Detection of Nuisance Call Centres using Improved Hybrid Time Series Classification Algorithms

Matthew Ben Middlehurst

School of Computing Sciences

University of East Anglia

This thesis is submitted for the degree of

Doctor of Philosophy

May 2023

Abstract

Nuisance calling has become a major problem in the last couple of decades with the advent of digital VoIP and increasing automation. Millions of these nuisance calls are made every month, wasting the time of those who receive them or aiming to scam vulnerable consumers. To take action against call centres making these calls, reliably detecting that a Calling Line Identification (CLI) number is carrying nuisance traffic and identifying the source using the CLI is key. This must be done carefully, however. Phone calls and call records contain sensitive information, and consumer privacy must be taken into account and protected.

Time Series Classification (TSC) is a field of research focusing on classifying data in the format of ordered series. We further develop and investigate the use of these algorithms for detecting nuisance call centres. A high accuracy TSC algorithm, HIVE-COTE, is a hybrid heterogenous ensemble of TSC approaches, with each ensemble member extracting a different type of feature from a time series. While HIVE-COTE has shown to generally perform well on the UCR archive of time series datasets, it is slow and many algorithms have caught up to its performance as the field has developed.

In this thesis, we answer two research questions. The first question is whether we can detect nuisance traffic from call centres with minimal information using time series data and TSC algorithms. The second is whether we can improve HIVE-COTE through updating the members of the ensemble which have fallen behind in performance and scalability.

We investigate improvements to HIVE-COTE through introducing more accurate and scalable alternatives to its phase-independent interval and bag-of-words dictionary based constituent classifiers, and altering the way it generates accuracy estimates. This results in the HIVE-COTE 2.0 (HC2) algorithm. We demonstrate the HC2 is much more usable than the original HIVE-COTE for a wide variety of datasets, and once again performs as a state-of-the-art classifier in terms of accuracy on the UCR and UEA time series archives.

With our improved TSC algorithms, we classify time series data from nuisance and legitimate call centres. Given a CLI number, we format the call volume and call duration of calls made from a CLI into a time series. This approach requires no identifying information from any consumer or call centre using the CLI, while providing a profile of how calls are being made by that number. We show that our algorithms can perfectly separate legitimate and nuisance call centres using data provided by British Telecom (BT). An investigation into classifying time series with as few calls as possible shows it is still possible to achieve 100% accuracy with only a portion of the day's traffic, allowing action to be taken in real-time.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Anthony Bagnall, his constant support throughout this PhD project has been irreplaceable, and this thesis would not have been possible without it. I would like to thank both Prof. Gerard Parr for his supervision efforts during his undoubtably busy tenure as CMP Head of School during a crisis, and Rob Claxton for supporting my nuisance call prevention research and welcoming me to Adastral Park during my placement. Thank you to BT and the EPSRC for funding my PhD, it has been an amazing time.

I would like to thank everyone in the TSC group ran by Tony, especially my fellow PhD students at the time James, George and Michael. Our group conference trips and numerous lunches at UEA with others from CMP were always a great time, it is a shame COVID-19 caused these to stop. I wish the best for all of you. I'd also like to thank all those around UEA who indirectly supported my PhD, including CMP support and the HPC team.

Finally, I would like to thank my family and friends. While I may not be the most social at times, the time we spend together matters to me greatly. Of course, a special thanks to my cats for keeping me company during the hours spent writing, and to my mother for always looking out for me.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Matthew Ben Middlehurst

May 2023

Publications

As First Author

- Middlehurst, M., Vickers, W., & Bagnall, A. (2019). Scalable dictionary classifiers for time series classification. *In International Conference on Intelligent Data Engineering and Automated Learning* (pp. 11-19). Springer, Cham.

- Middlehurst, M., Large, J., Cawley, G., & Bagnall, A. (2020). The Temporal Dictionary Ensemble (TDE) Classifier for Time Series Classification. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases* (pp. 660-676). Springer, Cham.

- Middlehurst, M., Large, J., & Bagnall, A. (2020). The Canonical Interval Forest (CIF) Classifier for Time Series Classification. In *2020 IEEE International Conference on Big Data (Big Data)* (pp. 188-195), IEEE.

- Middlehurst, M., Large, J., Flynn, M., Lines, J., Bostrom, A. & Bagnall, A. (2021). HIVE-COTE 2.0: a new meta ensemble for time series classification. *Machine Learning* 110(3), 3211–3243

- Middlehurst, M., & Bagnall, A. (2022). The FreshPRINCE: A simple transformation based pipeline time series classifier. In *International Conference on Pattern Recognition and Artificial Intelligence* (pp. 150–161). Springer, Cham.

As Co-author

- Bagnall, A., Flynn, M., Large, J., Lines, J., & Middlehurst, M. (2020). On the usage and performance of the Hierarchical Vote Collective of Transformation-based Ensembles version 1.0 (HIVE-COTE V1. 0). In *International Workshop on Advanced Analytics and Learning on Temporal Data* (pp. 3-18). Springer, Cham.

- Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., & Bagnall, A. (2021). The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery* 35(2), 401-449.

Table of contents

List of figures

List of tables

List of algorithms

Chapter 1

Introduction

The threat of nuisance calls harassing consumers has expanded drastically in the last two decades, both in the United Kingdom and internationally, with many governmental bodies and industry leaders taking action and developing solutions for the problem. We define nuisance calls as phone calls received from an organised call centre which fails to follow government regulation in its calling practice or outputs calls with malicious content. The contents of nuisance calls can vary, common types include fully automated marketing calls, abandoned or silent calls, live telesales calls, and callers attempting to scam and defraud consumers. While at a minimum these calls will be an annoyance trying to sell consumers products, at worst they can be malicious actors trying to acquire money, financial details or access to computers and service accounts.

The Information Commissioner's Office (ICO), the United Kingdom national watchdog for information rights, and the communications regulator, the Office of Communications (Ofcom), have made an ongoing joint effort with the prevention of nuisance calls in mind. In their 2018 joint action plan, that estimated that there were 3.9 billion nuisance calls received per year in the UK [130]. This is a decrease of around 20% from the previous estimate of 4.8 billion from 2015 where it hit its peak. While the decrease in calls shows that progress has been made from the

actions taken so far, there are still a large amount of nuisance calls received every year. The ICO and Ofcom can only take actions against nuisance calls made within its jurisdiction, and this initial decrease could be from callers who make little or no effort to conceal themselves. This is reinforced in their 2021 update [131], which attributes more recent drops in nuisance calling to the Covid-19 pandemic, and notes that a rise in nuisance calling is expected as lockdowns end and the organisations running nuisance call centres adapt to current detection techniques and regulation.

In the US, there has been a major increase in complaints to the Federal Trade Commission (FTC) regarding 'robocalls' according to their Biennial report for 2016 and 2017 [132]. The number of complaints has risen from 63,000 per month in 2009 to 375,000 in 2016, with an additional 185,000 complaints to the Federal Communication Commission (FCC). The German Federal Network Agency (BNetzA) reported an increase in complaints in its 2017 annual report [50], rising to 164,351 written complaints and 26,861 telephone complaints for 'number misuse' from 78,209 written and 22,338 telephone complaints the previous year. The number of complaints for nuisance marketing calls also significantly increased, rising to 57,426 from 29,298 from the previous year.

While these reports show an increase of reported cases of nuisance calls for a few specific countries, the general trend continues through many other nations as well. This is shown in a study comparing the problem and policies in multiple nations by StepChange Debt Charity in 2015 [91] and its appendix [90] detailing the situation in a country by country basis. The report shows a trend of increasing action taken against nuisance calls internationally, such as Do Not Call (DNC) registers and/or increased regulation. This increase in nuisance calls globally is attributed to an increase in Voice over Internet Protocol (VoIP) as a method of making calls and international communications. VoIP has become a norm in the telecommunications systems of organizations and is expected to continue growing [22]. The main

attractiveness of VoIP for nuisance calls is that it allows cheaper national and international calls and is much more easily automated, allowing the increases in outgoing nuisance calls found in recent years.

Due to the volume of calls being made on a daily basis, manually combing through data to identify these callers is infeasible if action is to be taken in a reasonable timeframe. As such, an automated solution is required to detect nuisance callers as quickly as possible to prevent more nuisance calls from being made. In this thesis, we collaborate with British Telecom (BT), which provides us with the data to perform our nuisance call detection experiments. BT has systems in place to deal with nuisance calling, such as its Call Protect system, but is always looking to improve its detection systems and better understand the problem of nuisance calling. As the methods of nuisance callers evolve, so too must the systems used to detect the traffic they output. Care must be taken, however, as telecommunications is a sensitive area and the privacy of consumers must be protected.

For any such system developed, there are two performance requirements outlined by BT. The first and main requirement is for the system to misclassify as few legitimate call centres as possible. For a fully automatic blocking system, this would require essentially perfect specificity if a system is to see any real use. The cost of accidentally blocking the calls of a legitimate call centre is severe, and the risk of an automated system doing so is a large concern. Beyond financial and legal repercussions, if calls from a hospital call centre were to be blocked, lives could be put at risk. The second requirement is that any classification of calls as legitimate or nuisance must be processed quick enough to avoid an unreasonable delay in calls. There were no specific performance thresholds provided by BT for this, but the scalability of an algorithm in an important factor in its usability for the nuisance call detection problem.

It is expected that advanced statistical and machine learning techniques could be used to automatically discriminate between nuisance and legitimate call centres.

Much of the data available in-network for the problem consists of time series. An example of this is the call volume for a single Calling Line Identification (CLI) number. If a certain phone number is outputting hundreds of calls, we can identify when each of these calls were made and create a series of call volume over time. Formatting data in this way has the major benefit of being largely anonymous for any machine learning algorithm, only requiring a CLI and the knowledge of when its calls have taken place.

To classify this data as nuisance or legitimate, we use a sub-set of machine learning algorithms specifically designed for classifying time series data. Classification is the task of categorising cases for some problem into one of a set of class labels. Classification algorithms are trained using a representative example of expected data with accompanying class labels, allowing the algorithm to map the patterns seen in the training data to the labels. This mapping is then used to assign a predicted label to new data. Time Series Classification (TSC) follows the same setting as regular classification, but aims to learn patterns from ordered series data specifically.

Of the available TSC algorithms, one of the more accurate is the heterogenous ensemble, the Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) [78]. HIVE-COTE is a meta-ensemble of TSC classifiers built on different data transformations. These transformations extract different types of features from the time series, categorised using a taxonomy presented in Chapter 2. The main feature representations we introduce are use the follow approaches: distance measures; phase-independent shapelets; bag-of-words dictionaries; phase-dependent intervals; convolutions; and hybrid approaches. HIVE-COTE belongs to the final category, and it built using algorithms from the former. With the presence of expert knowledge, individual representations can be selected to best fit the problem. Without this knowledge, hybrid algorithms using multiple feature representations are the best approach. At the time of a 2017 comparison of TSC

algorithms, commonly known as the bake off [8], HIVE-COTE was shown to be significantly more accurate than all other TSC algorithms compared. As the field has progressed, multiple approaches have been produced which match the accuracy of HIVE-COTE [5, 43, 114]. HIVE-COTE also has troubles with scalability, with three of its five ensemble components running slowly compared to more recent algorithms.

Our main hypothesis in this thesis is that TSC algorithms can learn the calling patterns from call centre data formatted as a time series, accurately detecting nuisance call centres and separating them from legitimate call centres and regular consumer traffic. Another objective of this thesis is to improve the HIVE-COTE ensemble, to both increase its accuracy and make it scalable enough to use in our nuisance call detection efforts.

1.1 Thesis Contributions

1. **Improved single representation TSC algorithms.** We introduce improvements to the dictionary, interval and convolution based feature representations of TSC algorithms. Each of our improvements aim to improve a flaw of a component or candidate for inclusion in the HIVE-COTE ensemble. The cBOSS [88] and TDE [86] algorithms improve on the BOSS [105] components. CIF [85] and DrCIF [87] improve on TSF [40]. Arsenal [87] adapts ROCKET [37] for use in HIVE-COTE. DrCIF and TDE in particular represent a new state-of-the-art for their respective feature representations, performing significantly more accurate than alternative algorithms producing similar features.

2. **State-of-the-art TSC with HIVE-COTE 2.0.** In Chapter 6 we describe the newly restructured HIVE-COTE algorithm, and demonstrate that it performs

significantly better than the current state-of-the-art for TSC using a variety of performance metrics on two dataset archives. We exhibit our scalability and usability improvements, which allows HIVE-COTE to run on large datasets it previously would not be able to.

3. **Multivariate TSC benchmarking and improvements.** We extend HIVE-COTE and its components to introduce the capability for classifying multivariate time series, where each problem case consists of multiple time series. In Chapters 4, 5 and 6 we demonstrate the effectiveness of these multivariate extensions, all of which perform as well as or significantly better than current approaches designed for multivariate data. This comparison also serves to benchmark current multivariate techniques on the relatively new UEA multivariate archive [3].

4. **Evaluation of TSC algorithms for nuisance call centre detection.** In Chapter 7 we demonstrate the aptitude of TSC techniques for detecting nuisance call centres using time series data. We show that multiple techniques can perfectly separate the legitimate and nuisance call centres in data provided by BT, and further show that a decision can be made early enough in a day to significantly reduce the amount of nuisance calls a CLI is outputting. Chapter 8 extends this work, looking at the impact of spoofing to split call output between multiple CLIs, and shows that TSC techniques are still viable for detecting this traffic.

1.2 Thesis Outline

This thesis is organised into nine chapters. In the following, we outline the contents of the remaining chapters.

Chapter 2 provides a detailed background for nuisance calling and time series classification. We describe the types of nuisance calls, their effects on consumers, current actions taken to prevent these calls from reaching consumers, and the limitations of nuisance call prevention. We review the literature for univariate and multivariate TSC, discussing the types of approaches and how they can be grouped into a taxonomy used to produce the HIVE-COTE algorithm.

Chapter 3 describes the experimental methodology used throughout the thesis, including the statistics and methods of comparison used. We detail the dataset archives used for comparing our algorithms and case study datasets. We outline the open source software packages used and contributed to in an effort to make our research and results reproducible.

Chapters 4 through 6 presents our advances in the field of TSC. We present improvements to the dictionary based representation of classifiers in Chapter 4 with the cBOSS and TDE classifiers, and to the interval based representation in 5 with the CIF and DrCIF classifiers. These improvements to individual representation classifiers culminate in Chapter 6, where we introduce the state-of-the-art HIVE-COTE 2.0 classifier, a hybrid ensemble encompassing multiple TSC representations.

Chapters 7 and 8 focus on our research into nuisance call detection. Chapter 8 explores the classification of single high volume CLIs as either nuisance or legitimate, and evaluates the effectiveness of early classification to provide real time classifications. Chapter 8 explores the issue of CLI rotation, where nuisance classers split their traffic over multiple CLIs to avoid current detection techniques.

Chapter 9 provides a conclusion to this thesis, summarises the contributions made, and discusses future work for TSC and nuisance call detection.

Chapter 2

Background and Related Work

This chapter introduces relevant background material for this thesis. We further motivate the problem of detecting nuisance calling, and investigate background literature and related works on the detection of nuisance calls using machine learning. We present current techniques for detecting nuisance callers in both academia and the telecommunications industry. Following this, we describe the field of Time Series Classification (TSC), introducing relevant classifiers using a taxonomy based on the time series features extracted by an algorithm.

2.1 Nuisance Calls

We first expand on the problem of nuisance calling we introduced in Chapter 1. We discuss the types of nuisance call and the impact these nuisance calls have on consumers to further motivate the problem. We then review actions taken against nuisance callers and efforts to detect nuisance calls from a regulatory, industry and academic standpoint.

2.1.1 Types of Nuisance Call

To detect nuisance calling, it is important to understand the characteristics of a nuisance call and how it may differ from regular calling. Unfortunately, there is no standard definition for what a nuisance call is. The contents of nuisance calls can wildly differ, and different nations regulatory bodies will categorise them differently. Defining a nuisance call as 'a call the recipient does not want to receive' is too broad, and could encompass anything from personal calls to large scale fraud attempts. We categorise types of nuisance calling using the definitions provided by the United Kingdom Information Commissioner's Office (ICO) and Office of Communications (Ofcom).

Abandoned calls and silent calls[1] are both similar in nature. With silent calls, there is no talking or messages played throughout the call. An abandoned call will play an information message about the organisation trying to contact the receiver before said silence. In both cases, no human is involved with the call from the side of the sender. Ofcom identifies the cause of a majority of these calls to be the result of automatic diallers, primarily used in call centres to increase the amount of outgoing calls.

Automated marketing calls[2] involves the caller's audio being from a robotic source, whether that be a recorded message or something more sophisticated. Unlike abandoned calls which will just relay organisation details, automated marketing calls will try to market a product or service to consumers. Common calls of this type involve relaying to the called party that they are due compensation from a car accident or other incident, whether this is true or not, with the aim of finding potential customers who have been involved with those situations. Other common

[1] https://www.ofcom.org.uk/phones-telecoms-and-internet/advice-for-consumers/problems/tackling-nuisance-calls-and-messages/abandoned-and-silent-calls

[2] https://www.ofcom.org.uk/phones-telecoms-and-internet/advice-for-consumers/problems/tackling-nuisance-calls-and-messages

areas for automatic calls are mis-selling of payment protection insurance and debt management services.

Live telesales calls[3] are calls involving humans on both sides of the call, where the caller is attempting to sell the called party a product or service. These may be cold calls or calls to a previous client of the business. Examples of a live telesales call would be a call from an unknown number trying to sell window double glazing, or a mobile phone provider informing customers about service deal upgrades. Live telesales calls are not necessarily nuisance calls, however. While advertising through telecommunications is regulated, if these regulations are kept to, there is no reason to block these calls.

Some types of calls may not fall under any of these categories. For example, scams and frauds may require their own category, despite also being automated or live calls. An example of this are 'missed call' scams which are recognized by Ofcom[4] and the German BNetzA has received numerous complaints about recently [50]. The scam involves calling then instantly terminating the call to appear as a missed call in the receiver's call history, with the goal of having them call back and be charged for a premium rate number. This type of call would differ quite a bit from the other categories, as it does not involve getting a message across to the person being called or even connecting in the first place.

While we have provided examples of the possible contents of nuisance calls, from the network side of detection, none of this information is available. In our effort to detect nuisance callers, we want to avoid any identifying information such as location, age or gender. It goes without saying that using the contents of consumer phone calls is out of the question, even if such data were to be available. What we can derive from this information is the method nuisance callers use to

[3]https://www.ofcom.org.uk/phones-telecoms-and-internet/advice-for-consumers/problems/tackling-nuisance-calls-and-messages/live-marketing-calls

[4]https://www.ofcom.org.uk/about-ofcom/latest/features-and-news/missed-call-scams

output their calls. Automated calls and most live callers skirting regulation will use fully automated methods or aggressively configured auto diallers for their callers. It is the difference between these methods and regulation abiding methods of calling by call centres we are hoping to capture with our time series data.

2.1.2 Impact of Nuisance Calling

The effect a nuisance call has on a member of the public who is called can vary based on the contents of the call and who the call recipient is. It can range from a minor annoyance, to more severe cases where individuals are frightened, stressed or suffer a monetary loss to scams or fraud. Since 2013, GfK UK has produced yearly survey data for Ofcom regarding the pervasiveness and impact of nuisance calls done over a four-week period using a nationally representative sample. This study covers areas such as the amount of nuisance calls received, who is receiving them and how the receiver feels about receiving these calls. While there has been a drop from previous years, in 2019 79% of survey participants received at least one nuisance call during the four-week period, those who did receiving 7.4 nuisance calls on average during the month [129].

When it comes to the attitudes of people who receive nuisance calls, 83% of participants found nuisance calls to be just annoying. More seriously, a smaller percentage of 11% of participants found the calls to be worrying or distressing [129]. Silent calls were found to be the most annoying type of nuisance call, while abandoned calls were the most distressing. Mirroring this unpopular sentiment regarding nuisance calling, a campaign by Which calling the government, regulators and businesses to take action against nuisance calls has received almost 800,000 signatures at the time of writing[5].

[5]https://campaigns.which.co.uk/nuisance-calls/#?intcmp=GNH.Campaigns.Stop-Nuisance-Calls-And-Texts

While all groups on average received over five nuisance calls over the survey period, those over the age of 55 or unemployed received significantly more nuisance calls. While there is no evidence of specific targeting for these groups, the fact that consumers which may be viewed as more vulnerable are receiving more nuisance calls is cause for concern. According to StepChange Debt Charity, 1.2 million adults in the UK were tempted to take out high interest loans such as payday loans as a result of nuisance marketing calls or texts [25]. In addition to the annoyance and stress from nuisance calls, some are much more malicious in their intentions, attempting to scam or defraud the person they are calling. There have been many different scam and fraud methods used to cause monetary loss and stress to consumers in both the UK[6] and the US[7] using telecommunications as a medium. Newer ones include the previously mentioned 'missing call' scam and scams relating to the Covid-19 pandemic.

It is not difficult to conclude that nuisance calling has a negative impact on society, and while we should be careful as to what we consider nuisance calling, reducing the amount of calls avoiding telecommunications regulations would have a positive impact.

2.1.3 Regulatory Actions Taken Against Nuisance Calls

Most of the actions taken to date have been regulatory, such as threatening action against organisations that break calling regulations or telecommunications companies that fail to put in place proper preventative measures. A popular method of prevention internationally are Do Not Call (DNC) registries such as the TPS[8] in the UK, BLOCTEL in France and NDNC in India. These registries in most cases legally prevent contact without consent from people on the registry. Though

[6]https://www.bbc.co.uk/news/business-45265609

[7]https://www.marketwatch.com/story/heres-how-much-phone-scams-cost-americans-last-year-2017-04-19

[8]https://www.tpsonline.org.uk//tps/index.html

DNC registers have continued to increase the number of people signed up to them each year, the amount of reported nuisance calls does not drop as a result, with the amount either growing or remaining stagnant [91]. While call centres with legitimate activities or marketing agencies may adhere to these lists, it is unlikely that nuisance callers who are intentionally breaking regulations anyway will take any notice.

In the UK, the ICO and Ofcom have taken numerous enforcement and regulatory actions to prevent nuisance calls. In 2017 the ICO issued 29 fines totalling to £2,833,500, with the largest penalty being £400,000 to an organisation for making over 99 million automated marketing calls [130]. Ofcom has recently focused on making sure communications providers are protecting their customers, primarily through ensuring that calls with an invalid CLI are blocked when feasible. Efforts have also been made to increase communications between ICO, Ofcom and industry regarding nuisance calls. This has been done through the establishment of a Strategic Working Group (SWG) consisting of communications providers and communications with the Internet Engineering Task Force (IETF) regarding the reliability of CLI. The UK government Department for Digital, Culture, Media & Sport (DCMS) is also keen on taking action to prevent nuisance calls, in their 2014 action plan [128] they detailed increased enforcement powers for the ICO and Ofcom and information sharing abilities between the regulators. More recently, in a 2018 press release[9] it was revealed that only half of the money from fines since 2010 has been collected, as companies go into liquidation to avoid penalties. To combat this, a new proposal to provide the ICO the authority to hold company directors liable for these fines has been made.

The German BNetzA opened 4,481 administrative proceedings in 2016 related to number misuse, with resulting disconnection of phone numbers and billing

[9]https://www.gov.uk/government/news/new-measures-to-beat-plague-of-nuisance-calls

ban. Additionally, services to provide free information on charges for calling specific country codes to help prevent 'missed call' scams and the provision of early warnings to companies receiving complaints about the use of predictive auto diallers were introduced [50]. In late 2018, the chairman of the US FCC threatened to take action against companies that did not put call authentication systems in place by the end of 2019[10]. A collection of other nuisance call prevention policies implemented by different nations is complied in the StepChange Debt Charity nuisance call study appendix [90].

While legal action is an effective measure and has shown success in the prevention of nuisance calls locally, the problem is still significant and nuisance callers are expected to evolve their strategt to circumvent these actions. Two major concerns for nuisance call enforcement are international calls and spoofing (falsifying CLI to appear as a different number). It is much more difficult, if not impossible in some cases, to take effective regulatory action against callers operating internationally. Unexpected calls coming from an international number may appear untrustworthy to a person receiving them. However, spoofing allows the callers to change their CLI. If the operation is sophisticated enough, the CLI can even appear as a local number to the recipient.

With the difficulties of taking legal action against many of the perpetrators of nuisance calling, there is little incentive for them to stop while it remains profitable. While it is not possible to take regulatory action against some of the organisations outputting nuisance calls, it is possible to prevent the calls from taking place if they can be detected. By blocking CLIs used for nuisance calling on a short term basis, or taking action against the networks which carry considerable nuisance traffic, the number of nuisance calls reaching consumers can be reduced.

[10]www.reuters.com/article/us-usa-wireless-fcc/u-s-regulator-demands-companies-take-action-to-halt-robocalls-idUSKCN1NA2KH?il=0

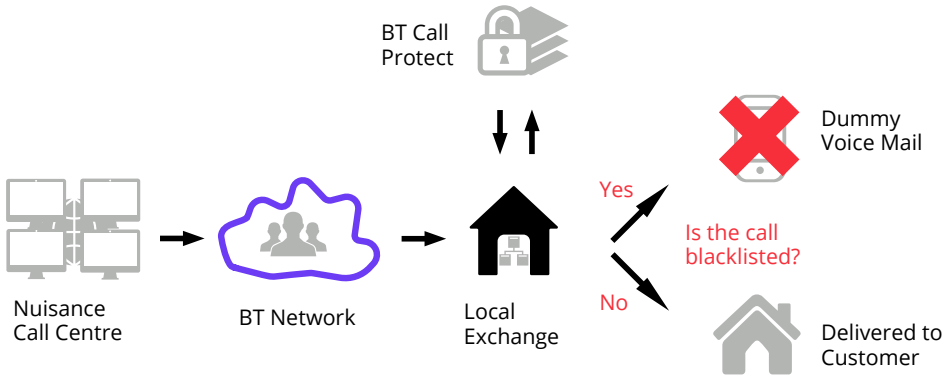


Fig. 2.1 An overview of the BT Call Protect system. CLIs are checked against a database of blacklisted numbers before being delivered to consumers. CLIs with suspicious characteristics are added to a queue to be evaluated for nuisance traffic.

2.1.4 Industry Nuisance Call Prevention Systems

As the regulations surrounding nuisance calling increase and customers face contiunual harassment by malicious users, the telecommunications industry has also looked for solutions to the problem of nuisance calls. The primary system for detecting nuisance calls employed by BT is Call Protect[11]. The system is free for BT customers and is made up of two main parts: a personal blacklist and a network blacklist, both of which send calls from listed phone numbers to the receiver's junk voicemail. The personal blacklist is controlled by the user, and allows for individual phone numbers or calls from specific categories to be blocked at their discretion. The network blacklist on the other hand is maintained by BT, blocking calls to all users from phone numbers on it. An overview of the Call Protect system is shown in Figure 2.1.

A CLI can be flagged either automatically through suspicious characteristics such as a high volume of calls with low call duration, repeated complaints about the CLI or its addition to many personal blacklists. These suspicious phone numbers are added to a list to be manually reviewed to prevent false positives, and added to the network blacklist if found to be carrying significant nuisance traffic. Phone

[11]https://www.bt.com/help/security/bt-call-protect----how-to-----guide

numbers are flagged and added to the network blacklist daily, with the blacklist being periodically cleansed. The requirement to manually process suspicious CLIs adds a layer of safety, but also a significant overhead in time required to add a CLI to the blacklist. Spoofing can also be used to circumvent the blacklist, meaning in a lot of cases by the time a nuisance CLI has been blacklisted the callers have already moved onto another CLI.

Other solutions provide call receivers with the tools to block unwanted calls. Systems such as the BT8600 by BT or CallSafe by TalkTalk use both a whitelist and a blacklist. Contacts on the whitelist can immediately call through to the user. Calls not on the whitelist will be screened, with the blocker asking for the callers name. From there, the user can choose to accept or decline the call. Over time the blocker learns from this to make a decision on certain callers. The extra steps required for accepting calls and setting up the two lists may make people less likely to use systems such as these, however. Additionally, while not affecting the user directly, the screening processes could cause annoyance to legitimate callers.

2.1.5 Previous Solutions for Detection of Nuisance Calls

Machine learning as a solution to spam is not a new concept, the detection of text based email spam is a popular research field [127, 56, 66]. Compared to its email counterpart, spam in the telephony domain is a relatively unresearched area, with the majority of studies relating to detection of spam in VoIP systems, also known as Spam over Internet Telephony (SPIT).

One popular method for SPIT detection are trust based systems that make use of graphs connecting users that contact one another. These approaches make use of the assumption that if User A trusts User B, and User B trusts User C, then User A should somewhat trust User C. This also applies for untrustworthiness. Using this trust system, the reputation of a user can quickly spread around the graph

network as untrustworthy, resulting in them getting effectively blacklisted. A graph based trust system is developed in [23], using the duration of calls between users and blacklists to determine trust. Assuming that a short call duration indicates an undesirable call, having multiple low duration calls will decrease trust. Getting blacklisted by another user will also decrease the user's trust.The opposite of this is true as well, with multiple long duration calls to the same user resulting in a gain of trust. When calling a user, callers with a high trust value will be accepted and those below a set SPIT threshold will be rejected. Another trust base system makes use of Bayesian learning to determine the probability of a call being spam using its trustworthiness and past activities as features is shown in [35].

While both these trust based methods were shown to perform well, there are some issues with applying a similar system to networks such as the one operated by BT. In [23] a buddy list is used to determine trust, which is not feasible on a landline. The amount of calls and size of the network will be substantially larger than those tested, and processing must be done for every call made. Whether the method is feasible under these conditions in undetermined. There is also an open path to exploit by nuisance callers in the form of a Sybil attack, where the scammers call each other to intentionally increase their trust score. This last problem is described in [23] as not cost-effective due to the need for long call durations, but as with the rise in the number of nuisance calls the cost of making calls has become cheaper and this may not be the case any more.

Works applying machine learning for spam call detection has also been produced. In [136] the effectiveness of a semi-supervised clustering method of separating SPIT and non-SPIT calls is studied. This is done using their own VoIP feature specific version of the MPCK-Means [16] algorithm, an extended version of the unsupervised K-Means algorithm. This algorithm makes use of features derived from the Session Initiation Protocol (SIP) used by VoIP to separate the data into discriminative clusters. A study into applying machine learning to the nuisance

call problem in China is performed in [73]. A dataset of 9 billion anonymised call records from 500 million distinct phone number generated from the mobile app TouchPal is used. Features present in the dataset include simple call data such as whether the call is in-going or out-going, the other device's number, the call date and the call duration. From these call characteristics, a number of additional features were derived. The two most effective features being the number of calls made between the two phone numbers and the number of calls made by the caller. Six different classifiers were tested, with the two best performers being the Random Forest algorithm and a neural network. The best model shows a 90% detection rate of nuisance calls while retaining a 99.9% true negative rate.

A study by Microsoft on detecting fraudulent users using Skype using machine learning is detailed in [72]. While in this project we seek to detect nuisance calls as soon as possible, the study focuses on fraudulent users who have been active for a length of time and gotten past the initial Skype spam detection. It makes use of static profile features, activity time series data and a contacts graph similar to the trust based systems. From time series data, the likelihood of the user being fraudulent and non-fraudulent is generated using a Hidden Markov Model (HMM) and used as features. The contacts graph is transformed into usable features using the PageRank score commonly used in information retrieval and the local clustering coefficient of the node being classified. These features are then concatenated into a single feature vector and built on a random forest classifier, which achieved a true positive rate of 68% and a false positive rate of 5%. While less than other methods, the results are deemed significant due to the fact that these users were able to get past initial prevention methods.

Some features of this research makes its application to our BT context infeasible. A lot of data used is either fully simulated based on perceived differences between nuisance calling and legitimate [23] or labelled through user interaction with an app deciding whether a phone number is nuisance or not [73]. None of the approaches

are focusing on call centres specifically, and no mention of legitimate call centres and their similarities to nuisance ones are made. The large amounts of regular low volume traffic may mean some of the statistics shown are misleading.While both legitimate call centres (assuming they are included at all) and regular consumer traffic would be labelled as legitimate, it is much more difficult to separate legitimate call centres from nuisance call centre traffic than it is to separate low volume consumer traffic. If there is one legitimate call centre number for every 10,000 numbers used by a regular consumer, a simple call volume threshold would give you a 99.99% accuracy in standard cases. This is demonstrated in [73], where the number of calls made by the caller is selected as a top feature. Both nuisance and legitimate call centres would have this in common. Due to spoofing, it is not even necessarily true for nuisance callers, as they may swap their CLI after a certain number of calls are made.

As we only use time series data extracted from the usage of a CLI, very little of the techniques discussed can be applied to our work in this thesis. The only piece of literature to make use of time series data is [72], and this is part of a larger system extracting only a few features from the series. While HMM approaches have been used for TSC, it is considered niche and uncompetitive compared to recent advances [8].

2.2 Time Series Classification

Time Series Classification (TSC) is the problem of predicting a discrete target variable from a (possibly multivariate) time series. TSC problems are seen in many areas of machine learning applications, including medical use in seizure detection [24], earthquake and wave monitoring [2] and insect classification [97]. An example of a time series used in this thesis is the amount of outgoing calls made for a CLI over the period of a day, captured every minute.

We define a TSC problem $\boldsymbol{T}$ as a list of n cases $\boldsymbol{X}$, each with an associated class label $\boldsymbol{y}$

$$\boldsymbol{T} = \{(\boldsymbol{X}_1, \boldsymbol{y}_1), (\boldsymbol{X}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{X}_n, \boldsymbol{y}_n)\} \tag{2.1}$$

where each case is made up of d time series

$$\boldsymbol{X}_i = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_d}\} \tag{2.2}$$

and each series consists of m real-valued and ordered observations.

$$\boldsymbol{x}_j = \{t_1, t_2, \ldots, t_m\} \tag{2.3}$$

A univariate problem will only have one time series per case, while a multivariate time series problem will have multiple. Our nuisance calling dataset would have n time series, each from a unique CLI, and 1440 observations for each minute in a day. Some machine learning tasks will not require labels, or only require some cases with labels. For classification, we require all cases have a corresponding class label. In this thesis, we look at problems where the time series are of equal length and have no missing values. It is worth noting that unequal length time series is a common trait in real world data, however.

In a large survey of TSC algorithms [8], commonly referred to as the time series bake off, a taxonomy is defined based on the type of discriminatory features the algorithm is trying to find. As the field has progress, this has expanded from the originally defined representations. This taxonomy underpins the TSC improvements we make in this thesis. In this section, we group TSC algorithms into categories similarly based on the features they extract.

2.2.1 Exploratory Transformations

A popular theme in recent time series research is proposing tools that summarise series in an unsupervised manner and can be used in a range of machine learning tasks, including classification. There are several popular toolkits for forming summary features. The **Highly Comparative Time Series Analysis (*hctsa*) [47]** toolbox can create over 7700 features for exploratory time series analysis.

The Canonical Time Series Characteristics (Catch22) [80] are 22 features determined to be the most discriminatory of the full *hctsa* [47] set through an evaluation over the UCR [36] datasets. The *hctsa* [47] features were initially pruned, removing those which are sensitive to mean and variance and any which could not be calculated on over 80% of the UCR datasets. A feature evaluation was then performed based on predictive performance. Any features which performed below a threshold were removed. For the remaining features, a hierarchical clustering was performed on the correlation matrix to remove redundancy. From each of the 22 clusters formed, a single feature was selected, taking into account balanced accuracy, computational efficiency and interpretability. The Catch22 features cover a wide range of concepts such as basic statistics of time series values, linear correlations, and entropy. The reported results for Catch22 builds a decision tree classifier after applying the transform to each time series [80].

Time Series Feature Extraction based on Scalable Hypothesis Tests (tsfresh) [29] is a collection of just under 800 features extracted from time series. While the features can be used on their own, a feature selection method called FRESH is provided to remove irrelevant features for a problem. FRESH considers each feature using multiple hypotheses tests, including Fisher's exact test, the Kolmogorov-Smirnov test and the Kendal rank test. The Benjamini-Yekutieli procedure is then used to control the false discovery rate caused by comparing multiple hypotheses and features simultaneously. Results for the base features and after using the FRESH algorithm are reported using both a random forest [18] and AdaBoost [46] classifier.

Generalised Signatures [93] are a set of feature extraction techniques primarily for multivariate time series based on rough path theory. We specifically look at the generalised signature method [93] and the accompanying canonical signature pipeline. Signatures are collections of ordered cross-moments. The pipeline begins by applying two augmentations by default. The basepoint augmentation simply adds a zero at the beginning of the time series, making the signature sensitive to translations of the time series. The time augmentation adds the series timestamps as an extra coordinate to guarantee each signature is unique and obtain information about the parameterisation of the time series. A hierarchical dyadic window is run over the series, with the signature transform being applied to each window. The output for each window is then concatenated into a feature vector. The signature transform and windowing both require a depth parameter, which is optimised using cross-validation. The reported results for the canonical signature pipeline make use of a Random Forest [18] classifier. Out-of-bag error is used to select the best max depth and number of trees for the forest, with 20 parameter sets being randomly selected from a grid of parameter options.

A comparison of unsupervised transforms for classification was presented in [84]. The evaluation showed that simple pipelines of transformation and standard

classifier could achieve reasonable accuracy on the benchmark UCR data (described in detail in Chapter 3), although the best approach evaluated, the **Fresh Pipeline with Rotation Forest Classifier (FreshPRINCE)**, is not competitive with current state-of-the-art, it is significantly more accurate than many of the algorithms presented in the bake off. The FreshPRINCE uses the full set of ts-fresh features with a Rotation Forest classifier [103].

2.2.2 Distance Based

Distance based classifiers includes approaches which use one or more distance function to measure the (dis)similarity of two time series with a classification algorithm that uses the distance function(s). Standard distance functions such as Euclidean distance cannot usually make the best use of the ordered nature of time series data, however. Elastic distance measures take account for misalignment (distortions), such as skews or translations, between two time series while computing their similarity. Figure 2.2 provides an overview of the distance based space and the relation of classifiers we introduce.

The most widely researched and used elastic distance measure is **Dynamic Time Warping (DTW) [15]**. DTW applies an elastic transformation to the ordering axis to account for distortions by realigning (warping) two time series to best match each other. Figure 2.3 illustrates this temporal alignment for two time series with a phase shift and different amplitudes. The goal of DTW is to find the optimal alignment of the time axis such that its total distance is minimised. DTW with a 1-nearest neighbour classifier (1-NN DTW) was one of the oldest time series classification techniques and was said to be exceptionally hard to beat in terms of accuracy [102, 137], though more recent advances have surpassed it [8].

Many other elastic measures exist, many of which are variations on DTW [53, 60, 62, 59]. One of the DTW extensions which differs from the standard usage of

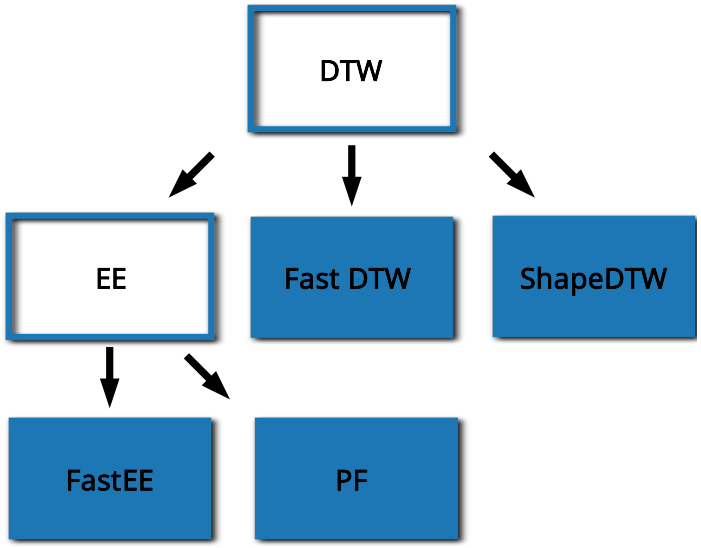


Fig. 2.2 An overview of distance based approaches and the relationship between them. Filled classifiers were released post-bake off [8]

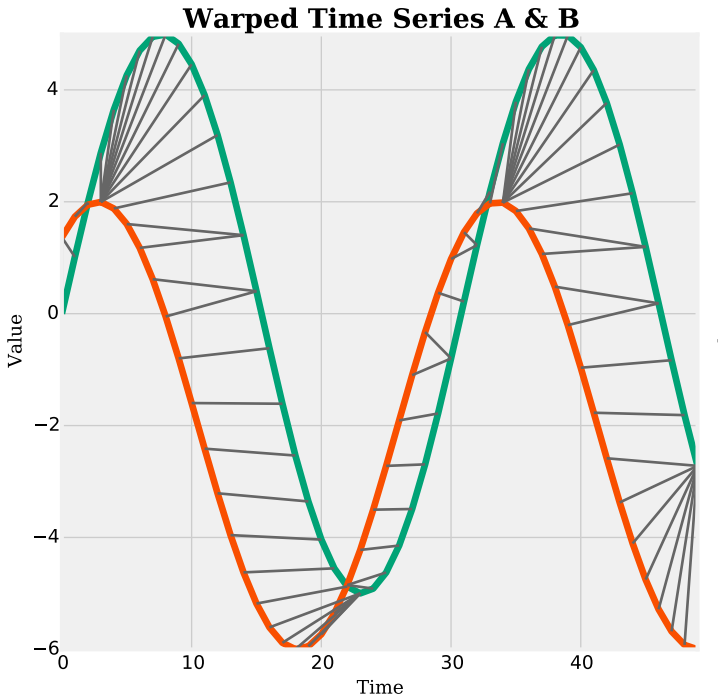


Fig. 2.3 DTW applies a warping of the time axis of two time series to best match each other.

distance functions is **Shape based DTW (ShapeDTW) [143]**, which is a pipeline classifier that converts input series into a range of new time series describing shape features before applying DTW. It can convert a univariate problem into a multivariate problem. A sliding window is run along the series, and a range of summary statistics are found.

The matrix profile [141] of a time series is technically a series-to-series transform that is based on nearest neighbours to sliding windows. The core concept is to run a sliding window of length l over a series, then find the nearest non-overlapping neighbour for each window from all other windows using normalised Euclidean distance. The matrix profile has many uses in related areas to TSC and has been applied to problems such as motif discovery [140] and segmentation [51]. The **Matrix Profile Distance (MPDist) [52]** is a recently introduced distance measure for time series using the matrix profile which can be applied to classification tasks. It defines two time series to be similar if they share many phase-independent subsequences. The MPDist was shown to be robust towards spikes, warping, linear trends, dropouts, wandering baseline and missing values [52].

Historically, distance functions have been used in conjunction with a 1-Nearest neighbour classifier for TSC. Experimentation in [78] showed there was little difference in accuracy between a range of distance measures, but that tuning distance parameters improves performance. It was also observed that when ensembling different 1-nearest neighbour distance classifiers together, the resulting **Elastic Ensemble (EE) [78]** was significantly more accurate than its constituents.

The EE is a combination of 11 nearest neighbour classifiers that use whole series elastic distance measures in the time domain and with first order derivatives. The 11 classifiers in EE are 1-NN with: Euclidean distance; full window DTW; DTW with window size set through cross validation; derivative DTW with full window [62]; derivative DTW with window set through cross validation; weighted DTW [60]; derivative weighted DTW; longest common subsequence; edit distance

with real penalty [27]; time warp edit distance [83], and the move-split-merge distance metric [121].

Work has been made toward improving the runtime efficiency of distance based classifiers. Nearest neighbour classifiers are lazy learners, but tuning the distance parameters introduces a heavy computational overhead. The Fast Elastic Ensemble [95] shows that much of the tuning was unnecessary, and equivalent accuracy could be obtained with a fraction of the computational overhead. [124] describes a caching and cascading lower bounds based approach that provides two to three orders of magnitude speed up for the warping parameter tuning with DTW, then extends the approach to EE, creating FastEE [125].

Proximity Forest (PF) [81] is an ensemble of Proximity Tree base classifiers. PF uses the same 11 distance functions used by EE, but is more accurate and more scalable than EE. At every node of a tree, one of these 11 distances is selected to be applied with a fixed parameter value. An exemplar single series is selected randomly for each class value. At every node, r combinations of distance function, parameter value and class exemplars are randomly selected, and the combination with the highest Gini index split measure is selected. Series are passed down the branch with the exemplar that is closest to it, and the tree grows recursively until a node is pure.

2.2.3 Interval Based

Interval based classifiers extract phase dependent intervals of fixed offsets and compute summary statistics from these subseries. The motivation for taking intervals is to mitigate for confounding noise. Figure 2.4 shows some time series for the classification problem of detecting forged alcoholic spirits. Each series is a light spectrum taken through the bottle, and the forged products contain a lower level of ethanol than the genuine spirits. The greatest variation is in the visible light

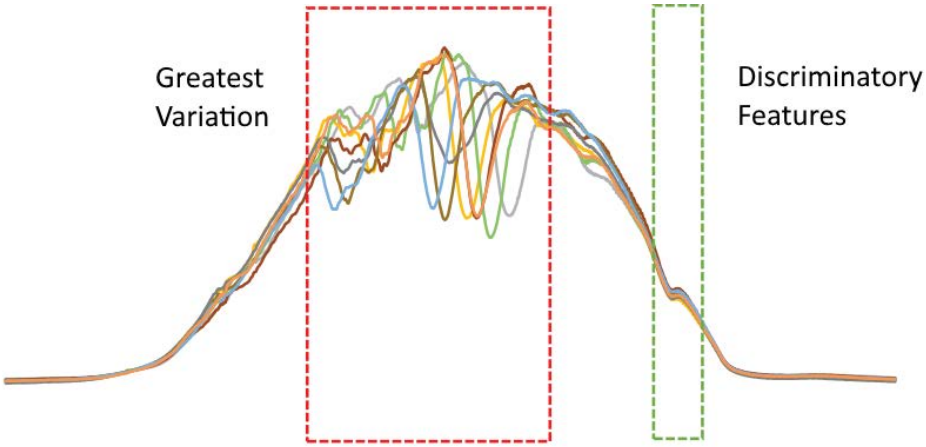


Fig. 2.4 An example of a problem where interval based approaches may be superior. Using intervals containing just the discriminatory range of features is likely to make classification easier.

spectra, whereas the discriminatory features are in the near infrared. Using the whole series may confound the classifier. Figure 2.5 summarises the relationship of the classifiers we detail in this section, and shows how the interval space has developed over time.

Time Series Forest (TSF) [40] is the simplest interval based classifier. As its name implies, TSF is an ensemble of decision trees and is based on the random forest ensemble. For each tree, $\sqrt{m}$ intervals are selected with a random position and length. The same interval offsets are applied to all series. For each interval, three summary statistics (the mean, variance and slope) are extracted and concatenated into a feature vector. This feature vector is used to build the tree, and features extracted from the same intervals are used to make predictions. The ensembles predictions are made using a majority vote of all trees. The TSF base classifier is a modified decision tree classifier referred to as a Time Series Tree, which considers all attributes at each node and uses a metric called margin gain to break ties.

The TSF training process is described in Algorithm 2.1, with a visual aid for the TSF interval feature extraction provided in Figure 2.6. For the esnemble r trees are created, each one from $r \times 3\lfloor\sqrt{m}\rfloor$ features. Random intervals are extracted from

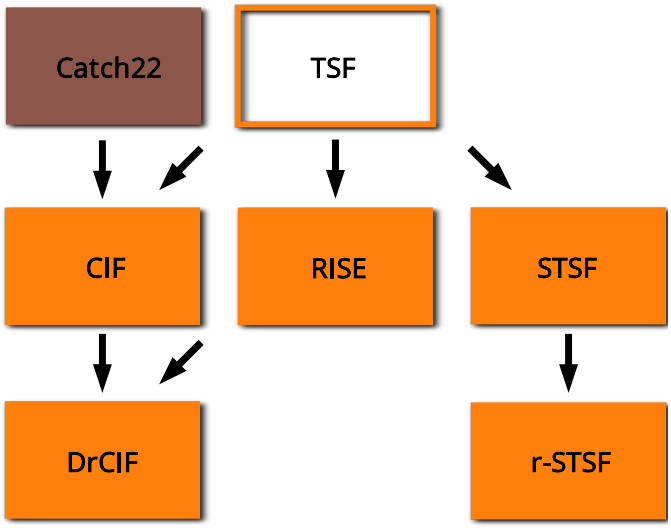


Fig. 2.5 An overview of interval based approaches and the relationship between them. Filled classifiers were released post-bake off. One unsupervised transform relating to CIF in Catch22 is included.

all series at the same offsets (lines 5–7), and their mean, standard deviation and slope are computed (lines 8–10). From these are decision tree is learned (line 11).

Algorithm 2.1 TSF(A list of n cases of length m with d dimensions, $\boldsymbol{T} = (\boldsymbol{X}, \boldsymbol{y})$))

Parameters: the number of trees, r, the number of intervals per tree k

1: Let $\boldsymbol{F} = (\boldsymbol{F}_1 \dots \boldsymbol{F}_r)$ be the trees in the forest
2: **for** $i \leftarrow 1$ to r **do**
3: Let $\boldsymbol{S}$ be a matrix of n cases $(\boldsymbol{s}_1 \dots \boldsymbol{s}_n)$ with ak attributes
4: **for** $j \leftarrow 1$ to k **do**
5: $b = rand(1, m-3)$ *interval length*
6: $l = rand(3, m)$ *interval position*
7: **for** $t \leftarrow 1$ to n **do**
8: $\boldsymbol{s}_{t,3(j-1)+1} \leftarrow mean(\boldsymbol{X}_{t,b:(b+l-1)})$
9: $\boldsymbol{s}_{t,3(j-1)+2} \leftarrow standardDeviation(\boldsymbol{X}_{t,b:(b+l-1)})$
10: $\boldsymbol{s}_{t,3(j-1)+3} \leftarrow slope(\boldsymbol{X}_{t,b:(b+l-1)})$
11: $\mathbf{F}_i.buildTimeSeriesTree(\boldsymbol{S}, \boldsymbol{y})$

Interval based classifiers are generally simple and fast. The best of the class in the bake off [8], TSF, is significantly more accurate than DTW. Nevertheless, TSF was one of the worst performing algorithms of the nine significantly better than DTW, which suggested there was room for improvement. All of the following

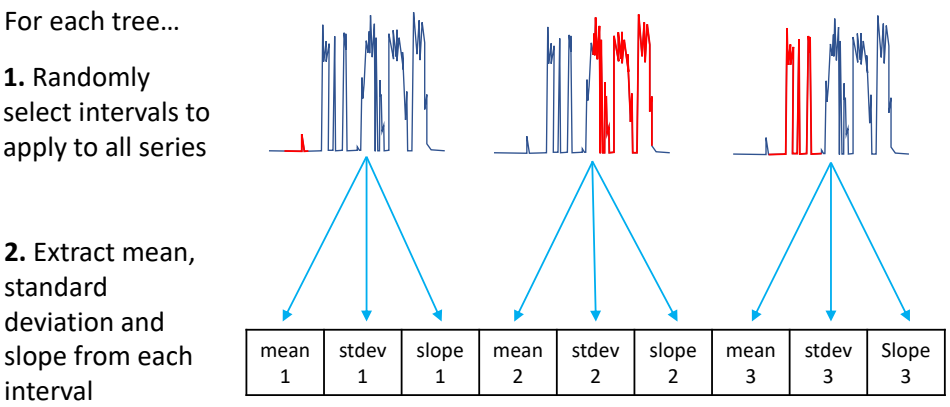


Fig. 2.6 An example of how the Time Series Forest [40] creates its feature vector from each series. Three summary statistics are extracted from each of the randomly selected intervals used by all series.

algorithms can trace their origin to TSF, and are significantly more accurate. Time series bag of features (TSBF) [12] and Learned Pattern Similarity (LPS) [11] are extensions of TSF, but neither perform significantly better nor faster when compared in the survey comparison [8].

First developed for the HIVE-COTE ensemble (described in Section 2.2.7), the **Random Interval Spectral Ensemble (RISE) [44]** is an interval based classifier that uses spectral features. RISE is an ensemble of simple decision trees built using information gain. Unlike TSF, RISE selects a single random interval for each base classifier. The periodogram and auto-regression function are calculated over each randomly selected interval, and these features are concatenated to build the tree. The class of new cases is chosen using a majority vote on tree predictions. RISE was primarily designed for use with audio problems, where spectral features are more likely to be discriminatory.

Supervised Time Series Forest (STSF) [19] is an interval based tree ensemble that includes a supervised method for extracting intervals. Intervals are found and extracted for a periodogram and first order differences representation, as well as the base series. STSF introduces bagging for each tree and extracts seven simple summary statistics from each interval. For each tree, an initial split point for the

series is randomly selected. For both of these splits, the remaining subseries is cut in half, and the half with the higher Fisher score is retained as an interval. This process is then run recursively using the higher scored interval until the series is too small. This is repeated for each of the seven summary statistic features, with the extracted statistic being used to calculate the Fisher score. **Randomised STSF (r-STSF) [20]** is an extension of STSF, altering its components with more randomised elements. The all split points for interval selection are selected randomly instead of splitting each candidate in half after the first, and a randomised binary tree is used to build the extracted features. Intervals extracted from an autoregressive representation are included alongside the previous additions. Features are extracted multiple times from each representation into a single pool, with a subsample of this pool randomly selected for each tree.

The Canonical Interval Forest (CIF) [85] and **The Diverse Representation Canonical Interval Forest (DrCIF) [87]** are also extensions of TSF, incorporating the Catch22 features previous introduced. Chapter 5 describes the development of both these algorithms.

2.2.4 Shapelet Based

Shapelets [139, 94] are phase independent discriminatory subsequences, the presence or absence of which can be an indicator of the class value in a series. The expression of a shapelet in a series is found by moving a sliding window across a series, and computing the Euclidean distance between the shapelet and the underlying window. An example of an extracted shapelet is displayed in Figure 2.7. A shapelet closely matching parts of a series is indicative of the series belonging to the same class as the shapelet. The similarity of a shapelet to a time series is found using the $sDist()$ function, which finds the minimum distance of the shapelet to all subsequences same length of the shapelet from the time series. A visual example of

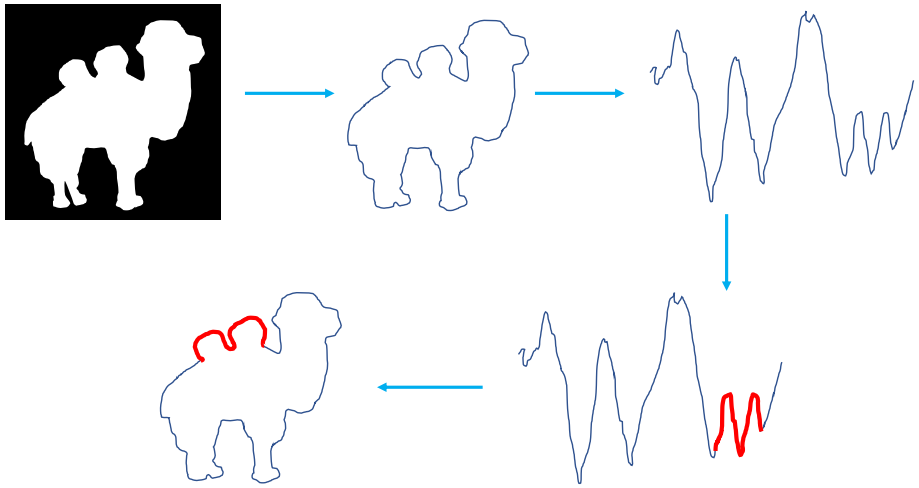


Fig. 2.7 Example of a shapelet extracted from the image outline of a camel. Even if the camel image were to move positions or rotate, altering the time series, the shapelet would still be able to detect the humps by searching through the whole series to detect similar patterns.

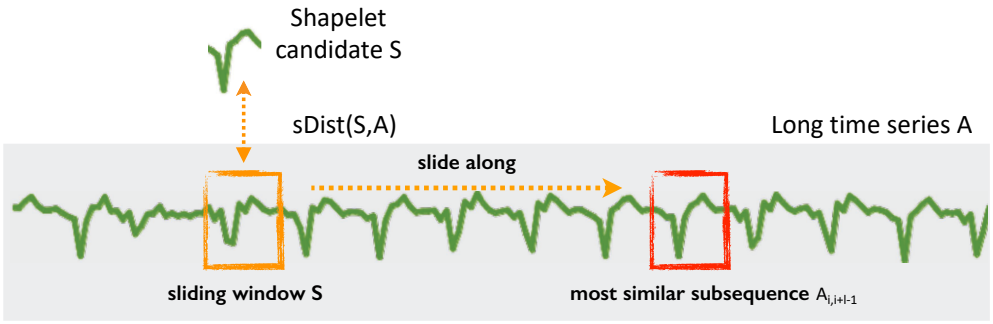


Fig. 2.8 Visualisation of the shapelet distance operation $sDist()$ between a shapelet $\boldsymbol{S}$ and a series $\boldsymbol{A}$, which finds the closest distances to the shapelet from all possible subseries of the same length.

the $sDist()$ function is shown in Figure 2.8. Figure 2.9 shows an overview of the shapelet based algorithms covered in this section.

Shapelets were first proposed as a primitive in [139], and were embedded in a decision tree classifier. The **Shapelet Tree** classifier discovers and embeds shapelets within the nodes of its decision tree at build time. The shapelet in each node is chosen from the train data entering the node through an exhaustive search of all shapelets given a minimum and maximum shapelet length.

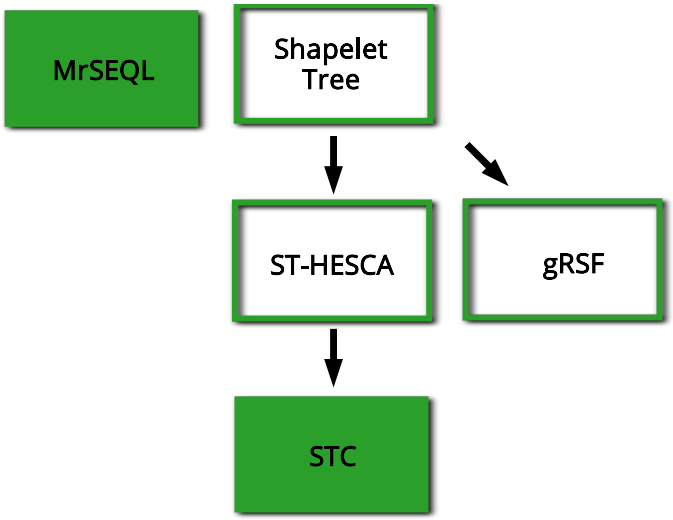


Fig. 2.9 An overview of shapelet based approaches and the relationship between them. Filled classifiers were released post-bake off.

The **Generalised Random Shapelet Forest (gRSF) [61]** is a bagging based tree ensemble that attempts to improve the computational efficiency and predictive accuracy of the Shapelet Tree through randomisation and ensembling. gRSF was proposed as a technique for multivariate TSC. At each node in a generalised tree a dimension, k, is randomly selected. From this dimension, r shapelets are selected from the training set at random. Each shapelet has a randomly selected length between predefined upper and lower limits. The quality of a shapelets is measured in the standard way with $sDist()$ and information gain, and the best is selected. The data is split, and a tree is recursively built until a stopping condition is met. New samples are predicted by a majority vote on the tree's predictions, and multiple trees are ensembled.

The Shapelet Transform Classifier (STC) [58] is a pipeline classifier which searches the training data for shapelets, then transforms the series to vectors of $sDist()$ distances using a filtered set of selected shapelets based on information gain, then builds a classifier on the transformed shapelet distances. This is in contrast to the decision tree based approaches, which search for the best shapelet at each tree node.

The first version of STC performed a full enumeration of all shapelets from all train cases before selecting the best k. The base classifier used was the HESCA (later renamed CAWPE) [68] ensemble of classifiers, a weighted heterogeneous ensemble of 8 classifiers including a diverse set of linear, tree based and Bayesian classifiers. Due to its full enumeration and large pool of base classifiers requiring weights, the algorithm does not scale well. It has been shown that full enumeration often causes overfitting [17]. Recent research has focused on making the transform faster and more useful for classification. The following incremental changes have been made to STC over the course of multiple publications:

1. Search has been randomised, and the search time is now a parameter, unless the data set is so small that enumeration is possible. Search time defaults to one hour, and it has been shown that this does not lead to significantly worse performance on the UCR datasets.

2. Shapelets are now binary, in that they represent the class of the origin series and are evaluated against all other classes as a single class. This facilitates greater use of the early abandon of the order line creation (described in [139]), and makes evaluation of split points faster.

3. The heterogeneous ensemble of base classifiers in HESCA has been replaced with a single Rotation Forest [103] classifier.

While they share the same premise, both parts of the pipeline in the method for selecting shapelets and the base classifier used have changed since the original version. To differentiate both, we have the updated version remain named as STC, while we call the original full enumeration version **ST-HESCA**.

The **Multiple Representation Sequence Learner, MrSEQL [71]**, is an ensemble classifier that extends previous adaptations of the SEQL classifier [70]. MrSEQL looks for the presence or not of a pattern (shapelet) in the data. Rather

than using a distance based approach to measure the presence or not of a shapelet, MrSEQL uses the bag-of-words/dictionary based approach of discretising subsequences into words. Words are generated through two symbolic representations, using SAX [74] for time domain and SFA [107] for frequency domain. A set of discriminative words is selected through Sequence Learner (SEQL) and the output of training is a logistic regression model, which in concept is a vector of relevant subsequences and their weights. Diversification is achieved through the two different symbolic representations and varying the window size.

2.2.5 Dictionary Based

Like shapelet approaches, dictionary based methods [74, 105] also use phase-independent subsequences by sliding a window over time series. But rather than measure distance to a subsequence, the windowing operation is used to count the frequency of occurrence of repeating patterns. The principle of dictionary approaches is that they aim to discriminate based on differences in frequency of patterns between classes, rather than detecting their presence in series. Figure 2.10 shows simulated data where a dictionary based approach may be most appropriate. There are two patterns in the data, which occur in both classes. The discriminatory feature is how often the pattern occurs. Shapelet approaches look for the presence or absence of a pattern, and hence would not be appropriate.

Counting repeating patterns requires some form of discretisation of windows. To achieve this, dictionary based classifiers follow the bag-of-words method commonly employed in fields such as computer vision and audio processing [116, 14, 111]. Each sliding window is compressed, then discretised into a word using a fixed alphabet of symbols. Each series is thus summarised by a histogram of word frequencies, which can then be used as a feature vector for classification. The sparsity

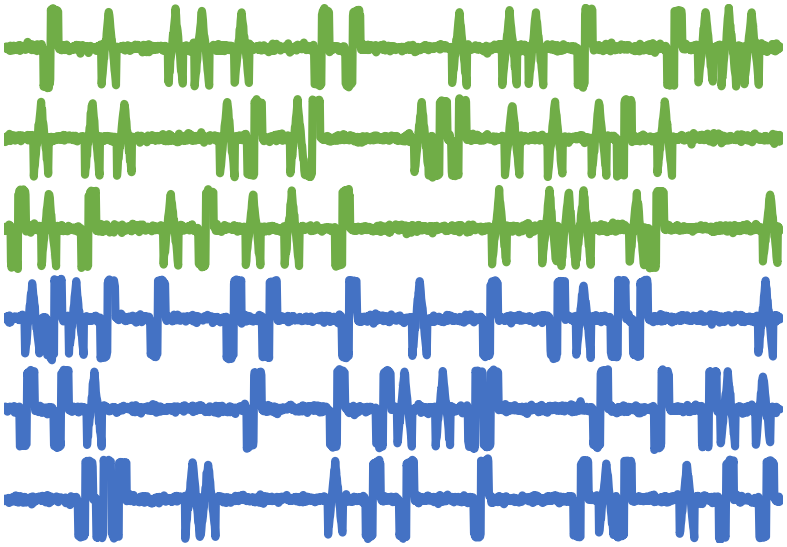


Fig. 2.10 A simulated data to demonstrate the features the dictionary approaches attempt to capture. Both classes have a repetition of two patterns, a spike and a step. The difference is that the spike pattern occurs more commonly in the first class, whereas steps are more common in the second.

of the histogram depends on the alphabet size and the word length. Figure 2.11 shows an overview of the dictionary based algorithms covered in this section.

The **Bag-of-SFA-Symbols (BOSS) [105]** is an ensemble of 1-nearest neighbour classifiers built on dictionaries of words. A bespoke non-symmetric distance function is used to compare dictionaries. BOSS uses cross-validation to form its ensemble, estimating the accuracy of each classifier built and only keeping those within 92% of the highest accuracy classifier.

For its individual classifiers, BOSS converts each sliding window into a word by the use of the Symbolic Fourier transformation (SFA) [107]. Binning partitions the value range into disjoint intervals, called bins. Each bin is labelled by a symbol. A real value that falls into an interval is represented by its discrete label, e.g. A, B, C, D for an alphabet of size 4. The technique BOSS uses to find its bins is called Multiple Coefficient Binning (MCB). The window is transformed into l real and imaginary values using the Fourier transform, and the bins are used to create this window's word. The count for the word found is then incremented in the bag-of-words

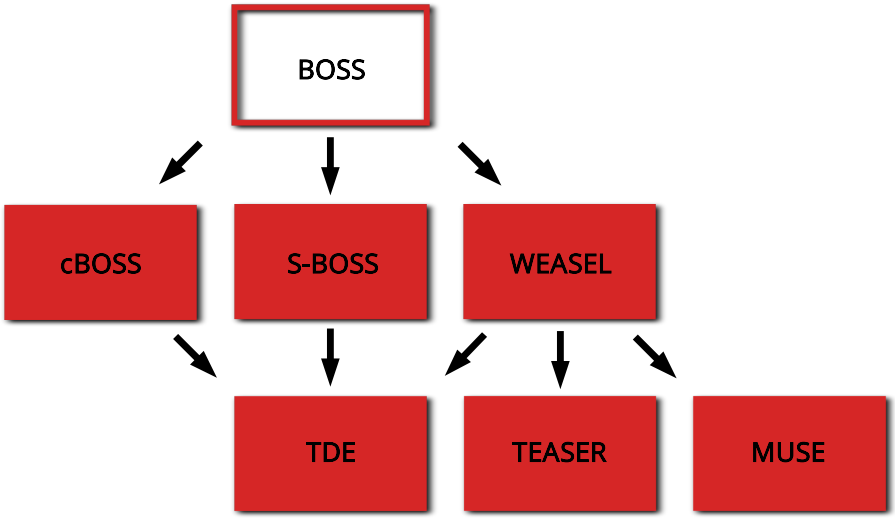


Fig. 2.11 An overview of dictionary based approaches and the relationship between them. Filled classifiers were released post-bake off.

histogram. Figure 2.12 visualises the process BOSS uses to transform a time series into a histogram of word counts

Individual BOSS classifiers have a number of parameters, which heavily influence their performance. Of particular importance are the window length w, whether to normalise the windows p, the word length l, the alphabet size α. The alphabet size is typically fixed to 4. The BOSS ensemble performs a grid search of remaining parameter values to find the top performers for its ensemble. Algorithm 2.2 outlines the BOSS transformation of a set of n series into n histograms. Sliding windows are extracted (line 5), approximated using the Fourier transform (line 6), and the real and imaginary values are discretised using the pre-computed MCB histograms (lines 7–11). Finally, the corresponding histogram count is increased (lines 12–14).

BOSS was one of the best performing algorithms in the bake off paper, and it triggered significant subsequent research into dictionary based classifiers. Dictionary approaches such as (Bag of Patterns) BOP [75] and SAX-VSM [112] built on Symbolic Aggregate Approximation (SAX) [74] performed significantly worse than their counterpart built on SFA.

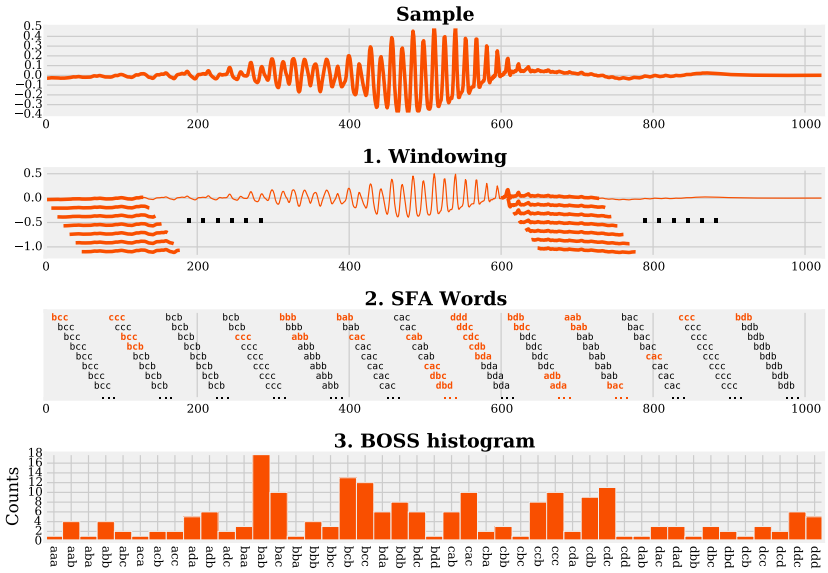


Fig. 2.12 Transformation of a time series into a dictionary model using overlapping windows (second to top), discretisation of windows to words (second from bottom), and word counts (bottom). Figure used in [105]

Algorithm 2.2 BOSS(A list of n cases of length m with d dimensions, $\boldsymbol{T} = (\boldsymbol{X}, \boldsymbol{y})$)

Parameters: the word length l, the alphabet size α, the window length w, normalisation parameter p

Let $\boldsymbol{H}$ be a list of n histograms $(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n)$
Let $\boldsymbol{B}$ be a matrix of l by α breakpoints found by MCB
for $i \leftarrow 1$ to n **do**
  for $j \leftarrow 1$ to $m - w + 1$ **do**
    $\boldsymbol{s} \leftarrow \boldsymbol{x}_{i,j} \ldots \boldsymbol{x}_{i,j+w-1}$
    $\boldsymbol{q} \leftarrow \text{DFT}(\boldsymbol{s}, l, \alpha, p)$ { $\boldsymbol{q}$ *is a vector of the complex DFT coefficients*}
    if p **then**
      $\boldsymbol{q}' \leftarrow (\boldsymbol{q}_2 \ldots \boldsymbol{q}_{l/2+1})$
    else
      $\boldsymbol{q}' \leftarrow (\boldsymbol{q}_1 \ldots \boldsymbol{q}_{l/2})$
    $\boldsymbol{r} \leftarrow \text{SFAlookup}(\boldsymbol{q}', \boldsymbol{B})$
    if $\boldsymbol{r} \neq \boldsymbol{p}$ **then**
      $pos \leftarrow index(\boldsymbol{r})$
      $\boldsymbol{h}_{i,pos} \leftarrow \boldsymbol{h}_{i,pos} + 1$
    $\boldsymbol{p} \leftarrow \boldsymbol{r}$

WEASEL [108] is a pipeline classifier designed around the idea of identifying those words that are most discrimiative for each class, i.e. they are more frequent in one particular class, removing words that are equally frequent among all classes. Just like stop words get removed from natural language. Similar to BOSS, WEASEL performs the Fourier transform on each window, creating words by discretisation, and forms histograms of word counts, which is done for a range of window sizes and word lengths. Other than BOSS, which is based on ensembling individual classifiers, one for each parameter range, to increase its robustness and accuracy, WEASEL builds one high-dimensional feature vector per time series using a wide range of different parameters. It then identifies the most discriminate words by the use of (a) a supervised symbolic discretisation, (b) bigrams of words, (c) feature selection and (d) ridge regression. WEASEL extracts histograms by windowing over a time series for different parameter ranges. A supervised discretisation retains the best class-separating Fourier terms by the application of an ANOVA F-test. The retained Fourier terms are discretised using Information Gain Binning (IGB), in a manner similar to MCB in BOSS. Bigrams of the previous non-overlapping window for each word are added to the histogram. All words of different parameter ranges are concatenated into one histogram, forming a single high-dimensional feature vector per instance. WEASEL then applies Chi-squared feature selection to each histogram, removing all words below a threshold. Finally a ridge classifier is trained to further up-weight most discriminative words per class.

The Multivariate Symbolic Extension (MUSE, or WEASEL-MUSE) [109] is a variant of the bag-of-words model for classifying multivariate time series. MUSE builds on the same pipeline as WEASEL by creating a single high-dimensional feature vector per instance, which is a concatenation of all dimensions of the MTS. It first applies the sliding-window approach to each dimension of the MTS separately, extracting discrete features per window and dimension using a symbolic dimensionality reduction. The resulting histograms are concatenated into one his-

togram, forming a single high-dimensional feature vector per instance, whereas each dimension forms its own dictionary, to distinguish between words from different dimensions. This feature vector is subsequently fed through Chi-squared feature selection, removing non-discriminative features, and trained through a ridge regression classifier. To increase diversification, the same pipeline is applied to the first order differences of the training data, too.

Bag-of-words approaches, such as BOSS, are by design invariant to the locations of words in a time series, due to counting occurrences over a dictionary. However, for some datasets the position of a word contains information about the class of a time series, i.e. some words may gain importance only when in a particular location, or mutually occurring words in different locations may be indicative of separate classes. **SpatialBOSS (S-BOSS) [67]** incorporates spatial pyramids into the BOSS algorithm to add locality information to word counts. Spatial pyramids are commonly used in computer vision for adding locality information [69]. A spatial pyramid recursively divides a time series into halves, building a bag on each section. At the top level of the pyramid, a bag over the whole series is constructed, baring no locality information. While at each lower level, more locality information is retained in the bag. The number of splits and therefore the amount of locality considered is defined by the height of the pyramid. At the lower scales the locality of a word can be traced down to a small section of the time series, e.g. the word is present in the first, second, third, or fourth percentile of the time series for level 3. Bags at smaller scales may be weighted to give local word occurrences higher importance than global word occurrences. Finally, the histograms for each scale are concatenated, while keeping a unique dictionary for each split, to form one singular feature vector per instance. S-BOSS creates BOSS transformations using the standard BOSS algorithm for each section. Cross-validation is applied to find the best height of the pyramid, which defines the degree of locality added to the model.

The size of the parameter grid searched by BOSS is data dependent, and BOSS uses a method of retaining ensemble members using a threshold on training accuracy. This makes it unpredictable in its usage of time and memory resources. BOSS was one of the slower algorithms tested in the bake off and turned out to be impractical for larger datasets. Chapter 4 focuses on improving these flaws in the BOSS approach, describing the **Contractable BOSS (cBOSS) [88]** and the **Temporal Dictionary Ensemble (TDE) [86]** algorithms.

2.2.6 Convolution Based

Shapelets are essentially convolutions, with a min-pooling operation on the array of windowed Euclidean distances. The similarity between shapelets and convolutions was first observed in [54]. One of the main difference between convolutions and shapelets is that shapelets are searched for directly using subsequences extracted from the training data as candidates, whereas convolutions are found through searching the entire space of possible values. The very large search space for all possible shapelets has meant that convolutional neural networks (CNNs) have often not performed that well. We maintain a separation between the approaches, but note that they are fundamentally similar and numerous attempts have been made to integrate the two areas. Figure 2.13 shows an overview of the convolution based algorithms covered in this section.

CNNs use convolutional kernels to detect patterns in the input, and learn the weights of these kernels. The kernel is convolved with an input time series by using a sliding dot product, also referred to as cross-correlation. This process is essentially the same as the sliding window algorithm used with shapelets. The result of the convolution operation of a kernel with a time series is an activation array/map of size $m-l+1$, where l is the kernel length. It represents the extent to which the pattern, represented by the kernel, is present in a time series. The activation map

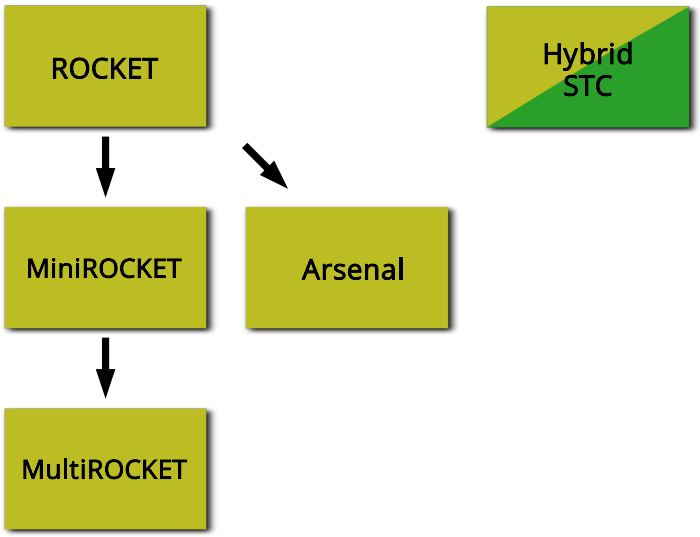


Fig. 2.13 An overview of convolution based approaches and the relationship between them. All classifiers were released post-bake off.

can be used as a feature vector for a classifier from the machine learning repertoire. However, a pooling operation is usually applied after convolution to reduce the size of the activation map to one value per operation. For example, a max-pooling operation extracts the maximum value from each activation array. Kernels are often made more complex than shapelets, and include parameters such as dilation, which stretches the convolution by skipping values in the comparison series.

The balance between searching for shapelet candidates in the real numbers using gradient descent or by restricting the set of candidates to the training data was explored with the **Hybrid STC [55]**. This hybrid approach performs a fast shapelet candidate search in the training data, which are then used to seed a convolutional neural network. It was found to be more accurate than using just the training data driven discovery or solely the gradient descent discovery. However, the most successful recent use of convolutions in TSC is through using randomised convolutions in a pipeline classifier.

The most well known convolutional approach is the **Random Convolutional Kernel Transform (ROCKET) [37]**. ROCKET is a pipeline classifier. It generates

a large number of randomly parameterised convolutional kernels (typically in the range of thousands to tens of thousands), then uses these to transform the data through two pooling operations: the max value and the proportion of positive values (PPV). These two features are concatenated into a feature vector for all kernels. For k kernels, the transformed data has $2k$ features.

In ROCKET, each kernel is randomly initialised with respect to the following parameters: The *kernel length* l, randomly selected from $\{7,9,11\}$; The *kernel weights* w, randomly initialised from a normal distribution; A *bias term* b added to the result of the convolution operation; The *dilation* d to define the spread of the kernel weights over the input instance, which allows for detecting patterns at different frequencies and scales; and *padding* p the input series at the start and the end (typically with zeros), such that the activation map has the same length as the input.

The feature vectors are then used to train a ride regression classifier using cross-validation to train the L_2-regularisation parameter α. A logistic regression classifier is suggested as a replacement for larger datasets. The combination of ROCKET with logistic (ridge) regression is conceptually the same as a single-layer CNN with randomly initialised kernels and softmax loss.

Algorithm 2.3 illustrates the process of rocket transformation and training a ridge classifier. First, k random kernels are initialised. Each time series is convoluted with the kernel, max and ppv pooling are applied to extract two features each, and a feature map is constructed from these two features over all time series (lines 4–7). A ridge classifier is learned through cross-validation on the novel feature maps.

ROCKET is a fast classifier that also performs well in terms of accuracy on the UCR datasets. ROCKET has two extensions. The first of these extensions is **MiniROCKET [38]**, which speeds up ROCKET by over an order of magnitude with no significant difference in accuracy. MiniROCKET removes many of the random components of ROCKET, making the classifier almost deterministic. The

Algorithm 2.3 ROCKET(A list of n cases of length m with d dimensions, $\boldsymbol{T} = (\boldsymbol{X}, \boldsymbol{y})$)

Parameters: the number of kernels per classifier, k

$\boldsymbol{F} \leftarrow$ initialize matrix of dimensionality $n \times 2k$
for $j \leftarrow 1$ to k **do**
  $\boldsymbol{\omega} \leftarrow randomKernel()$
  for $i \leftarrow 1$ to n **do**
    $\boldsymbol{A} \leftarrow sliding\_dot\_product(\boldsymbol{X}_i, \boldsymbol{\omega})$
    $\boldsymbol{F}_{i,(2j-1)} \leftarrow \max(\boldsymbol{A})$
    $\boldsymbol{F}_{i,2j} \leftarrow ppv(\boldsymbol{A})$
$buildRidgeClassifierCV(\boldsymbol{F}, \boldsymbol{y})$

kernel length fixed to 9, only two weight values are used, and the bias value is drawn from the convolution output. Only the PPV is extracted, discarding the max. These changes alongside general optimisations taking advantage of the new fixed values provide a considerable speed-up to the algorithm. **MultiROCKET [123]** further extends the MiniROCKET improvements, extracting features from first order differences and adding three new pooling operations extracted from each kernel: mean of positive values (MPV), mean of indices of positive values (MIPV) and longest stretch of positive values (LSPV).

Using its linear classifier, ROCKET struggles to produce high quality probability estimates if required. In Chapter 6 we propose an extention of ROCKET, **the Arsenal [87]** to mitigate this issue.

2.2.7 Hybrid Algorithms

The nature of the data and the problem dictate the appropriateness of each approach. The most accurate algorithms over data with no apriori knowledge of the best approach combine multiple transformations types in a hybrid algorithm. We define a hybrid algorithm as one which by design encompasses or ensembles multiple of the discriminatory representations we have previously described. Figure 2.14

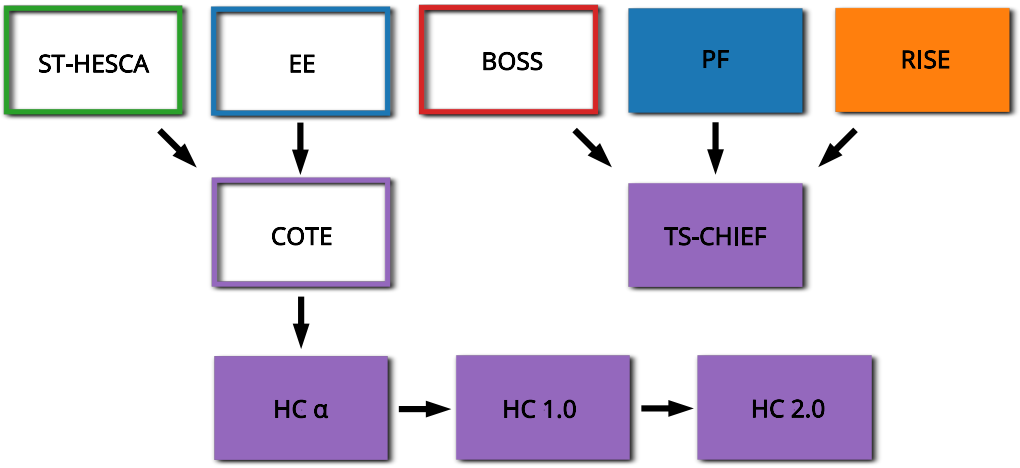


Fig. 2.14 An overview of hybrid approaches coloured purple and the relationship between them. Filled classifiers were released post-bake off.

shows the classifiers we detail in this section and the classifiers from different representations which inspired them.

The best performing approach in the bake off [8] by a significant margin was the **Collective of Transformation Ensembles (COTE) [9]**, which at the time was the only algorithm that explicitly ensembles over different representations. Since renamed Flat-COTE due to its structure: it is an ensemble of 35 time series classifiers built in the time, auto-correlation, power spectrum and shapelet domains. The components of the ST-HESCA [58] and EE [76] ensembles are pooled with classifiers built on autocorrelation (ACF) and power spectrum (PS) representation. All together, this includes the 8 classifiers built on the shapelet transform from ST-HESCA, the 11 elastic distance 1-nearest neighbour classifiers from EE and the 8 HESCA classifiers built on ACF and PS transformed series. A weighed vote is used to determine new cases, with each classifier being weighted using its train set cross-validation accuracy.

The COTE family of classifiers has evolved since Flat-COTE, and new hybrid algorithms have been produced following the success shown by pooling multiple representations. **The Hierarchical Vote Collective of Transformation Ensem-**

bles (HIVE-COTE) [78] was proposed to overcome some of the problems with Flat-COTE. This first version of HIVE-COTE, subsequently called HIVE-COTE$_\alpha$ (HC$_\alpha$), is a heterogeneous ensemble containing five modules each from a different representation: EE from the distance based representation; TSF [40] from interval based methods; BOSS [105] from dictionary based approaches and ST-HESCA from shapelet based techniques. The fifth module is introduced in the spectral based RISE, also using intervals. The five modules are ensembled using the Cross-validation Accuracy Weighted Probabilistic Ensemble (CAWPE, known at the time as HESCA) [68]. CAWPE uses a tilted probability distribution using exponential weighting of probabilities estimated for each module found through cross-validation on the train data. The weighted probabilities from each module are summed and standardised to produce the HIVE-COTE probability prediction. Whilst state of the art in terms of accuracy, HC$_\alpha$ scales poorly. A range of improvements to make HIVE-COTE more usable were introduced in 2020 with HIVE-COTE 1.0 (HC1) [5], using improvements developed in this thesis. HC1 has four modules: it drops the computationally intensive EE algorithm. BOSS is replaced by the more configurable cBOSS [88]. The improved randomised version of STC [17] is included with a default one hour shapelet search and the Rotation Forest classifier. TSF and RISE had usability improvements. HC1 is designed to be contractable, in that you can specify a maximum train time.

In Chapter 6, we further update HIVE-COTE address scalability issues and reflect recent innovations to individual TSC representations with **HIVE-COTE 2.0 (HC2) [87]**.

The Time Series Combination of Heterogeneous and Integrated Embedding Forest (TS-CHIEF) [114] is a homogeneous ensemble where hybrid features are embedded in tree nodes rather than modularised through separate classifiers. The TS-CHIEF is made up of an ensemble of trees which embed distance, dictionary and spectral base features. At each node, a number of splitting criteria from each

of these representations are considered. These splits use randomly initialised parameters to help maintain diversity in the ensemble. The dictionary based splits are based on BOSS, distance splits based on EE and interval splits based on RISE. The goal of TS-CHIEF was to obtain the benefits of multiple representations without the massive processing requirement of the original HIVE-COTE.

2.2.8 Deep Learning

Despite their strength in making predictions for 2D image data, first shown in AlexNet's performance on the ImageNet dataset [65], deep learning approaches have only recently been heavily studied in the 1D time series data. The first large-scale comparative study of deep learning architectures for TSC was supplied in [42]. This study is limited to end-to-end discriminative models, leaving out generative models which tend to be less accurate and more complex [8, 70] and aiming to reduce bias due to manually designed features [96]. Nine architectures were evaluated on 85 univariate UCR archive datasets and 13 multivariate datasets, including convolution based Fully Convolutional Neural Networks (FCNs) [134] and Multi-scale Convolutional Neural Network (MCNN) [70] approaches, and echo state networks such as the Time Warping Invariant Echo State Network (TWIESN) [126]. We direct interested readers to the full review for a detailed overview of the types of networks compared and decisions made in building each algorithm.

The Residual Network, ResNet [134] was found to be significantly better than all other deep learning architectures on the univariate datasets, and on all univariate and multivariate datasets combined. ResNet is a network of three consecutive blocks, each comprised of three convolutional layers, which are connected by residual 'shortcut' connections that add the input of each block to its output. Residual connections allow the flow of gradient directly through the network, combating the

vanishing gradient effect [57]. The residual blocks are followed by global average pooling and softmax layers to form features and subsequent predictions.

Many other deep learning approaches have been suggested since the deep learning comparison with mixed success. The Time Series Attentional Prototype Network (TapNet) [142] is designed for multivariate time series data and claims state-of-the-art performance. However, we were unable to reproduce this success satisfactorily in our multivariate experiments in later chapters and exclude it from our results.

Building on the success of ResNet, InceptionTime [43] incorporates Inception modules [122] and ensembles over five instantiations of the network each with random initial weights for greater stability. A single network out of the ensemble is composed of two blocks of three Inception modules each, as opposed to the three blocks of three traditional convolutional layers in ResNet. These blocks maintain residual connections, and are followed by global average pooling and softmax layers as before.

Figure 2.15 summarises the Inception Module. It takes a multivariate series input, and first uses a bottleneck layer with length and stride 1 to reduce the dimensionality to $d' < d$ while maintaining output length m, reducing the number of parameters to learn at later stages. Convolutions of different lengths are applied to the output of the bottleneck layer to find patterns of different sizes. The outputs of these convolutions are combined with an additional source of diversity, a Max Pooling followed by bottleneck (with the same value of d') applied to the original time series, and all stacked to form the dimensions of the output multivariate time series to be fed into the next layer.

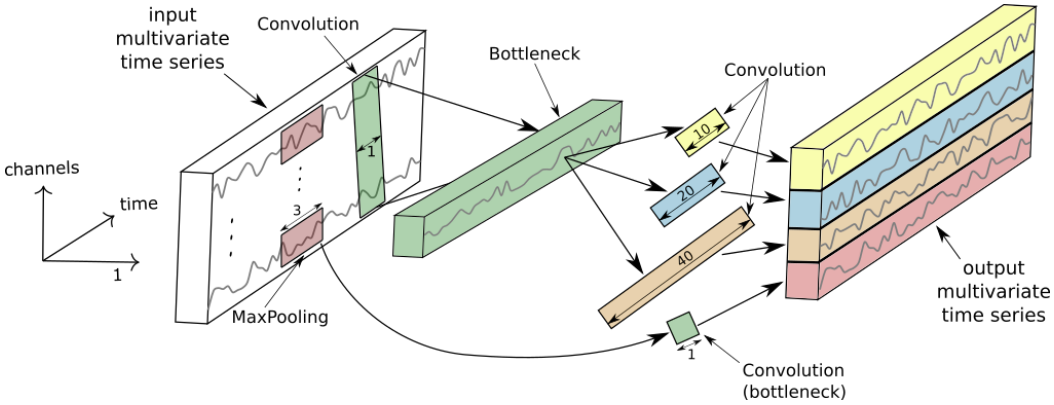


Fig. 2.15 An Inception module with example parameters, figure from [43]. Three of these are concatenated to form a block in InceptionTime.

2.2.9 Model Based

Defined in [8], model based algorithms are comparatively underrepresented compared to other categories and cover a wider range of approaches. These approaches include auto-regressive models [6, 31], hidden Markov models [117] and kernel models [26]. Model based approaches are usually for tasks other than classification [82]. and the approaches were deemed not competitive for classification apart from specific datasets in the informal opinion of the survey writers [8]. A niche for model based approaches is in classification for long series with unequal length [6].

2.2.10 Early Classification

Early TSC (eTSC) is the problem of classifying a time series after as few measurements as possible with the highest possible accuracy. eTSC was first introduced in [138]. The most critical issue of any eTSC method is to decide when enough data of a time series has been seen to take a decision. eTSC has two goals: Classifying time series early and with high accuracy. However, these two goals are contradictory in nature: The earlier a time series has to be classified, the less data is available for this classification, usually leading to lower accuracy. In contrast, the higher the desired classification accuracy, the more data points have to be inspected and

the later eTSC will be able to take a decision. Thus, a critical issue in any eTSC method is the determination of the point in time at which an incoming time series can be classified. A common method for eTSC is to have a set number of series length thresholds, and build a separate TSC classifier for each of them. It is then a job of an eTSC controller to decide whether a prediction made at a certain time threshold is reliable to use, or if more data is required to make a safe prediction

Two-tier Early and Accurate Series Classifier (TEASER) [110] is an extension of WEASEL [108] for eTSC. It is an algorithm that models eTSC as a two-tier classification problem: In the first tier, a slave classifier periodically assesses the incoming time series to compute class probabilities. However, these class probabilities are only used as output label if a second-tier master classifier decides that the predicted label is reliable enough, which can happen after a different number of measurements. Intuitively, TEASER trains a pair of classifiers for varying prefix lengths, which are factions of the full time series length m: A slave classifier computes class probabilities which are passed on to a master classifier that decides whether these probabilities are high enough that a safe classification can be emitted. When confronted with a new time series, TEASER waits for the next data points to arrive and then calls the appropriate slave classifier which outputs probabilities for all classes. Next, TEASER passes these probabilities to the slave's paired master classifier which either returns a class label or NIL, meaning that no decision could be derived, and keeps waiting for more data until the next prefix length is met. TEASER uses WEASEL as its slave classifier to output class probabilities and a one-class SVM as its master classifier. By default, total of 20 pairs of slaves/masters are trained for each fraction of $20/s$ with $s \in [1 \cdots 20]$.

SR1-CF1 [92] approaches eTSC as an optimisation problem, balancing accuracy and earliness using a stopping rule and cost function. SR1-CF1 trains group of Gaussian Process classifiers, each on data of a set time series length. A stopping rule based on the probablitiy output of a classifier and amount of series present

when making a predicsion is used to decide whether a decision should be deemed safe or not. The training process finds weights for this stopping rule, deciding how confident probabilites must be and how trustworthy low series length predictions are. A genetic algorithm is used to select the best values by maximising the cost function. The cost function balances accuracy and earliness, with a settable α parameter available to weight the cost function towards either.

2.3 Vector Machine Learning Algorithms

Many TSC approaches use vector based algorithms as a base to train using features extracted from the series data. This section describes the vector machine learning algorithms mentioned previously in this chapter and seen in later chapters. In this thesis we use the term traditional, standard or vector classifier interchangeably to refer to classification algorithms which use a feature vector data format to differentiate them from TSC algorithms. These algorithms do not assume any ordering in the features provided, and as such do not extract any information from an attribute's position or from groups of attributes which close together. This makes vector based algorithms generally poor at classifying time series data. When applying time series transformations selecting the correct classifier for the resulting features is important, however. We introduce the traditional classifiers used in this thesis as a benchmark against the TSC algorithms we present, or as base classifiers for TSC algorithms introduced previously.

2.3.1 *k*-Nearest Neighbour

Nearest neighbour classification is one of the oldest and simplest methods of classification in the field of machine learning, since its introduction in [33] the

nearest neighbour classifier still remains relevant as both a benchmark and as a component of more complex algorithms.

The nearest neighbour classifier makes its predictions by looking at the closest instance to a new case in the train dataset and assigning it the same class label. For this, a measure of (dis)similarity or distance is required, the most commonly used being Euclidean distance.

$$EuclideanDistance(\boldsymbol{p},\boldsymbol{q}) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{2.4}$$

The assumption that the closest neighbour will be of the same class is not always the case, however. The *k*-Nearest Neighbour (k-NN) algorithm looks at the closest *k* instances and votes on the class of new instances from these.

With the classifiers simplicity comes a few quirks. The classifier is sensitive to a difference in scale between attributes and irrelevant features. For the difference in scale, it is common to normalise each feature to 0 mean and 1 standard deviation, as features with large values will produce a larger distance. *k*-NN is a lazy classifier, meaning to train it just has to store the data. The distances to each train instance must be computed for each new case, however. The time required to make a prediction with k-NN can be substantial when using a large train dataset, though work has been done aiming to improve this [49, 1].

2.3.2 Naive Bayes

Naive Bayes is based on Bayes Theorem shown formulated by Thomas Bayes (1702 - 1761).

$$P(y|\boldsymbol{x}) = \frac{P(y)P(\boldsymbol{x}|y)}{P(\boldsymbol{x})} \tag{2.5}$$

Where $\boldsymbol{x}$ is a feature as part of a feature vector and y is a class value, the formula will give an estimate of the probability of class y given $\boldsymbol{x}$. Naive Bayes has a strong

assumption of independence between features used in it, giving it the 'naive' part of its name. The classifier is inherently probabilistic, outputting a set of probabilities of an instance being each class instead of the predicted class value, though most non-probabilistic classifiers can estimate posterior probabilities.

The classifier can be extended upon to take into account dependant attributes, as the assumption of independence Naive Bayes has is unrealistic in a lot of situations. This is called a Bayesian Network, it works by representing the features as nodes in a directed acyclic graph. Using this representation child nodes can be affected by the results of their parent, one implementation of such is shown in [30].

2.3.3 Support Vector Machine

The Support Vector Machine (SVM) is a linear classifier first introduced in [32] which holds similarities to older linear perceptron classifiers. The goal of the SVM is to find the optimal weights and bias offset to form a hyperplane to separate the problem classes, this is most effective with problems that are linearly separable. Separating the SVM from other linear classifiers is that it attempts to obtain these values by finding the maximum margin between classes present. This is done by selecting appropriate 'support vectors' from the closest data points between the classes from which the optimal hyperplane is derived.

In a lot of cases however data is not linearly separable, a solution to this problem can be found in the Kernel Trick. By using kernel functions to transform the data to a higher dimensional space, it is possible to separate the problem linearly despite the inability to do so in the original data. Common kernels for doing this are polynomial kernels or radial basis function kernels. The SVM is notoriously sensitive to its parameters values, with its kernel, regularisation parameter and gamma for certain kernels needing to be tuned for it to perform well.

2.3.4 C4.5 Decision Tree

The C4.5 decision tree classifier [100] is one of the most well known of the decision tree based classifiers [63, 101, 46], built on the author's previously developed classifier the ID3 decision tree [99]. The process used by the C4.5 and other decision tree classifiers is to recursively select an attribute at each tree node, then split the data based on the selected attributes values. If any of the data in the split contains more than a single class, a child node is created and another attribute is chosen to separate the data with until all leaf nodes are of a single class. C4.5 improves upon its predecessor ID3 by using information gain ratio instead of information gain to decide which attributes are most informative at each tree node, allows for continuous attributes and prunes the tree by removing nodes in an effort to reduce overfitting.

2.3.5 Random Forest

Random Forest [18] is a homogenous ensemble of decision tree classifiers. Random forest uses bagging as a method for injective diversity into the ensemble, selecting a random subset of the training data to build each tree. Additional diversity is generated through the use of a Random Tree base classifier, selecting a random subset of attributes at each tree node. The forest builds k trees total, and predictions are made using a majority vote of from the tree outputs. The strength of Random Forest lies in its diversity, with a large enough sample of trees with low correlation (in this thesis 500 is usually the default) the ensemble can heavily mitigate the inherent overfitting seen in decision trees.

2.3.6 Rotation Forest

Rotation forest [103] is also an ensemble of decision trees classifiers, and shares similarities with Random Forest. For each tree attributes are split into a number of

distinct groups. A subsample of training instances are taken for each group from a random subset of class values. This subsample is transformed using Principal Component Analysis (PCA), and is concatenated with the transformed data of other groups. This new feature vector is used to build a C4.5 decision tree, with k trees built in the forest. Majority vote is used to make a final prediction.

An evaluation of Rotation Forest found that it performs well with continuous attributes [4]. As time series are made up of continuous attributes and most time series transformations output continuous attributes, the algorithm often sees use as a base classifier and benchmark for TSC [8].

2.3.7 Gaussian Process

While this thesis focuses on the task of classification, in Chapter 4 we find ourselves with a regression problem when selecting ensemble parameters based on the accuracy performance of previous parameter sets. While classification tasks output one of a finite set of class values, for the regression task we want to predict a real value. For this task, we use a simple Gaussian Process regressor.

A Gaussian Process [135] describes a distribution over functions, $f(\boldsymbol{x}) \sim \mathscr{GP}(m(\boldsymbol{x}, k(\boldsymbol{x}, \boldsymbol{x}')))$, characterised by a mean function, $m(\boldsymbol{x})$, and a covariance function, $k(\boldsymbol{x}, \boldsymbol{x}')$, such that

$$m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] \tag{2.6}$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}\left[(f(\boldsymbol{x}) - m(\boldsymbol{x}))(f(\boldsymbol{x}') - m(\boldsymbol{x}'))\right] \tag{2.7}$$

where any finite collection of values has a joint Gaussian distribution. Commonly the mean function is constant, $m(\boldsymbol{x}) = \gamma$, or even zero, $m(\boldsymbol{x}) = 0$. The covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$ encodes the expected similarity of the function evaluated at pairs of input-space vectors, $\boldsymbol{x}$ and $\boldsymbol{x}'$. For example, the squared exponential covariance

function,

$$k(\boldsymbol{x},\boldsymbol{x}') = \sigma_f^2 \exp\left\{-\frac{(\boldsymbol{x}-\boldsymbol{x}')^2}{2\ell^2}\right\} \tag{2.8}$$

encodes a preference for smooth functions, where ℓ is a hyperparameter that specifies the characteristic length-scale of the covariance functions (large values yield smoother functions) and σ_f governs the magnitude of the variance.

Typically, in a regression setting the response variables of the training samples, $\mathscr{D} = \{(\boldsymbol{x}_i, y_i) \mid i = 1,2,\ldots,n\}$, are assumed to be realisations of a deterministic function that have been corrupted by additive Gaussian noise, i.e.

$$y_i = f(\boldsymbol{x}_i) + \varepsilon_i, \qquad \text{where} \qquad \varepsilon_i \sim \mathscr{N}\left(0, \sigma_n^2\right) \tag{2.9}$$

In that case, the joint distribution of the training sample, and a single test point, $\boldsymbol{x}_*$, is given by,

$$\begin{bmatrix} \boldsymbol{y} \\ f_* \end{bmatrix} \sim \mathscr{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K} + \sigma_n^2 \boldsymbol{I} & \boldsymbol{k}_* \\ \boldsymbol{k}_*^T & k(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix}\right) \tag{2.10}$$

where $\boldsymbol{K}$ is the matrix of pairwise evaluation of the covariance function for all points belonging to the training sample and $\boldsymbol{k}_*$ is a column vector of the evaluation of the covariance function for the test point and each of the training points. The Gaussian predictive distribution for the test point is then specified by

$$\bar{f}_* = \boldsymbol{k}_*^T \left(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y} \tag{2.11}$$

$$\mathbb{V}[f_*] = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^T \left(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}_* \tag{2.12}$$

The hyperparameters of the Gaussian process can be handled by tuning them, often via maximisation of the marginal likelihood, or by full Bayesian marginalisa-

tion, using an appropriate hyper-prior distribution. For further details, see Williams and Rasmussen [135].

Chapter 3

Experimental Methodology

In this chapter, we describe the experimental methodology used throughout our evaluation in the remaining chapters of this thesis. The experimentation covers the comparison of both multiple classification algorithms over multiple datasets and multiple algorithms over a single dataset. The following sections will explain the tests used, the types of figures we produce for both cases, and the various performance statistics used in our evaluation. We introduce the dataset archives used throughout our algorithmic improvements, as well as individual case study datasets. Finally, we discuss the practicalities of running our experiments, describing the environment we run our experiments in and the open source packages we use in our experiments and where contribute our algorithms to aid in reproducing our results.

3.1 Statistics for Evaluating Performance

When comparing classification algorithms using any of our methods for displaying results, we use a multitude of performance statistics. While simple metrics such as error and wins/losses are useful to report, they do not accurately capture many important aspects which can be used to determine the strengths and weaknesses of

a classifier. In the following, we describe and define the performance metrics used throughout this thesis.

Our dataset is defined as $\boldsymbol{T} = \{(\boldsymbol{X}_1, y_1), \ldots, (\boldsymbol{X}_n, y_n)\}$ made up of n cases, where y is drawn from a set of c possible class labels $y \in \{1, \ldots, c\}$. When making predictions on new cases, classification algorithms output a probability distribution of all class labels $\hat{\boldsymbol{p}} = \{p(y=1), \ldots, p(y=c)\}$ for each case. Using these probabilities, the predicted class for a case $\hat{y}$ is the class label with the maximum probability.

$$\hat{y} = \arg\max_{i=1\ldots c} \hat{p}(i) \tag{3.1}$$

For algorithms which are unable to output probability estimates, a vector filled with 0 values except for the estimate of the predicted class, which is set to 1, is used. Our first metrics aim to measure the correctness of classifiers predictions. We define a function for the correctness of a prediction, where a value of 1 is returned for a correct prediction and 0 is return for an incorrect one.

$$f(y, \hat{y}) = \begin{cases} 1, & y = \hat{y} \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

The most basic statistic for comparing classifiers, and the statistic we most commonly report is accuracy (the inverse of error).

$$Accuracy = \frac{\sum_{i=1}^{n} f(y_i, \hat{y}_i)}{n} \tag{3.3}$$

Accuracy can sometimes be misleading when a problem has an imbalance in the number of class labels, however. For example, if a binary problem has 100 cases to predict where 98 cases belong to the first class and the remaining 2 cases to the second, a simple function predicting every case as the first class will have an accuracy of 0.98. Balanced accuracy mitigates the impact of class imbalance by

determining the proportion of correct values for each class.

$$BalancedAccuracy = \frac{\sum_{j=1}^{c} \frac{\sum_{i=1}^{n} f(f(y_i,j),f(y_i,\hat{y}_i))}{\sum_{i=1}^{n} f(y_i,j)}}{c} \tag{3.4}$$

When comparing binary classification problems, the importance of correctly predicting one class may differ from the other. This is the case for our BT nuisance calling data, where correctly classifying the negative class in legitimate call centres is much more important. While classifying a nuisance call centre as legitimate maintains the status quo somewhat, misclassifying a legitimate call centre as nuisance can have disastrous effects if action is taken using the predicted label. In these cases, metrics which measure the performance of individual class values such as sensitivity and specificity are useful. We measure these statistics using the values from a confusion matrix. Given a 'positive' and 'negative' pair of class labels, a confusion matrix displays the number of true negatives (TN) and true positives (TP), which are the number of negative classes correctly predicted, and positive classes correctly predicted respectively. Inversely, false positives (FP) record the number of negative cases incorrectly classified as positive and false negatives (FN) record the number of positive cases classified as negative.

		$\hat{y}$	
		Negative	Positive
y	Negative	TN	FP
	Positive	FN	TP

Sensitivity (also known as true positive rate) measures the proportion of positive cases correctly predicted.

$$Sensitivity = \frac{TP}{TP+FN} \tag{3.5}$$

Specificity (also known as true negative rate) measures the inverse, with the proportion of negative cases correctly predicted.

$$Specificity = \frac{TN}{TN+FP} \tag{3.6}$$

Area under the receiver operator characteristic curve (AUROC) is another confusion matrix based metric we use we make use of. The statistic requires the formation of a ROC curve, plotting unique false positive rates (FPR) against true positive rates (TPR, or Sensitivity) from each predicted case.

$$FPR = \frac{FP}{FP+TN} \tag{3.7}$$

The ROC curve aims to plot the change in FPR and TPR at different probability thresholds for predicting new cases as positive. A list of the predicted probabilities of each case belonging to the positive class t is formed $\boldsymbol{l} = \{\boldsymbol{p_1(t)}, \ldots, \boldsymbol{p_n(t)}\}$, and sorted in descending order. The FPR and TPR is calculated for each of these probability thresholds, with a case classified as positive only if its probability is less than or equal to the threshold. Duplicate FPR values are removed, with the points having a higher TPR value retained. Two additional points are added to the ROC curve, with (0,0) is inserted at the beginning and (1,1) at the end. This process gives us a ROC curve of length v, $ROC = \{(fpr_1, tpr_1), \ldots, (fpr_v, tpr_v),\}$

The follow is an equation for calculating AUROC using a ROC curve where class label t is the positive class.

$$AUROC_t = \sum_{i=1}^{v-1} fpr_{i+1} \cdot (tpr_{i+1} - tpr_i) \tag{3.8}$$

For multi-class datasets, we use the method suggested by [98], summing the AUROC with each class as the positive class weighted by the train data class

frequency cf.

$$MultiClassAUROC = \sum_{i=1}^{c} cf_i \cdot AUROC_i \tag{3.9}$$

Our final statistic is negative log likelihood (NLL), which we use to determine the reliability of classifier probability estimates. We replace any predicted probabilities of 0 with 0.01 to prevent errors when calculating the statistic.

$$NLL = \frac{-\sum_{i=1}^{n} log_2(\hat{p}_i(y_i))}{n} \tag{3.10}$$

We report accuracy, balanced accuracy, AUROC and NLL throughout this thesis to evaluate the classification algorithms we develop and compare against. While methods are available for reporting sensitivity and specificity in multi-class scenarios, we only report them in the context of nuisance call centre detection, where we can define nuisance callers as our positive case and legitimate callers as a negative case.

3.2 Comparison of Classification Algorithms

In the following, we describe the procedures we take to perform statistically sound experiments. It is possible that unfavourable data distributions between the training and test set or random elements of some classifiers could have a significant impact on experiments. As such, we make sure to resample our datasets multiple times and average the results over each resample for a classifier. When performing significance tests for two classifiers on a dataset, we use a pairwise Wilcoxon signed-rank test on the scores averaged over resamples to determine if one classifier is significantly better than the other ($p < .05$). We note that this has the potential to introduce bias, as the resamples of a single dataset are not independent, and violate the assumptions of the test Many algorithms we run require either tuned hyperparameters or an estimate of accuracy for purposes such as weighting in an

ensemble. We ensure that this process is fully contained within the training process, and only the training data is seen by the classifier. Our evaluations are performed on the test set relating to the train data for its particular resample, unless specified that we are evaluating the ability to estimate accuracy using the training data.

We compare classification algorithms in two different scenarios, comparing two or more classifiers over multiple datasets and comparing two or more classifiers on a single dataset. We describe our approach to both and explain the types of figure we use to broadcast our results.

3.2.1 Over Multiple Datasets

When comparing our classification algorithms over multiple datasets such as the UCR or UEA archives (described in section 3.3), we follow a number of procedures to ensure the integrity of our experiments. To mitigate the impact of data distribution and random elements on our results, we perform 30 random stratified resamples for each dataset. These random resamples are seeded, with all classifiers built on the same training data and evaluated on the same test data. The random seed for both the resamples and classifiers is set to the resample index. If a default train and test split for the dataset is provided, we use it in place of the first resample.

The most common way we present results of multiple classifiers over multiple data sets is by using an adaptation of critical difference diagrams [39], with the change that all classifiers are compared with pairwise Wilcoxon signed-rank tests, and cliques formed using the Holm correction rather than the post-hoc Nemenyi test originally used by [39]. This is done following recommendations from [48] and [13].

An example critical difference diagram is shown in Figure 3.1. Critical difference diagrams plot the average rank of classifiers over multiple datasets, and display whether classifiers are significantly different from each other by placing them into

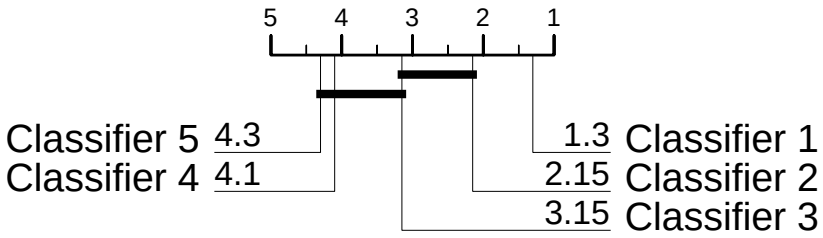


Fig. 3.1 An example critical difference diagram using dummy classifiers and datasets.

cliques. The average rank over all datasets is displayed next to the classifier name, with classifiers being displayed in descending order of rank. The lower the average rank of a classifier, the better it performs using the selected performance metric in comparison to the other classifiers displayed. Classifiers which black bars over them are part of the same clique, and are not significantly different from each other. In our example diagram, Classifier 1 is significantly better than all other classifiers. Classifier 2 has a better rank than Classifier 3, but the difference is not significant. Classifier 3 shows no significant difference to Classifier 2, 4 and 5. It is part of two cliques, however, as while Classifier 4 and 5 are not significantly worse than Classifier 3, they are significantly worse than Classifier 2.

While critical different diagrams are useful for displaying significant differences in rank over multiple datasets, large differences in rank do not necessarily mean large differences in performance. Relatively small differences such as a single extra case being classified correctly out of potentially thousands will result in a higher rank despite being a small difference. This issue is most likely to appear when comparing similar classifiers or the same classifier parameterised differently. We accompany our critical difference diagrams with tables showing the averaged results for each of our chosen metrics in an appendix for relevant sections. For classifiers of interest, we display the difference in our metrics for all datasets using pairwise scatter diagrams. An example pairwise scatter diagram is shown in Figure 3.2. In this example, we plot the accuracy for 10 datasets in a scatter diagram, with the

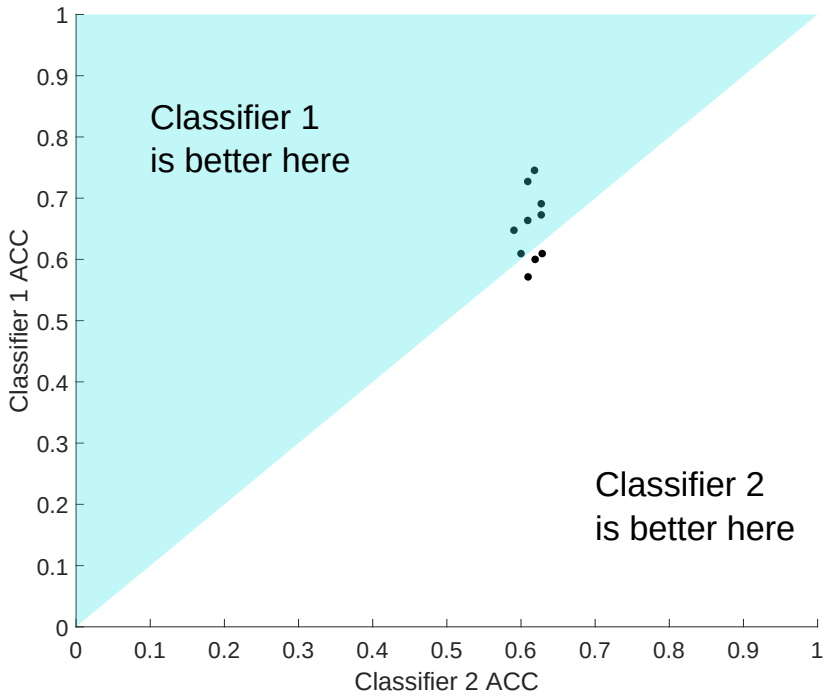


Fig. 3.2 An example pairwise scatter diagram using dummy classifiers and datasets.

first classifiers accuracies plotted on the y-axis and the second classifiers accuracies on the x-axis. From this diagram, we can see that Classifier 1 is more accurate on the majority of datasets, with the classifier being over 10% more accurate on some datasets.

3.2.2 Over a Single Dataset

When comparing classifiers using singular datasets, such as our nuisance call centre data or case studies in our algorithmic improvement chapters, a slightly different approach is taken. We run a 10-fold cross validation instead of the random resamples used for multiple datasets. When looking at a singular dataset, we are more interested in looking at individual cases and predictions. Cross validation guarantees that each case will have a single prediction made using a large portion of the data for training. These can then be collated into a single set of results. For

single problems, we report our results primarily using tables displaying relevant performance metrics.

3.3 Datasets

Our experiments make use of numerous public datasets, contained with two separate time series data archives. The University of California, Riverside (UCR) time series archive [36] is the most well known dataset archive for time series classification. The archive is constantly expanding, containing over 100 univariate datasets presently. The University of East Anglia (UEA) archive [3] is the multivariate counterpart to the UCR archive. Both archives are used extensively throughout the thesis, as such we provide information on both in the following sections.

Datasets from both UCR and UEA archives as well as our case study data excluding those on the topic of nuisance calls are available to download from the time series classification dataset webpage[1].

3.3.1 UCR Univariate Time Series Archive

For our algorithmic advances in univariate time series classification, we run our experiments on 112 datasets from the expanded UCR archive [36]. Previous works using the archive would experiment on a subset of the 47 or 85 datasets present at the time [8, 17, 105]. Since then, the amount of datasets in the archive has increased to 128. Of these new datasets, 15 include either missing values or the time series within are of unequal length. These cases are not the focus of the experimentation we perform, and how to properly deal with them is a research question of its own. As such, we exclude them to stop these factors from having an impact on our results. We additionally remove the Fungi data, which only provides a single train case for each class label. Being unable to properly process data with single case classes is a

[1] http://www.timeseriesclassification.com/dataset.php

Table 3.1 Summary statistics for the 112 univariate UCR datasets we use in our algorithmic advancements. Provides statistics for the train set sizes (a), test set sizes (b), number of classes (c) and series lengths (d) of the archive as well as counts for each dataset type available (e).

1-200	58 (51.79%)
201-500	32 (28.57%)
500+	22 (19.64%)
Minimum	16
Maximum	8926

(a) Train Size

1-200	35 (31.25%)
201-500	36 (32.14%)
500+	41 (36.61%)
Minimum	20
Maximum	16800

(b) Test Size

2	57 (50.89%)
3-8	33 (29.46%)
9+	22 (19.64%)
Minimum	2
Maximum	60

(c) No. Classes

1-200	40 (35.71%)
201-500	31 (27.68%)
500+	41 (36.6%)
Minimum	15
Maximum	2844

(d) Series Length

Dataset Type	No. Datasets
Device	9
Biosignal	13
Image	32
Motion	17
Sensor	21
Simulated	8
Spectrum	12

(e) Dataset Types

general limitation for approaches which rely on cross-validation or train accuracy estimates. We deemed the case unrealistic enough to remove it from our evaluation. The UCR archive provides a default split into train and test sets.

Tables 3.1 provides summary information for characteristics of interest of the UCR archive. The archive contains datasets from a diverse set of domains, and has a large variety of train sizes and series lengths to evaluate algorithms with. Table A.1 in Appendix A summarises each of these 112 datasets individually.

3.3.2 UEA Multivariate Time Series Archive

For our multivariate time series classification experiments, we use all 26 equal length datasets of the 30 total from the UEA multivariate time series archive [3]. The UEA archive is relatively new compared to the UCR archive, but has seen use in a large comparison of multivariate TSC algorithms [104].

Tables 3.2 provides summary information for characteristics of interest of the UEA archive. While not as large as the UCR archive, the UEA archive is the largest available set of multivariate data available. A potential issue to note is the large amount of motion and biosignal datasets which make up the majority of the archive. This issue may lead to a set of algorithms which favour these types of problems to the detriment of other data domains, but this is not an issue to be tackled in this thesis and multivariate data is not the primary focus of our nuisance call research and algorithmic developments. Table A.2 in Appendix A summarises each of these 26 datasets individually.

3.3.3 Case Study Datasets

For each of our chapters covering algorithmic advancements for TSC, we include experiments on a case study dataset or problem alongside our archive experiments. These datasets are selected because of these differences to our archive datasets, primarily in the size of the dataset or its contents containing unequal length series. We do not describe the call centre data provided by BT in this section, with these datasets being introduced in dedicated chapters for the problem of nuisance calling in Chapters 7 and 8. In our algorithmic sections we are more interested in the impact of the data characteristics rather than the problem details, but provide more information on each case study here for interested readers. The datasets included here are publicly available and can be used in publications, unlike the call data provided by BT. In this section, we introduce each case study problem.

Table 3.2 Summary statistics for the 26 multivariate UEA datasets we use in our algorithmic advancements. Provides statistics for the train set sizes (a), test set sizes (b), number of classes (c), series lengths (d) and number of dimensions (e) of the archive as well as the counts for each dataset type available (f).

1-200	15 (57.69%)
201-500	7 (26.92%)
500+	4 (15.38%)
Minimum	12
Maximum	7494

(a) Train Size

1-200	15 (57.69%)
201-500	6 (23.08%)
500+	5 (19.23%)
Minimum	15
Maximum	3524

(b) Test Size

2	8 (30.77%)
3-8	11 (42.31%)
9+	7 (26.92%)
Minimum	2
Maximum	39

(c) No. Classes

1-200	12 (46.15%)
201-500	6 (23.08%)
500+	8 (30.77%)
Minimum	8
Maximum	17984

(d) Series Length

2-6	15 (57.69%)
7-50	6 (23.08%)
51+	5 (19.23%)
Minimum	2
Maximum	1345

(e) No. Dimensions

Dataset Type	No. Datasets
Biosignal	8
Audio	3
Motion	12
Other	3

(f) Dataset Types

Right Whales

Chapter 4 contains a case study on the detection of Atlantic right whales. Recently, the protection of endangered whales has been a prominent global issue. Being able to accurately detect marine mammals is important to monitor populations and provide appropriate safeguarding for their conservation. North Atlantic right whales are one of the most endangered marine mammals, with as few as 350 individual remaining in the wild [64]. We use a dataset from the Marinexplore and Cornell University Whale Detection Challenge[2] that features a set of right whale up-calls. Up-calls are the most commonly documented right whale vocalisation with an acoustic signature of approximately 60Hz-250Hz, typically lasting 1 second. Right

[2]https://www.kaggle.com/c/whale-detection-challenge/data

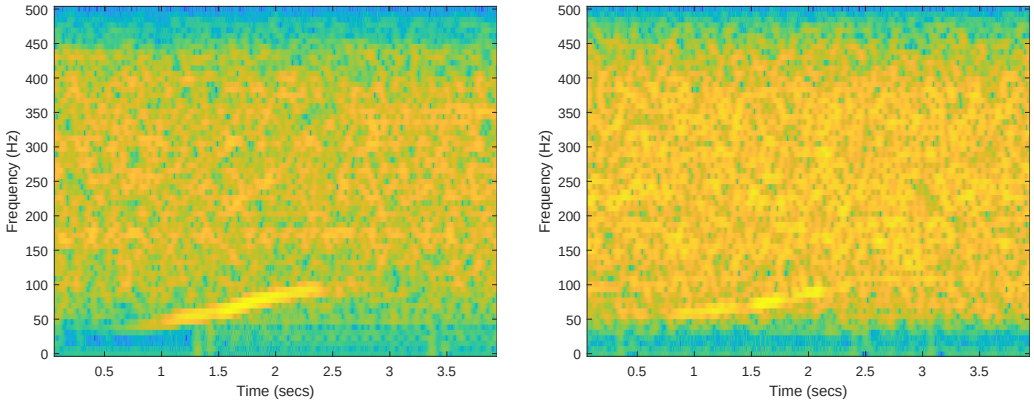


Fig. 3.3 Atlantic Right Whale audio spectrograms containing whale up-calls, the right spectrogram contains a large amount of noise.

whale calls can often be difficult to hear as the low frequency band can become congested with anthropogenic sounds such as ship noise, drilling, piling, or naval operations [34]. The dataset contains audio sampled at 2kHz with a total length for each series of 4000 samples, with 10,934 train cases and 1962 test cases. These audio clips contain any mixture of right whale calls, non-biological noise, or other whale calls. Each series is labelled as either containing a right whale or not, with the aim to correctly identify the series that contain up-calls. The problem is relatively large, exceeding the largest train set from ElectricDevices of 8926 and series length from Rock of 2844 in the 128 UCR archive. An example spectrogram of the whale call audio is displayed in Figure 3.3.

The large size of this dataset compared to the UCR archive datasets is why we selected it as a case study dataset. One of the main goals of Chapter 4 is to improve the scalability of the BOSS [105] algorithm, which failed to complete this dataset within our standard 7-day limit for running experiments. Being able to show results for dictionary approaches on the data was a driver for development.

Asphalt

The case study in Chapter 5 covers the three Asphalt datasets first presented in [119]. The problem involves predicting the condition of a road based on motion data recorded smart phone installed inside a vehicle using a flexible suction holder near the dashboard. This offers the potential for automated monitoring and assessment of road conditions, leading to earlier and less costly interventions with faults. The Android application *Asfault* [120] was used to collect accelerometer data in the form of the three physical axes, latitude, longitude, and velocity from GPS (A_x, A_y, A_z). These axes are converted into a univariate time series that represents the acceleration magnitude (A_m). This magnitude forms the time series used in each problem. A sampling rate of 100 Hz was used for each time series. These datasets cover three separate problems related to road surface conditions. Asphalt-Regularity looks at road deterioration using driver comfort as a metric. A road is classified as regular or deteriorated. The Asphalt-PavementType problem is to classify the surface type of the road as either dirt, cobblestone or asphalt. The Asphalt-Obstacles problem is to classify whether the vehicle is crossing one of a set of common road obstacles: speed bumps; vertical patches; raised pavement markers; and raised cross walks. Experts were responsible for labelling the data, using the application, and driving the vehicle. Table 3.3 presents basic summary stats for the datasets. The series are unequal length, centred around zero, but not normalised. This unequal length characteristic is what differentiated the datasets from the 112 UCR datasets, and the performance of interval based algorithms on this kind of data is what we wanted to measure. Figure 3.4 plots the default train series for the Asphalt-PavementType problem, separating each class value.

Table 3.3 Summary information for the Asphalt data used in [119].

	Asphalt-Regularity	Asphalt-PavementType	Asphalt-Obstacles
Train Size	751	1055	390
Test Size	751	1056	391
Min Series Length	66	66	111
Max Series Length	4201	2371	736
No. Classes	2	3	4

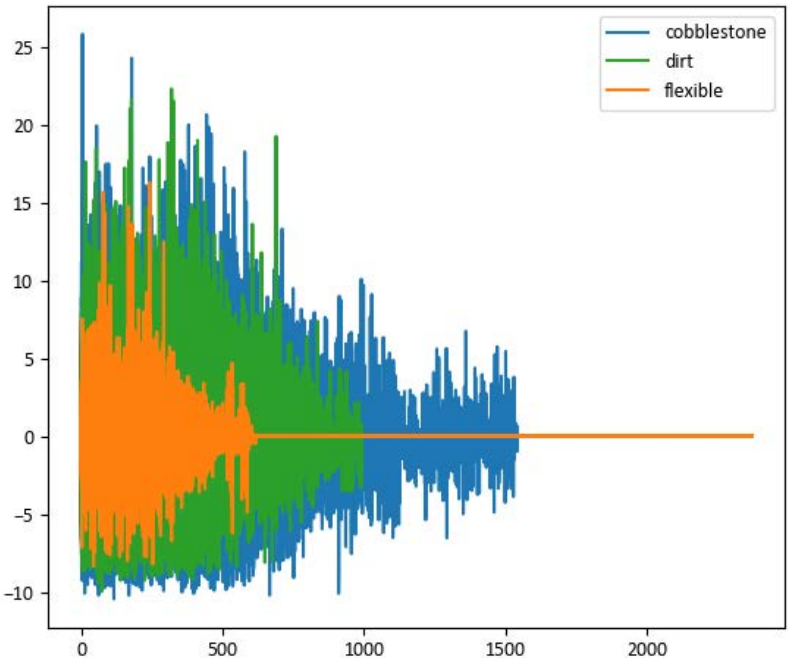


Fig. 3.4 Training time series for the Asphalt-PavementType problem.

Insect Audio

In Chapter 6 we look at two insect audio datasets. The first of these datasets is FruitFlies, the aim of which is to classify an insect as one of three species of fruit fly, based on the reconstructed audio of its wingbeat sampled at 8,000Hz. The time series represent the change in amplitude of an infra-red light as it is occluded by the wings of a fruit fly during flight. The series are recorded from an optical sensor inserted in-turn into six insectary boxes containing only one fly species of both sexes. As the insects fly through the sensor, their wings partially occludes the light from the transmitter to the receiver. The light fluctuation recorded follows

Table 3.4 Table showing the attributes of the large insect audio datasets.

	InsectSound	FruitFlies
Train Size	25000	17255
Test Size	25000	17256
No. Classes	10	3
Series Length	15,883	5000
Total size (Gb)	14.75	2.44

the rhythm of the wingbeat. The second is the InsectSound dataset. The cases of this dataset are also amplitude modulated intervals produced by the occlusion of an infrared beam set within an enclosure housing a single species. A larger variety of insects are used for this dataset, with some split by the sex of the insect, which can be relevant for species of mosquitos. The dataset consists of 10ms segments sampled at 6,000Hz. Table 3.4 provides the basic summary stats for both datasets. The large size of these datasets makes them useful use cases for the usability improvements and contracting capability introduced in Chapter 6 for our improved HIVE-COTE.

3.4 Software and Research Reproducibility

Accessibility and reproducibility of research has become increasingly important in recent times. Following this trend, we make the extensive results of our algorithmic advancements in Chapters 4 through 6 available publicly. We provide webpages containing links to these results in their relevant chapters. These results are obtained from publicly available and widely used datasets, allowing other researchers to easily compare against them. We are unfortunately unable to provide data and in-depth results files for our work on nuisance call detection.

There have been recent advances in the open source software available for TSC since the bakeoff comparison [8]. When used in conjunction with public data repositories, open source implementations of classifiers facilitate reproducible

research and easy comparison to current classifiers. Providing implementations of algorithms in a common framework can greatly increase the impact and reach of research. There are two primary open source toolkits for this purpose: the Python toolkit `sktime`[3] and `tsml`[4] in Java. Other packages include the relatively young `tempo`[5] in C++, an alternate and `sktime` interoperable Python package *tslearn*[6], and repositories of time statistics such as the tsfresh[7] and hctsa[8] toolkits. The toolkits we use primarily throughout this thesis are *tsml* and *sktime*. We also include implementations of all our Chapter 4 through 6 TSC algorithmic advances in both of these packages.

During our experimentation, we use the algorithms implemented in *tsml* unless specified otherwise. In Chapter 8 we use implementations from *sktime*, due to lack of a *tsml* implementation for one algorithm and to better interface with our collaborator's framework. Deep learning algorithms such as InceptionTime [43] we run using the *sktime* extension package, *sktime-dl*[9]. For feature vector base classification algorithms, we use the Java based *Weka* toolkit [45].

All experiments were run single threaded on a high performance computing (HPC) cluster with a run time limit of 7 days unless specified otherwise. `sktime-dl` experiments were performed on desktops GPUs. Each job is run on a single GPU, with each GPU running only one job at a time. There is no time limit for running these jobs. However, they are limited by the GPU memory of 12 GB per card.

With the caveat that both *tsml* and *sktime* are under active development, we briefly review the classification usage and functionality of these toolkits at the time of writing.

[3]https://github.com/alan-turing-institute/sktime
[4]https://github.com/uea-machine-learning/tsml
[5]https://github.com/MonashTS/tempo
[6]https://github.com/tslearn-team/tslearn
[7]https://github.com/blue-yonder/tsfresh
[8]https://github.com/benfulcher/hctsa
[9]https://github.com/sktime/sktime-dl

3.4.1 Time Series Classification in Python: `sktime`

`sktime` is an open source, Python based, scikit-learn compatible toolkit for time series analysis. `sktime` is designed to provide a unifying API for a range of time series tasks such as annotation, prediction and forecasting. For a description of the overarching design of `sktime` see [79], and for an experimenal comparison of some of the classification algorithms available see [7]. For classification, `sktime` reads data from text files in the Weka ARFF format and a bespoke time series format (.ts). Whilst suitable for equal length series, ARFF is designed for tabular data and not time series, and hence cannot store unequal length series. Because of this, `sktime` requires its own bespoke format. There are converters and examples of both formats provided by in the package. The classification module[10] in `sktime` groups classifiers using the TSC taxonomy described in chapter 2.

Standard classifier usage is provided in Jupyter Notebook format in the examples package[11]. An experiment to generate a results file for the Time Series Forest (TSF) algorithm on the Chinatown problem using `sktime` is given below.

```
from sktime.benchmarking.experiments import
    load_and_run_classification_experiment
from sktime.classification.interval_based import
    TimeSeriesForestClassifier

load_and_run_classification_experiment(
    # Where to get the data
    problem_path="../Data/",
    # Where to write the results
    results_path="../Results/",
    # Classifier to use in the experiment
    classifier=TimeSeriesForestClassifier(),
    # Problem name
    dataset="Chinatown",
    # Resample number/seed: 0 gives the default train/test
    split
    resampleID=0,
```

[10] https://www.sktime.org/en/latest/api_reference/classification.html#module-sktime.classification

[11] https://github.com/alan-turing-institute/sktime/tree/main/examples

```
)
```

Classifiers can also be built using standard scikit-learn methods, an example is provided in the listing below. Some of the classifiers in `sktime` can handle multivariate and/or unequal length series. Capabilities for each classifier are defined through a system of tags.

```
from sktime.utils.data_io import load_arrowhead
from sktime.classification.interval_based import
    TimeSeriesForestClassifier

# Load dataset
trainX, trainY = load_arrowhead(split="train")
testX, testY = load_arrowhead(split="test")

# Create classifier and build on training data
tsf = TimeSeriesForestClassifier(n_estimators=50)
tsf.fit(trainX, trainY)

# Find accuracy on testing data
accuracy = tsf.score(testX, testY)
```

3.4.2 Time Series Classification in Java: `tsml`

The time series machine learning (`tsml`) library is a Java based, Weka compatible toolkit for time series classification and clustering. Like `sktime`, it can input either ARFF or TS files. An experiment to generate a results file in the same format as `sktime` is given below.

```
import experiments.Experiments;

static void tscExperiment() throws Exception {
    String[] settings=new String[5];
    // Where to get the data
    settings[0]="-dp=../Data/";
    // Where to write the results
    settings[1]="-rp=../Results/";
    // Classifier name: See ClassifierLists for valid options
    settings[2]="-cn=TSF";
    // Problem name
```

```
    settings[3]="-dn=Chinatown";
    // Resample number/seed: 1 gives the default train/test
   split
    settings[4]="-f=1";
    Experiments.ExperimentalArguments expSettings =
        new Experiments.ExperimentalArguments(settings);
    Experiments.setupAndRunExperiment(expSettings);
}
```

`tsml` also has functionality to compare classifier results files generated through either `sktime` or `tsml`. This can output files and figures used to compare classifiers using a variety of metrics.

```
import evaluation.MultipleClassifierEvaluation;

static void classifierEvaluation() throws Exception {
    MultipleClassifierEvaluation x = new
   MultipleClassifierEvaluation(
        // Evaluation write path
        "../ResultsEvaluation/",
        // Evaluation identifier
        "Evaluation",
        // Number of results folds
        30
    );
    // A list, array or path to a txt file of datasets
    x.setDatasets(datasetList);
    x.readInClassifiers(
        // Classifier results to be read in
        new String[]{"TSF", "BOSS", "ROCKET"},
        // Path to the classifier results
        "../Results/"
    );
    x.setUseAllStatistics();
    x.setBuildMatlabDiagrams(true);
    x.setDebugPrinting(true);
    x.setTestResultsOnly(true);
    x.runComparison();
}
```

`tsml` classifiers can be used in the same way as standard Weka classifiers.

```
import experiments.data.DatasetLoading;
import tsml.classifiers.interval_based.TSF;
```

```
import utilities.ClassifierTools;

static void runClassifier() throws Exception {
    // Load dataset
    Instances train = DatasetLoading.loadData("../Data/
   data_TRAIN.arff");
    Instances test = DatasetLoading.loadData("../Data/
   data_TEST.arff");

    // Create classifier and build on training data
    TSF tsf = new TSF();
    tsf.setNumTrees(500);
    tsf.buildClassifier(train);

    // Find accuracy on testing data
    double accuracy = ClassifierTools.accuracy(test, tsf);
}
```

Chapter 4

Dictionary Based Advances: The Temporal Dictionary Ensemble

Contributing Publications

- Middlehurst, M., Vickers, W., & Bagnall, A. (2019). Scalable dictionary classifiers for time series classification. *In International Conference on Intelligent Data Engineering and Automated Learning* (pp. 11-19). Springer, Cham.
- Middlehurst, M., Large, J., Cawley, G., & Bagnall, A. (2020). The Temporal Dictionary Ensemble (TDE) Classifier for Time Series Classification. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases* (pp. 660-676). Springer, Cham.

4.1 Introduction

In Chapter 1 we discuss the need to improve the scalability of the Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) [78] to make its usage feasible for the large data sizes expected from the BT nuisance calling

data. To achieve this, the constituent classifiers which make up the meta ensemble would have to be made more efficient. The three problem classifiers when it comes to efficiency are the Shapelet Transform (ST-HESCA) [58], the Elastic Ensemble (EE) [76] and the Bag of Symbolic Fourier Approximation Symbols (BOSS) [105].

The first component we looked at making more scalable was the dictionary based BOSS. From the 2017 bake off comparison of TSC algorithms, BOSS was found to be significantly more accurate than other dictionary classifiers, and the third most accurate algorithm overall [8]. This performance lead to BOSS being incorporated into HIVE-COTE, significantly improving its accuracy, and placing it in the state-of-the-art for TSC at the time. We cover BOSS more thoroughly in Chapter 2, but to summarise, the algorithm is an ensemble of one nearest neighbour classifiers built on data from the Symbolic Fourier Approximation (SFA) [107] transform. The parameters for the SFA transform are set using a grid search, with the most accurate classifiers included in the BOSS ensemble. BOSS and other dictionary base classifiers excel in cases where the difference between classes is defined by the frequency of certain patterns.

The BOSS ensemble has some drawbacks. Firstly, the need to cross-validate each parameter combination means that it scales poorly in terms of train time. This parameter range also scales with the series length, containing $10 \cdot m/4$ unique parameter combinations. Secondly, the fact that it retains a variable number of base classifiers means that it is often very memory intensive. The histograms can be very large as well, so storing them all for each base classifier requires a significant memory commitment for large problems. One proposed method to solve this, the BOSS Vector Space (BOSS-VS) classifier [106], has been shown to be significantly less accurate than the full BOSS [108, 81]. Our initial hypothesis is that random parameter selection can produce a significantly faster algorithm without any significant loss in accuracy. This results in our first contribution to improving the dictionary based TSC representation, Contractable BOSS (cBOSS) [88], which

uses an alternative ensemble mechanism to provide an order of magnitude speed up while retaining accuracy. Our changes also allow for the easy implementation of contracting, i.e. setting a build time limit for the classifier instead of an ensemble size.

Since the bake off, a number of dictionary algorithms other than cBOSS have been proposed. These algorithms are mostly extensions of BOSS, making alterations and additions to different parts of the original algorithm. While we focus on improving train time and memory efficiency with cBOSS, these algorithms improve accuracy [108, 67] and prediction time efficiency [108]. Word Extraction for Time Series Classification (WEASEL) [108] abandons the ensemble structure in favour of feature selection and changes the method of word discretisation. Spatial BOSS (S-BOSS) [67] introduces temporal information and additional features using spatial pyramids. We detail these algorithms and their changes in Chapter 2.

Both of these algorithms constitutes an improvement to the dictionary representation from BOSS. Our main contribution in this chapter is to combine the design features of these four classifiers (BOSS, WEASEL, S-BOSS and cBOSS) to make a new algorithm, the Temporal Dictionary Ensemble (TDE) [86]. Like BOSS, TDE is a homogeneous ensemble of nearest neighbour classifiers that use distance between histograms of SFA word counts and injects diversity through parameter variation. TDE takes the ensemble structure from cBOSS, which is more robust and scalable. The use of spatial pyramids is adapted from S-BOSS. From WEASEL, TDE uses bigrams and an alternative method of finding word breakpoints.

We found the simplest way of combining these components did not result in significant improvement. We speculate that the massive increase in the parameter space made the randomised diversity mechanism result in too many poor learners in the ensemble. We propose a novel mechanism of base classifier model selection based on an adaptive form of Gaussian Process (GP) modelling of the parameter

space. We show that TDE is significantly more accurate than WEASEL and S-BOSS while retaining the usability and scalability of cBOSS.

Table 4.1 summarises the different features extracted and design choices made for each of the dictionary based algorithms we describe and introduce in this chapter.

The rest of this chapter is structured as follows. Section 4.2 describes our initial changes to the BOSS algorithm to increase its scalability with cBOSS. Section 4.3 describes the TDE algorithm. Section 4.4 presents the performance evaluation of cBOSS and TDE against other dictionary based algorithms and the best classifiers of other domains. The impact of individual TDE components and decisions made during the algorithm's development are analysed in Section 4.5. Conclusions are drawn in Section 4.6.

Table 4.1 Summary table of the dictionary based classifiers covered in this chapter and the design choices and features included in each.

	BOSS	S-BOSS	WEASEL	cBOSS	TDE
SFA bag-of-words	X	X	X	X	X
Ensemble	X	X		X	X
Feature Selection			X		
Grid Parameter Selection	X	X	X		
Random Parameter Selection				X	X
GP Parameter Selection					X
Multiple Coefficient Binning	X	X		X	X
Information Gain Binning			X		X
ANOVA F-test			X		
Spatial Pyramids		X			X
Bigrams			X		X
Histogram Intersection		X			X
Multivariate Capable			X		X

4.2 A Contractable BOSS [88]

Our initial changes to BOSS focus on the ensemble technique of the classifier, which is computationally expensive and unpredictable. Ensembling has been shown to be an essential component of BOSS, resulting in significantly higher accuracy [67].

We assess whether we can replace the current ensemble mechanism with a more stable and efficient scheme without a significant reduction in accuracy. One of the easiest ways to speed up BOSS would be to eliminate the requirement to estimate the accuracy of each individual classifier. However, we show that complete randomisation, i.e. selecting random parameter combinations for a fixed number of base classifiers works reasonably well, but on some data performs very badly.

As full randomisation without accuracy loss is not feasible, we retain an internal evaluation of each possible member through leave-one-out cross-validation (LOOCV) on the train data, then use this value to both select and weight classifier votes. The primary difference to BOSS is that we do not determine which members to retain after a complete search. Rather, we introduce a new parameter, k, the number of parameter sets to randomly sample, and maintain a list of weights. The option to replace the number of parameter sets evaluated k with a time limit t through contracting is made available through this change in building process, when set the classifier will continue building until the time since building started is greater than t.

Even with weighting, classifiers with poor parameters for a particular problem can degrade performance of the ensemble. Because of this, we set a max ensemble size s to filter these out, with any classifier built after the ensemble has reached size s replacing the current lowest accuracy member of the ensemble if its accuracy is higher. To further diversify the ensemble and increase efficiency, we take a randomly selected 70% subsample of the training data for each individual classifier. The parameter space we randomly sample for cBOSS is the same as that which BOSS exhaustively searches. For classifying new instances, we adopt the exponential weighting scheme used in the Cross-validation Accuracy Weighted Probabilistic Ensemble (CAWPE) [68] to amplify small difference in weights and results in significantly improved performance. cBOSS is more formally described in Algorithm 4.1.

Algorithm 4.1 cBOSS(A list of n cases of length m, $\boldsymbol{T} = (\boldsymbol{X}, \boldsymbol{y})$)

Parameters: the number of parameter samples k, the max ensemble size s

Let w be window length, l be word length, p be normalise/not normalise and α be alphabet size.
Let $\boldsymbol{c}$ be a list of s BOSS classifiers $(c_1, \ldots, c_s)$
Let $\boldsymbol{e}$ be a list of s classifier weights $(e_1, \ldots, e_s)$
Let $\boldsymbol{R}$ be a set of possible BOSS parameter combinations
$i \leftarrow 0$
$lowestAcc \leftarrow \infty, lowestAccIdx \leftarrow \infty$
while $i < k$ AND $|\boldsymbol{R}| > 0$ **do**
  $[l, a, w, p] \leftarrow$ randomSample($\boldsymbol{R}$)
  $\boldsymbol{R} = \boldsymbol{R} \setminus \{[l, a, w, p]\}$
  $\boldsymbol{T'} \leftarrow$ subsampleData($\boldsymbol{T}$, 0.7)
  $cls \leftarrow$ baseBOSS($\boldsymbol{T'}, l, a, w, p$)
  $acc \leftarrow$ LOOCV(cls) { *train data accuracy*}
  if $i < s$ **then**
    if $acc < lowestAcc$ **then**
      $lowestAcc \leftarrow acc$, $lowestAccIdx \leftarrow i$
    $c_i \leftarrow cls$, $e_i \leftarrow acc^4$
  else if $acc > lowestAcc$ **then**
    $c_{lowestAccIdx} \leftarrow cls$, $e_{lowestAccIdx} \leftarrow acc^4$
    $[lowestAcc, lowestAccIdx] \leftarrow$ findNewLowestAcc($\boldsymbol{c}$)
  $i \leftarrow i + 1$

4.3 The Temporal Dictionary Ensemble [86]

In our comparison of dictionary based algorithms after the development of cBOSS [7], we found that each BOSS extension improved the algorithm in different but compatible ways. We experiment with including the advancements made in S-BOSS and WEASEL in cBOSS. The easiest algorithms to combine are cBOSS and S-BOSS. cBOSS speeds up BOSS through subsampling training cases and random parameter selection. S-BOSS improves the algorithm's accuracy by introducing spatial information through applying a spatial pyramid to the bag of words. The number of pyramid levels parameter introduced by S-BOSS can be included in the random parameter selection used by cBOSS. For comparisons, we call this naive hybrid of algorithms cS-BOSS. We use this as a baseline to justify the extra complexity we introduce in TDE.

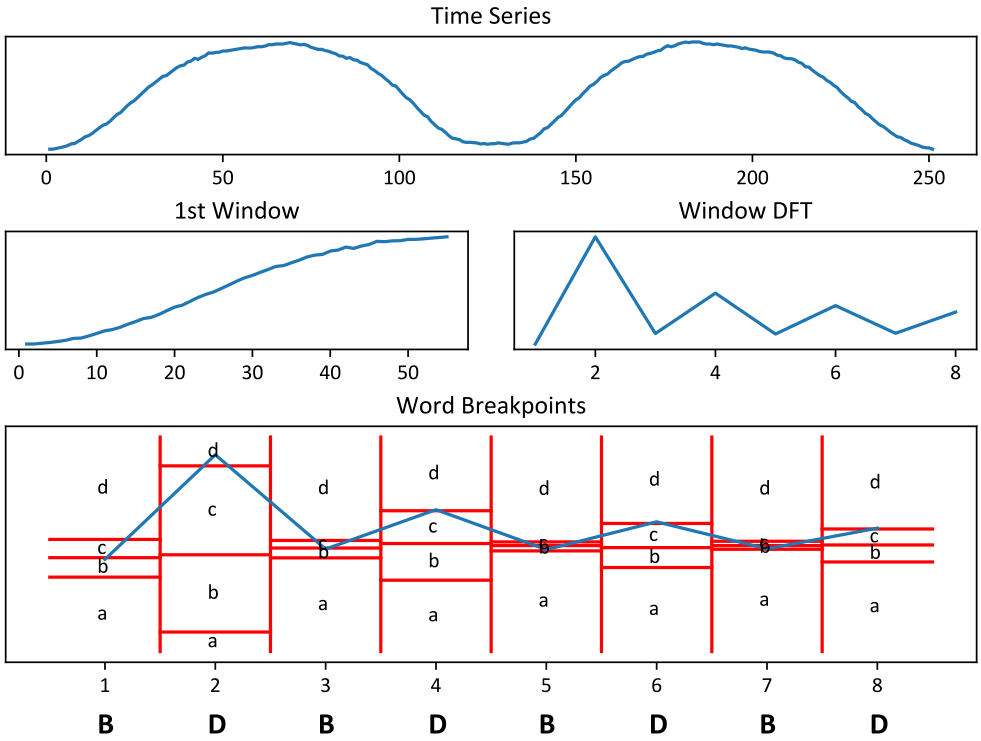


Fig. 4.1 Example of the SFA transform on a case from the ArrowHead UCR archive dataset. A window is extracted, then discretised into the length 8 word BDBDBDBD using a set of learned breakpoints.

TDE is an ensemble of one nearest neighbour classifiers that transform each series into a histogram of word counts. A sliding window of length w is run along each series, and the subseries are discretised into a word of length l from an alphabet of size α. TDE discretises the window into a word using the SFA transform, binning the first l Discrete Fourier Transform (DFT) values from the window using a set of breakpoints. This process is summarised in Figure 4.1, and is so far the same as BOSS. Instead of using the bespoke distance of BOSS, TDE finds the distance between histograms is found using histogram intersection. Histogram intersection simply compares each bin for two histograms and sums the minimum value for each as a measure of similarity. In addition to word frequencies, TDE also captures the frequencies of bigrams found from non overlapping windows, a feature used in WEASEL. Thus, a transformed case includes a histogram of word counts and bigram counts for a given trio of parameters (w,l,α). TDE also includes

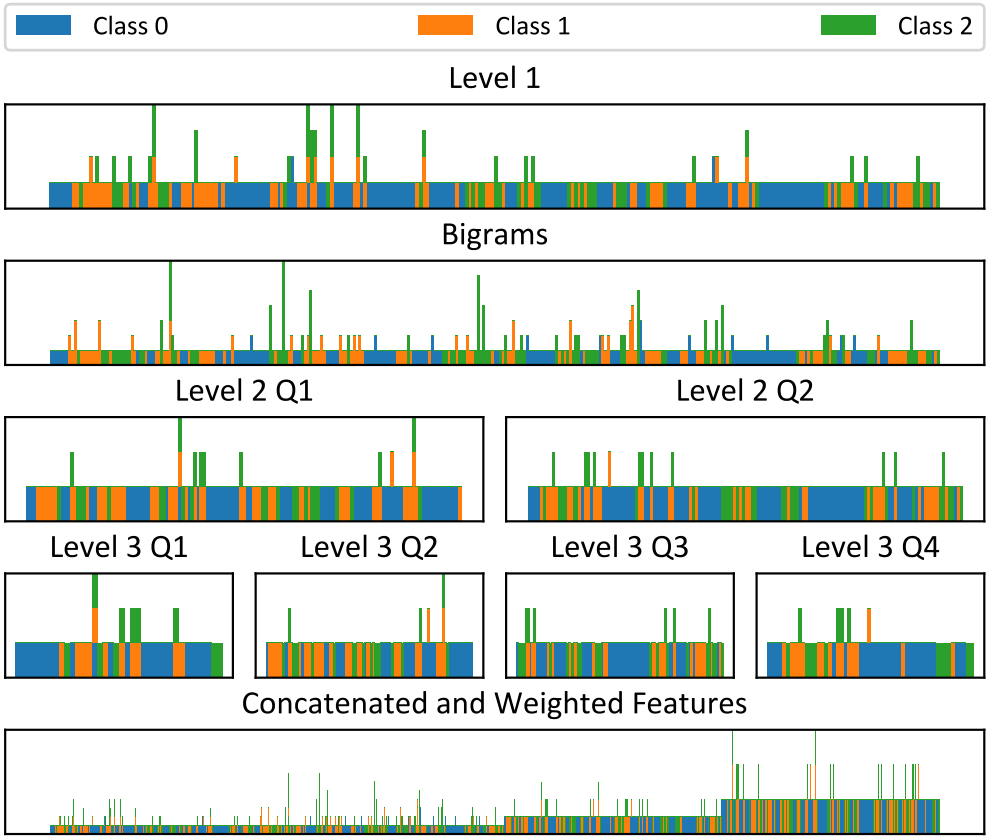


Fig. 4.2 Example bag of words with 3 spatial pyramid levels from the ArrowHead UCR archive dataset, created by merging the histograms of all train cases. Histograms for individual cases would have a much smaller variety of words and belong to a single class.

the spatial information by the utilisation of spatial pyramids [69] from S-BOSS. This involves splitting a series into h levels each with 2^v disjoint subseries, where v is the current pyramid level. Word counts are found for each spatial subseries independently, then the resulting histograms are concatenated. Like S-BOSS, the distance to histograms of deeper levels with smaller spatial areas in the series are weighted higher than global similarity. Bigrams are only recorded for the first level, consisting of the full series. An example of this process from the ArrowHead dataset is shown in Figure 4.2. The SFA transform requires a set of breakpoints when creating words. The method of generating these breakpoints b is selected between Multiple Coefficient Binning (MCB) [105] and Information Gain Binning

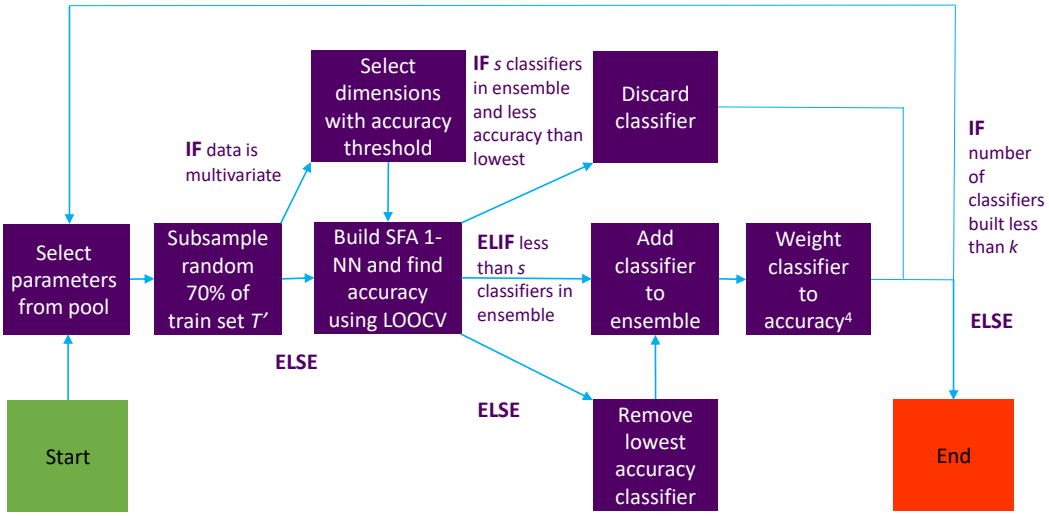


Fig. 4.3 The TDE ensemble structure inherited from cBOSS with multivariate capabilities. Handling of parameter selection in the first step of the build process is displayed in figure 4.4.

(IGB) [108]. Windows can optionally be normalised during the transform with the p parameter.

Like cBOSS, the TDE ensemble is filtered into s total classifiers from k candidates. The accuracy of each candidate is estimated using LOOCV, with the highest s being retained. At first thought, using the leftover data from the cBOSS subsampling would seem a better choice for generating a train accuracy estimate. However, this data would first have to be transformed before it can be used to make an estimation, while the subsampled data is already transformed. Diversity is achieved through altering the parameters (w,l,h,b,p) for each new classifier and a 70% sampling of the train data. New cases are classified with a weighted majority vote, using the exponential accuracy weights from CAWPE [68]. An overview of the ensemble structure used by TDE is provided in Figure 4.3.

Table 4.2 shows the range from which individual classifier parameters are subsampled, with m being the series length. To properly make full use of this expanded range, an increase in the amount of parameter sets considered is required. However, this comes with an accompanying increase in train time, as the building and evalua-

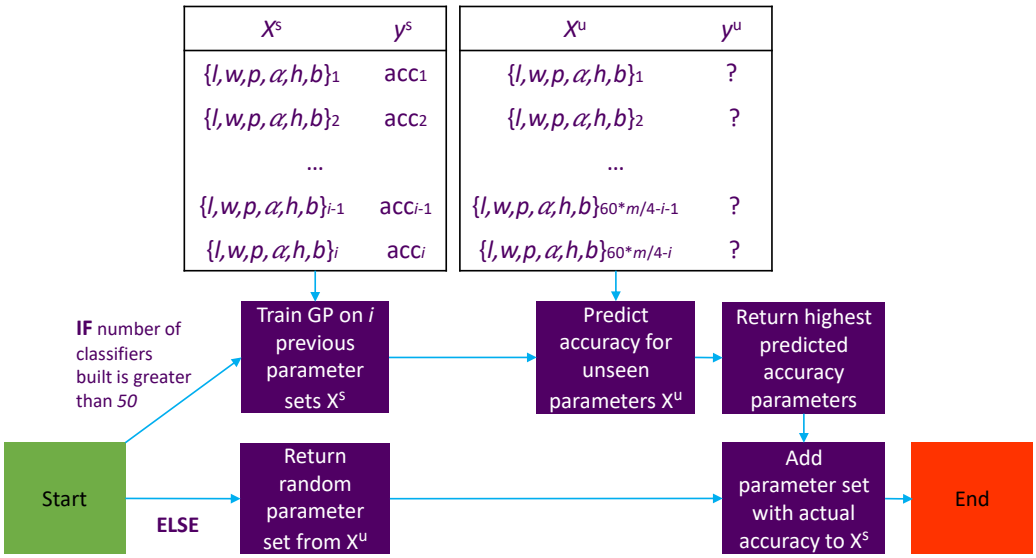


Fig. 4.4 The TDE parameter selection structure, using a Gaussian Process model to predict the best parameter sets to builds.

tion of base ensemble members is expensive. Instead, TDE uses a guided parameter selection for ensemble members inspired by Bayesian optimisation [118]. A GP model is built over the regressor parameter space $\mathbf{R}$ for parameters $[l, a, w, p, h, b]$ ($\boldsymbol{x}$) to predict the accuracy (y), using n previously observed ($\boldsymbol{x}$,y) pairs $\mathbf{G}$ from previous classifiers. We use a basic form of GP and treat all the regressors (TDE parameters) as continuous. The first 50 classifiers use randomly selected parameters to ensure a suitable sample for the guided selection, while those after are selected using the GP regressor. For unseen parameter sets, a prediction of accuracy is made using the parameters of all previously built classifiers, with the highest predicted accuracy being chosen for the next classifier build. Figure 4.4 visualises the parameter selection method used by TDE. The full ensemble build process for TDE is described in Algorithm 4.2. The bestPredictedParameters operation in line 13 is limited to the same parameter ranges used for random search given in Table 4.2.

We introduce the capability for multivariate time series classification to TDE by making a number of additions to the individual classifier. WEASEL also has a multivariate version, the Multivariate Unsupervised Symbols and Derivatives

Table 4.2 Parameter ranges for TDE base classifier selection.

Parameter	Range
Word lengths	$l = \{16, 14, 12, 10, 8\}$
Window lengths	$w = \{10...m\}$
Normalise	$p = \{true, false\}$
Alphabet size	$\alpha = \{4\}$
No.pyramid levels	$h = \{1, 2, 3\}$
Breakpoints	$b = \{MCB, IGB\}$

(WEASEL+MUSE, or just MUSE) [109]. Current dictionary approaches are noticeably memory intensive due to the requirement to store multiple transformed versions of the original data. We aim to mitigate this issue with the TDE multivariate extension.

For each dimension, we extract words using the same process for univariate series, with a dimension having its own set of breakpoints. Words from different dimensions are stored separately in each case's bag. For many multivariate time series, some dimensions hold little or redundant information. Additionally, for problems with many dimensions, storing the words extracted from all of them is the cause of the significant memory issues described. As such, prior to creating any bags, we take a subsample of dimensions based on an accuracy estimate. We find this estimate using LOOCV on bags created from disjoint windows rather than a sliding one. Any dimension with an accuracy estimate less than 85% of the highest accuracy dimension is not retained for this classifier. Additionally, we set a limit of 20 maximum dimensions retained for each classifier, keeping those with the highest accuracy. To reduce the memory impact of saving features from multiple dimensions, we do not record bigrams and reduce the max number of classifiers retained for multivariate datasets. The build process for the individual classifiers used in TDE is displayed in Algorithm 4.3.

Algorithm 4.2 TDE(A list of n cases of length m with d dimensions, $\boldsymbol{T} = (\boldsymbol{X}, \boldsymbol{y})$)

Parameters: the number of parameter samples k, the max ensemble size s

Let w be window length, l be word length, p be normalise/not normalise, α be alphabet size, h be the number of pyramid levels and b be MCB or IGB discretisation.
Let $\boldsymbol{c}$ be a list of s classifiers $(c_1, \ldots, c_s)$
Let $\boldsymbol{e}$ be a list of s classifier weights $(e_1, \ldots, e_s)$
Let $\boldsymbol{g}$ be a list of k parameter and accuracy pairs $(g_1, \ldots, g_k)$
Let $\boldsymbol{R}$ be a set of possible parameter combinations
$i \leftarrow 0$
$lowest\_acc \leftarrow \infty, lowest\_acc\_idx \leftarrow \infty$
while $i < k$ AND $|\mathbf{R}| > 0$ **do**
  if $i < 50$ **then**
    $[l, \alpha, w, p, h, b] \leftarrow$ randomSample($\boldsymbol{R}$)
  else
    $gp \leftarrow$ buildGaussianProcesses($\mathbf{G}$)
    $[l, \alpha, w, p, h, b] \leftarrow$ bestPredictedParameters($\boldsymbol{R}, gp$)
  $\boldsymbol{R} = \boldsymbol{R} \setminus \{[l, \alpha, w, p, h, b]\}$
  $\boldsymbol{T'} \leftarrow$ subsampleData($\boldsymbol{T}$, 0.7)
  $cls \leftarrow$ buildIndividual($\boldsymbol{T'}, l, \alpha, w, p, h, b$)
  $acc \leftarrow$ LOOCV(cls) { *train data accuracy*}
  if $i < s$ **then**
    if $acc < lowestAcc$ **then**
      $lowestAcc \leftarrow acc$, $lowestAccIndex \leftarrow i$
    $c_i \leftarrow cls$, $e_i \leftarrow acc^4$
  else if $acc > lowestAcc$ **then**
    $c_{lowestAccIndex} \leftarrow cls$, $e_{lowestAccIndex} \leftarrow acc^4$
    $[lowestAcc, lowestAccIndex] \leftarrow$ findNewLowestAcc($\boldsymbol{c}$)
  $g_i \leftarrow \{[l, \alpha, w, p, h, b], acc\}$
  $i \leftarrow i + 1$

4.4 Results

Our experiments are run primarily on 112 datasets from the univariate UCR archive [36], removing any datasets that are unequal length or contain missing values. For our multivariate experiments, we use 26 datasets from the UEA archive [3]. For each classifier-dataset combination, we run 30 stratified resamples, with the resample and classifier seeded to its resample number. Our results are summarised primarily using critical difference diagrams over average ranks and other figures for our performance metrics. Both archives, our choice of datasets and our exper-

Algorithm 4.3 IndividualTDE(A list of n cases of length m with d dimensions, $\boldsymbol{T} = (\boldsymbol{X}, \boldsymbol{y})$)

Parameters: word length l, alphabet size α, window length w, normalisation parameter p, No. pyramid levels h, breakpoint function b

Let $\boldsymbol{H}$ be a list of n histograms $(\mathbf{h}_1, \ldots, \mathbf{h}_n)$
Let $\boldsymbol{k}$ be a list of dimension indexes found using LOOCV
Let $\boldsymbol{B}$ be a list of $|\boldsymbol{k}|$ l by α matrices of breakpoints found by b $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{|\boldsymbol{k}|})$
Let $\boldsymbol{O}$ be a matrix of $|\boldsymbol{k}|$ by n by $m - w + 1$ containing the words for each window
for $i \leftarrow 1$ to n **do**
  for $g \leftarrow 1$ to $|\boldsymbol{k}|$ **do**
    for $j \leftarrow 1$ to $m - w + 1$ **do**
      $\mathbf{s} \leftarrow \boldsymbol{x}_{i,k_g,j} \ldots \boldsymbol{x}_{i,k_g,j+w-1}$
      $\boldsymbol{q} \leftarrow \text{DFT}(\mathbf{s}, l, \alpha)$ { $\boldsymbol{q}$ *is a vector of the complex DFT coefficients*}
      if p **then**
        $\boldsymbol{q'} \leftarrow (q_2 \ldots q_{l/2+1})$
      else
        $\boldsymbol{q'} \leftarrow (q_1 \ldots q_{l/2})$
      $r \leftarrow \text{SFAlookup}(\boldsymbol{q'}, b_g)$
      if $r \neq o_{g,i,j-1}$ **then**
        for $v \leftarrow 1$ to h **do**
          $pos \leftarrow$index(spatialPosition(r,v),g)
          $h_{i,pos} \leftarrow h_{i,pos} + 1$
        $bi \leftarrow$index(bigram(r,$o_{g,i,j-w}$),g)
        $h_{i,bi} \leftarrow h_{i,bi} + 1$
      $o_{g,i,j} \leftarrow r$

imental methodology are described in detail in Chapter 3. Appendix B provides the average score for over the relevant archive for each of our experiments, with more in depth and dataset specific results files being available on the accompanying webpage for the TDE publication [86] linked within. Table 4.3 provides parameter configurations for the algorithms used in our experiments.

We split our results into four sections, each aiming to answer a single question regarding our dictionary advances.

1. Does cBOSS speed up the BOSS algorithm without a significant different in accuracy? (Section 4.4.1)

Table 4.3 Classifier parameters used in our dictionary based experimentation.

Classifier	Parameters
cBOSS	250 parameter sets sampled, 50 max ensemble size, 0.7 subsample proportion
RBOSS	250 parameter sets sampled
TDE	250 parameter sets sampled, 100 max ensemble size (50 if multivariate), 0.7 subsample proportion, 50 initial random parameter sets
cS-BOSS	250 parameter sets sampled, 50 max ensemble size, 0.7 subsample proportion
WEASEL	0.1 χ feature selection threshold
MUSE	2 χ feature selection threshold
DTW_D/I	Full warping window
TSF	500 trees, $\sqrt{m}$ intervals per tree
RISE	500 trees
STC	Rotation Forest classifier
ROCKET	10,000 kernels, Ridge Regression classifier

2. Does TDE outperform other dictionary based approaches while retaining scalability? (Section 4.4.2)

3. Is TDE competitive on multivariate datasets compared to benchmarks and other dictionary algorithms. (Section 4.4.3)

4. How do TSC approaches perform on our Right Whale case study dataset, and do our changes allow dictionary based algorithms to finish on the large dataset? (Section 4.4.4)

4.4.1 Contractable BOSS vs. BOSS

Our main aim with the cBOSS changes is to improve the algorithm's scalability without a noticeable performance drop. In our comparison, we include a version of BOSS called RBOSS with no weights or filtering, just fully random parameter selection. The critical difference diagram in Figure 4.5 shows the accuracy impact of our cBOSS changes, while Figure 4.6 shows the difference in training time against accuracy rank. There is no significant difference in accuracy between

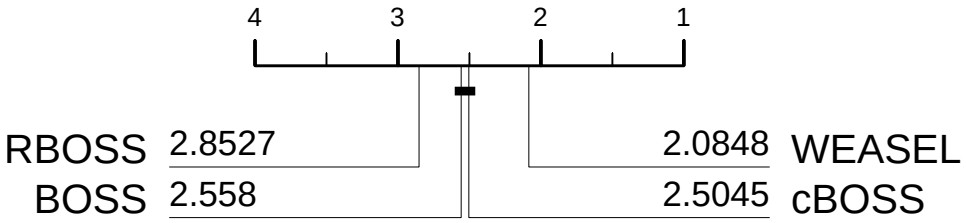


Fig. 4.5 Accuracy critical difference diagram comparing cBOSS to 3 dictionary classifiers on the 112 equal length UCR datasets.

Fig. 4.6 Scatter graph plotting average accuracy rank against build time on a log scale for cBOSS and other dictionary classifiers. Classifiers close to the bottom are more accurate, and those close to the left are faster.

cBOSS and BOSS, with cBOSS building over an order of magnitude faster than BOSS.

To reduce the risk of bias or any suggestion of cherry-picking, we have conducted experiments with all the datasets in the archive. However, many of these problems are small. To examine the effect on larger problems, we take a closer look at the results for 16 problems on which BOSS takes over an hour to build. We list these datasets in Appendix B. Table 4.4 shows the pattern of results is the same on these 16 data. cBOSS is still not significantly worse than BOSS, but over 13 times faster.

Table 4.4 Average large dataset performance by classifier using accuracy rank, accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL). Included is total build time over all datasets relative to BOSS.

Classifier	AccRank	Accuracy	BalACC	AUROC	NLL	Build Time
cBOSS	2.6250	0.8199	0.8127	**0.9558**	0.7951	7.44%
RBOSS	3.3750	0.7661	0.7595	0.9330	0.9587	**0.76%**
WEASEL	**1.3750**	**0.8374**	**0.8327**	0.9550	1.0432	57.79%
BOSS	2.6250	0.8171	0.8101	0.9533	**0.7908**	100%

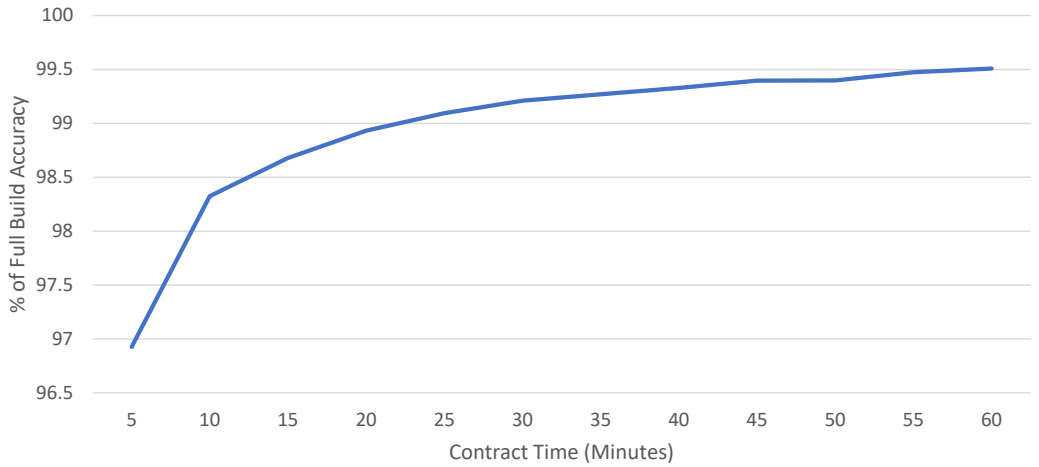
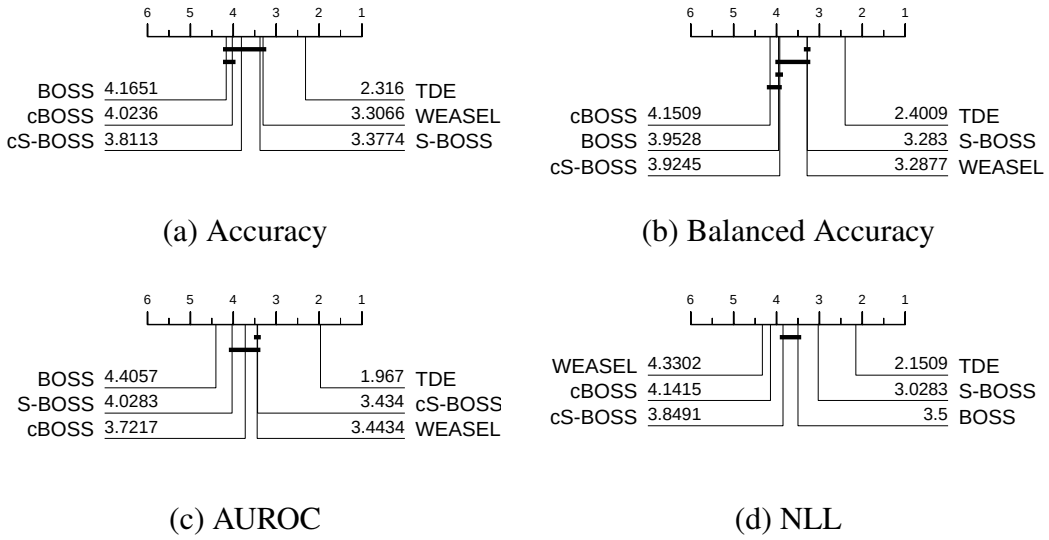


Fig. 4.7 Average accuracy value of cBOSS for a range of contract times on 16 large datasets. Times range from 5 to 60 minutes linearly spaced in increments of 5.

To demonstrate the effectiveness of building cBOSS using a time contract, Figure 4.7 shows the change in accuracy over a range of t values on the 16 large problems. As shown, an increase in time the classifiers contract increases accuracy on average, though this increase slows as the parameter search progresses. Within 20 minutes, average accuracy is within 99% of the ful build and reaches 99.5% after an hour. The versatility of being able to build the best classifier within a train time limit is a large usability boost to the classifier, making this aspect much more predictable.



(a) Accuracy

(b) Balanced Accuracy

(c) AUROC

(d) NLL

Fig. 4.8 Critical difference diagrams for six dictionary based classifiers over 106 UCR problems. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

4.4.2 Temporal Dictionary Ensemble vs. Dictionary Approaches

For our comparison of current dictionary based classifiers, we were only able to obtain complete results for 106 of the 112 datasets. This was due to the long run time of S-BOSS surpassing our seven day limit. The missing problems are: ElectricDevices; FordA; FordB; HandOutlines; NonInvasiveFetalECGThorax1; and NonInvasiveFetalECGThorax2. We provide scores on all 112 datasets for the classifiers which finished them in Appendix B.

Figure 4.8 shows a critical difference diagrams for the six dictionary based classifiers considered. TDE is significantly more accurate than all other classifiers on all performance metrics. For accuracy there are two cliques: cBOSS is significantly worse than S-BOSS, cS-BOSS and WEASEL. BOSS and cBOSS show no significant difference. This also confirms that the simple hybrid cS-BOSS is no better than S-BOSS in terms of accuracy. For all metrics apart from negative

Table 4.5 Summary of run time for six dictionary based classifiers over 106 UCR problems. The median time over 30 resamples is used for each dataset.

Classifier	Max Build Time (Hours)	Total Build Time (Hours)
BOSS	11.33	52.10
cBOSS	**0.63**	**3.74**
S-BOSS	34.11	148.82
cS-BOSS	2.91	12.62
WEASEL	4.86	28.50
TDE	5.67	21.73

log-likelihood, WEASEL performs close to the top. This suggests the classifier may have trouble producing probability estimates with its logistic regression classifier.

Table 4.5 summarises the run time of each dictionary based classifier. TDE completes all the problems in under a day. It is faster than WEASEL and considerably faster than S-BOSS. The max run timings for BOSS and S-BOSS demonstrate the problem with the traditional BOSS algorithm addressed by cBOSS and cS-BOSS: the ensemble design means they may require a long runtime, and it is not very predictable when this will happen. Figure 4.9 shows the scatter plot of runtime for BOSS vs TDE and demonstrates that TDE scales much better than BOSS into larger problem sizes, but is slower on smaller problems, most likely due to the addition of the GP and spatial pyramids.

Figure 4.10 compares TDE to two of the better scaling approaches in cBOSS and WEASEL using accuracy pairwise scatter diagrams. While TDE consistently beats cBOSS with few datasets performing noticeably better for the latter, WEASEL is much more varied. TDE still wins on the majority of datasets, but its losses are much larger. The differing structure used by WEASEL with its pipeline is able to achieve higher accuracy than ensemble methods like TDE and cBOSS on some datasets, but is considerably less accurate on others.

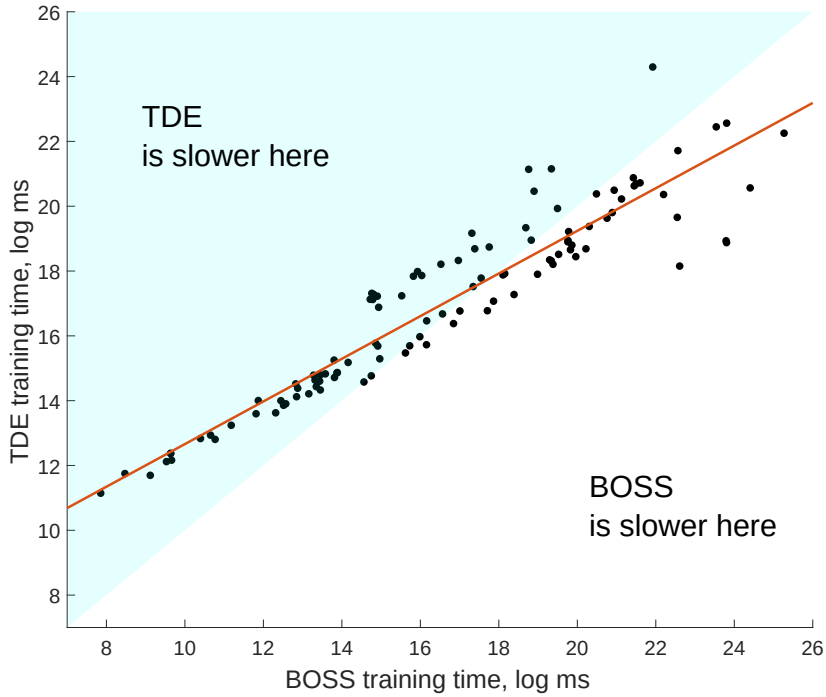
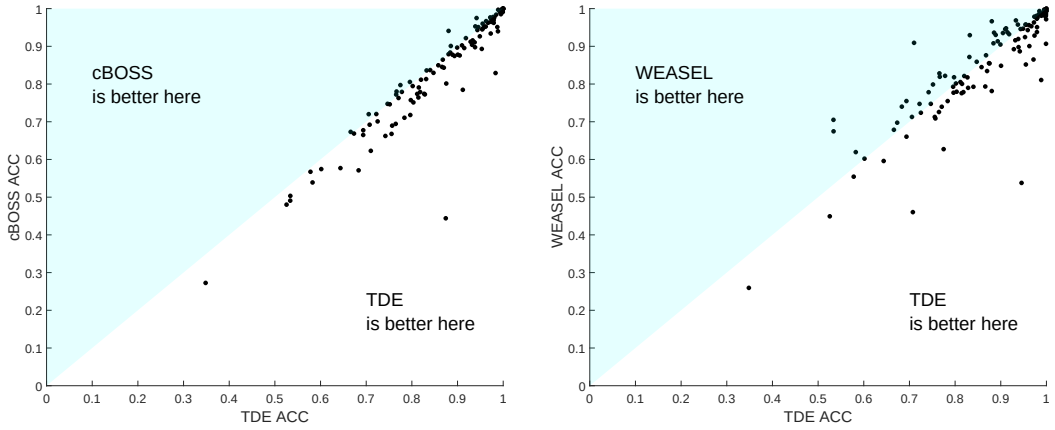


Fig. 4.9 Pairwise scatter diagram, on log scale, of TDE and BOSS training times. TDE has larger overheads which make it slower on smaller problems, but it scales much better towards larger problems.

4.4.3 Temporal Dictionary Ensemble on Multivariate Data

We compare the multivariate capabilities of TDE against two multivariate benchmarks in DTW_I and DTW_D, and two dictionary classifiers in MUSE and a majority vote ensemble of cBOSS classifiers built on each dimension (cBOSS_I). MUSE was only able to finish on 20 of the 26 UEA archive datasets, ceasing build attempts when over 500 gigabytes of memory was required. The datasets MUSE failed to build are DuckDuckGeese; EigenWorms; FaceDetection; MotorImagery; PEMS-SF; and PhonemeSpectra.

Figure 4.11 shows a critical difference diagram for the 20 remaining datasets. Figure 4.12 removes MUSE, building the diagram with all 26 datasets. TDE is not significantly different to MUSE on 20 datasets, both of which are significantly better than our benchmark classifiers. However, TDE can complete the six larger



82 / 2 / 22 WDL 66 / 1 / 39 WDL

Fig. 4.10 Pairwise scatter diagram comparing the accuracy of TDE to cBOSS and WEASEL. The Win/Draw/Loss of each dataset for TDE against both of these classifiers is shown below the figure.

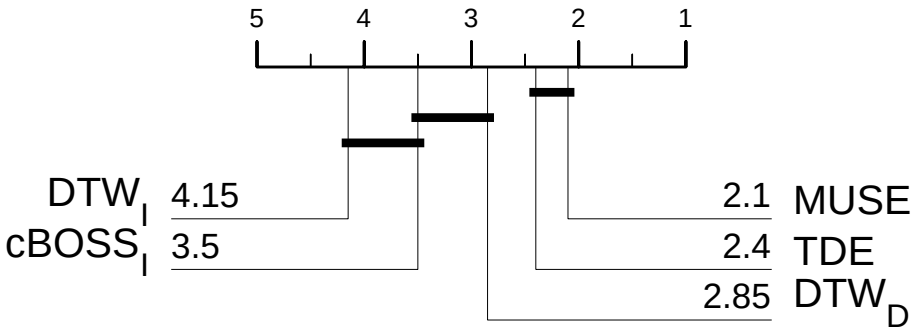


Fig. 4.11 Accuracy critical difference diagram comparing TDE to MUSE and multivariate benchmarks on the 20 equal length UEA datasets.

datasets MUSE cannot, and is still significantly better than the benchmarks with them added.

4.4.4 A Case Study on the Classification of Right Whales

Alongside wanting to improve HIVE-COTE, our interest in making BOSS scalable arose due to our desire to use it with an application in classifying North Atlantic right whales based on acoustic samples. These whales are endangered, and being able to detect them accurately would help with conservation. Previous work has been done in classifying the presence of whale species using acoustic data with

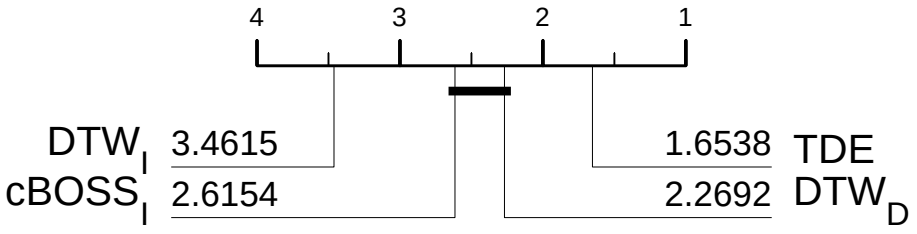


Fig. 4.12 Accuracy critical difference diagram comparing TDE to multivariate benchmarks on the 26 equal length UEA datasets.

whale vocalisations [41, 113]. However, TSC approaches have not been applied. We provide more detail on the dataset used in Chapter 3.

This problem is a good example of a large dataset for which it is infeasible to use BOSS. The dataset contains 12,896 cases, with a series length of 4000. For benchmarking alongside cBOSS and TDE on this dataset, we test the five classifiers that make up the HIVE-COTE ensemble, as well as three potential candidates for the ensemble WEASEL and Proximity Forest (PF) [81] and ROCKET [37]. Only three of these classifiers, Time Series Forest (TSF) [40], Random Interval Spectral Ensemble (RISE) [78] and ROCKET, will complete within seven days. The Shapelet Transform Classifier (STC) [17] is contractable, and we present results with a contracted time limit of two days for the tranform. BOSS did not complete, and also required huge amounts of memory (greater than 100 gigabytes) to run at all. Attempts to build WEASEL ceased after a 300 GB memory limit was exceeded. With default parameters, neither cBOSS nor TDE completed, even with the scalability improvements made. The new ensemble structure allows for contacting, however. We present the results for both with a two-day contract. Table 4.6 shows the accuracy and build time on the whales dataset for each of the finished classifiers over a 10-fold cross-validation. These exploratory results suggest shapelet approaches may be useful for this application, with the next best classifier RISE being designed for audio data also performing well.

Table 4.6 Accuracy and build time in hours for each of the potential HIVE-COTE components which managed to finish processing the right whale dataset over a 10-fold cross-validation.

Classifier	Accuracy	Build Time (Hours)
cBOSS	0.8037	48.1
TDE	0.7969	50.49
STC	**0.8559**	92.37
RISE	0.8326	6.70
TSF	0.7550	**2.95**
ROCKET	0.7969	5.21

4.5 Temporal Dictionary Ensemble Component Importance

At its core, TDE builds on the SFA based bag-of-words model first introduced in BOSS, incorporating improvements from other extensions along with its own novelty. In this section, we aim to answer three questions regarding the necessity of these individual additions to improve the algorithm's accuracy. These are:

1. Are the additions made for TDE necessary, and which ones have the largest impact on performance? (Section 4.5.1)

2. Can our Gaussian process based parameter selection be replaced by just increasing the number of parameters randomly selected? (Section 4.5.2)

3. What is the impact of including WEASEL features not included in the TDE algorithm? (Section 4.5.3)

4.5.1 Temporal Dictionary Ensemble Ablation Study

The most straightforward way of determining the significance of TDE features is individually removing each in an ablation study. Figure 4.13 compares the full TDE algorithm to variations with important components removed in a critical difference diagram. TDE-Bigrams removes the SFA word bigrams; TDE-Spatial

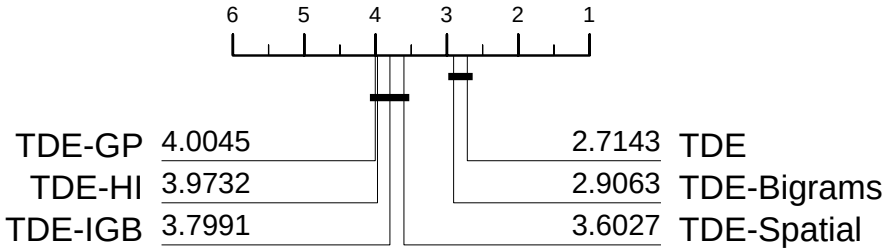


Fig. 4.13 Accuracy critical difference diagrams for variants of TDE with individual features removed over 112 UCR problems.

removes spatial pyramids with the accompanying parameter; TDE-HI replaces histogram intersection with BOSS distance to compare the bag-of-words; TDE-IGB removes the breakpoint parameter, only using MCB; TDE-GP returns to the random parameter selection used in cBOSS.

With the exception of bigrams, removing any TDE feature results in a significantly worse classifier accuracy wise. It should be noted that bigrams do improve the classifier, and without them, it is not significantly different from WEASEL. The gains are noticeably smaller than the other components, however, and can reasonably be removed to improve memory and prediction time efficiency, like we do in our multivariate experiments.

4.5.2 Impact of the Gaussian Process Parameter Selection

While Gaussian process parameter selection used by TDE provides a significant boost in accuracy, it is not readily apparent how the change contributes to this compared to just evaluating more parameter sets randomly. Figure 4.14 compares cS-BOSS and TDE to: cS-BOSS using TDE's GP parameter search (GP-cS-BOSS); TDE without the GP parameter search but all else equal (TDE-NoGP); and TDE without the GP parameter search once more but with triple the k parameter value, 750 (TDE-NoGP750). Removing either the GP or additional algorithm features from WEASEL results in a significantly worse classifier, though still significantly better than cS-BOSS. Randomly selecting 750 parameter sets is also significantly

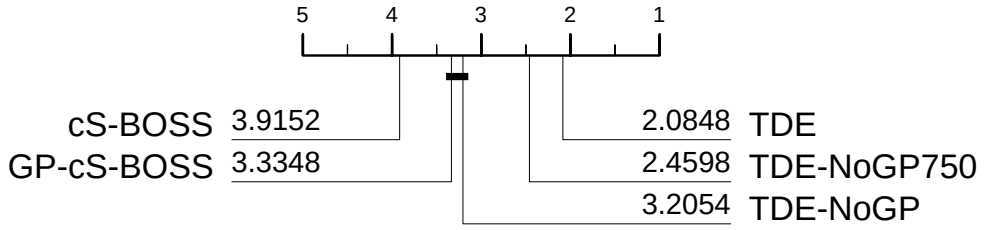


Fig. 4.14 Accuracy critical difference diagrams for comparing TDE with different variations of parameter selection over 112 UCR problems.

worse than selecting 250 using GP, as well as being nearly twice as slow to train on average over the 112 datasets.

4.5.3 Additional WEASEL Components

While all the improvements from the S-BOSS algorithm were included in TDE, only a few elements of WEASEL are incorporated. Despite abandoning the ensemble structure shown to have a significant impact on the accuracy of BOSS [67], WEASEL makes an improvement in accuracy. The large feature space and the swap to a logistic regression base classifier are incompatible with the ensemble structure of TDE, as the required train time to create a reasonable sized ensemble would render it practically unusable. A number of individual components could be leveraged however, these are the IGB for word discretisation, the ANOVA F-test to retain Fourier terms, bigrams and the chi-squared test for feature selection. Of these features, we include IGB as an alternative to the MCB and word bigrams.

While we include IGB and bigrams as a parameter, we elect to exclude the ANOVA F-test and feature selection. Our initial decision to leave out the feature selection is simple, each individual TDE classifier has a smaller number of words than WEASEL, as it only contains words from a single window length. Feature importance is also determined through the weighted ensemble. Compared to even the small improvement from bigrams, we found the ANOVA F-test did not provide an increase in performance to justify the extra complexity. Figure 4.15 compares

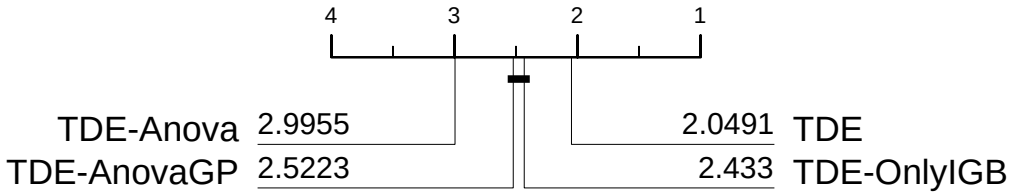


Fig. 4.15 Accuracy critical difference diagrams for comparing TDE with different WEASEL features over 112 UCR problems.

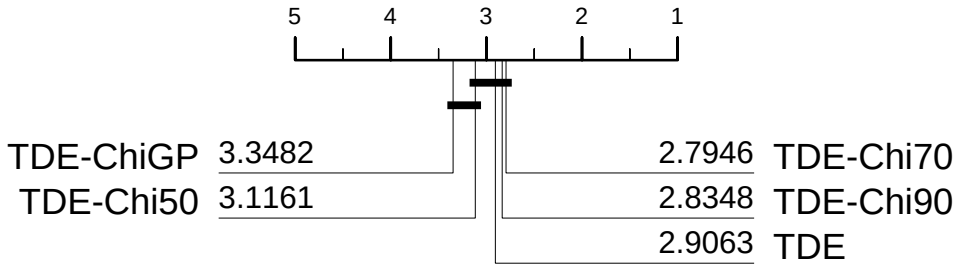


Fig. 4.16 Accuracy critical difference diagrams for comparing TDE with different variations of chi-squared feature selection over 112 UCR problems.

the ANOVA F-test to IGB, showing that the impact of adding the feature as a parameter for the GP or always applying it is significantly worse than adding IGB as a parameter. We also include a version where IGB is the only method of generating breakpoints, showing that there is still value in keeping MCB.

Figure 4.16 shows TDE with feature selection for each individual classifier for different χ values. TDE-Chi50, TDE-Chi70 and TDE-Chi90 set the χ value to 0.5, 0.7 and 0.9 respectively. TDE-ChiGP selects the χ value using GP parameter selection, with the available values being 0.4, 0.6, 0.8 and 1. Other than selecting χ using the GP model, none of the χ thresholds result in a significantly worse model. Table 4.7 contains the raw accuracy and total build time in hours for each model. There is a noticeable drop in accuracy, though the pairwise diagram in Figure 4.17 shows that this is due to a small amount of datasets losing a very large amount of accuracy with feature selection introduced. Always performing feature selection does speed up the train time for TDE on average, with the gain when estimating accuracy with LOOCV outweighing the time taken to remove words.

Table 4.7 Accuracy and build time in hours for TDE with different levels of feature selection.

Classifier	Accuracy	Total Build Time (Hours)
TDE	**0.8589**	66.39
TDE-Chi50	0.8397	**49.74**
TDE-Chi70	0.8405	54.55
TDE-Chi90	0.8407	57.73
TDE-ChiGP	0.8396	65.36

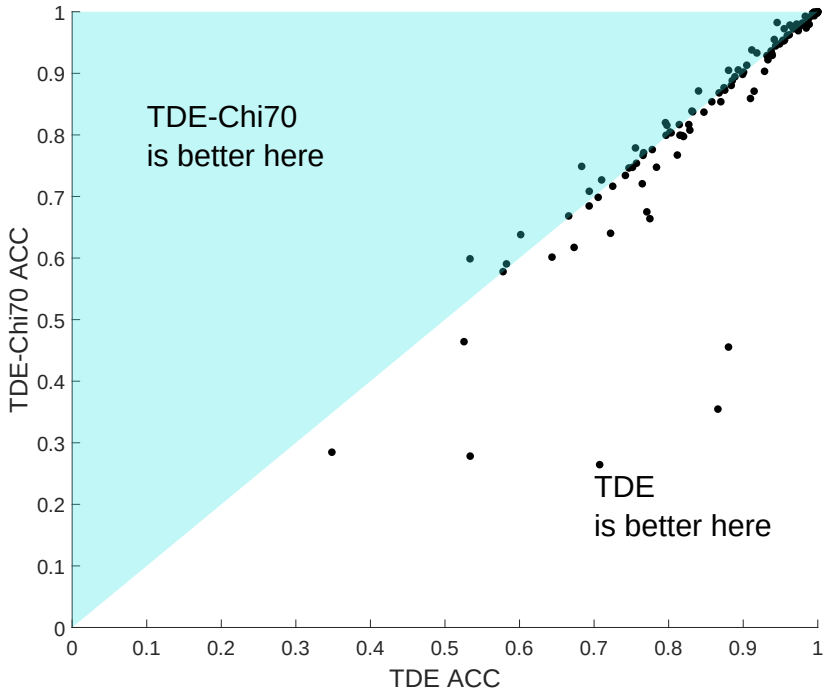


Fig. 4.17 Pairwise scatter diagram comparing the accuracy of TDE and the highest ranking feature selection variant. Applying feature selection causes large losses of accuracy on some problems.

4.6 Conclusion

In conclusion, this chapter proposes two new algorithms building off the SFA bag-of-words model for the dictionary based representation of time series classifiers. We initially hypothesise that the ensemble structure of BOSS is unnecessarily thorough with its grid search, and similar accuracy could be achieved at a smaller cost with random selection. We show that while full random selection is not feasible, retaining a filtered and weighted ensemble with random selection in cBOSS achieves this,

building over an order of magnitude faster. This work leads to the development of TDE, which combines the best elements of existing dictionary based classifiers with a novel method of improving the ensemble through a Gaussian process model for parameter selection. TDE is more accurate and scalable than current top performing dictionary algorithms.

TDE has some drawbacks. It is still memory intensive, while this memory usage is more predictable than BOSS, the addition of spatial pyramids and bigrams require more words to be stored per ensemble member. The set amount of ensemble members does not necessarily guarantee there will be less than BOSS retains on large problems as well. This is especially true for multivariate data, where extra words are stored for multiple dimensions. The benefit of the out ensemble structure is that this can be controlled with the max ensemble size parameter, however. Like all nearest neighbour classifiers, TDE is relatively slow to classify new cases. If fast predictions are required, WEASEL may be preferable. Future dictionary based work could focus on making TDE more scalable. Our experiments in Section 4.5.3 show that there is promise in applying feature selection, and a different approach or including it as a simple on or off parameter similar to IGB and MCB could mitigate both the above issues. While TDE will not always be the best approach, we believe it forms a strong core for the dictionary representation along with cBOSS for when fast builds are required and WEASEL for fast predictions with a feature selection based approach.

The next chapter of this thesis continues the theme of improving the HIVE-COTE ensemble through developing its components. The two remaining problem classifiers for scalability are ST-HESCA [17] and EE [76]. However, ST-HESCA can be easily contracted with the improvements made to form STC [17], and EE can be removed from the ensemble without a major loss in accuracy [5]. The new HIVE-COTE 1.0 ensemble after our development in this chapter contains both cBOSS replacing BOSS, and an upgraded contractable STC. The Time Series Forest

(TSF) [40] and Random Interval Spectral Ensemble (RISE) [78] interval based classifiers are retained as is. We detail the improvements made for HIVE-COTE 1.0 in Chapter 2, but note that we consider this the first practically usable version of HIVE-COTE, and groundwork for the significantly more accurate HIVE-COTE 2.0 using TDE. With the scalability issues of HIVE-COTE improved, we look at the components which have fallen behind in accuracy.

Chapter 5

Interval Based Advances: The Canonical Interval Forest

Contributing Publications

- Middlehurst, M., Large, J., & Bagnall, A. (2020). The Canonical Interval Forest (CIF) Classifier for Time Series Classification. In *2020 IEEE International Conference on Big Data (Big Data)* (pp. 188-195), IEEE.
- Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., & Bagnall, A. (2021). The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery* 35(2), 401-449.

5.1 Introduction

Chapter 4 describes two advancements to the dictionary based representation of Time Series Classification (TSC). The first of these is the better scaling improvement to the Bag of Symbolic Fourier Approximation Symbols (BOSS) [105] component of the Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-

COTE) [78], Contractable BOSS (cBOSS) [88]. The second is a significantly more accurate algorithm than others in the representation, the Temporal Dictionary Ensemble (TDE) [86]. With these improvements, we advanced our goal of making HIVE-COTE scalable for medium to large sized datasets with HIVE-COTE 1.0 [5]. With the meta ensemble's scalability issue significantly improved, we looked at increasing the weaker component's accuracy.

The Time Series Forest (TSF) [40] and Random Interval Spectral Ensemble (RISE) [78] are both interval based classifiers, extracting subseries and transforming these subseries into a set of features to build a base classifier. Both TSF and RISE are members of the HIVE-COTE 1.0 ensemble. They are also the weakest members of the ensemble in terms of accuracy, performing significantly worse than cBOSS and the Shapelet Transform Classifier (STC) [17]. TSF is a forest of trees built on multiple time series intervals of random length and position. Summary statistics in the mean, median and standard deviation are extracted from each interval and concatenated to form the feature vector used to build each tree. RISE is similar in structure, but uses a single interval for each tree and spectral features rather than summary statistics. Both algorithms are detailed in Chapter 2.

Our hypothesis is that the algorithm's weak performance compared to other single representation domains is due to the features extracted from intervals, rather than the forest structure or the method of interval extraction used. By extracting more informative statistics, we can produce an interval approach with similar performance to other HIVE-COTE components. Both TSF and RISE are by far the fastest HIVE-COTE components, and can afford the extra cost to extract more informative features. Our contributions in the chapter involve developing an accurate interval based algorithms leveraging the Canonical Time-series Characteristics (Catch22) [80] set of features.

This new classifier, the Canonical Interval Forest (CIF) (originally published in [85]) is significantly more accurate than TSF and RISE. This improvement

is achieved by incorporating the Catch22 features. Another variant, the Diverse Representation CIF (DrCIF) (enhancements introduced in [87]), makes further improvements by extracting intervals from transformed series in the first order differences and power spectrum. This multiple series representation approach combines the feature extraction of TSF and the spectral features of RISE used in HIVE-COTE 1.0. We find that DrCIF is not significantly worse than the best recently proposed algorithms based on a single representation.

The rest of this chapter is structured as follows. Section 5.2 explains the alterations to TSF in order to include the Catch22 features and form CIF, and then the further enhancements made for the DrCIF algorithm. Section 5.3 describes the results of our experiments comparing the improved interval approaches to TSF, RISE and the top performing algorithms of from other single feature representations. We further analyse CIF in Section 5.4, looking at the impact of decisions made, experiments which influenced DrCIF and the generation of train accuracy estimates required for inclusion in HIVE-COTE. Finally, in Section 5.5 we conclude.

5.2 The Canonical Interval Forest [85]

The Catch22 features summarise time series characteristics very concisely, but classifiers built on the Catch22 transformation over the whole series are significantly worse than alternative feature based approaches [84]. TSF is very fast and provides diversity that improves HIVE-COTE, but as a stand-alone classifier it is significantly worse than two HIVE-COTE 1.0 components, cBOSS and STC [5]. Our research question was whether by replacing the three simple summary features used in TSF with the more descriptive Catch22 feature sets we could find a significantly better interval classifier, which in turn could lead to an improvement in HIVE-COTE. Our first approach is to simply replace feature extraction operations in TSF with the Catch22 features. For this chapter, we call this simple feature-swapped

version 'Hybrid'. Our feature set changes from $f(\cdot) = \{mean, stdev, slope\}$ to $f(\cdot) = \{c22f_1, c22f_2, \ldots, c22f_{22}\}$, where $c22f_i$ indicates the i^{th} canonical feature.

We describe the Catch22 extraction process in Chapter 2, but summarise the process and features again in the following text. The Catch22 features are a set of 22 features designed for time series data filtered from the features available in the highly comparative time series analysis (hctsa) toolbox [47]. After a pruning process removing feautres sensitive to mean and variance, the Catch22 features were derived from a clustering and filtering of the 7658 hctsa features based on the balanced accuracy metric, scalability and interpretability. A table of the Catch22 features is provided in Table 5.1 and a visualisation of the feature extraction process used in Figure 5.1, both originating from [80].

Experimentation presented in Section 5.3 shows the Hybrid is significantly more accurate than TSF, but this comes at a major time overhead. the Catch22 features are near-linear to compute, but clearly still more expensive overall than calculating just the mean, standard deviation and slope. There are also characteristics of the Catch22 features that cause a large increase in transform time for some data. Two features in particular, the positive and negative *'DN_OutlierInclude'* features, took a very long time to compute on certain intervals of unnormalised datasets, which exhibit large absolute value extremes. Because of this, CIF uses normalised intervals for these two Catch22 features (covered further in Section 5.4.3). The rest of the features are calculated on the unnormalised intervals, as the absolute values within particular intervals can hold important discriminatory information.

CIF is an ensemble of trees built using features extracted from randomly selected intervals. For a time series of length m there are $m(m-1)/2$ possible intervals that can be extracted. CIF extracts any interval with a length between 3 and m from any position in the series. Multiple intervals are selected for each decision tree base classifier, the derived features of which are concatenated to form a diverse training set for each ensemble member. For CIF, we include the three TSF features with

Table 5.1 Each of the Catch22 features with descriptions from Table 1 in [80]. All credit to the original authors for table layout and feature descriptions.

Feature Name	Description
Distribution:	
DN_HistogramMode_5	Mode of z-scored distribution (5-bin histogram)
DN_HistogramMode_10	Mode of z-scored distribution (10-bin histogram)
Simple temporal statistics:	
SB_BinaryStats_mean_longstretch1	Longest period of consecutive values above the mean
DN_OutlierInclude_p_001_mdrmd	Time intervals between successive extreme events above the mean
DN_OutlierInclude_n_001_mdrmd	Time intervals between successive extreme events below the mean
Linear autocorrelation:	
CO_f1ecac	First $1/e$ crossing of autocorrelation function
CO_FirstMin_ac	First minimum of autocorrelation function
SP_Summaries_welch_rect_area_5_1	Total power in lowest fifth of frequencies in the Fourier power spectrum
SP_Summaries_welch_rect_centroid	Centroid of the Fourier power spectrum
FC_LocalSimple_mean3_stderr	Mean error from a rolling 3-sample mean forecasting
Nonlinear autocorrelation:	
CO_trev_1_num	Time-reversibility statistic, $\langle (x_t+1-x_t)^3 \rangle_t$
CO_HistogramAMI_even_2_5	Automutual information, $m=2$, $\tau=5$
IN_AutoMutualInfoStats_40_gaussian_fmmi	First minimum of the automutual information function
Successive differences:	
MD_hrv_classic_pnn40	Proportion of successive differences exceeding 0.04σ [89]
SB_BinaryStats_diff_longstretch0	Longest period of successive incremental decreases
SB_MotifThree_quantile_hh	Shannon entropy of two successive letters in equiprobable 3-letter symbolization
FC_LocalSimple_mean1_tauresrat	Change in correlation length after iterative differencing
CO_Embed2_Dist_tau_d_expfit_meandiff	Exponential fit to successive distances in 2-d embedding space
Fluctuation Analysis:	
SC_FluctAnal_2_dfa_50_1_2_logi_prop_r1	Proportion of slower timescale fluctuations that scale with DFA (50% sampling)
SC_FluctAnal_2_rsrangefit_50_1_logi_prop_r1	Proportion of slower timescale fluctuations that scale with linearly rescaled range fits
Others:	
SB_TransitionMatrix_3ac_sumdiagcov	Trace of covariance of transition matrix between symbols in 3-letter alphabet
PD_PeriodicityWang_th0_01	Periodicity measure of [133]

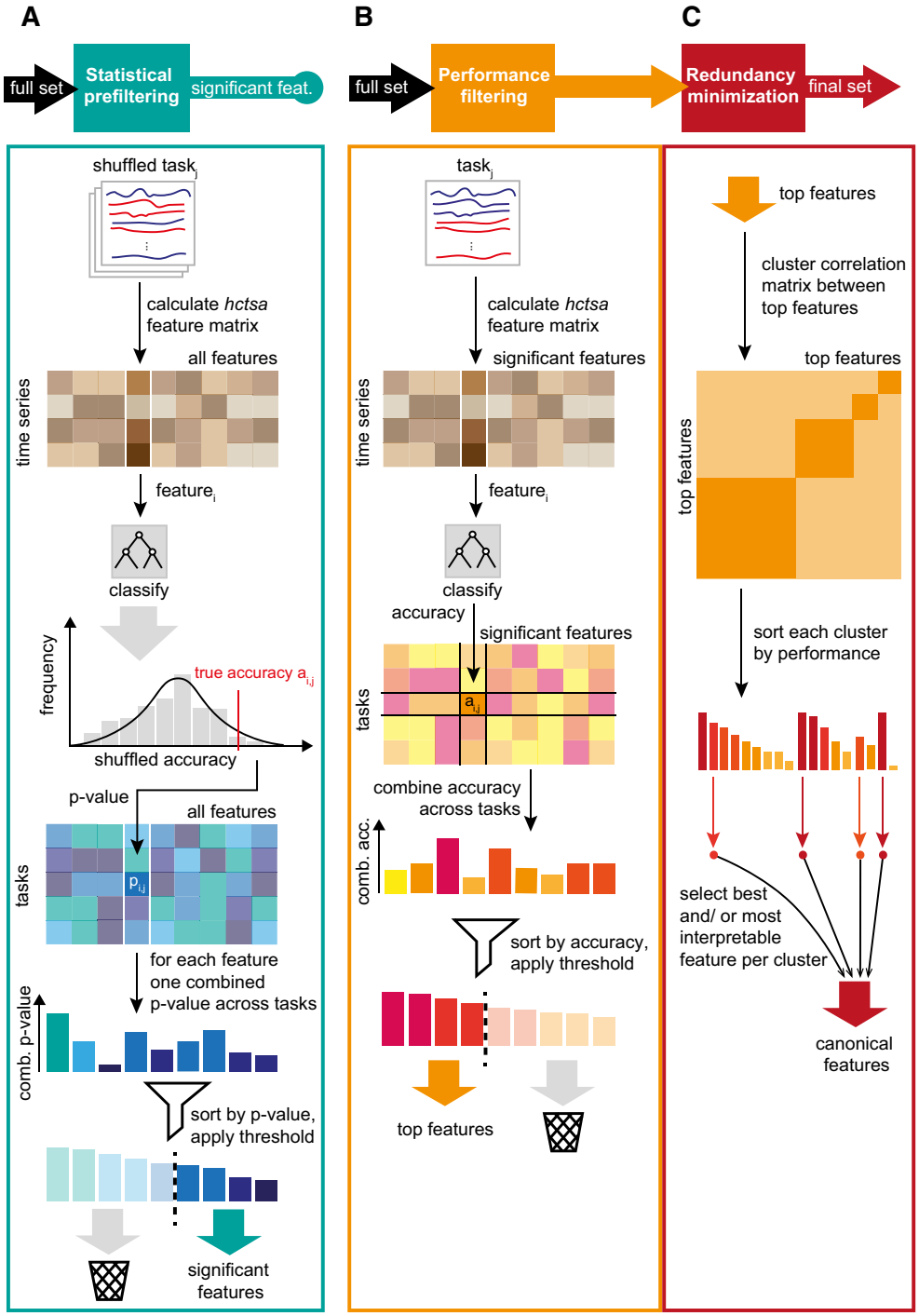


Fig. 5.1 Figure 1 from [80] visualising the feature extraction process used to obtain the Catch22 features. All credit to the original authors.

Algorithm 5.1 CIF(A list of n cases of length m with d dimensions, $\boldsymbol{T} = (\boldsymbol{X}, \boldsymbol{y})$))

Parameters: the number of trees, r, the number of intervals per tree, k, and the number of attributes subsampled per tree, a

Let $\boldsymbol{f} = (f_1 \dots f_r)$ be the trees in the forest
for $i \leftarrow 1$ to r **do**
  Let $\boldsymbol{S}$ be a matrix of n cases $(\boldsymbol{s}_1 \dots \boldsymbol{s}_n)$ with $a \cdot k$ attributes
  Let $\boldsymbol{u}$ be a list of a randomly selected attribute indices $(u_1 \dots u_a)$
  for $j \leftarrow 1$ to k **do**
    $b = rand(1, m-3)$ { *interval position*}
    $l = rand(3, m)$ { *interval length*}
    $o = rand(1, d)$ { *interval dimension*}
    for $t \leftarrow 1$ to n **do**
      for $c \leftarrow 1$ to a **do**
        $s_{t,a(j-1)+c} \leftarrow summaryStat(u_c, \boldsymbol{X}_{t,o,b:(b+l-1)})$
  $f_i.buildTimeSeriesTree([\boldsymbol{S}, \boldsymbol{y}])$

the Catch22 features. Catch22 is designed for normalised data, and these simple summary statistics help CIF to capture scale when necessary, as well as being very cheap to process. We leverage the larger feature space and inject additional diversity into the ensemble by randomly sampling features for each tree. This has the added benefit of improving time efficiency. Finally, we employ the Time Series Tree (TST) [40] originally used by TSF rather than the Random Tree used in the open source Java implementation contained in *tsml* for the 2017 bake off comparison of TSC algorithms [8]. For multivariate data, CIF randomly selects the dimension used for each interval. The build process for CIF is provided in Algorithm 5.1.

While CIF does have some spectral features through the Catch22 feature set, we aim to further extend this though integrating the power spectrum series representation used by RISE. While RISE builds its tree on the full transformed series, we look to extract summary statistics using the Catch22 features as we have done for the base series. Another interval method in Supervised TSF (STSF) [19] also integrates spectral features through extracting basic summary statistics from the periodogram of the series, and adds the third representation in first order differences.

Algorithm 5.2 DrCIF(A list of n cases of length m with d dimensions, $\boldsymbol{T} = (\boldsymbol{X}, \boldsymbol{y})$)

Parameters: the number of trees, r, the number of intervals per representation for each tree, k, and the number of attributes subsampled per tree, a { *where rm is the length of a representations series* ($|\boldsymbol{V}_y|$) }

Let $\boldsymbol{f} = (f_1 \dots f_r)$ be the trees in the forest
Let $\boldsymbol{V}$ be a 3 by n by d list of series with variable length, containing the base series list $\boldsymbol{X}$ and $\boldsymbol{X}$ transformed to periodograms and first order differences
for $i \leftarrow 1$ to r **do**
    Let $\boldsymbol{S}$ be a matrix of n cases $(\boldsymbol{s}_1 \dots \boldsymbol{s}_n)$ with $a \cdot k \cdot 3$ attributes
    Let $\boldsymbol{u}$ be a list of a randomly selected attribute indices $(u_1 \dots u_a)$
    for $y \leftarrow 1$ to 3 **do**
        for $j \leftarrow 1$ to k **do**
            $b = rand(1, rm-4)$ { *interval position*}
            $l = rand(4, rm/2)$ { *interval length*}
            $o = rand(1, d)$ { *interval dimension*}
            for $t \leftarrow 1$ to n **do**
                for $c \leftarrow 1$ to a **do**
                    $s_{t,ya(j-1)+c} \leftarrow summaryStat(u_c, \boldsymbol{V}_{y,t,o,b:(b+l-1)})$
    $f_i.buildTimeSeriesTree([\boldsymbol{S}, \boldsymbol{y}])$

First order differences or derivative series have seen usage in other approaches, such as distances [62, 123].

We differentiate this variant of CIF extracting intervals from a diverse set of series representations as DrCIF. DrCIF extracts intervals from the base series, a periodogram representation, and adopts the first order differences representation from STSF. For both of these new representations, the full dataset is transformed, and intervals are then extracted from the newly transformed series. A subset of features is extracted from each interval, as is done in CIF. The pool of 25 features used previously are used with four additions in the median; interquartile range; minimum; and maximum for a total of 29 features. For each of the 3 series representations, k phase dependent intervals with randomly selected positions and lengths are extracted. Features are extracted from each interval and series representation, and concatenated into a single a $3 \cdot k \cdot a$ size feature vector to build the tree. Generally, we select k as a function of the representation series length rm. Each representation will differ in its length, with the periodogram being half

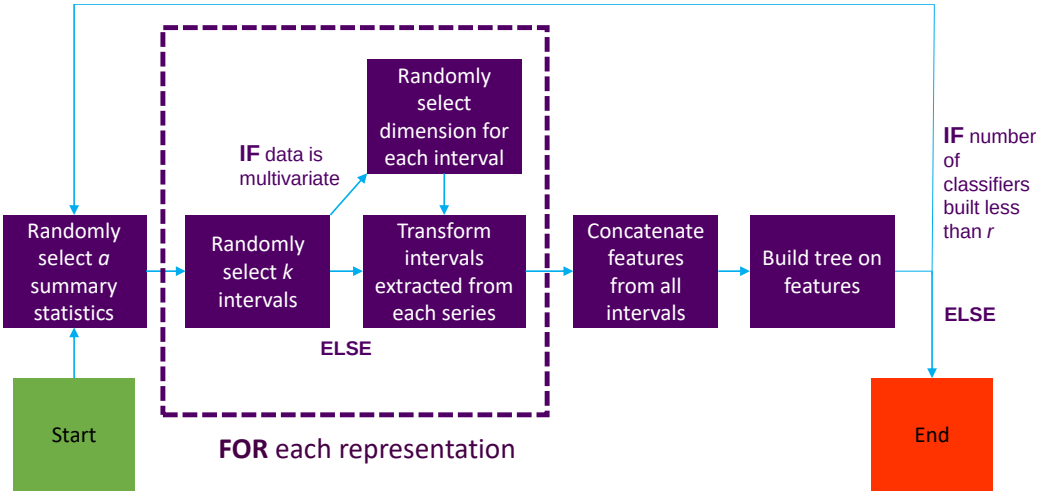


Fig. 5.2 The DrCIF ensemble. For each tree, intervals are extracted from the base series, a periodogram representation and first order differences.

the size of the base series and the differences having one less value. As such, it is likely the number of intervals selected for each representation will differ. The maximum length of an interval is set to $rm/2$ rather than the full series length to mitigate the build time increase from extracting more features. We also alter the method of random interval selection for DrCIF after an investigation with CIF to select more central intervals (see section 5.4.4). The build process for the DrCIF ensemble is described in Algorithm 5.2. A visualisation of the ensemble structure used by DrCIF is provided in Figure 5.2. Our goal with DrCIF is to replace both TSF and RISE in HIVE-COTE with an algorithm which encompasses the features of both without losing to other single representation algorithms in accuracy.

5.3 Results

Like the previous chapter, our experiments are run on 112 datasets from the univariate UCR archive [36] excluding those that are unequal length or contain missing values. The 26 equal length datasets from the UEA archive [3] are used for our multivariate experiments. Both archives and our choice of datasets are described in

Table 5.2 Classifier parameters used in our interval based experimentation.

Classifier	Parameters
CIF	500 trees, $\sqrt{m}\sqrt{d}$ intervals per tree 8 attributes subsampled per tree
DrCIF	500 trees, $4+(\sqrt{d}\sqrt{rm})/3$ intervals per representation 10 attributes subsampled per tree
Hybrid	500 trees, $\sqrt{m}$ intervals per tree
Catch22	Random Forest classifier, 500 trees
TSF	500 trees, $\sqrt{m}$ intervals per tree
RISE	500 trees
TDE	250 parameter sets sampled, 50 max ensemble size 0.7 subsample proportion, 50 initial random parameter sets
cBOSS	250 parameter sets sampled, 50 max ensemble size, 0.7 subsample proportion
WEASEL	0.1 χ feature selection threshold
STC	1 hour transform contract, Rotation Forest classifier
ROCKET	10,000 kernels, Ridge Regression classifier
PF	100 trees, 5 candidates per node
DTW_D/I	Full warping window
gRSF	100 trees, 10 shapelets per tree
HIVE-COTE 1.0	Weighting exponent of 4

Chapter 3. Our experiments run 30 seeded stratified resamples for each classifier-dataset combination. Appendix C provides the average score for over the relevant archive for each of our experiments in this chapter, with the accompanying webpage for the relevant publications [85, 87], providing more detailed and downloadable results files. Table 5.2 provides parameter configurations for the algorithms used in our experiments.

Our experiments are designed to answer the following questions.

1. Do our interval based improvements beat current interval techniques and perform on par with the best of other representations? (Section 5.3.1)
2. What is the impact on scalability for the new interval approaches, and is the interpretability of TSF retained? (Section 5.3.2)
3. Are the newly multivariate capable interval approaches competitive on the UEA datasets compared to benchmark algorithms? (Section 5.3.3)

4. How do our algorithms and other TSC approaches perform for the unequal length Asphalt case study datasets? (Section 5.3.4)

5.3.1 Interval Based Improvements

Our first experiment compares the Hybrid classifier that uses all of the Catch22 features instead of the three TSF features. This serves as a basic test of concept. Figure 5.3 presents a pairwise scatter diagram comparing the accuracies of TSF and the Hybrid. The Hybrid is significantly better than TSF. If we examine the resamples on each dataset and perform a paired sample t-test, we find that the hybrid is significantly better on 69 datasets and significantly worse on 6. There is improvement from using the Catch22 features with TSF. However, the simple usage of Catch22 presents some problems. The Hybrid is drastically slower than TSF on some problems. There are only results 109 problems in Figure 5.3 because the Hybrid failed to complete three problems within 7 days, these being HouseTwenty; EOGHorizontalSignal; and EOGVerticalSignal. CIF is designed to improve efficiency (see Section 5.3.2) but also inject further diversity.

Figure 5.4 shows the rank data for current interval techniques, CIF, DrCIF, Catch22 and the Hybrid. It demonstrates that CIF improves the accuracy and AUROC of the simple Catch22-TSF hybrid. The adjustments produce worse probabilities, however. All of our proposed methods are significantly better than TSF, RISE and the Catch22 features on all metrics. The actual accuracy improvements over TSF and Catch22 are not small. The average improvement of CIF over all data sets is 4.56% against TSF and 6.34% vs Catch22. CIF has higher accuracy on 99 of the 109 problems. The addition of multiple representations to CIF also provides an improvement, with DrCIF performing significantly better than all other methods on all metrics. While not as large as the increase from TSF to CIF, the average accuracy improvement from CIF to DrCIF is not negligible at 1.49%.

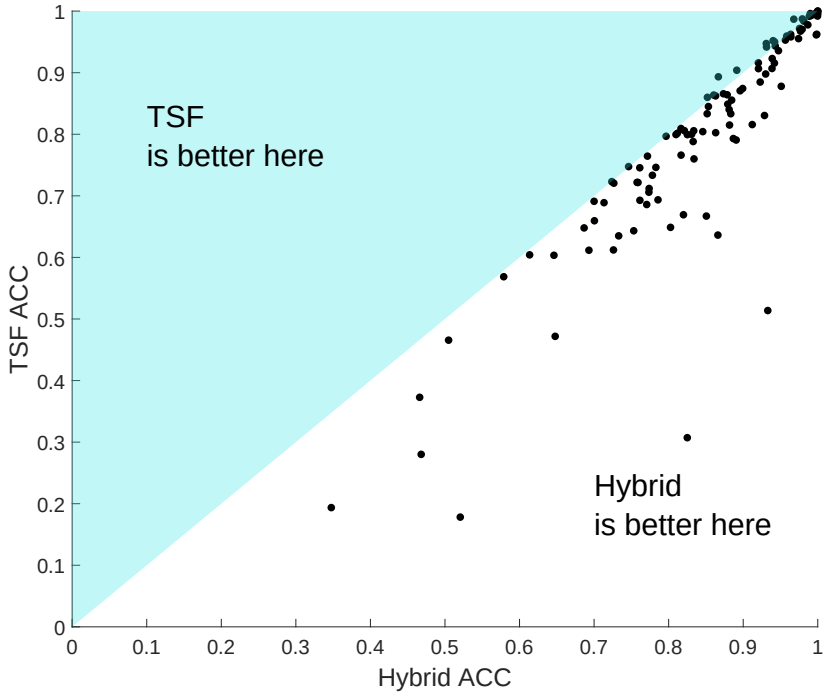
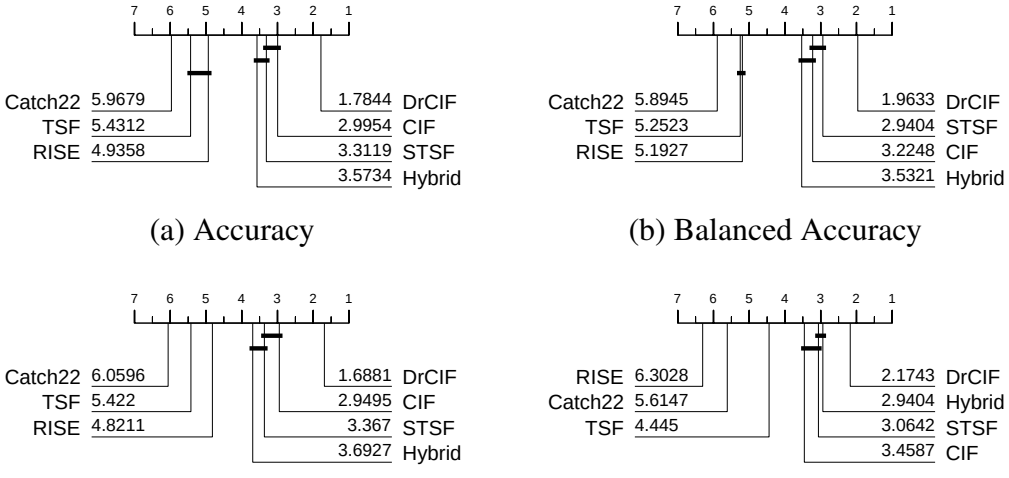


Fig. 5.3 Scatter plot of TSF vs Hybrid on 109 problems. Hybrid wins on 89 problems, ties on 2, and loses on 18 problems.



(a) Accuracy

(b) Balanced Accuracy

(c) AUROC

(d) NLL

Fig. 5.4 Accuracy rank comparison for five interval classifiers and Catch22 on 109 UCR problems. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

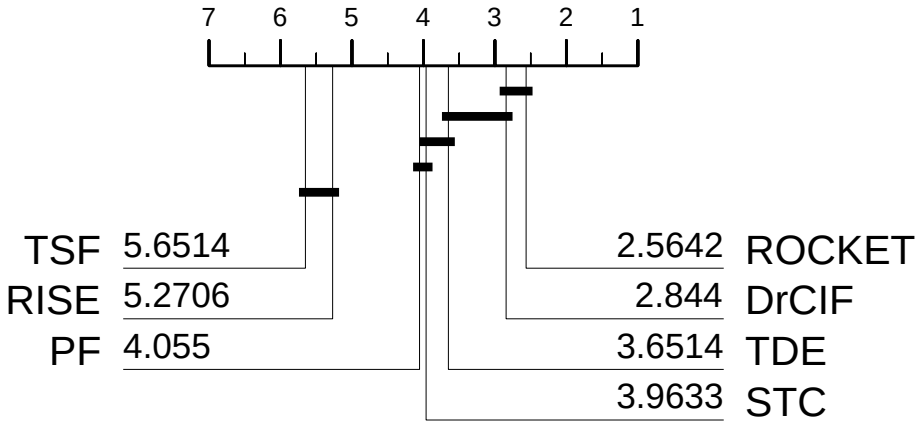


Fig. 5.5 Accuracy rank comparison for DrCIF and the latest single representation classifiers.

Figure 5.5 compares DrCIF to the latest single representation classifiers: PF; STC; TDE; and ROCKET. Three datasets failed to complete for PF, these being HandOutlines; NonInvasiveFetalECGThorax2; and NonInvasiveFetalECGThorax1. There is no significant difference between DrCIF and the state-of-the-art for dictionary and convolution based classification in TDE and ROCKET. DrCIF is significantly better than the distance based PF and shapelet based STC.

ROCKET is an accurate classifier for using a single representation, showing no significant difference to hybrid algorithms such as HC1 and TS-CHIEF [37]. We further compare DrCIF and ROCKET in a pairwise scatter diagram in Figure 5.6. Despite winning on considerably more datasets than DrCIF, the difference in average accuracy to ROCKET is relatively small at 0.2%. Looking for significant differences with a paired sample t-test, we find that ROCKET is significantly better on 49 datasets and DrCIF better on 39. We conclude that our advancements with CIF and DrCIF achieve our goal of developing an accurate interval based classifier that is at least as good as other algorithms based on a single representation.

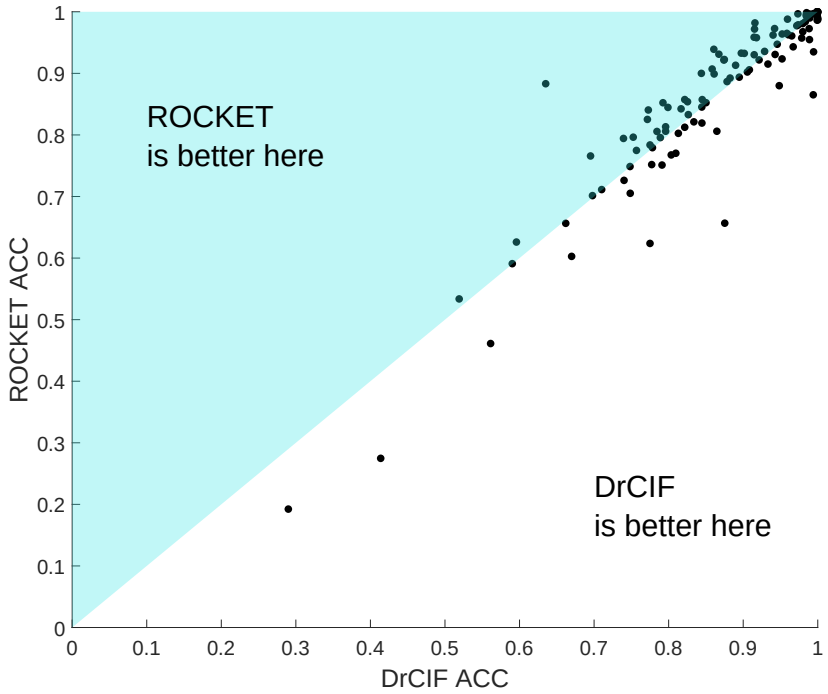


Fig. 5.6 Scatter plot of ROCKET vs DrCIF on 109 problems. DrCIF wins on 39 problems, ties on 2, and loses on 68 problems.

5.3.2 Scalability and Interpretability

CIF and DrCIF are significantly more accurate than TSF, but at what cost in terms of build time? Figure 5.7 plots the average accuracy rank against the average run time on the UCR data for Hybrid, CIF, DrCIF and 7 algorithms used in our previous comparisons. All experiments were conducted in a single thread. TSF presents as slightly slower than seen in previous comparisons, as the Random Tree base classifier used in other comparisons [8, 85] was replaced with the original TST algorithm [40]. Our original Hybrid classifier is the slowest alongside PF, both of which failed to complete 3 separate datasets. The transform portion of STC is contracted, taking longer on smaller datasets than likely required and raising the average build time for the algorithm. DrCIF is almost an order of magnitude faster than the Hybrid, and is the fastest of the top performing single representation classifiers displayed in Figure 5.5 except for ROCKET. It may be surprising that DrCIF builds faster than CIF on average, but as the Catch22 features

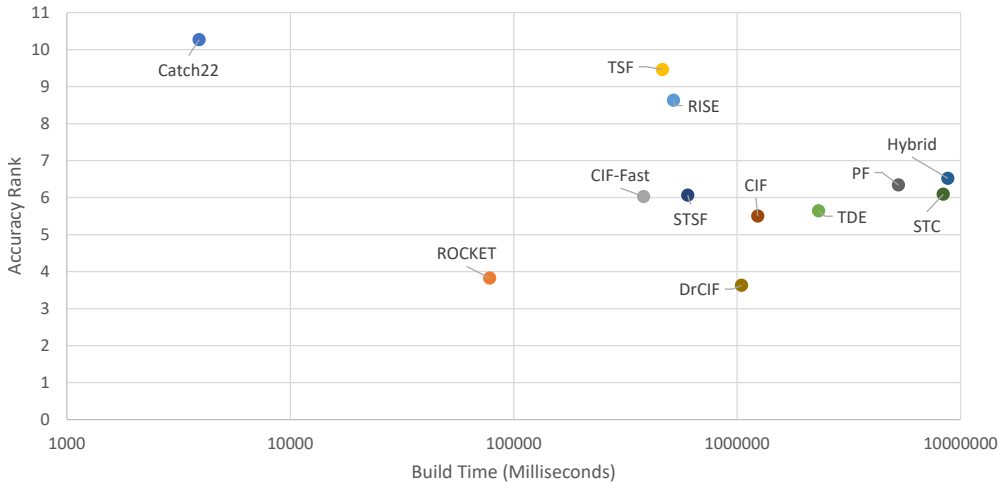


Fig. 5.7 Scatter graph plotting average accuracy rank against build time on a log scale of interval based classifiers and algorithms we compare against on 106 UCR datasets. Classifiers close to the bottom are more accurate, and those close to the left are faster.

mostly scale on the series length, reducing the max interval size can have a large impact. Additionally, on larger datasets having the number of intervals as a function of the representation length rather than the original series length can cause fewer intervals to be selected. CIF-Fast is a reduced form of CIF with fewer trees (250) and intervals per tree ($\sqrt{m}^{0.85}$). We include it to demonstrate that, if the problem is very large, CIF can be configured to be faster than current interval techniques with little loss of accuracy rankings. To further improve usability, we provide a contracted version of CIF in line with other HIVE-COTE components [17, 44, 88] that adds trees to the ensemble incrementally. The ability to set a fixed time to train until allows a degree of certainty for time sensitive training requirements, such as clusters with a maximum job time limit.

One benefit of the original TSF algorithm is that it can be used to evaluate the relevance of regions of the time series. Temporal importance curves [40] are a visualisation method that displays the importance of time series intervals to classification. To achieve this, the information gain of each node's splitting attribute is collected for all trees in the forest. Each attribute the trees are built on corresponds

to a summary feature and a time interval. A curve is created for each summary feature, adding the gain from any node split to its curve at the time points covered by its interval. We can adapt this algorithm for CIF, although displaying all 25 features can be too cluttered. We display the top $v = 3$ features by max gain over all time points, as these have effectively been discovered to be the most important, in addition to the mean of all features for each time point.

Figure 5.8 displays the CIF temporal importance curves for the EthanolLevel problem, shown previously in Figure 2.4. It demonstrates that the most important region for classification is the same region identified by a domain expert. While by default CIF subsamples features per tree, we have not done this for our example as it can have a large impact on the production of these curves. The temporal importance curves can be generated for model visualisation and are still meaningful while subsampling. However, randomly restricting access to features, and therefore promoting weaker features that would otherwise have lost to more consistently informative ones when splitting, may not give the best representation to perform a higher level analysis of a time series problem. The goals of maximising accuracy, minimising computational cost, and maximising interpretability, often run counter to each other.

5.3.3 Multivariate Interval Based Comparison

We have demonstrated CIF and DrCIF's performance on univariate data. Strong performance here does not necessarily transfer to the multivariate case, however. To demonstrate this, we compare to against a range of multivariate classifiers on the UEA multivariate archive.

DTW is still a reasonable benchmark in the multivariate case, using the different strategies for generalisation described in [115]. DTW_I assumes independence between dimensions, while DTW_D assumes dependence. The generalised random

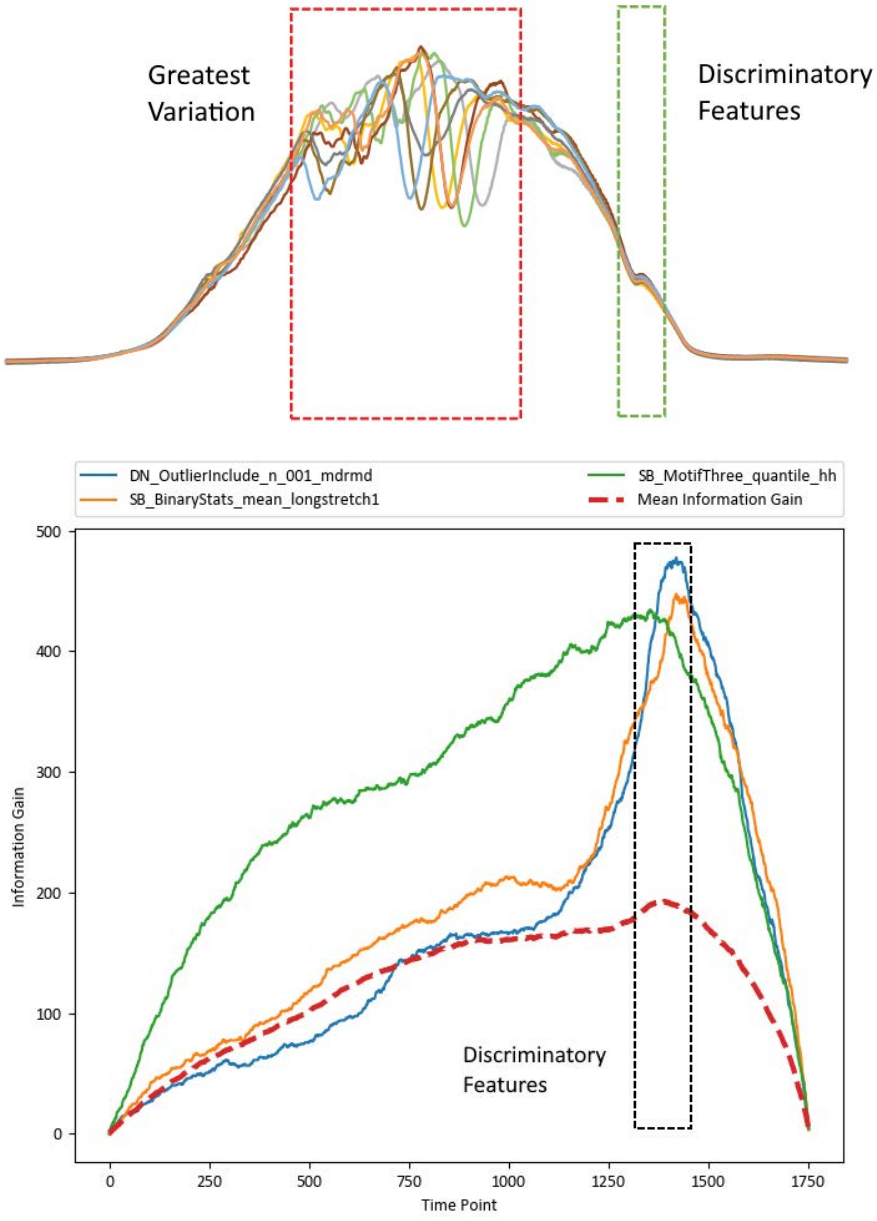
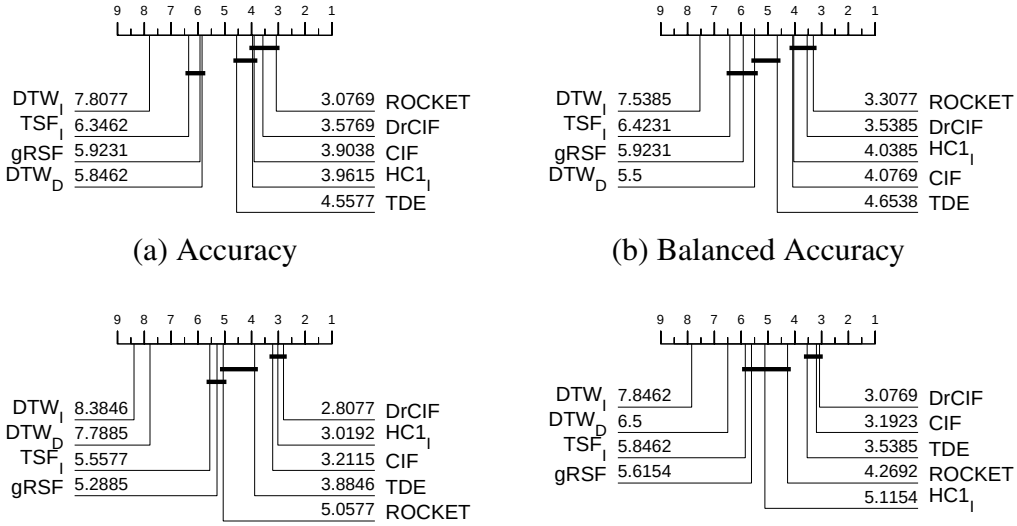


Fig. 5.8 Temporal importance curves for the EthanolLevel problem. Higher values indicate greater importance for classification. The dotted box indicate the most important region as specified by a domain expert. A previously shown visualisation of the EthanolLevel problem is shown on top of the curves produced by CIF

shapelet forest (gRSF) [61] constructs an ensemble of shapelet trees, randomly selecting the dimension each shapelet is generated from. We include our multivariate capable TDE classifier from Chapter 4. Also included are TSF and HC1. For both classifiers, a model is trained for each dimension independently, which are then ensembled with new cases being classified using majority vote. We call these TSF_I and $HC1_I$. We attempted to obtain results for two other classifiers, however, we have needed to omit them: the deep learning method TapNet [142], for which we could not satisfactorily reproduce the results for using the author's code; and WEASEL+MUSE [109], which required more than 500 GB on multiple problems.

Figure 5.9 shows critical difference diagrams for the multivariate classifiers presented using multiple metrics. TSF is significantly better than univariate DTW [8], this does not translate to the multivariate case. However, CIF and DrCIF perform significantly better than both variants of DTW and TSF. Both classifiers remain in the top clique for all metrics. Like our univariate comparison, we have shown our interval based improvements can at least match the best performers from other representations. HC1 never ranked above DrCIF, showing that even for hybrid algorithms, accuracy may not translate to the multivariate case without taking into account the relationship between dimensions.

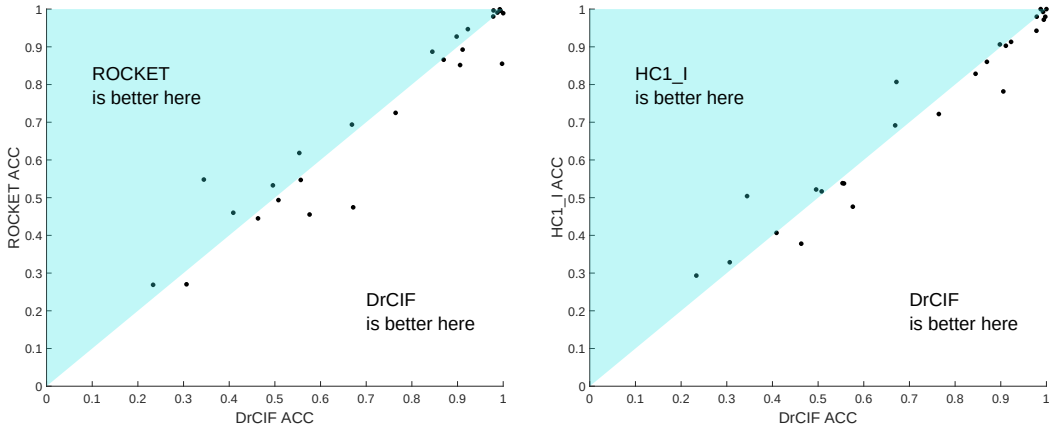
Figure 5.10 shows the relative performance of DrCIF against ROCKET and $HC1_I$. The difference in average accuracy between DrCIF and both classifiers is not large, with DrCIF performing 0.46% better against ROCKET and 0.19% better against $HC1_I$. On individual problems, there can be quite a large difference in accuracy between the approaches, however. While $HC1_I$ has a large issue with scalability, ROCKET and DrCIF provide a good baseline for future multivariate developments.



(a) Accuracy (b) Balanced Accuracy

(c) AUROC (d) NLL

Fig. 5.9 Accuracy rank comparison for five interval classifiers and Catch22 on 109 UCR problems. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).



12 / 0 / 14 WDL 14 / 1 / 11 WDL

Fig. 5.10 Scatter plot of DrCIF against ROCKET and $HC1_I$ on 26 multvariate problems. The Win/Draw/Loss of each dataset for DrCIF against both of these classifiers is shown below the figure.

5.3.4 Asphalt Dataset Case Study

We further demonstrate the usefulness of CIF by using the three Asphalt datasets first presented in [119]. The problem involves predicting the condition of a road based on motion data. Data is recorded on a smart phone installed inside a vehicle using a flexible suction holder. This offers the potential for automated monitoring and assessment of road conditions, leading to earlier and less costly interventions with faults. Two versions of the dataset are available, one univariate and the other multivariate. The series are unequal length, centred around zero, but not normalised. More detail on these datasets is provided in Chapter 3.

A number of 1-NN classifiers were used for this problem in [119], each using a range of elastic distance measures combined with Complexity Invariant Distance (CID) [10]. Many of the classifiers we test are unable to handle series of different length. A common initial strategy for dealing with unequal length series is to zero pad the data if the series is normalised, and mean pad otherwise so that all series are the same length as the longest series. As the data is centred around zero, we zero pad all cases to the longest series length of each dataset. Whilst standard, this approach can introduce features that confound classifiers, particularly if the padding is extreme, as in this case. We believe interval based classifiers such as CIF have an inherent tolerance to issues caused by padding.

Results for the three asphalt datasets are shown in Table 5.3. The best performing classifier overall is DrCIF, with ROCKET performing the best on one of the datasets. DrCIF beats HC1 on all problems, the latter potentially being dragged down by the poor performance of the RISE component. Analysis of RISE takes spectral features on a single interval per tree, which supports our belief that the multiple intervals taken by CIF and DrCIF mitigate the impact of padding.

Table 5.3 Classifier accuracy on three asphalt data sets over a 10-fold cross-validation.

	Asphalt-Regularity	Asphalt-PavementType	Asphalt-Obstacles
CIF	**0.9893**	0.9124	0.8501
DrCIF	**0.9893**	**0.9365**	0.8758
TSF	0.9734	0.8773	0.8773
RISE	0.5087	0.3889	0.7131
STSF	**0.9893**	0.9318	0.8796
TDE	0.8416	0.6201	0.8297
cBOSS	0.8216	0.6452	0.8284
WEASEL	0.9341	0.7949	0.8463
STC	0.9354	0.8224	0.8732
ROCKET	0.9521	0.8432	**0.8937**
HC1	0.9567	0.8816	0.8732

5.4 Canonical Interval Forest Analysis

While our interval improvements stand on their own right, the changes were made with the HIVE-COTE ensemble in mind. In this section, we further analyse our Hybrid classifier and CIF, looking at the impact of decisions made in creating the algorithm and at components which would impact HIVE-COTE. This analysis was run prior to the development of DrCIF, and had an influence on some inclusions made when designing the algorithm to replace TSF and RISE.

We ask the following questions, with each split into its own subsection:

1. How do the additions to CIF impact the accuracy and scalability of the algorithm? (Section 5.4.1)

2. Are all of the Catch22 features required, and which are the most expensive to process? (Section 5.4.2)

3. Why do the two 'Outlier' Catch22 features take so long to process, and how can we mitigate this? (Section 5.4.3)

4. Is there a better method for selecting random intervals than the one used in TSF and CIF? (Section 5.4.4)

5. Can we use bagging to efficiently produce train accuracy estimates for HIVE-COTE? (Section 5.4.5)

5.4.1 Canonical Interval Forest Ablation Study

A number of additions and alterations were made to from the initial Catch22 and TSF Hybrid classifier, resulting in significantly increased accuracy and time efficiency in CIF. These are the addition of the basic TSF features alongside those from Catch22, normalising the series prior to computing the two 'OutlierInclude' Catch22 features, replacing the Random Tree classifier used with the original TST base classifier from TSF, and subsampling 8 of the 25 Catch22 and TSF features per tree. We explore the impact of these algorithm changes by observing the difference in performance when each change is not present.

Figure 5.11 shows a critical difference diagram comparing the complete CIF and versions of CIF with a removed component. While none of these algorithm features create a significantly worse classifier when removed, combined they create a significantly better classifier than the Hybrid. Additionally, an improvement in accuracy was not the main intention of these changes, with the TSF features, outlier feature normalisation and attribute subsampling primarily being introduced to increase time efficiency. The swap of classifier from Random Tree to TST was made to increase the classifiers interpretability when creating temporal importance diagrams [40].

Table 5.4 shows the difference in time and accuracy from CIF for each variation. As shown, removing any algorithm feature other than the TST results in an increase in train time, with attribute subsampling and outlier normalisation having the most impact. Attribute subsampling accounts for the largest accuracy gain of the features,

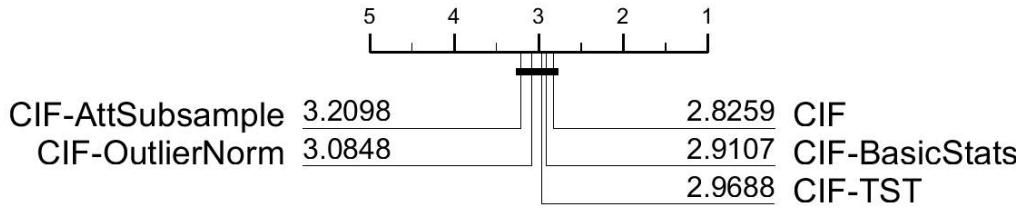


Fig. 5.11 Critical difference diagram for rankings by classification accuracy of CIF and variations with a single feature removed.

Table 5.4 Absolute difference from the CIF average accuracy of 0.84870 and relative difference from the average train time 1791 seconds for CIF variations.

	Accuracy Difference	Time difference (%)
CIF-BasicStats	-0.00024	110.64%
CIF-TST	0.00256	96.38%
CIF-OutlierNorm	-0.00001	504.45%
CIF-AttSubsample	-0.00314	255.82%

suggesting a lack of classifier diversity in the ensemble without it. While the addition of the mean, standard deviation and slope from TSF doesn't initially seem like it would save time, they are comparatively cheap to calculate and in combination with attribute subsampling can replace some of the more expensive Catch22 features in a number of trees. While the gain in time and accuracy for adding these features is minor, doing so adds negligible complexity to the algorithm.

Using TST instead of a Random Tree increases train time by a relatively small amount, but more noticeably causes a drop in average accuracy contrary to the increase in average rank shown. A pairwise scatter diagram comparing CIF with and without TST is shown in Figure 5.12. Three points stand out as having a noticeable difference in accuracy relative to other points, these being datasets recently introduced into the UCR archive, PigArtPressure; PigCVP; and PigAirwayPressure. These datasets consist of 52 classes and 104 training cases, with each class having 2 cases. This is very small relative to other datasets in the archive, and further investigation would be required to determine if this lack of data is the cause of this

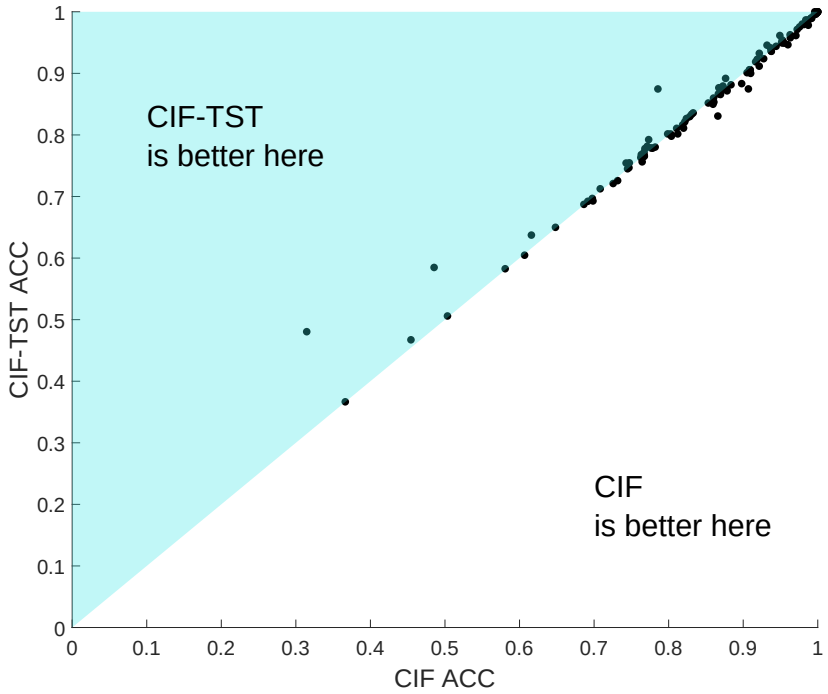


Fig. 5.12 Pairwise scatter diagram for CIF and CIF built using a Random Tree instead of the TST (CIF-TST).

drop in accuracy or if it is a characteristic of the data these problems were derived from.

5.4.2 Catch22 Feature Importance and Scalability

One of the earlier experiments we ran was an evaluation of the importance and transformation cost for each of the Catch22 features to determine if all 22 are necessary. To determine the importance of a Catch22 feature, we look at the impact of removing it from the TSF and Catch22 Hybrid using a subsample of 21 UCR archive datasets. Figure 5.13 shows a critical difference diagram displaying the accuracy rankings for each Hybrid classifier missing a single Catch22 feature. Of all Catch22 features, only 3 can be removed without a significant loss in accuracy. There are DN_HistogramMode_5 (hybrid-01), CO_FirstMin_ac (hybrid-07), and DN_OutlierInclude_n_001_mdrmd (hybrid-05). The feature with the largest impact when removed is FC_LocalSimple_mean1 _tauresrat (hybrid-17). Removing any

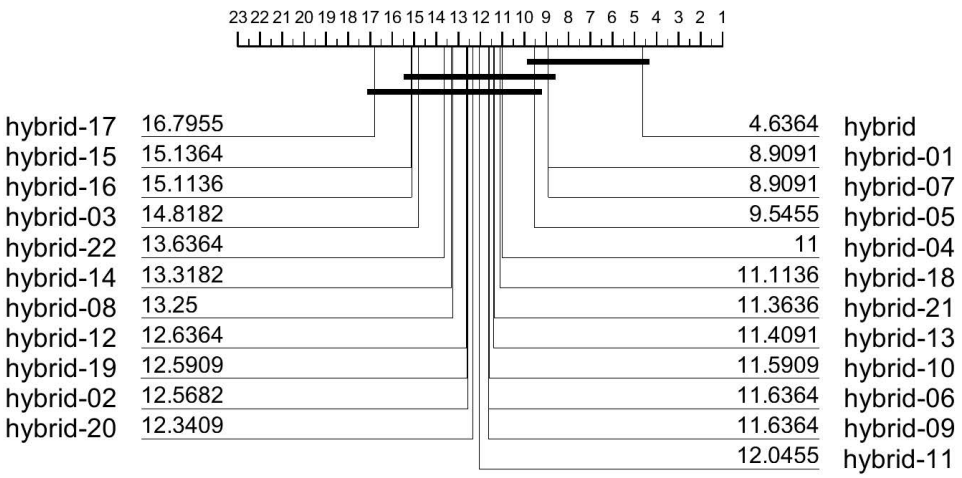


Fig. 5.13 Critical difference diagram for rankings by classification accuracy of the hybrid classifier and variations removing a single catch22 feature.

single feature results in an average accuracy decrease of 0.5% to 0.8% from the full 22. A table containing the datasets used and the full names of other features for each hybrid classifier is provided in Appendix C.

Figure 5.14 shows the transformation time for each Catch22 feature on 100 simulated time series, averaged over 50 folds for differing series lengths. The two noticeably poor scaling features are PD_PeriodicityWang_th0_01 which shows the largest increase in transform time with series length and FC_LocalSimple_mean1_tauresrat which shows large step increases of transform time at certain series length thresholds. While these would be the first contenders for removal to improve the classifiers train time, we have already shown doing so will produce a significantly worse Hybrid classifier on our subsample. Of the features which removing does not cause a significant difference, CO_FirstMin_ac and DN_OutlierInclude_n_001_mdrmd are relatively costly, reaching 4th and 8th slowest respectively at a series length of 10,000. Removing both of these would most likely also result in a significantly worse classifier, however.

We were surprised at the importance of each individual Catch22 feature to the predictive power of the feature set. One of the primary goals of the Catch22 feature

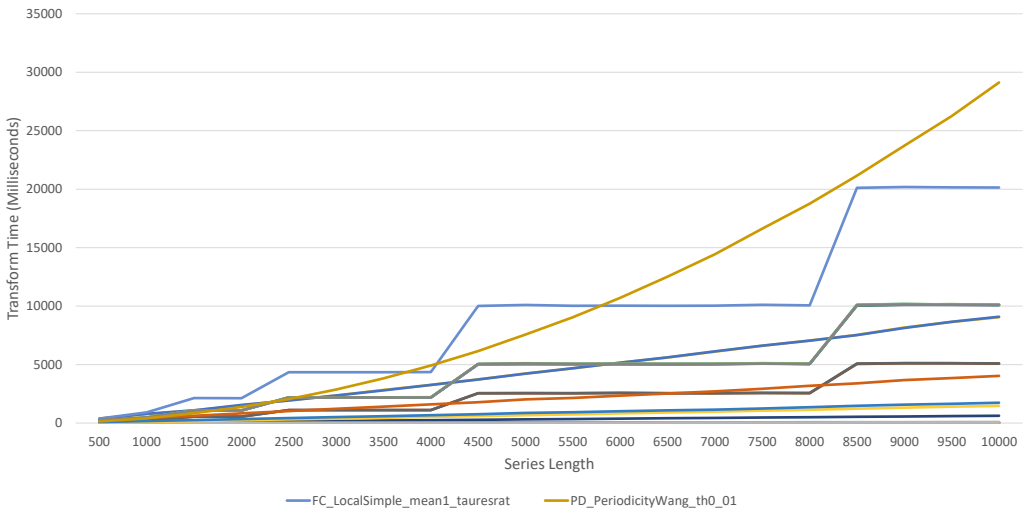


Fig. 5.14 Transform time for each catch22 feature for varying series lengths.

set was minimal redundancy between features, our experiment suggests this has been achieved. With the introduction of attribute subsampling to CIF, the impact of removing whole features on the time efficiency of the classifier is highly reduced, as the total amount of features processed will remain the same.

5.4.3 Interval Normalisation Methods

During our initial experimentation, two Catch22 features took noticeably longer than other Catch22 features to transform on non-normalised datasets. These are DN_OutlierInclude_p_001_mdrmd and DN_OutlierInclude_n_001_mdrmd, looking for the time intervals between positive and negative outliers respectively. Figure 5.15 shows the transform time for the Catch22 features on the non-normalised UCR archive datasets, averaged over 50 folds. While some datasets have little effect on timing, those with large positive or negative values see a dramatic increase in transform time for the outlier features. This increase makes the time spent transforming the remaining 20 features miniscule in comparison.

The easiest way of mitigating this is to normalise the series, removing any large values. The Catch22 features were designed for normalised data, and the process

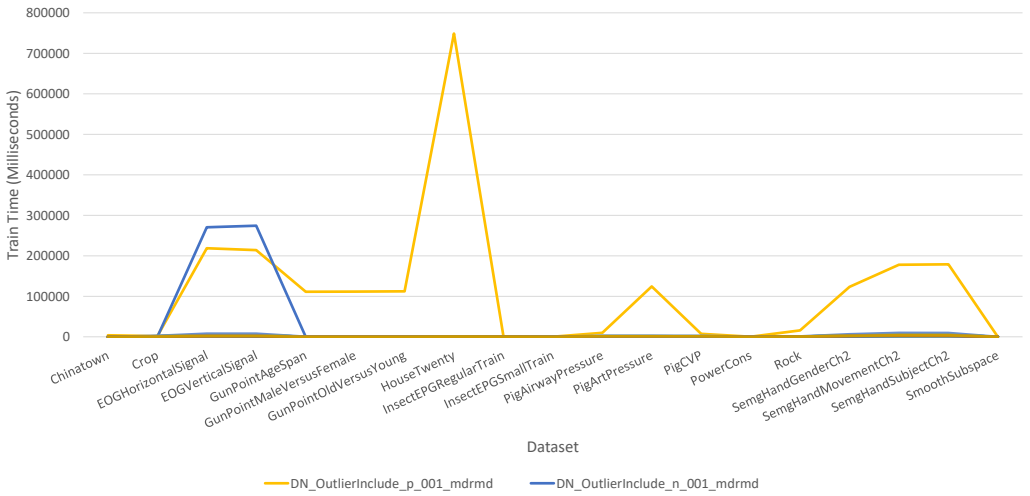


Fig. 5.15 Transform time for each catch22 feature for non-normalised UCR datasets.

used to extract them from the full set of hctsa only used normalised data. This dramatic increase in train time with unnormalised data is a quirk of this decision. Figure 5.16 shows the effect of normalising each series (Hybrid-PreNorm), normalising each interval (Hybrid-Norm) and normalising each interval for the outlier statistics only (Hybrid-Outlier) for the Hybrid classifier. We use the previous 21 datasets subsample and all unnormalised UCR datasets. Like our previous experiments, the Hybrid with no normalisation was unable to complete HouseTwenty and both EOG datasets within 7 days. Only the outlier normalisation is not significantly worse than the base Hybrid. While the Catch22 features were selected using normalised data only, they can still extract useful data from unnormalised datasets where scale contains meaningful information. As such, we use the outlier normalisation in CIF to reduce transform time while retaining performance. The issue with Catch22 and normalisation was also a driver for adding basic summary statistics which can draw information from scale to CIF and DrCIF.

5.4.4 Random Interval Selection

CIF selects a number of time intervals for each tree it builds, randomly selecting the position and length for all of them. In this section, we evaluate five different

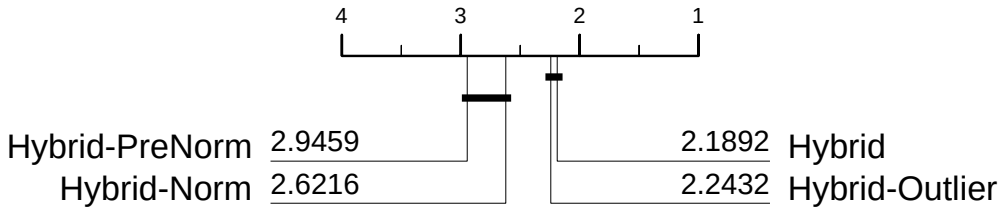


Fig. 5.16 Critical difference diagram for rankings by classification accuracy of the hybrid classifier using normalisation at varying points in the algorithm on the 37 UCR datasets.

methods for selecting the length and position. We give each interval selection (IS) method an ID of 1 through 5, with each doing the following:

- **IS1** selects the start position for the interval then interval length, resulting in bias to time points later in the series.
- **IS2** selects the end position for the interval then interval length, resulting in bias to time points earlier in the series.
- **IS3** randomly selects between IS1 and IS2 with equal weighting.
- **IS4** selects the middle position for the interval then interval length, resulting in larger intervals and better coverage for start and end of the series.
- **IS5** select middle position then interval length, limiting length to the difference between the point and the beginning or end of the series (randomly chosen). This results in shorter length intervals, but retains the better start and end coverage of IS4.

CIF by default uses IS1, as a holdover from the *tsml* TSF implementation which CIF was based on. This is unlikely to be an intentional decision, rather a quirk of the code used to randomly select intervals. To better show the effect of each interval selection method, we show the number of intervals covering each time point in Figure 5.17 for 10,000 randomly generated intervals for a series of length 1000.

Table 5.5 Mean start position, mean length and mean end position for 10000 randomly generated intervals using different methods of interval selection.

	Mean Start Position	Mean Length	Mean End Position
IS1	497.0009	250.7424	747.7433
IS2	253.1084	250.8719	503.9803
IS3	375.1403	251.2190	626.3593
IS4	287.7338	416.9260	704.6598
IS5	389.8472	220.3479	610.1951

The mean start position, length and end positions for the intervals produced by each method are shown in Table 5.5. Each method has an impact on the average length and position of the intervals.

Figure 5.18 displays a critical difference diagram for these different interval selection methods. The two methods which bias interval selection towards the beginning and end of the series, including the one currently used by CIF, are the worst performers of the methods. The other three form the top clique, with no significant difference in accuracy. While this is the case, IS4 has a noticeably larger mean interval length than the other options, leading to more required computation. This leaves IS3 and IS5, with IS3 being slightly more accurate and simpler. For our DrCIF improvements, we replace the IS1 interval selection method with IS3.

5.4.5 Accuracy Estimate Generation

HIVE-COTE requires an estimate of the test accuracy from the train data for its weighting scheme. Currently in HC1, these estimates for all constituent classifiers are found through ten-fold cross-validation. This extra time overhead is one of the major disadvantages of HIVE-COTE, since it increases the run time of the base classifiers by almost an order of magnitude. Hence, we investigate other ways of estimating the test accuracy for CIF.

One alternative for CIF is to mimic Random Forest [18] and use bagging. This means the out-of-bag (OOB) error can be used for the weight for HIVE-COTE.

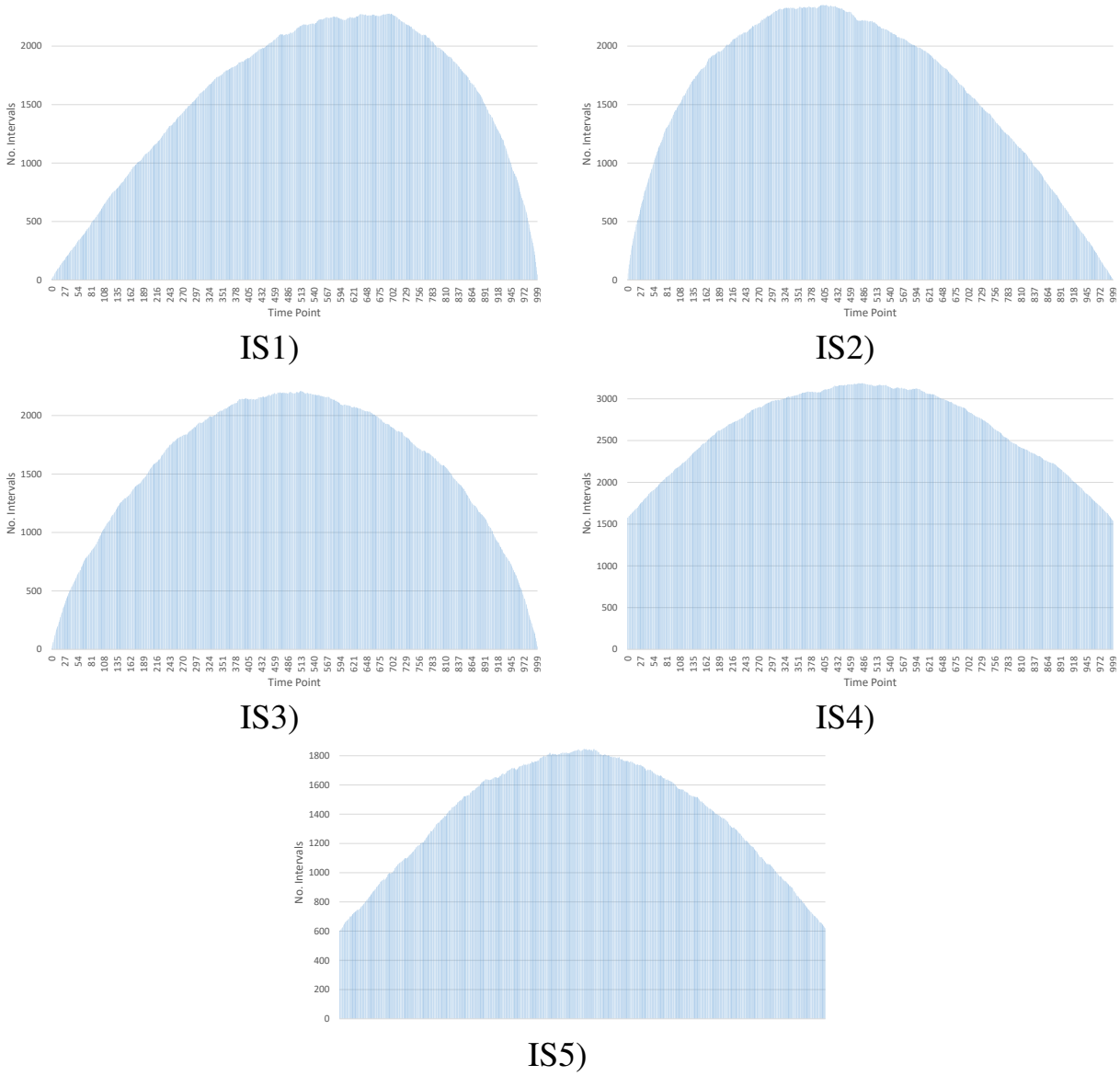


Fig. 5.17 Number of intervals covering each time point for 10000 randomly generated intervals using different methods of interval selection.

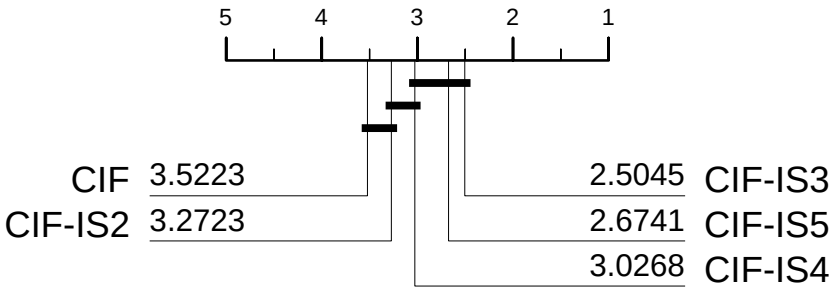


Fig. 5.18 Critical difference diagram for rankings by classification accuracy of CIF using different methods of interval selection. CIF by default uses IS1.

Table 5.6 Performance in estimating test accuracy by various means for CIF, averaged over 112 UCR datasets.

	CV+Full	Bagging	Bagging+Full
Actual Test Acc	0.8461	0.8346	0.8461
Estimated Test Acc	0.8175	0.8210	0.8210
Difference	2.86%	1.37%	2.52%

Unfortunately, our experience with bagging for other tree based ensembles for time series data has shown it makes the final classifier significantly worse than building on the whole train data. If this is also the case for CIF, it would mean we want to build the final model on the full data for testing. The question is whether there is any difference using OOB estimates or cross-validation estimates, and if OOB estimates are better, can the bagging model be used for test data?

We experiment with three methods for generating an accuracy estimate. The first is to perform a ten-fold cross-validation. This will build 11 models total, one for each fold and the final model on all data. The second introduces bagging to CIF, only one model is built and OOB error is used for our estimate. The third builds two models, one with bagging and one without. For this method, we use the bagging model to produce our estimate, and uses the non-bagging model to make new predictions. Table 5.6 shows the difference in actual test accuracy and that predicted from the train data. Cross-validation underestimates test accuracy by 2.86% on average. Bagging has lower error when predicting the test accuracy of a bagged classifier, but the actual test accuracy is significantly lower. Our compromise of using bagging to estimate test accuracy but another full model without bagging to predict new cases would us to achieve lower estimation error than cross-validation while only having to build a single additional model instead of ten.

5.5 Conclusion

Our contribution in this chapter is to propose two new interval based time series classifiers, CIF, that combines the best elements of TSF and Catch22 in a novel way, and DrCIF, which extends CIF to include multiple series representations. Both CIF and DrCIF are significantly more accurate than TSF, RISE and Catch22. DrCIF is not significantly worse than the best other classifiers built on a single representation (TDE, STC, PF and ROCKET) and is generally faster than them. When experimenting with the multivariate case, we show that simply randomly selecting the dimension of each interval in the ensemble puts both CIF and DrCIF on par with top multivariate approaches and performs significantly better than the DTW benchmark.

We show that we can replicate the interpretability of TSF by creating temporal importance curve diagrams with CIF, however with increasing complexity from multiple representations from DrCIF or multiple dimensions in multivariate data, plotting these curves becomes increasingly difficult. Fundamentally the information is still there, with information gain being available for each time point, feature, representation and dimension combination. Further work is required to determine whether this can be displayed in an easily interpretable manner.

In this chapter and the previous, we have significantly improved the flaws of the dictionary and interval based HIVE-COTE components. In the next chapter, we look to improve the HIVE-COTE ensemble itself using our enhanced algorithms from both chapters.

Chapter 6

HIVE-COTE 2.0: an Improved Meta Ensemble for Time Series Classification

Contributing Publications

- Middlehurst, M., Large, J., Flynn, M., Lines, J., Bostrom, A. & Bagnall, A. (2021). HIVE-COTE 2.0: a new meta ensemble for time series classification. *Machine Learning* 110(3), 3211–3243

6.1 Introduction

In the previous two chapters, we improved the dictionary and interval based representations of the original Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) [78] algorithm. Two of these algorithms, the Temporal Dictionary Ensemble (TDE) and the Diverse Representation Canonical Interval Forest (DrCIF) [85, 87], perform significantly better than current algorithms extracting similar features. In our opinion, it is worthwhile researching new algorithms based

on single representations. Some data will be best approached by one representation based on the domain specific problem definition. However, it has been shown that in the absence of domain specific knowledge, hybrid algorithms from multiple representations will perform better on average than those that extract only a single feature type [8, 78, 114]. The development of our individual representation algorithms was done with the end goal of improving the hybrid algorithm HIVE-COTE.

The original version of the heterogeneous meta ensemble HIVE-COTE was first introduced in 2016 [77, 78] and, at the time, was significantly more accurate on average than other known approaches [8] on the 85 datasets that were then the complete UCR archive [36]. We summarise the HIVE-COTE ensemble here, with further detail provided in Chapter 2. HIVE-COTE contains five constituent ensembles that each worked on features from different data transformation domains: the Elastic Ensemble (EE) [76]; Shapelet Transform (ST-HESCA) [58]; Time Series Forest (TSF) [40]; Bag of Symbolic-Fourier-Approximation Symbols (BOSS) [105]; and the Random Interval Spectral Ensemble (RISE) that was introduced alongside HIVE-COTE [78]. Each module was encapsulated and built on the train data independently of the others. For new data, each module passes an estimate of class probabilities to the control unit, which combines them to form a single prediction. It does this by weighting the probabilities of each module by an estimate of its testing accuracy formed from the training data.

The goal of HIVE-COTE was to achieve the highest level of accuracy without concern for computational resources. This initial target has since led to a perception that HIVE-COTE is very slow and does not scale well. While this is true if it is used in its basic form, which we will refer to as HIVE-COTE alpha (HCα), it is very simple to restructure HIVE-COTE alpha to achieve a more scalable ensemble with the same level of accuracy. A streamlined version of the ensemble, HIVE-COTE 1.0 (HC1) [5], was introduced to demonstrate its utility and scalability. HC1 is based on simple refinements and enchantments to the original HCα base constituents. As a

result, HC1 was shown to be faster to build than competitor hybrid algorithms such as TS-CHIEF [5] and the original HCα with no significant difference in accuracy.

In 2021, there were four algorithms with reasonable claim to being state of the art for TSC, based on experimentation on the recently expanded UCR archive [36]. These are: the deep learning approach called InceptionTime [43]; the tree based Time Series Combination of Heterogeneous and Integrated Embedding Forest (TS-CHIEF) [114]; the Random Convolutional Kernel Transform (ROCKET) [37]; and HIVE-COTE. In this chapter, we propose a new version of HIVE-COTE that is significantly more accurate than all four current state-of-the-art algorithms. We call this classifier HIVE-COTE 2.0, or HC2 for short [87].

The key principle behind HIVE-COTE is that TSC problems are best approached by careful consideration of the data representation, and that with no expert knowledge to the contrary, the most accurate algorithm design is to ensemble classifiers built on different representations. The changes from HC1 to HC2 relate to the component classifiers and a redefinition of the underlying data representations used. HC2 contains four component classifiers: the dictionary based TDE; the interval based DrCIF; an adaptation of convolution based ROCKET we call the Arsenal; and the latest version of STC [17, 5]. Each of these classifiers represents the best in class for a particular representation.

We structure the remaining sections of this chapter as follows. Section 6.2 introduces the HC2 algorithm and its constituent classifiers. Section 6.3 presents an evaluation of HC2 against the current state-of-the-art and its individual components. We further analyse HC2 and consider alternate ensemble structures in Section 6.4. Section 6.5 explores the usability of HC2, making use of its contracting functionality for large datasets. We conclude the chapter in Section 6.6.

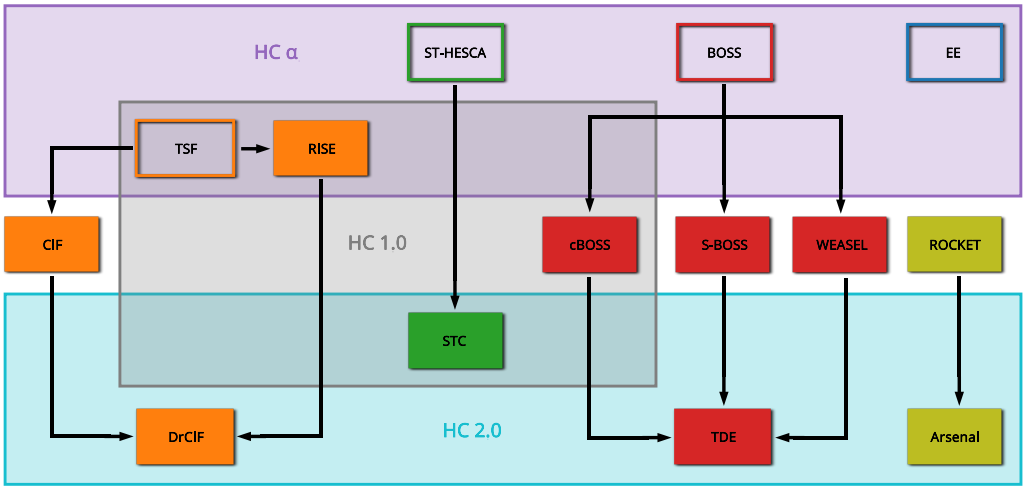


Fig. 6.1 A history of the classifiers included in the three HIVE-COTE ensembles and the relationship between them. Filled classifiers were introduced after the 2017 bake off comparison [8].

6.2 HIVE-COTE 2.0 (HC2) [87]

Like the HIVE-COTE versions before it, HC2 is a heterogenous ensemble of time series algorithms from different feature representations. HC2 retains the use of the Cross-validation Accuracy Weighted Probabilistic Ensemble (CAWPE) [68] as its ensemble method, weighting each algorithm using an estimate of its testing accuracy found using training data. HC2 replaces three of the four classifiers that make up HC1. The component modules are: the shapelet based STC [17]; the convolution based ensemble of ROCKET classifiers we call the Arsenal; the dictionary based representation TDE; and the interval based DrCIF. Like HC1, the distance based representation is left out, as the algorithms either perform poorly in comparison to current techniques or are too slow to justify inclusion. A history of algorithms making up each version of HIVE-COTE is provided in Figure 6.1, and an overview of the updated HC2 structure is displayed in Figure 6.2.

Each component is trained independently and, in addition to the final model, it is required to produce an estimate of its accuracy on unseen data. For new data, each module produces a probability estimate for each class. The controller constructs

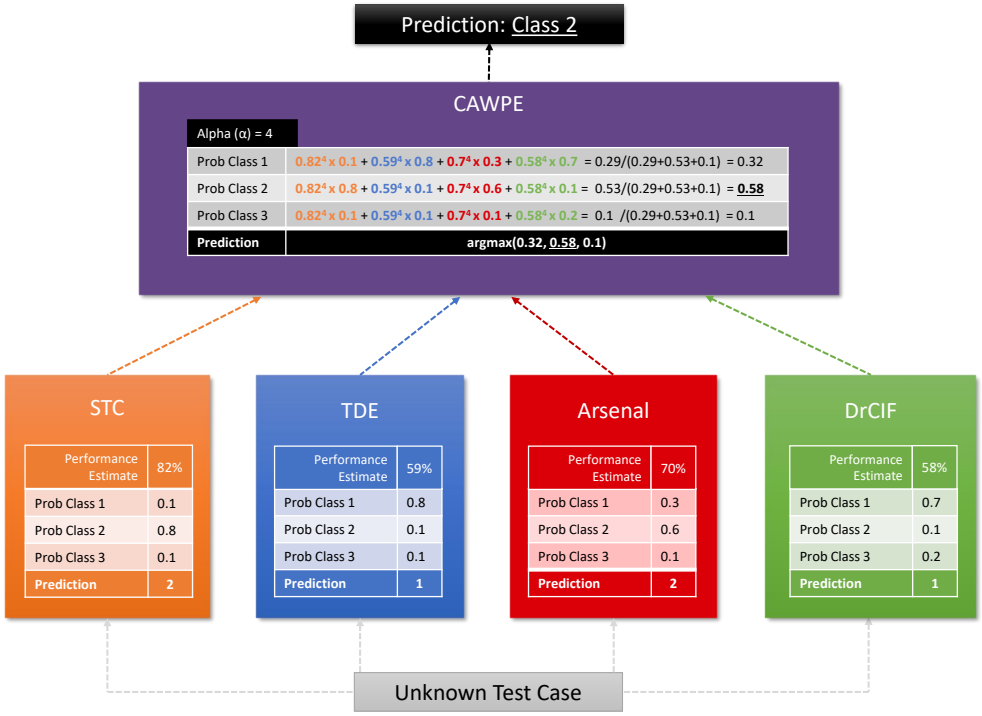


Fig. 6.2 An overview of the ensemble structure of HIVE-COTE 2.0 for a three class problem. Each module is trained independently and produces an estimate of the probability of membership of each class for unseen data. The control unit (CAWPE) combines these probabilities, weighted by an estimate of the quality of the module found on the train data.

a tilted distribution through exponentiation to extenuate differences in classifiers and weighting with the accuracy estimate. For classifier n in the ensemble, the classifiers weight is simply its accuracy estimate to the power of α.

$$Weight_n = {acc_n}^{\alpha} \tag{6.1}$$

Given that the accuracy is scaled from 0 to 1, this results in classifiers being more harshly penalised in the ensemble the worse they perform on the training data. This exponent in weighting is used to greatly improve the impact of feature type(s) which can find information in the input series, while lowering the impact of those which fail to find much or any information. We keep the default value for α of 4 used in [68].

Each module of HC2 contains new features and improvements over previous versions. These include novel algorithm improvements, multivariate extensions and contracting improvements. In addition, the method for estimating the accuracy from the train data has been improved based on experimentation in Section 5.4.5. Generally, there are three ways of estimating test accuracy from train data. Firstly, the final model can be assessed directly on the train data. This is likely to be biased and over optimistic, particularly if some form of model selection has occurred without careful regularisation. Secondly, a cross-validation can be performed on the train data in addition to the final build. Whilst this will probably be less biased (and generally pessimistic) it is time-consuming. Thirdly, some form of hold-out evaluation can be embedded in the full model build, such as bagging. HC1 uses a mixture of cross-validation approaches based on the classifier. HC2 adopts a standardised bagging approach, which is possible since all four classifiers in HC2 use ensembles. However, in Chapter 5 we found that whilst using out-of-bag performance produces acceptable estimates of test accuracy, the bagged classifiers themselves were significantly less accurate than those built on the full data. Hence, we adopted a hybrid approach. Rather than build 11 total models for ten-fold cross-validation or a single model using bagging, we construct one bagged model to estimate accuracy for those that require it using out-of-bag error and a full model to predict new cases. Specifics on how each module generates its estimate and the impact of this design choice are discussed in Section 6.4.

HC2 can train each component concurrently. Even so, the components can be slow on problems which have numbers of cases or series lengths much larger than those seen in the UCR datasets, such as the use cases we introduce in Chapter 3. Hence, we allow the user to configure HC2 so that it has a time contract. If contracted, each component ensemble simply builds as many base classifiers as it can in the time provided. This simple form of contracting is an adequate first fix. However, a problem does arise for very large data with short contracts: building

a single ensemble member may exceed the contract. It would be better for the components to self configure when this is likely, by, for example, subsampling cases or series.

Two of our components, DrCIF and TDE, have been introduced and described in detail in the previous two chapters of this thesis. STC is essentially unchanged from the HC1, and still uses a one-hour transform contract with the Rotation Forest classifier. Further details for the STC algorithm can be found in Chapter 2. The last component is the Arsenal, based on the ROCKET classifier.

6.2.1 The Arsenal: A ROCKET Ensemble

The Random Convolutional Kernel Transform (ROCKET) [37] uses a large number of randomly parameterised convolution kernels applied to each case. As each kernel is applied to a series, the max value and proportion of positive values are recorded and concatenated into a feature vector. These features are then used to build a linear ridge regression classifier with built-in cross-validation to select the alpha parameter.

For each kernel generated, the parameters are selected from the following spaces: The length, l, is selected such that, $l \in \{7,9,11\}$; the value of each weight, w_i, is randomly sampled from a normal distribution $\sim \mathcal{N}(0,1)$, and are then mean centered; bias b is sampled from a uniform distribution $\sim \mathcal{U}(-1,1)$; dilation, a, is sampled from an exponential scale up to series length; the binary decision to pad the series p is chosen with equal probability, if true the series is zero padded at the start and end equally such that middle element of the kernel is applied to every point in the input series. Stride is always set to 1. For multivariate datasets, each kernel is assigned a random number of randomly selected dimensions. The kernel for the multivariate case is still one dimensional, but with weighting being different

for each dimension. The max and proportion of positive values is calculated across all selected dimensions.

ROCKET is a very fast classifier that has state-of-the-art accuracy, and we believe it is the most important recent development in the field. It represents a different class of approach, and as such is a candidate for assimilation into the collective. However, an issue arises when trying to include ROCKET in HIVE-COTE: the ridge regressor used by ROCKET is hard to configure to produce useful probability values for each class when making predictions. The CAWPE ensemble structure of HIVE-COTE uses weighted probabilities, and relies on classifiers to produce a distribution representative of the classifiers strength of belief in predictions. One solution would be to replace the ridge regressor with a classifier that does produce representative probability estimates. However, our experimentation with suitable replacement classifiers did not yield a candidate algorithm that was as accurate as the ridge regressor for ROCKET.

To solve this problem, the version of ROCKET we use in HIVE-COTE is an ensemble of smaller ROCKET classifiers. We refer to this fusillade of ROCKETs as the Arsenal. New cases are classified using the CAWPE exponential weighted majority vote, with the weights obtained from the ridge regression classifiers cross-validation. Arsenal is slower to build than ROCKET, but its improved probabilities make it a better candidate for HC2. The build process for Arsenal is described in Algorithm 6.1.

6.3 Results

We perform our univariate time series experiments on 112 of the 128 datasets from the UCR time series archive [36]. We exclude datasets containing series of unequal length or missing values, as we do not want an algorithm's aptitude for these cases to alter results and most implementations are not set up to handle these kinds of

Algorithm 6.1 Arsenal(A list of n cases of length m with d dimensions, $\boldsymbol{T} = (\boldsymbol{X}, \boldsymbol{y})$)

Parameters: the ensemble size, r and the number of kernels per classifier, k

Let $\boldsymbol{f}$ be a list of ROCKET classifiers $(f_1, \ldots, f_r)$
for $i \leftarrow 1$ to r **do**
  Let $\boldsymbol{X'}$ be a n by $2k$ list of transformed cases
  for $j \leftarrow 1$ to k **do**
    $[l, w, b, a, p, O] \leftarrow randomKernelParameters(m, d)$
    for $t \leftarrow 1$ to n **do**
      $max \leftarrow -\infty$
      $ppv \leftarrow 0$
      for $g \leftarrow 1$ to $(m + (2p)) - ((l-1)a)$ **do**
        $v \leftarrow 0$
        for $c \leftarrow 1$ to $|O|$ **do**
          $v \leftarrow v + applyKernel(\mathbf{X}_t, g, O_c, l, w, b, a)$
        if $v > max$ **then**
          $max \leftarrow v$
        if $v > 0$ **then**
          $ppv \leftarrow ppv + 1$
      $\boldsymbol{X'}_{t,2j-1} \leftarrow max$
      $\boldsymbol{X'}_{t,2j} \leftarrow ppv/((m + (2p)) - ((l-1)a))$
  $f_i.buildRidgeClassifierCV([\boldsymbol{X'}, \boldsymbol{y}])$

data. For our multivariate experiments, we use all 26 equal length datasets of the 30 total from the UEA multivariate time series archive [3]. For each dataset, we present performance as an average over 30 resamples. Both archives provide a default split into train and test sets, which we use for the first resample. The remaining 29 are randomly resampled from the original split in a stratified manner. We seed each classifier and data resample using the fold index to ensure our results are reproducible. We present summarised results, with full tables and links to our raw results files available in Appendix D. Further description of our experimental process is provided in Chapter 3.

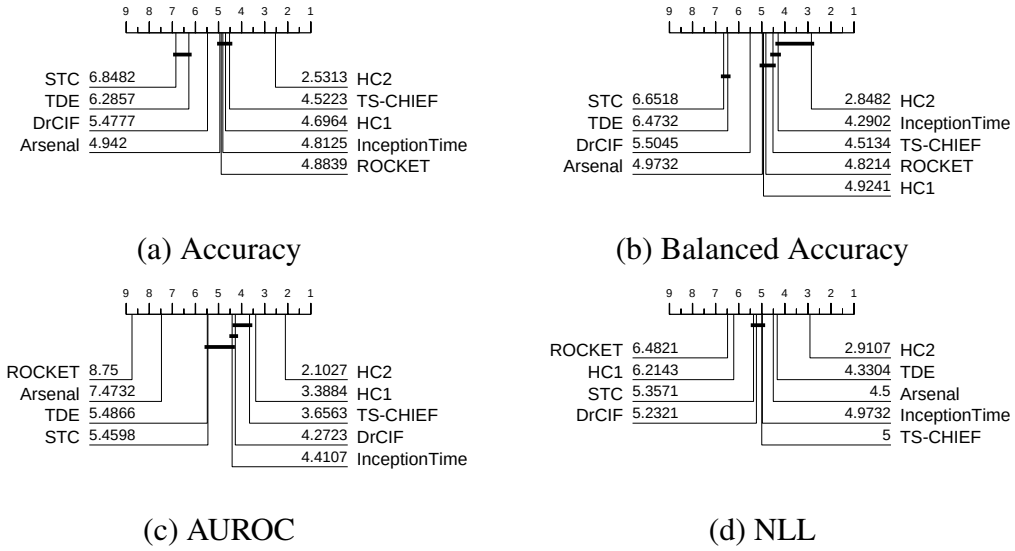
All of our non-deep learning experiments were run using the Java `tsml` toolkit implementations. For deep learning approaches, we use Python `sktime` companion package `sktime-dl`. The configuration for each algorithm is provided in Table 6.1.

Table 6.1 Classifier configurations for our experiments where m is the series length, d is the number of dimensions and rm is the lengths of DrCIF representations.

Classifier	Parameters
TDE	250 parameter sets sampled, 50 max ensemble size 0.7 subsample proportion, 50 initial random parameter sets
DrCIF	500 trees, $4+(\sqrt{d}\sqrt{rm})/3$ intervals per representation 10 attributes subsampled per tree
STC	1 hour transform contract, Rotation Forest classifier
Arsenal	2,000 kernels per classifier, 25 ensemble size
ROCKET	10,000 kernels, Ridge Regression classifier
HIVE-COTE 1.0	Weighting exponent of 4
HIVE-COTE 2.0	Weighting exponent of 4
TS-CHIEF	500 trees, 5 EE splitters, 100 RISE splitters, 100 BOSS splitters
InceptionTime	Epochs 1500, batch size 64, learning rate $1e-3$ and halved after no improvement for 50 epochs. Two residual blocks, each with three Inception modules with kernel sizes per module [10, 20, 40] plus bottleneck filters for all conv layers of 32
CIF	500 trees, $\sqrt{m}\sqrt{d}$ intervals per tree 8 attributes subsampled per tree
DTW_D	Full warping window

6.3.1 Performance on 112 Univariate TSC Problems

Our core result is that HC2 is significantly better than the current state of the art on the 112 UCR equal length datasets. Figure 6.3 shows the critical difference diagrams for accuracy, balanced accuracy, NLL and AUROC for HC2, its four components and the four state-of-the-art algorithms. They demonstrate that HC2 is significantly better than its components and the four benchmarks using three of the four metrics. For accuracy, STC and TDE form the lowest clique. DrCIF is between these and the current state of the art, which contains Arsenal. ROCKET is the worst for probability estimates, since it only produces 0/1 estimates. The better probability estimates of Arsenal justify our design decision, since HC2 weights distributions, not predictions. TDE is surprisingly good at probabilistic prediction. The poor probability estimates of HC1 highlights one source of improvement in HC2. HC1 does much better at AUROC, whereas Arsenal does much worse, indicating further



(a) Accuracy

(b) Balanced Accuracy

(c) AUROC

(d) NLL

Fig. 6.3 Critical difference diagrams for nine classifiers over 112 UCR problems. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

calibration may benefit the Arsenal. HC2 and InceptionTime show no significant difference using balanced accuracy. As InceptionTime is also significantly better than ROCKET and HC1 in this metric, it is that the algorithm is better than the non-deep learning approaches at dealing with unbalanced classes.

Figure 6.4 shows the accuracy scatter plots of HC2 against each of the baseline classifiers, and Table 6.2 summarises the differences in test accuracy between HC2 and the four baselines. We observe that there is lower variance between HC1 and HC2, but that HC2 consistently outperforms HC1 with an average accuracy of more than 1%. The variation in difference to HC2 is greater with the other three classifiers, in particular ROCKET. The median difference is lower than the mean in all cases. This suggests skew, which supports the core hypothesis that the heterogenous ensemble can compensate for the shortcomings of its components. It also suggests that HC2 has a higher representational power, in that it can find a more diverse set of features.

Accuracy is not the only consideration. Table 6.3 summarises the run time and memory requirements for the classifiers compared in Figure 6.4. There are a few

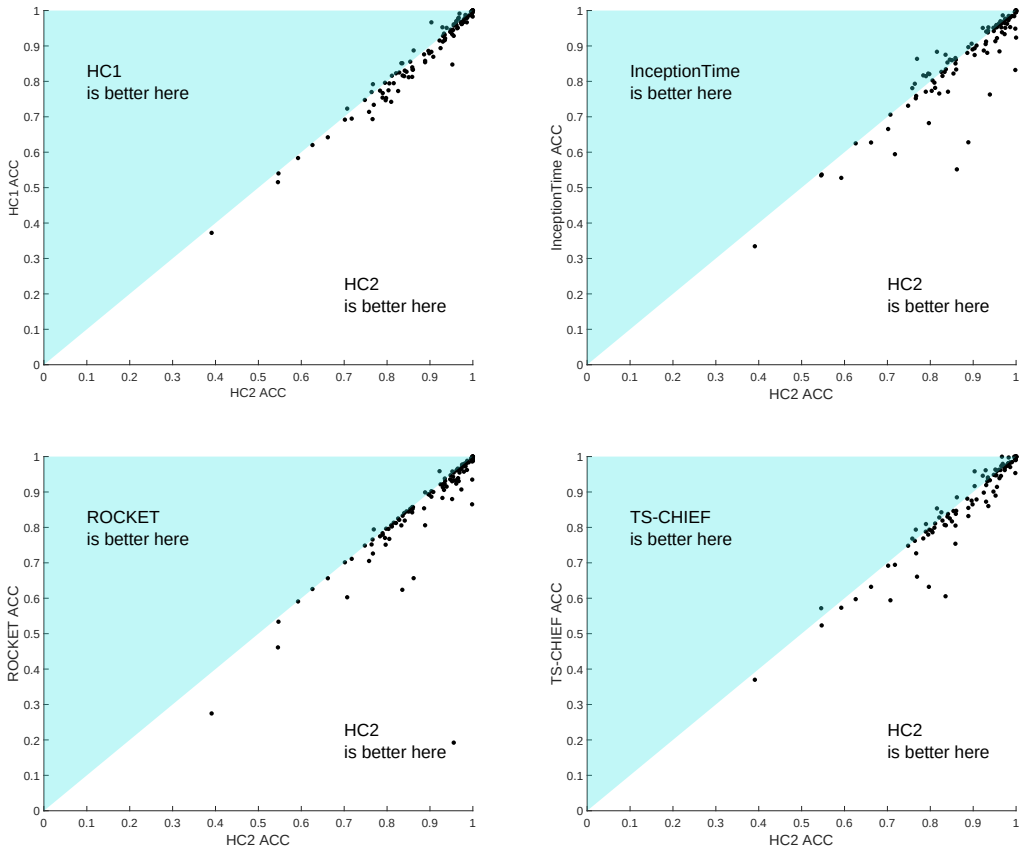


Fig. 6.4 Accuracy scatter plots of HIVE-COTE 2.0 against each of the baseline classifiers.

caveats to these results. Firstly, all of the results except InceptionTime are run in a single thread on a CPU. Thus, the InceptionTime time experiments are not really directly comparable, since it runs on a GPU. ROCKET and HC2 are forced to run in a single thread, despite being threadable. The times for the HC2 components are without the time to estimate performance, but these are included in the HC2 times. Memory is the maximum memory used throughout the run, as obtained from the Java garbage collector, and should be considered approximate. We are not set up to measure the maximum memory used with InceptionTime in practice, but we know it did not exceed 12 GB, because that was the memory available on the GPU. Since the run times are sequential, we also use the sequential memory for HC1 and HC2.

Table 6.2 Summary of the differences between HC2 and the benchmarks accuracy. A negative value means the HC2 is better.

	TS-CHIEF	InceptionTime	HC1	ROCKET
Mean	-1.36%	-1.69%	-1.06%	-2.49%
Median	-0.41%	-0.37%	-0.69%	-0.72%
Max	-22.99%	-31.04%	-10.47%	-76.31%
Min	5.50%	9.46%	6.33%	3.64%
StDev	3.64%	5.13%	2.10%	7.92%
HC2 Wins	77	74	82	97
HC2 Loses	29	32	25	11

Table 6.3 Run time and memory requirements to train single resample of 112 UCR problems. The median of 30 runs is taken for each dataset.

Algorithm	Total (hrs)	Average (mins)	Max Mem (MB)
ROCKET	**2.85**	**1.53**	4349
Arsenal	27.91	14.95	1683
DrCIF	45.40	24.32	**920**
TDE	75.41	40.40	6565
InceptionTime	86.58	46.38	-
STC	115.88	62.08	4219
HC2	340.21	182.26	6677
HC1	427.18	228.84	4876
TS-CHIEF	1016.87	544.75	26052

These would be higher if the classifier were threaded, but of course the run time would be much lower.

With this in mind, we can make the following observations. ROCKET lives up to its name and can build models on all 112 data in under 3 hours, even when not threaded. If speed is the main criteria, ROCKET is a good starting point in any analysis. STC is the slowest component, but this is caused by the configuration rather than an inherent problem: STC defaults to a one hour shapelet search or a full evaluation of the shapelet search if this will take less than an hour. For the very small problems, it takes a lot longer than the other algorithms (although still less than an hour). HC2 is faster than HC1, primarily because of improvements to estimating accuracy and the change in classifiers. TS-CHIEF is the slowest algorithm by far, and seems to scale less well than the others. On the slowest five problems

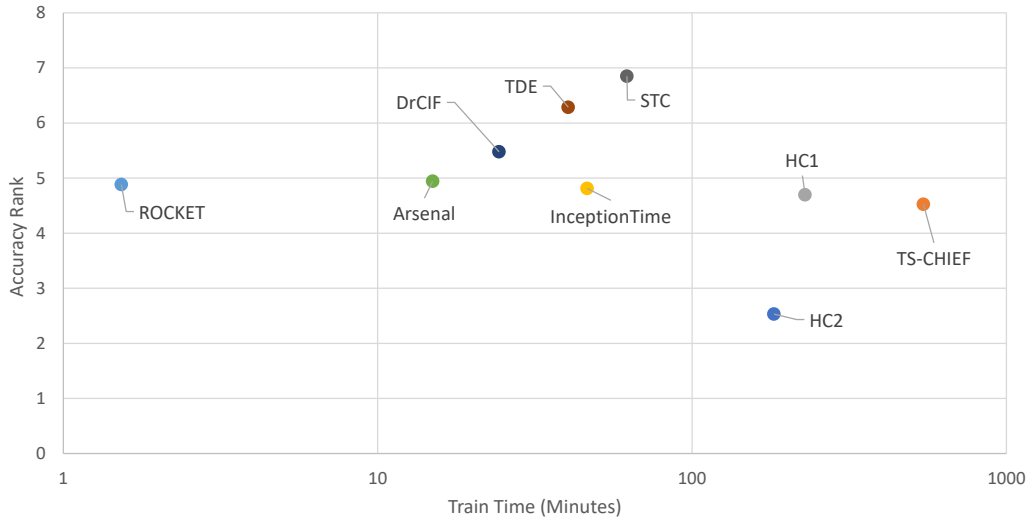
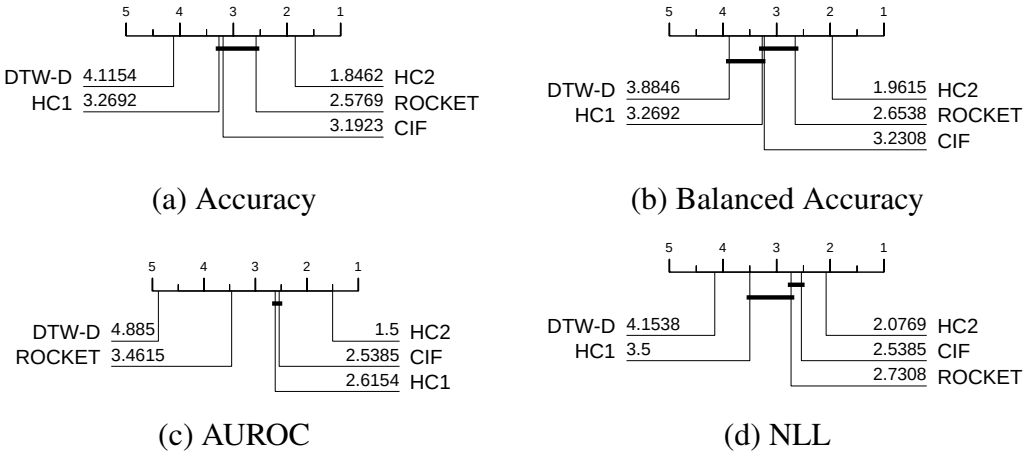


Fig. 6.5 A comparison of classifiers in terms of accuracy rank and train time. The time and accuracy are averaged over 112 UCR problems. The train time is on a log scale.

(HandOutlines, NonInvasiveFetalECGThorax1 and 2, SemgHandMovementCh2 and EthanolLevel), it takes ten times longer than HC2, but the difference is minimal on smaller problems. All of the classifiers are within reasonable bounds for memory. TS-CHIEF has the highest memory requirement, with a max requirement of 26 GB on HandOutlines. As with run time, it seems to scale worse than the others. HC2 requires more memory than HC1, but it is not unreasonable. ROCKET has a worse max memory case (ElectricDevices) than Arsenal. Overall, ROCKET tends to use less memory than Arsenal but appears to scale worse for larger datasets with many cases. Arsenal uses a smaller amount of kernels for each individual classifier, meaning that each transformed set of data is smaller in size and discarded before the next is built. ROCKET on the other hand must transform using its larger amount of kernels at a single point. Figure 6.5 summarises the accuracy and run time results by plotting the log of the train time against the rank.



(a) Accuracy (b) Balanced Accuracy

(c) AUROC (d) NLL

Fig. 6.6 Critical difference diagrams for five classifiers over 26 UEA MTSC problems. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

6.3.2 Performance on 26 Multivariate TSC Problems

A comparison of TSC algorithms using the MTSC UEA archive [104] and Chapter 5 of this thesis found three algorithms (that could complete all 26 data sets and are not part of the HC2 ensemble) were significantly more accurate than DTW-D. These algorithms are ROCKET, HC1 (build independently on each dimension) and CIF. We have repeated these experiments with HC2. While we evaluated both DrCIF and TDE on the 112 univariate datasets prior to their inclusion in HC2, none of the HC2 constituent classifiers were evaluated on any of the datasets used in this experiment with the exception of STC, which performed relatively poorly. Multivariate experiments in previous chapters were performed after the development of HC2, to reduce bias from selecting components based on their individual test accuracy performance. Figure 6.6 show that HC2 is significantly better than DTW-D, ROCKET, HC1 and CIF on the 26 datasets for accuracy, balanced accuracy, NLL and AUROC.

Figure 6.7 shows the accuracy scatter plots, and Table 6.4 summarises the differences of HC2 against the benchmarks. We think these results strongly support

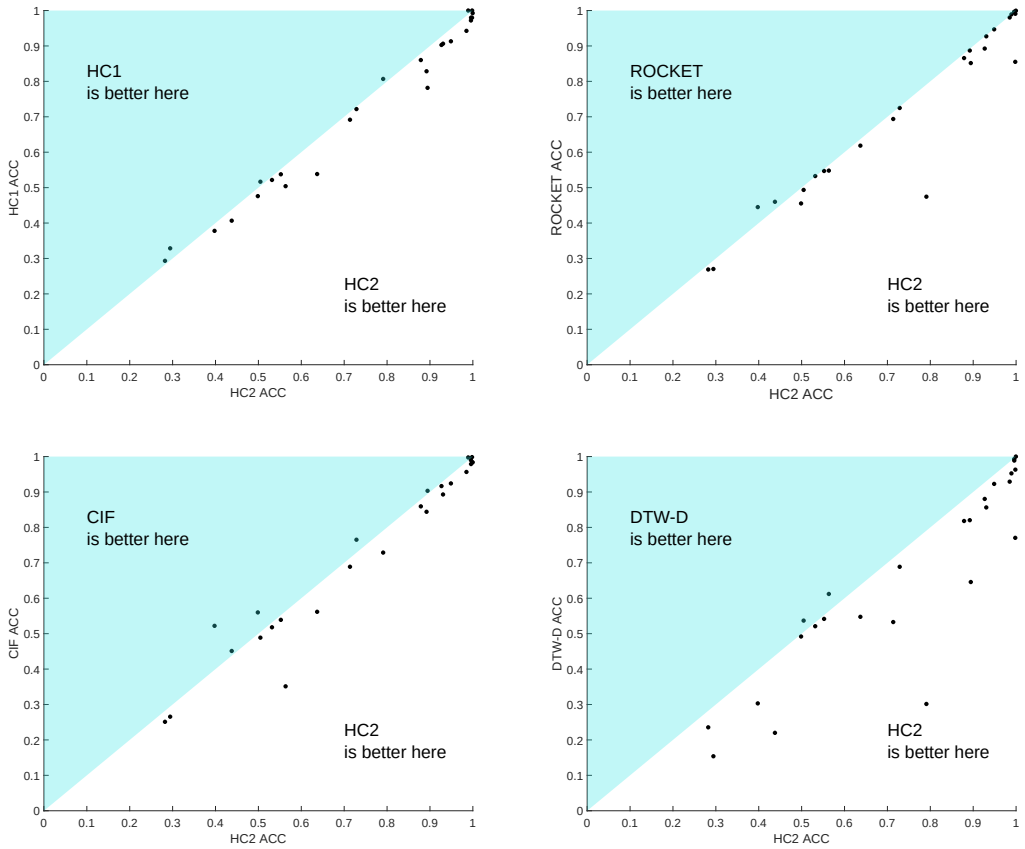


Fig. 6.7 Scatter plots of HIVE-COTE 2.0 against each of the baseline classifiers

the assertion that HC2 represents a new state of the art for multivariate time series classification.

6.4 Analysis

In this section, we address the question of why HC2 works so well, and evaluate design decisions made in the change from HC1 to HC2. We ask the following questions:

1. Are all of the HC2 components necessary for its performance, and how much accuracy is lost from removing each from the ensemble? (Section 6.4.1)

Table 6.4 Summary of the differences between HC2 and the benchmarks. A negative value means the HC2 is better.

	ROCKET	CIF	HC1	DTW-D
Mean	-2.52%	-1.71%	-2.25%	-8.22%
Median	-0.64%	-1.58%	-1.92%	-4.66%
Max	-31.65%	-21.21%	-11.27%	-48.94%
Min	4.73%	12.43%	3.44%	4.87%
StDev	6.75%	5.56%	3.30%	11.39%
HC2 Wins	19	19	20	23
HC2 Loses	5	7	6	3

2. Does the use of out-of-bag error for HC2 accuracy estimates introduce any issues, and how does it compare to cross-validation? (Section 6.4.2)

3. With our alterations to HIVE-COTE, could a new ensemble structure lead to better results? (Section 6.4.3)

4. Is the ensemble of ROCKET classifiers, the Arsenal, worth using over ROCKET itself? (Section 6.4.4)

6.4.1 Ablation Study

HC2 is significantly better than other algorithms using our selected metrics, but are all of its algorithms necessary for this performance? While encompassing multiple representations is how we achieve our high accuracy, it is possible that not all of the included representations are required. This was shown with HC1, as removing the distance based module in EE resulted in only a small drop in accuracy for a large gain in scalability [5]. Our first issue is to quantify what impact each component has on the overall performance. Ignoring single component variants (which are presented in Figure 6.3), there are 11 possible combinations, identified as HC-1 to HC-10 in Table 6.5, with the eleventh being the Full HC2, referred to as just HC2 elsewhere.

Table 6.5 Possible variants of HC2 components.

	DrCIF	Arsenal	STC	TDE
HC-1	X	X		
HC-2	X		X	
HC-3	X			X
HC-4		X	X	
HC-5		X		X
HC-6			X	X
HC-7	X	X	X	
HC-8	X	X		X
HC-9	X		X	X
HC-10		X	X	X

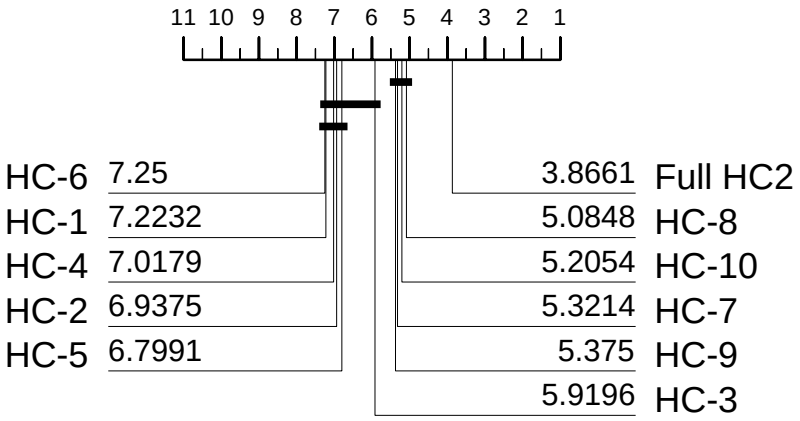


Fig. 6.8 Critical difference diagram for 11 variants of HIVE-COTE 2.0 described in Table 6.5. Full HC2 contains all four components and is referred to as simply HC2 elsewhere.

Figure 6.8 shows the relative performance of the 11 possible variants. The two component models (HC-1 to HC-6) form a clear clique, followed by another clique of three component versions. However, the full four component classifier is significantly more accurate than all of the other variants. Removing any component results in a 0.3% reduction in average accuracy at minimum (removing DrCIF in HC-10). This demonstrates that each element contributes to the overall whole.

Table 6.6 Summary of the difference between estimated and observed test accuracy for HC2 and its components. A positive figure means that the classifier is overestimating accuracy from the train data.

Classifier	Mean	Median	Min	Max	MSE
DrCIF	-2.15%	-0.93%	-46.78%	9.64%	0.40%
Arsenal	-1.17%	-0.40%	-23.13%	9.23%	0.15%
STC	1.24%	0.78%	-55.54%	26.88%	0.86%
TDE	-1.14%	-0.77%	-18.89%	10.02%	0.14%
HC2	0.47%	0.11%	-19.81%	19.17%	0.13%

6.4.2 Out-of-bag Accuracy Estimates

HC1 uses cross-validation to estimate the test accuracy from the train data for each component. HC2 modules are all ensembles, and so it was natural to attempt to use bagging and the resulting out-of-bag error estimate to speed up HIVE-COTE training. However, whilst this produces good estimates of the test accuracy, the models were less accurate on unseen data for every module. Hence, we made the decision to fit a separate bagging model for the estimation stage for those that need it, thus providing an order of magnitude speed up compared to cross-validation. DrCIF and Arsenal both create separate models with bagging to generate their estimates. STC builds a Rotation Forest model with bagging for its estimate, but uses the same transformed shapelet data for both. TDE naturally takes a 70% subsample when creating its ensemble, as such a new model is not required to generate its out-of-bag error. However, we were concerned that these estimates may be biased and/or not consistent. Table 6.6 summarises the distributions of the differences between estimated and observed test accuracy for HC2 and its components.

Whilst there is small bias for each component, HC2 ensemble method compensates for this and has the lowest average deviation (and MSE deviation) between estimated and observed test accuracy. This is due to the averaging ensemble effect, and the biasing effect of reusing estimates from the components: a full nested

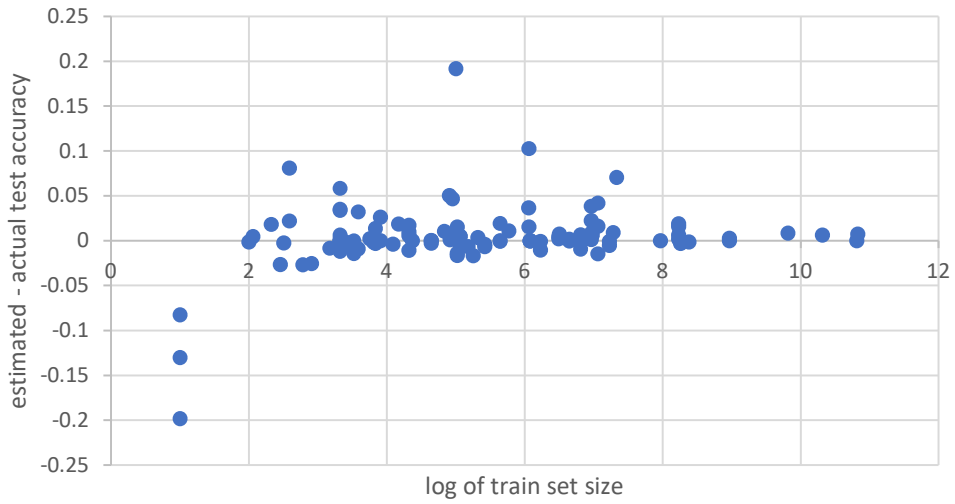


Fig. 6.9 Difference in estimated and observed test set accuracy against the log of the train set size for 112 UCR datasets.

cross-validation estimate would be computationally demanding and is not necessary. STC is the only component that is over optimistic. This is to be expected. STC performs a random search on the whole train data, then bags rotation forest. This introduces bias, and is a possible area for future improvement. The min and the max show that there are some very large differences between estimate and observed. These primarily arise in problems where there are very few cases per class, such as PigAirwayPressure, PigCVP and PigArtPressure, which each have only two cases per class. Every classifier underestimates the test accuracy by over 10% on these problems. Figure 6.9 shows the difference in the test accuracy estimate and actual plotted against the log of the train set size for HC2. The picture is not conclusive, but it could be argued that the variance of the difference is decreasing, which is encouraging evidence for the consistency of the HC2 estimate.

Another benefit of accurate estimates from the train data is that they can be used to compare classifiers with a Texas Sharpshooter plot [10]. These compare two classifiers by comparing the ratio of estimates from the train data with those of the test data to form a kind of contingency table. Computing train estimates through

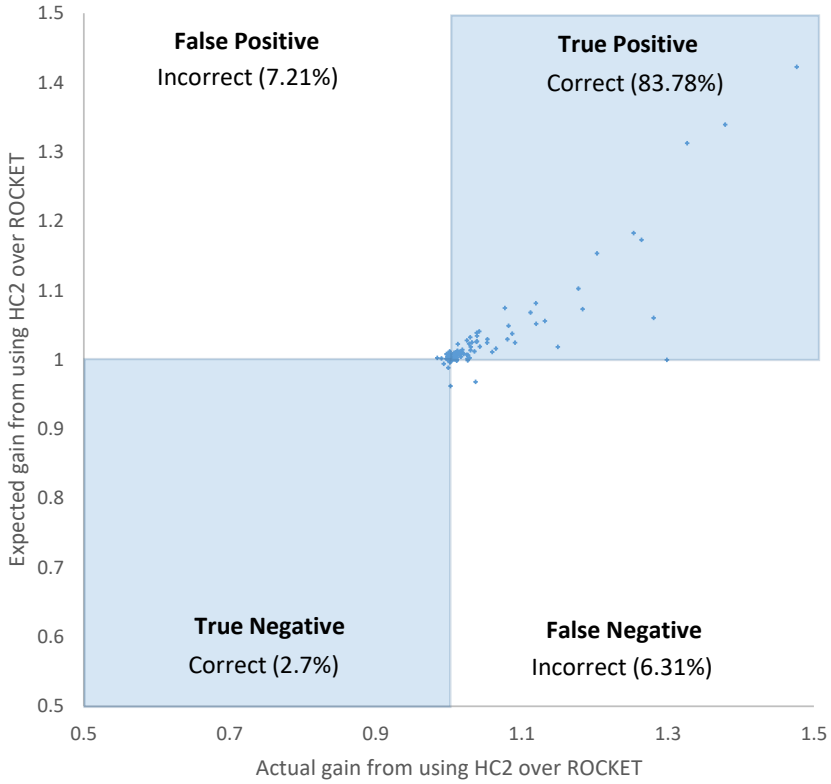


Fig. 6.10 Texas sharpshooter plot for HC2 vs ROCKET. Each point represents a single dataset. The x-axis is the ratio of HC2 and ROCKET actual test accuracy, and the y-axis is the ratio of predicted test accuracy.

cross-validation for TS-CHIEF and InceptionTime is unpractical due to run times. However, it is easy with ROCKET, since it is so fast. Figure 6.10 shows the plot for ROCKET vs HC2. Using the train estimates would lead to the correct decision of choosing HC2 on 94 of the 112 datasets.

Figure 6.11 compares four versions of HC2 using different methods for generating train accuracy estimates. HC2-oob is the version we compare in section 6.3, using out-of-bag error estimates for all components. HC2-cv uses cross-validation to generate the train accuracy estimate, configured so that the data for each classifier is only transformed a single time to improve efficiency. HC2-fastest uses the fastest method for generating estimates, using out-of-bag error for all components but TDE, which obtains leave-one-out cross-validation estimates through its building process. Lastly, HC2-closest uses the component with the average train accuracy estimate

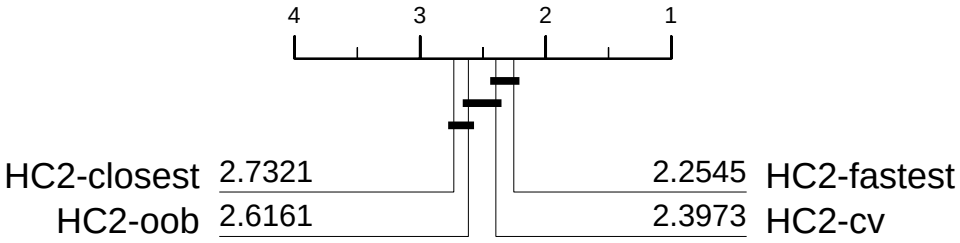


Fig. 6.11 Critical difference diagram for variants of HC2 built using different methods for extracting its accuracy estimate. HC2-oob is referred to as simply HC2 elsewhere.

closest to the average test accuracy of the component, consisting of out-of-bag estimates from TDE and STC and cross-validation estimates from DrCIF and Arsenal. We recognise that the final variant would be biased towards the UCR datasets, but include it for context. There is no significant difference between out-of-bag and cross-validation estimates. However, using out-of-bag estimates is considerably faster than cross-validation at 342 hours to classify the 112 UCR datasets against 651 hours. The difference in speed from HC2-fastest to HC2-oob is only a few hours, but using cross-validation for TDE only ranks significantly better than with out-of-bag error.

6.4.3 Ensembling Methods

In extending HC1 into HC2, the two main factors are what representations should be included in the ensemble, and how the predictions drawn from each representation should be combined. Here we investigate the latter. HC1 uses the CAWPE ensembling scheme, which was found to be the best combination method for small sets of diverse classifiers across two different dataset archives with limited domain specialisation or prior knowledge [68]. It was also shown that it improved HC1's performance relative to the previous simple majority voting. With updated components, which may be more or less specialised into their own representation formats with different degrees of overlap in their expertise, does this still hold true, or would

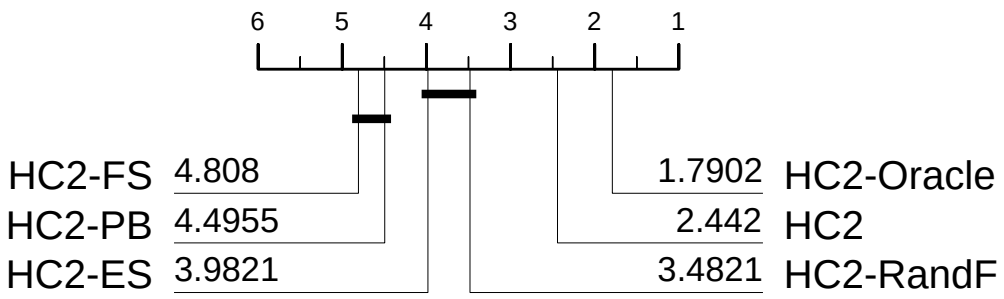


Fig. 6.12 Critical difference diagram comparing different ensemble schemes to default HC2 over the 112 univariate archive datasets.

a different scheme be better? We compare various ensemble selection and stacking schemes to assess whether a more complex scheme than CAWPE could improve HC2. To avoid suspicions of overfitting, we make it clear that we performed this analysis after generating the results presented in Section 6.3 using the design we selected a priori and inherited from previous versions of HIVE-COTE.

For context, we also compare the individual model selection schemes of picking the best classifier per dataset resample based on the train estimates, and picking the best based on the test data (i.e. cheating) as an oracle scheme. In general, a reasonable ensembling scheme will on average perform somewhere between these two landmarks across arbitrarily large dataset spaces. Combining the predictions of the classifier pool has a beneficial averaging effect by accounting for the imperfections of performance estimation mechanisms from the train data. However, they may have an overly conservative averaging effect compared to looking at test performance to pick the best.

Figure 6.12 summarises our comparison of alternate ensembling schemes over the HC2 components. HC2-Oracle cheats by picking the single best component based on test accuracy, while HC2-PB picks the best based on the train estimate of test accuracy. HC2-FS executes a forward selection of components per dataset, ranking the train estimates and continuing to include components into the ensemble

in order while the ensemble's own train estimate continues to improve. HC2-ES uses ensemble selection per dataset as described in [21], which selects and weights components based on a repeated bagging with replacement strategy. Lastly, HC2-RandF stacks a random forest classifier onto the meta-data of the components' predicted probability distributions.

We can see that, unsurprising, the oracle selection scheme is still the best on average, and that perfectly selecting the best representation per dataset would still be better than combining them. If there is enough training data to produce reliable train estimates that are unbiased and have very low variance, this may be achievable in practice. However, the reality on our data is that picking the best on the train data (HC-PB) is significantly worse than HC2 and HC2-Oracle. It is worth stressing that selecting the single best component on test accuracy is not always the best. In accordance with the original hypothesis for HIVE-COTE, there are 31 datasets where combining representations is outright better than picking the best, even with perfect hindsight of stochastic differences brought about by resampling. For many problems, discriminatory features may exist in multiple domains. This is often counter to received wisdom, it is always tempting to think a single type of model is the best approach. HC2 can discover complex interactions between domains. Figure 6.13 compares the ranks of HC2 (with its default CAWPE) and its individual constituents in isolation. HC2 is in fact best or tied for best on 57 of the 112 datasets, and rarely if ever ranked worse than second. This shows that beyond the requirement for perfect domain knowledge being required to beat ensembling on average, on many individual datasets more representations increase the accuracy outright.

Otherwise, Figure 6.12 shows that using the CAWPE scheme over the HC2 components is significantly better than the alternatives on average. Most popular ensemble schemes in the literature assume a large pool of potentially homogeneous classifiers. We have a small pool of heterogeneous classifiers, and evidence from

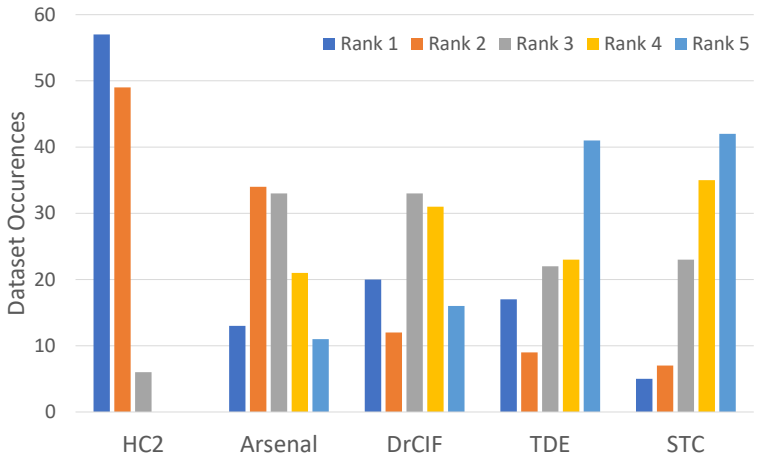


Fig. 6.13 Histograms of ranks between HC2 and its components over the 112 univariate datasets.

these experiments and an extensive study on standard classification problems [68] suggest that CAWPE is the best ensemble scheme for this scenario.

HC2 has a single parameter through the CAWPE ensemble scheme, the weighing factor α. We set α to four when first developing the CAWPE algorithm with the UCI datasets, and have kept it the same to avoid the danger of parameter selection bias. However, it is worthwhile considering how sensitive HC2 is to this parameter. We evaluated HC2 for $\alpha = \{1,2,\ldots,10\}$. We found that in fact $\alpha = 8$ is the best overall, but that there was very little difference between all values greater than three.

6.4.4 Importance of the Arsenal and Probability Estimates

Finally, we explore the effect of using Arsenal instead of ROCKET within HC2. Figure 6.14 shows both versions of ROCKET and three versions of HC2, two with a single version of both included in the ensemble and one containing a version of Arsenal where the probability of the predicted class is set to 1 rather than generated through the ensemble (Ar1H). Arsenal makes no improvement over default

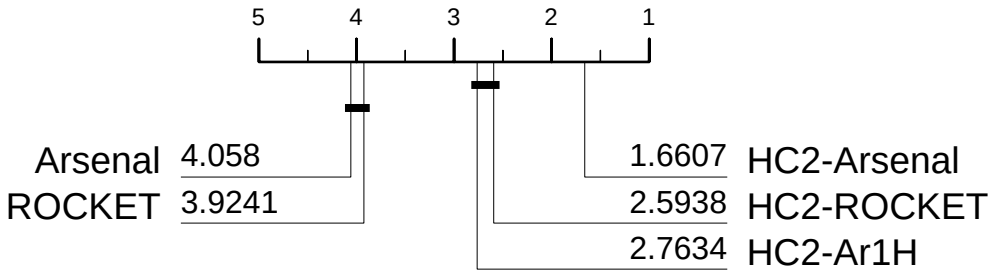


Fig. 6.14 Critical difference diagram for both versions of ROCKET and versions of HIVE-COTE using them on 112 UCR datasets. HC2-Ar1H represents HIVE-COTE using the Arsenal classifier, with probabilities generated in the same way as ROCKET.

ROCKET in terms of accuracy, and Arsenal using the same method for generating probabilities as ROCKET makes no improvement in HC2. However, the HC2 version including an unaltered Arsenal is significantly better. Even with probabilities estimated through the votes of a small sized ensemble, a large difference is made over having none at all in HIVE-COTE.

To investigate the type of datasets that most benefit from using Arsenal rather than ROCKET, we looked at the datasets where HC2-Arsenal is on average more accurate than HC2-ROCKET by more than 0.5%, but where we do not see the same difference in the difference between Arsenal and ROCKET. Twenty-two data sets fulfil these criteria. These 22 have, on average, more class values than the other 90 datasets. The mean number of classes for the 22 HC-Arsenal winners is 16.22 (median 6.5), whereas for the rest the mean is 6.34 (median 3). The 22 are also on average longer. The mean series length for the 22 is 860, whereas for the rest it is 480.

6.5 HC2 Usability

Table 6.3 shows that when run sequentially, HC2 is slower than the current state of the art, particularly ROCKET. If speed is more important than a small accuracy

Table 6.7 Train times in hours on problems where a sequential run of HC2 takes longer than 12 hours.

Problem	Rocket	InceptionTime	TS-CHIEF	HC1	HC2
ElectricDevices	0.43	6.15	7.94	56.80	54.55
Crop	0.22	4.37	4.59	73.54	28.41
FordB	0.19	6.06	43.33	19.33	24.67
FordA	0.19	6.01	40.88	20.25	23.52
HandOutlines	0.23	7.11	166.76	7.77	18.53

gain, this is an argument against using HC2. HC2 is simply not designed to be trained in seconds, and we would not recommend its use in scenarios where models need to trained incredibly quickly. However, we have designed HC2 so that the run time can be controlled by the user through a time contract. We make the assumption that when time is a serious constraint, the problem must be fairly large. There are only five problems where a sequential build of HC2 would take more than half a day with a single processor run. The list of problems, and the time taken by TS-CHIEF, InceptionTime and ROCKET, are listed in Table 6.7. ROCKET is very fast, InceptionTime hard to compare to and TS-CHIEF is similar to HC2 on average but is unpredictable. If we run HC2 with a four-hour contract on these problems we achieve 98% of the final accuracy, and if we run it for 12 hours we achieve 99% of the full build accuracy. If a reasonable model on bigger problems is required in hours, then contracting HC2 offers a good solution. However, if the problem is truly large, then all TSC have usability issues. TS-CHIEF can require massive amounts of memory and/or time, and the memory usage of ROCKET can rapidly increase with the number of cases used.

For genuinely large data where HC2 may take weeks for a full run, it is worthwhile considering how long it would take to converge. With many algorithms, it is often the case that most of the gain in accuracy is made relatively quickly. On large datasets, this can equate to days of processing that contributes relatively little to the overall performance. Furthermore, a practitioner's concern may not be

Table 6.8 Table showing both accuracy achieved by last checkpoint and variance in accuracy between first and last checkpoint for each HC2 constituent on the FruitFlies dataset.

	Accuracy	Variance	Constituents
TDE	0.8016	-0.0822	20
STC	0.8901	-0.0431	N/A
DrCIF	0.9278	-0.1084	65
Arsenal	0.6433	-0.0131	25

accuracy, and an understanding of the evolution of performance over time provides a foundation on which parameterisation decisions can be made.

In this section, we comment on the evolution of accuracy over time for each of the HC2 components. Experiments were run on 2 large insect audio datasets, described in chapter 3. The datasets used are notoriously problematic for complex approaches. This is typically because internal transforms are sensitive to series length, number of cases, or both. These datasets were deliberately chosen to explore the limitations of the HC2 constituents.

In order to overcome the imposed runtime limitations on the UEA HPC cluster a checkpointing mechanism was utilised to periodically save an approaches state during the training phase. This allowed both the continuation of training beyond what would usually be feasible, via reloading and continuing training from the saved point, and the opportunity to assess the approach at each saved state, via invoking the test phase after reloading the saved state. Figures 6.15 & 6.16 show how relative accuracy changes with respect to time for the FruitFlies and InsectSound datasets. Each data point shows the relative difference of the accuracy achieved at each checkpoint with respect to the last checkpoint recorded. Tables 6.8 & 6.9 present the real accuracy achieved with the last checkpoint processed, the variance in accuracy between the first and last recorded checkpoint, and the number of constituents built in by the last checkpoint.

Table 6.9 Table showing both accuracy achieved by last checkpoint and variance in accuracy between first and last checkpoint for each HC2 constituent on the InsectSound dataset.

	Accuracy	Variance	Constituents
TDE	0.2746	-0.0735	14
STC	0.7357	-0.1509	N/A
DrCIF	-	-	-
Arsenal	0.2951	-0.0313	10

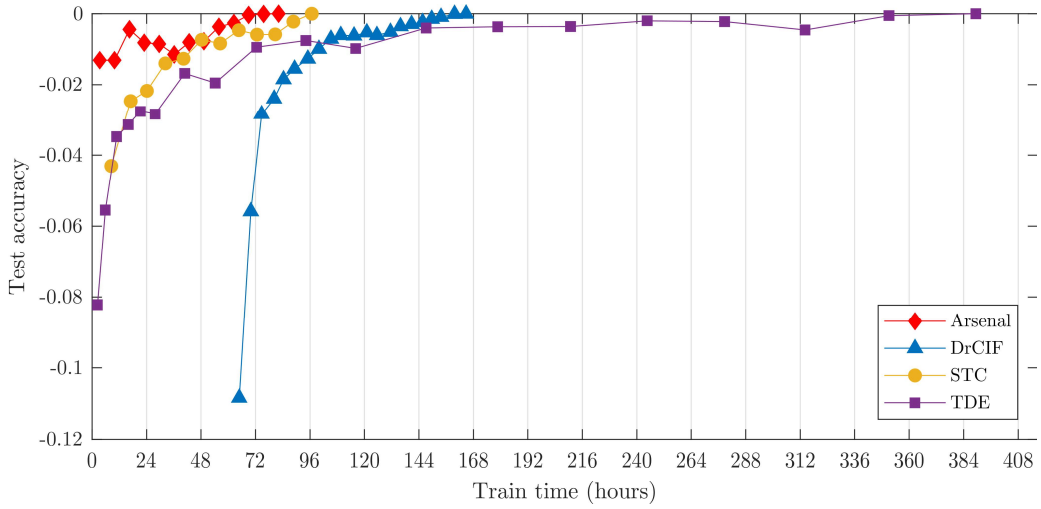


Fig. 6.15 Accuracy as a function of train time for HC2 components on the FruitFlies dataset.

Figures 6.15 & 6.16 show the accuracy of DrCIF, STC and TDE follow a similar trend with respect to time. In the case of these approaches accuracy increases quickly as the number of constituents present increases, before reaching the point of diminishing returns. For these approaches 80% of the accuracy achieved is done so in less than 50% of the train time used by the last recorded checkpoint. The Arsenal approach does not follow this trend and instead the total variance in accuracy throughout the training period is not as pronounced. Also, the changes in accuracy appear to be more erratic with additional constituents producing decreases in accuracy as well as increases.

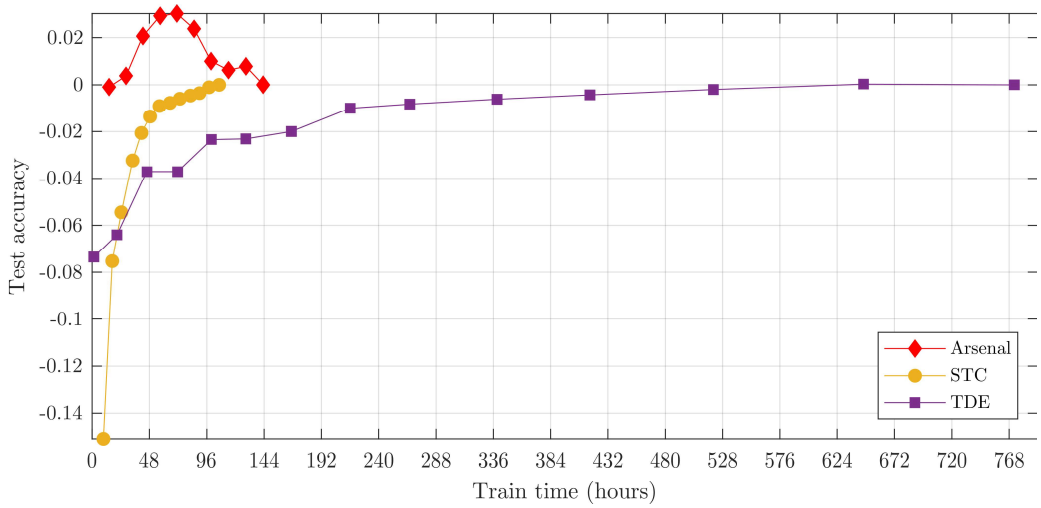


Fig. 6.16 Accuracy as a function of train time for HC2 components on the InsectWingbeat dataset.

Furthermore, of the 8 combinations presented only the Arsenal approach on the FruitFlies dataset was able to complete. In most cases the outstanding experiments were prohibited by inflation in the time taken to test. Checkpointing during the test phase is not implemented and as a result the entire test process is subject to a hard time limit of 7 days. This effected STC and DrCIF on the FruitFlies dataset and STC and TDE on the InsectSounds dataset. Aditionally, the Arsenal approach was limited by its memory requirement on the InsectSounds dataset, for which we are limited to 700Gb.

6.6 Conclusion

In this chapter we propose an upgraded HIVE-COTE in HC2, a meta ensemble of four very different classifiers, each of which is designed to capture different discriminatory features. It represents a new state of the art in terms of time series classification, significantly outperforming the previous best on both univariate and multivariate problems in terms of accuracy. Our ablation study showed that HC2 is

better than any one of its constituents, and that each component makes a significant contribution to the overall performance. We believe its strength lies in the fact that many problems have discriminatory features in multiple data domains; a shapelet might be indicative of one class value, whereas a repeating pattern may characterise another. HC2 uses a simple yet highly effective ensemble scheme to combine this information in CAWPE, which we demonstrated was significantly better than alternatives such as stacking or a selection strategy.

HC2's weakness is that it does not scale well to very large problems. We showed that for problems with thousands of series of length in the tens of thousands, build times can be excessive, but that with contacting an estimate be obtained at least. We note that the current state of the art has similar limitations. Even ROCKET, which is by far the fastest algorithm, can struggle to scale in terms of memory with increasing number of cases.

There is room for further improvement with HC2. The STC design remains the same in HC1 and HC2, and we believe there are ways it could be improved. Contracting could be enhanced so that components could produce bespoke estimates of the time, and reconfigure themselves more intelligently to make the best use of the available run time. Variability in estimates from the train data could be incorporated into the ensemble process, and weights per case might improve predictions. Over the last six years, the COTE classifiers have been developed using the UCR data sets for evaluation. There is a danger that we have simply been overfitting the archives. We believe the diversity and number of datasets used, and the repeated resampling, make this unlikely. However, it is a genuine concern, in particular for future development. We are compiling new datasets for both archives, and once this process is complete we will repeat all experiments with new data. However, we need to wait until the new archive datasets are publicly available to avoid any suspicion of selection bias. Hence, we classify this as future work.

In the next chapter, we apply the classification algorithms we have developed in this thesis to the time series nuisance call centre detection problem introduced in Chapters 1 and 2.

Chapter 7

Detecting Nuisance Call Centres using Time Series Classification

7.1 Introduction

In Chapters 1 and 2 we outlined the problem of nuisance calling, the harm to consumers caused by nuisance callers and current techniques used to detect nuisance calls. The threat of nuisance calls harassing consumers is a problem that has expanded drastically in the last decade, in part due to the increasing availability of cheaper Voice over Internet Protocol (VoIP) systems and automated robotic calls. A 2018 report by two UK regulatory bodies, The Office of Communications (Ofcom) and the Information Commissioner's Office (ICO), estimated 3.9 billion nuisance calls were received per year in the UK [130]. The 2021 report indicates a drop in nuisance call complaints in recent years, but attributes some of these annual drops to the initial Covid-19 lockdowns [131]. This problem is not isolated to the UK. The number of complaints has risen significantly for 'robocalls' in the US [132] and the number of cases for both 'number misuse' and nuisance marketing in Germany [50] has risen.

Due to the extreme volume of traffic in networks such as the ones maintained by BT, an automated solution is required for a quick response to nuisance callers. Current processes have the activity of individual Calling Line Identification (CLI) numbers manually screened if they are suspected to be carrying nuisance traffic by the BT Call Protect system. If this number is found to be distributing significant amounts of nuisance calls, it is temporarily added to a blacklist. However, the large number of candidates can cause a large backlog of numbers. By the time many of these are processed, the damage has already been done and callers have moved to a new CLI, avoiding the blacklist entirely.

In this chapter, we look into differentiating between nuisance and legitimate call centres using time series of outgoing call volumes from individual CLIs which send high volumes of total traffic through the British Telecom (BT) network. We evaluate this approach using state-of-the-art Time Series Classification (TSC) algorithms seen throughout this thesis. Legitimate call centres play an important role in communications to consumers for many organisations. It is important that any system has a high enough specificity to keep the amount of legitimate centres misclassified as nuisance to an absolute minimum. For example, in the event any automated system blocks calls to or from a hospital call centre, lives could be put at risk. Each series is collected over 24 hours, meaning that to make a classification, the full day's worth of nuisance calls will have already gone out to consumers. We investigate early TSC (eTSC) algorithms to reduce the amount of data required, so action can be taken in real-time.

This chapter is structured as follows. Section 7.2 describes the synthesised call centre volume dataset altered from real examples which we explore in this chapter. We show the results of our algorithms in detecting nuisance call centres in Section 7.3 in both a regular and early classification setting. Section 7.4 explores the more interpretable algorithms used to further analyse the data. Conclusions are drawn in Section 7.5.

7.2 Synthesised Call Volume Profiles

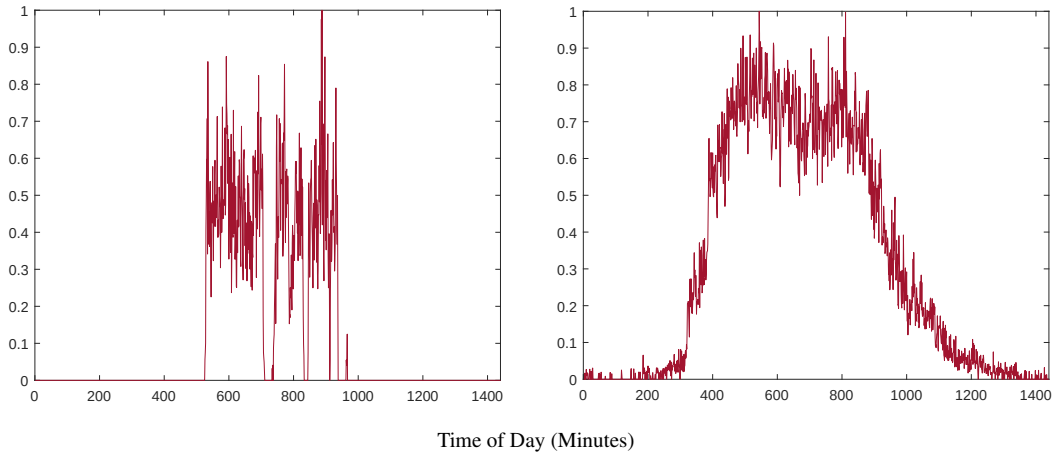
We experiment on synthesised time series data based on the traffic from real nuisance and legitimate call centres through the BT network. There are a total of 500 time series, half of which are labelled as legitimate and the other half nuisance. Each data instance consists of a class value and a time series made of 1440 ordered values, each representing the number of outgoing calls that have successfully connected in that minute for a single day. We define successfully connected calls as ones which have reached their destination, regardless of whether the recipient answers or the call is handled by an answering machine. Calls dialled to an invalid number or which have failed to connect due to network congestion or similar issues are not included in the count. Each time series was collected from the traffic of a single CLI over the period of a single day, with any series under an unspecified total volume discarded to ensure only call centres are included. We label an instance as nuisance if the CLI has been added to the BT CLI blacklist after going through a manual screening process of its activity. All other instances are assumed legitimate.

Call centres will operate differently, mainly differing in the varying degree of automation present. Some call centres will be close to fully automated, with an algorithm making calls and robotic messages being played. Most call centres will have some form of automation in the form of auto-diallers to increase the outgoing rate of calls for employees, though this is not always the case. These auto-diallers are regulated to reduce the amount of calls made without an available operator, better known as silent calls. Legitimate call centre operations will configure their auto-diallers to meet these regulations in most cases. Nuisance call centres on the other hand will be more likely to disregard this, and operate to output as many calls as possible regardless of calls made with no operator to deliver a message. It is expected that both human based and robotic call centres will be present in

both nuisance and legitimate classes. However, the distribution of fully robotic and unmanaged auto-diallers are thought to be much more prevalent in nuisance call centres.

Privacy is a major issue in providing call data; even volumes for a single CLI are viewed as sensitive. While call data from real CLIs is available, the data we use in this section has been altered slightly. Call volumes have had minor adjustments from their original state, and portions of traffic has been moved earlier or later in the series in ways which retain the key identifiers of nuisance calling. All time series are normalised to contain values between zero and one, both to prevent the size of the call centres from impacting the decision and to better anonymise the data. While the data is synthetic, it is sourced from real high call volume CLIs which BT has carried traffic for, and the patterns shown are near identical to unaltered examples.

We display an example case from each call centre class in Figure 7.1. For manually determining the legitimacy of a call centre, an indicator used currently is how fast the call volume ramps up to its standard operation volume at the beginning of the day and down at the end. In robotic centres, sudden jumps in outgoing calls are seen as the systems responsible for making calls go online or offline, suddenly changing at a single time point or in batches depending on how the system is set up. Call centres which are primarily human based have more gradual ups and downs as callers start working, take breaks and leave at different times, giving a more natural start and finish shown in the legitimate example. For legitimate call centres with human operators, a drop in the afternoon is common, which may not show in nuisance ones. Call recipients become more likely to converse for longer at these times, which will lower call output as operators are spending more time occupied with calls rather than making them. This is less likely to be the case for nuisance calls however, as the call contents are most likely undesirable regardless of the time of day, or they are fully robotic and have limitless operators. Other factors such



(a) Nuisance (b) Legitimate

Fig. 7.1 Plotted graphs displaying the daily call volume (normalised) for each minute of a day to form a time series of a legitimate call centre (a) and a robotic nuisance call centre (b) from the synthesised call centre volume dataset.

as structured times for breaks and meetings in legitimate centres are also likely to have an impact.

7.3 Results

For each approach in our experiments, we run a 10-fold cross-validation and collate the test results from each fold. To keep our comparison fair and reproducible, we seed both creation of folds and any random elements in the individual classifiers, with all folds and classifiers using a single seed.

Before running complex TSC algorithms, we test whether simple summary statistics are enough to separate legitimate and nuisance series. The summary statistics we experiment with as features the mean, variance, skewness, kurtosis, slope, minimum of first order difference and maximum of first order differences. These features are simple, interpretable and quick to process, as such it is a good solution if they are all that are needed to make accurate predictions, and they provide a benchmark for more complex models if not. Which of these stats performs best can also provide some insight to which parts of the series matter most for classification.

Table 7.1 Results on the synthesised call centre volume dataset over a 10-fold cross-validation. Displays the accuracy performance of simple summary statistics using a TimeSeriesTree [40] and Rotation Forest [103]

Summary Statistics	Tree Accuracy	RotF Accuracy
Mean+Variance+Skewness+Diff Max	0.8860	**0.9580**
Mean+Variance+Kurtosis+Diff Min+Diff Max	**0.9100**	0.9340
Mean	0.7120	0.7600
Variance	0.5380	0.5000
Skewness	0.7580	0.7340
Kurtosis	0.6160	0.6280
Slope	0.6400	0.6440
Diff Min	0.5260	0.6260
Diff Max	0.5500	0.5840

To test these summary features, we build a simple tree classifier used as a base for many interval based forests [40, 85] and Rotation Forest (RotF) [103], which has shown to perform well with continuous attributes [4]. Table 7.1 shows the accuracy for each of our summary statistics, as well as the most accurate combination of summary statistics we found for both classifiers (Mean+Variance+Skewness+Diff Max for our tree, and Mean+Variance+Kurtosis+Diff Min+Diff Max for RotF). The best performing individual features are the mean and skewness of the series, a mix of features is required to obtain a respectable level of accuracy, however. While simple statistics for Rotation Forest perform well, a more complex group of summary features specialised for time series data in the Catch22 feature set can perform even better on the problem, as we will show in the following results.

We now take a look at the current state-of-the-art for TSC on our synthesised call center volume data. To benchmark against the TSC algorithms alongside the previous summary statistics, we include five traditional machine learning algorithms in Naive Bayes (NB), a linear (SVML) and quadratic (SVMQ) Support Vector Machine (SVM), RotF and XGBoost [28]. Our hypotheses is that these benchmark methods will not be sophisticated to capture all the features which discriminate between nuisance and legitimate centres. We experiment with a large range of

Table 7.2 Results on the synthesised call centre volume dataset over a 10-fold cross-validation. Displays classifier performance using Accuracy, Area Under the Receiver Operating Characteristic (AUROC), Sensitivity (True Positive Rate) and Specificity (True Negative Rate).

Classifier	Accuracy	AUROC	Specificity	Sensitivity
NB	0.7020	0.4867	0.8000	0.6040
SVML	0.7320	0.5782	0.7800	0.6840
SVMQ	0.7960	0.7032	0.7360	0.8560
RotF	0.9540	0.9946	0.9320	0.9760
XGBoost	0.9420	0.9936	0.9160	0.9680
1NN-DTW	0.9980	0.9960	0.9960	**1**
Arsenal [87]	0.8920	0.8877	0.8800	0.9040
catch22 [80]	**1**	**1**	**1**	**1**
cBOSS [88]	0.9980	**1**	0.9960	**1**
CIF [85]	**1**	**1**	**1**	**1**
DrCIF [87]	**1**	**1**	**1**	**1**
FreshPRINCE [84]	**1**	**1**	**1**	**1**
HC1 [5]	**1**	**1**	**1**	**1**
HC2 [87]	**1**	**1**	**1**	**1**
InceptionTime [43]	**1**	**1**	**1**	**1**
ROCKET [37]	0.8960	0.8042	0.8880	0.9040
STC [17]	0.9980	**1**	0.9960	**1**
STSF [19]	0.9980	**1**	0.9960	**1**
TDE [86]	0.9940	**1**	0.9920	0.9960
TS-CHIEF [114]	**1**	**1**	**1**	**1**
TSF [40]	0.9980	**1**	0.9960	**1**
WEASEL [108]	**1**	**1**	**1**	**1**

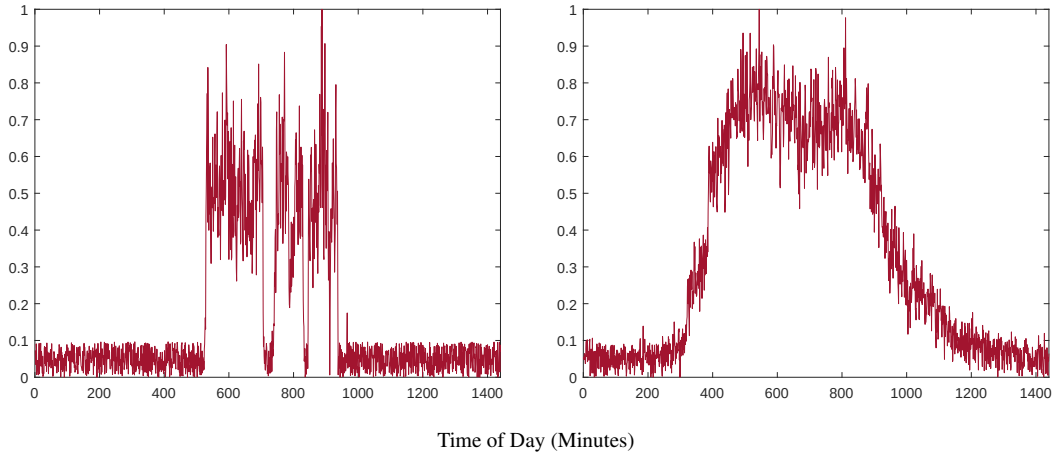
time series specific algorithms which have proven to be competitive on general TSC problems from the commonly used UCR archive [36, 8], and which we have developed and compared against throughout this thesis. Details of the algorithms used in our experiments which have not been developed throughout this thesis can be found in Chapter 2.

Results for all classifiers on our data are shown in Table 7.2. With the notable exception of ROCKET and Arsenal, all the time series classifiers we selected performed perfectly on the data or close to perfect. Our selection of traditional classification algorithms could not reach the same level of performance, showing the benefits of time series specific features. Of the algorithms with perfect results, the

best in terms of scalability was the Canonical Time-series Characteristics (Catch22). On average, Catch22 with a Random Forest classifier took 5 milliseconds to make predictions on unseen series and 2790 milliseconds to train on the dataset.

Other than the expected features we have already described for differentiating nuisance and legitimate call centres, there is an unexpected yet visible quirk with the provided data. Looking back at the data examples shown in Figure 7.1, there is a noticeable lack of any calls for long stretches of time in the nuisance call centre example, creating a long string of zeros in the series. The legitimate call centre, on the other hand, has calls throughout the day. While not true for all examples, this is a trend continues in the majority of remaining cases for both classes. It is possible this is a reasonable feature to extract for our classifiers, rather than a quirk of a dataset. It is in the interest of nuisance callers to operate in profitable hours for their campaigns, with little benefit to operating robotic systems for a small amount of calls. Legitimate call centres on the other hand may operate globally or have obligations requiring 24 hour contact. However, in the interest of selecting a more robust classifier if this is deemed an undesirable feature to base classifications on, we look at making predictions for our cases without these long periods of silence by adding random noise. This added noise does not necessarily provide a more accurate representation of real call volume profiles, but aims to make the problem more difficult by removing some features that could potentially be misleading for models given the training cases provided. Noisy profiles such as these are seen in other nuisance calling problems such as CLI rotation which we sover in Chapter 8, however. Figure 7.2 shows our example series with 10% noise randomly applied.

We show our results on this noisy dataset in Table 7.3. While dictionary based approaches and simpler classifiers such as TSF are impacted negatively slightly, most classifiers retain their perfect or near-perfect accuracy. Interestingly, ROCKET and Arsenal perform much better with the added noise, showing that perhaps the



(a) Nuisance (b) Legitimate

Fig. 7.2 Plotted graphs displaying the daily call volume (normalised) for each minute of a day to form a time series of a legitimate call centre (a) and a robotic nuisance call centre (b) from the synthesised call centre volume dataset with added noise.

long string of zeros is more confounding to some of our approaches rather than a useful feature.

7.3.1 Early Classification

Early Time Series Classification (eTSC) aims to maintain accurate classifications with as little of the time series as possible. By its nature, this goal is somewhat contradictory, as reducing the amount of data in the series for an earlier prediction is likely to reduce accuracy. As such, the effectiveness of an approach will differ by how it balances this trade-off. There is an argument to be made that the complexity and overhead accompanying eTSC approaches is unnecessary with proper data preprocessing. If only the first half of the series is required to make an accurate classification, why not just use the first half as the full series? While this is correct in some cases, the nature of classifying call centres makes this difficult due to the variability in position of our discriminatory features in the series. We have no guarantee when a call centre starts and ends its business, and calls can be received from all time zones.

Table 7.3 Results on the synthesised call centre volume dataset with added noise over a 10-fold cross-validation. Displays classifier performance using Accuracy, Area Under the Receiver Operating Characteristic (AUROC), Sensitivity (True Positive Rate) and Specificity (True Negative Rate).

Classifier	Accuracy	AUROC	Specificity	Sensitivity
NB	0.6720	0.4637	0.7720	0.5720
SVML	0.7260	0.5526	0.7640	0.6880
SVMQ	0.7940	0.6858	0.7960	0.7920
RotF	0.9200	0.9859	0.8600	0.9800
XGBoost	0.9060	0.9712	0.8600	0.9520
1NN-DTW	0.9980	0.9960	0.9960	**1**
Arsenal	**1**	**1**	**1**	**1**
catch22	0.9980	**1**	**1**	0.9960
cBOSS	0.9900	0.9995	0.9840	0.9960
CIF	0.9980	**1**	0.9960	**1**
DrCIF	**1**	**1**	**1**	**1**
FreshPRINCE	**1**	**1**	**1**	**1**
HC1	**1**	**1**	**1**	**1**
HC2	**1**	**1**	**1**	**1**
InceptionTime	**1**	**1**	**1**	**1**
ROCKET	**1**	**1**	**1**	**1**
STC	0.9980	**1**	0.9960	**1**
STSF	0.9980	0.9998	0.9960	**1**
TDE	0.9880	**1**	0.9800	0.9960
TS-CHIEF	0.9960	**1**	0.9920	**1**
TSF	0.9780	0.9984	0.9760	0.9800
WEASEL	**1**	**1**	**1**	**1**

If the goal is to reduce the amount of out-going nuisance calls by making predictions in real-time, classifying nuisance call centres as early as possible is of the utmost importance. The earlier a nuisance call centre is detected, the earlier action can be taken in preventing it from contacting consumers. Making a classification using the full series means that all the calls for the day have already gone out, and there is no guarantee the same CLI will be used the following day. While a decrease in specificity resulting from earlier predictions is likely to cause issues, a decrease in the amount of nuisance call centres detected may be acceptable to allow for earlier action against the remaining ones. We test three eTSC decision-making controllers using a subsample of the accurate, scalable and diverse TSC

algorithms used in our previous experiment. These classifiers are: 1NN-DTW, catch22, cBOSS, DrCIF, HIVE-COTE 2.0 and STC.

To summarise the workflow of our eTSC controllers, each controller considers whether to make a classification decision using a limited amount of data at set breakpoints. For the case of nuisance calling, we make a decision every hour, starting at 60 time points of data and making another decision at each increment of 60 until a classification has been deemed safe or the full 1440 series length has been reached, in which case the decision is always positive. In a real scenario, data would be fed to the model as a stream as it becomes available in these hourly intervals. The decision whether to make a classification or not is made using the output of a TSC algorithm built on time series of the same incrementing lengths. As such, each eTSC method uses 24 trained models, one for each hour where we want to evaluate whether the currently obtained data is suitable to make a classification. While this notably increases train time, only a single model is used when making a prediction for a given hour and deciding whether said prediction is safe to use. For brevity, we again omit the intricate details for each of these eTSC controllers. Full descriptions are available in Chapter 2. The first eTSC controller is P85, a simple probability filter requiring probability estimates above 0.85 for a single class from three consecutive hourly classifications before a prediction is deemed safe. SR1CF1 [92] treats eTSC as an optimisation problem, optimising its stopping rule using a cost function based on accuracy and how early a classification is made. TEASER [110] trains a one-class SVM to decide whether a prediction is safe, and tunes the number of consecutive decisions required to make a final safety decision.

We show the results for each controller and classifier combination in Table 7.4. We introduce two new result statistics from eTSC literature. Earliness represents the average proportion of the series required to make a classification, with a lower earliness score meaning quicker classification are being made. Harmonic Mean (HM) represents the balance between accuracy and earliness. All of our algorithms

Table 7.4 Results on the synthesised call centre volume dataset over a 10-fold cross-validation for a range of early classification controllers using TSC algorithms. Displays classifier performance using Accuracy, Area Under the Receiver Operating Characteristic (AUROC), Specificity (True Negative Rate), Earliness and Harmonic Mean (HM).

Controller	Classifier	Accuracy	Specificity	Earliness	HM
P85	1NN-DTW	0.8660	0.7400	0.1320	0.8670
P85	catch22	0.9900	0.9800	0.3140	0.8100
P85	cBOSS	0.8860	0.7720	0.1900	0.8450
P85	DrCIF	0.8800	0.7600	0.1530	0.8620
P85	HC2	0.8800	0.7600	0.1578	0.8600
P85	STC	**0.9980**	**1**	0.3300	0.8020
SR1CF1	1NN-DTW	0.9820	0.9800	0.4170	0.7320
SR1CF1	catch22	0.9600	0.9280	0.2120	0.8650
SR1CF1	cBOSS	0.9620	0.9240	0.3870	0.7480
SR1CF1	DrCIF	0.8480	0.6960	**0.0520**	**0.8950**
SR1CF1	HC2	0.8920	0.7880	0.1670	0.8600
SR1CF1	STC	0.9700	0.9560	0.2340	0.8560
TEASER	1NN-DTW	0.9960	0.9960	1	0
TEASER	catch22	0.8600	0.7280	0.0960	0.8810
TEASER	cBOSS	0.8600	0.7320	0.1070	0.8750
TEASER	DrCIF	0.9780	0.9640	0.2820	0.8280
TEASER	HC2	0.9760	0.9560	0.2450	0.8510
TEASER	STC	0.8560	0.7200	0.0950	0.8790

take a loss in accuracy and specificity to achieve an earlier classification, with the exception of TEASER using a 1NN-DTW classifier, which refuses to make an early classification at any point. The stand-out classifier is STC with a simple probability threshold, correctly classifying all legitimate centres and all but one nuisance case. On average, cases were classified using just under 1/3 of the series or at 07:55 A.M. While it is hard to determine how many calls would be prevented by making classifications at this time due to the series normalisation, calls prior to 07:55 A.M account for 21.33% of traffic on average in our 250 nuisance cases. Catch22 has the next highest accuracy while classifying earlier than STC, making a prediction at 07:32 A.M on average. Only 19.5% of traffic nuisance traffic remains on average using this as a cut-off point, but five legitimate call centres are misclassified.

Two of our eTSC controllers contain parameters which can be altered to favour earliness or accuracy. P85 can increase the number of consecutive prediction or increase the probability required to make a final decision. SR1CF1 weights its cost function with an α parameter. As a major impact in the usability of the algorithms we included was a drop in specificity, we run a second experiment with more conservative parameters to retain accuracy. P85 now requires 5 consecutive predictions above the specified probability, and the α of SR1CF1 has been increased from 0.8 to 0.9. Table 7.4 shows the results for these new settings. Multiple combinations achieve a perfect accuracy, the earliest classifier among these being HIVE-COTE 2.0 with the SR1CF1 controller classifying at 10:03 A.M on average with 32.94% of nuisance traffic remaining. Classifications are made on average two hours later than the best from the previous experiment, with a considerable amount of extra nuisance traffic going out during that time. It is possible this extra time is worth the increased reliability, as mentioned previously mislabelled legitimate call centres could be incredibly costly. While more nuisance calls are reaching consumers, it is still a significant drop from the full day's worth and would likely damage the profitability of running these nuisance call centres.

We use our eTSC controllers out of the box except for the base classifier attached to them. These are modifications specific to the nuisance calling problem that could be made to produce earlier or more accurate results, however. While we want to stop nuisance call centres, there's not necessarily a reason to stop reassessing a series with more data after it has been classified as legitimate. Both classes are not equal, as such, tuning the controllers decision using accuracy or HM may not be the best approach to ensuring specificity remains high.

Table 7.5 Results on the synthesised call centre volume dataset over a 10-fold cross-validation for early classification controllers weighed towards accuracy. Displays classifier performance using Accuracy, Area Under the Receiver Operating Characteristic (AUROC), Specificity (True Negative Rate), Earliness and Harmonic Mean (HM).

Controller	Classifier	Accuracy	Specificity	Earliness	HM
P85	1NN-DTW	0.8830	0.7800	**0.2250**	0.8250
P85	catch22	**1**	**1**	0.4400	0.7180
P85	cBOSS	0.9080	0.8150	0.3280	0.7710
P85	DrCIF	0.9100	0.8200	0.2760	0.8050
P85	HC2	0.9100	0.8200	0.2740	0.8070
P85	STC	**1**	**1**	0.4310	0.7260
SR1CF1	1NN-DTW	0.9800	0.9750	0.4170	0.7310
SR1CF1	catch22	0.9750	0.9450	0.2650	0.8360
SR1CF1	cBOSS	0.9950	0.9900	0.4730	0.6890
SR1CF1	DrCIF	0.9550	0.9150	0.4170	**0.8690**
SR1CF1	HC2	**1**	**1**	0.4190	0.7350
SR1CF1	STC	0.9980	**1**	0.3450	0.7910

7.4 Analysis

We have presented our primary results on the synthesised call volume data problem, using a range of TSC and eTSC algorithms. We further analyse two of the more interpretable algorithms used in our experiments, Shapelet Transform Classifier (STC) and the Canonical Interval Forest (CIF), to see what features were extracted from the time series and if they match prior knowledge.

7.4.1 Shapelets

Shapelets are a phase independent subseries extracted from training time series aiming to capture discriminatory patterns. First used in Ye and Keogh [139], the high-level idea involves extracting a number of informative shapelets using a metric such as entropy or information gain to determine how well it discriminates between classes. New cases are then generally classified using the distance to extracted shapelets, with each being compared to all possible subseries of the same size as

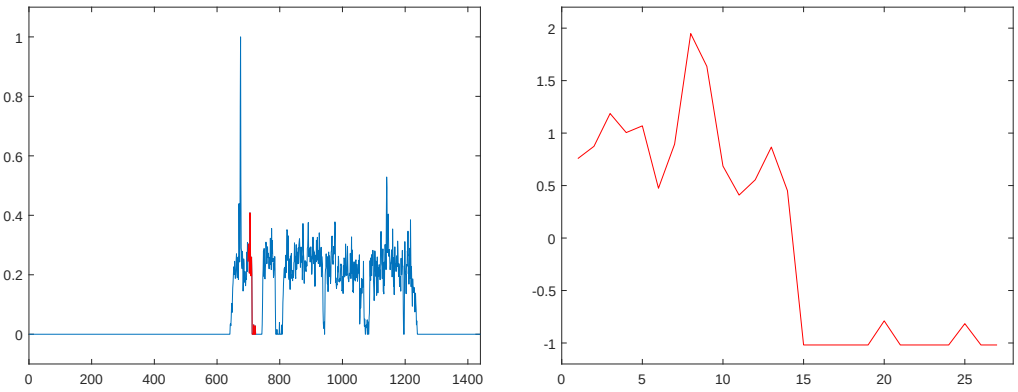
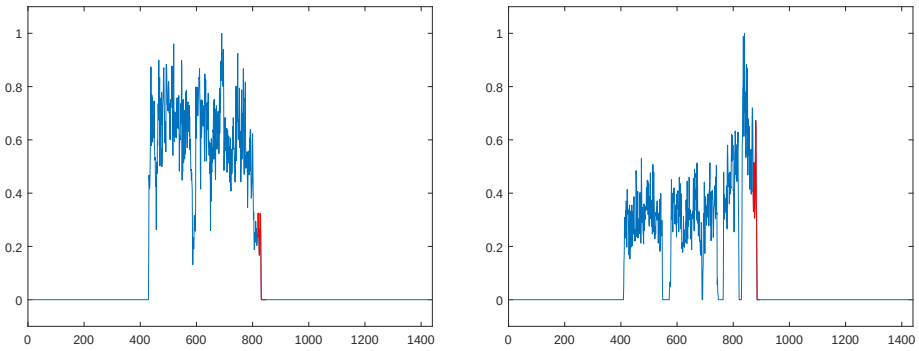


Fig. 7.3 The highest information gain shapelet from the STC training process. Displays the series the shapelet was extracted from, with its location highlighted in red (left) and the z-normalised shapelet (right).

the shapelet from the series. The distance of a shapelet to a series is determined to be the shortest distance from the shapelet to all subseries of the same length.

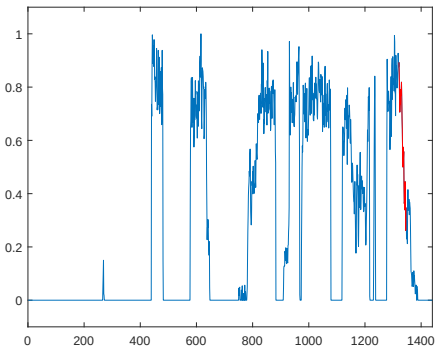
Given the prior knowledge of discriminatory features for nuisance calling, it was thought that a shapelet based approach one would find some success. This is due to their ability to find similarity based on ramp ups and downs from the start and end of call activity, regardless of when in the day the centre starts and finishes calling. This turned out to be true, as the STC and HIVE-COTE 2.0 (which includes STC in its ensemble) are two of our best performers in terms of accuracy for the early classification experiments. Fortunately, shapelets are very interpretable by their nature, as such we can directly look at the extracted shapelets with the most information gain to see how closely they match our manually selected discriminatory feature.

Figure 7.3 shows the most informative shapelet extracted when training STC using all 500 time series. As expected, the sudden drops by nuisance call centres are one of the most informative features for separating them from legitimate centres. As shapelets are phase independent, the selected shapelet can just as easily match the end of the active period and other sudden dropouts.

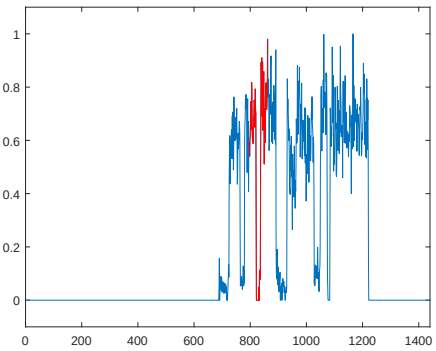


(2) 0.8507 IG

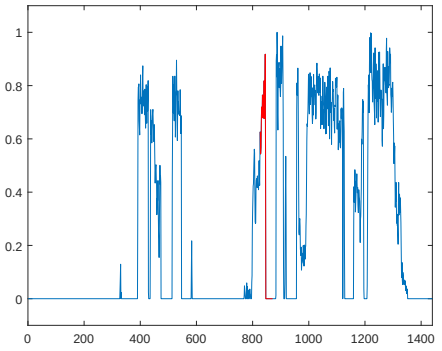
(3) 0.8445 IG



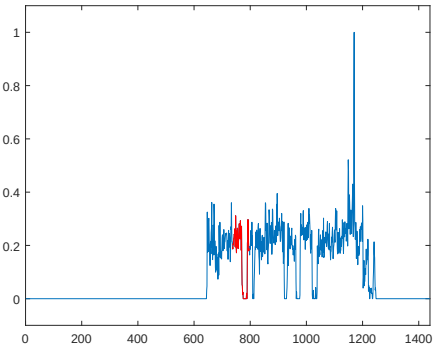
(4) 0.8409 IG



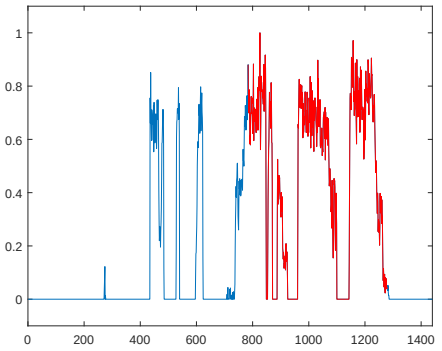
(5) 0.8321 IG



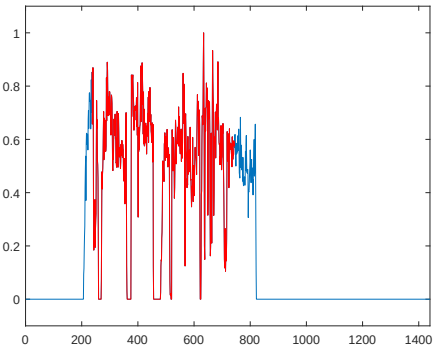
(6) 0.827 IG



(7) 0.8078 IG



(8) 0.8045 IG



(9) 0.7877 IG

Fig. 7.4 Shapelets ranked 2 through 9 by information gain from the STC training process. Displays each shapelet overlapping the series and position it was extracted from, alongside its rank and information gain.

The following shapelets ranking from 2 to 9 are shown in Figure 7.4. The next six are similar to the first, being small shapelets which focus on the sudden drops from nuisance call centres. It is worth noting that the shapelets extracted are all from drops in the centre of the active period or at the very end of the series rather than the start. This suggests that sudden falls in volume are more indicative of nuisance calling than rises. Rank 8 and 9 cover a large portion of the middle of our time series, covering multiple hours of calls with multiple sudden drops. All of the top informative shapelets we have shown have been extracted from nuisance call centres. An explanation for this may be that legitimate centres vary more, while nuisance ones follow a similar pattern.

7.4.2 Temporal Importance Curve

Similar to shapelets, interval approaches look at numerous subseries of the original whole series. While shapelets are phase independent by comparing the extracted subseries to all parts of the whole series, intervals are phase dependent comparing only to subseries extracted from the same time points. Instead of using distance, most interval approaches extract summary statistics from each subseries to be used for comparison. Looking at the most important intervals and summary statistics used could gleam further insights into the problem.

Figure 7.5 shows a temporal importance diagram [40] for the CIF classifier. This is created by summing the information gain of each node in the forest, giving us the more informative time points for each feature. CIF uses the Catch22 features, giving us insight into which features are informative for another stronger performer as well. We retain the curves for the three features with the most information gain and the mean of all features to remove clutter. Two of the best performing features are the longest period of successive incremental decreases and the first minimum of the automutual information function, which peak at the time points around 600

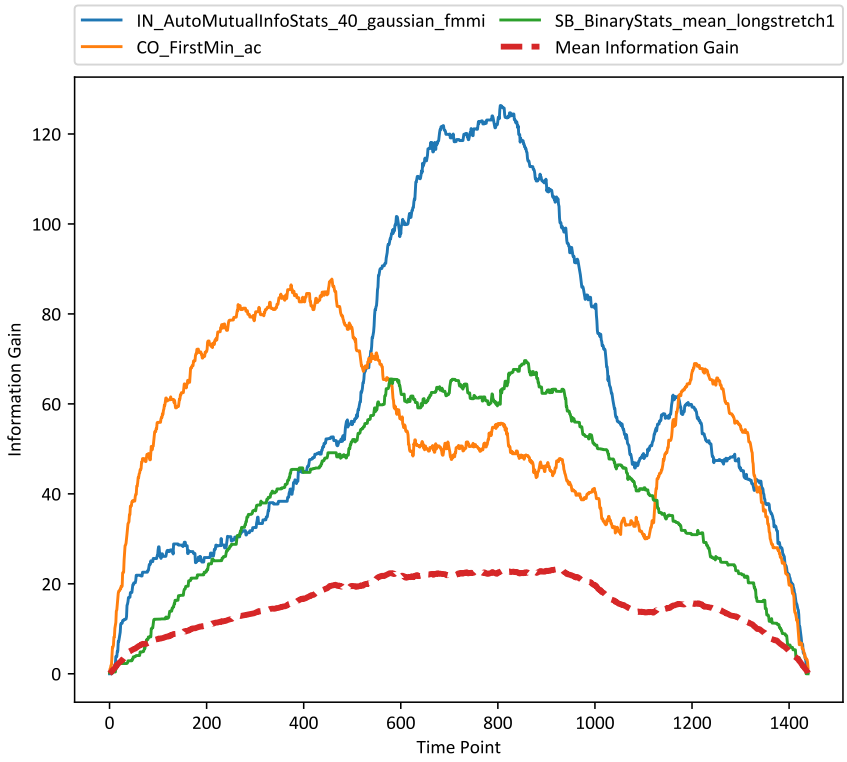


Fig. 7.5 CIF temporal importance diagram build on all 500 cases. Displays the information gain for the top 3 features at each time point in the series.

through to 1100, corresponding to 10:00AM to 6:20PM respectively, when many call centres will be fully active. The remaining feature is the first minimum of the autocorrelation function, which captures features mainly at the beginning and end of this active period.

7.5 Conclusions

In this chapter, we have explored the issue of nuisance call centres and shown the utility of TSC algorithms in separating nuisance and legitimate traffic using a time series representation of the data.

Using simple daily call volume time series, most of our TSC algorithms were able to perfectly separate the nuisance call centres from legitimate cases, where traditional machine learning algorithms were unable to. In an evaluation of early

TSC techniques, we found that a perfect classification could still be made with only a portion of the data readings required, making a prediction by 10:00 A.M on average and eliminating a significant amount of nuisance traffic if a decision to block further calls would be made at that point. Such a decision is not easy to make, however, as the cost for misclassifying a legitimate call centre is high. An early decision is unlikely to be useful unless it is verified or action is taken quickly. Further investigation is warranted, and the eTSC algorithms can be further configured to retain specificity if required. Regardless, these automated tools could be useful as an aid in human decision-making by selecting potential nuisance call centres from the massive amount of traffic going through the BT network daily.

A large issue in detecting nuisance callers is spoofing; the capability for callers to change what CLI their calls present as when it reaches the network to be distributed. So far in this chapter, we have operated under the assumption that nuisance callers that make use of spoofing constrain all or large amounts of their traffic to a single CLI. While some nuisance callers stick to a single CLI, others spoof their calls through multiple CLIs at low volume in an action we call CLI rotation. This is done in an effort to subvert current detection techniques, by keeping the traffic in each CLI used at low volumes. Our algorithms would not be able to detect traffic split in this way when looking at single CLIs. In the next chapter, we look at detecting traffic split over multiple CLIs for use in a TSC scenario.

Chapter 8

Detecting Calling Line Identification Rotation

8.1 Introduction

In Chapter 7 we introduced the problem of detecting nuisance call centres using call volume time series data and Time Series Classification (TSC) techniques. As we have mentioned previously, nuisance callers are able to change what their Calling Line Identification (CLI) appears as using spoofing. While misuse of spoofing is troublesome as it allows nuisance callers to avoid CLI blacklists and hides the true source of calls, it does have uses for legitimate call centres needing to present as a single CLI, such as for a marketing campaign. As such, actions against the act of spoofing itself is difficult. In the previous chapter, we introduced techniques and operated with the assumption all nuisance traffic would appear under a single CLI. It cannot be reasonably assumed that an individual is capable of making hundreds of calls in a single day, such volume must come from an organisation or a robotic system. However, there is no technical limitation in spoofing traffic to multiple CLIs, outputting only a single or small amount of calls to each. This not only obfuscates the large call volumes indicative of call centres, but also deprives

nuisance call detection techniques of a reliable call history for the CLI and prevents the creation of the time series data we have used previously. Current approaches from literature and the time series we have displayed so far require this history of usage for a single CLI to make their predictions, and will struggle to find nuisance calls attempting to present as low volume consumer traffic. The term used for this continuous swapping of CLIs by nuisance callers is CLI rotation.

In this chapter, we experiment with two datasets containing CLI rotation. Both contain series of length 1440, capturing output from a CLI every minute for a single day, the same as the series shown in the previous chapter. However, this data includes series containing regular consumer traffic, and intentionally filters out series with a total call volume of 500 or more to remove large call centres. Effective tools are already available for detecting nuisance calling for high volume single CLIs, and we treat detecting CLI rotation as a separate problem. Other than the CLI being obfuscated, this data was provided without alterations and unnormalised. The first dataset is a single continuous range of CLIs, each with an accompanying call volume series. As this range only contains two examples of nuisance CLI rotation, we use our previous synthesised call volume examples to create a train set for detecting the spoofed CLI rotation traffic within the range. The second dataset contains multiple continuous ranges of CLIs taken from a variety of times and locations, each providing a multivariate time series including call duration data as well as call volume.

Given the sensitivity of the data used, all experiments were run on hardware provided by BT. Partially due to this and time constraints, we limit the algorithms tested to the better performing ones from our previous results. We also run the following experiments using TSC implementations from *sktime* to better fit into the workflow of the BT research team working on nuisance calling.

The remaining content of this chapter is structured as follows. Section 8.2 describes our initial look into the CLI rotation problem using the call volume CLI

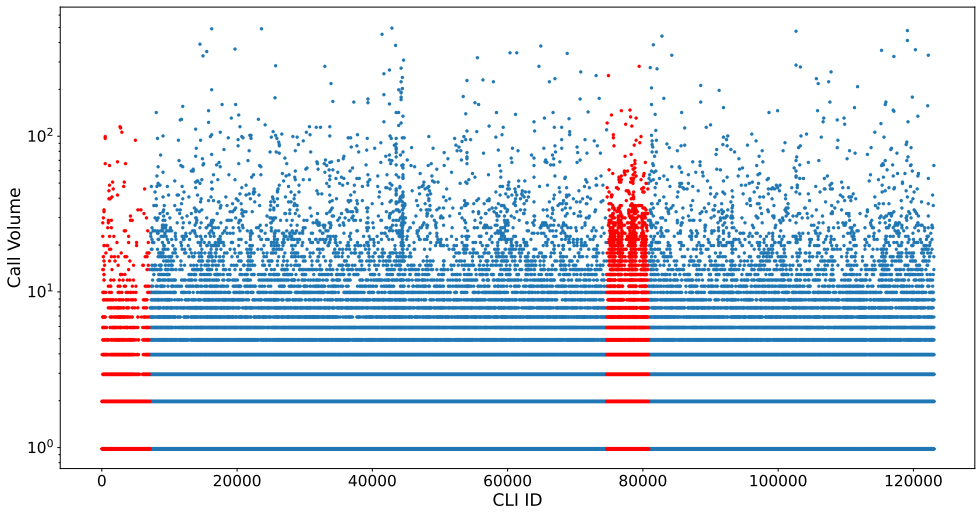


Fig. 8.1 A scatter diagram displaying daily call volume for CLIs which made at least one call in a continuous range. The CLIs have been anonymised using their index in the range, and areas labelled as containing suspicious traffic have been marked in red.

rotation dataset and introduces our system for detecting CLI rotation in a scalable fashion using TSC. Section 8.3 introduces an expanded CLI rotation data using call duration and presents results using call duration as a feature. Finally, conclusions are made in Section 8.4.

8.2 Detection using Call Volume Series with Synthesised Train Data

Our first dataset consists of 122,936 CLIs, of which 13,100 are believed to contain nuisance traffic in two separate continuous sub-ranges of CLIs. Unlike the previous synthesised data, almost all of the time series which can be derived from individual CLIs in this dataset would be unusable in detecting nuisance traffic due to the low call volume. Additionally, the computational overhead of having to process every low volume CLI sending traffic through the BT network immediately makes processing these individually unfeasible. A different approach is required to detect

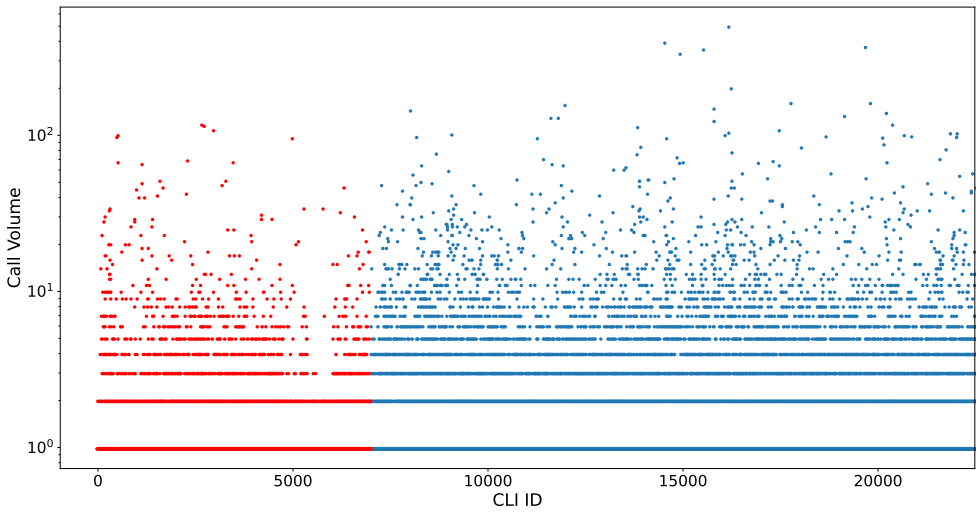


Fig. 8.2 A scatter diagram displaying daily call volume for CLIs which made at least one call in a continuous range, focused on the first 20,000 CLIs. An anomaly is present with a gap in high volume calls, but nuisance calls persist outside this in the ID range 0 to 7200.

this spread nuisance traffic. Figure 8.1 displays a scatter diagram displaying the total call volume over a single day for our anonymised CLIs. The call volume for each CLI is shown on a logarithmic scale, ranging from 1 to 500. Anomalies in this graph can show us where potential CLI rotation is taking place.

Shortening the range to make viewing these anomalies clearer, Figure 8.2 shows the volumes for the first 20,000 CLIs in our range. Around the CLIs with ID 5000 to 6000, there is a sudden gap in call volumes values above two for a short continuous range. It is believed that this lack of traffic occurs when nuisance call centres spoof their CLI to an invalid CLI in a range either not yet assigned by Ofcom, one which is off limits for normal traffic or otherwise lacking in regular consumer traffic. The callers make a small volume of calls on each CLI before switching, but the lack of higher volume traffic from legitimate callers makes the range stand out. Alongside the anomaly range containing nuisance calls, all the CLIs before it contain nuisance traffic which is hidden amongst the regular traffic.

Figure 8.3 shows another focused range, this one containing four separate but similar condensed blocks of calls around the same volume. This is unusual

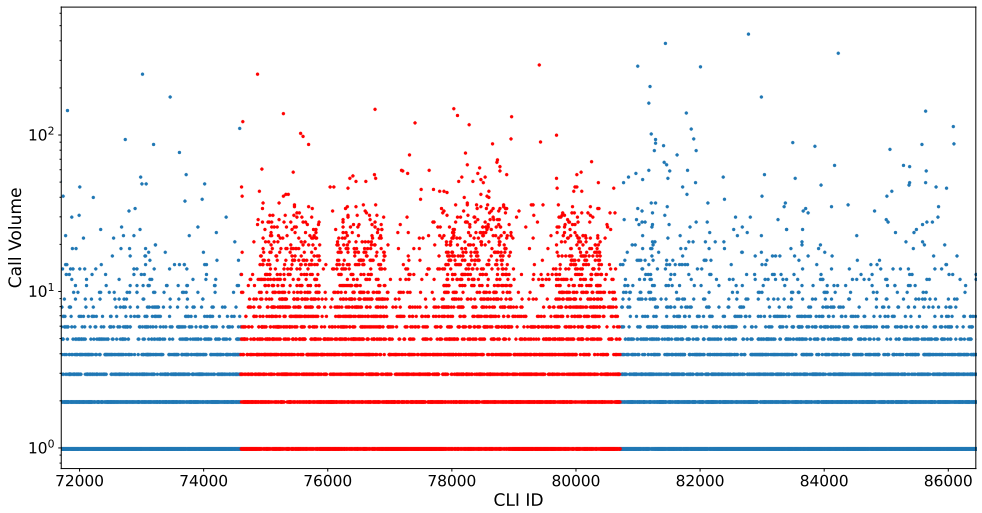
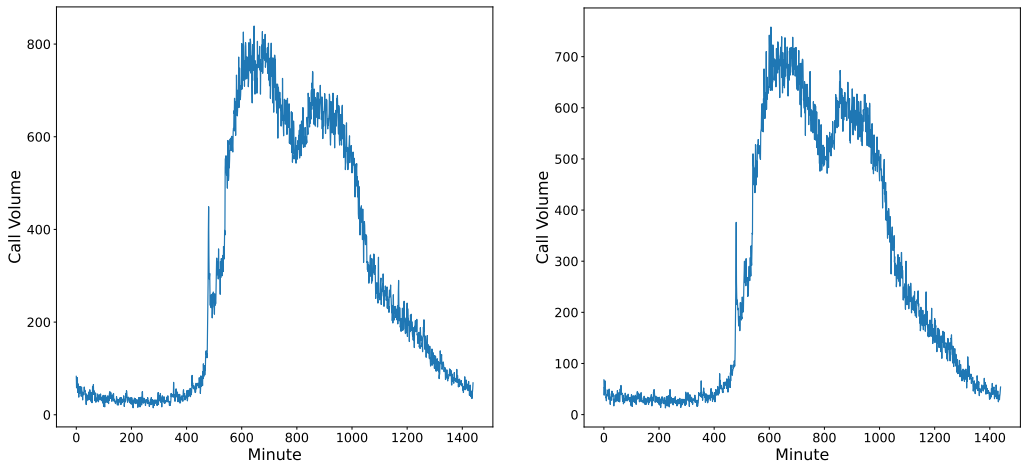


Fig. 8.3 A scatter diagram displaying daily call volume for CLIs which made at least one call in a continuous range, focused on the CLI IDs from 72,000 to 86,000. An anomaly is present with four dense clusters of high-volume calls between IDs 74,600 to 80,700.

behaviour, possibly from a system set up to make up to a certain number of calls before moving over to a new CLI in the block. While odd, it is difficult to determine if this anomaly is caused by nuisance callers, however. A characteristic of both these anomalies is they form a continuous range of numbers, either incrementing the CLI used by a small amount when swapping or selecting randomly from a short range.

Given the description of calling patterns, we provide a problem definition for detecting CLI rotation: Given a set of continuous CLI numbers, $C = \{c_1, c_2, \ldots, c_x\}$, we want to detect any continuous nuisance subsets as a result of CLI rotation $U = \{u_1, u_2, \ldots, u_y\}$ present in C, $CU = \{U_1, U_2, \ldots, U_p\}$, of which there can be singular, multiple or zero occurrences of nuisance continuous sub-ranges U. During this detection process, we want to make as many nuisance predictions on CLIs contained in CU as possible, while avoiding the legitimate traffic in $C - CU$.

Like our previous approach, we make use of TSC algorithms to detect this CLI rotation in our experiments. The previous definition and figures may give the idea that time series anomaly detection may be better suited for the problem. We don't



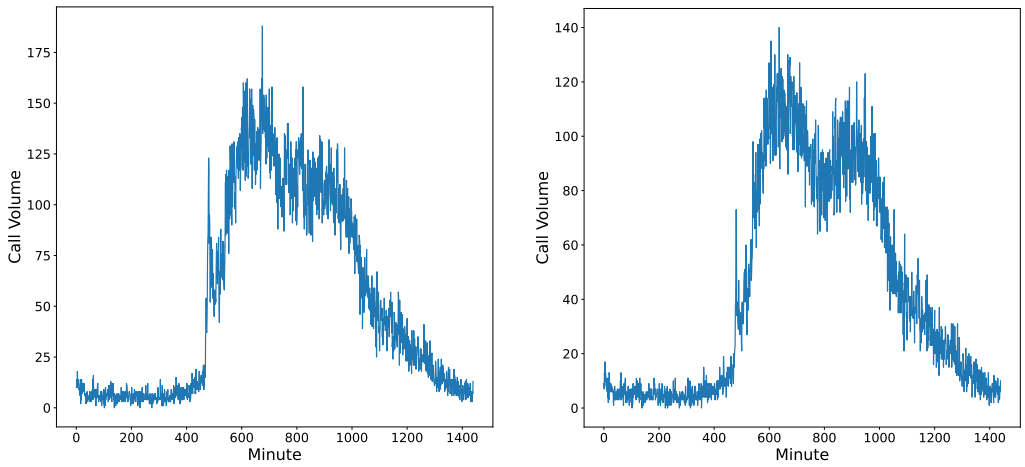
(a) All Series

(b) Nuisance Ranges Removed

Fig. 8.4 Call volume time series created by aggregating the series generated from the CLI range displayed in figure 8.1. Plot (a) aggregates all series, while (b) leaves out the nuisance CLI ranges.

discount that this may also be a valid approach, but we note that our example was selected due to its easily visible anomalies. Nuisance traffic can be much harder to detect like this when spread among legitimate traffic at low volumes, which our algorithm attempts to mitigate. Anomalies in call volumes from concurrent CLIs will not necessarily be the result of nuisance traffic as well. It is expected that multiple approaches will be required to stay on top of the ever-evolving landscape as nuisance callers adapt to prevention efforts.

While looking at information from individual CLI time series is useless in determining the presence of nuisance traffic using CLI rotation, in aggregate the volume of multiple CLIs can produce usable call centre profiles. Figure 8.4(a) shows the aggregated call volume per minute for all CLIs in our range, while Figure 8.4(b) removes CLIs with suspected nuisance traffic. The difference is subtle, with both series looking almost identical due to the high amounts of legitimate traffic. The peak in plot (a) is very slightly larger at the 480 time point from the nuisance calls contained within the traffic from CLIs 0 to 7200. Despite sudden jumps in traffic being an indicator of nuisance traffic, this peak occurs naturally in aggregated



(a) First 25,000 Series

(b) Nuisance Ranges Removed

Fig. 8.5 Call volume time series created by aggregating the series generated from the CLI range displayed in figure 8.1. Plot (a) aggregates the first 25,000 series, plot (b) does the same but leaves out the nuisance CLI range.

regular traffic. The jump is at 8AM in the morning, when many businesses and general practitioner surgeries open in the UK. These openings create a natural coordination in traffic from multiple individual consumers.

Figure 8.5 shows a smaller range of CLIs aggregated, using the call volumes from ID 0 to 25,000. As we decrease the amount of CLIs aggregated, the resulting series becomes noisier, but nuisance traffic is easier to pick out. The spike in calls at time point 480 is much more pronounced, and the dip in the centre of the series starts to disappear as the proportion of nuisance to legitimate traffic rises.

There is a trade-off between series quality and computation required when determining how many series to aggregate when creating our profiles. While it is easier to detect these nuisance profiles when aggregating a small number of CLIs, there are still billions of valid CLIs which are assigned or are available to be assigned by Ofcom. Even with aggregation, scanning all these number ranges is likely to be too computationally expensive if done using small amounts of CLIs. The call volume profile in Figure 8.4 may be difficult to differentiate by humans, but it is possible machine learning algorithms may be able to pick up these deviations

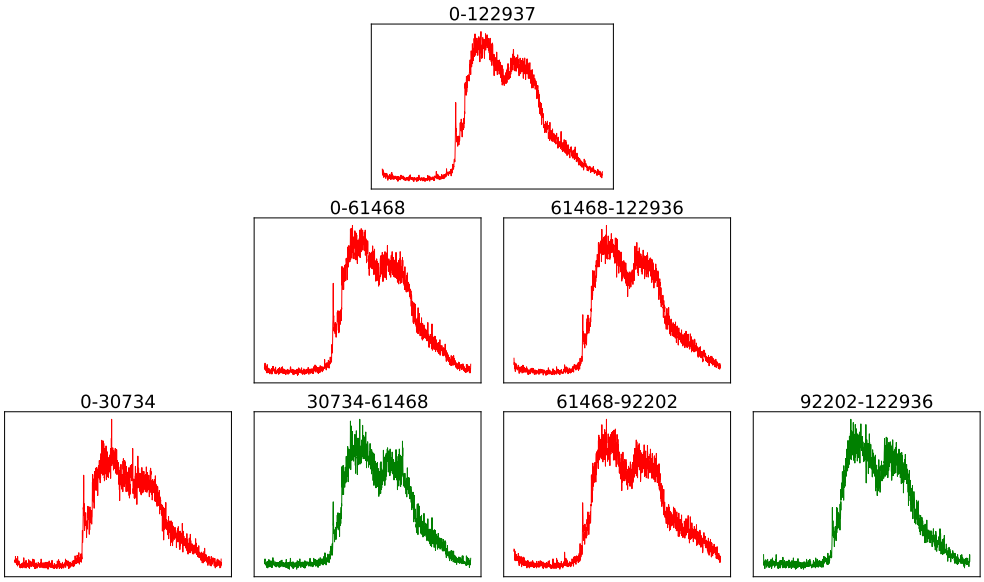


Fig. 8.6 Aggregated call volume time series for progressively halved CLI ranges. Series containing nuisance traffic are marked as red, while those without are green. As we keep halving the CLI range used to create the series, we filter out the ranges which do not contain nuisance traffic.

from regular traffic. It is expected that some nuisance callers will focus on spoofing as CLIs in certain areas to appear more legitimate, which will concentrate nuisance call to stand out in these large aggregations.

Our initial proposed strategy is to classify a larger range of aggregated CLIs, and slowly filter down to the range where nuisance traffic is located if the traffic is detected in the initial large range scan. Figure 8.6 shows aggregated series for progressively smaller ranges of CLIs in a pyramid-like structure. At each level, the CLI range is split into half, with red series containing nuisance traffic and green series not. We continue halving until either a CLI range has been classified as legitimate twice consecutively, or until the next split will cause the range to go below 2000 CLIs, which we deemed would produce a series too noisy to classify accurately. The final prediction for an aggregated series is used as the prediction for all CLIs in the range used to produce it, i.e., if we split down to the CLI ID range 0 to 2000 and still decide that this aggregated contains nuisance traffic, all CLI in the 0 to 2000 range will be predicted as nuisance. In this process, we can

Algorithm 8.1 nuisanceCLIPrediction(A list of n cases of length m, $T = (\boldsymbol{X}, \boldsymbol{y})$)

Parameters: The start point of the range to aggregate s, the end point of the range e, consecutive legitimate predictions cl, a trained time series classifier clf

Let $\boldsymbol{preds}$ be a list of n CLI label predictions initialised to 0, $(preds_1, \ldots, preds_n)$
$\boldsymbol{a} \leftarrow aggregateSeries(\mathbf{X}, s, e)$
$p \leftarrow clf.predictNuisanceProbability(a)$
if $p > 0.5$ **then**
  if $(e-s)/2 > 2000$ **then**
    nuisanceCLIPrediction(s, (s + e)/2, 0, clf)
    nuisanceCLIPrediction((s + e)/2, e, 0, clf)
  else
    for $i \leftarrow s$ to e **do**
      $preds_i \leftarrow 1$
else
  $cl \leftarrow cl + 1$
  if $(e-s)/2 > 2000$ **and** $cl < 2$ **then**
    nuisanceCLIPrediction(s, (s + e)/2, cl, clf)
    nuisanceCLIPrediction((s + e)/2, e, cl, clf)

continually filter out ranges which do not contain nuisance calls at a high level, while isolating the ranges that are detected to contain nuisance traffic by continually splitting. Pseudocode for predicting continuous ranges of CLIs in this manner is provided in Algorithm 8.1.

We evaluate seven algorithms for classifying the presence of nuisance traffic in these aggregated time series. Five of these are the stronger performing algorithms from our previous experiments in Catch22, DrCIF, FreshPRINCE, HC2 and STC. 1NN-DTW is included as a benchmark, and we attempt to use ROCKET as a generally accurate and fast performer again on this new data, despite poor performance in the first experiment. We perform a 4-fold cross-validation over our CLI range, where each fold is a quarter of the CLIs from our collection selected using the provided CLI IDs. Each fold is evenly split to contain a similar amount of CLIs, and the original ordering of CLIs is retained within each split. So for the first fold, data extracted from the CLIs with the ID 0 to 30,734 from the plot of 122,938 CLIs displayed in Figure 8.1 will be used to test our algorithms while

the remaining 92,202 CLIs are used to train the model. Our models are trained on series aggregated from the training set, and tested on series aggregated from the test set to make predictions on CLIs in the test range. Using the remaining training CLIs from the cross-validation, we would not be able to create a suitable dataset for training, however. There would only be a single nuisance profile in some folds, and both of these profiles present differently when aggregated. Additionally, naively creating training series using the halving process shown in Figure 8.6 on the full training fold range will produce series aggregated from different amounts of CLIs than what will be used in the test set.

We produce legitimate train CLI ranges by removing all CLIs marked as nuisance from the training folds. We then select continuous ranges from the concatenated train CLIs with a random start point with a length of 30,734. To introduce diversity to the train set to compensate from the limited pool of CLIs, the series is randomly shifted up to 30 time points to the left or right. While this gives us a CLI range to aggregate for legitimate profiles, we still require nuisance examples. To produce these, we re-use the synthesised nuisance call volume profiles from our earlier experiments in Chapter 7. For our nuisance CLI ranges, we repeat the process used to create our legitimate ranges, but add a random synthesised call centre profile. The nuisance profile is scaled to have a volume between 10% and 30% of the total volume from the legitimate range, and then each call is distributed in a continuous sub-range with a randomly chosen position in the generated legitimate CLIs and a randomly chosen size for the sub-range between 1000 and 5000.

We repeat this process 200 times to generate our training CLI ranges for each cross-validation fold. The first 100 labelled as nuisance containing the implanted synthesised nuisance call profiles, and the second 100 labelled as legitimate with no nuisance profiles added. These CLI ranges are then used to create the aggregated call volume time series to train our models. The halving process down to the minimum of 2000 CLIs is then performed for each CLI range, with a random

Table 8.1 Results on the CLI call volume dataset over a 4-fold cross-validation of CLI ranges. Displays classifier performance using Accuracy, Sensitivity (True Positive Rate) and Specificity (True Negative Rate).

Classifier	Accuracy	Specificity	Sensitivity
1NN-DTW	0.9448	**0.9938**	0.5344
catch22 [80]	**0.9812**	0.9792	**0.9982**
DrCIF [87]	0.9499	0.9442	**0.9982**
FreshPRINCE [84]	0.9499	0.9442	**0.9982**
HC2 [87]	**0.9812**	0.9792	**0.9982**
ROCKET [37]	0.8622	0.9300	0.2932
STC [17]	0.9499	0.9442	**0.9982**

aggregated time series of the correct label from each halving split being added to the train dataset. Meaning that for each CLI range, the training dataset will include four series, one for each of the following split sizes: 30,734, 15,367, 7683 and 3841. This allows us to include training cases created from different amount of CLI, matching what will be seen in the prediction series. Like the previous experiment, the aggregated series for both train and test sets are normalised both to help the process of implanting our nuisance example into the train series and prevent the amount of CLIs aggregated from impacting the decision due to scale.

8.2.1 Univariate Data Results

Table 8.1 shows the cross-validation prediction results for our TSC algorithms. The Catch22 features and HC2 perform identically, identifying both nuisance ranges, with only small parts of legitimate ranges misclassified when aggregated together with the ends of the nuisance ranges. Figure 8.7 shows the predictions made by Catch22 and HC2 on a scatter diagram of total call volumes for each CLI.

The next best performers: DrCIF, STC and FreshPRINCE also detected both nuisance ranges successfully, but also predicted the same legitimate range as nuisance. This range is located at CLI 111,389 to 115,230, and is displayed in the scatter diagram of Figure 8.8 marked in red. This range also has an anomaly like

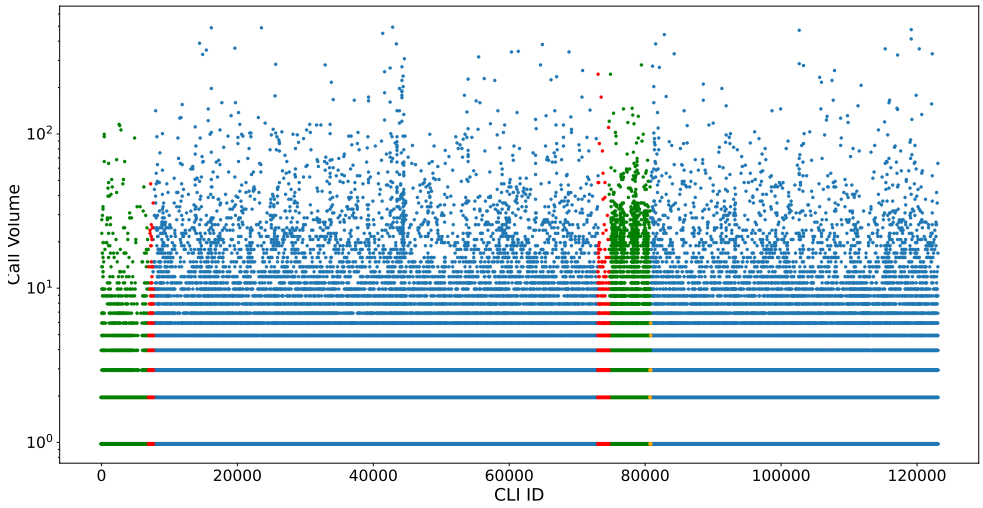


Fig. 8.7 A scatter diagram displaying daily call volume for CLIs which made at least one call in a continuous range. Green points are true positives, blue points true negatives, red points false positives and orange points false negatives.

the one shown in Figure 8.2 with a gap in high volume calls, but is much smaller. However, it is difficult to tell if this anomaly is caused by nuisance calling, however. Figure 8.9 shows the range aggregated (a) next to the previous correctly classified range for comparison (b). While there are small differences, such as the missing spike from the 8 A.M call surge, nothing about the aggregated series solidly stands out as a nuisance profile. There is little ground truth to the nuisance calling and CLI rotation problem, and it is hard to determine whether the range is actually of interest without knowing the contents of the calls made. Our later work with call duration in Section 8.3 suggests that this may be a correct classification by the three algorithms, as we decide to relabel this range as nuisance, however.

While in the previous experiment, misclassifying a legitimate call centre could have costly consequences if action is taken in real-time, There is very little real-time action that can be taken for CLI rotation. Blocking thousands of CLIs per nuisance call centre potentially in use by consumers is not a reasonable action to take, especially since the likelihood of these numbers being reused by the spoofing call centre is unlikely. HIVE-COTE 2.0 and Catch22 successfully capture both of our

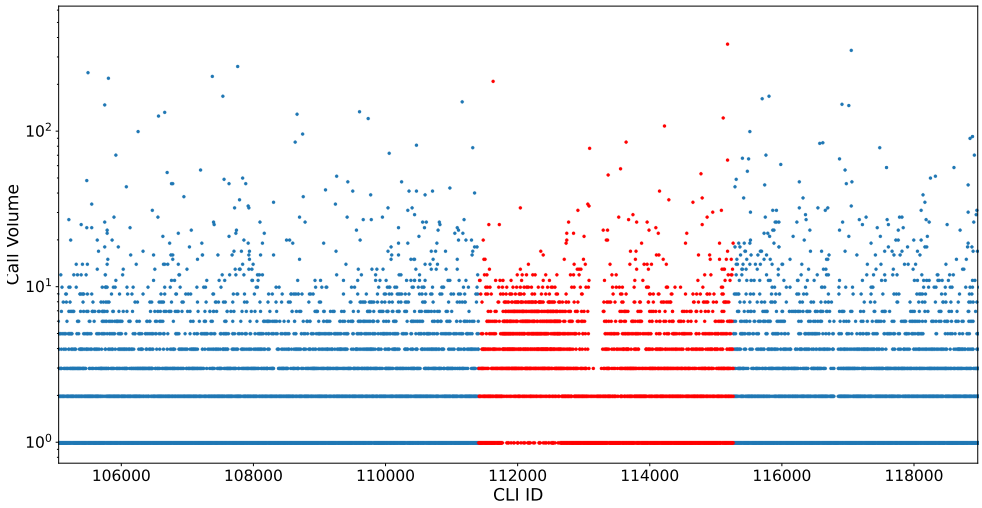
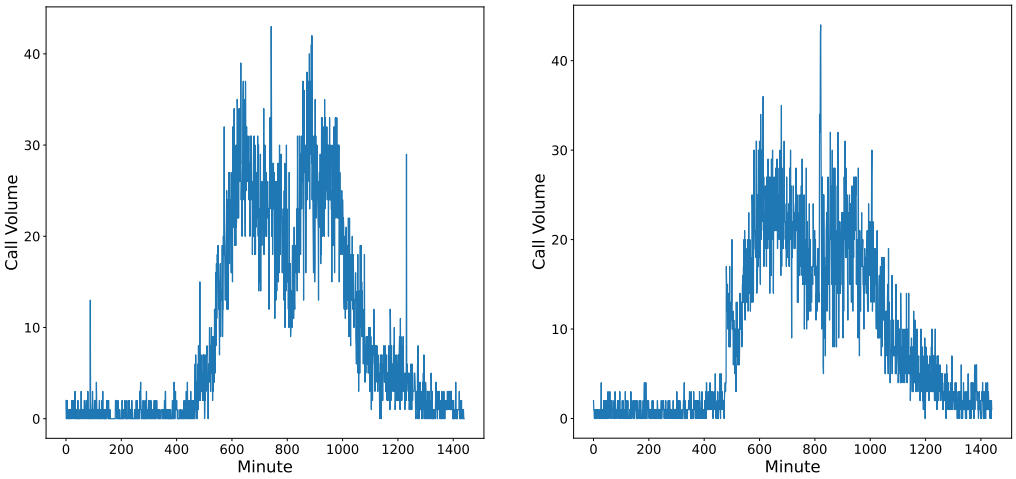


Fig. 8.8 A scatter diagram displaying daily call volume for CLIs which made at least one call in a continuous range, focused on the CLI IDs from 106,000 to 118,000. A small anomaly is present with a gap in high volume calls, which has caused a nuisance prediction.

marked nuisance CLI ranges, but what utility is there in this? While it is infeasible for organisations such as BT to identify and take action against international call centres, networks sending traffic into the BT network with unusually high levels of nuisance calling are within reach. By identifying the presence of nuisance calls, the source network of them can be discovered, and the carrier can be informed of its presence or action against the networks can be taken if necessary.

8.3 Detection using Multivariate Call Duration Series

So far in this chapter and Chapter 7, we have only used time series derived from the call volume of a CLI over the period of a day. While other nuisance call detection approaches are not applicable for time series data, the primary feature used by most is the duration of a call and historic call durations for the CLI [73, 23]. Call duration is also the basis of the Call Protect system used by BT currently, with the primary feature for detecting nuisance traffic being a large output of short duration calls.



(a) CLI 111389 to 115230

(b) CLI 107548 to 111389

Fig. 8.9 Call volume time series created by aggregating the series generated from the CLI range displayed in figure 8.1. Plot (a) aggregates CLI 111,389 to 115,230 which contains the anomaly shown in figure 8.8. Plot (b) aggregates the adjacent CLI 107,548 to 111,389 for comparison.

Intuitively this makes sense, while nuisance callers have control over the volume they output and the CLI they appear as, call duration is somewhat in the control of the receiver, who can terminate the call whenever they please. Consumers having this control over the duration is likely to result in a difference between nuisance call centres and legitimate centres, as it is more likely people still stay on the line for longer if the call is important or interesting, rather than just spam. In this section, we investigate call duration as a feature in addition to call volume, using both to create a multivariate time series for our experiments.

Only small amounts of work have been conducted previously using call duration time series at BT, with call volume being preferred for analysis. Like call volume, we can plot the average call duration in a continuous scatter diagram of CLIs to detect anomalies and label our CLI rotation data. Figure 8.10 mirrors Figure 8.2 on our previous range of CLIs, but instead displays average call duration on a logarithmic scale. Perhaps unsurprisingly, there is a large amount of variation in the real number call duration values compared to the natural number call volume.

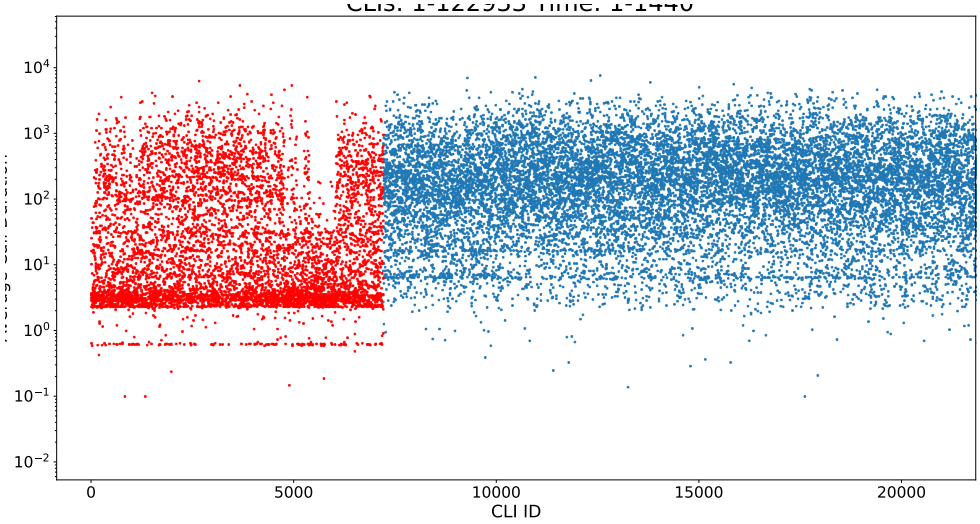


Fig. 8.10 A scatter diagram displaying the daily average call duration for CLIs which made at least one call in a continuous range, focused on the first 20,000 CLIs. Alongside the previous call volume anomaly, there is a clustering of low duration calls.

Additionally, our previously selected range containing nuisance traffic is even more visibly anomalous than the call volume plot. The dip in values remains, but two clusters of low volume durations have appeared in the 2-5 second and sub-zero second duration ranges. These clusters better reflect the full range of CLIs hijacked by the nuisance callers, while the call volume anomaly only captures a portion within the dip.

New information provided by the call duration in conjunction with call volume lead us to reassess labels for our range of CLIs. Figure 8.11 shows the call duration scatter diagram for our example range, focusing on the two remaining anomalies. Marked in red on the left the four blocks of high volume calls shown in Figure 8.3 and the small gap in high volume calls labelled as nuisance by some algorithms from Figure 8.8 is present on the right. Looking at the call duration, the labelled anomaly is almost indistinguishable from the surrounding regular traffic. The unlabelled rightmost one, however, is much more visible than it was using call volume. For later experiments, we change the labelling of this range to treat the current nuisance labelled section as legitimate and the smaller unlabelled anomaly

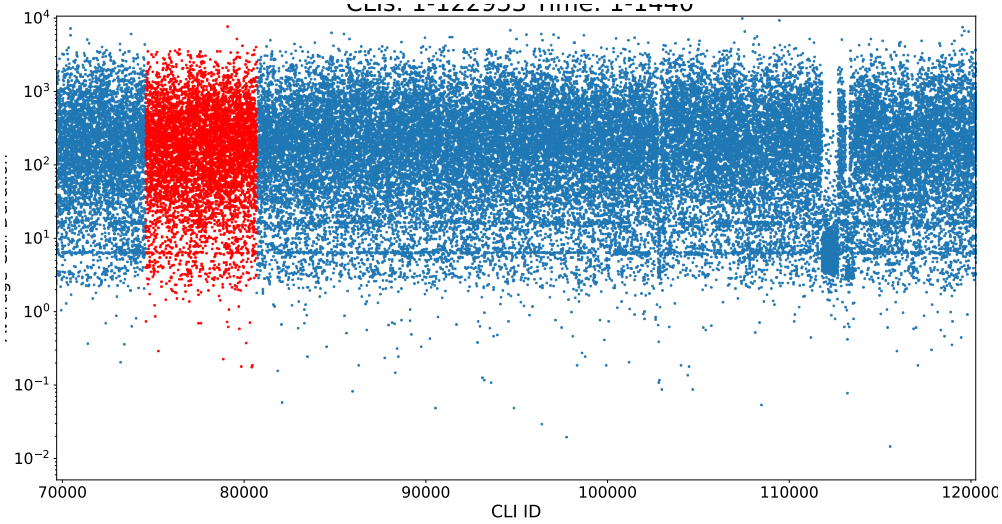


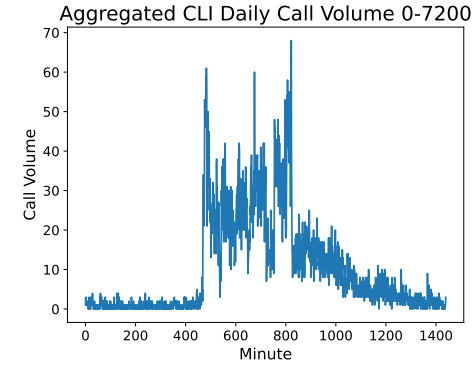
Fig. 8.11 A scatter diagram displaying the daily average call duration for CLIs which made at least one call in a continuous range, focused on the CLI IDs from 70,000 to 120,000. The area marked in red was previously labelled as nuisance.

as nuisance. While there certainly is an anomaly visible looking at call volume, it becomes more difficult to label it as containing nuisance traffic when paired together with call duration. It could be caused by actions such as telemetry or artificial inflation of traffic, which could cause anomalies like these in ranges of CLIs but are not the nuisance traffic we are attempting to detect in this work.

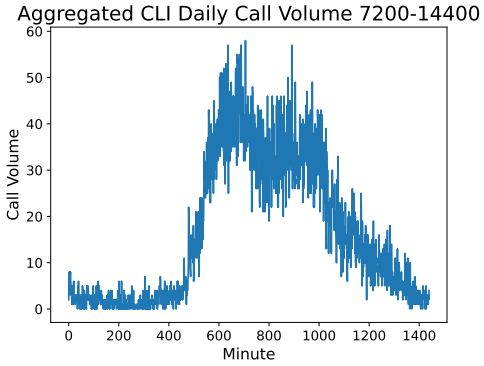
While the experience tackling nuisance calling from BT was used in the creation of our previous synthesised call centre profiles using call volume, there is currently no reference for legitimate and nuisance call duration profiles from which to synthesise data with. As such, more CLI ranges with nuisance CLI rotation examples are required to effectively evaluate call duration as a feature. This led to the creation of our larger multivariate CLI rotation dataset. Including the previously used CLI range, the new dataset is sourced from seven CLI ranges from different United Kingdom dialling codes. While the previous range consisted of weekday traffic only, three of the seven ranges in this dataset contain weekend traffic, which is expected to present differently as calling patterns change outside of typical working days. Data from the BT Call Protect blacklist suggests that nuisance calling significantly

drops on Saturdays, with an even further drop on Sundays. These ranges contain varying amounts of CLIs, to compare each of them we split the larger ranges into smaller folds. We split the ranges into a total of 16 folds, each containing 40,000 CLIs. Some folds contain no nuisance CLI rotation examples, while others contain multiple. Nine folds contain traffic from weekdays, and the remaining seven from the weekend. There are 14 nuisance sub-ranges, with the number of nuisance CLIs for all folds equalling 55,864 of 640,000 CLIs total.

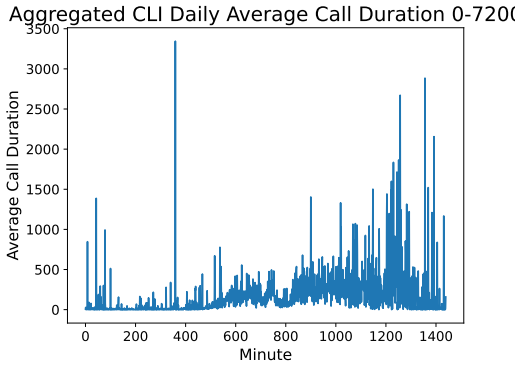
For each CLI, we have two values to create our time series, call volume per minute and the total call duration for that minute. We format this into three series, which make up our multivariate time series. The first is the daily call volume per minute seen previously. The second is the mean call duration for each minute in a day. Our final dimension attempts to capture the frequency of durations from CLIs, rather than the change in duration over time from an aggregated series. We capture this using a histogram of call durations for each CLI we are aggregating, where each bin width is a single second starting at 0 seconds and ending at 24 minutes. Figure 8.12 displays a nuisance and legitimate example series for each dimension in our multivariate dataset. The call volume series present like previous examples shown, except for the legitimate traffic remaining from where the callers have spoofed to in-use CLIs for the nuisance example. In our daily average call duration time series, there is a noticeable drop in call duration during the operating hours of the nuisance series compared to the legitimate example. We previously mentioned that call volume in legitimate call centres is likely to drop later in the day, as call durations become higher, leading to agents spending more time in calls rather than making new ones. This is shown in our duration time series, as the average duration rises as typical working hours end. For both our nuisance and legitimate histogram examples, the majority of calls are in the first 30 bins. This makes viewing the time series a little difficult, but there are noticeably more calls in bins 30-200 in the legitimate histogram compared to the nuisance one.



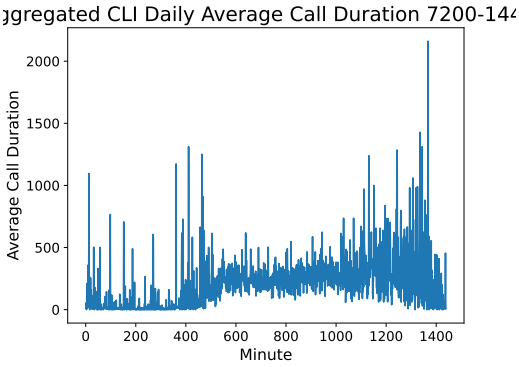
(A1) Nuisance Call Volume



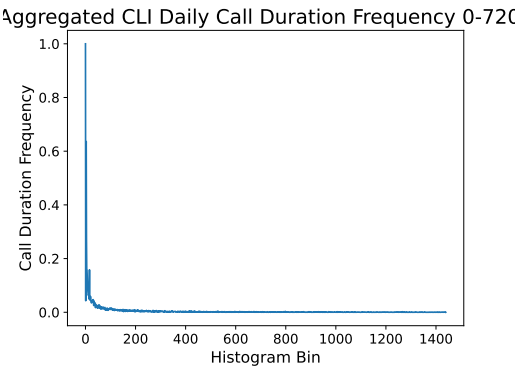
(A2) Legitimate Call Volume



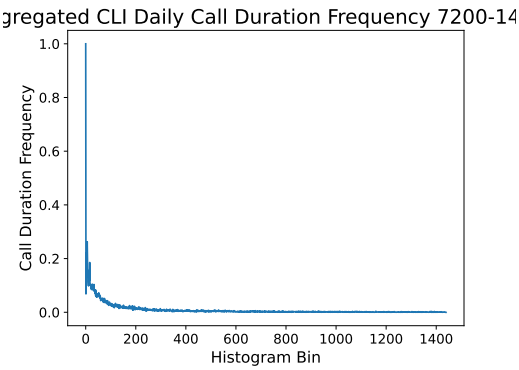
(B1) Nuisance Call Duration



(B2) Legitimate Call Duration



(C1) Nuisance Duration Histogram



(C2) Legitimate Duration Histogram

Fig. 8.12 Example time series for each dimension in the multivariate CLI rotation dataset. The nuisance examples are aggregated from the first 7200 CLIs in our example range, while the legitimate examples are aggregated from the proceeding CLIS at 7200 to 14,400.

Unlike previous experiments, this data was provided unnormalised and there is no necessity to normalise this data to add synthesised examples or further anonymise the data. In our classifier comparison, we leave the call volume time series unnormalised. While there is an argument that the scale of a call centre or the amount of CLIs we aggregate may be irrelevant, many of the classifiers we compare extract scale invariant features or normalise as part of their process. For our duration time series, there is the potential for large outliers where single calls last for hours. These large duration calls would have an extremely large impact on the series if normalised. Scale is less important for our histogram, where the proportion of each bin count compared to the largest still provides useful information. As such, we normalise the histogram to include some scale invariance to our time series for algorithms which don't make use of normalisation.

8.3.1 Multivariate Data Results

To provide our results, we run a cross-validation using our 16 folds of 40,000 CLIs each. Table 8.2 shows the result for our multi-level prediction process described in Algorithm 8.1. Like previously, aggregated series for each split size are included in the training dataset, providing a variety of legitimate to nuisance traffic ratios. With the larger and more diverse dataset, the best approaches can still correctly predict whether a CLI contains CLI rotation traffic or not. FreshPRINCE is the most accurate of our algorithms, and detects 9 out of 14 of our labelled CLI rotation ranges. Almost 7.5% of legitimate CLIs are misclassified as nuisance, but this mainly comes from two sources. As a result of our method of classification, the smallest grouping of CLIS that can be predicted at once is 2500. This results in the edges of some of our nuisance ranges including CLIs labelled as legitimate, where the CLI rotation does not neatly fall into our 2000 length windows. The

Table 8.2 Results on the multivariate CLI dataset over a 16-fold cross-validation of CLI ranges. Displays classifier performance using Accuracy, Sensitivity (True Positive Rate) and Specificity (True Negative Rate).

Classifier	Accuracy	Specificity	Sensitivity
1NN-DTW	0.8161	0.8636	0.3194
catch22	0.9048	**0.9443**	0.4919
DrCIF	0.8998	0.9202	0.6870
FreshPRINCE	**0.9076**	0.9245	0.7317
HC2	0.8853	0.9229	0.4919
STC	0.9030	0.9176	**0.7496**

Table 8.3 Results on the multivariate CLI dataset over a 16-fold cross-validation of CLI ranges using multiple models for each level of CLI aggregation. Displays classifier performance using Accuracy, Sensitivity (True Positive Rate) and Specificity (True Negative Rate).

Classifier	Accuracy	Specificity	Sensitivity
1NN-DTW	0.8696	0.9314	0.2227
catch22	0.9088	0.9422	**0.5595**
FreshPRINCE	**0.9247**	**0.9459**	0.4938
STC	0.9072	0.952	0.4383

second cause is in fold 13 of our cross-validation, where all of the 40,000 CLIs are classified nuisance, making up over 5% of our 7.5% mislabelled cases.

Our previous approach uses a single model for all levels in our prediction process. We investigate the impact of splitting each level, creating 5 separate models for each amount CLIs aggregated. While this will not negatively affect prediction efficiency, it comes with the overhead of having to train multiple models. The slowest of our classifiers in DrCIF and HC2 have been excluded due to this. The results for our remaining four algorithms are shown in Table 8.3. Having a separate model for each split makes the overall model more conservative, classifying only 7 of 14 nuisance ranges while reducing the amount of misclassified CLIs to 5.5%.

The primary purpose of our prediction process is to lower the amount of predictions made by filtering out ranges which show no clear signs of nuisance traffic at higher levels. As higher levels will include more legitimate traffic, it is highly likely that some nuisance call centres are lost in the noise of legitimate traffic. We

Table 8.4 Results on the multivariate CLI dataset over a 16-fold cross-validation of CLI ranges only aggregating 2500 CLIs. Displays classifier performance using Accuracy, Sensitivity (True Positive Rate) and Specificity (True Negative Rate).

Classifier	Accuracy	Specificity	Sensitivity
1NN-DTW	0.6183	0.5841	0.9765
catch22	0.9273	0.9235	0.9672
DrCIF	0.9468	0.9427	0.9896
FreshPRINCE	**0.9527**	**0.9513**	0.9672
HC2	0.9316	0.9258	**0.9921**
STC	0.8958	0.8891	0.9658

test how accurately our algorithms can pick out nuisance CLI rotation regardless of the amount of predictions required, discarding the multi-level process and immediately making a prediction using 2500 aggregated CLIs, the fewest amount of CLIs aggregated in our previous two experiments. For this experiment, the training set used is only comprised of series aggregated from 2500 CLIs, formed using a disjoint windowing of the train fold ranges. Table 8.4 displays the results for this simplified process. Sensitivity has improved significantly, with the most accurate algorithm, FreshPRINCE detecting all 14 nuisance call ranges. Like the multiple model experiment, including a single size when aggregating series has increased specificity.

Given the visible difference in durations seen when creating our previous scatter diagrams plotting the average duration for a single day for each CLI, it would be remiss if we did not attempt to create a simple benchmark using this information to compare our TSC approaches against. We create our benchmark by running a disjoint window of length l over a list of daily average call durations for each CLI in a range, which when plotted would look similar to Figure 8.10. We take the mean of each window, and compare it to a duration threshold t. If the window mean is less than the threshold, all CLIs in the window are predicted as nuisance, else they are predicted as legitimate. Pseudocode for this duration threshold is provided in Algorithm 8.2.

Algorithm 8.2 averageDurationWindowBenchmark(A list of n average call durations for a single day, $\boldsymbol{X}$)

Parameters: The window length l, the average call duration threshold t

Let $\boldsymbol{preds}$ be a list of n CLI label predictions initialised to 0, $(preds_1, \ldots, preds_n)$
for $i \leftarrow 0$ to $n/l - 1$ **do**
  $m \leftarrow 0$
  for $j \leftarrow i * l$ to $(i+1) * l$ **do**
    $m \leftarrow m + X_j$
  if $m/l < t$ **then**
    for $i \leftarrow i * l$ to $(i+1) * l$ **do**
      $preds_i \leftarrow 1$

Table 8.5 shows the results of our duration threshold benchmark for a variety of window lengths and threshold values. There are multiple accuracies above the best performing classifier FreshPRINCE from our previous experiment, and it goes without saying that the benchmark is quicker and more interpretable in its predictions than any TSC algorithm we have presented. While the TSC algorithms did not perform poorly, there is a scalability issue which is not present in our simple benchmark. Generally, a low mean threshold value resulted in a higher specificity, which is to be expected when nuisance callers have a lower call duration on average. The parameter offers a trade-off for specificity and sensitivity. A higher window length also resulted in higher specificity. As the groups of nuisance calling tend to be small, it is likely having a high window length will include more legitimate calls when grouped and increase the mean duration. The success of such a simple method over our previous machine learning approaches shows that outlier detection approaches may be better suited to the problem, given that the nuisance called remain in a sequential grouping of CLIs.

Table 8.5 Results on the multivariate CLI dataset for our average call duration benchmark. Displays classifier performance using Accuracy, Sensitivity (True Positive Rate) and Specificity (True Negative Rate).

Window Length	Mean Threshold	Accuracy	Specificity	Sensitivity
250	100	0.9628	0.9705	0.8821
250	150	0.9499	0.9508	0.9404
250	200	0.8418	0.8285	**0.9809**
500	100	**0.9671**	0.9776	0.8597
500	150	0.9535	0.9590	0.8960
500	200	0.8610	0.8506	0.9704
1000	100	0.9626	0.9820	0.7601
1000	150	0.9508	0.9558	0.8988
1000	200	0.878	0.8688	0.9735
2000	100	0.9608	**0.9852**	0.7048
2000	150	0.9403	0.9535	0.8026
2000	200	0.8903	0.8901	0.8920

8.4 Conclusions

In this chapter, we have investigated the nuisance call detection is the case of CLI rotation and explored the usage of TSC algorithms in re-aggregating continuous subranges of split nuisance call centre traffic hidden within regular traffic.

There is further work to be done for detecting CLI rotation, and our TSC system has noticeable areas for improvement. Our CLI rotation system includes unnecessary legitimate CLIs in its nuisance predictions, currently due to the constraint of using an aggregated series to classify individual CLIs. Some extra processing could be done to optimise the selected range once a general area has been decided as nuisance. Additionally, it is undetermined whether the multi-layer system classifying using high numbers of CLIs is necessary, which would be dependent on the TSC algorithm used. Our algorithms performed better using smaller numbers of aggregated CLIS, but this could require millions of predictions to be made a day, depending on the amount of numbers which must be processed. Rather than improving the TSC system, time may better be spent designing and improving a

simpler bespoke system based on call duration similar to our benchmark to avoid scalability issues, however.

A constraint of all CLI detection methods we have displayed is that the nuisance caller must select the CLIs they used sequentially. If a nuisance call centre was to randomly select CLIs, they would be much harder to detect. While the nuisance traffic could be detectable if a wide enough range of CLIs was aggregated, there would be little hope of finding the individual CLIs carrying that traffic. There are still possible methods to catch this traffic, however. While the traffic would still be spread out, filtering CLIs by the network they arrived from would greatly lower the range which has to be searched. It is potentially not as useful as finding the actual CLIs used, but determining that large amounts of nuisance traffic is coming in from a particular network could still be useful information.

Chapter 9

Conclusion

In this thesis, the initial research question in Chapter 1 was whether Time Series Classification (TSC) algorithms could be used to detect nuisance call centres, using time series data extracted with effort made to protect the privacy of consumers. Very little work has been done using time series data for the detection of nuisance calling, and research that does use time series data have not used dedicated TSC algorithms. The most accurate TSC algorithm at the start of this project, the Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) [78] would be too slow to feasibly use on such a problem, however. As such, our secondary research question was whether we can improve HIVE-COTE through updating the members of the ensemble which have fallen behind in performance and scalability. This resulted in the development of the improved HIVE-COTE 2.0 (HC2) algorithm.

In Chapter 7 we demonstrate that using time series data recording the number of call made per minute for a Calling Line Identification (CLI) number, we can perfectly separate nuisance and legitimate call centres using TSC algorithms. Simple summary statistics and vector based classification algorithms fell considerably behind TSC algorithms, showing the necessity of bespoke time series techniques for the problem. Using a set of Early TSC (eTSC) approaches, we have shown that the newly developed HC2 algorithm can make a classification at 10:03 A.M on

average given data from the same day, making a 68.06% reduction in nuisance calls while till classifying all call centres perfectly if a decision was to be made at that point.

Our initial investigation in the problem of CLI rotation in Chapter 8 shows TSC techniques can still detect aggregated traffic to reconstitute the nuisance call profile, but not perfectly. The requirement to aggregate large amounts of traffic also risks misclassifying legitimate CLIs which are caught up in the aggregation, and causes concerns when it comes to scalability.

The results of this thesis suggest that when it comes to accuracy, time series of call volumes are suitable for detecting nuisance call centres with the level of specificity required to ensure legitimate call centres are not impacted. Automatic action taken in real-time against detected CLIs is always going to be difficult to suggest, due to the high consequences of mislabelling a legitimate call centre. However, if the data provided by BT is accurate to the reality of nuisance calling, it is hard to imagine that better results when it comes to accurately making a classification on call centres can be produced.

For an automatic system making use of the TSC algorithms we present, more sophisticated experiments using a higher volume of data from a wider range of dates would be required to determine if TSC algorithms are indeed as accurate as reported in our opinion. It is still to be determined if the algorithms will remain accurate as calling patterns change over time. An investigation on how fast an algorithm must process its data to limit disruption should take place, with discussion on what actions should be taken if an input were to go over the threshold due to a hardware fault or unforeseen quirk of the data. While adding extra steps to the process of connecting a call from one customer to another is doable, it must be ensured that reliability is maintained and there is no disruption or significant delay added.

For systems where the stakes are lower, such as analysing CLI rotation or selecting CLIs for manual blacklist consideration after the fact, we are confident the

TSC methods we have displayed in this thesis will meet the accuracy and scalability requirements. A simple system filtering CLI numbers by daily total call volume would reduce the amount of processing required dramatically, and CLIs predicted to contain nuisance traffic for that day could be forwarded onto relevant teams at BT.

9.1 Discussion of Contributions

The first contribution made in this thesis was an improvement to the dictionary based representation of TSC algorithms with the introduction of cBOSS and TDE in Chapter 4. The cBOSS development improved the scalability of the BOSS HIVE-COTE component by an order of magnitude, and was introduced as a replacement for BOSS in version 1.0 of HIVE-COTE (HC1) [5]. This development was further extended into TDE, combining the key features of cBOSS, S-BOSS [67] and WEASEL [108] with a novel parameter selection method. TDE performed significantly better than other dictionary based approaches on 112 UCR datasets, while retaining the scalability of cBOSS for large datasets. In our analysis of TDE, we show that each of its components play an important part in achieving its accuracy, with the exception of bigram words. The Gaussian Process in particular was shown to still provide a significant improvement in accuracy even if the number of parameters randomly selected from the cBOSS ensemble structure is tripled.

With the dictionary based representation of HIVE-COTE replaced, the next representation we looked at was interval based algorithms in Chapter 5. Our initial improvements replace the features used by TSF [40] with the Catch22 set of features [80]. While accurate, this proves to scale poorly. The CIF algorithm introduces scalability improvements to adapt the Catch22 features to the ensemble of randomly selected intervals. When configured with speed in mind, CIF can perform as fast as TSF while performing significantly more accurately. DrCIF extends CIF,

introducing intervals extracted from a power spectrum and first order differences transformation of the base series. DrCIF was shown to perform significantly better than all current interval based approaches, while remaining as scalable as CIF.

The final TSC algorithm we contribute in this thesis is the upgraded HIVE-COTE 2.0, in Chapter 6. HC2 introduces TDE and DrCIF as components, as well as an iteration of ROCKET [37] called the Arsenal. We introduce Arsenal as a method of generating probability estimates to ROCKET, which is required for HIVE-COTE. HC2 is significantly more accurate than current state-of-the-art algorithms, including HC1. With the introduction of better scaling components and out-of-bag error for generating accuracy estimates, HC2 also scales better than HC1. An ablation study of components showed that all four components make an impact on performance, with any single removal resulting in a significantly worse classifier. An investigation of alternate ensemble schemes after our primary results shows that the CAWPE [68] ensemble scheme remains the best choice for HC2.

Compared to univariate TSC, multivariate is relatively undeveloped. We made an effort with our contributions to ensure they are capable of classifying multivariate series. We compared this capability using the recently introduced UEA multivariate archive [3], using 26 equal length datasets and benchmarking current multivariate techniques on the new archive. TDE, DrCIF and HC2 all performed better than the DTW benchmark on the archive, with HC2 performing significantly better than current multivariate approaches.

9.2 Future Work and Extensions

We have demonstrated that TSC approaches can accurately classify nuisance call centres. It remains to be seen how this work would in practice, especially concerning scalability, however. While a majority of CLIs could be discarded by setting a filter based on call volume, our eTSC experiments expect to see the time series at set

time points, and any CLI is unlikely to hit the threshold in the early hours of the day. The next step for evaluating the feasibility of quick interventions using eTSC would be to set up a prototype system to see how the system performs with this constraint and further investigate the amount of processing required for such a system. If making classification in real-time is deemed unfeasible, a similar prototype using regular TSC approaches could be developed to aid in the selection of CLIs for manual blacklist processing.

Our later CLI rotation experiments introduce call durations comprised of average call durations rather than call volume. Revisiting our Chapter 7 eTSC experiments with call duration series as a standalone or as part of a multivariate series with call volume could provide better results. As the data we used was simulated, further analysis into call duration to allow for the creation of synthesised examples would have to be done or a wider range of unsynthesised examples would have to be provided.

The focus of this thesis is primarily on classification, and our CLI rotation experiments were implemented with that in mind. As shown in the Chapter 8 scatter diagrams and the benchmark we compared against, another approach from a field such as anomaly detection could provide better results.

The algorithmic TSC advancements in this thesis were made with scalability in mind. More work can be done in this area, in particular for TDE where feature selection has shown promise and in a proper multivariate extension for STC, which in our experiments built an individual classifier on each dimension. More generally, lessons can be learned from ROCKET, which is extremely fast relative to other TSC approaches. The scalability of other algorithms could be improved if features of ROCKET such as dilation can be incorporated into other algorithms successfully.

HIVE-COTE is an evolving algorithm, where HC2 is the latest iteration. As algorithms individual feature representations continue to improve, so will HIVE-COTE. A notable missing representation from HC2 are distance based algorithms.

This exclusion is primarily due to scalability concerns, but if better scaling distance based algorithms are produced it could be reintroduced. As HIVE-COTE continues to evolve, care must be taken to avoid overfitting on the UCR and UEA archives. While large amounts of data and resamples are used currently, testing HC2 with an expanded archive with new datasets would alleviate these concerns. Currently, HIVE-COTE is only an algorithm for classification. Extensions for other machine learning fields such as regression, clustering and anomaly detection could be developed in the future.

References

[1] Angiulli, F. (2005). Fast condensed nearest neighbor rule. In *Proceedings of the 22nd international conference on Machine learning*, pages 25–32.

[2] Arul, M. and Kareem, A. (2021). Applications of shapelet transform to time series classification of earthquake, wind and wave data. *Engineering Structures*, 228:111564.

[3] Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. (2018a). The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*.

[4] Bagnall, A., Flynn, M., Large, J., Line, J., Bostrom, A., and Cawley, G. (2018b). Is rotation forest the best classifier for problems with continuous features? *arXiv preprint arXiv:1809.06705*.

[5] Bagnall, A., Flynn, M., Large, J., Lines, J., and Middlehurst, M. (2020). On the usage and performance of the hierarchical vote collective of transformation-based ensembles version 1.0 (hive-cote v1. 0). In *International Workshop on Advanced Analytics and Learning on Temporal Data*, pages 3–18. Springer.

[6] Bagnall, A. and Janacek, G. (2014). A run length transformation for discriminating between auto regressive time series. *Journal of classification*, 31(2):154–178.

[7] Bagnall, A., Large, J., and Middlehurst, M. (2019). A tale of two toolkits, report the second: bake off redux. chapter 1. dictionary based classifiers. *arXiv preprint arXiv:1911.12008*.

[8] Bagnall, A., Lines, J., Bostrom, A., Large, J., and Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660.

[9] Bagnall, A., Lines, J., Hills, J., and Bostrom, A. (2015). Time-series classification with cote: the collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535.

[10] Batista, G. E., Keogh, E. J., Tataw, O. M., and de Souza, V. (2014). Cid: an efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery*, 28(3):634–669.

[11] Baydogan, M. G. and Runger, G. (2016). Time series representation and similarity based on local autopatterns. *Data Mining and Knowledge Discovery*, 30(2):476–509.

[12] Baydogan, M. G., Runger, G., and Tuv, E. (2013). A bag-of-features framework to classify time series. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2796–2802.

[13] Benavoli, A., Corani, G., and Mangili, F. (2016). Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17(1):152–161.

[14] Bermejo Nievas, E., Deniz Suarez, O., Bueno García, G., and Sukthankar, R. (2011). Violence detection in video using computer vision techniques. In *International conference on Computer analysis of images and patterns*, pages 332–339. Springer.

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

[16] Bilenko, M., Basu, S., and Mooney, R. J. (2004). Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 11.

[17] Bostrom, A. and Bagnall, A. (2017). Binary shapelet transform for multiclass time series classification. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXII*, pages 24–46. Springer.

[18] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[19] Cabello, N., Naghizade, E., Qi, J., and Kulik, L. (2020). Fast and accurate time series classification through supervised interval search. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 948–953. IEEE.

[20] Cabello, N., Naghizade, E., Qi, J., and Kulik, L. (2021). Fast, accurate and interpretable time series classification through randomization. *arXiv preprint arXiv:2105.14876*.

[21] Caruana, R., Niculescu-Mizil, A., Crew, G., and Ksikes, A. (2004). Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18.

[22] CenturyLink (2018). Hosted voip: What it decision-makers really think.

[23] Chaisamran, N., Okuda, T., Blanc, G., and Yamaguchi, S. (2011). Trust-based voip spam detection based on call duration and human relationships. In *2011 IEEE/IPSJ 11th International Symposium On Applications and the Internet (SAINT)*, pages 451–456. IEEE.

[24] Chaovalitwongse, W. A., Prokopyev, O. A., and Pardalos, P. M. (2006). Electroencephalogram (eeg) time series classification: Applications in epilepsy. *Annals of Operations Research*, 148(1):227–250.

[25] Charity, S. D. (2013). Got their number: Ending the harm caused by nuisance calls and texts.

[26] Chen, H., Tang, F., Tino, P., and Yao, X. (2013). Model-based kernel for efficient time series analysis. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 392–400.

[27] Chen, L. and Ng, R. (2004). On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 792–803.

[28] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

[29] Christ, M., Braun, N., Neuffer, J., and Kempa-Liehr, A. W. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77.

[30] Cooper, G. F. and Herskovits, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347.

[31] Corduas, M. and Piccolo, D. (2008). Time series clustering and classification by the autoregressive metric. *Computational statistics & data analysis*, 52(4):1860–1872.

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

[33] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

[34] Cox, T. M., Ragen, T., Read, A., Vos, E., Baird, R. W., Balcomb, K., Barlow, J., Caldwell, J., Cranford, T., and Crum, L. (2006). Understanding the impacts of anthropogenic sound on beaked whales. Technical report, SPACE AND NAVAL WARFARE SYSTEMS CENTER SAN DIEGO CA.

[35] Dantu, R. and Kolan, P. (2005). Detecting spam in voip networks. *SRUTI*, 5:5–5.

[36] Dau, H. A., Bagnall, A., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., and Keogh, E. (2019). The ucr time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6):1293–1305.

[37] Dempster, A., Petitjean, F., and Webb, G. I. (2020). Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495.

[38] Dempster, A., Schmidt, D. F., and Webb, G. I. (2021). Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 248–257.

[39] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30.

[40] Deng, H., Runger, G., Tuv, E., and Vladimir, M. (2013). A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153.

[41] Dugan, P. J., Rice, A. N., Urazghildiiev, I. R., and Clark, C. W. (2010). North atlantic right whale acoustic signal processing: Part i. comparison of machine learning recognition algorithms. In *2010 IEEE Long Island Systems, Applications and Technology Conference*, pages 1–6. IEEE.

[42] Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., and Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963.

[43] Fawaz, H. I., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., Webb, G. I., Idoumghar, L., Muller, P.-A., and Petitjean, F. (2020). Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962.

[44] Flynn, M., Large, J., and Bagnall, T. (2019). The contract random interval spectral ensemble (c-rise): The effect of contracting a classifier on accuracy. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 381–392. Springer.

[45] Frank, E., Hall, M. A., and Witten, I. (2016). The weka workbench. online appendix. *Data mining: practical machine learning tools and techniques*.

[46] Freund, Y., Schapire, R. E., et al. (1996). Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer.

[47] Fulcher, B. D. and Jones, N. S. (2017). hctsa: A computational framework for automated time-series phenotyping using massive feature extraction. *Cell systems*, 5(5):527–531.

[48] Garcia, S. and Herrera, F. (2008). An extension on"statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons. *Journal of machine learning research*, 9(Dec):2677–2694.

[49] Gates, G. (1972). The reduced nearest neighbor rule (corresp.). *IEEE transactions on information theory*, 18(3):431–433.

[50] German Federal Network Agency (BNetzA) (2017). Annual report 2017, networks for the future.

[51] Gharghabi, S., Ding, Y., Yeh, C.-C. M., Kamgar, K., Ulanova, L., and Keogh, E. (2017). Matrix profile viii: domain agnostic online semantic segmentation at superhuman performance levels. In *2017 IEEE international conference on data mining (ICDM)*, pages 117–126. IEEE.

[52] Gharghabi, S., Imani, S., Bagnall, A., Darvishzadeh, A., and Keogh, E. (2018). Matrix profile xii: Mpdist: a novel time series distance measure to allow data mining in more challenging scenarios. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 965–970. IEEE.

[53] Gharghabi, S., Imani, S., Bagnall, A., Darvishzadeh, A., and Keogh, E. (2020). An ultra-fast time series distance measure to allow data mining in more complex real-world deployments. *Data Mining and Knowledge Discovery*, 34(4):1104–1135.

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

[55] Guijo-Rubio, D., Gutiérrez, P. A., Tavenard, R., and Bagnall, A. (2019). A hybrid approach to time series classification with shapelets. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 137–144. Springer.

[56] Guzella, T. S. and Caminhas, W. M. (2009). A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, 36(7):10206–10222.

[57] 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.

[58] Hills, J., Lines, J., Baranauskas, E., Mapp, J., and Bagnall, A. (2014). Classification of time series by shapelet transformation. *Data mining and knowledge discovery*, 28(4):851–881.

[59] Hirschberg, D. S. (1977). Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM)*, 24(4):664–675.

[60] Jeong, Y.-S., Jeong, M. K., and Omitaomu, O. A. (2011). Weighted dynamic time warping for time series classification. *Pattern recognition*, 44(9):2231–2240.

[61] Karlsson, I., Papapetrou, P., and Boström, H. (2016). Generalized random shapelet forests. *Data mining and knowledge discovery*, 30(5):1053–1085.

[62] Keogh, E. J. and Pazzani, M. J. (2001). Derivative dynamic time warping. In *Proceedings of the 2001 SIAM international conference on data mining*, pages 1–11. SIAM.

[63] Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada.

[64] Kraus, S. D., Brown, M. W., Caswell, H., Clark, C. W., Fujiwara, M., Hamilton, P. K., Kenney, R. D., Knowlton, A. R., Landry, S., Mayo, C. A., et al. (2005). North atlantic right whales in crisis. *Science*, 309(5734):561–562.

[65] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. In *proceedings of Advances in Neural Information Processing Systems 25*, pages 1097–1105.

[66] Lai, C.-C. (2007). An empirical study of three machine learning methods for spam filtering. *Knowledge-Based Systems*, 20(3):249–254.

[67] Large, J., Bagnall, A., Malinowski, S., and Tavenard, R. (2019a). On time series classification with dictionary-based classifiers. *Intelligent Data Analysis*, 23(5):1073–1089.

[68] Large, J., Lines, J., and Bagnall, A. (2019b). A probabilistic classifier ensemble weighting scheme based on cross-validated accuracy estimates. *Data mining and knowledge discovery*, 33(6):1674–1709.

[69] Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2169–2178. IEEE.

[70] Le Nguyen, T., Gsponer, S., and Ifrim, G. (2017). Time series classification by sequence learning in all-subsequence space. In *2017 IEEE 33rd international conference on data engineering (ICDE)*, pages 947–958. IEEE.

[71] Le Nguyen, T., Gsponer, S., Ilie, I., O'Reilly, M., and Ifrim, G. (2019). Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. *Data mining and knowledge discovery*, 33(4):1183–1222.

[72] Leontjeva, A., Goldszmidt, M., Xie, Y., Yu, F., and Abadi, M. (2013). Early security classification of skype users via machine learning. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 35–44. ACM.

[73] Li, H., Xu, X., Liu, C., Ren, T., Wu, K., Cao, X., Zhang, W., Yu, Y., and Song, D. (2018). A machine learning approach to prevent malicious calls over telephony networks. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 53–69. IEEE.

[74] Lin, J., Keogh, E., Wei, L., and Lonardi, S. (2007). Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144.

[75] Lin, J., Khade, R., and Li, Y. (2012). Rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, 39(2):287–315.

[76] Lines, J. and Bagnall, A. (2015). Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29(3):565–592.

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

[78] Lines, J., Taylor, S., and Bagnall, A. (2018). Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(5):52.

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

[80] Lubba, C. H., Sethi, S. S., Knaute, P., Schultz, S. R., Fulcher, B. D., and Jones, N. S. (2019). catch22: Canonical time-series characteristics. *Data Mining and Knowledge Discovery*, 33(6):1821–1852.

[81] Lucas, B., Shifaz, A., Pelletier, C., O'Neill, L., Zaidi, N., Goethals, B., Petitjean, F., and Webb, G. I. (2019). Proximity forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, 33(3):607–635.

[82] Maharaj, E. A. (2000). Cluster of time series. *Journal of Classification*, 17(2).

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

[84] Middlehurst, M. and Bagnall, A. (2022). The freshprince: A simple transformation based pipeline time series classifier. In *International Conference on Pattern Recognition and Artificial Intelligence*, pages 150–161. Springer.

[85] Middlehurst, M., Large, J., and Bagnall, A. (2020a). The canonical interval forest (cif) classifier for time series classification. In *2020 IEEE international conference on big data (big data)*, pages 188–195. IEEE.

[86] Middlehurst, M., Large, J., Cawley, G., and Bagnall, A. (2020b). The temporal dictionary ensemble (tde) classifier for time series classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 660–676. Springer.

[87] Middlehurst, M., Large, J., Flynn, M., Lines, J., Bostrom, A., and Bagnall, A. (2021). Hive-cote 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110(11):3211–3243.

[88] Middlehurst, M., Vickers, W., and Bagnall, A. (2019). Scalable dictionary classifiers for time series classification. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 11–19. Springer.

[89] Mietus, J., Peng, C., Henry, I., Goldsmith, R., and Goldberger, A. (2002). The pnnx files: re-examining a widely used heart rate variability measure. *Heart*, 88(4):378–380.

[90] Milne, C. (2015a). Appendix to combatting nuisance calls and texts, a comparative policy study of practices in multiple countries.

[91] Milne, C. (2015b). Combatting nuisance calls and texts, a comparative policy study of practices in multiple countries.

[92] Mori, U., Mendiburu, A., Dasgupta, S., and Lozano, J. A. (2017). Early classification of time series by simultaneously optimizing the accuracy and earliness. *IEEE transactions on neural networks and learning systems*, 29(10):4569–4578.

[93] Morrill, J., Fermanian, A., Kidger, P., and Lyons, T. (2020). A generalised signature method for multivariate time series feature extraction. *arXiv preprint arXiv:2006.00873*.

[94] Mueen, A., Keogh, E., and Young, N. (2011). Logical-shapelets: an expressive primitive for time series classification. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1154–1162.

[95] Oastler, G. and Lines, J. (2019). A significantly faster elastic-ensemble for time-series classification. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 446–453. Springer.

[96] Ordóñez, F. J. and Roggen, D. (2016). Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115.

[97] Potamitis, I. (2014). Classifying insects on the fly. *Ecological informatics*, 21:40–49.

[98] Provost, F. and Domingos, P. (2003). Tree induction for probability-based ranking. *Machine learning*, 52(3):199–215.

[99] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

[100] Quinlan, J. R. (1993). *C4. 5: Programs for Machine Learning*. Morgan Kaufmann.

[101] Quinlan, J. R. (1996). Improved use of continuous attributes in c4. 5. *Journal of artificial intelligence research*, 4:77–90.

[102] Ratanamahatana, C. A. and Keogh, E. (2005). Three myths about dynamic time warping data mining. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 506–510. SIAM.

[103] Rodriguez, J. J., Kuncheva, L. I., and Alonso, C. J. (2006). Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630.

[104] Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., and Bagnall, A. (2021). The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401–449.

[105] Schäfer, P. (2015). The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530.

[106] Schäfer, P. (2016). Scalable time series classification. *Data Mining and Knowledge Discovery*, 30(5):1273–1298.

[107] Schäfer, P. and Högqvist, M. (2012). Sfa: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 516–527.

[108] Schäfer, P. and Leser, U. (2017a). Fast and accurate time series classification with weasel. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 637–646. ACM.

[109] Schäfer, P. and Leser, U. (2017b). Multivariate time series classification with weasel+ muse. *arXiv preprint arXiv:1711.11343*.

[110] Schäfer, P. and Leser, U. (2020). Teaser: early and accurate time series classification. *Data mining and knowledge discovery*, 34(5):1336–1362.

[111] Schmitt, M., Ringeval, F., and Schuller, B. (2016). At the border of acoustics and linguistics: Bag-of-audio-words for the recognition of emotions in speech.

[112] Senin, P. and Malinchik, S. (2013). Sax-vsm: Interpretable time series classification using sax and vector space model. In *2013 IEEE 13th international conference on data mining*, pages 1175–1180. IEEE.

[113] Shamir, L., Yerby, C., Simpson, R., von Benda-Beckmann, A. M., Tyack, P., Samarra, F., Miller, P., and Wallin, J. (2014). Classification of large acoustic datasets using machine learning and crowdsourcing: Application to whale calls. *The Journal of the Acoustical Society of America*, 135(2):953–962.

[114] Shifaz, A., Pelletier, C., Petitjean, F., and Webb, G. I. (2020). Ts-chief: a scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery*, 34(3):742–775.

[115] Shokoohi-Yekta, M., Hu, B., Jin, H., Wang, J., and Keogh, E. (2017). Generalizing dtw to the multi-dimensional case requires an adaptive approach. *Data mining and knowledge discovery*, 31(1):1–31.

[116] Sikka, K., Wu, T., Susskind, J., and Bartlett, M. (2012). Exploring bag of words architectures in the facial expression domain. In *European Conference on Computer Vision*, pages 250–259. Springer.

[117] Smyth, P. (1996). Clustering sequences with hidden markov models. *Advances in neural information processing systems*, 9.

[118] Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959.

[119] Souza, V. M. (2018). Asphalt pavement classification using smartphone accelerometer and complexity invariant distance. *Engineering Applications of Artificial Intelligence*, 74:198–211.

[120] Souza, V. M., Cherman, E. A., Rossi, R. G., and Souza, R. A. (2017). Towards automatic evaluation of asphalt irregularity using smartphone's sensors. In *International Symposium on Intelligent Data Analysis*, pages 322–333. Springer.

[121] Stefan, A., Athitsos, V., and Das, G. (2012). The move-split-merge metric for time series. *IEEE transactions on Knowledge and Data Engineering*, 25(6):1425–1438.

[122] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[123] Tan, C. W., Dempster, A., Bergmeir, C., and Webb, G. I. (2022). Multirocket: Multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, pages 1–24.

[124] Tan, C. W., Herrmann, M., Forestier, G., Webb, G. I., and Petitjean, F. (2018). Efficient search of the best warping window for dynamic time warping. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 225–233. SIAM.

[125] Tan, C. W., Petitjean, F., and Webb, G. I. (2020). Fastee: Fast ensembles of elastic distances for time series classification. *Data Mining and Knowledge Discovery*, 34(1):231–272.

[126] Tanisaro, P. and Heidemann, G. (2016). Time series classification using time warping invariant echo state networks. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 831–836. IEEE.

[127] Trivedi, S. K. (2016). A study of machine learning classifiers for spam detection. In *Computational and Business Intelligence (ISCBI), 2016 4th International Symposium on*, pages 176–180. IEEE.

[128] United Kingdom Department for Digital; Culture; Media; & Sport (DCMS) (2014). Nuisance calls action plan.

[129] United Kingdom Office of Communications (Ofcom) and GfK UK (2019). Landline nuisance calls wave 6.

[130] United Kingdom Office of Communications (Ofcom) and Information Commissioner's Office (ICO) (2018). Nuisance calls and messages, update to ICO-Ofcom joint action plan.

[131] United Kingdom Office of Communications (Ofcom) and Information Commissioner's Office (ICO) (2021). Nuisance calls and messages, update to ICO-Ofcom joint action plan.

[132] United States Federal Trade Commission (FTC) (2017). Biennial report to congress under the do not call registry fee extension act of 2007.

[133] Wang, X., Wirth, A., and Wang, L. (2007). Structure-based statistical features and multivariate time series clustering. In *Seventh IEEE international conference on data mining (ICDM 2007)*, pages 351–360. IEEE.

[134] Wang, Z., Yan, W., and Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE.

[135] Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

[136] Wu, Y.-S., Bagchi, S., Singh, N., and Wita, R. (2009). Spam detection in voice-over-ip calls through semi-supervised clustering. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 307–316. IEEE.

[137] Xi, X., Keogh, E., Shelton, C., Wei, L., and Ratanamahatana, C. A. (2006). Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, pages 1033–1040.

[138] Xing, Z., Pei, J., and Yu, P. S. (2012). Early classification on time series. *Knowledge and information systems*, 31(1):105–127.

[139] Ye, L. and Keogh, E. (2011). Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data mining and knowledge discovery*, 22(1):149–182.

[140] Yeh, C.-C. M., Kavantzas, N., and Keogh, E. (2017). Matrix profile vi: Meaningful multidimensional motif discovery. In *2017 IEEE international conference on data mining (ICDM)*, pages 565–574. IEEE.

[141] Yeh, C.-C. M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H. A., Silva, D. F., Mueen, A., and Keogh, E. (2016). Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1317–1322. Ieee.

[142] Zhang, X., Gao, Y., Lin, J., and Lu, C.-T. (2020). Tapnet: Multivariate time series classification with attentional prototypical network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6845–6852.

[143] Zhao, J. and Itti, L. (2018). shapedtw: Shape dynamic time warping. *Pattern Recognition*, 74:171–184.

Appendix A

Archive Datasets

In this appendix, we provide individual dataset information about the archives primarily used in our experimentation for those interested. Details of all 112 datasets from the UCR archive [36] used in our experiments in Table A.1. Table A.2 contains details for the 26 datasets used from the UEA archive [3]. Further information and descriptions on individual datasets can be found on the time series classification dataset webpage[1]. This page also include download links to all the archive datasets we have used, as well as newer datasets and those with missing values or unequal length series.

[1] http://www.timeseriesclassification.com/dataset.php

Table A.1 The 112 equal length datasets without missing values from the UCR time series archive [36] used in our experimentation.

Name	#Train	#Test	#Classes	Length	Type
ACSF1	100	100	10	1460	Device
Adiac	390	391	37	176	Image
ArrowHead	36	175	3	251	Image
Beef	30	30	5	470	Spectrum
BeetleFly	20	20	2	512	Image
BirdChicken	20	20	2	512	Image
BME	30	150	3	128	Simulated
Car	60	60	4	577	Sensor
CBF	30	900	3	128	Simulated
Chinatown	20	345	2	24	Sensor
ChlorineConcentration	467	3840	3	166	Sensor
CinCECGTorso	40	1380	4	1639	Sensor
Coffee	28	28	2	286	Spectrum
Computers	250	250	2	720	Device
CricketX	390	390	12	300	Motion
CricketY	390	390	12	300	Motion
CricketZ	390	390	12	300	Motion
Crop	7200	16800	24	46	Image
DiatomSizeReduction	16	306	4	345	Image
DistalPhalanxOutlineAgeGroup	400	139	3	80	Image
DistalPhalanxOutlineCorrect	600	276	2	80	Image
DistalPhalanxTW	400	139	6	80	Image
Earthquakes	322	139	2	512	Sensor
ECG200	100	100	2	96	Biosignal
ECG5000	500	4500	5	140	Biosignal
ECGFiveDays	23	861	2	136	Biosignal
ElectricDevices	8926	7711	7	96	Device
EOGHorizontalSignal	362	362	12	1250	Biosignal
EOGVerticalSignal	362	362	12	1250	Biosignal
EthanolLevel	504	500	4	1751	Spectrum
FaceAll	560	1690	14	131	Image
FaceFour	24	88	4	350	Image
FacesUCR	200	2050	14	131	Image
FiftyWords	450	455	50	270	Image
Fish	175	175	7	463	Image
FordA	3601	1320	2	500	Sensor
FordB	3636	810	2	500	Sensor
FreezerRegularTrain	150	2850	2	301	Sensor
FreezerSmallTrain	28	2850	2	301	Sensor
GunPoint	50	150	2	150	Motion
GunPointAgeSpan	135	316	2	150	Motion
GunPointMaleVersusFemale	135	316	2	150	Motion
GunPointOldVersusYoung	135	316	2	150	Motion
Ham	109	105	2	431	Spectrum
HandOutlines	1000	370	2	2709	Image
Haptics	155	308	5	1092	Motion
Herring	64	64	2	512	Image
HouseTwenty	34	101	2	2000	Device
InlineSkate	100	550	7	1882	Motion
InsectEPGRegularTrain	62	249	3	601	Biosignal
InsectEPGSmallTrain	17	249	3	601	Biosignal
InsectWingbeatSound	220	1980	11	256	Sensor
ItalyPowerDemand	67	1029	2	24	Sensor
LargeKitchenAppliances	375	375	3	720	Device
Lightning2	60	61	2	637	Sensor

Lightning7	70	73	7	319	Sensor
Mallat	55	2345	8	1024	Simulated
Meat	60	60	3	448	Spectrum
MedicalImages	381	760	10	99	Image
MiddlePhalanxOutlineAgeGroup	400	154	3	80	Image
MiddlePhalanxOutlineCorrect	600	291	2	80	Image
MiddlePhalanxTW	399	154	6	80	Image
MixedShapes	500	2425	5	1024	Image
MixedShapesSmallTrain	100	2425	5	1024	Image
MoteStrain	20	1252	2	84	Sensor
NonInvasiveFetalECGThorax1	1800	1965	42	750	Biosignal
NonInvasiveFetalECGThorax2	1800	1965	42	750	Biosignal
OliveOil	30	30	4	570	Spectrum
OSULeaf	200	242	6	427	Image
PhalangesOutlinesCorrect	1800	858	2	80	Image
Phoneme	214	1896	39	1024	Sensor
PigAirwayPressure	104	208	52	2000	Biosignal
PigArtPressure	104	208	52	2000	Biosignal
PigCVP	104	208	52	2000	Biosignal
Plane	105	105	7	144	Sensor
PowerCons	180	180	2	144	Device
ProximalPhalanxOutlineAgeGroup	400	205	3	80	Image
ProximalPhalanxOutlineCorrect	600	291	2	80	Image
ProximalPhalanxTW	400	205	6	80	Image
RefrigerationDevices	375	375	3	720	Device
Rock	20	50	4	2844	Spectrum
ScreenType	375	375	3	720	Device
SemgHandGenderCh2	300	600	2	1500	Spectrum
SemgHandMovementCh2	450	450	6	1500	Spectrum
SemgHandSubjectCh2	450	450	5	1500	Spectrum
ShapeletSim	20	180	2	500	Simulated
ShapesAll	600	600	60	512	Image
SmallKitchenAppliances	375	375	3	720	Device
SmoothSubspace	150	150	3	15	Simulated
SonyAIBORobotSurface1	20	601	2	70	Sensor
SonyAIBORobotSurface2	27	953	2	65	Sensor
StarLightCurves	1000	8236	3	1024	Sensor
Strawberry	613	370	2	235	Spectro
SwedishLeaf	500	625	15	128	Image
Symbols	25	995	6	398	Image
SyntheticControl	300	300	6	60	Simulated
ToeSegmentation1	40	228	2	277	Motion
ToeSegmentation2	36	130	2	343	Motion
Trace	100	100	4	275	Sensor
TwoLeadECG	23	1139	2	82	Biosignal
TwoPatterns	1000	4000	4	128	Simulated
UMD	36	144	3	150	Simulated
UWaveGestureLibraryAll	896	3582	8	945	Motion
UWaveGestureLibraryX	896	3582	8	315	Motion
UWaveGestureLibraryY	896	3582	8	315	Motion
UWaveGestureLibraryZ	896	3582	8	315	Motion
Wafer	1000	6164	2	152	Sensor
Wine	57	54	2	234	Spectrum
WordSynonyms	267	638	25	270	Image
Worms	181	77	5	900	Motion
WormsTwoClass	181	77	2	900	Motion
Yoga	300	3000	2	426	Image

Table A.2 The 26 equal length datasets without missing values from the UEA multivariate time series archive [3] used in our experimentation.

Dataset	#Train	#Test	#Classes	Length	#Dimensions	Type
ArticularyWordRecognition	275	300	25	144	9	Motion
AtrialFibrillation	15	15	3	640	2	Biosignal
BasicMotions	40	40	4	100	6	Motion
Cricket	108	72	12	1197	6	Motion
DuckDuckGeese	50	50	5	270	1345	Audio
EigenWorms	128	131	5	17984	6	Motion
Epilepsy	137	138	4	206	3	Motion
ERing	30	270	6	65	4	Motion
EthanolConcentration	261	263	4	1751	3	Other
FaceDetection	5890	3524	2	62	144	Biosignal
FingerMovements	316	100	2	50	28	Biosignal
HandMovementDirection	160	74	4	400	10	Biosignal
Handwriting	150	850	26	152	3	Motion
Heartbeat	204	205	2	405	61	Audio
Libras	180	180	15	45	2	Motion
LSST	2459	2466	14	36	6	Other
MotorImagery	278	100	2	3000	64	Biosignal
NATOPS	180	180	6	51	24	Motion
PEMS-SF	267	173	7	144	963	Other
PenDigits	7494	3498	10	8	2	Motion
PhonemeSpectra	3315	3353	39	217	11	Audio
RacketSports	151	152	4	30	6	Motion
SelfRegulationSCP1	268	293	2	896	6	Biosignal
SelfRegulationSCP2	200	180	2	1152	7	Biosignal
StandWalkJump	12	15	3	2500	4	Biosignal
UWaveGestureLibrary	120	320	8	315	3	Motion

Appendix B

Dictionary Based Results

We present the averaged scores for our Chapter 4 experiments on the UCR [36] and UEA [3] archives in this appendix. Table B.1 presents the results from univariate experiments in Section 4.4. Results for multivariate experiments are contained in Table B.2. Table B.3 contains scores for experiments in Section 4.5. Table B.4 provides the 16 datasets BOSS is slow to build on from our cBOSS experiments. Further results, figures and spreadsheet files are available on the webpage accompanying the TDE publication[1].

[1] http://timeseriesclassification.com/TDE.php

Table B.1 Average scores from univariate experiments presented in section 4.4. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

Name	Accuracy	BalAcc	AUROC	NLL
Section 4.4.1				
cBOSS	0.8349	0.8084	0.9406	0.7016
BOSS	0.8382	0.8169	0.9358	**0.6863**
RBOSS	0.7998	0.7756	0.9414	0.8453
WEASEL	**0.8478**	**0.8248**	**0.9477**	0.9204
Section 4.4.2				
TDE	**0.8589**	**0.8343**	**0.9545**	**0.5968**
cS-BOSS	0.8373	0.8114	0.9422	0.6882
cBOSS	0.8349	0.8084	0.9406	0.7016
WEASEL	0.8478	0.8248	0.9477	0.9204
BOSS	0.8382	0.8169	0.9358	0.6863
Section 4.4.2 (106 UCR datasets)				
TDE	**0.8574**	**0.8323**	**0.9530**	**0.6010**
cS-BOSS	0.8351	0.8085	0.9404	0.6946
cBOSS	0.8325	0.8054	0.9387	0.7085
WEASEL	0.8431	0.8192	0.9455	0.9471
S-BOSS	0.8423	0.8211	0.9343	0.6614
BOSS	0.8358	0.8141	0.9337	0.6941

Table B.2 Average scores from multivariate experiments presented in section 4.4. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

Name	Accuracy	BalAcc	AUROC	NLL
Section 4.4.3				
TDE	**0.7021**	**0.6818**	**0.8231**	**1.0461**
cBOSS$_I$	0.6639	0.6391	0.8125	1.3460
DTW$_I$	0.6035	0.5931	0.6531	2.6330
DTW$_D$	0.6626	0.6488	0.6915	2.2404
Section 4.4.3 (20 UEA datasets)				
TDE	0.7345	0.7111	0.8370	**0.9301**
MUSE	**0.7568**	**0.7381**	**0.8551**	0.9429
cBOSS$_I$	0.6998	0.6751	0.8191	1.2383
DTW$_I$	0.6520	0.6399	0.6931	2.3108
DTW$_D$	0.7056	0.6932	0.7277	1.9551

Table B.3 Average scores from experiments presented in section 4.5. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

Name	Accuracy	BalAcc	AUROC	NLL
Section 4.5.1				
TDE	**0.8589**	**0.8343**	**0.9545**	**0.5968**
TDE-Bigrams	0.8576	0.8324	0.9541	0.6055
TDE-HI	0.8466	0.8212	0.9500	0.6379
TDE-Spatial	0.8509	0.8259	0.9517	0.6272
TDE-IGB	0.8463	0.8213	0.9495	0.6327
TDE-GP	0.8466	0.8209	0.9499	0.6659
Section 4.5.2				
TDE	**0.8589**	**0.8343**	**0.9545**	**0.5968**
cS-BOSS	0.8373	0.8114	0.9422	0.6882
GP-cS-BOSS	0.8451	0.8196	0.9445	0.6383
TDE-NoGP	0.8466	0.8209	0.9499	0.6659
TDE-NoGP750	0.8537	0.8285	0.9513	0.6312
Section 4.5.3				
TDE	**0.8589**	**0.8343**	**0.9545**	**0.5968**
TDE-Anova	0.8343	0.8079	0.9411	0.6868
TDE-AnovaGP	0.8512	0.8261	0.9519	0.6205
TDE-OnlyIGB	0.8559	0.8310	0.9540	0.6022
TDE-Chi50	0.8397	0.8201	0.9487	0.6735
TDE-Chi70	0.8405	0.8197	0.9489	0.6608
TDE-Chi90	0.8407	0.8184	0.9486	0.6515
TDE-ChiGP	0.8396	0.8197	0.9474	0.6758

Table B.4 The 16 UCR time series archive [36] datasets which BOSS takes over an hour to build.

Name	#Train	#Test	#Classes	Length	Type
Crop	7200	16800	24	46	Image
ElectricDevices	8926	7711	7	96	Device
EOGHorizontalSignal	362	362	12	1250	Biosignal
EOGVerticalSignal	362	362	12	1250	Biosignal
EthanolLevel	504	500	4	1751	Spectrum
FordA	3601	1320	2	500	Sensor
FordB	3636	810	2	500	Sensor
HandOutlines	1000	370	2	2709	Image
MixedShapes	500	2425	5	1024	Image
NonInvasiveFetalECGThorax1	1800	1965	42	750	Biosignal
NonInvasiveFetalECGThorax2	1800	1965	42	750	Biosignal
SemgHandGenderCh2	300	600	2	1500	Spectrum
SemgHandMovementCh2	450	450	6	1500	Spectrum
SemgHandSubjectCh2	450	450	5	1500	Spectrum
StarLightCurves	1000	8236	3	1024	Sensor
UWaveGestureLibraryAll	896	3582	8	945	Motion

Appendix C

Interval Based Results

This appendix contains additional information to help improve reproducibility of our Chapter 5 interval based experiments. Algorithm C.1 gives a pseudocode description of the simple Hybrid TSF/catch22 classifier used in the chapter. Table C.1 contains the subsample of datasets used in Section 5.4.2 and Table C.2 a list of unnormalised UCR datasets. Table C.3 contains the results from univariate experiments in Section 5.3, with multivariate experiments in Table C.4. Table C.5 contains scores for experiments in Section 5.4. Further results, figures and spreadsheet files are available on the webpages accompanying the CIF[1] and HC2[2] publications.

Algorithm C.1 Hybrid(A list of n cases length m, $\mathbf{T} = (\mathbf{X}, \mathbf{y})$)

Parameters: the number of trees, r, the number of intervals per tree k

Let $\mathbf{F} = (\mathbf{F_1} \dots \mathbf{F_r})$ be the number of trees in the forest
for $i \leftarrow 1$ to r **do**
  Let $\mathbf{S}$ be a list of n cases $(s_1 \dots s_n)$ with $22k$ attributes
  for $j \leftarrow 1$ to k **do**
    $b = rand(1, m-3)$ { *interval position*}
    $l = rand(b+3, m)$ { *interval position*}
    for $t \leftarrow 1$ to n **do**
      for $c \leftarrow 1$ to 22 **do**
        $s_{t,22(j-1)+c} = catch22Feature(c, \mathbf{X}_{t,b:(b+l-1)})$
  $\mathbf{F}_i.buildRandomTree([\mathbf{S}, y])$

[1] https://timeseriesclassification.com/CIF.php
[2] https://timeseriesclassification.com/HC2.php

Table C.1 List of 21 UCR time series archive [36] datasets used in the Catch22 importance experiment in section 5.4.2

Name	#Train	#Test	#Classes	Length	Type
ACSF1	100	100	10	1460	Device
Beef	30	30	5	470	Spectrum
BeetleFly	20	20	2	512	Image
Car	60	60	4	577	Sensor
CricketX	390	390	12	300	Motion
FaceFour	24	88	4	350	Image
FiftyWords	450	455	50	270	Image
Fish	175	175	7	463	Image
Herring	64	64	2	512	Image
ItalyPowerDemand	67	1029	2	24	Sensor
Lightning7	70	73	7	319	Sensor
MixedShapesSmallTrain	100	2425	5	1024	Image
OliveOil	30	30	4	570	Spectrum
PhalangesOutlinesCorrect	1800	858	2	80	Image
SmoothSubspace	150	150	3	15	Simulated
SwedishLeaf	500	625	15	128	Image
Symbols	25	995	6	398	Image
UWaveGestureLibraryZ	896	3582	8	315	Motion
Wafer	1000	6164	2	152	Sensor
Wine	57	54	2	234	Spectrum
Worms	181	77	5	900	Motion

Table C.2 List of 19 equal length UCR time series archive [36] datasets which are not normalised.

Name	#Train	#Test	#Classes	Length	Type
Chinatown	20	345	2	24	Sensor
Crop	7200	16800	24	46	Image
EOGHorizontalSignal	362	362	12	1250	Biosignal
EOGVerticalSignal	362	362	12	1250	Biosignal
GunPointAgeSpan	135	316	2	150	Motion
GunPointMaleVersusFemale	135	316	2	150	Motion
GunPointOldVersusYoung	135	316	2	150	Motion
HouseTwenty	34	101	2	2000	Device
InsectEPGRegularTrain	62	249	3	601	Biosignal
InsectEPGSmallTrain	17	249	3	601	Biosignal
PigAirwayPressure	104	208	52	2000	Biosignal
PigArtPressure	104	208	52	2000	Biosignal
PigCVP	104	208	52	2000	Biosignal
PowerCons	180	180	2	144	Device
Rock	20	50	4	2844	Spectrum
SemgHandGenderCh2	300	600	2	1500	Spectrum
SemgHandMovementCh2	450	450	6	1500	Spectrum
SemgHandSubjectCh2	450	450	5	1500	Spectrum
SmoothSubspace	150	150	3	15	Simulated

Table C.3 Average scores from univariate experiments presented in section 5.3. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

Name	Accuracy	BalAcc	AUROC	NLL
Section 5.3.1				
CIF	0.8487	0.8255	0.9553	0.7461
CIF-Fast	0.8468	0.8233	0.9540	0.7619
DrCIF	0.8637	0.8406	**0.9617**	0.6970
TSF	0.7983	0.7766	0.9268	0.8403
RISE	0.8043	0.7763	0.9381	1.0685
STSF	0.8464	0.8281	0.9556	0.9521
Catch22	0.7850	0.7603	0.9256	0.9356
TDE	0.8589	0.8341	0.9495	**0.5940**
STC	0.8584	0.8337	0.9582	0.6856
ROCKET	**0.8663**	**0.8462**	0.8713	0.888
Section 5.3.1 Figure 5.4 (109 UCR datasets)				
CIF	0.8495	0.8257	0.9548	0.7409
DrCIF	**0.8644**	**0.8406**	**0.9612**	**0.6927**
Hybrid	0.8472	0.8245	0.9535	0.7052
TSF	0.7999	0.7776	0.9265	0.8836
RISE	0.8066	0.7778	0.9382	1.0475
STSF	0.8475	0.8287	0.9552	0.7468
Catch22	0.7861	0.7608	0.9248	0.9291
Section 5.3.1 Figure 5.5 (109 UCR datasets)				
DrCIF	0.8618	0.8383	**0.9609**	0.7035
TSF	0.7954	0.7734	0.9253	0.8492
RISE	0.8016	0.7732	0.9370	1.0670
TDE	0.8585	0.8334	0.9486	**0.5947**
STC	0.8562	0.8311	0.9574	0.6947
PF	0.8348	0.8162	0.9402	0.6870
ROCKET	**0.8639**	**0.8435**	0.8695	0.9038

Table C.4 Average scores from multivariate experiments presented in section 5.3. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

Name	Accuracy	BalAcc	AUROC	NLL
Section 5.3.3				
CIF	**0.7277**	**0.7114**	0.8489	1.0878
DrCIF	0.7243	0.7091	0.8450	1.0705
TDE	0.7021	0.6818	0.8231	**1.0461**
gRSF	0.6726	0.6572	0.8096	1.2865
ROCKET	0.7196	0.6987	0.7892	1.5213
DTW_I	0.6035	0.5931	0.6531	2.633
DTW_D	0.6626	0.6488	0.6913	2.2404
TSF_I	0.6685	0.6492	0.8136	1.4172
$HC1_I$	0.7223	0.6997	**0.8495**	1.2617

Table C.5 Average scores from experiments presented in section 5.4. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

Name	Accuracy	BalAcc	AUROC	NLL
Section 5.4.1				
CIF	0.8487	0.8255	**0.9553**	0.7461
CIF-BasicStats	0.8485	0.8245	0.9552	0.7638
CIF-TST	**0.8513**	**0.8285**	**0.9553**	0.7385
CIF-OutlierNorm	0.8487	0.8256	0.9552	0.7448
CIF-AttSubsample	0.8456	0.8233	0.9547	**0.7001**
Section 5.4.3 (37 datasets)				
Hybrid	**0.8541**	**0.8483**	0.9666	**0.7411**
Hybrid-PreNorm	0.8287	0.8215	0.9583	0.8906
Hybrid-Norm	0.8317	0.8228	0.9610	0.9082
Hybrid-Outlier	0.8529	0.8471	**0.9670**	0.7435
Section 5.4.4				
CIF	0.8487	0.8255	0.9553	0.7461
CIF-IS2	0.8488	0.8253	0.9551	0.7413
CIF-IS3	**0.8515**	0.8284	**0.9562**	0.7347
CIF-IS4	0.8509	**0.8289**	0.9554	**0.7192**
CIF-IS5	0.8501	0.8268	0.9557	0.7438

Appendix D

Hybrid Results

We present the extended for our Chapter 6 experiments in this appendix. Table D.1 presents the results from univariate experiments in Section 6.3.1. Table D.2 presents scores for the multivariate experiments in Section 6.3.2. Table D.3 contains scores for the analysis experiments in Section 6.4. The HC2 publication companion website[1] provides individual results files.

[1] https://timeseriesclassification.com/HC2.php

Table D.1 Average scores from univariate experiments presented in section 6.3. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

Name	Accuracy	BalAcc	AUROC	NLL
Section 6.3.1				
HC2	**0.8912**	**0.8693**	**0.9687**	**0.5232**
DrCIF	0.8637	0.8406	0.9617	0.6970
TDE	0.8589	0.8341	0.9495	0.5940
STC	0.8584	0.8337	0.9582	0.6856
Arsenal	0.8659	0.8452	0.9066	0.7430
HC1	0.8801	0.8551	0.9653	0.6764
TS-CHIEF	0.8776	0.8572	0.9595	0.6100
InceptionTime	0.8743	0.8589	0.9587	0.7431
ROCKET	0.8663	0.8462	0.8713	0.888

Table D.2 Average scores from multivariate experiments presented in section 6.3. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

Name	Accuracy	BalAcc	AUROC	NLL
Section 6.3.2				
HC2	**0.7448**	**0.7234**	**0.8584**	**0.9860**
CIF	0.7277	0.7114	0.8489	1.0878
ROCKET	0.7196	0.6987	0.7892	1.5213
DTW_D	0.6626	0.6488	0.6913	2.2404
HC1	0.7223	0.6997	0.8495	1.2617

Table D.3 Average scores from experiments presented in section 6.4. Reports accuracy, balanced accuracy (BalAcc), Area Under the Receiver Operating Characteristic curve (AUROC) and Negative Log-Likelihood (NLL).

Name	Accuracy	BalAcc	AUROC	NLL
Section 6.4.1				
Full HC2	**0.8912**	**0.8693**	**0.9687**	0.5232
HC-1	0.8736	0.8526	0.9639	0.5773
HC-3	0.8848	0.8609	0.9661	0.5704
HC-4	0.8814	0.8604	0.9633	0.5435
HC-5	0.8819	0.8608	0.9558	0.5127
HC-6	0.8808	0.8571	0.9648	0.5757
HC-7	0.8870	0.8657	0.9673	0.5438
HC-8	0.8872	0.8658	0.9668	**0.5092**
HC-9	0.8889	0.8652	0.9682	0.5759
HC-10	0.8881	0.8662	0.9664	0.5150
Section 6.4.2				
HC2-oob	0.8912	0.8693	0.9687	0.5232
HC2-cv	**0.8920**	**0.8700**	**0.9688**	0.5245
HC2-fastest	0.8917	0.8697	0.9686	0.5238
HC2-closest	0.8912	0.8693	**0.9688**	**0.5221**
Section 6.4.3				
HC2	0.8912	0.8693	**0.9687**	**0.5232**
HC2-RandF	0.8689	0.8473	0.9599	0.6498
HC2-ES	0.8853	0.8627	0.9293	0.5583
HC2-FS	0.8801	0.8584	0.9123	0.6197
HC2-PB	0.8659	0.8451	0.9066	0.7431
HC2-Oracle	**0.8968**	**0.8757**	0.9453	0.5432
Section 6.4.4				
HC2-Arsenal	**0.8912**	**0.8693**	**0.9687**	**0.5232**
HC2-ROCKET	0.8885	0.8673	0.9682	0.5240
HC2-Ar1H	0.8882	0.8664	0.9682	0.5257
Arsenal	0.8659	0.8452	0.9066	0.7430
ROCKET	0.8663	0.8462	0.8713	0.8880