guo, X., Yin, Y., Dong, C., Yang, G., and Zhou, G. (2008). On the class imbalance problem. In *2008 Fourth International Conference on Natural Computation*, volume 4, pages 192–201. IEEE.

Hajian-Tilaki, K. (2013). Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation. *Caspian Journal of Internal Medicine*, 4(2):627.

Hansen, J. V., McDonald, J. B., and Nelson, R. D. (2006). Some evidence on forecasting time-series with support vector machines. *Journal of the Operational Research Society*, 57(9):1053–1063.

Hazinski, M. F., Idris, A. H., Kerber, R. E., Epstein, A., Atkins, D., Tang, W., and Lurie, K. (2005). Lay rescuer automated external defibrillator (“public access defibrillation”) programs: lessons learned from an international multicenter trial: advisory statement from the american heart association emergency cardiovascular committee; the council on cardiopulmonary, perioperative, and critical care; and the council on clinical cardiology. *Circulation*, 111(24):3336–3340.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., and Scholkopf, B. (1998). Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28.

Hoheisel, T., Kanzow, C., and Schwartz, A. (2013). Theoretical and numerical comparison of relaxation methods for mathematical programs with complementarity constraints. *Mathematical Programming*, 137(1):257–288.

Hong, H., Pradhan, B., Bui, D. T., Xu, C., Youssef, A. M., and Chen, W. (2017). Comparison of four kernel functions used in support vector machines for landslide susceptibility mapping: a case study at suichuan area (china). *Geomatics, Natural Hazards and Risk*, 8(2):544–569.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR.

Jabeur, S. B., Sadaaoui, A., Sghaier, A., and Aloui, R. (2020). Machine learning models and cost-sensitive decision trees for bond rating prediction. *Journal of the Operational Research Society*, 71(8):1161–1179.

Japkowicz, N. (2000). The class imbalance problem: Significance and strategies. In *Proc. of the Int'l Conf. on Artificial Intelligence*, volume 56. Citeseer.

Japkowicz, N. and Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449.

Jo, Y.-Y., Cho, Y., Lee, S. Y., Kwon, J.-m., Kim, K.-H., Jeon, K.-H., Cho, S., Park, J., and Oh, B.-H. (2021). Explainable artificial intelligence to detect atrial fibrillation using electrocardiogram. *International Journal of Cardiology*, 328:104–110.

Kanu, I. R., Aspray, T. J., and Adeloye, A. (2015). Understanding and predicting foam in anaerobic digester. *International Journal of Biological, Biomolecular, Agricultural, Food and Biotechnological Engineering*, 9(10):1056–1060.

Kanu, I. R. et al. (2018). *Biological investigation and predictive modelling of foaming in anaerobic digester*. PhD thesis, Heriot-Watt University.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kiranyaz, S., Ince, T., and Gabbouj, M. (2015). Real-time patient-specific ECG classification by 1-d convolutional neural networks. *IEEE Transactions on Biomedical Engineering*, 63(3):664–675.

Kleinbaum, D. G., Dietz, K., Gail, M., Klein, M., and Klein, M. (2002). *Logistic regression*. Springer.

Knops, R. E., Olde Nordkamp, L. R., Delnoy, P.-P. H., Boersma, L. V., Kuschyk, J., El-Chami, M. F., Bonnemeier, H., Behr, E. R., Brouwer, T. F., Kääb, S., et al. (2020). Subcutaneous or transvenous defibrillator therapy. *New England Journal of Medicine*, 383(6):526–536.

Kozik, R. and Choraś, M. (2016). Solution to data imbalance problem in application layer anomaly detection systems. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 441–450. Springer.

Kramer, O. (2013). K-nearest neighbors. In *Dimensionality Reduction with Unsupervised Nearest Neighbors*, pages 13–23. Springer.

Krishnan, S. M., Dutt, D. N., Chan, Y., and Anantharaman, V. (2007). Phase space analysis for cardiovascular signals. In *Advances in Cardiac Signal Processing*, pages 339–354. Springer, Berlin, Germany.

Kunapuli, G., Bennett, K. P., Hu, J., and Pang, J.-S. (2008). Classification model selection via bilevel programming. *Optimization Methods & Software*, 23(4):475–489.

Kusumoto, F. M., Bailey, K. R., Chaouki, A. S., Deshmukh, A. J., Gautam, S., Kim, R. J., Kramer, D. B., Lambrakos, L. K., Nasser, N. H., and Sorajja, D. (2018). Systematic review for the 2017 aha/acc/hrs guideline for management of patients with ventricular arrhythmias and the prevention of sudden cardiac death: a report of the american college of cardiology/american heart association task force on clinical practice guidelines and the heart rhythm society. *Circulation*, 138(13):e392–e414.

Kwon, J.-m., Cho, Y., Jeon, K.-H., Cho, S., Kim, K.-H., Baek, S. D., Jeung, S., Park, J., and Oh, B.-H. (2020). A deep learning algorithm to detect anaemia with ECGs: a retrospective, multicentre study. *The Lancet Digital Health*, 2(7):e358–e367.

Kwon, J.-m., Jung, M.-S., Kim, K.-H., Jo, Y.-Y., Shin, J.-H., Cho, Y.-H., Lee, Y.-J., Ban, J.-H., Jeon, K.-H., Lee, S. Y., et al. (2021). Artificial intelligence for detecting electrolyte imbalance using electrocardiography. *Annals of Noninvasive Electrocardiology*, 26(3):e12839.

Laptev, N., Amizadeh, S., and Flint, I. (2015). Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1939–1947.

Larson, S. C. (1931). The shrinkage of the coefficient of multiple correlation. *Journal of Educational Psychology*, 22(1):45.

Lawrence, W. B., Ellis, W. D., Hekking, F. J., Lagache, M. P., Maddagiri, A. M., and Madsen, A. C. (1977). Steam plant surface condenser leakage study, technical report epri imp-481. Technical report, Electric Power Research Institute.

Lee, S.-H., Lim, J. S., Kim, J.-K., Yang, J., and Lee, Y. (2014). Classification of normal and epileptic seizure eeg signals using wavelet transform, phase-space reconstruction, and euclidean distance. *Computer Methods and Programs in Biomedicine*, 116(1):10–25.

Leevy, J. L., Khoshgoftaar, T. M., Bauder, R. A., and Seliya, N. (2018). A survey on addressing high-class imbalance in big data. *Journal of Big Data*, 5(1):1–30.

Lewis, J. (2019). Private communication as part of his role of former Lead Consultant at DAS Ltd.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2017). Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*.

Li, Q., Li, Z., and Zemkoho, A. (2021). Bilevel hyperparameter optimization for support vector classification: theoretical analysis and a solution method. *arXiv preprint arXiv:2110.01697*.

Lih, O. S., Jahmunah, V., San, T. R., Ciaccio, E. J., Yamakawa, T., Tanabe, M., Kobayashi, M., Faust, O., and Acharya, U. R. (2020). Comprehensive electrocardiographic diagnosis based on deep learning. *Artificial Intelligence in Medicine*, 103:101789.

Liu, W., Zhang, M., Zhang, Y., Liao, Y., Huang, Q., Chang, S., Wang, H., and He, J. (2017). Real-time multilead convolutional neural network for myocardial infarction detection. *IEEE Journal of Biomedical and Health Informatics*, 22(5):1434–1444.

Lohweg, V. (2013). Banknote Authentication. UCI Machine Learning Repository. archive.ics.uci.edu/ml/datasets/banknote+authentication.

Lugovaya, T. S. (2005). Biometric human identification based on electrocardiogram. Master's thesis, Faculty of Computing Technologies and Informatics, Electrotechnical University 'LETI', Saint-Petersburg, Russian Federation.

Luo, M., Wang, K., Cai, Z., Liu, A., Li, Y., and Cheang, C. F. (2019). Using imbalanced triangle synthetic data for machine learning anomaly detection. *Computers, Materials and Continua*, 58(1):15–26.

Madias, J. E. (2005). Qtc interval in patients with changing edematous states: implications on interpreting repeat qtc interval measurements in patients with anasarca of varying etiology and those undergoing hemodialysis. *Pacing and Clinical Electrophysiology*, 28(1):54–61.

Madias, J. E., Bazaz, R., Agarwal, H., Win, M., and Medepalli, L. (2001). Anasarca-mediated attenuation of the amplitude of electrocardiogram complexes: a description of a heretofore unrecognized phenomenon. *Journal of the American College of Cardiology*, 38(3):756–764.

Mahesh, B. (2020). Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9:381–386.

Masters, D. and Luschi, C. (2018). Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*.

Maurya, C. K., Toshniwal, D., and Venkoparao, G. V. (2015). Online anomaly detection via class-imbalance learning. In *2015 Eighth International Conference on Contemporary Computing (IC3)*, pages 30–35. IEEE.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Members, A. F., Priori, S. G., Blomström-Lundqvist, C., Mazzanti, A., Blom, N., Borggrefe, M., Camm, J., Elliott, P. M., Fitzsimons, D., Hatala, R., et al. (2015). 2015 esc guidelines for the management of patients with ventricular arrhythmias and the prevention of sudden cardiac death: The task force for the management of patients with ventricular arrhythmias and the prevention of sudden cardiac death of the european society of cardiology (esc) endorsed by: Association for european paediatric and congenital cardiology (aepc). *Ep Europace*, 17(11):1601–1687.

Miao, F., Wen, B., Hu, Z., Fortino, G., Wang, X.-P., Liu, Z.-D., Tang, M., and Li, Y. (2020). Continuous blood pressure measurement from one-channel electrocardiogram signal using deep-learning techniques. *Artificial Intelligence in Medicine*, 108:101919.

Moniz, N., Branco, P., and Torgo, L. (2016). Resampling strategies for imbalanced time series. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 282–291. IEEE.

Moore, G. M., Bergeron, C., and Bennett, K. P. (2009). Nonsmooth bilevel programming for hyperparameter selection. In *2009 IEEE International Conference on Data Mining Workshops*, pages 374–381. IEEE.

Müller-Steinhagen, H. (1999). Cooling-water fouling in heat exchangers. In *Advances in Heat Transfer*, volume 33, pages 415–496. Elsevier.

Myles, A. J., Feudale, R. N., Liu, Y., Woody, N. A., and Brown, S. D. (2004). An introduction to decision tree modeling. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 18(6):275–285.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Icml*.

Nanda, M. A., Seminar, K. B., Nandika, D., and Maddu, A. (2018). A comparison study of kernel functions in the support vector machine and its application for termite detection. *Information*, 9(1):5.

Nguyen, C. T., Alcantara, J. H., Okuno, T., Takeda, A., and Chen, J.-S. (2021). Unified smoothing approach for best hyperparameter selection problem using a bilevel optimization strategy. *arXiv preprint arXiv:2110.12630*.

Nishimori, M., Kiuchi, K., Nishimura, K., Kusano, K., Yoshida, A., Adachi, K., Hirayama, Y., Miyazaki, Y., Fujiwara, R., Sommer, P., et al. (2021). Accessory pathway analysis using a multimodal deep learning model. *Scientific Reports*, 11(1):1–8.

Nocedal, J. and Wright, S. J. (1999). *Numerical optimization*. Springer.

Okuno, T., Takeda, A., Kawana, A., and Watanabe, M. (2021). On lp-hyperparameter learning via bilevel nonsmooth optimization. *Journal of Machine Learning Research*, 22:245–1.

Omar, S., Ngadi, A., and Jebur, H. H. (2013). Machine learning techniques for anomaly detection: an overview. *International Journal of Computer Applications*, 79(2).

Park, S. H., Goo, J. M., and Jo, C.-H. (2004). Receiver operating characteristic (roc) curve: practical review for radiologists. *Korean Journal of Radiology*, 5(1):11–18.

Patro, G. K. and Sahu, K. K. (2015). Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *Ussr Computational Mathematics and Mathematical Physics*, 4(5):1–17.

Pourbabaee, B., Roshtkhari, M. J., and Khorasani, K. (2018). Deep convolutional neural networks and learning ECG features for screening paroxysmal atrial fibrillation patients. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(12):2095–2104.

Prajapati, G. L. and Patle, A. (2010). On performing classification using svm with radial basis and polynomial kernel functions. In *2010 3rd International Conference on Emerging Trends in Engineering and Technology*, pages 512–515. IEEE.

Prechelt, L. (1998). Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, Berlin/Heidelberg, Germany.

Priori, S. G., Blomström-Lundqvist, C., Mazzanti, A., Blom, N., Borggrefe, M., Camm, J., Elliott, P. M., Fitzsimons, D., Hatala, R., Hindricks, G., et al. (2016). 2015 esc guidelines for the management of patients with ventricular arrhythmias and the prevention of sudden cardiac death. the task force for the management of patients with ventricular arrhythmias and the prevention of sudden cardiac death of the european society of cardiology. *Giornale Italiano di Cardiologia (2006)*, 17(2):108–170.

Probst, P., Boulesteix, A.-L., and Bischl, B. (2019). Tunability: Importance of hyperparameters of machine learning algorithms. *The Journal of Machine Learning Research*, 20(1):1934–1965.

Reed, R. and MarksII, R. J. (1999). *Neural smithing: supervised learning in feedforward artificial neural networks*. MIT Press.

Refaeilzadeh, P., Tang, L., and Liu, H. (2009). Cross-validation. *Encyclopedia of Database Systems*, 5:532–538.

Roberts, F. M., Povinelli, R. J., and Ropella, K. M. (2001). Identification of ECG arrhythmias using phase space reconstruction. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 411–423. Springer.

Rocha, T., Paredes, S., De Carvalho, P., Henriques, J., and Antunes, M. (2008). Phase space reconstruction approach for ventricular arrhythmias characterization. In *2008*

30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pages 5470–5473. IEEE.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Sáez, J. A., Krawczyk, B., and Woźniak, M. (2016). Analyzing the oversampling of different classes and types of examples in multi-class imbalanced datasets. *Pattern Recognition*, 57:164–178.

Sangaiah, A. K., Arumugam, M., and Bian, G.-B. (2020). An intelligent learning approach for improving ECG signal classification and arrhythmia analysis. *Artificial Intelligence in Medicine*, 103:101788.

Scholtes, S. (2001). Convergence properties of a regularization scheme for mathematical programs with complementarity constraints. *SIAM Journal on Optimization*, 11(4):918–936.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.

Shi, D., Zurada, J., Karwowski, W., Guan, J., and Çakıt, E. (2019). Batch and data streaming classification models for detecting adverse events and understanding the influencing factors. *Engineering Applications of Artificial Intelligence*, 85:72–84.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48.

Singh, A. and Purohit, A. (2015). A survey on methods for solving data imbalance problem for classification. *International Journal of Computer Applications*, 127(15):37–41.

Siontis, K. C., Noseworthy, P. A., Attia, Z. I., and Friedman, P. A. (2021). Artificial intelligence-enhanced electrocardiography in cardiovascular disease management. *Nature Reviews Cardiology*, 18(7):465–478.

Smith, J. W., Everhart, J. E., Dickson, W., Knowler, W. C., and Johannes, R. S. (1988). Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, page 261. American Medical Informatics Association.

Sokolova, M., Japkowicz, N., and Szpakowicz, S. (2006). Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. In *Australasian Joint Conference on Artificial Intelligence*, pages 1015–1021. Springer.

Srikanth, T., Branch, P., Jin, J., and Jugdutt, S. (2020). A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, 7(1).

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147. PMLR.

Tbarki, K., Said, S. B., Ksantini, R., and Lachiri, Z. (2016). Rbf kernel based svm classification for landmine detection and discrimination. In *2016 International Image Processing, Applications and Systems (IPAS)*, pages 1–6. IEEE.

van Rees, J. B., Borleffs, C. J. W., de Bie, M. K., Stijnen, T., van Erven, L., Bax, J. J., and Schalij, M. J. (2011). Inappropriate implantable cardioverter-defibrillator shocks: incidence, predictors, and impact on mortality. *Journal of the American College of Cardiology*, 57(5):556–562.

Vanwinckelen, G. and Blockeel, H. (2012). On estimating model accuracy with repeated cross-validation. In *BeneLearn 2012: Proceedings of the 21st Belgian-Dutch Conference on Machine Learning*, pages 39–44.

Vapnik, V. (2000). *The nature of statistical learning theory*. Springer.

Vemishetty, N., Acharyya, A., Das, S., Ayyagari, S., Jana, S., Maharatna, K., and Puddu, P. E. (2016). Classification methodology of cvd with localized feature analysis using phase space reconstruction targeting personalized remote health monitoring. In *2016 Computing in Cardiology Conference (CinC)*, pages 437–440. IEEE.

Vemishetty, N., Gunukula, R. L., Acharyya, A., Puddu, P. E., Das, S., and Maharatna, K. (2019). Phase space reconstruction based cvd classifier using localized features. *Scientific Reports*, 9(1):1–18.

Walker, M. E., Safari, I., Theregowda, R. B., Hsieh, M.-K., Abbasian, J., Arastoopour, H., Dzombak, D. A., and Miller, D. C. (2012). Economic impact of condenser fouling in existing thermoelectric power plants. *Energy*, 44(1):429–437.

Watson, N., Hendricks, S., Stewart, T., and Durbach, I. (2020). Integrating machine learning and decision support in tactical decision-making in rugby union. *Journal of the Operational Research Society*, pages 1–12.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs.

Wu, J. (2017). Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495.

Yang, P., Peng, Y., Tan, H., Liu, H., Wu, D., Wang, X., Li, L., and Peng, X. (2021). Foaming mechanisms and control strategies during the anaerobic digestion of organic waste: A critical review. *Science of The Total Environment*, page 146531.

Ye, J. J., Zhu, D., and Zhu, Q. J. (1997). Exact penalization and necessary optimality conditions for generalized bilevel programming problems. *SIAM Journal on Optimization*, 7(2):481–507.

Yekkehkhany, B., Safari, A., Homoyouni, S., and Hasanlou, M. (2014). A comparison study of different kernel functions for svm-based classification of multi-temporal polarimetry sar data. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(2):281.

Zhang, J., Liu, A., Gao, M., Chen, X., Zhang, X., and Chen, X. (2020). ECG-based multi-class arrhythmia detection using spatio-temporal attention-based convolutional recurrent neural network. *Artificial Intelligence in Medicine*, 106:101856.

Zhu, B., Baesens, B., Backiel, A., and Vanden Broucke, S. K. (2018). Benchmarking sampling techniques for imbalance learning in churn prediction. *Journal of the Operational Research Society*, 69(1):49–65.

Zhu, H., Cheng, C., Yin, H., Li, X., Zuo, P., Ding, J., Lin, F., Wang, J., Zhou, B., Li, Y., et al. (2020). Automatic multilabel electrocardiogram diagnosis of heart rhythm or conduction abnormalities with deep learning: a cohort study. *The Lancet Digital Health*, 2(7):e348–e357.