Mining Time-series Data using Discriminative Subsequences

Jonathan Frederick Francis Hills

A Thesis Submitted for the

Degree of Doctor of Philosophy

University of East Anglia

School of Computing Sciences

September 2014

Abstract

Time-series data is abundant, and must be analysed to extract usable knowledge. Local-shape-based methods offer improved performance for many problems, and a comprehensible method of understanding both data and models.

For time-series classification, we transform the data into a local-shape space using a *shapelet transform*. A *shapelet* is a time-series subsequence that is discriminative of the class of the original series. We use a heterogeneous ensemble classifier on the transformed data. The accuracy of our method is significantly better than the time-series classification benchmark (1-nearest-neighbour with dynamic time-warping distance), and significantly better than the previous best shapelet-based classifiers. We use two methods to increase interpretability: first, we cluster the shapelets using a novel, parameterless clustering method based on Minimum Description Length, reducing dimensionality and removing duplicate shapelets. Second, we transform the shapelet data into binary data reflecting the presence or absence of particular shapelets, a representation that is straightforward to interpret and understand.

We supplement the ensemble classifier with partial classification. We generate rule sets on the binary-shapelet data, improving performance on certain classes, and revealing the relationship between the shapelets and the class label. To aid interpretability, we use a novel algorithm, *BruteSuppression*, that can substantially reduce the size of a rule set without negatively affecting performance, leading to a more compact, comprehensible model.

Finally, we propose three novel algorithms for unsupervised mining of approximately repeated patterns in time-series data, testing their performance in terms of speed and accuracy on synthetic data, and on a real-world electricity-consumption device-disambiguation problem. We show that individual devices can be found automatically and in an unsupervised manner using a local-shape-based approach.

Table of Contents

List of Algorithms

List of Publications

Publications as first author:

- J. Hills, L. Davis, A. Bagnall
  Interestingness Measures for Fixed Consequent Rules
  Proceedings of the 13th International Conference on Intelligent Data Engineering and Automated Learning, Lecture Notes in Computer Science, pages 68–75, Springer-Verlag: 2012.

- J. Hills, A. Bagnall, B. De La Iglesia, and G. Richards
  BruteSuppression: a Size Reduction Method for Apriori Rule Sets
  Journal of Intelligent Information Systems 40 (3), pages 431–454, Springer: 2013.

- J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall
  Classification of Time Series by Shapelet Transformation
  Data Mining and Knowledge Discovery 28 (4), pages 851–881, Springer: 2014.

Publications as co-author:

- A. Bagnall, L. Davis, J. Hills, and J. Lines
  Transformation Based Ensembles for Time Series Classification
  Proceedings of the SIAM International Conference on Data Mining, 12, pages 307–318, SIAM: 2012.

- J. Lines, L. Davis, J. Hills, and A. Bagnall
  A Shapelet Transform for Time Series Classification
  Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 289–297, ACM: 2012.

- A. Bagnall, J. Lines, J. Hills, and A. Bostrom
  Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles

https://ueaeprints.uea.ac.uk/id/eprint/49614

This paper is currently under review.

Chapter 1

Introduction

There is an abundance of time-series data that must be analysed in order to derive usable knowledge. Our hypothesis is that much of this analysis would benefit from an approach based on local similarity of shape; that is, a methodology focused on subsequences of the time series that are particularly informative.

Local similarity of shape can inform prediction and other tasks, but can also enhance understanding of time-series data. Consider Figure 1.1. Examined as a whole, there are long stretches of the series where they barely differ. In the middle third, however, are local shapes that differ substantially from series to series. By examining local-shape-based similarity, we can ensure that these differences are not swamped by the general similarities, but are used to distinguish the series, exactly as they are under intuitive visual inspection.

Local shapes are uniquely comprehensible aspects of time series. A very long time series cannot be grasped intuitively, which makes systems based on global similarity of shape uninformative. Data-mining techniques based on auto-correlation or Fourier transformation can reduce comprehensibility, and make visual interpretation very difficult, especially for non-specialists. Our hypothesis is that local similarity of shape offers an intuitively comprehensible way of understanding long time series. In addition to this, we believe that local similarity of shape can be used to improve the quantitative aspects of time-series data mining.

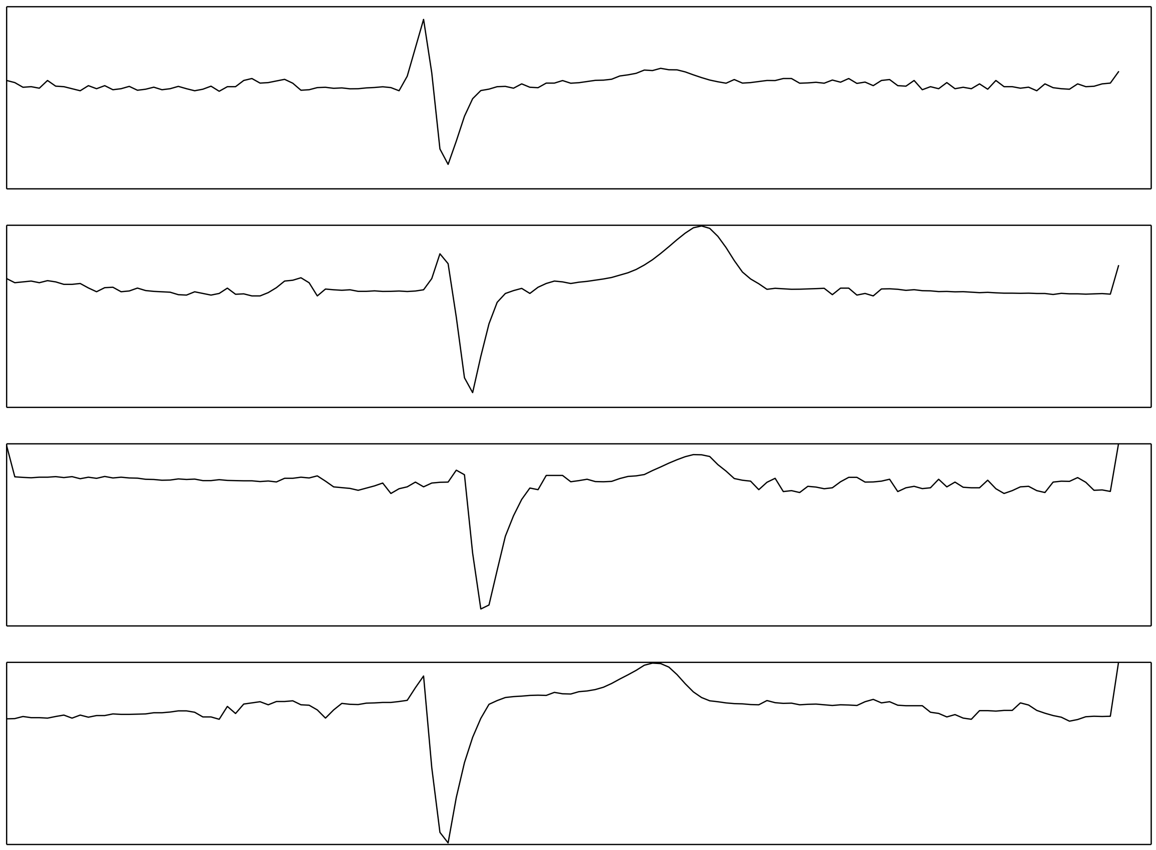

Figure 1.1: Four ECG time series from the ECGFiveDays dataset. The beginning third and final third of the series are very similar, differing only a little and seemingly at random. The middle third, in contrast, has a number of local shapes that differ substantially between the series.

We apply the subsequence-based approach to supervised data mining, showing that time-series classification accuracy can be improved by transforming the data into a space of local shapes, and that rule induction for partial classification of time series can be fruitfully applied to the transformed data. We also investigate unsupervised data mining, finding approximately repeated subsequences in time-series data that can provide primitives for future data mining, and can aid in understanding and analysing the data.

We make a number of novel research contributions to time-series classification, clustering, rule induction, and repeated pattern mining over the course of this thesis; these contributions are detailed in Section 1.2.

1.1 Research objectives

The overall objective of this thesis is to address the following question: how best can we use methods based on local similarity of shape for mining time-series data? More specifically, the question can be broken down into two sub-questions:

1. How can notions of local similarity of shape be used to create high-performing approaches to existing time-series data-mining tasks?
2. How can the interpretability offered by methods based on local similarity of shape be maximised without compromising performance?

1.1.1 Objectives

The objectives of the thesis, which address the research questions, are as follows:

1. Develop and test subsequence-based representations for time-series.
2. Create and refine algorithms and methods for discovering and extracting subsequences to represent locally-similar features.
3. Implement and test approaches to make best use of the representations for solving specific data-mining problems.
4. Design methods and representations that maximise interpretability without substantially diminishing performance.

1.1.2 Challenges

Achieving the objectives listed in the previous subsection, and hence satisfactorily answering the research questions, requires surmounting three main challenges:

1. Existing methods for time-series classification, with which a large proportion of this thesis is concerned, are well developed and highly accurate. The new methods proposed must offer substantial improvements above the existing methods, either in terms of accuracy or interpretability. In terms of classification accuracy, benchmark approaches must be bettered, or at least equalled (if there is some other advantage to using local similarity, e.g. increased interpretability). This is challenging, because the benchmark level of accuracy is high.

2. One strength of the local-shape-based approach is interpretability. Interpretability is inversely proportional to complexity; to maximise interpretability, complexity must be minimised. Over simplification, however, is detrimental to performance. A key challenge in answering the overriding research question of this thesis is balancing interpretability with performance.

3. Discovering and extracting representative subsequences is a complex problem; the time complexity of proposed algorithms must be reasonable enough that they can be tested on a range of problems. The same is true of dimensionality-reduction methods to improve interpretability: the time complexity of the method must not be so high as to exclude reasonable applications. This is partly so that the new methods can be deployed in a wide range of situations, and partly to ensure that they can be tested on a large number of datasets to ensure robust results.

In overcoming these challenges and answering the overriding research question, we have made a number of novel contributions, listed in the next section.

1.2 Novel contributions

1.2.1 Shapelet transform and ensemble classifier

In Chapter 5, we propose and extensively test a shapelet transform, using heterogeneous ensemble classification, that provides significantly better classification accuracy than the benchmark approach to time-series classification (1NN with DTW distance and cross-validated warping window size). It is also significantly more accurate than the leading shapelet-based approaches (Fast Shapelets and Logical Shapelets).

Work on the shapelet transform is published in [88]; legacy results are published in [119]. Many of the results of our extensive experimentation with the up-to-date version of the transform and the heterogeneous ensemble classifier are used in [9] (currently under review), which combines a large number of different transforms, distance measures, and classifiers to create a more accurate time-series classification system than anything in the current literature. The shapelet transform with ensemble classifier is one of the best-performing elements of the COTE ensemble [9].

1.2.2 Clustering shapelets and binary transformation

In Chapter 5, we propose and test a novel, parameterless approach to clustering shapelets. Our method finds the correct number of shapelets for a given problem using Minimum Description Length, maintaining classification accuracy while reducing the number of shapelets to increase interpretability. This is a useful feature for reducing the dimensionality of our shapelet-transformed datasets, but more importantly, shows that clustering with MDL could be suitable in the more general case for correctly clustering time-series subsequences.

The preliminary work on clustering shapelets is published in [88].

Following on from our clustering method in Chapter 5, we propose a binary transform for shapelet data that we believe offers the most interpretable form of shapelet-transformation. The binary-transformed data sacrifices a small amount of accuracy for some (but not all) datasets, but delivers high interpretability, with each series of the transformed data represented by binary values indicating the presence or absence of each shapelet in that series. Models built on this data have the potential to be comprehensible to end-users working in domains like medicine and finance, where every decision must be justifiable.

1.2.3 Partial classification of binary-shapelet data

In Chapter 6, we propose a partial classification method based on association rules that significantly improves upon the accuracy of our state-of-the-art classifier in cases where the ensemble predicts poorly. It also offers highly interpretable insight into the relationship between the data and the class in the form of association rules that hold over binary shapelets.

Our approach is best used in one of two ways: as a supplement to the ensemble classifier to improve performance for poorly-predicted classes of interest, or as a general-use, highly interpretable approach to analysing the relationships between the attributes of the data and the class label.

1.2.4 BruteSuppression

In Chapter 6, we propose *BruteSuppression*, a novel algorithm for reducing rule set size that can dramatically decrease the number of rules in the rule set with no loss of classification accuracy. This creates more compact and interpretable rule sets. The BruteSuppression algorithm is published in [86], along with a large amount of analysis into its effects on the distribution of rules in rule sets. Here, we restrict our qualitative analysis to a single standard dataset (Adult [164]), and focus on the quantitative effects of BruteSuppression on the size and predictive performance of rule sets built on binary-shapelet data.

1.2.5 Interestingness measures for partial classification rules

In Chapter 2, we show that twelve commonly used interestingness measures impose the same ordering on a partial classification rule set as confidence, making them redundant as assessment tools for these rule sets. Many algorithms, including Apriori, allow the user to select a measure other than confidence to use for rule induction, and we require an interestingness measure for the BruteSuppression algorithm proposed in Chapter 6. By proving that there is no tangible difference between these measures in terms of partial classification, we justify our use of confidence as an interestingness measure, and also clarify the choice of interestingness measure for partial classification in the general case.

This work is published in [87].

1.2.6 Mining approximately repeated patterns

In Chapter 7, we propose three novel algorithms for finding approximately repeating patterns in time series, testing them extensively on bespoke synthetic data, and on a real-world electrical device disambiguation problem. We show that a subsequence-based approach can be used to detect specific instances of distinct device usage, suggesting that motifs could be suitable for similar problems in other domains where approximately repeated patterns must be discovered in time series.

1.3 Thesis structure

The thesis is structured as follows:

- In Chapter 2, we discuss time-series classification and rule induction.

- In Chapter 3, we give an up-to-date overview of the shapelet approach and motifs.
- Chapter 4 describes the datasets we have used for our experiments; we cover existing time-series problems and our own contributed data.
- In Chapter 5, we propose and test a shapelet transform with an ensemble classifier that outperforms the benchmark time-series classification method and the best published shapelet-based classifiers. We also create a novel parameterless clustering method for finding the 'right' number of shapelets for a given problem, and a binary transform for shapelets that maximises interpretability.
- Chapter 6 describes a system for partial classification using binary-shapelet data that can significantly improve upon the performance of the ensemble classifier in certain cases, and which offers a highly interpretable model. We also propose a novel algorithm that substantially reduces the size of association rule sets without compromising partial classification or removing potentially interesting rules.
- In Chapter 7, we propose three novel algorithms for finding approximately repeating patterns in time series, testing them extensively on bespoke synthetic data, and on a real-world electrical device disambiguation problem.
- Chapter 8 presents our conclusions and suggests directions in which our work could be extended.

Chapter 2

Time-series Classification and Rule Induction

Section 2.10 of this chapter is based on a novel analysis of existing interestingness measures published in the following paper:

J. Hills, L. Davis, A. Bagnall

Interestingness Measures for Fixed Consequent Rules

Proceedings of the 13th International Conference on Intelligent Data Engineering and Automated Learning, Lecture Notes in Computer Science, pages 68–75, Springer-Verlag: 2012.

2.1 Introduction

We focus our interest on three time-series data-mining tasks: time-series classification, partial classification of time series with association rules, and the discovery of repeated patterns in time series. We address the background of the final task in Chapter 3, as our work is very dependent on the particular representation we use for the repeated patterns, and the information is better presented in context. In the first half of this chapter, we address time-series classification; the subject of the second half is induction of partial classification rules.

In Section 2.2, we discuss time-series classification; we make novel contributions to this area in Chapters 5 and 6. We discuss the classifiers we have used in Section 2.3, and dynamic time warping, a benchmark approach in time-series classification, with which we compare our method, in Section 2.4. We use an ensemble classifier; the relevant background is given in Section 2.5. In Section 2.6, we discuss the two main tests we use to compare classifiers: the Wilcoxon Signed Rank Test (Section 2.6.1) and the Friedman Test with post-hoc Nemenyi test (Section 2.6.2). We also describe the visual method we use to display our results: the critical-difference diagram (Section 2.6.3).

In Section 2.7, we shift focus from classification to partial classification with association rules, a field to which we make a novel contribution in Chapter 6. We describe the association rule algorithm we use, Apriori, in Section 2.8. In Section 2.9, we discuss interestingness measures. Which interestingness measures to use in Chapter 6 is an important design choice, motivated by the theoretical assessment given in Section 2.10.

We close our discussion with conclusions in Section 2.12.

2.2 Time-Series classification (TSC)

2.2.1 Classification

In *classification*, a *learner* uses labelled training examples to search its hypothesis space for a *classifier*. The classifier generalises from the training examples to predict labels for new examples. Different learners use different strategies to search their hypothesis space; the hypothesis space itself varies from learner to learner.

A size N dataset $\mathbf{D}$ is a set of instances $\{D_1, D_2, \ldots, D_N\}$, where each instance $D_i = \{< x_{i,1}, x_{i,2}, \ldots, x_{i,m} >, c_i\}$ consists of a set of m attribute values and a class label. The order of the attributes is unimportant, and interaction between variables is considered to be independent of their relative positions. $\mathbf{D}$ is split into a training set $\mathbf{D}_{train}$ and a test set $\mathbf{D}_{test}$, such that $\mathbf{D}_{train} \cup \mathbf{D}_{test} = \mathbf{D}$ and $\mathbf{D}_{train} \cap \mathbf{D}_{test} = \emptyset$.

A classifier is *trained* by inputting $\mathbf{D}_{train}$ to the learner. The algorithm uses the labelled examples to infer the relationship between the attributes and the class label. Given an instance, the trained classifier produces a prediction based on the attributes of the instance. The accuracy of the classifier's predictions on unseen data (often the instances in $\mathbf{D}_{test}$) is calculated to provide an estimate of how well the classifier represents the relationship between the attributes and the class label.

We focus on a sub-problem within classification: *time-series classification* (TSC).

2.2.2 Time-series

A time series is a sequence of data that is typically recorded in temporal order at fixed intervals. Suppose we have a set of N time series $\mathbf{T} = \{T_1, T_2, \ldots, T_N\}$, where each time series T_i has m real-valued ordered readings $T_i =< t_{i,1}, t_{i,2}, ..., t_{i,m} >$, and a class label c_i. We assume that all series in $\mathbf{T}$ are of length m, but this is not a requirement (see [91] for discussion of this issue). Given a dataset $\mathbf{T}$, the time-series classification problem is to find a function that maps from the space of possible time

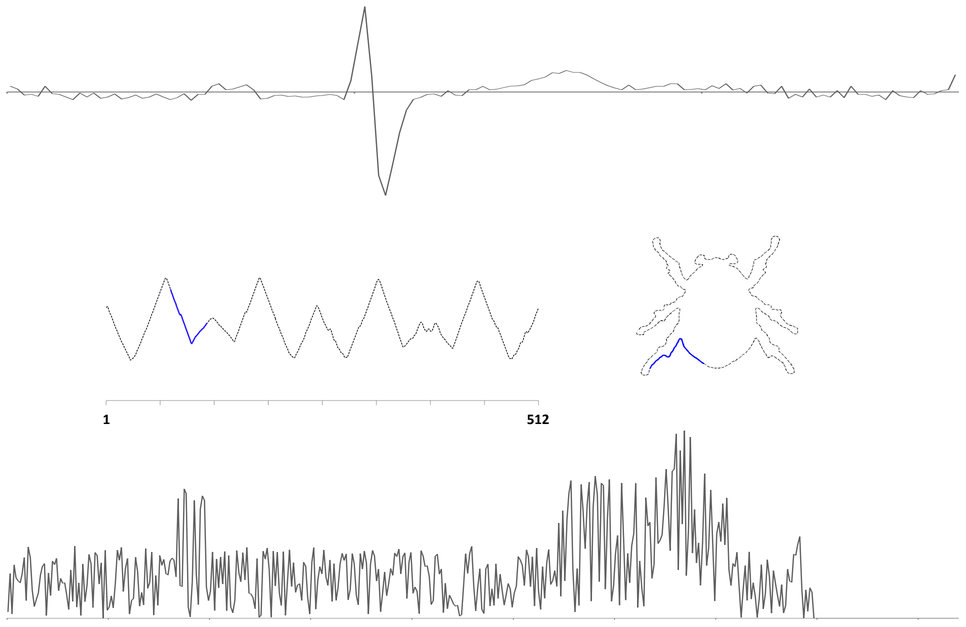

Figure 2.1: *Top*: a time series representing electrical activity in the heart. *Middle*: a 1-D series (left) representing the outline of an image (right). *Bottom*: an ordered series representing a spectrograph of a sample of beef.

series to the space of possible class labels.

In TSC, a class label is applied to an unlabelled set of data. The attributes of time-series objects represent ordered data, typically the same quality measured over time. Data need not be ordered temporally to be treated as a time series, however. Examples of sequence data include temporally-ordered data, such as recordings of electrical activity in the heart (Figure 2.1 top), spatially-ordered data, e.g. images (Figure 2.1 middle), and other ordered data, for example spectrographs (Figure 2.1 bottom). For time-series data, the order of the variables is often crucial for finding the best discriminating features.

TSC problems occur in a variety of domains, including image processing [143], robotics [166], healthcare [146], and gesture recognition [122].

Various algorithms are used for TSC, including tree-based classifiers (e.g. *C4.5*), lazy classifiers (e.g. *k-nearest neighbour* (kNN)), and probabilistic classifiers (e.g.

Naive Bayes). TSC research has focused on alternative distance measures for kNN classifiers, based on either the raw data, or on compressed or smoothed data (see [51] for a comprehensive summary). This idea has propagated through current research. For example, Batista *et al.* state that *"there is a plethora of classification algorithms that can be applied to time series; however, all of the current empirical evidence suggests that simple nearest neighbor classification is very difficult to beat"* [12]. Recently, several alternative approaches have been proposed, such as weighted dynamic time warping [99], support vector machines built on variable intervals [150], tree-based ensembles constructed on summary statistics [46], and a fusion of alternative distance measures [29].

There are three general types of within-class similarity that may be relevant to TSC: similarity in time, similarity in change, and similarity in shape.

Similarity in time encompasses time series that are observations of variation of a common curve in the time dimension. kNN classifiers with elastic distance measures perform well when classes are distinguished by this kind of similarity [51, 7]. Similarity in change occurs when series of the same class have a similar form of autocorrelation. A common approach used for TSC with this kind of similarity is to fit an Autoregressive Moving Average (ARMA) model and base classification on differences in the model parameters [8]. Our interest is focused on similarity in shape.

2.2.3 Similarity in shape

Similarity in shape is a feature of time series that are distinguished by some phase-independent sub-shape that may appear at any point in the series. A *global* sub-shape is one that approaches the length of the series. A *local* sub-shape is one that is short relative to the length of the series, and may appear anywhere in the series.

If a global common sub-shape is phase-shifted to a large degree within instances of

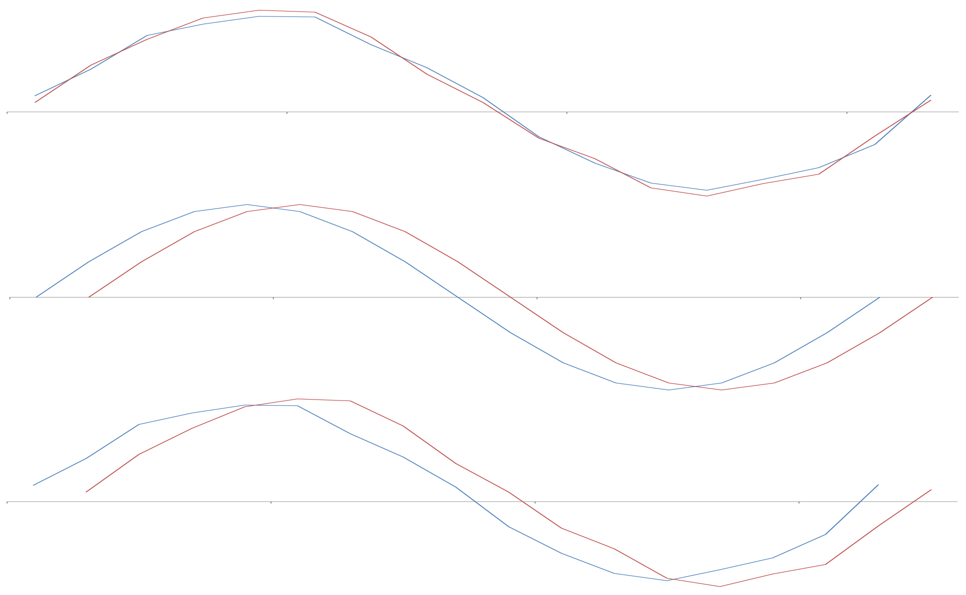

Figure 2.2: Globally similar sine waves. *Top*: noise in measurement. *Middle*: noise in indexing. *Bottom*: noise in measurement and indexing.

the same class, transformation into the frequency domain is likely to be fruitful [173, 98, 30, 7].

If a global common sub-shape is found within instances of the same class and there is little phase shift, kNN classifiers with elastic distance measures will perform well [51, 7]. Variation around the underlying shape is caused by noise in observation, and also by noise in indexing, which may cause a slight phase shift. Consider two series produced by the same sine function. Noise in measurement alters values in the series; noise in indexing offsets the sine wave (Fig. 2.2).

A standard example of this global similarity of shape with little phase shift is the Cylinder-Bell-Funnel artificial dataset, where there is noise around the underlying shape and in the index where the shape transitions (see Figure 2.3).

Locally-similar series have common sub-shapes that are short relative to the series. Local similarity can be obscured by noise in measurement; in addition, if the similar subshapes appear at very different indexes, they will be difficult to detect

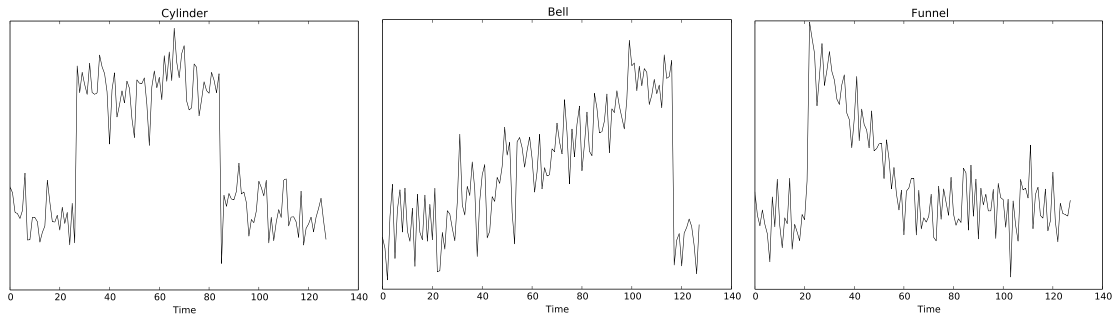

Figure 2.3: Series from the Cylinder-Bell-Funnel dataset. Each series represents a noisy, offset instance of a particular curve, specific to one of the three types. From left to right: Cylinder, Bell, Funnel.

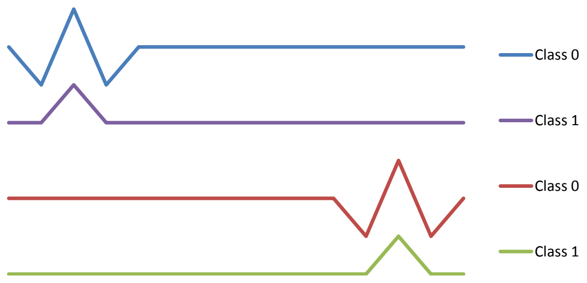

Figure 2.4: Time series of two different classes. A global approach will pair the series incorrectly, due to the offset in the time dimension. A local approach will pair the instances correctly, as the subsequences are exact matches.

with global approaches (Fig. 2.4). Transformation into the frequency domain and kNN with elastic distance measures might not discriminate well between the classes. The *shapelet* approach is tailored to discriminate on local similarity (see Chapters 3 and 5).

2.3 Classifiers

We make use of a number of standard classifiers: *C4.5*, kNN, *Naive Bayes*, *Bayesian Network*, *Random Forest*, *Rotation Forest*, and two *Support Vector Machines*. We choose these classifiers as they are widely used, are diverse in their operation, and are all implemented in the *WEKA* machine learning tool kit [78].

For brevity, we exclude descriptions of the simpler classifiers - C4.5, kNN, and

Naive Bayes - referring the reader to [79]. The more complex classifiers operate as follows. A Bayesian network is an acyclic directed graph with associated probability distributions [63]. It predicts class labels without assuming independence between variables. The Random Forest algorithm classifies examples by generating a large number of decision trees with controlled variation, and taking the modal classification decision [24]. The Rotation Forest algorithm trains a number of decision trees by applying principal components analysis on a random subset of attributes [151]. A support vector machine finds the best separating hyperplane for a set of data by selecting the margin that maximises the distance between the nearest examples of each class [38]. It can also transform the data into a higher dimension to make it linearly separable.

2.4 Dynamic time warping

Dynamic Time Warping (*DTW*) distance is used with kNN classifiers in TSC to mitigate problems caused by distortion in the time axis [18, 144]. Measuring the DTW distance between two length m series, $t =< t_1, t_2, \ldots, t_m >$, and $s =< s_1, s_2, \ldots, s_m >$, involves computing the $m \times m$ distance matrix $\mathbf{M}$, where $\mathbf{M}_{i,j} = (t_i - s_j)^2$.

A warping path, P, is a set of points that defines a traversal of $\mathbf{M}$, where e and f are row and column indices respectively:

$$P =< (e_1, f_1), \ldots, (e_m, f_m) > .$$

The Euclidean distance, for example, is the path that follows the diagonal of $\mathbf{M}$, $P =< (e_1, f_1), (e_2, f_2), \ldots, (e_{m-1}, f_{m-1}), (e_m, f_m) >$. All warping paths must begin at point $(1, 1)$ and end at point (m, m), and satisfy the conditions $(e_{i+1} - e_i) \in \{0, 1\}$ and $(f_{i+1} - f_i) \in \{0, 1\}$.

Other constraints can be placed on the warping paths. One common additional constraint restricts the *warping window* by defining a value, r, which determines the maximum allowable distance between any pair of indexes in the warping path. This reduces computation time, and may prevent some pathological warping paths, for example paths that map many points in one series to the same point in the other. Setting the warping window through cross validation significantly improves the accuracy of 1NN with DTW distance [144, 117].

The DTW distance between two series is defined as the total distance covered by the shortest warping path. The optimal path is found using dynamic programming [144]. Extensions to DTW have been proposed in [101, 99, 74].

Extensive experimentation has led to 1NN with DTW distance (and cross validation to select the size of the warping window) being regarded as the benchmark for TSC [51, 117]. The evidence [51, 152, 174] suggests that, for smaller datasets, elastic similarity measures such as DTW outperform simple Euclidean distance. However, as the number of series increases, *"the accuracy of elastic measures converges with that of Euclidean distance"* [51].

2.5 Ensemble classifiers

An ensemble of classifiers combines a set of base classifiers by fusing the individual predictions to classify new examples [50, 127]. Majority vote fusion [106] is an intuitive approach; for alternative fusion schemes see [104]. Beyond simple accuracy comparison, there are three common approaches to analyse ensemble performance: diversity measures [105, 162], margin theory [127, 145], and Bias-Variance decomposition [13, 23, 60, 97, 163, 167]. These have all been linked [162, 52].

The key concept in ensemble design is the requirement that the ensemble be diverse [50, 70, 76, 80, 123, 154]. Diversity can be achieved in the following ways:

- Employing different classification algorithms to train each base classifier to form a heterogeneous ensemble.

- Changing the training data for each base classifier through a sampling scheme or by directed weighting of instances [22, 24, 59, 170].
- Selecting different attributes to train each classifier [24, 89, 151].
- Modifying each classifier internally, either through re-weighting the training data or through inherent randomization [57, 59].

Ensembles have been applied to time-series data-mining problems, and have shown promising results [29, 46, 150]. For example, Deng *et al.* propose a version of random forest [24] that uses the mean, slope, and variance of subseries as the attribute space, offering better accuracy than random forest used on the raw data [46]. Buza [29], and Lines and Bagnall [117], use ensembles of different distance measures to improve TSC accuracy.

2.6 Comparing classifiers

Demšar [45] discusses tests for comparing classifiers over multiple datasets. He argues that the appropriate test for comparing two classifiers over multiple datasets is the Wilcoxon Signed Rank test; for comparing multiple classifiers over multiple datasets, he advocates the Friedman test with post-hoc Nemenyi test.

2.6.1 Comparing two classifiers: the Wilcoxon Signed Rank test

Demšar [45] recommends using the Wilcoxon Signed Rank test to compare two classifiers over multiple datasets for three reasons. First, the Wilcoxon Signed Rank test does not require that classfication accuracies across different problem domains be commensurable. Accuracies can be wildly different across problems; using ranks prevents ten differences of 0.01 being balanced by a single difference of 0.1. Second,

Table 2.1: Left: accuracies for two classifiers over 28 datasets with differences. Right: differences sorted by absolute value, ranks, and W_+ and W_- statistics.

Dataset	Classifier A	Classifier B	Difference
1	0.537	0.537	0.000
2	0.468	0.531	-0.063
3	0.530	0.530	0.000
4	0.700	0.700	0.000
5	0.274	0.077	0.197
6	0.662	0.622	0.041
7	0.496	0.000	0.496
8	0.358	0.394	-0.037
9	0.357	0.417	-0.060
10	0.251	0.050	0.201
11	0.282	0.154	0.127
12	0.365	0.255	0.110
13	0.446	0.151	0.295
14	0.441	0.182	0.259
15	0.347	0.264	0.083
16	0.183	0.057	0.126
17	0.346	0.342	0.004
18	0.417	0.371	0.046
19	0.179	0.069	0.110
20	0.204	0.146	0.058
21	0.342	0.146	0.196
22	0.308	0.204	0.104
23	0.453	0.454	-0.001
24	0.382	0.271	0.112
25	0.032	0.273	-0.241
26	0.208	0.227	-0.019
27	0.255	0.241	0.014
28	0.430	0.326	0.104

Sorted Difference	Absolute Difference	Ranks	W_+	W_-
0.000	0.000	0	0	0
0.000	0.000	0	0	0
0.000	0.000	0	0	0
-0.001	0.001	1	0	1
0.004	0.004	2	2	0
0.014	0.014	3	3	0
-0.019	0.019	4	0	4
-0.037	0.037	5	0	5
0.041	0.041	6	6	0
0.046	0.046	7	7	0
0.058	0.058	8	8	0
-0.060	0.060	9	0	9
-0.063	0.063	10	0	10
0.083	0.083	11	11	0
0.104	0.104	12	12	0
0.104	0.104	13	13	0
0.110	0.110	14	14	0
0.110	0.110	15	15	0
0.112	0.112	16	16	0
0.126	0.126	17	17	0
0.127	0.127	18	18	0
0.196	0.196	19	19	0
0.197	0.197	20	20	0
0.201	0.201	21	21	0
-0.241	0.241	22	0	22
0.259	0.259	23	23	0
0.295	0.295	24	24	0
0.496	0.496	25	25	0
Sum		325	274	51

the Wilcoxon Signed Rank test does not assume normality for the distribution of the accuracies; this is appropriate, as we cannot rely on this assumption. Third, because it uses ranks, the Wilcoxon Signed Rank test is not badly affected by outliers. For these reasons, we perform all pairwise comparisons of classifiers over multiple datasets using the Wilcoxon Signed Rank test.

The Wilcoxon Signed Rank test [172] is a non-parametric test. We use it to compare the accuracies of two classifiers over a number of different datasets (for example, Table 2.1). The difference is computed for each pair of accuracies. The list of differences is sorted by absolute value. The non-zero absolute differences are ranked in ascending order of absolute value (Table 2.1). The test statistics are created by summing the ranks for positive differences (W_+) and negative differences (W_-).

We take N_r to be the highest rank (in Table 2.1, 25). If $N_r \geq 15$, the distribution of W_+ or W_- (we concentrate on W_+, but W_- can be substituted *mutatis mutandis*) approaches the normal distribution where:

$$\mu_{W_+} = \frac{N_r(N_r+1)}{4}, \tag{2.6.1}$$

$$\sigma_{W_+} = \sqrt{\frac{N_r(N_r+1)(2N_r+1)}{24}}. \tag{2.6.2}$$

Hence, the statistic:

$$Z = \frac{W_+ - \mu_{W_+}}{\sigma_{W_+}} \tag{2.6.3}$$

can be used to test for significant difference where $N_r \geq 15$ (for cases where $N_r < 15$, a statistical table can be used to find critical values, see for example [168]).

In our example (Table 2.1), $N_r = 25$, $W_+ = 274$, $\mu_{W_+} = 162.5$, and $\sigma_{W_+} = 37.165$. Hence, $Z = \frac{274-162.5}{37.165} = 3$; there is a statistically significant difference between Classifier A and Classifier B at a significance level of 0.01.

Where we compare two classifiers over multiple datasets, we use the Wilcoxon Signed Rank test with a significance level of 0.01.

2.6.2 Comparing multiple classifiers: the Friedman Test with post-hoc Nemenyi test

Demšar [45] argues that a Friedman test [61, 62] with post-hoc Nemenyi test [133] is the best way to compare multiple classifiers over multiple datasets. The Friedman test does not rely on the same assumptions as the repeated measures ANOVA, which include the assumption that the samples (i.e. the accuracies) are drawn from a normal distribution, and the assumption of sphericity, which requires that the variances of the differences between all possible pairs of groups are equal. When comparing classifiers, there is no guarantee that these assumptions will not be violated. Hence, we prefer the Friedman test to the Anova for multiple comparisons.

The Friedman test ranks each classifier separately on its performance on each dataset, assigning ranks in ascending order (rank 1 for the best classifier, average ranks where classifiers tie). r_i^j is the rank of the j^{th} of k classifiers on the i^{th} of N datasets, and $R_j = \frac{1}{N}\sum_i r_i^j$, the average ranks of the algorithms. The Friedman statistic:

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)}{4}\right] \tag{2.6.4}$$

is used to compute:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2}, \tag{2.6.5}$$

which follows an F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom. We use the F_F statistic to determine whether there is a significant difference between the classifiers, testing the null hypothesis (that there is no difference) at a significance level of 0.01.

If the Friedman test allows us to reject the null hypothesis (i.e. there is a significant difference between the classifiers), we perform a post-hoc Nemenyi test. In a Nemenyi test, the performance of two classifiers is significantly different if the average ranks of the two classifiers differ by at least the *critical difference*:

$$CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}} \tag{2.6.6}$$

The critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$ (see, for example, [168], for statistical tables). If the difference in average rank between two classifiers (where we have rejected the null hypothesis using the Friedman test) exceeds the critical difference, we take the classifiers to be significantly different.

Where we compare multiple classifiers over multiple datasets, we use the Friedman test with post-hoc Nemenyi test at a significance level of 0.01.

2.6.3 Critical-difference diagram

When comparing multiple classifiers over multiple datasets, we present our results perspicuously using a *critical-difference diagram* [45]. An example critical-difference diagram is shown in Figure 2.5.

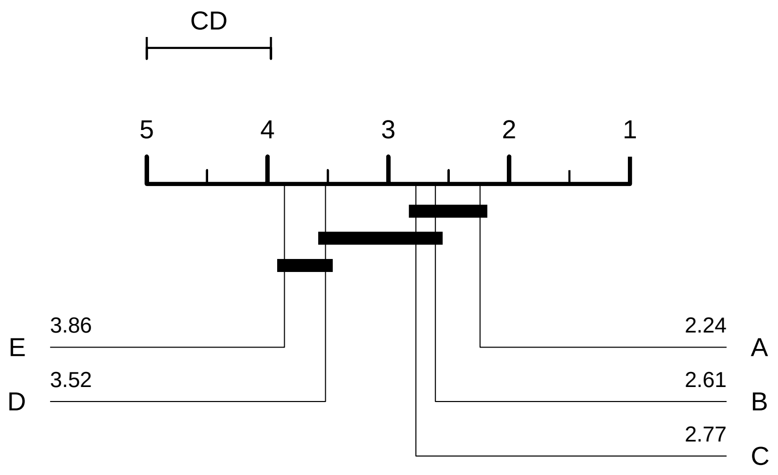

Figure 2.5: Example critical-difference diagram comparing five classifiers over multiple datasets. A pair of classifiers is significantly different if they do not belong to the same clique, represented by the black bars.

In Figure 2.5, each classifier is aligned along the axis based on its average rank over the datasets. The black bars show the *cliques*. If two classifiers belong to the same clique, there is no significant difference between their performances on the datasets. In Figure 2.5, classifiers A, B, and C all belong to the same clique, meaning there is no significant difference between their performances on the datasets. There is a significant difference between classifier E and classifiers A, B, and C, as they do not belong to the same clique.

Where we compare multiple classifiers over multiple datasets, we will present the results using critical-difference diagrams.

2.7 Mining association rules

The previous sections have focused on classification. The second half of this chapter is focused on *partial classification* (*nugget discovery*) through *association rules*, which is the method we employ to make the novel contribution in Chapter 6. The goal of partial classification is to discover association rules that reveal characteristics of some pre-defined class(es); such rules need not cover all classes or all records in the dataset [4, 44, 43]. Rule induction algorithms (e.g. [2]) can be used to generate a set of partial classification rules. Partial classification rules provide a comprehensible set of predictors for certain outcomes. We are interested in partial classification rules with a single, fixed consequent, i.e. rules for a single class. As well as as their use for partial classification, association rules can provide insight into the data [2]. Association rules allow the end-user to achieve an understanding of the data that may not be forthcoming from other models (e.g. neural network approaches, which provide predictive power without an immediately comprehensible model, see [14]).

The problem of discovering association rules from large datasets is formulated in [2] as the market basket problem: given a large set of transactions, how can we efficiently discover the associations that hold between various items? Their approach was not targeted at discovering associations with a particular class label; rather, all associations were discovered.

The general problem of association rule mining for partial classification is as follows. Given a labelled dataset and one class label from that dataset (designated the target), find all associations (subject to certain constraints) that hold between other attributes and the target class label.

Association rules take the form:

$$Antecedent \Rightarrow Consequent$$

where the antecedent and consequent of the rule are some conjunction of *Attribute Tests* (*AT*s). An AT (see, e.g. [149]) takes the form: $< ATT, OP, VAL >$, where ATT is one of the attributes of the records in the dataset, OP is one member of the set $\{<, \leq, >, \geq, =\}$ and VAL is a permissible value for the attribute. In Chapter 6, we use Apriori on binary data. Hence, each AT is of the form: $< ATT, =, \{0, 1\} >$; an example AT is $\{Shapelet_1\} = \{0\}$. An example rule is: $\{Shapelet_1\} = \{0\} \Rightarrow \{Class\} = 1$. The class attribute is the only non-binary attribute in the data we use for rule induction in Chapter 6.

We use $N(A)$ to indicate the number of records in the dataset that satisfy the antecedent of a given rule, and $N(C)$ for the number of records that satisfy the consequent of the rule. $N(U)$ is used to represent the size of the dataset. The propositional connectives $\wedge$, $\vee$, and $\neg$ are used to indicate conjunction, disjunction, and negation. For example, $N(A \vee \neg C)$ represents the number of records that satisfy the antecedent or the negation of the consequent.

For any association rule $R = A \Rightarrow C$ (where A and C are conjunctions of ATs representing the antecedent and consequent of the rule respectively), the support of the rule ($Sup(R)$) is calculated as follows:

$$Sup(R) = N(A \wedge C) \tag{2.7.1}$$

which is the number of records in the dataset that satisfy both the antecedent and the consequent of the rule. The confidence of R (denoted $Conf(R)$) is:

$$Conf(R) = \frac{N(A \wedge C)}{N(A)}. \tag{2.7.2}$$

The confidence of a rule is the support of the rule divided by the support of the antecedent. The coverage [124] of R ($Cov(R)$) is calculated as:

$$Cov(R) = \frac{N(A \wedge C)}{N(C)}. \tag{2.7.3}$$

The coverage of a rule is the support for the rule divided by the support for the consequent, and represents the proportion of records satisfying the consequent that are correctly covered by the rule.

For any dataset that is not trivially small, there will be a large number of potential rules, and it is necessary to have an efficient algorithm to mine the data. There are many different algorithms for mining association rules, for example [3, 14, 112, 36, 149]. We use the Apriori algorithm [3] in Chapter 6; the algorithm is described in the next section.

2.8 Apriori

We use the *Apriori* [3] association rule discovery algorithm to generate rule sets for our experiments (see Algorithm 1). The Apriori algorithm operates on *itemsets*. An itemset is a combination of *items*, where each item is an AT. Itemsets are arbitrarily ordered (this ordering is used in the generateCandidates procedure). We denote the kth AT as I_k. The algorithm first determines which itemsets are *large* (above the minimum support constraint) and their support (it is common to use a proportion or percentage for the minimum support parameter; in Algorithm 1, *minsup* is assumed to be a count, so proportions, for example, need to be multiplied by the number of records). To determine the large itemsets, Apriori establishes which pairs of items have support above the minimum; these items are retained for the next pass, which finds the sets of three items that have support above the minimum, and so on, until no itemsets have sufficient support (or the maximum number of items has been reached). The maximum support of any itemset containing n items $\{I_1, I_2, ..., I_n\}$ is $Min(Sup(I_1), Sup(I_2), ..., Sup(I_n))$, so this approach is computationally more efficient than assessing every possible itemset. Itemsets are pruned if they have any subset that has not appeared in a previous pass, further reducing the final set of large itemsets, denoted L_{Tot}.

The `generateCandidates` procedure produces candidate itemsets (C_k) of k items from the set L_{k-1} (Algorithm 2). In the *join* stage, any $k-1$ itemsets that differ only in their $k-1^{th}$ item are combined to form a k itemset including the $k-1^{th}$ item of both itemsets. In the *prune* stage, any itemset generated in the join stage that includes a $k-1$ itemset not included in the set L_{k-1} is removed from the set C_k. The set is returned, the support is calculated, and the set of large k-itemsets, L_k is formed from those candidate itemsets exceeding the minimum support [3].

Algorithm 1 `apriori`(**D**, the set of all instances, *minsup*, the minimum support parameter, *minconf*, the minimum confidence parameter)

```
$L_1 \leftarrow$ {large 1-itemsets} // *Generate all large 1-itemsets*
$k \leftarrow 2$
**while** $L_{k-1} \neq \emptyset$ **do**
  $C_k \leftarrow$ generateCandidates($k, L_k - 1$) // *New candidate itemsets*
  **for** all candidate itemsets $c \in C_k$ **do**
    $c.count \leftarrow 0$
  **for** all instances $t \in D$ **do**
    $C_t \leftarrow C_k \cap \mathcal{P}(t)$ // *Candidate itemsets that are subsets of t*
    **for** all candidate itemsets $c \in C_t$ **do**
      $c.count++$
  $L_k \leftarrow \{c \in C_k : c.count \geq minsup\}$
  $k++$
$L_{Tot} \leftarrow \bigcup_k L_k$
$RS = \emptyset$; // *Generate rule set*
**for** all large $k$ itemsets $l_k \in L_{Tot}$ where $k \geq 2$ **do**
  $H \leftarrow$ {consequents of rules derived from $l_k$ with one item in the consequent}
  $RS \leftarrow RS \cup$ generateRules($k, l_k, 1, H$)
**return** $RS$
```

Once the set L_{Tot} is established, rules are generated by the procedure `generateRules` (Algorithm 3). For each large itemset (l_k), every subset a produces a rule $a \Rightarrow l - \{a\}$, which is added to the rule set (RGR) if the confidence of the rule exceeds the $minconf$ parameter. The algorithm shown here will generate all association rules above the minimum support and confidence thresholds. To generate association rules with a

Algorithm 2 `generateCandidates`(k, L_{k-1}, the set of all large $k-1$-itemsets)

```
$C_k \leftarrow \emptyset$
for all $X \in L_{k-1}, Y \in L_{k-1}$ do
  if $(X - \{X_{k-1}\} = Y - \{Y_{k-1}\}) \wedge (X_{k-1} \neq Y_{k-1})$ then
    $c \leftarrow X \cup Y$
    $C_k \leftarrow C_k \cup \{c\}$
for all $c \in C_k$ do
  for all $k-1$ subsets of $c, s$ do
    if $s \notin L_{k-1}$ then
      $C_k \leftarrow C_k - \{c\}$
return $C_k$
```

fixed, single AT consequent, we generate only itemsets that contain the consequent, and replace the `generateRules` procedure with an assessment of the confidence of the rule $R = I - \{c\} \Rightarrow c$, where I is the itemset and c is the item representing the fixed consequent.

Algorithm 3 `generateRules`(k, l_k, a k-itemset, m, H, a set of consequents)

```
$RGR \leftarrow \emptyset$
if $k > m$ then
  $m++$
  for all $h \in H$ do
    $conf \leftarrow$ support$(l_k)$/support$(l_k - h)$
    if $conf \geq minconf$ then
      $r \leftarrow < rule, conf,$support$(l_k) >$, where $rule = (l_k - \{h\}) \Rightarrow h$
      $RGR \leftarrow RGR \cup \{r\}$
    else
      $H \leftarrow H - \{h\}$
  $H \leftarrow$ generateCandidates$(H)$
  $RGR \leftarrow RGR \cup$ generateRules$(k, l_k, m, H)$
return $RGR$
```

The minimum support and minimum confidence parameters are used to reduce the size of the rule set, eliminating rules on the basis of counts from the dataset.

2.9 Interestingness measures

Rule set quality is assessed in terms of the quality of the individual rules in the set. The term typically used in the literature for the quality of a rule is *interestingness*, and we shall adopt this convention. The rules generated by algorithms may be interesting or not. Objective interestingness measures are used to assess how interesting an association rule might be from the structure of the dataset. Support and confidence are two common measures by which the interestingness of a rule is assessed, and form the basis of most objective interestingness measures.

Blanchard *et al.* [20], following [138], define a rule interestingness measure as a function from a rule onto the real numbers, which increases with $N(A \wedge C)$ and decreases with $N(A)$ when all other parameters are fixed. Piatetsky-Shapiro and Frawley [138] suggest that a good measure should decrease with $N(C)$. This is restrictive, however; interestingness measures should not have any determined behaviour with regard to $N(C)$ and $N(U)$ [20].

In [20], rules are represented as ordered quadruples of the form:

$$R =< N(A \wedge C), N(A), N(C), N(U) > . \tag{2.9.1}$$

For an association rule with a fixed consequent, $N(C)$ and $N(U)$ are constants. Hence, only the $N(A)$ and $N(A \wedge C)$ values of rules in a rule set vary. This greatly restricts the usefulness of this kind of interestingness measure. For example, [20], demonstrate that the orderings imposed by the confidence and lift measures on a rule set are not the same. However, the example in their proof uses different values for $N(C)$ between rules. With a fixed consequent, it can be proved that lift and confidence impose the same ordering [15]. Thus, for association rules with a fixed consequent, the scope for objective interestingness measures to differentiate rules is greatly diminished.

Many measures of rule interestingness have been proposed. These include support, confidence, and coverage (see Section 2.7), *novelty* [109], *relative risk* [4], *chi-square* [72, 129], *gain* [65], *k-measure* [136], *entropy* [35, 129], *Laplace accuracy* [35], *interest/lift* [95, 26], *conviction* [26, 14], and *Gini* [129]. Comprehensive surveys of interestingness measures can be found in [161, 33, 135].

The majority of interestingness measures are based on counts, such that different rules that happen to have the same counts have the same value. Over 100 of these measures are documented; however, in [15], the authors demonstrate that many different measures impose the same partial ordering on a rule set. In Section 2.10, we apply a similar line of argument to the type of rule we are interested in: partial classification rules with fixed consequent.

Subjective interestingness is a measure of how interesting the discovered rule is to a domain expert. Researchers have tested the assumption that objective and subjective interestingness are correlated [136, 33, 111, 135]. The experiments involve ranking rules based on various objective measures and on interestingness to domain experts. The performance of certain objective measures, such as relative risk [4], uncovered negative [66], and accuracy [109], was reasonable on the medical datasets studied in [135]. The effectiveness of any individual interestingness measure for a given rule set is highly correlated to the dataset in question, suggesting that this approach is unlikely to yield any projectible insight; we do not believe that the measures that performed best in the study would perform best given rule sets created from different data.

As an approach, data mining is most useful when finding rules that are difficult to find by manual analysis, and this suggests an alternative method to discover potentially interesting rules. We may assume that general statistical analysis reveals good single predictors of the class of interest. Instances where the combination of predictors produces a surprising result (poor predictors combining to form a good rule, or predicted classes changing with the addition of predictors to a rule, see [32, 139]) are of particular interest because they are unlikely to be revealed by such analysis. This is particularly evident in cases where the number of predictors is high and the dataset is very large. In Chapter 6, we propose two novel interestingness measures that assess this quality of a rule, and use them as a supplement to count-based measures for assessing rule quality.

2.10 Selecting an appropriate interestingness measure

We wish to select a count-based interestingness measure to use in our assessment of partial classification rules. In this section, we examine a number of commonly used interestingness measures, and show that, under conditions that hold for our particular area of interest, they all impose the same ordering as confidence. This is a novel analysis of existing interestingness measures that shows they are not fit for our purpose.

We focus on partial classification rules with a fixed consequent, derived from a fixed dataset. Hence, we make the following assumptions:

1. The dataset is fixed. That is, $N(U)$ is a constant. We do not compare rules across datasets.

2. The consequent of the rule is fixed. That is, $N(C)$ is a constant, as is $N(\neg C)$.

Under these assumptions, we prove theoretically that twelve interestingness measures proposed in the literature are monotonic with respect to confidence. Several of the proofs rely on the following equivalence:

$$\frac{N(A \wedge \neg C)}{N(A)} = 1 - \frac{N(A \wedge C)}{N(A)} = 1 - Conf. \tag{2.10.1}$$

Satisfaction

The formula for *Satisfaction* [109] is:

$$\frac{N(\neg C) \times N(A) - N(A \wedge \neg C) \times N(U)}{N(C) \times N(A)}. \tag{2.10.2}$$

The formula can be rearranged to give:

$$\frac{N(\neg C)}{N(C)} - \frac{N(A \wedge \neg C)}{N(A)} \times \frac{N(U)}{N(C)} \tag{2.10.3}$$

(cancelling $N(A)$ in the first term). The first and third terms are constants, and the second term is equal to $1 - Confidence$, so satisfaction is proportional to Confidence.

Ohsaki's Conviction

Ohsaki's Conviction [136] is calculated as:

$$\frac{N(A) \times N(\neg C)^2}{N(A \wedge \neg C) \times N(U)^2}. \tag{2.10.4}$$

The formula can be rearranged as:

$$\frac{N(A)}{N(A \wedge \neg C)} \times \frac{N(\neg C)^2}{N(U)^2}. \tag{2.10.5}$$

By dividing both the numerator and denominator of the first term by $N(A)$, we have:

$$1/\frac{N(A \wedge \neg C)}{N(A)}$$

which is equal to $1/(1 - Confidence)$; the second term is a constant. Hence, Ohsaki's conviction is proportional to Confidence.

Added Value

The formula for the *Added Value* measure [179] is:

$$\frac{N(A \wedge C)}{N(A)} - \frac{N(C)}{N(U)}. \tag{2.10.6}$$

As the second term is a constant, this measure is proportional to Confidence.

Brin's Interest/Lift/Strength

Brin's Interest/Lift/Strength [26, 14, 48] is calculated as:

$$\frac{N(A \wedge C)}{N(A)} \times \frac{N(U)}{N(C)}. \tag{2.10.7}$$

The second term is fixed, so this measure is proportional to Confidence.

Brin's Conviction

Brin's Conviction [26] is:

$$\frac{N(A) \times N(\neg C)}{N(U) \times N(A \wedge \neg C)}. \tag{2.10.8}$$

As shown in [15], the formula can be rearranged by dividing both the numerator and the denominator by $N(A)$, giving

$$\frac{N(\neg C)}{N(U) \times N(A \wedge \neg C)/N(A)}. \tag{2.10.9}$$

$\frac{N(A\wedge\neg C)}{N(A)}$ is equal to $1-Confidence$. $N(U)$ and $N(\neg C)$ are fixed, so Brin's Conviction is monotonic with respect to Confidence.

Certainty Factor/Loevinger

The formula for *Certainty Factor/Loevinger* [161, 110] is:

$$\left(\frac{N(A \wedge C)}{N(A)} \times \frac{N(U)}{N(\neg C)}\right) - \frac{N(C)}{N(\neg C)}. \tag{2.10.10}$$

Both $\frac{N(U)}{N(\neg C)}$ and $\frac{N(C)}{N(\neg C)}$ are constants, so Certainty Factor/Loevinger is proportional to Confidence.

Mutual Information

The *Mutual Information* measure [161] is:

$$log_2\left(\frac{N(A \wedge C)}{N(A)} \times \frac{N(U)}{N(C)}\right). \tag{2.10.11}$$

The log_2 function is monotonic, and $\frac{N(U)}{N(C)}$ is a constant, so mutual information is proportional to Confidence.

Interestingness

Interestingness [178] is calculated as follows:

$$\frac{N(A \wedge C)}{N(A)} \times log_2 \left(\frac{N(A \wedge C)}{N(A)} \times \frac{N(U)}{N(C)} \right). \tag{2.10.12}$$

Interestingness is Confidence multiplied by the Mutual Information measure. Hence, it is monotonic with respect to Confidence.

Sebag-Schonauer

The *Sebag-Schonauer* measure [155] is:

$$\frac{N(A \wedge C)}{N(A \wedge \neg C)}. \tag{2.10.13}$$

By dividing both the numerator and the denominator by $N(A)$, we see that the formula is equivalent to $Confidence/(1 - Confidence)$, which is proportional to Confidence. This measure is proportional to Confidence even if we relax the assumption that the consequent is fixed.

Ganascia Index

The *Ganascia index* [67] for a rule is:

$$\frac{N(A \wedge C) - N(A \wedge \neg C)}{N(A)}. \tag{2.10.14}$$

$N(A \wedge \neg C) = N(A) - N(A \wedge C)$, so the numerator is equal to:

$$N(A \wedge C) - N(A) + N(A \wedge C) = 2N(A \wedge C) - N(A). \tag{2.10.15}$$

Hence, the formula can be rearranged as $2\frac{N(A \wedge C)}{N(A)} - \frac{N(A)}{N(A)}$, which is proportional to Confidence. Like Sebag-Schonauer, Ganascia Index is proportional to Confidence even if we relax our assumption of fixed consequent.

Odd Multiplier

Odd Multiplier [69] is calculated as:

$$\frac{N(A \wedge C) \times N(\neg C)}{N(C) \times N(A \wedge \neg C)}. \tag{2.10.16}$$

We rearrange the formula as:

$$\frac{N(A \wedge C)}{N(A \wedge \neg C)} \times \frac{N(\neg C)}{N(C)}. \tag{2.10.17}$$

The second term is a constant, and the first term is the Sebag-Shonauer measure, which is proportional to Confidence. Hence, odd multiplier is proportional to Confidence.

Example/counter-example Rate

We calculate the *Example/counter-example Rate* [94] as:

$$\frac{N(A \wedge C) - N(A \wedge \neg C)}{N(A \wedge C)}. \tag{2.10.18}$$

The formula can be rearranged as:

$$\frac{N(A \wedge C)}{N(A \wedge C)} - \frac{N(A \wedge \neg C)}{N(A \wedge C)}. \tag{2.10.19}$$

By dividing both the numerator and the denominator of the second term by $N(A)$, and cancelling $N(A \wedge C)$ in the first term, we have $1 - \frac{1-Confidence}{Confidence}$, which is proportional to Confidence. This measure is proportional to Confidence without the assumption that the consequent is fixed.

2.10.1 Analysis

We have shown that a large number of commonly-used interestingness measures impose the same ordering on a set of partial classification rules with a fixed consequent as confidence. On the basis of this, we proceed with the view that confidence is an appropriate interestingness measure for rule induction tasks of the type we are interested in, and that there is little benefit to be gained by employing any of the other measures we have examined in detail.

2.11 Mining rules from time-series data

Mining association rules from time-series data using a market-basket approach requires transforming the time-series into a representation that is compatible with Apriori (or other market-basket association rule algorithms). One canonical approach is [39], where repeated patterns are discovered in time series; the patterns are the input used for finding association rules. Keogh and Lin [100] criticise the approach in [39], and similar approaches, on the grounds that the discovered patterns are meaningless because no account has been taken of overlapping subsequences. We discuss this point in greater length in Chapters 3 and 5.

Our own contribution to the problem of finding repeated patterns in time-series can be found in Chapter 7. Our interest does not lie in discovering rules in long time series, however; we wish to discover partial classification rules that can be used for (partial) TSC. We do this by transforming the data (Chapter 5) before building rule sets (Chapter 6). Our rule induction from time series is much more closely aligned with the type of rule discovery discussed in this chapter, and with TSC, than is the approach in, for example, [39].

2.12 Conclusions

In this chapter, we describe two areas of data mining that are key to our novel contributions: TSC and partial classification using association rules.

In the first half of the chapter, we discuss classification, time-series, TSC, and our particular interest in classifying time-series by similarity in shape. Specifically to support Chapter 5, we outline the classifiers that we use and 1NN with DTW distance (a benchmark to which we compare our method), and review ensembling, which we employ to improve accuracy. We also describe the statistical tests we use when comparing our methods with other approaches in Chapters 5 and 6.

In the second half of the chapter, we discuss partial classification with association rules, the focus of Chapter 6. We describe the algorithm we use, Apriori, and how we select interestingness measures for the algorithms and assessment methods in that chapter.

The next chapter focuses on a particular aspect of TSC that is key to our novel contributions: local similarity of shape.

Chapter 3

Representing Time Series with Localised Shapes

A shapelet is a time-series subsequence that can be used as a primitive for TSC based on local, phase-independent similarity in shape (Figure 3.1). Shapelet-based classification involves measuring the similarity between a shapelet and each series, then using this similarity as a discriminatory feature for classification.

In Section 3.1, we define shapelets and shapelet candidates, present a generic algorithm for shapelet discovery, and define two central concepts: shapelet distance and shapelet quality. In Section 3.2, we present and analyse a number of techniques from the literature for speeding up the shapelet search. In Section 3.3, we discuss a number of different domains where shapelets have been successfully applied to classification problems, including image outlines, motion capture, and spectrographs. In Section 3.4, we examine some extensions to the shapelet approach that have been suggested in the literature. Finally, in Section 3.5, we discuss an unsupervised use of local-shape-based similarity, where repeated patterns, called *motifs*, are mined from time-series data.

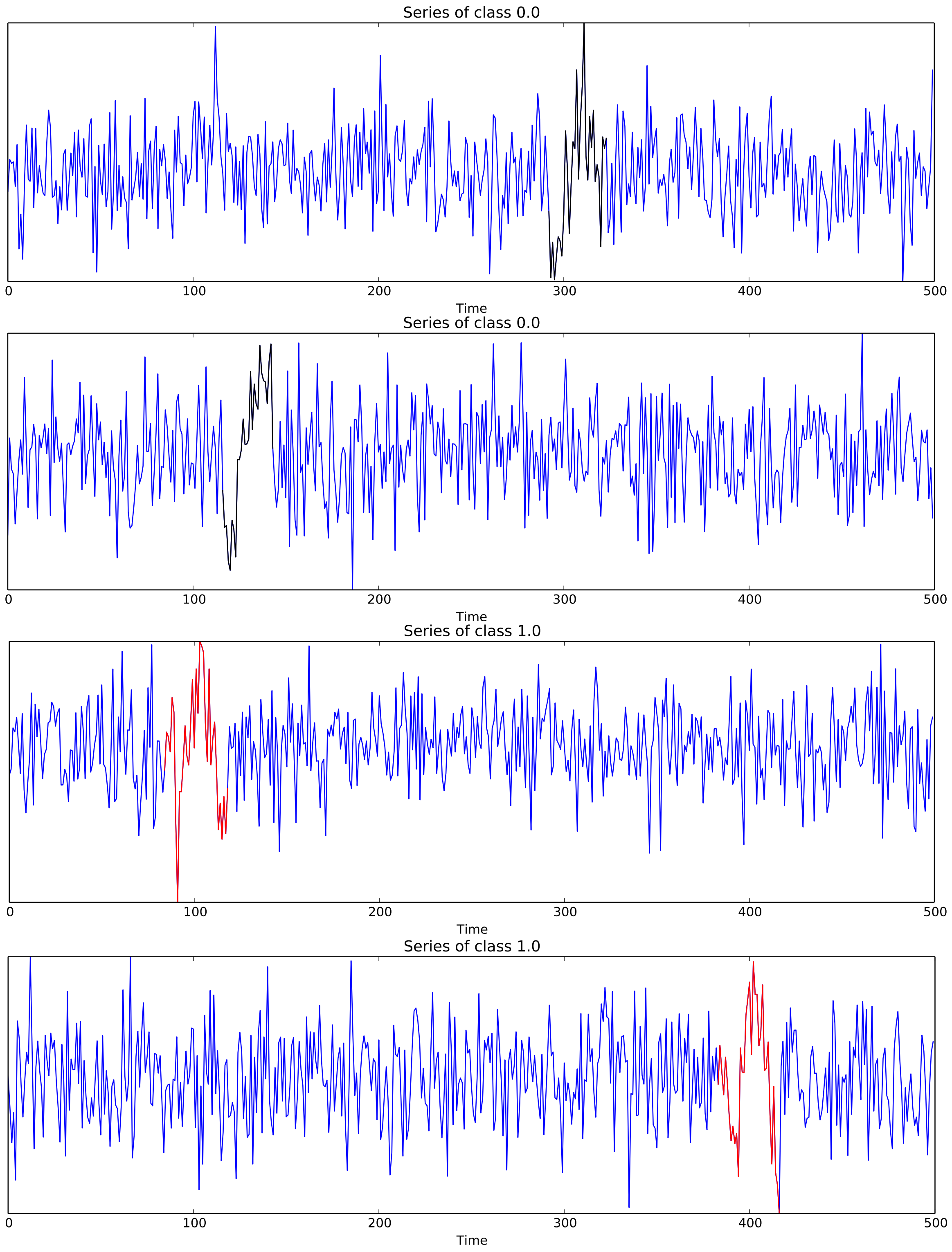

Figure 3.1: Four series from our simulated dataset (see Chapter 4). The shapelets for class 0 are highlighted in black, for class 1 in red. The simulated data demonstrates the utility of shapelets for detecting phase-independent local similarity of shape in noisy data.

3.1 Shapelet definition

A time-series dataset $\mathbf{T} = \{T_1, T_2, ..., T_n\}$ is a set of n time series. A length m time series $T_i =< t_{i,1}, t_{i,2}, ..., t_{i,m} >$ is an ordered set of m real numbers. A length l subsequence of T_i is an ordered set of l contiguous values from T_i. T_i has a set $W_{i,l}$ of $(m-l)+1$ subsequences of length l. Each subsequence $w =< t_j, t_{j+1}, ..., t_{j+l} >$ in $W_{i,l}$ is a time series of length l where $1 \leq j < m-l$.

A shapelet is a subsequence of one time series in a dataset $\mathbf{T}$ (see [180, 181]) that is discriminative of the class of the series. A good shapelet is a subsequence found only in series of a particular class (this is for the binary case. For multi-class problems, good shapelets may appear in more than one class - the important feature is that they appear in some and not in others). Whether or not a shapelet is considered to be present in a series is determined by a distance calculation (see Section 3.1.3). Every subsequence of every series in $\mathbf{T}$ is a *candidate* - a potential shapelet. Shapelets are found via an exhaustive search of every candidate of length min to max. Shapelets are normalised; since we are interested in detecting localised shape similarity, they must be invariant to scale and offset, and it is insufficient to normalise the series as a whole [140].

Ye and Keogh introduce shapelets in [180], and expand upon the research in [181]. They use a decision tree to classify with shapelets. Mueen, Keogh, and Young [130] propose an extension from shapelets to *logical shapelets*. The authors also provide a method to speed up the shapelet search by reusing information from distance calculations. McGovern *et al.* [125] apply the shapelet approach to weather prediction. Much of the research into shapelets has focused on methods to speed up the shapelet search. Chang *et al.* implement the shapelet search in parallel on the GPU [34]. In [73], the authors propose a randomised search for shapelets. He *et al.* experiment with speeding up the search by restricting the number of candidates that are examined [84].

There has also been an expansion in applications of the shapelet approach; examples include gait recognition [157], early classification [176, 71], and clustering [183]. The shapelet-discovery algorithm is extended with alternate quality measures [116], dissociation from the tree classifier [119], and modification into a faster system for finding approximate shapelets [141].

3.1.1 Generating candidates

A time series T_i of length m contains $(m-l)+1$ distinct candidate shapelets of length l. We denote the set of all normalised subsequences of length l of series T_i as $W_{i,l}$. Typically, a range of values will be used for l, from min to max. $\mathbf{W}_i$, the set of all sets of shapelet candidates from T_i is:

$$\mathbf{W}_i = \{W_{i,min}, W_{i,min+1}, \cdots, W_{i,max}\}.$$

$\mathbf{W}_i$ contains each set of shapelet candidates from T_i. The size of $\mathbf{W}_i$ is $O(m^2)$ where m is the series length; for a dataset of size n, there are $O(m^2n)$ shapelet candidates.

3.1.2 Shapelet algorithm

Algorithm 4 is a generic algorithm for finding the best shapelet [119]. Shapelet discovery has three main components: candidate generation, a similarity measure between a shapelet and a time series, and some measure of shapelet quality.

The `generateCandidates` method creates the set of sets of normalised subsequence of length from min to max of one time series, T_i ($\mathbf{W}_i$). The `findDistances` method returns the set of shapelet distances between the candidate and the series in $\mathbf{T}$. The `assessCandidate` method assigns a value to each subsequence. If, for example, the Information Gain (Section 3.1.4) measure is used, `assessCandidate` finds the optimum split point, and the Information Gain of the shapelet for that split point.

Algorithm 4 `shapeletDiscovery` (Dataset **T**, *min*,*max*)

```
1: $best \leftarrow 0$
2: $bestShapelet \leftarrow \emptyset$
3: for all series $T_i$ in $\mathbf{T}$ do
4:    $\mathbf{W}_i \leftarrow$ generateCandidates($T_i, min, max$)
5:    for all $W_{i,l}$ in $\mathbf{W}_i$ do
6:       for all candidates $S$ in $W_{i,l}$ do
7:          $D_S \leftarrow$ findDistances($S, \mathbf{T}$)
8:          $quality \leftarrow$ assessCandidate($S, D_S$)
9:          if $quality > best$ then
10:            $best \leftarrow quality$
11:            $bestShapelet \leftarrow S$
12: return $bestShapelet$
```

The candidates are checked in turn; the best candidate is retained as the shapelet. The best shapelet is the combination of subsequence and distance threshold that best distinguishes between classes. Our implementation of the `shapeletDiscovery` algorithm does not create the set $\mathbf{W}_i$; instead, the candidates are created as needed from the original series. We feel that referring to the set $\mathbf{W}_i$ in the pseudo-code makes the algorithm more transparent than it would be if it followed our implementation.

In [180, 181, 130], the authors classify data by embedding the shapelet search within a decision-tree classfier (Algorithm 5).

Algorithm 5 `buildShapeletTree`(**T**,*min*,*max*)

```
1: if pureClass($\mathbf{T}$) then
2:    return $< \mathbf{T} >$
3: else
4:    $S \leftarrow$ shapeletDiscovery($\mathbf{T}, min, max$)
5:    $sp \leftarrow$ findSplitPoint($S, \mathbf{T}$)
6:    $\mathbf{T_a} \leftarrow \emptyset$
7:    $\mathbf{T_b} \leftarrow \emptyset$
8:    for all Records $T$ in $\mathbf{T}$ do
9:       if sDist($S, T$) $< sp$ then
10:         $\mathbf{T_a} \leftarrow \mathbf{T_a} \bigcup \{T\}$
11:      else
12:         $\mathbf{T_b} \leftarrow \mathbf{T_b} \bigcup \{T\}$
13:   return $<$ buildShapeletTree($\mathbf{T_a}$), buildShapeletTree($\mathbf{T_b}$) $>$
```

The `pureClass` method returns true if and only if the class labels of dataset **T** are all the same (Algorithm 5 is for a decision tree with no pruning - see [25]). The algorithm finds the shapelet in the initial case, and after each split, and splits the data using the shapelet, just as a standard decision tree splits on attributes values, until each node is pure. A pure node is designated a leaf, and terminates that branch of the algorithm.

Integrating the classification with the shapelet search has a number of disadvantages. First, the classification accuracy can suffer compared to alternative classification algorithms. In Chapter 5, we show that decision tree implementations of the shapelet approach are less accurate classifiers than our method. The second disadvantage of embedding the shapelet search within a decision tree is that it requires the search to be performed at every branch of the tree. Searching for shapelets is time intensive, and a deep decision tree will require many searches. The third disadvantage is one of interpretability. Decision trees with multiple layers can be difficult to interpret, as they entail a hierarchy of binary splits, with each subsequent split conditional on the splits above it. This diminishes the interpretability of the process, which is one of the intended qualities of the shapelet approach.

3.1.3 Shapelet distance

The squared Euclidean distance between two subsequences S and R, where both are of length l, is defined as:

$$dist(S,R) = \sum_{i=1}^{l}(s_i - r_i)^2. \tag{3.1.1}$$

For a shapelet S of length l, and a time series T, the *shapelet distance* (*sDist*) is the minimum squared Euclidean distance between the shapelet and any length l subsequence of T, where the set of length l subsequences of T is W_l:

$$sDist(S,T) = \min_{w \in W_l}(dist(s,w)). \tag{3.1.2}$$

For any shapelet and series, the *sDist* is the squared Euclidean distance between the shapelet and the closest subsequence of the series of the same length as the shapelet.

3.1.4 Shapelet quality with Information Gain

The method `assessCandidate` in Algorithm 4 assesses shapelet quality. Shapelet quality is based on how well the class values are separated by the set of distances between the shapelet and the series in **T**. The standard approach [180, 181, 130] is to use Information Gain (IG) [158] to determine the quality of a shapelet. The IG of a split is the difference between the entropy of the whole dataset, and the sum of the entropies of the two halves of the split, weighted by the amount of data in each split.

The entropy, H, of a set of objects, **T**, is:

$$H(\mathbf{T}) = -p(A)\log(p(A)) - p(B)\log(p(B)), \tag{3.1.3}$$

where $p(X)$ is the proportion of objects of class X in the set (this definition is for two-class problems, and is trivial to extend to multi-class problems).

A splitting strategy, in this case a shapelet and a distance threshold, divides **T** into two disjoint subsets, $\mathbf{T_a}$ and $\mathbf{T_b}$, where $\mathbf{T_a} \cap \mathbf{T_b} = \emptyset$ and $\mathbf{T_a} \cup \mathbf{T_b} = \mathbf{T}$. For shapelet S, distance threshold sp, and series T_i, T_i is assigned to $\mathbf{T_a}$ if and only if $sDist(S, T_i) < sp$. If $sDist(S, T_i) \geq sp$, then T_i is assigned to $\mathbf{T}_b$.

The weighted average entropy, $\hat{H}$, of a split (a shapelet and distance threshold), S_D, is:

$$\hat{H}(S_D) = \frac{|\mathbf{T}_a|}{|\mathbf{T}|}H(\mathbf{T}_a) + \frac{|\mathbf{T}_b|}{|\mathbf{T}|}H(\mathbf{T}_b). \tag{3.1.4}$$

The total Information Gain for a split S_D on dataset **T** is:

$$IG(S_D) = H(\mathbf{T}) - \hat{H}(S_D). \tag{3.1.5}$$

The IG quality measure of a shapelet, S, is the highest IG of any split point, S_D, in the set of possible splits, $\mathbf{S_D}$:

$$IG(S) = \max_{S_D \in \mathbf{S_D}} IG(S_D). \tag{3.1.6}$$

The IG calculation requires sorting the set of distances and then evaluating all split points. This introduces a time overhead of $O(n \log n)$ for each shapelet, although this is generally trivial in comparison to the time taken to calculate the set of distances, which is $O(nml)$.

3.2 Speed-up techniques for distance calculation

The major weakness of the shapelet approach is the time required to find shapelets. The exhaustive search requires each subsequence of each series of each length to be compared to each subsequence of that length of every other series in the training set.

There are $n(m-l+1)$ candidates for any given shapelet length l. Finding the set of distances for a single candidate requires a scan along every series, performing $O(m)$ distance function calls, each of which requires $O(l)$ pointwise operations. Hence, the complexity for a single shapelet is $O(nml)$, and the full search is $O(n^2m^4)$. Clearly, this is untenable for even moderately large datasets.

In this section, we review a number of different methods to increase the speed of the shapelet search.

3.2.1 Early abandon of the shapelet

An early abandon of the shapelet assessment, *entropy pruning*, is proposed in [180, 181]. After the calculation of each distance between the shapelet and a series, an upper bound on the IG is found by assuming the most optimistic future assignment. If this upper bound falls below the best found so far, the candidate shapelet can be abandoned. This modification offers a potentially huge speed up at a small extra overhead for calculating the best split and upper bound for each new distance. Entropy pruning can reduce the time taken to find a shapelet by more than two orders of magnitude [180]. This is an empirical finding; it may not hold for all cases. One disadvantage is that, for multi-class problems, a correct upper bound can be found only through enumerating split assignments for all possible classes, which can dramatically increase the overhead.

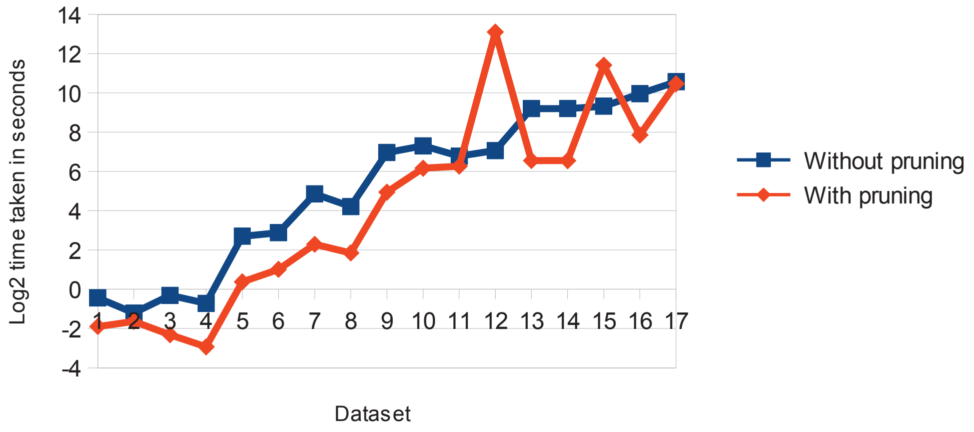

Figure 3.2: Graph of speed up offered by early abandon of the candidate shapelet on 17 UCR datasets. Graph taken from [11].

Experiments performed by our research group [11] found that early abandon of the shapelet candidate delivered a 68% speed up on finding the shapelet on the set of UCR datasets with four classes or fewer (Fig. 3.2). With more than four classes, the speed up diminished, as the overhead for calculating the optimistic split with IG increases with the number of classes.

3.2.2 Speed improvement by reusing information

Mueen *et al.* [130] offer two contributions to improve the speed of shapelet algorithms: a method to reduce the time complexity of normalised Euclidean distance calculations by storing summary statistics, and a form of candidate pruning that uses existing calculated values.

Reusing information can decrease the time complexity of the shapelet search; because every subsequence is compared to every other, there is duplication in the calculations. For example, when the subsequence starting at position a is compared to the subsequence at position b, many of the calculations that were previously performed in comparing the subsequence starting at position $a-1$ to the one starting at $b-1$ are duplicated. A method involving trading memory for speed is proposed in [130]. For each pair of series T_i, T_j, cumulative sum, squared sum, and cross products of T_i and T_j are precalculated. With these statistics, the distance between subsequences can be calculated in constant time, making the shaplet-discovery algorithm $O(n^2m^3)$. However, precalculating of the cross products between all series prior to shapelet discovery requires $O(n^2m^2)$ memory, which is infeasible for most problems. Instead, [130] propose calculating these statistics prior to the start of the scan of each series, reducing the requirement to $O(nm^2)$ memory, but increasing the time overhead.

The normalised Euclidean distance between two series, X and Y is calculated in amortised constant time by storing five statistics; $\sum X, \sum Y, \sum X^2, \sum Y^2$, and $\sum XY$. The stored statistics allow computations to be reused, exchanging time complexity for space requirements. If X and Y are of length m, the mean of X, $\bar{X}$, is:

$$\bar{X} = \frac{1}{m}\sum X, \tag{3.2.1}$$

and the variance, s_X^2 (the square of the standard deviation, s_X), is:

$$s_X^2 = \frac{1}{m}\sum X^2 - \bar{X}^2. \tag{3.2.2}$$

The same is true for Y, respectively. The positive correlation:

$$C(X,Y) = \frac{\sum XY - m\bar{X}\bar{Y}}{ms_X s_Y}, \tag{3.2.3}$$

is used to compute the normalised Euclidean distance as follows:

$$dist(X,Y) = \sqrt{2(1 - C(X,Y))}. \tag{3.2.4}$$

The best distance between a subsequence, x, and a time series, Y is the *subsequence distance*:

$$sDist(x, Y) = \sqrt{2(1 - C_s(x, Y))} \tag{3.2.5}$$

where:

$$C_s(x, Y) = \min_{0 \leq v \leq n-l} \frac{\sum_{i=0}^{l-1} x_i Y_{i+v} - l\bar{x}\bar{y}}{l s_x s_y}, \tag{3.2.6}$$

x is a length l candidate shapelet taken from X beginning at index u, Y is a length n time series, y is a length l subsequence of Y starting at index v, $l \leq n$, and x is offset relative to Y by v. $\bar{y}$ and s_y denote the mean and standard deviation of y, a subsequence representing l consecutive values from Y starting at position v. Changing the offset v in the *min* function is equivalent to sliding the candidate shapelet x against the series Y.

For every pair of points, five arrays are computed, saving the cumulative sums for X and Y ($\mathbb{S}_X$,$\mathbb{S}_Y$), the cumulative sums of X^2 and Y^2 ($\mathbb{S}^2_X$,$\mathbb{S}^2_Y$), and a $2D$ array that stores the sum of the product of X and Y for different subsequences ($\mathbb{M}$). The mean, variance, and sum of products for any two length l subsequences x and y can be calculated as:

$$\bar{x} = \frac{\mathbb{S}_X[u + l - 1] - \mathbb{S}_X[u - 1]}{l} \tag{3.2.7}$$

$$\bar{y} = \frac{\mathbb{S}_Y[v + l - 1] - \mathbb{S}_Y[v - 1]}{l} \tag{3.2.8}$$

$$s_x^2 = \frac{\mathbb{S}^2_X[u + l - 1] - \mathbb{S}^2_X[u - 1]}{l} - \bar{x}^2 \tag{3.2.9}$$

$$s_y^2 = \frac{\mathbb{S}^2_Y[v + l - 1] - \mathbb{S}^2_Y[v - 1]}{l} - \bar{y}^2 \tag{3.2.10}$$

$$\sum_{i=0}^{l-1} X_{u+i} Y_{v+i} = \mathbb{M}[u + l - 1, v + l - 1] - \mathbb{M}[u - 1, v - 1], \tag{3.2.11}$$

where the starting index of x in relation to X is u (fixed for a given candidate shapelet), and the starting index of y in relation to Y is v (dependent on the offset).

From these values, the normalised Euclidean distance can be calculated in constant time, speeding up the search by a factor of m.

Mueen *et al.* [130] also propose candidate pruning with reused calculations, based on the intuition that, if a candidate is a poor shapelet, any similar subsequence can be abandoned. This is exact; no good shapelet candidate will be abandoned. Let the distance between S_i and S_{i+1} be R. By the triangle inequality, the distance between S_{i+1} and some subsequence D_k is $sdist(S_i, D_k) - R \leq sdist(S_{i+i}, D_k) \leq sdist(S_i, D_k) + R$. The points on the order line for the original shapelet are shifted optimistically by R in either direction, giving the best possible information gain for the new candidate. This optimistic IG can be used to prune shapelets that are worse than the best-so-far.

The `findDistances` method in line six of Algorithm 4 can be modified to incorporate the use of pre-calculated statistics, reducing the complexity of the shapelet search from $O(n^2m^4)$ to $O(n^2m^3)$. The space complexity can be as high as $O(nm^2)$, however. This is an untenable memory footprint for large datasets.

Research performed by our group on the UCR datasets (see Figure 3.3) found that using pre-calculated statistics offers an average speed up of 88% for finding the first shapelet [11].

3.2.3 Speed improvements from distance calculations

$sDist(S, T_i)$ is the minimum of the $m - l + 1$ subsequence distances between S and T_i. Hence, individual calculations can be abandoned if they are larger than the best found so far. Empirically, this has been shown to reduce the time taken by a constant factor of two [180].

Rakthanmanon *et al.* propose a method to speed up the indexing and querying of time-series databases. They achieve this speed up by normalising subsequences

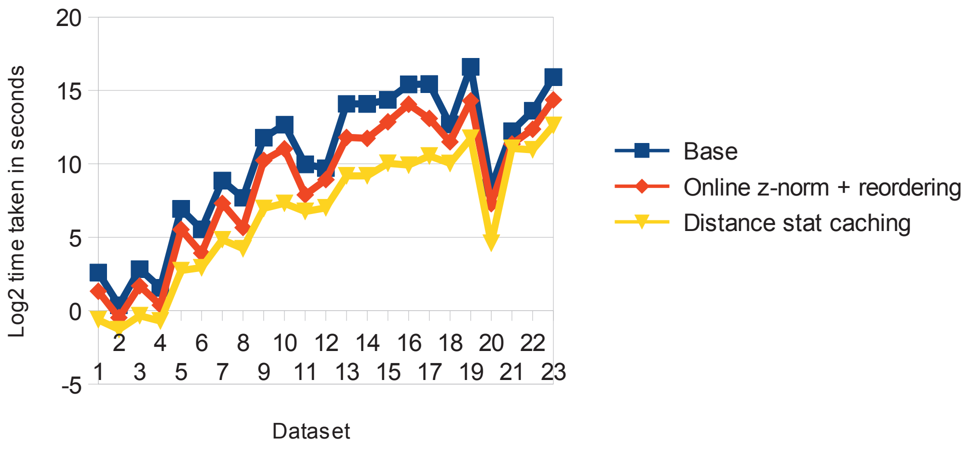

Figure 3.3: Graph comparing speed up offered by reusing information to that offered by improving the distance calculation. Graph taken from [11].

during the distance calculation, and by reordering the query sequence to compare the largest values first [140]. We modify their approach and apply it to the shapelet search, with the query subsequence replaced with the shapelet candidate. This is a novel application of their methods.

Algorithm 6 includes these speed ups, and is incorporated into the shapelet search (Algorithm 4) at line six, as part of the `findDistances` method (`findDistances` finds all *sDists* for a shapelet and set of time series).

Rakthanmanon *et al.* [140] z-normalise candidates and subsequences during the search, using summary statistics, rather than as a batch prior to the search. As the z-normalisation is incrementally computed, the Euclidean distance can also be computed. Thus, if the distance calculation is abandoned, the z-normalisation can be abandoned, reducing the number of calculations performed (see Algorithm 6).

It takes only one scan of the sample to calculate the mean ($\bar{x}$) and variance (s_x^2):

$$\bar{x} = \frac{1}{m} \sum x_j \tag{3.2.12}$$

$$s_x^2 = \frac{1}{m} \sum x_j^2 - \bar{x}^2. \tag{3.2.13}$$

The values and the squares of the values are summed, the mean is computed from the sum of the x_j terms, and the variance from the sum of x_j^2 and the mean. For subsequences, we store the cumulative sum of the values for each point in the series, and the cumulative sum of the squares of the values. This requires linear space in m, the total length.

Every subsequence must be normalised before comparison; the mean of the subsequence can be obtained by keeping two running sums of the long time series with a lag of k values, where k is the subsequence length. The variance of the subsequence can also be computed in this manner:

$$\bar{x}_{sub} = \frac{1}{k}\left(\sum_{j=0}^{i+k-1} x_j - \sum_{j=0}^{i-1} x_j\right) \tag{3.2.14}$$

$$s_{sub}^2 = \frac{1}{k}\left(\sum_{j=0}^{i+k-1} x_j^2 - \sum_{j=0}^{i-1} x_j^2\right) - \bar{x}_{sub}^2. \tag{3.2.15}$$

If i is the initial index of a subsequence of length k, the subsequence mean is computed as the cumulative sum at $i+k-1$ minus the cumulative sum at $i-1$, divided by k. The variance is the cumulative sum of squares at $i+k-1$ minus the cumulative sum of squares at $i-1$, divided by k, minus the square of the subsequence mean.

For each point in the subsequence, we normalise, then add the normalised length, abandoning if the best-so-far is exceeded. Hence, the normalisation is abandoned as well as the distance calculation.

Rather than compute the distance point-by-point left to right, Rakthanmanon *et al.* [140] propose that the distances are computed in order of the absolute value of the original normalised subsequence (see Algorithm 6). The reasoning behind this is as follows. Sorting the indexes is a cheap operation; because the other subsequences are normalised with a mean of zero, large values are more likely to produce large differences, and cause the calculation to be abandoned. The authors support their claim by comparing the true best ordering (found by brute force) with their ordering over a million subsequences taken from the ECG dataset and compared with a query subsequence. They found a 0.999 rank correlation between their ordering and the best ordering. On average, reordering the distance calculation allows distance calculations to be abandoned earlier.

Rakthanmanon *et al.* [140] report a time reduction for the shapelet search on the FaceFour dataset from 18.9 minutes to 12.5 minutes. Our group [11] has performed extensive experiments on the UCR dataset, finding an average speed increase of 63% for finding the first shapelet (see Fig. 3.3).

Algorithm 6 `findDistance`(Time series T, Shapelet S)

```
1: $S' \leftarrow$ normalise$(S, 1, l)$
2: $A \leftarrow$ sortIndexes$(S')$ $\{A_i$ is the index of the $i^{th}$ largest absolute value in $S'\}$
3: $F \leftarrow$ normalise$(T, 1, l)$
4: $p \leftarrow 0$, $q \leftarrow l$ $\{p$ stores the running sum, $q$ the running sum of squares$\}$
5: $b \leftarrow$ dist$(S, F)$ $\{$Find first distance, set to best so far, $b\}$
6: $\{$Scan through all subseries$\}$
7: for $i \leftarrow 1$ to $m - l$ do
8:     $p \leftarrow p - t_i$ $\{$Update running sums$\}$
9:     $q \leftarrow q - t_i^2$
10:    $p \leftarrow p + t_{i+l}$
11:    $q \leftarrow q + t_{i+l}^2$
12:    $\bar{x} \leftarrow \frac{p}{l}$
13:    $s \leftarrow \frac{q}{l} - \bar{x}^2$
14:    $j \leftarrow 1, d \leftarrow 0$
15:    $\{$Distance between $S$ and $< t_{i+1} \ldots t_{i+l+1} >$ with early abandon$\}$
16:    while $j \leq l$ & $d < b$ do
17:        $d \leftarrow d + \left(S_{A_j} - \frac{t_{i+A_j} - \bar{x}}{s}\right)^2$ $\{$Reordered online normalisation$\}$
18:        $j \leftarrow j + 1$
19:    if $j = l$ & $d < b$ then
20:        $b \leftarrow d$
21: return b
```

3.2.4 Using the GPU

Chang *et al.* [34] propose an algorithm that parallelises the shapelet search on the GPU, and they report that the GPU implementation is one to two orders of magnitude faster than the CPU implementation.

Using the GPU for shapelet discovery may be useful for very large datasets. Algorithmic methods, however, such as those in [140], and approximate methods such as in [73], offer speed improvements that make searching very large datasets tenable, without using the GPU. Our shapelet transform, discussed in Chapter 5, is embarrassingly parallel, such that using the GPU, or any parallel system, is trivial. Many of our experiments on large datasets make use of this feature.

3.2.5 Discrete shapelets

Discretising the shapelet search decreases the time complexity by reducing the number of distance calculations. There is no guarantee that the best shapelets will be found, however. Rakthanmanon and Keogh [141] propose an algorithm to discover discrete approximate shapelets that offers two to three orders of magnitude speed improvement over the exact brute-force algorithm, with no substantial loss of accuracy. McGovern *et al.* [125] discretise the entire set of series prior to the shapelet search. We focus on the discretisation used in the first approach; the discretisation used by McGovern *et al.* applies the Symbolic Aggregate Approximation (*SAX*) technique to whole time series, and is much simpler to implement.

The time complexity of the algorithm proposed in [141] is $O(nm^2)$. The series are represented discretely, using the SAX ([171, 115]) method. Each series is transformed into a number of SAX words; each SAX word is masked at random points, referred to as a *random projection* [28]. For each random projection, a small portion of the SAX word is masked. The results are hashed for multiple different random masks, and the collisions are summed by class. There is a very strong correlation between the best SAX words and the best shapelets.

The best SAX words are checked in the original data space as shapelet candidates. The number of iterations and the number of candidates promoted has a strong effect on the running time, but not on the accuracy, so should be kept low.

Rakthanmanon and Keogh also employ a cross-validation technique from [152] to estimate whether the shapelet approach is likely to be useful. Their results are preliminary, but look promising, although the cross-validation fold used is large enough to be a significant time factor.

3.2.6 Approximate shapelets

Shapelets can be found by random searching, possibly combined with a hill-climbing approach. Gordon *et al.* [73] propose a method for finding shapelets through a randomised algorithm called *SALSA-R*; their results show that the algorithm finds shapelets that offer comparable accuracy to those found by exhaustive search, and that they are discovered orders of magnitude more quickly.

The accuracy of a shapelet classifier does not increase monotonically with the number of shapelets examined; rather, there is a point after which accuracy begins to decrease [34, 73], possibly because the shapelet model begins to overfit the data. Hence, a random search can attain equal or greater accuracy than an exhaustive search. Random sampling works particularly well because good shapelets are tightly clustered in the shapelet space. Finding a shapelet in a cluster yields a good shapelet; the rest of the space is almost empty, so the search time is wasted.

The SALSA-R algorithm searches the shapelet space at random; the search is terminated when shapelet quality fails to improve by a specified amount over a given number of iterations. The distances searched in building the root node of the tree are saved to disk, and only the candidates already searched are used to build the remaining nodes. This reduces the search space for the child nodes. SALSA-R cannot use any pruning method that abandons distance calculations because of this.

For six of seven datasets, the accuracy of SALSA-R is within 2% of the shapelet tree from [130], and for three, it outperforms it. For most datasets, the minimum accuracy is within 10% of the final accuracy, suggesting that the algorithm does not produce poor models very often. The standard deviation of the accuracies is also small, indicating that there is little likelihood of generating a poor model. SALSA-R examines several orders of magnitude fewer shapelets than the exhaustive search, and this is reflected in the search speed. The advantage is greater for larger datasets.

Hartman *et al.* [82] use an evolutionary algorithm that finds prototypes (shapelets, see Section 3.4.5) with, on average, more than 95% of the maximal target function score, while searching only 5% of the full target space. The maximal target function score is equivalent to a shapelet quality measure. Hence, their algorithm offers a large increase in speed in exchange for a small loss in shapelet quality. The algorithm begins with five randomly-selected candidates, which are randomly permuted to produce children. The best five shapelets are selected from the set of ten as the next generation. This is iterated over a large number of generations to find approximate shapelets.

Approximate approaches appear to offer an avenue by which to scale shapelet classification to very large datasets.

3.3 Shapelet applications

Applications for shapelets include image-outlines, motion-capture data, and spectrographs, among others. In this section, we review a number of these applications.

3.3.1 Image-outline classification

Intuitively, shapelets are suited to image classification based on outlines. Image outlines are often distinguished by local features, and may have large intraclass variation due to different rotation, zoom, and light intensity between images.

Image outlines are converted into $1D$ series by measuring the distance between a

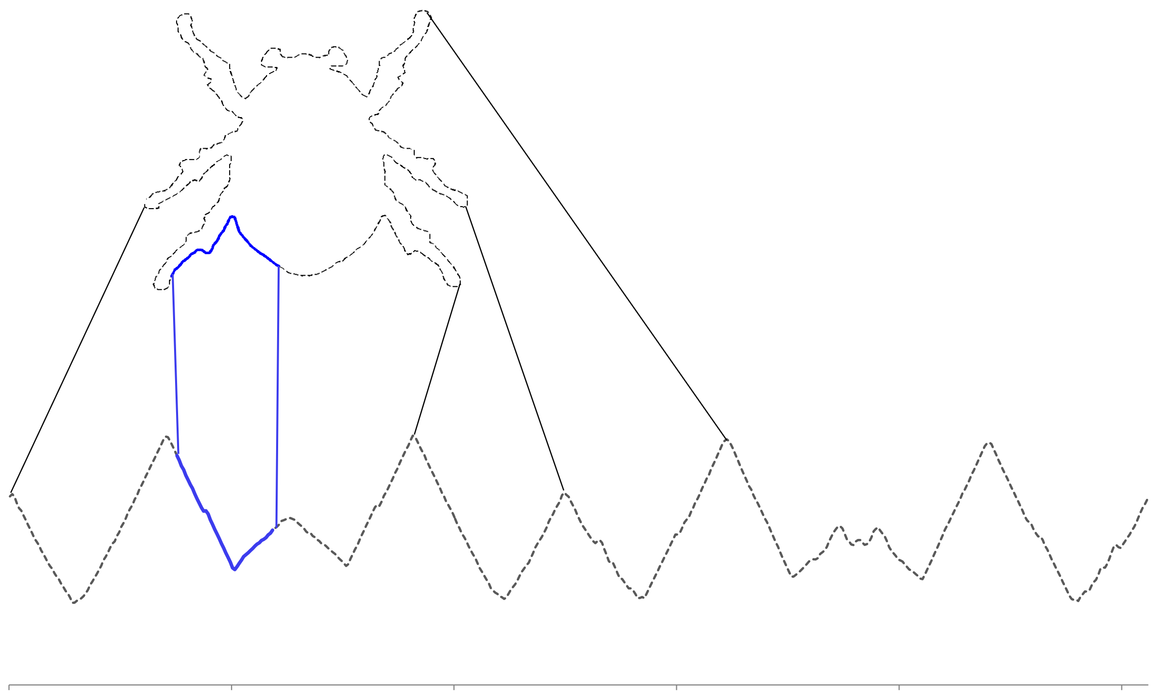

Figure 3.4: A beetle outline unfolded into a 1D series. The area highlighted in blue, and the corresponding subsequence of the series, is the best shapelet for the Beetle/Fly dataset.

central point and each point on the outline; see Fig. 3.4. The number of points used to form the series determines the granularity of the representation.

There are a number of image-outline-classification examples in the literature. Ye and Keogh [180] classify leaf outlines, and, in [141], researchers from the same group use shapelets to classify skull outlines. The problems are contrived, but suggest an application area. Ye and Keogh also apply shapelets to two genuine image-classification problems: classifying arrowheads, and classifying heraldic shields from historical documents [180]. For these two problems, the shapelet tree is more accurate than 1NN, the next best classifier. The heraldic shield problem is especially interesting because it has clear applications to image mining heterogeneous data.

We apply the shapelet approach to a number of image-outline-classification problems, including classifying image outlines of bones in the hand, the MPEG-7 image dataset, and fish type (see Chapter 4 for information on these problems, and Chapter 5 for our experiments and results).

3.3.2 Motion capture

Shapelets have been applied to motion-capture data. For gesture recognition [81, 82], movement analysis [180, 119], or gait recognition [157, 181], it is likely that local movements will be better discriminative features than the whole series, which can be noisy, with unclear start and end points, and large intraclass variation.

The *Gun/No Gun* problem is based on motion-capture data. Each series represents the movement of an actor drawing a gun. The classification problem is to distinguish whether the actor is holding a prop. The task is difficult because there is large intraclass variation, and noise in measurement and indexing. The shapelet tree [180] is over 93% accurate, beating 1NN with DTW. The top shapelet reveals interpretable information. Actors without props 'overshoot' the holster position. Lines *et al.* [119] also discuss this problem, and the work is extended in Chapter 5. Figure 3.5 shows the best two shapelets from the Gun/No Gun problem. The best shapelet represents overshoot when the prop is absent, the second best the extra movement required to lift the gun when the prop is present.

Shapelets are applied to gait analysis in [181, 157]. The work in [157] is preliminary, but they anticipate using shapelets to distinguish individuals by their gait. Ye and Keogh [181] analyse data from the CMU Graphics Lab Motion Capture Database (`http://mocap.cs.cmu.edu/`). By aligning two time series, they generate sets of $2D$ shapelets, where a pair of subsequences is the candidate. Ye and Keogh's shapelet classifier has much better accuracy than rotation-invariant kNN, unless intricate segmentation is performed on the data, in which case the two classifiers converge. Such segmentation is not scalable, making the shapelet tree preferable for large datasets.

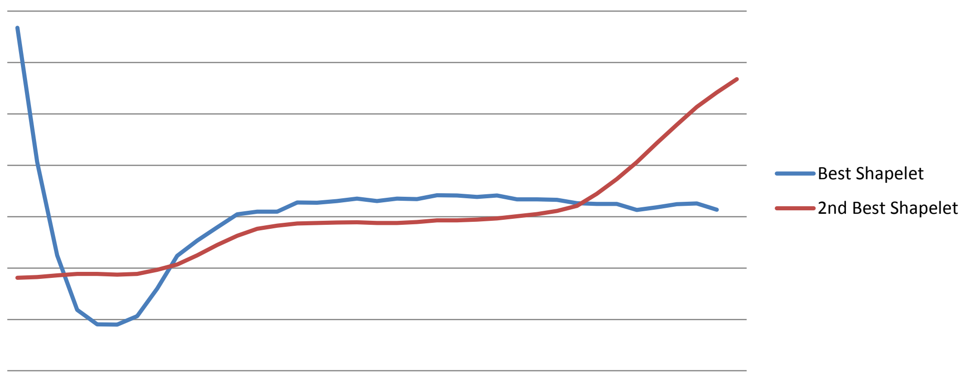

Figure 3.5: The best two shapelets from the GunPoint dataset. The best shapelet (blue) demonstrates overshoot: the hand is lowered too far at the end of the movement because there is no prop. The other shapelet shows the extra movement required to lift the gun when the prop is present.

Hartmann *et al.* and Hartmann and Link [82, 81] apply a shapelet variant to gesture recognition. They refer to their shapelets as *representative prototypes*, and use DTW distance, rather than Euclidean, with flexible start and end points for the warping path. Their prototypes tolerate noise in indexing and focus on local similarity.

Appropriate prototypes are chosen based on minimising intraclass distance (distance to members of the same class) and maximising interclass distance (distance to members of other classes). The authors use a number of functions to measure this; they show that the most effective are the simple *Minimal Interclass to Maximal Intraclass Distance (min_max)*:

$$min\_max = min_{j,k}(DTW(C_{opt}, T'_{j,k})) - max_i(DTW(C_{opt}, C'_i)), \qquad (3.3.1)$$

and the complex *Error Function Integral (erf_int)*. *min_max* is the shortest distance to a time series of the non-target class minus the longest distance to a time series of the target class. The distance threshold used for a given prototype is the mean distance to the target class members plus two standard deviations.

The authors use range normalisation:

$$A_{norm} = \frac{A - min(A)}{max(A) - min(A)}, \tag{3.3.2}$$

rather than z-normalisation. Range normalisation normalises to the [0,1] range; the *min* and *max* values are restricted by training set evaluation to prevent extreme scaling.

Edgcomb and Vahid [55] use shapelets to detect falls in privacy-enhanced video - video where the appearance of the subject is obscured. They achieve similar accuracies on the privacy-enhanced video as on the raw video using the shapelet classifier.

3.3.3 Spectrographs

Shapelets are effective for classifying spectrographs. Spectrography involves the measurement of radiated light from a sample. The light is radiated at different frequencies, and can be used to determine the nature of the sample. For example, different varieties of coffee can be distinguished by their spectrographs.

Ye and Keogh [180] demonstrate that the shapelet tree is considerably more accurate than 1NN on the wheat-spectrography dataset. In the follow-up paper [181], they show that the shapelet tree has perfect accuracy on the coffee-spectrography dataset, even when very little training data is used.

In Chapter 5, we show that classifying with shapelet-transformed data increases accuracy over classifying with untransformed data for the Beef- and Coffee-spectrography datasets. Using shapelets to classify spectrographs merits more research, as the preliminary results have been excellent, and there are applications in, for example, food testing.

3.3.4 Other applications

Ye and Keogh [180] test their shapelet tree on synthetic data, beating 1NN Euclidean when 2% or more of the data are examined. The shapelet tree is also more robust to noise, and a faster classifier once training is complete. On the Mallat dataset, Ye and Keogh [180] show that using 2/15 of the data to train a shapelet tree yields better results than a CART tree trained with 1/3 of the data. This is significant because the shapelet tree can be trained more quickly with a smaller dataset.

McGovern *et al.* [125] use multi-dimensional shapelets on discretised time series for tornado prediction. They define a multi-dimensional shapelet as a set of single-dimensional shapelets, where the $i + 1^{th}$ $1D$ shapelet begins after or simultaneously with the i^{th} $1D$ shapelet. This is a looser definition than is used for motion-capture data in [181]; the $2D$ shapelets used there represent a special case of the shapelets defined in [125].

The algorithm they use to find multi-dimensional shapelets involves finding single-dimensional shapelets meeting certain criteria (for example, the ratio of true positives to false positives). The algorithm searches across dimensions for other single-dimensional shapelets to add to the multi-dimensional shapelet. As each shapelet is a good predictor, the conjunction of shapelets should be a better predictor (but may cover fewer cases).

It is an open problem to extend the shapelet approach to multiple dimensions for non-discretised series. The main issue is retaining scalability. Speed-up techniques (discussed in Section 3.2) offer the most likely solution to this problem.

3.4 Extensions to the shapelet approach

Most research into shapelets has concentrated on time-series classification with the shapelet tree of [180, 181]. There are other methods that make use of the shapelet intuition; they are addressed in this section.

3.4.1 Logical Shapelets

Mueen *et al.* [130] propose a method to improve the descriptive power of shapelets by using conjunctions and disjunctions of shapelets. Effectively, the recursive search at each node of the tree is the same as that of [180, 181], but performed multiple times until one class in the data is partitioned wholly into one side of the binary split. Logical Shapelets improves the power of the shapelet tree on three datasets.

3.4.2 Early classification

Xing *et al.* [175] use shapelets for *early classification*, where predictions are made before the whole series is examined. Problems that benefit from early classification include [77], where sepsis in infants is detected early by observing ECG data, or [17], who show that the traffic flow of a TCP connection can be classified by observing only the first five packages.

Xing *et al.* use shapelets because early classification must be interpretable if professionals are to trust the system, and because shapelets capture local similarity, which is necessary for early classification. *Earliest match length* (EML) is a measure of the utility of a shapelet for early classification:

$$EML(f,t) = \min_{len(s) \leq i \leq len(t)} dist(t[i - len(s) + 1, i], s) \leq \delta, \tag{3.4.1}$$

where $f = (s, \delta, c)$, s is a shapelet, δ is the distance threshold associated with the shapelet, and c is the class label associated with that shapelet. EML is the first index of the first subsequence in t that is within the threshold for f. This measure could be incorporated easily into our shapelet transform, creating an early classifier.

In [71], Ghalwash *et al.* use multivariate shapelets for early classification of medical time series. They focus on producing results that can be used by medical professionals, who may be uncomfortable with uninterpretable, black-box methods. They present the task of finding multivariate shapelets as one of optimisation, from a starting point of having extracted all shapelets from each dimension.

3.4.3 Shapelets for clustering

Zakaria *et al.* [183] perform clustering using shapelets, and show that it can be more effective than clustering using the whole series. An *unsupervised shapelet* (*u-shapelet*) is a subsequence of a time series for which the distances between the subsequence and one group of time series are much smaller than those to another group of time series.

The algorithm searches for a u-shapelet that can separate a subset of the data, which is then removed for the next iteration, until no data remain to be separated. The u-shapelets are found by a greedy search aimed at maximising the *gap* between two subsets of the data:

$$gap = \bar{x}_B - s_B - (\bar{x}_A + s_A), \tag{3.4.2}$$

where $\bar{x}$ is the mean distance between the u-shapelet and the members of subset X, and s_X is the standard deviation of the distances between the members of subset X and the u-shapelet. All subsequences of the time series are candidates, and have their distance vectors computed. The vector represents an *orderline*, which is searched to find the optimum split point that maximises the gap function. D_A is the set of points to the left of the split, D_B the set of points to the right. Pathological splits (e.g. splits with all cases except a single outlier in one group) are prevented by checking that the ratio of D_A to D_B is within the range

$$\frac{1}{k} < \frac{|D_A|}{|D_B|} < \left(1 - \frac{1}{k}\right), \tag{3.4.3}$$

where k is the total number of clusters. The optimum split point for an orderline is the point that minimises the distance to D_A while maximising the distance to D_B. The u-shapelet candidate giving the split point with the best gap value is selected as the u-shapelet.

After a u-shapelet is selected, time series containing similar subsequences to the u-shapelet are removed from the dataset. A time series is considered to contain a similar subsequence if the distance between the subsequence and the u-shapelet is less than the mean distance to D_A plus one standard deviation. Zakaria *et al.* show that, for clustering of rock spectral signatures, synthetic data, electrical devices, and ECG data, u-shapelets generally find better clusters than if whole series are used.

3.4.4 Alternative quality measures

Lines and Bagnall [116] propose two quality measures as alternatives to using Information Gain for the shapelet-tree classifier. They show that their measures offer equivalent accuracy and increased speed. More measures are tested in [88]; the F-stat measure is found to be faster to calculate and more accurate than the other shapelet quality measures.

3.4.5 Shapelet-like approaches

The following papers present work that is intuitively similar to the shapelet approach, though not close enough to be considered shapelet research.

In [113], the authors apply a shapelet-like approach to the problem of detecting friendship relationships in time-series GPS data. For this problem, local similarity is more important than global similarity; proximity on a weekday is less predictive than proximity on a Saturday. The position in the time series is the most important factor. Hence, instead of searching for local shapes that are discriminative, they search for the most discriminative time interval, almost the converse of the shapelet approach.

Di Fatta *et al.* [49] aim to detect faults in software by mining tree structures that represent successful or failed executions. Their methodology is similar to the shapelet approach, but uses a different representation. Rather than search a set of $1D$ series for the most discriminative subsequence, they search a set of trees for the most discriminative sub trees. Their method is interpretable: the sub tree that best discriminates failed executions is likely to contain sub routines that cause failures.

In [102], Ko *et al.* use time-series classification for *context recognition*, where data from multiple sensors are fused into a representation of a situation. Unlike time-series classification with shapelets, they use full series, and employ DTW distance to accommodate noise in indexing. Class prototypes are used in the same way as shapelets; new examples are classified by assigning the class of the closest prototype.

Hartman *et al.* [82] apply a shapelet-like approach to gesture recognition. There are two main differences between their algorithm and the algorithm in [180]. First, they use DTW, rather than Euclidean distance. Second, they use their own quality measures, rather than Information Gain, to evaluate prototypes, see Section 3.3.2. These differences are minor; [82] might be considered part of the shapelet literature; we examine another paper from the same research group [81] in Section 3.3.

3.5 Motifs

In this section, we move away from the supervised task of classification, and focus on applying the shapelet intuition to the unsupervised task of finding approximately repeated patterns in time series. The central insight of the shapelet approach is that, for many problems, local similarity of shape can provide more accurate classification than global similarity of shape. Data mining by extracting local features from time series is not limited to classification. By extracting repeated subsequences from longer time series, we can examine them in ways that may prove more fruitful than examining the entire series. Repeated subsequences may represent significant events (such as heartbeats embedded in an ECG series), or help to distinguish genuine events from noise. Repeated subsequences can be used as primitives for data-mining tasks such as clustering or rule discovery. There are substantial similarities between a shapelet and a repeated time-series subsequence, which we refer to as a *motif*.

A time-series motif is a pattern that approximately recurs in a time series [114]. A *motif set* is the set of subsequences deemed to be instances of a given motif. The problem of finding motifs is analogous to the computational biology problem of finding repeated patterns in discrete DNA sequences [54, 28]. Motifs can be used to characterise the typical behaviour of a time series for classification or anomaly detection (e.g. [114]), or as a primitive in, for example, association rule mining (e.g. [39, 90]) or clustering [56]. Areas of application include medicine [114, 126], image processing [114], information retrieval [107], and robotics [134].

The key contributions to the study of motifs are presented in [114, 131, 132]. The motif-discovery algorithm described in [114] involves compressing the series with a piecewise aggregate approximation transform, then discretising the series into a fixed alphabet size, before finding motifs through matrix approximation and using an approximate distance map. We adopt many of the definitions from [114], but focus our interest on finding exact motifs; we do not discretise the data, instead searching the real-valued series. Hence, we concentrate on the work in [131, 132], discussed in Section 3.5.3. Before discussing algorithms for motif discovery, we examine motif definitions and the problem of trivial matching.

3.5.1 Motif definition

There is variation in the literature over the definition of a motif and the nature of the association between a motif and other subsequences. We use the term *motif* to refer to a single subsequence (which can be a concrete instance, or the average of the members of its motif set), and the term *motif set* to mean the set of subsequences that are associated with a given motif. In [114], the 1-motif is defined as the subsequence with the largest count of non-trivial matches, and the K-motif is the subsequence with the K^{th} largest count of non-trivial matches, subject to the constraint that the K-motif is at least $2r$ distance away from the previous $K-1$ motifs, and where two series match if the distance between them is less than some threshold parameter, r, and the match is *non-trivial.* In [131, 132], the nearest equivalent definition to the motif set is the *range motif.* The range motif with range r is the maximal set of time series that have the property that the maximum distance between them is less than $2r$. The actual motif for the range motif is not, in fact, a subsequence of the original series. Rather, it is the average of all members of the motif set. In clustering parlance, the motif is the centroid of the motif set.

3.5.2 Trivial matches

A key aspect to successful motif discovery is avoiding *trivial matches.* Failing to prevent trivial matching renders motif detection meaningless [100]. Informally, trivial matches are those matches where the similarity is the result of the series overlapping, and therefore sharing a common subsequence. Formally, there are alternative definitions of a trivial match. In [114], a match between two subsequences S_a and S_b (where $a < b$) is defined to be a trivial match if every series beginning from $a+1$ to $b-1$ is also a match with S_a. This allows overlapping matches, as long as there is one series in the overlap section that is not a match. Conversely, non-overlapping series may be considered trivial matches. This removes the matching of, for example, long periods of no change in the series. However, it is somewhat counter intuitive, and in [131], a simpler definition is employed. For MK pair matching, two matching series are trivially matched if their starting points are within w places (i.e. $b - a < w$). This approach introduces a new parameter into the algorithm; hence, we adopt the definition used in [100], and take it that two matched series are trivially matched if they overlap, i.e. if $b - a < n$, where n is the length of the series. This is equivalent to the definition used in [131] for the case where $w = n$.

3.5.3 Mueen-Keogh (MK) best-matching pair algorithm

MK finds exact matches between time-series subsequences using a form of early abandon that dramatically speeds up the matching process in the average case. A formal description of MK is given in [131]. We restrict ourselves to an overview of the algorithm. The core lower-bounding routine is as follows:

1. A random subsequence is selected.
2. The distance between this reference subsequence and all other subsequences is calculated, and the subsequences are ordered by this distance to give a list $< S_1, S_2, \ldots, S_{m-(n-1)} >$. The best-so-far distance d is set to $d(S_1, S_2)$.
3. When forming the sorted list, the distance between adjacent pairs is calculated and stored. This key step allows MK to exploit the triangle inequality; the relative distance between two points in relation to the reference point is a lower bound on the true distance between them.
4. A linear scan across the list is conducted, and the true distance between adjacent points $d(S_i, S_{i+1})$ for $i = 2$ to $(p-1)$ is calculated, the lower bounds are updated and the best-so-far adjusted if necessary (ignoring trivial matches).
5. The enumerative search can then proceed, but distance calculations are avoided for all proposed pairs where the lower bound is worse than the current best-so-far.

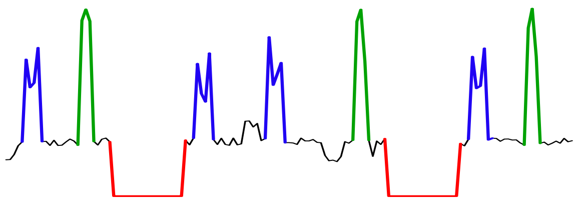
Figure 3.6: A simulation of an electricity demand profile, with three motif sets. The red pattern represents a period of no usage, the blue pattern approximates the usage pattern of a dishwasher, and the green pattern approximates the usage pattern of an immersion heater.

The algorithm is further enhanced by the selection of q multiple reference sequences. The distance from each of these selected subsequences and the rest of the subsequences is calculated, and the reference subsequences are sorted by their distance standard deviation. When the enumerative search begins, the lower bound can be checked against each reference sequence.

3.5.4 Algorithms for discovering motif sets

In [114], the 1-motif is found through enumerating the search space. Finding the K-motifs requires that we enforce a separation of at least $2r$ between each motif and the previous $K-1$ motifs. The obvious solution is to remove matches to each motif found before continuing the search; in Chapter 7 we describe our approach to this problem.

MK finds closest matches between time-series subsequences using a form of early abandon that dramatically speeds up the matching process in the average case. Finding best-matching pairs is of less interest to us than detecting repeated patterns. This is an important point because the members of the best-matching pair are not necessarily members of a high-cardinality motif set; for example, in Fig. 3.6, the red subsequences form the best-matching pair, but the blue and green motif sets have higher cardinality.

The method proposed in [132] for finding the the range motif (broadly equivalent to our motif set), involves finding the best-matching pair with MK, then performing a linear scan of the series to find all subsequences within $2r$ of the pair. This set is then *"trivially condensed"* to find the range motif.

In Chapter 7, we propose a novel extension of this process, generalised for finding the K-motif sets, called the Scan MK algorithm.

3.6 Conclusions

Shapelets are time-series subsequences that represent the class of the series. They have been used for classification in a range of domains, with promising results. The shapelet search is very slow; speed-up techniques based on early abandon, re-using information, discretisation, and greedy search have been used to make the search tenable.

Most uses of shapelets have focused on embedding the shapelet search within a decision tree, which requires a shapelet search at each node, increasing the time complexity, reduces interpretability because of the hierarchical nature of decision trees, and yields relatively poor classification accuracy compared to other classifiers. In Chapter 5, we extend the shapelet approach, making several novel contributions to the field, including our shapelet transform, which dissociates the shapelet search from the decision tree, allowing more accurate classifiers to be used and replacing multiple calls to the shapelet discovery algorithm with a single pass through the data. Our contribution to research into shapelets is in Chapter 5.

Data mining with local-shape-based similarity is not limited to classification with shapelets. The unsupervised equivalent of a shapelet is a motif. The interesting problem in motif discovery is to find sets of approximately repeated patterns within a time series, while avoiding trivial matches (matches that overlap) and enforcing a distance-based separation between motif sets. We contribute to research into motifs in Chapter 7.

Chapter 4

Data

4.1 Introduction

In this chapter, we discuss the datasets we use for the experiments in Chapters 5, 6, and 7. We have attempted to experiment with every commonly used TSC dataset, and have contributed a number of new datasets that our local-shape based approach is suitable for.

4.2 Summary of datasets

4.2.1 Classification problems

We experiment on 75 time-series datasets. We use every dataset from the UCR Time-series Classification/Clustering page [165] except the broken ECG200 dataset (see [7]), and also contribute a number of new datasets. The datasets we use, and the relevant parameters associated with them, are shown in Tables 4.1 - 4.5.

The datasets are partitioned into training and testing sets. Shapelet selection, model selection, and classifier training are performed exclusively on the training set; the test set is used only with the final trained classifier. We report the accuracy on the test set. The datasets we use that are not included on the UCR page can be found at [85].

Table 4.1: Summary of image-outline datasets.

Dataset	Orig. length	Size train	Size test	Number of Classes
Adiac	176	390	391	37
ArrowHead	251	36	175	3
BeetleFly	512	20	20	2
BirdChicken	512	20	20	2
DiatomSizeReduction	345	16	306	4
DistalPhalanxOutlineAgeGroup	80	400	139	3
DistalPhalanxOutlineCorrect	80	600	276	2
DistalPhalanxTW	80	400	139	6
FaceAll	131	560	1690	14
FaceFour	350	24	88	4
FacesUCR	131	200	2050	14
fiftywords	270	450	455	50
fish	463	175	175	7
Herrings	512	64	64	2
MedicalImages	99	381	760	10
MiddlePhalanxOutlineAgeGroup	80	400	154	3
MiddlePhalanxOutlineCorrect	80	600	291	2
MiddlePhalanxTW	80	399	154	6
OSULeaf	427	200	242	6
PhalangesOutlinesCorrect	80	1800	858	2
ProximalPhalanxOutlineAgeGroup	80	400	205	3
ProximalPhalanxOutlineCorrect	80	600	291	2
ProximalPhalanxTW	80	400	205	6
SwedishLeaf	128	500	625	15
Symbols	398	25	995	6
WordSynonyms	270	267	638	25
yoga	426	300	3000	2

We have divided the datasets into five different problem types:

1. Image-outline problems (Table 4.1).

2. Motion classification problems (Table 4.2).

3. Sensor reading classification problems (Table 4.3).

4. Human sensor reading classification problems (Table 4.4).

5. Simulated classification problems (Table 4.5)

Each of these problem types should lend itself to a greater or lesser degree to classification by local similarity in shape. In Chapter 5, we break down our findings by problem type to gain additional insight into the applicability of our method.

We do not use every dataset for every experiment. We will indicate in the appropriate sections where we have used a subset of the 75 datasets. Our motivation for

Table 4.2: Summary of motion-classification datasets.

Dataset	Orig. length	Size train	Size test	Number of Classes
Cricket_X	300	390	390	12
Cricket_Y	300	390	390	12
Cricket_Z	300	390	390	12
GunPoint	150	50	150	2
Haptics	1092	155	308	5
InlineSkate	1882	100	550	7
ToeSegmentation1	277	40	228	2
ToeSegmentation2	343	36	130	2
UWaveGestureLibrary_X	315	896	3582	8
UWaveGestureLibrary_Y	315	896	3582	8
UWaveGestureLibrary_Z	315	896	3582	8
Worms	900	181	77	5
WormsTwoClass	900	181	77	2

Table 4.3: Summary of sensor-reading datasets.

Dataset	Orig. length	Size train	Size test	Number of Classes
Beef	470	30	30	5
Car	577	60	60	4
ChlorineConcentration	166	467	3840	3
Coffee	286	28	28	2
Computers	720	250	250	2
Earthquakes	512	322	139	2
FordA	500	3601	1320	2
FordB	500	3636	810	2
ItalyPowerDemand	24	67	1029	2
LargeKitchenAppliances	720	375	375	3
Lightning2	637	60	61	2
Lightning7	319	70	73	7
MoteStrain	84	20	1252	2
OliveOil	570	30	30	4
Plane	144	105	105	7
PtNDeviceGroups	720	1750	1750	5
PtNDevices	720	1750	1750	14
RefrigerationDevices	720	375	375	3
ScreenType	720	375	375	3
SmallKitchenAppliances	720	375	375	3
SonyAIBORobotSurface	70	20	601	2
SonyAIBORobotSurfaceII	65	27	953	2
StarLightCurves	1024	1000	8236	3
Trace	275	100	100	4
wafer	152	1000	6164	2

Table 4.4: Summary of human sensor-reading datasets.

Dataset	Orig. length	Size train	Size test	Number of Classes
CinC_ECG_torso	1639	40	1380	4
ECGFiveDays	136	23	861	2
NonInvasiveFatalECG_Thorax1	750	1800	1965	42
NonInvasiveFatalECG_Thorax2	750	1800	1965	42
TwoLeadECG	82	23	1139	2

Table 4.5: Summary of simulated datasets.

Dataset	Orig. length	Size train	Size test	Number of Classes
CBF	128	30	900	3
MALLAT	1024	55	2345	8
SimulatedSet	500	100	1000	2
SyntheticControl	60	300	300	6
TwoPatterns	128	1000	4000	4

restricting the experiments to these datasets is usually one of speed: the full shapelet transform with ensemble classifier (see Chapter 5) is slow to transform and train, and we restrict ourselves to 50 tractable datasets, while searching for methods that will allow us to use the larger datasets (for example, dimensionality reduction, see Chapter 5).

4.2.2 Unsupervised data mining

In Chapter 7, we consider the problem of finding approximately repeated patterns in longer time series. For these experiments, we use data described in Sections 4.4.9 and 4.4.10.

4.3 UCR repository

Most of the datasets we experiment upon can be found on the UCR Time-series Classification/Clustering Page [165]. They are standard benchmark time-series datasets used in the literature, and where possible, we compare our methods to rival methods on these datasets. We use the existing train/test splits, and do not clean or modify the data in any way prior to shapelet transformation.

4.4 Contributed datasets

We provide a number of new datasets that we make freely available to researchers. We have selected these datasets to study because we feel they will lend themselves well to data mining using local-shape-based similarity. The datasets are available from [85].

4.4.1 Classification problems

We provide 13 new image-outline problems: ten bone-outline classification problems, (Section 4.4.3), two image-processing outline-classification problems derived from the MPEG-7 dataset [21] (Section 4.4.5), and an outline-classification problem involving classifying herring based on their otoliths (Section 4.4.6). We also provide seven datasets derived from sensor readings used for electrical device profiling, derived from the Powering the Nation project (Section 4.4.7), and two motion classification problems from behavioural genetics involving classifying worm movements (Section 4.4.8). Finally, we provide a novel simulated dataset designed to be optimal for the shapelet approach (Section 4.4.4).

4.4.2 Data for unsupervised data mining

We provide two novel datasets for unsupervised mining of approximately repeated patterns: a synthetic problem designed to test the efficacy of pattern-discovery algorithms (Section 4.4.9), and an electrical device disambiguation problem derived from smart-metering systems (Section 4.4.10).

4.4.3 Bone outlines

The bone datasets (DistalPhalanxOutlineAgeGroup, DistalPhalanxOutlineCorrect, DistalPhalanxTW, MiddlePhalanxOutlineAgeGroup, MiddlePhalanxOutlineCorrect, MiddlePhalanxTW, PhalangesOutlinesCorrect, ProximalPhalanxOutlineAgeGroup, ProximalPhalanxOutlineCorrect, and ProximalPhalanxTW) consist of image outlines from hand x-rays, where the three bones from the middle finger have been converted into $1D$ series by measuring the distance to each point on the outline from the centre point of the phalanx/epiphysis (Figure 4.1). The original images can be found at [96].

The distal, middle, and proximal phalanges are three different bones of the finger (Fig. 4.1 Left). There are three classification problems for each finger bone, forming nine datasets, and one more dataset formed from the full set of bones.

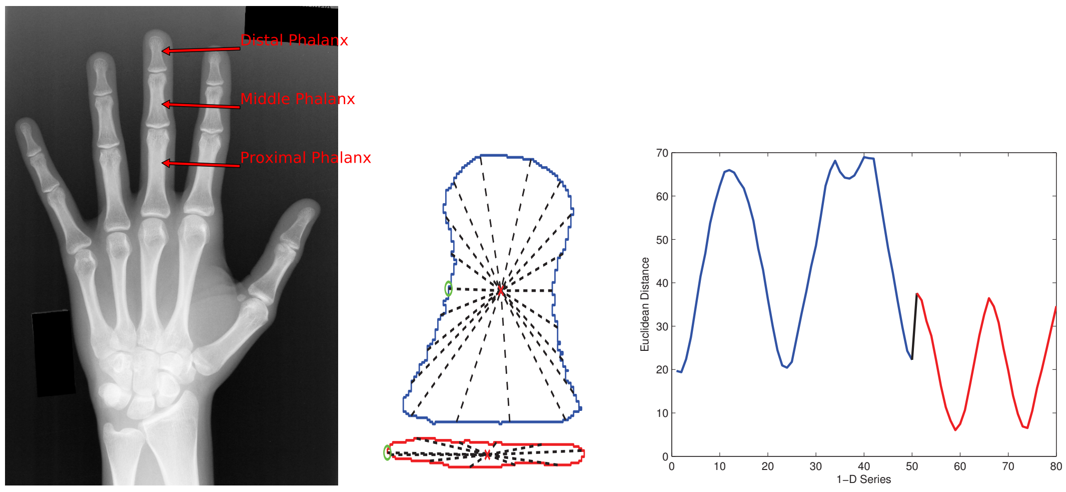

Figure 4.1: Left: a hand x-ray from [96], annotated to show the three bones of interest. Middle: a bone outline with mapping. Right: bone outline rendered as $1D$ series. Middle and right images from [6].

The 'Correct' problems are two class, corresponding to the following: 0 - bad segmentation, 1 - good segmentation (Figure 4.2). A hand x-ray has been segmented well if the different bones have been picked out correctly.

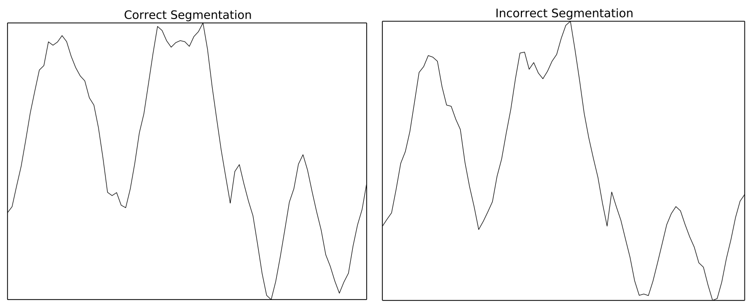

Figure 4.2: Left: an example of correct bone outline segmentation from the DistalPhalanxOutlineCorrect dataset. Right: an example of incorrect bone outline segmentation from the DistalPhalanxOutlineCorrect dataset.

The 'Age Group' problems are three class, corresponding to the following groups:

Table 4.6: Bone-outline dataset class labels.

Problem	Class labels
DistalPhalanxOutlineAgeGroup	0-6 (1), 7-12 (2), 13-19 (3)
DistalPhalanxOutlineCorrect	Bad segmentation (0), Good segmentation (1)
DistalPhalanxTW	3, 4, 5, 6, 7, 8
MiddlePhalanxOutlineAgeGroup	0-6 (1), 7-12 (2), 13-19 (3)
MiddlePhalanxOutlineCorrect	Bad segmentation (0), Good segmentation (1)
MiddlePhalanxTW	3, 4, 5, 6, 7, 8
ProximalPhalanxOutlineAgeGroup	0-6 (1), 7-12 (2), 13-19 (3)
ProximalPhalanxOutlineCorrect	Bad segmentation (0), Good segmentation (1)
ProximalPhalanxTW	3, 4, 5, 6, 7, 8
PhalangesOutlineCorrect	Bad segmentation (0), Good segmentation (1)

0-6 years old, 7-12 years old, and 13-19 years old (Figure 4.3). The task is to predict the age group of the patient at the time of the x-ray given just the bone outline.

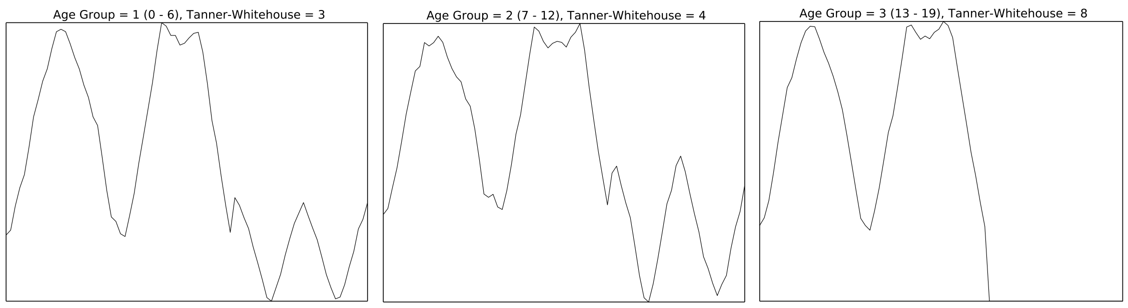

Figure 4.3: Example $1D$ bone image outlines from the DistalPhalanxOutlineAgeGroup and DistalPhalanxTW datasets. From left to right: Age group 0 - 6, TW 3; Age group 7 - 12, TW 4; Age group 13 - 19, TW 8.

The 'TW' problems are eight class, corresponding to the *Tanner-Whitehouse* bone age classification system (Figure 4.3).

The PhalangesOutlineCorrect problem is a much larger (1800 instances) two-class problem; the task is to classify whether the bones have been segmented correctly.

A list of the class labels for each problem is given in Table 4.6. For more information, see [41, 42, 40].

4.4.4 Synthetic data

Our synthetic dataset is intended to be optimal for classification using shapelets. The SimulatedSet dataset consists of 100 training series and 1000 test series. The series are divided into two classes. Series of class 0 are designated *Spike* series. Series of class 1 are designated *Triangle* series. The training and tests sets are evenly split into Spike series and Triangle series.

The basis of each series is normally-distributed ($\mathcal{N}(0,1)$) random noise of length 500, to which we add one of two shapes (see Fig. 4.4). For each Spike series, we select a random location in the range [0,471] and add a spike shape to the white noise. For each Triangle series, we select a random location in the range [0,471] and add a triangle shape to the white noise. The shapes are of length 29 (a value selected to be small enough that the shape does not dominate the series), with an underlying amplitude of 4, and a minimum value of -2.

We define a triangle shape as an ordered set of 29 real values, Triangle = <-2, -1.714, -1.429, -1.143, -0.857, -0.571, -0.286, 0, 0.286, 0.571, 0.857, 1.143, 1.429, 1.714, 2, 1.714, 1.429, 1.143, 0.857, 0.571, 0.286, 0, -0.286, -0.571, -0.857, -1.143, -1.429, -1.714, -2>. The triangle shape begins at point -2, increases in equal increments to a value of 2 at the central point, then decreases in equal intervals to -2 at the end point.

We define a spike shape as an ordered set of 29 real values, Spike = <0, -0.286, -0.571, -0.857, -1.143, -1.429, -1.714, -2, -1.714, -1.429, -1.143, -0.857, -0.571, -0.286, 0, 0.286, 0.571, 0.857, 1.143, 1.429, 1.714, 2, 1.714, 1.429, 1.143, 0.857, 0.571, 0.286, 0>. The spike shape begins at 0, decreases in equal increments to -2 before increasing in equal increments to 0 at the central index. It then increases in equal increments to 2, before decreasing in equal increments back to 0.

The SimulatedSet dataset is difficult to classify because the discriminating features are small relative to the lengths of the series, the features are very noisy (see Fig. 4.4), and the features occur at random places in the series. Such data should show great improvements in accuracy over standard time-domain classifiers when classified using

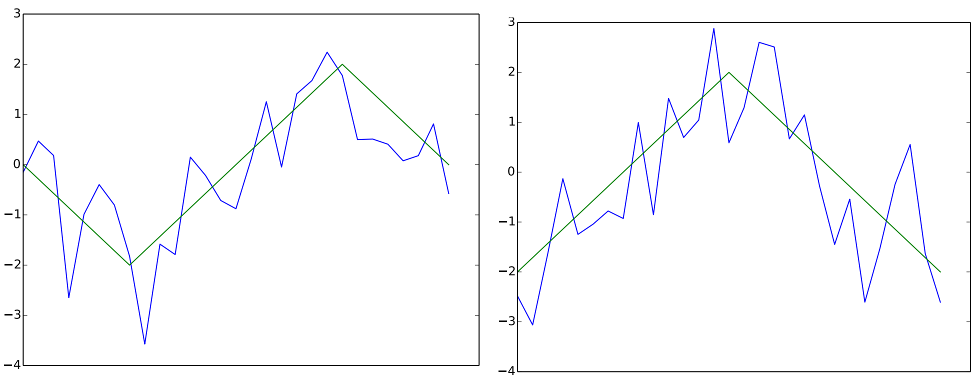

Figure 4.4: Left: the spike shape in green, and an example of the spike shape as it would appear in a series of the SimulatedSet dataset, in blue. Right: the triangle shape in green, and an example of the triangle shape as it would appear in a series of the SimulatedSet dataset, in blue.

shapelets.

4.4.5 MPEG-7 shapes

MPEG-7 CE Shape-1 Part B [21] is a database of binary images developed for testing MPEG-7 shape descriptors, and is available free online. It is used for testing contour/image and skeleton-based descriptors [108]. Classes of images vary broadly, and include classes that are similar in shape to one another. There are 20 instances of each class, and 60 classes in total. We have extracted the outlines of these images and mapped them into $1D$ series. We have created two time-series classification problems from the series, *Beetle/Fly* and *Bird/Chicken*. Figure 4.5 shows some of the images from the two problems. Figure 4.6 shows a beetle image rendered as a $1D$ series.

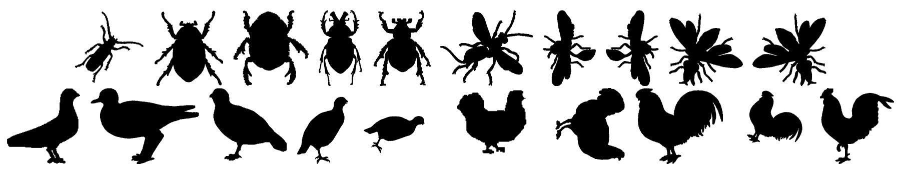

Figure 4.5: Five beetle images (top left) and five fly images (top right) from the Beetle/Fly problem. Five bird images (bottom left) and five chicken images (bottom right) from the Bird/Chicken problem. There is considerable intra-class variation, as well as inconsistent size and rotation.

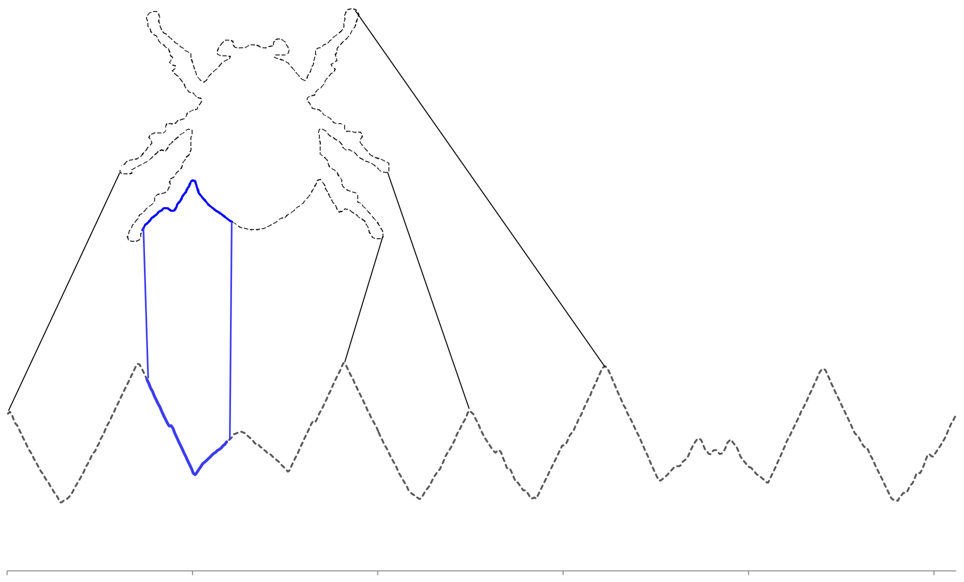

Figure 4.6: A beetle image outline rendered as a $1D$ series.

4.4.6 Otoliths

Otoliths are calcium carbonate structures present in many vertebrates, found within the sacculus of the *pars inferior*. Otoliths vary markedly in shape and size between species, but are of similar shape to other stocks of the same species (Figure 4.7). Otoliths contain information that can be used by ‘expert readers’ to determine several key factors important in managing fish stock. Analysis of otolith boundaries may allow estimation of stock composition, including whether the samples are from one stock or multiple stocks [53, 31, 47], allowing management decisions to be made [159]. We consider the problem of classifying herring stock (either North Sea or Thames) based on a $1D$ series derived from the image outline (Figure 4.8).

4.4.7 Powering the Nation

The seven PtN datasets (Computers, LargeKitchenAppliances, PtNDeviceGroups, PtNDevices, RefrigerationDevices, ScreenType, and SmallKitchenAppliances) are electricity consumption classification problems. The datasets are derived from domestic

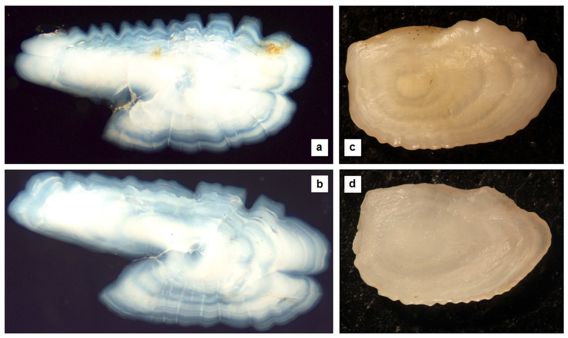

Figure 4.7: Otoliths from North-Sea Herring (a), Thames Herring (b) and two distinct populations of Plaice (c and d).

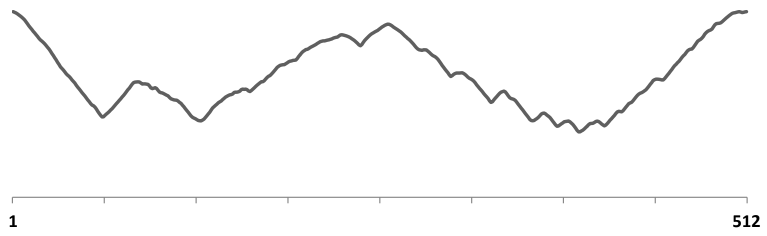

Figure 4.8: A $1D$ time-series representation of the herring otolith shown in Fig. 4.7.

electricity consumption within the United Kingdom, as part of government sponsored study [137] to help reduce the UK's carbon footprint.

The data were created by smart meters as part of an £11.1 billion project to reduce consumer energy consumption by alerting people to their real-time spending [137]. The original data are readings from 251 households, sampled in two-minute intervals over a month.

The classification problem for each PtN dataset in Table 4.7 is to classify each series as an example of the use of a particular device (see Figure 4.9). For the five datasets listed in Table 4.7, there are two distinct types of problem: problems with similar usage patterns (RefrigerationDevices, Computers, and ScreenType) and problems with dissimilar usage patterns (SmallKitchenAppliances and LargeKitchenAppliances).

Electricity consumption patterns, such as those shown in Figure 4.9, can appear

Table 4.7: PtN dataset class labels.

Problem	Class labels
Computers	Desktop, laptop
Large Kitchen Appliances	Dishwasher, tumble dryer, washing machine
Refrigeration	Fridge/freezer, refrigerator, upright freezer
Screen	CRT TV, LCD TV, computer monitor
Small Kitchen Appliances	Kettle, microwave, toaster

at any point in a single series of that class, and can appear at multiple points. They are not identical across households, and are unlikely to be identical across different uses of the same device (for example, the power used by a kettle will vary depending on how recently it has been boiled). Further, they are short compared to the length of a series.

The device types for the five datasets are shown in Table 4.7. The two remaining datasets, PtNDevices and PtNDeviceGroups, contain each series from the the other five datasets. The classification problem for PtNDevices is to classify by the class labels given in Table 4.7; the classification problem for PtNDeviceGroups is to classify each example by the dataset it is drawn from.

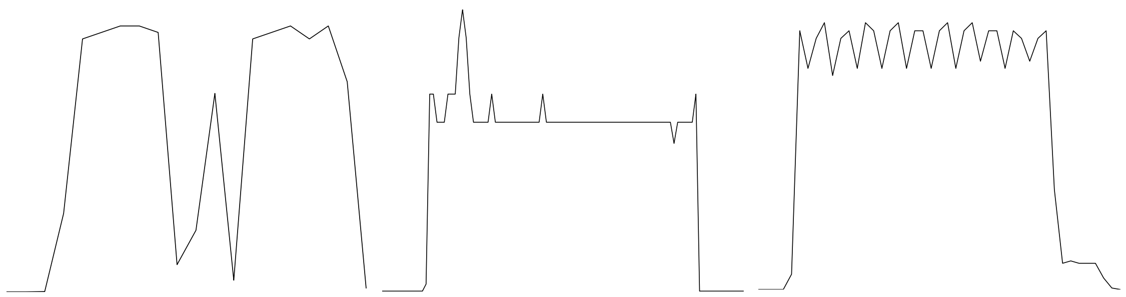

Figure 4.9: From left to right, electricity consumption time series for a dishwasher, monitor, and tumble drier.

4.4.8 Classifying mutant worms

The two worm datasets (Worms and WormsTwoClass) are classification problems based on data from behavioural genetics. *Caenorhabditis elegans* is a roundworm used as a model organism in behavioural genetics. Brown *et al.* [27] describe a system for measuring the motion of worms on an agar plate in terms of a range of human-defined features [182]. The space of shapes *Caenorhabditis elegans* adopts can be represented by combinations of four base shapes, or *eigenworms*. Once the worm outline is extracted, each frame of worm motion can be captured by four scalars representing the amplitudes along each dimension when the shape is projected onto the four eigenworms (see Figure 4.10).

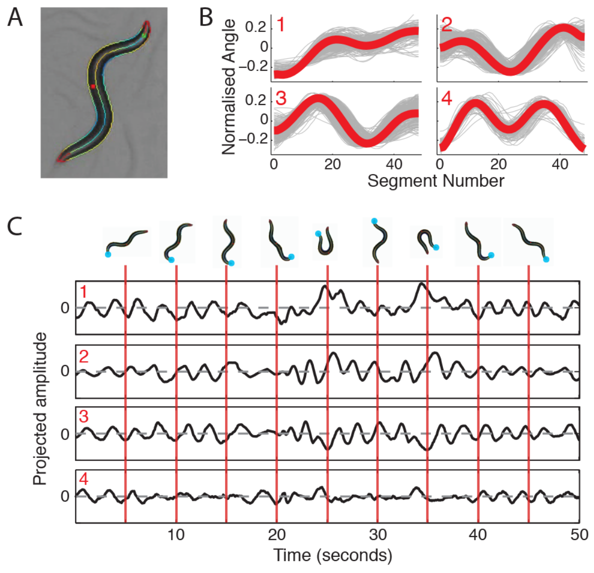

Figure 4.10: (A) a worm on an agar plate. (B) four representative eigenworms. (C) example time series. Images from [27].

Using data collected by Brown *et al.* [27] and extracted from the *C. elegans* behavioural database [1], we create two classification problems by generating $1D$ series based on the first eigenworm, down-sampled to second-long intervals. Each worm is labelled as either wild-type (the N2 reference strain - 109 cases) or one of four mutant types: goa-1 (44 cases), unc-1 (35 cases), unc-38 (45 cases), and unc-63 (25 cases).

The classification problem for the Worms dataset is to classify each series as coming from a wild-type worm or a specific type of mutant. The problem for the WormsTwoClass dataset is to classify each series as coming from a wild-type or mutant (any type) worm.

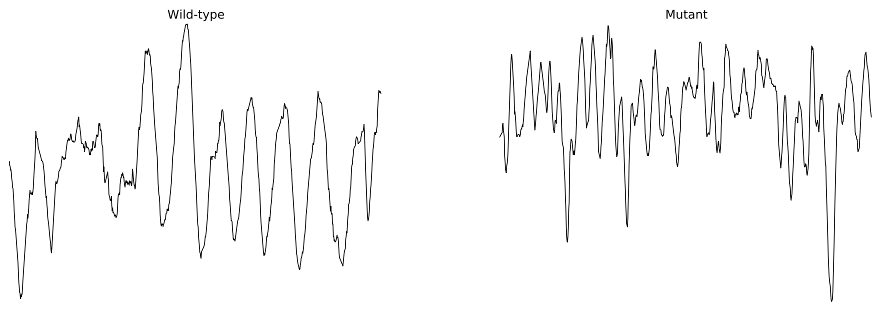

Figure 4.11: Left: a time series representing the movement of a wild-type worm. Right: a time series representing the movement of a mutant worm.

Figure 4.11 shows one time series derived from the movements of a wild-type worm (left) and one time series derived from a mutant worm (right).

4.4.9 Synthetic data space for unsupervised data mining

For the synthetic data, we specify a parameterised data space from which datasets are drawn, and randomly generate independent datasets for a given set of parameters. The simulated data is white noise (observations of i.i.d. normally-distributed random variables with $\mu = 0$ and $\sigma = 1$) with shapes added to the noise at random intervals. Figure 4.12 shows the five shapes used to create the datasets.

The minimum and maximum time series length, number of distinct shapes, and instances of each shape, are fixed parameters of the data, as are the length and amplitude of each instance. To generate a dataset, we randomly select one or two different shapes, and a number of instances for each shape. The shapes are added to the white noise at random locations, and do not overlap. An example of this distorted motif data is shown in Figure 4.13.

The dataset SimulatedSet (Section 4.4.4) contains instances drawn from this parameterised data space.

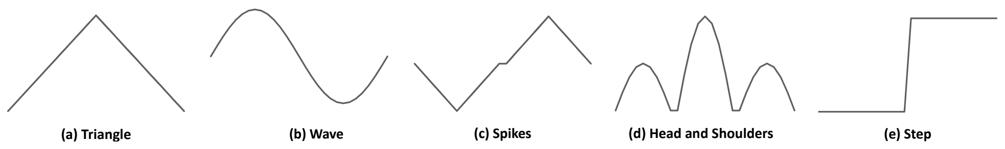

Figure 4.12: The five shapes inserted into our synthetic datasets. They are: (a) *Triangle*, (b) *Wave*, (c) *Spikes*, (d) *Head and Shoulders*, (e) *Step*. The shapes are shown here undistorted; in the synthetic data, they have added noise.

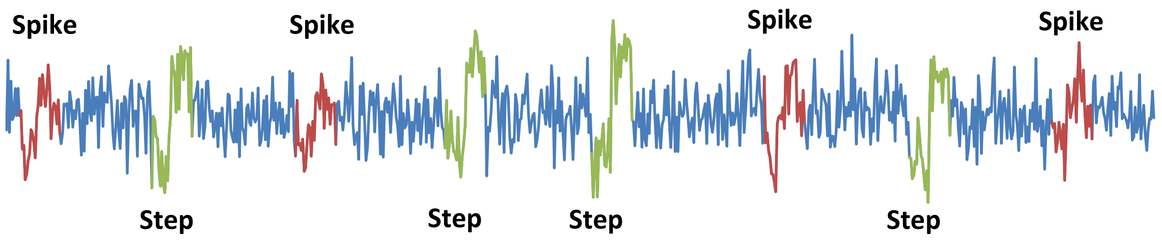

Figure 4.13: An example simulated motif problem, with two motif sets, a Spike set (highlighted in red), and a Step set (highlighted in green).

4.4.10 Electricity-usage data for unsupervised data mining

The electrical device data originates from a trial of smart meters in 187 homes across the United Kingdom (see [118, 7]). Smart meters were used to monitor the electricity consumption of each household in Watt Hours (Wh) at 15-minute intervals. Each series corresponds to the entire consumption of a household over the duration of the trial. The UK government has mandated that all households must be equipped with smart metering equipment by 2020. As a consequence, there will be very large quantities of data that must be processed in an efficient manner. Figure 4.14 shows the typical consumption of a number of device types.

One confounding factor is that devices of a similar nature have very similar usage profiles. Devices such as fridges and freezers, or computers and televisions, are very difficult to distinguish. In addition, the device-specific data is user-orientated. There is no central control over the devices that are monitored; the consumers have direct access to the monitoring equipment and all device labels are user-specified. Hence, labelling is potentially unreliable. Because of these confounding factors, it would be

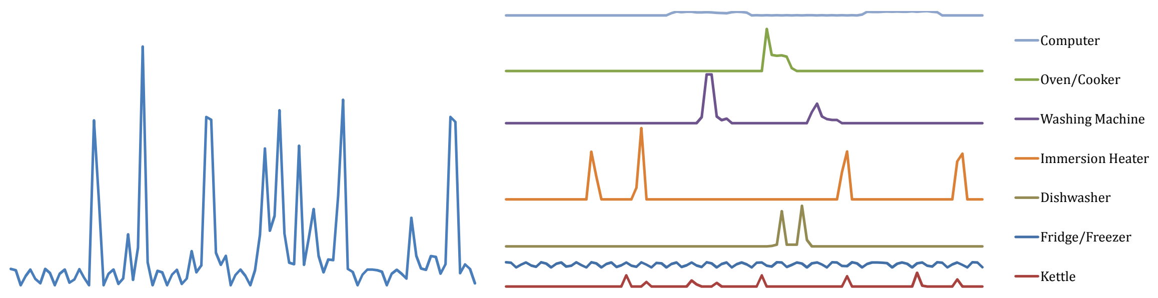

Figure 4.14: Example electricity-usage data for a single house over one day. The graph on the left shows household consumption; the graph on the right is decomposed by device.

beneficial to have a reliable, automated method of detecting and identifying specific device use. The algorithms we propose in Chapter 7 are a first step in this direction.

4.5 Conclusions

We have discussed 75 classification datasets that will be used for experimentation in Chapters 5 and 6, and a synthetic data space and electricity-usage profiles that will be used for unsupervised mining of approximately repeated patterns in Chapter 7.

We begin our experiments in the next chapter.

Chapter 5

Time-series Classification using Shapelet-transformed Data

The work in this chapter is published in a number of papers.

Results from the preliminary implementation of the shapelet transform were published in:

J. Lines, L. Davis, J. Hills, and A. Bagnall

A Shapelet Transform for Time Series Classification

Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 289–297, ACM: 2012.

The results from this paper are not in included in this thesis, as they were created using a legacy version of the shapelet transform implemented by the lead author of [119].

The accuracy results in this thesis for the shapelet transform with $10N$ shapelets are to be published in:

A. Bagnall, J. Lines, J. Hills, and A. Bostrom

Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles

`https://ueaeprints.uea.ac.uk/id/eprint/49614`

This paper is currently under review; all shapelet-related work in the paper is my own.

Some of the work on extensions to the shapelet transform, and some of the results and analysis, is published in:

J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall

Classification of Time Series by Shapelet Transformation

Data Mining and Knowledge Discovery 28 (4), pages 851–881, Springer: 2014.

As lead author, I made the largest contribution to this paper, modifying the original implementation of the shapelet transform, running experiments, extending the transform, analysing results, and writing the paper.

5.1 Introduction

Shapelets are time-series subsequences used for classification. They are intended to provide accurate classification and insight into the problem domain. Our novel contribution to the field is the *shapelet transform*. The shapelet transform (Section 5.2) improves upon both the accuracy and insight aspects of the shapelet approach.

We improve classification accuracy by dissociating the shapelet-discovery algorithm from the classification algorithm; we use the discovered shapelets to transform the original data into a space of shapelet features. Rather than being anchored to a decision tree (the standard format, see Chapter 3), we employ a diverse ensemble of classifiers on the shapelet-transformed data. Our accuracy results (Section 5.3) are significantly better than any other shapelet-based approach, and significantly better than using 1NN with DTW distance, which is a benchmark for time-series classification [51].

Shapelets can offer considerable insight into the problem domain (see Section 5.4). We aim to optimise the shapelet approach for providing insight along with classification accuracy.

We improve upon the ability of shapelets to offer insight into the problem domain in a number of ways. First, we eliminate the tree classifier, and its difficult to interpret hierarchy of binary splits on shapelets. Second, we focus on increasing interpretability by reducing dimensionality through filtering (Section 5.5) and clustering (Section 5.6) the shapelets. We compare a number of different hierarchical clustering methods to a novel form of clustering based on using the *Minimum Description Length* measure as a parameterless stopping criterion for the clustering. Our third contribution is to discretise the clustered shapelet data into a set of binary features representing the presence or absence of a particular shapelet in a given series. Combined with a dictionary of shapelets, this approach is entirely interpretable, and could be deployed in environments such as medicine or finance, where professionals must be able to justify the decisions they make to their customers and stakeholders (we explore this issue in more detail in Chapter 6).

The shapelet transform improves accuracy and interpretability over rival TSC algorithms, offering a solution to TSC problems that is both effective and comprehensible to non-experts.

5.2 Shapelet transform

The shapelet transform finds the best k shapelets in a dataset, then transforms each series into a set of features that represent the distances between the shapelets and the series. The intuition behind the shapelet transform is that classification and representation are two different stages of the TSC process, which can be separated to improve classification accuracy. Bagnall *et al.* [7] demonstrate the importance of separating the transformation from the classification algorithm with an ensemble approach, where each member of the ensemble is constructed on a different transform of the original data. They show two things: first, on problems where the discriminatory features are not in the time domain, operating in a different data space produces greater performance improvement than designing a more complex classifier. Second, a basic ensemble on transformed datasets can significantly improve simple classifiers. We apply this intuition to shapelets, and separate the transformation from the classifier.

Our transformation processes shapelets in two distinct stages. First, the algorithm performs a single scan of the data to extract the best k shapelets. k is a cut-off value for the maximum number of shapelets to store, and has no effect on the quality of the individual shapelets that are extracted. Second, a new transformed dataset is created, where each attribute represents a shapelet, and the value of the attribute is the distance between the shapelet and the original series. Transforming the data in this way disassociates finding the shapelets from classification, allowing the transformed dataset to be used in conjunction with any classifier.

5.2.1 Shapelet generation

The process to extract the best k shapelets is defined in Algorithm 7.

Algorithm 7 `shapeletCachedSelection`($\mathbf{T}$, min, max, k)

```
1: $kShapelets \leftarrow \emptyset$
2: for all series $T_i$ in $\mathbf{T}$ do
3:     $shapelets \leftarrow \emptyset$
4:     $\mathbf{W}_i = \texttt{generateCandidates}(T_i, min, max)$
5:     for all $W_{i,l}$ in $\mathbf{W}_i$ do
6:         for all candidates $S$ in $W_{i,l}$ do
7:             $D_S \leftarrow \texttt{findDistances}(S, \mathbf{T})$
8:             $quality \leftarrow \texttt{assessCandidate}(S, D_S)$
9:             $shapelets \leftarrow shapelets \cup < S, quality >$
10:    $shapelets \leftarrow \texttt{sortByQuality}(shapelets)$
11:    $shapelets \leftarrow \texttt{removeSelfSimilar}(shapelets)$
12:    $kShapelets \leftarrow \texttt{merge}(k, kShapelets, shapelets)$
13: return $kShapelets$
```

The algorithm processes data in a manner similar to the original shapelet algorithm [181] (See Algorithm 4; we retain the notation from that chapter). For each series in the dataset, all subsequences of lengths between *min* and *max* are examined. Unlike Algorithm 4, however, where all candidates are assessed and the best is stored, our caching algorithm stores all candidates for a given time series, along with their associated quality measures (line 9). Once all candidates of a series have been assessed, they are sorted by quality, and self-similar shapelets are removed. Self-similar shapelets are taken from the same series and have overlapping indices. We merge the set of non-self-similar shapelets from a series with the current best shapelets and retain the top k, iterating through the data until all series have been processed. We do not store all candidates indefinitely; after processing each series, we retain only those that belong to the best k so far, and discard all other shapelets. Thus, we avoid the large space overhead required to retain all candidates.

When handling self-similarity between candidates, it is necessary to store temporarily and evaluate all candidates from a single series before removing self-similar shapelets. This prevents shapelets being rejected incorrectly. For example, in a given series, candidate A may be added to the k-best-so-far. If candidate B overlaps with A and has higher quality, A will be rejected. If a third candidate of higher quality, C, is identified that is self-similar to B, but not to A, C would replace B, and the deleted A would be a valid candidate for the k-best. We overcome this issue by evaluating all candidates for a given series before deleting those that are self-similar (line 9 in Algorithm 7). Once all candidates for a given series have been assessed, they are sorted into descending order of quality (line 10). The sorted set of candidates can then be assessed for self-similarity in order of quality (line 11), so that the best candidates are always retained, and self-similar candidates are safely removed.

5.2.2 Length parameter approximation

Both the original algorithm and our caching algorithm require two length parameters, min and max. These values define the range of candidate shapelet lengths. Smaller ranges improve speed, but may compromise accuracy if they prevent the most informative subsequences from being considered. To accommodate running the shapelet filter on a range of datasets without any specialised knowledge of the data, we define a simple algorithm for estimating the min and max parameters.

The procedure described in Algorithm 8 [119] randomly selects ten series from dataset $\mathbf{T}$ and uses Algorithm 7 to find the best ten shapelets in this subset of the data. For this search, $min = 3$ and max is set to m, the length of the series in $\mathbf{T}$. The selection and search procedure is repeated ten times in total, yielding a set of 100

Algorithm 8 `estimateMinAndMax`($\mathbf{T}$)

```
1: $shapelets \leftarrow \emptyset$
2: $m \leftarrow T_1.length$
3: for $i \leftarrow$ 1 to 10 do
4:     $\mathbf{T} \leftarrow \texttt{randomiseOrder}(\mathbf{T})$
5:     $\mathbf{T}' \leftarrow \{T_1, T_2, ..., T_{10}\}$
6:     $currentShapelets \leftarrow \texttt{shapeletCachedSelection}(\mathbf{T}', 3, m, 10)$
7:     $shapelets \leftarrow shapelets \cup currentShapelets$
8: $shapelets \leftarrow \texttt{orderByLength}(shapelets)$
9: $min \leftarrow shapelets_{25}.length$
10: $max \leftarrow shapelets_{75}.length$
11: return $min, max$
```

shapelets. The shapelets are sorted by length, with the length of the 25th shapelet returned as *min* and the length of the 75th shapelet returned as *max*. While this does not necessarily result in the optimal parameters, it does provide an automatic approach to approximate *min* and *max* across a number of datasets. Table 5.1 shows the minimum and maximum shapelet lengths we used to create shapelet transforms for all 75 datasets.

5.2.3 F-statistic

Unlike the shapelet tree, our shapelet transform does not require an explicit split point to be found by the quality measure. IG introduces extra time overhead and may not be optimal for multi-class problems, since it is restricted to binary splits. In [88], we investigate alternative shapelet quality measures based on hypothesis tests of differences in distribution of distances between class populations. We look at three alternative ways of quantifying how well the classes can be split by the list of distances D_S: Kruskal-Wallis [103], F-statistic (F-Stat), and Mood's Median [128]. We find that classification on shapelet data transformed using the F-Stat is significantly more accurate on synthetic data, and has the most wins of any measure over 29 datasets, though the difference is not significant. We also find the F-Stat to be faster on average

Table 5.1: Minimum and maximum length parameters for 75 datasets, found using Algorithm 8, and used to create all of our shapelet transforms.

Dataset	Min. Length	Max. length	Dataset	Min. Length	Max. length
Adiac	3	10	MiddlePhalanxOutlineCorrect	5	12
ArrowHead	17	90	MiddlePhalanxTW	7	31
Beef	8	30	MoteStrain	16	31
BeetleFly	30	101	NonInvasiveFatalECG_Thorax1	5	61
BirdChicken	30	101	NonInvasiveFatalECG_Thorax2	12	58
Car	16	57	OliveOil	8	27
CBF	46	90	OSULeaf	141	330
ChlorineConcentration	7	20	PhalangesOutlinesCorrect	5	14
CinC_ECG_torso	697	814	Plane	18	109
Coffee	18	30	ProximalPhalanxOutlineAgeGroup	7	31
Computers	15	267	ProximalPhalanxOutlineCorrect	5	12
Cricket_X	120	255	ProximalPhalanxTW	9	31
Cricket_Y	132	262	PtNDeviceGroups	51	261
Cricket_Z	118	257	PtNDevices	100	310
DiatomSizeReduction	7	16	RefrigerationDevices	13	65
DistalPhalanxOutlineAgeGroup	7	31	ScreenType	11	131
DistalPhalanxOutlineCorrect	6	16	SimulatedSet	25	35
DistalPhalanxTW	17	31	SmallKitchenAppliances	31	443
Earthquakes	24	112	SonyAIBORobotSurface	15	36
ECGFiveDays	24	76	SonyAIBORobotSurfaceII	22	57
FaceAll	70	128	StarLightCurves	68	650
FaceFour	20	120	SwedishLeaf	11	45
FacesUCR	47	128	Symbols	52	155
fiftywords	170	247	SyntheticControl	20	56
fish	22	60	ToeSegmentation1	39	153
FordA	50	298	ToeSegmentation2	100	248
FordB	38	212	Trace	62	232
GunPoint	24	55	TwoLeadECG	7	13
Haptics	21	103	TwoPatterns	20	71
Herrings	30	101	UWaveGestureLibrary_X	113	263
InlineSkate	750	896	UWaveGestureLibrary_Y	122	273
ItalyPowerDemand	7	14	UWaveGestureLibrary_Z	135	238
LargeKitchenAppliances	13	374	wafer	29	152
Lightning2	47	160	WordSynonyms	137	238
Lightning7	20	80	Worms	93	382
MALLAT	52	154	WormsTwoClass	46	377
MedicalImages	9	35	yoga	12	132
MiddlePhalanxOutlineAgeGroup	8	31			

than the other measures, and significantly faster than Kruskal-Wallis or IG. Hence, we use the F-Stat, rather than IG, as our shapelet quality measure.

The F-Stat for analysis of variance is used to test the hypothesis of difference in means between a set of C samples. The null hypothesis is that the population mean from each sample is the same. The test statistic for this hypothesis is the ratio of the variability between the groups to the variability within the groups. The higher the value, the greater the between-group variability compared to the within-group variability. A high-quality shapelet has small distances to members of one class and large distances to members of other classes; hence, a high-quality shapelet yields a high F-stat. To assess a list of distances $D =< d_1, d_2, \ldots, d_n >$, we first split them by class membership, so that D_i contains the distances of the candidate shapelet to time series of class i. The F-Stat shapelet quality measure is:

$$F = \frac{\sum_i (\bar{D}_i - \bar{D})^2/(C-1)}{\sum_{i=1}^{C} \sum_{d_j \in D_i} (d_j - \bar{D}_i)^2/(n-C)}, \tag{5.2.1}$$

where C is the number of classes, n is the number of series, $\bar{D}_i$ is the average of distances to series of class i and $\bar{D}$ is the overall mean of D.

5.2.4 Data transformation

The main motivation for our shapelet transform is to allow shapelets to be used with a diverse range of classification algorithms, rather than the decision tree used in previous research. Our algorithm uses shapelets to transform instances of data into a new feature space; the transformed data can be viewed as a generic classification problem. The transformation process is defined in Algorithm 9.

The transformation is carried out using the subsequence distance calculation described in Chapter 3. A set of k shapelets, $\mathbf{S}$, is generated from the training data. For each instance of data T_i in the full dataset, $\mathbf{T}$, the subsequence distance is computed between T_i and shapelet S. The resulting k distances are used to form a new instance

Algorithm 9 `shapeletTransform`(Shapelets **S**,**T**)

```
1: $\mathbf{T}' \leftarrow \emptyset$
2: for all $T$ in $\mathbf{T}$ do
3:     $T' \leftarrow <>$
4:     for all shapelets $S$ in $\mathbf{S}$ do
5:         $dist \leftarrow \texttt{sDist}(S, T)$
6:         $T' \leftarrow \texttt{append}(T', dist)$
7:     $T' \leftarrow \texttt{append}(T', T.class)$
8:     $\mathbf{T}' \leftarrow \mathbf{T}' \cup T'$
9: return $\mathbf{T}'$
```

of transformed data, where each attribute corresponds to the distance between a shapelet and the original time series. When using data partitioned into training and test sets, the shapelet extraction is carried out on the training data to avoid bias.

5.2.5 Experimental parameters

For all of our experiments, we set the value of k, the maximum number of shapelets to cache, equal to $10N$, where N is the size of the training set. We aim to create as varied a set of shapelets as possible. In many cases, it is not possible to generate the full set of $10N$ shapelets, because the time series are relatively short compared to the shapelet lengths, or compared to the number of series, and self-similar (overlapping) shapelets are discarded. Table 8.4 (see appendix) shows, for each dataset, the maximum number of shapelets searched for, and the number found and used for the shapelet transform.

5.2.6 Classifier performance

In [88], we test the shapelet transform on a small number of datasets. Our findings are summarised in Figure 5.1, which shows the difference in accuracy of seven classifiers trained on the raw data against the same classifiers trained on the shapelet-transformed data. A positive value indicates that the classifier is more accurate on the transformed data, a negative value the converse.

As can be seen, the change in accuracy is strongly indexed to the dataset, as well as to the classifier. Some datasets, for example Adiac, are classified less accurately when transformed into the shapelet space. Others, like Beef and Coffee, are classified more accurately. These differences occur because not all classification problems are best classified using local similarity of shape. Such data might better be classified using models based on autocorrelation or the Fourier transform.

The best way to judge suitability of a dataset for a shapelet-based approach is by calculating a cross-validation accuracy on the training data and comparing it to the cross-validation accuracies of the other approaches (see, for example [141]). If the cross-validation accuracy is much worse in the shapelet space than in the original space, it suggests that using global similarity is a more appropriate way to classify the data. Similarly, if the cross-validation accuracy of a classifier trained on the autocorrelation space is much greater than that of a classifier trained on the shapelet space, it suggests that similarity in change is the appropriate paradigm in which to classify the data.

As well as using cross-validation accuracy, visual inspection of a few instances of different classes of the data can aid in the selection of an appropriate transform. It may be obvious for certain datasets that classification based on local shapes would be ineffective, but transformation into the Fourier space would be beneficial. In most cases, however, visual inspection is unlikely to show definitively which transformation will allow for the most accurate classification. Perhaps the best way to ensure good classification accuracy is to ensemble different transforms, a method explored at length in [7, 9].

There are broad similarities between the accuracies of the different classifiers on each dataset, but there are cases, such as Rotation Forest on the FaceFour dataset, where the change in accuracy is very different to that of the other classifiers. We conclude from this that an ensemble approach could benefit classification accuracy, as the different classifiers vary considerably in their predictions for particular datasets; rather than using a single classifier to asses the performance of the shapelet transform, we use an ensemble of classifiers. We discuss the ensemble in the next section, and in Section 5.3.2, we report experimental results for the ensemble classifier on 75 datasets, comparing it to accuracies reported by other leading approaches.

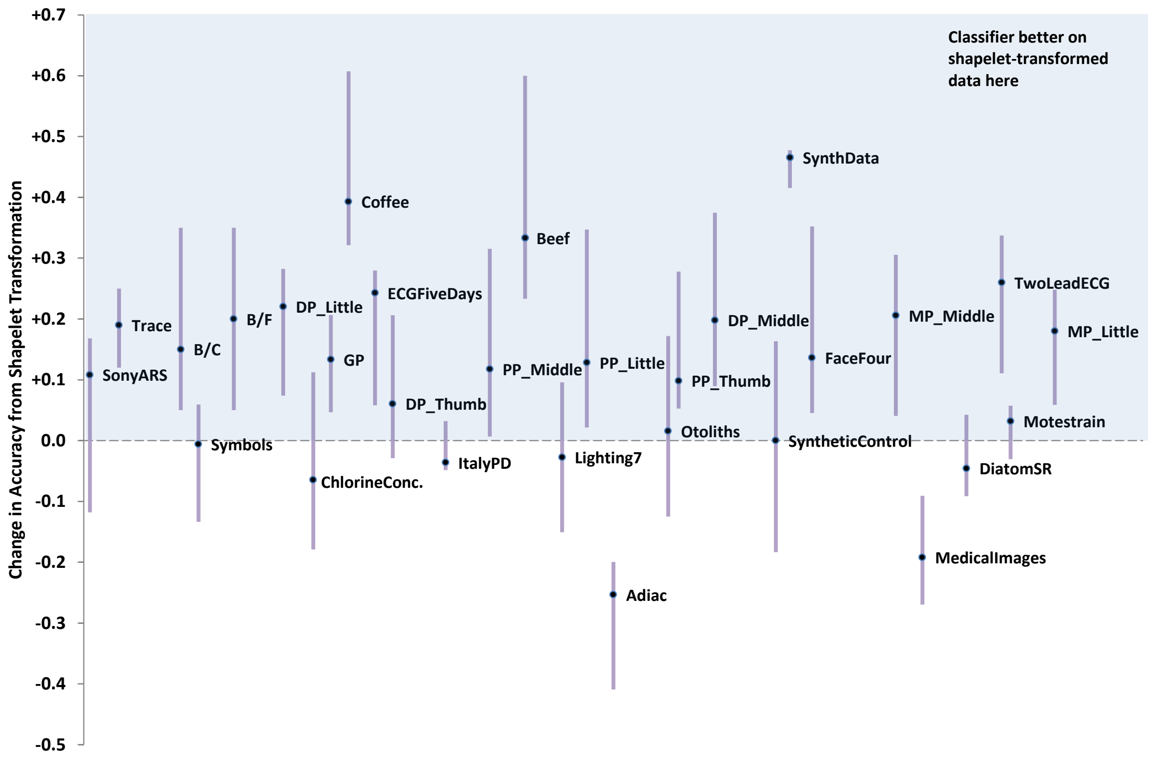

Figure 5.1: Change in classification accuracy between raw data and shapelet-transformed data. Each point is the median difference in accuracy of 7 classifiers; the bar shows the minimum and maximum change in classifier accuracy for that dataset. The figure is taken from [88].

5.2.7 Shapelet ensemble classifier

In [119], we show that the C4.5 decision tree used on the shapelet space has the same accuracy as the original shapelet decision tree from [181]. Several classifiers are significantly stronger than C4.5 on problems transformed into the shapelet space [88]. We can take advantage of the dissociation of transform and classifier to maximise the accuracy of our classification by ensembling a set of classifiers. In general, an ensemble of classifiers will outperform a single classifier [19].

We ensemble eight classifiers: C4.5, kNN with Euclidean distance (k is set using cross-validation, with 50 as the maximum value), Naive Bayes, Bayesian network, Random Forest with 200 trees, Rotation Forest with 10 forests, and two support vector machines, one with a linear kernel, the other with a quadratic kernel. This gives us a variety of different types of classifiers: three tree-based classifiers (C4.5, Random Forest, Rotation Forest), two probabilistic classifiers (Naive Bayes and Bayesian Network), and three instance-based classifiers (kNN and the linear and quadratic support vector machines). Ensembling a variety of different types of classifier maximises the performance of the ensemble [19]. All eight classifiers are implemented in the WEKA machine learning tool kit [78], which makes ensembling them straightforward.

In our ensemble, each classifier is given a weighted vote for each test set instance, based on the classifier's cross-validation accuracy on the training set. Each of the eight classifiers makes a prediction, the weighted votes are summed, and the prediction with the highest sum is returned as the prediction of the ensemble classifier.

In the next sections we report the accuracy of the ensemble classifier on 75 datasets, and compare it to other leading TSC approaches.

5.3 Shapelet transform results

We compare the accuracy of the ensemble classifier to the accuracy of the benchmark TSC algorithm, 1NN with dynamic time warping distance (1NNDTW), over 75 datasets. We also compare our approach to the previous best shapelet-based algorithms.

The previous state-of-the-art shapelet algorithms are the Fast Shapelet algorithm [141] and the Logical Shapelets algorithm [130]. The major weakness of both is that they are limited to an embedded decision-tree classifier (see Chapter 3 for discussion of the shapelet decision tree. In theory, the Fast Shapelets approach could be implemented as a transform; as it stands, it exists only as a tree classifier).

5.3.1 Comparison with 1NNDTW

The 1NNDTW algorithm has proved to be effective at accurately classifying time-series data [51]. Time series are classified significantly more accurately by 1NNDTW than by 1NN with Euclidean distance [117], assuming that the width of the warping window, R, is set using cross validation.

Our first experiment compares the ensemble classifier on shapelet-transformed data to 1NNDTW on the raw data. We set R, the width of the DTW warping window, using cross validation. We compare accuracies over all 75 datasets, and test the differences using a Wilcoxon Signed Rank test at a significance level of 0.01.

The results of our test show that the ensemble classifier on shapelet-transformed data is significantly more accurate than 1NNDTW. The p value is 3.92×10^{-3}. Table 8.1 in the appendix presents the complete set of results. The accuracy results are displayed graphically in Figure 5.2.

Table 5.2 shows the results broken down by problem type. For all problem types but Image, the ensemble classifier on shapelet-transformed data strongly outperforms 1NNDTW. On the Image problems, 1NNDTW marginally outperforms the ensemble, but the difference is not statistically significant. It seems plausible that this is a case of 1NNDTW performing better on Image problems than it does on the other problem types, rather than the ensemble classifier in the shapelet space performing worse. Intuitively, 1NNDTW is well suited to the Image problems we use, as the start and end points of the series are fixed in most cases, and the unrestricted warping window

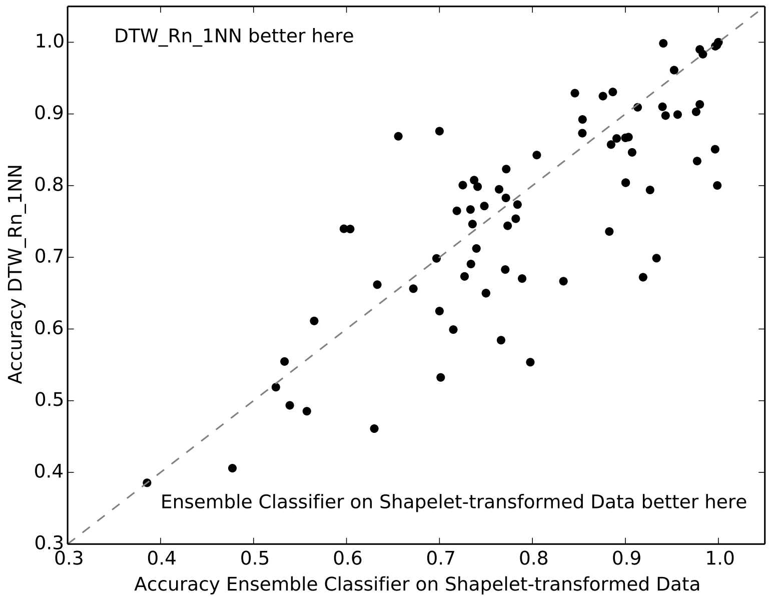

Figure 5.2: Comparison of accuracy over 75 datasets between ensemble classifier on shapelet-transformed data and 1NN with DTW distance, warping window size set by cross validation. The ensemble classifier on shapelet-transformed data is better on 44 datasets, 1NN with DTW is better on 27.

can correct for rotation in the other cases (at least to some extent). The image-outline datasets we use also tend to be less noisy and involve less variation than, for example, the motion-classification datasets. The motion-classification datasets have noise in both indexing and in measurement, and have greater variation between series of the same class. These factors diminish the classification accuracy of the 1NNDTW classifier much more than they affect the accuracy of the ensemble classifier in the shapelet space, as the shapelet space intrinsically corrects for noise in indexing, is more robust to noise in measurement (because the local features are shorter, and therefore have less total noise than the whole series), and is less affected by global within-class variation. Hence, the performance of 1NNDTW on image-classification problems is comparable to that of the ensemble classifier on the shapelet space because those problems have fewer of the factors that diminish the accuracy of the 1NNDTW classifier.

Table 5.2: Counts of wins broken down by problem type for ensemble classifier on shapelet-transformed data and 1NNDTW with warping window size set by cross validation.

Problem type	Wins shapelet	Wins DTW	Total
Image	12	15	27
Motion	8	4	12
Sensor	17	6	23
Human sensor	4	1	5
Simulated	3	1	4
Total	44	27	71

5.3.2 Comparison with Logical Shapelets

The Logical Shapelets algorithm [130] is the current best-performing shapelet-based classifier. It finds exact shapelets and deploys them in a more sophisticated variant of the shapelet tree. We compare the accuracies of the ensemble classifier on shapelet-transformed data to that of the Logical Shapelets algorithm on raw data.

We compare over the 31 datasets used in [141]. We have used every dataset from [141], except for ECG200 and Cricket(Small). Cricket(Small) is not in the UCR Repository [165], and ECG200 is a broken dataset that should no longer be included in such comparisons [7]. The datasets, and the accuracies and ranks, are shown in Table 8.5 (see appendix).

We test the differences in accuracy using a Wilcoxon Signed Rank test at a significance level of 0.01. The test shows that the ensemble classifier is significantly more accurate than the Logical Shapelets algorithm. The p value is 1.875×10^{-6}. Figure 5.3 displays the results graphically.

5.3.3 Comparison with Fast Shapelets

The Fast Shapelets algorithm [141] is a development of the original shapelet algorithm [181] that speeds up the shapelet extraction process by discretising the time

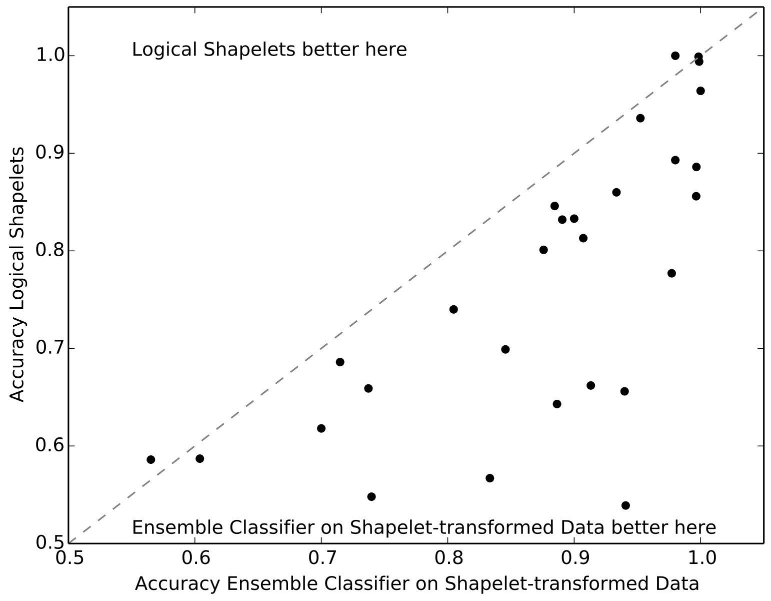

Figure 5.3: Comparison of accuracy over 31 datasets between ensemble classifier on shapelet-transformed data and Logical Shapelets. The ensemble classifier on shapelet-transformed data is better on 28 datasets, Logical Shapelets is better on 3.

series using Symbolic Aggregate Approximation (SAX) [115]. The Fast Shapelets algorithm, in contrast to the Logical Shapelets algorithm, does not find exact shapelets. This results in a large increase in speed, which makes experiments with the Fast Shapelets algorithm more tractable than with Logical Shapelets; hence, accuracy results are available on a wider variety of datasets.

We compare the accuracy of the ensemble classifier on shapelet-transformed data to that of the Fast Shapelets algorithm on raw data over 44 datasets. The datasets, accuracies, and ranks are listed in Table 8.6 (see appendix). We test the differences in accuracy using a Wilcoxon Signed Rank test at a significance level of 0.01. The test shows that the ensemble classifier is significantly more accurate than the Fast Shapelets algorithm. The p value is 4.522×10^{-8}. The results are displayed graphically in Figure 5.4.

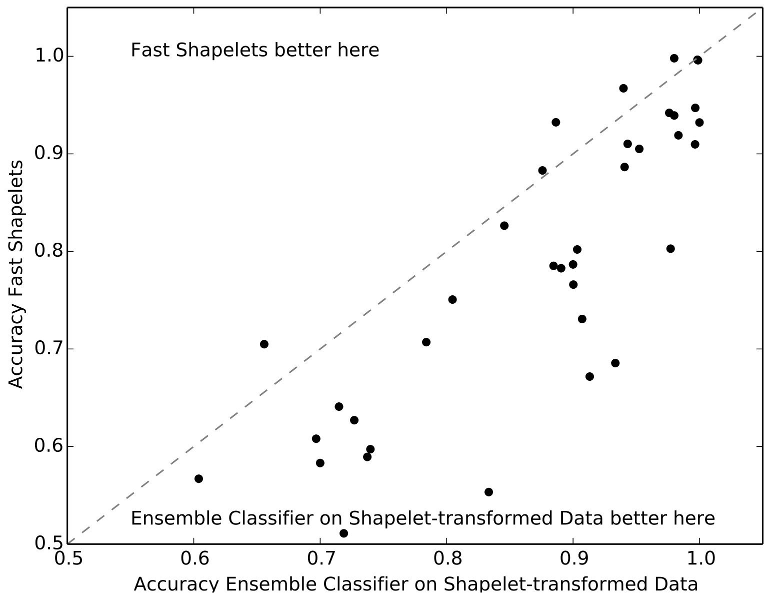

Figure 5.4: Comparison of accuracy over 44 datasets between ensemble classifier on shapelet-transformed data and Fast Shapelets. The ensemble classifier on shapelet-transformed data is better on 39 datasets, Fast Shapelets is better on 5.

5.4 Interpretability and insight

As we have seen in the previous sections, the ensemble classifier on shapelet-transformed data is a more accurate way to classify time-series data than any of the other leading approaches. For many TSC problems, however, accuracy must be accompanied by interpretability. For example, medical professionals may be reluctant to deploy and take ownership of systems they do not fully understand or cannot interpret. Similarly, in finance, fund managers responsible for large accounts must be able to understand and explain the models they use. Machine-learning techniques can solve problems in industry, but only if they are deployed and trusted by professionals.

Interpretability is one of the key reasons for the initial proposal of shapelets [180]. Shape-based approaches are intuitive, and can offer insight into the problem domain that goes beyond their use in accurate classification. In this section, we offer qualitative examples of this insight, drawn from the fields of behavioural genetics, motion analysis, and image processing.

5.4.1 Classifying *Caenorhabditis elegans* locomotion

In Chapter 4, we described two worm movement problems: Worms and WormsTwoClass. In each case, the problem is to classify the type of worm using a time-series representing its movements on an agar plate. We focus here not on the accuracy of this classification, but on the insight into the problem domain that can be extracted as a by-product of the shapelet transformation.

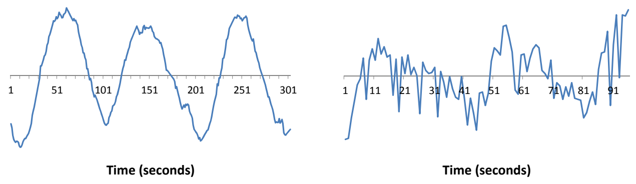

Figure 5.5: Best shapelets for wild-type (left) and mutant worms (right) for the WormsTwoClass dataset.

Figure 5.5 shows the top wild-type shapelet (left) and the top mutant shapelet (right) from the WormsTwoClass problem. A non-expert can immediately see the difference between the two shapelets: the wild-type worm's movement is highly regular, approximating a sine wave (this is referred to as the eigenworm1 shape, see [27]). The mutant worm's movement is much more erratic. It still roughly approximates a wave-like motion, but there are multiple, apparently random deviations that make the shapelet appear more random.

Figure 5.6 shows the best shapelet for each of the five classes (wild type and four mutant types). The shapelets representing the four types of mutant worm are very different under visual inspection, both from one another, and from the regular pattern of the wild-type worm. The shapelets give us an interpretable, visually meaningful, and immediate insight into the problem domain.

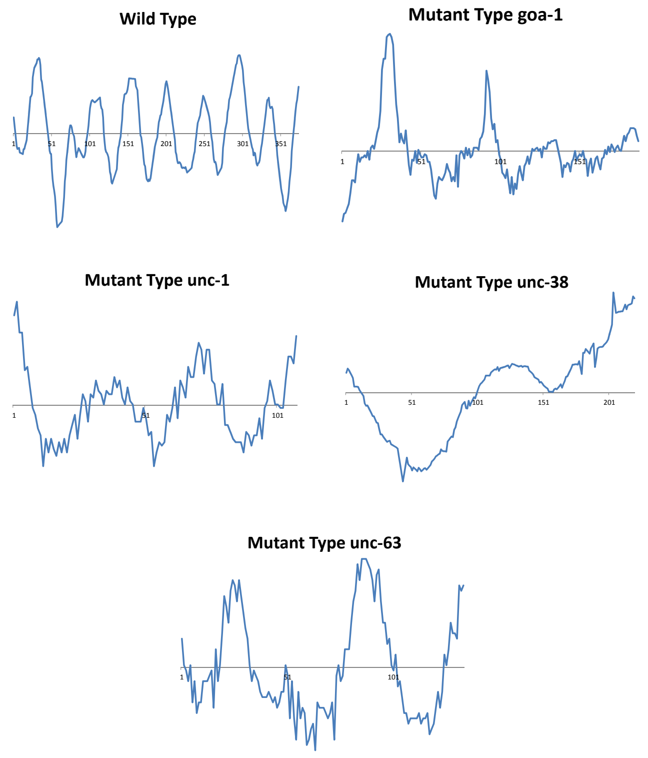

Figure 5.6: Best shapelets for the Worms dataset.

5.4.2 Classifying human motion: the GunPoint dataset

The GunPoint dataset consists of time series representing an actor appearing to draw a gun; the classification problem is to determine whether or not the actor is holding a prop (the *Gun/NoGun* problem). In [180], the authors identify that the most important shapelet for classification occurs when the actor's arm is lowered; if there is no gun, a phenomenon called 'overshoot' occurs, and causes a dip in the time-series data. This is summarised in Figure 5.7, taken from [180].

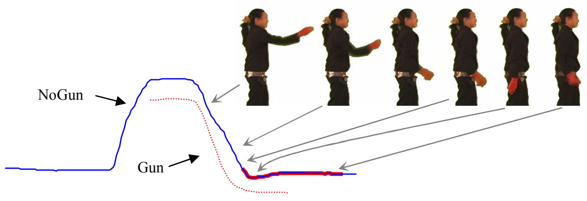

Figure 5.7: An illustration of the *Gun/NoGun* problem taken from [180]. The shapelet that they extract is highlighted at the end of the series.

The shapelet decision tree trained in [180] contains a single shapelet at the end of the series corresponding to the arm being lowered. We shapelet transform the GunPoint dataset using the length parameters specified in the [180]. The top five shapelets are shown in Figure 5.8, along with the shapelet reported in [180].

Figure 5.8 shows that each of the top five shapelets from our transform is closely matched with the shapelet from [180]. Figure 5.9 shows that the best ten shapelets form two distinct clusters. The shapelets to the right of the figure correspond to the moments where the arm is lifted, and are instances where there is a gun. These shapelets correspond to the subtle extra movements required to lift the prop, providing insight into the problem domain.

As a proof of concept, and to explore our findings further, we hierarchically cluster the shapelets extracted from the GunPoint dataset. We cluster by merging the shapelets that are most similar in terms of *sDist*, terminating the clustering when

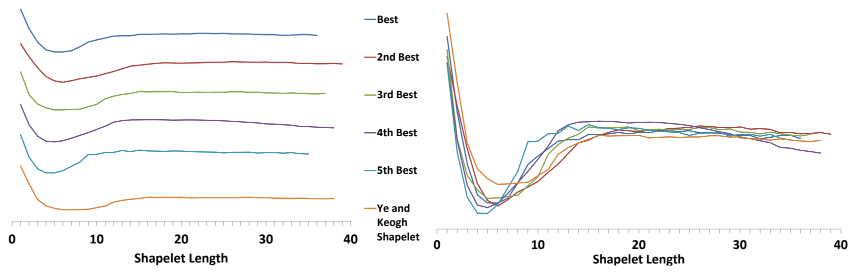

Figure 5.8: An illustration of the five best shapelets extracted by our filter and the shapelet found by Ye and Keogh. The graph to the right shows how closely they match.

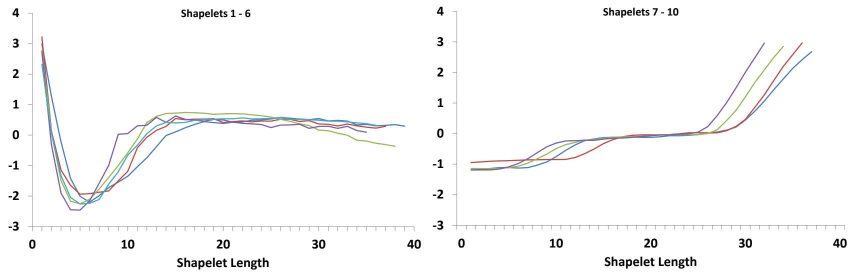

Figure 5.9: The 10 best shapelets for the *Gun/NoGun* problem. The shapelets form two distinct clusters. The graph on the left shows shapelets 1-6. They represent the 'overshoot' motion identified in [180]. The graph on the right shows shapelets 7-10. They represent the extra movement necessary to lift the prop gun when the arm is raised.

five clusters remain (the value is arbitrary; we explore clustering in more detail in Section 5.6). The main benefit of clustering the shapelets is improved interpretability. The top shapelets in the GunPoint dataset form two distinct clusters (Fig. 5.9). When we cluster the shapelets into 5 clusters, we find that the best shapelets in each of the top two clusters represent the two shapelets we found by visual inspection of the full set. This is an encouraging result for clustering as a means to improve interpretability.

5.4.3 Image outline analysis: Beetle/Fly and Bird/Chicken

We apply the simple clustering method we used on the GunPoint dataset to the Beetle/Fly and Bird/Chicken problems. Figure 5.10 shows the top five clustered shapelets from the Beetle/Fly dataset.

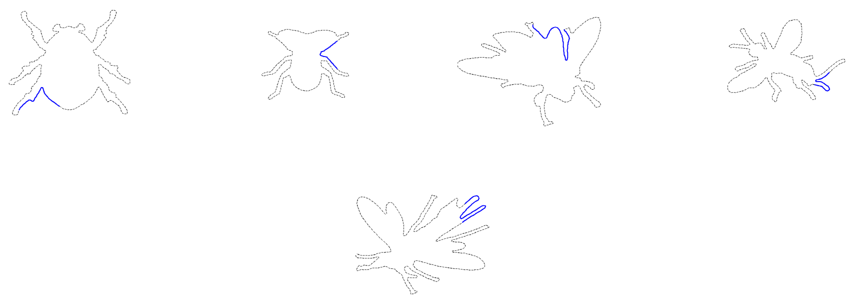

Figure 5.10: The five best clustered shapelets from the Beetle/Fly dataset. The shapelets are highlighted on the outline in blue.

In Figure 5.10, the first two shapelets distinguish members of the beetle class, and the remaining three distinguish members of the fly class. By clustering down to five shapelets, we gain insight into the problem that would be less obvious from the original 256 shapelets. The beetle class is distinguished by a relatively simple angle between the legs and body; the only feature is a knee joint on the leg. The fly class is distinguished by a more complex shapelet, related to the more intricate features of the fly images.

Figure 5.11 shows the top five clustered shapelets for the Bird/Chicken dataset. As is the case with the shapelets from the Beetle/Fly data, it is not obvious *a priori* that the clustered shapelets we have found are the distinguishing features of the two images. Hence, we have discovered something about the problem that would not have been known prior to the shapelet transform. Both Beetle/Fly and Bird/Chicken are toy problems, but the clustered shapelets show how useful the approach could be for analysing image outlines. Shapelets have the potential to provide accurate classification, but also surprising insights into the problem domain. Hence, having demonstrated superior classification accuracy, we shift our focus to improving interpretability.

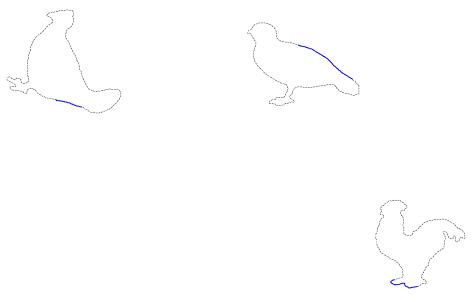
Figure 5.11: The five best clustered shapelets from the Bird/Chicken dataset. The shapelets are highlighted on the outline in blue.

5.4.4 Interpretability

The main difficulty with attempting to gain insight into the problem domain via examining shapelets is the sheer quantity of shapelets. Many shapelets in a set will be matching instances of the same underlying shape; what we are interested in are the underlying shapes that are associated with series of a given class. Finding these shapes manually can involve a long process of visual examination of different shapelets. The two shapelet transforms for the NonInvasiveFetalECG_Thorax datasets, for example, both have 18000 shapelets; examining this number of shapelets manually would be extremely time-consuming, and also very difficult, as the analyst would have to pick out the shapelets that represent repeated instances of the same shape.

We propose an automated system for increasing the insight that can be gleaned from the shapelet transform. The first part of this process involves reducing dimensionality. We do this in two stages. First, we remove any poor shapelets using a filter based on the statistical F test. This is described in Section 5.5. The second stage involves clustering the shapelets (Section 5.6). We cluster with two related aims: we aim to reduce dimensionality, making it easier to inspect the shapelets visually, and we aim to combine shapelets that are instances of the same underlying shape, again making the task of gleaning insight from shapelets easier.

The second part of the process for increasing interpretability involves transforming the shapelet distances into more interpretable binary features, where each binary feature indicates the presence or absence of a shapelet. We see this as the optimum in interpretable shapelet-based classification, see Section 5.8.

5.5 Filtering the shapelet-transformed data using the F-stat

We generate a large number of shapelets to ensure that a varied set is produced (see Table 8.4 in the appendix). Generating 10 shapelets for every series in the training set has the potential to produce very poor shapelets, which are retained because of the high value of k. Such shapelets do not necessarily reduce classification accuracy; rather, they reduce interpretability by increasing the number of shapelets without adding information. Our solution to this problem is to use the F-stat to filter poor shapelets.

We have established a good case for using the F-stat quality measure for assessing shapelets (Section 5.2.3). The measure is faster to compute than IG, particularly on multi-class problems, and is no less accurate. Using the F-stat quality measure has another advantage: shapelets with poor F-stat values can be eliminated using a statistical test.

The basis of the F-stat quality measure for shapelets (see [88]) is the statistical F test. The F-stat quality measure, as noted in Section 5.2.3, is calculated as follows:

$$F = \frac{\sum_i (\bar{D}_i - \bar{D})^2/(C-1)}{\sum_{i=1}^{C} \sum_{d_j \in D_i} (d_j - \bar{D}_i)^2/(n-C)}, \tag{5.5.1}$$

where C is the number of classes, n is the number of series, $\bar{D}_i$ is the average of distances to series of class i and $\bar{D}$ is the overall mean of D, the set of distances.

We can use the F distribution to assess whether a given shapelet divides the dataset into classes better than randomly. A good shapelet will have a high F-stat, indicating that it divides the dataset into the different classes very well. A poor shapelet will have a low F-stat, and it is these shapelets we aim to remove by filtering. Poor shapelets add little to the classification accuracy, and hinder interpretability by increasing the dimensionality of the transformed data.

Under the null hypothesis, the F-stat follows an F distribution with $C - 1$ and $N - C$ degrees of freedom (C is the number of classes and N the size of the training set). We consider a shapelet to have a low F-stat if the CDF of the F distribution for the shapelet's F-stat value is lower than 0.9.

Shapelets that fall below the threshold are removed from the transform, on the grounds that their separation of the classes is no better than a random split. The threshold is set low to avoid removing potentially good shapelets, but other values could be used to make the filter more or less exacting.

Figure 5.12 shows the change in accuracy for the ensemble classifier if it is trained on filtered, rather than unfiltered, data. The filtered datasets are no less accurate than the original shapelet transformed datasets, where difference is tested using a Wilcoxon Signed Rank Test. The average reduction in the number of shapelets is approximately 4%, with no loss of accuracy; the datasets that have lost the greatest number of shapelets (WormsTwoClass, TwoLeadECG, MoteStrain, and CinC_ECG_torso) show only marginal losses in accuracy (less than 2% for Cin_C_ECG_torso, and less than a tenth of a percent for MoteStrain) or increased accuracy (approximately 2% for WormsTwoClass and approximately a tenth of a percent for TwoLeadECG) despite the removal of between 15% and 20% of the shapelets in the transform. These datasets contain many poor shapelets by our F-Stat criteria, and as such, have reaped the most benefit from filtering in terms of the number of shapelets removed. Table 8.2 (see appendix) presents accuracies for the filtered datasets; the number of shapelets in the

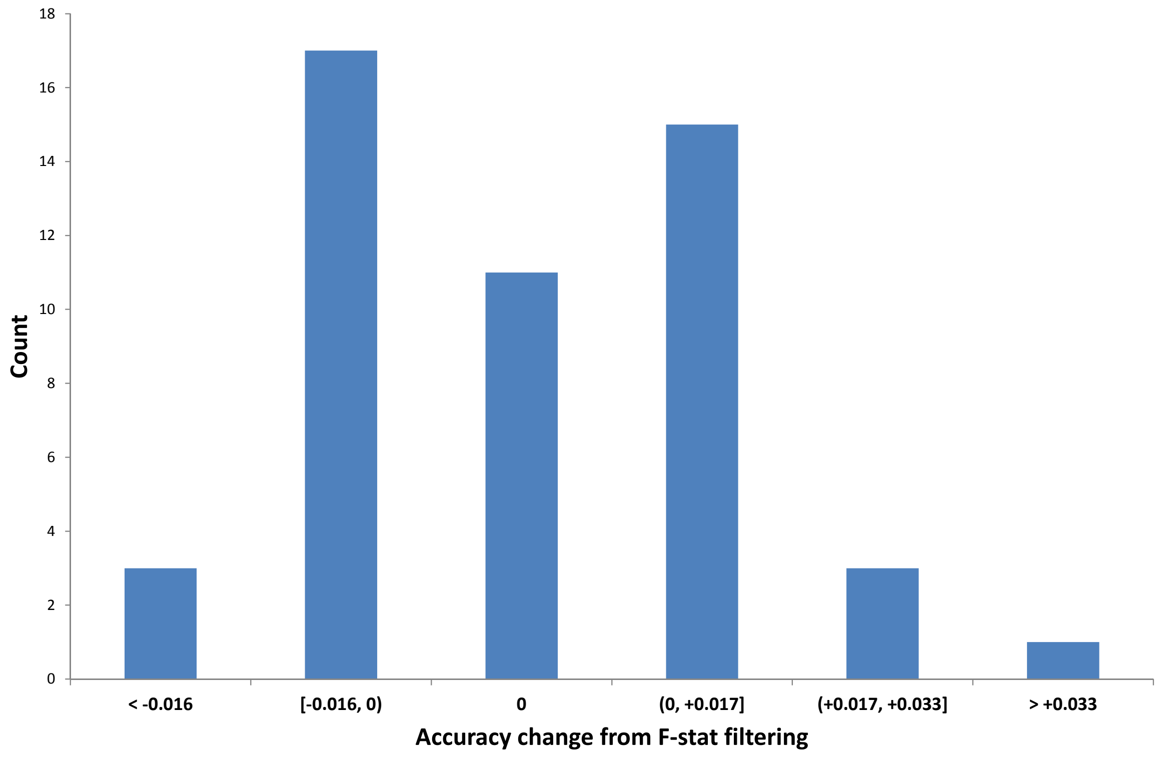

Figure 5.12: Histogram of difference in accuracy for the ensemble classifier before and after F-stat filtering (filtered accuracy - unfiltered accuracy). A positive value indicates that the ensemble is more accurate on filtered data.

filtered datasets is shown in Table 8.7.

5.6 Clustering shapelets

By definition, a shapelet that discriminates well between classes will be similar to a set of subsequences from other instances of the same class. A given shapelet has a high quality value because it has numerous matches throughout the set of instances of the same class. If the limit on the number of shapelets to find (k) is set to a high enough value, the transform will find all of these matches. It is very common for a given transform to include multiple shapelets that match one another. Each shapelet is an instance of the same underlying shape, a shape that occurs across different series of the same class. In terms of insight, what we care about is how a given shapelet represents the underlying shape, and about the shape itself, and what it tells us about the problem domain.

The maximum number of shapelets found using the transform is restricted to $10N$, where N is the number of instances in the training set. Clustering the shapelets offers a number of potential benefits. First, reducing the dimensionality of a dataset allows for faster classification, and may make tractable some techniques that do not scale well. Second, if the clustering sufficiently reduces the number of shapelets, interpretability is greatly increased. For example, a clustering that does not reduce accuracy and yields one shapelet per class provides highly interpretable data. This is the case when we cluster the FaceFour dataset, which goes from 174 shapelets to 4 shapelets with no loss of accuracy (see appendix, Tables 8.2 and 8.7). Third, clustering removes duplicate shapelets, and close duplicates decrease interpretability, as they are treated as different shapelets despite intuitively representing different instances of the same shapelet.

Hierarchical clustering based on shapelet distance ($sDist$) is an intuitive approach to selecting the best shapelets from a large initial set. Shapelets are defined in terms of their visual similarity, so it makes sense to use $sDist$ to cluster them. The shapelets are formed into clusters by combining the shapelets that are the most similar in terms of *sDist*. The most important design choice is choosing where to stop the clustering. We test four different methods, two that hierarchically cluster the set of shapelets by *sDist* down to a single cluster and select the clustering that maximise some value (silhouette or cross-validation accuracy), one that clusters shapelets by creating clusters and adding shapelets to them using Minimum Description Length (*MDL*; this method does not use *sDist*), and another method that clusters shapelets hierarchically using *sDist*, but stops the clustering when there is no longer a saving to be made, judged using MDL.

5.6.1 Hierarchical clustering with quality measure

We use the form of clustering shown in Algorithm 10 to find clusters hierarchically, using the cross-validation accuracy or the silhouette clustering metric to find the optimum number of clusters. The algorithm is designed to find the clustering with the best value of the given metric, choosing fewer clusters where values are equal. The clusters are stored as a set of sets, **C**, where the original cardinality of **C** is equal to the size of the original shapelet set **S**.

Algorithm 10 `hierarchicalClusteringWithAssessment`(**S**,**T**)

```
1:  $\mathbf{C} = \{\{S_1\}, \{S_2\}, ..., \{S_k\}\}$
2:  $bestAccuracy$ = assessAccuracy($\mathbf{C}$)
3:  $bestClustering = \mathbf{C}$
4:  $\mathbf{M}$ = createDistanceMap($\mathbf{C}$)
5:  while $|\mathbf{C}| > 1$ do
6:      $closestPair$ = searchDistanceMap($\mathbf{M}$)
7:      $\mathbf{C}$ = cluster($\mathbf{C}$, $closestPair$)
8:      $accuracy$ = assessAccuracy($\mathbf{C}$,$\mathbf{T}$)
9:      $\mathbf{M}$ = createDistanceMap($\mathbf{C}$)
10:     if $accuracy \geq bestAccuracy$ then
11:         $bestAccuracy = accuracy$
12:         $bestClustering = \mathbf{C}$
13: return convertClustersToShapelets($bestClustering$)
```

The `createDistanceMap` method takes a set of clustered shapelets, $\mathbf{C}$, as input and returns a $|\mathbf{C}| \times |\mathbf{C}|$ matrix of the distances between the clusters in $\mathbf{C}$. We take the distance between two clusters of shapelets to be the average *sDist* between the members of those clusters (*average linkage*). `searchDistanceMap` takes a distance map as an input and returns the indexes of the pair of clusters with the smallest average *sDist* between them. The `cluster` method takes the set of clusters, $\mathbf{C}$, and the indexes of the closest pair of clusters as input, merges the members of the closest pair into a new cluster, adds the new cluster to $\mathbf{C}$, deletes the clusters that formed the closest pair, and returns $\mathbf{C}$. `convertClustersToShapelets` takes a set of clusters, $\mathbf{C}$, as an input, and returns a set of shapelets of the same cardinality, where each cluster in $\mathbf{C}$ is represented by the best shapelet in that cluster (judged by the appropriate shapelet quality measure). We exclude full algorithmic descriptions of these methods. We implement two different versions of the `assessAccuracy` method, one which uses cross-validation accuracy, the other the silhouette clustering metric.

Hierarchical clustering using cross-validation accuracy

For this type of clustering, we implement the `assessAccuracy` function (line 8 of Algorithm 10) by measuring the leave-one-out-cross-validation accuracy of a 1NN classifier with Euclidean distance on the training set. The procedure is shown in Algorithm 11.

This method will select the clustering with the best cross-validation accuracy on the training set. In practice, we do not perform the shapelet transform at line 2 of Algorithm 11; rather, we select the fields from the full transformed dataset that correspond to the shapelets in $\mathbf{S}$. The main weakness of this method is that it is time consuming, particularly if the training set is large, as the time complexity will be in the order of $O(|\mathbf{S}|^2|\mathbf{T}|^2)$ where $|\mathbf{T}|$ is the size of the training set and $|\mathbf{S}|$ is the size of the set of shapelets in the original transformed dataset.

Algorithm 11 `assessAccuracyCV`($\mathbf{C}$,$\mathbf{T}$)

```
1: $\mathbf{S} = \texttt{convertClustersToShapelets}(\mathbf{C})$
2: $\mathbf{T_S} = \texttt{shapeletTransform}(\mathbf{S}, \mathbf{T})$
3: $totalCorrect = 0$
4: for all Records $T$ in $\mathbf{T_S}$ do
5:     $\texttt{build1NNClassifier}(\mathbf{T_S} - T)$
6:     $prediction = \texttt{classifyInstance}(T)$
7:     if $prediction = T.classValue$ then
8:         $totalCorrect$++
9: $accuracy = totalCorrect/|\mathbf{T_S}|$
10: return $accuracy$
```

Hierarchical clustering using silhouette

The silhouette measure of the quality of a clustering is defined as the mean $S(i)$ over all data where:

$$S(i) = \frac{b(i) - a(i)}{\texttt{max}(a(i), b(i))}. \tag{5.6.1}$$

$a(i)$ is the average distance (we use $sDist$, which is already computed as part of the clustering process) between data point i and the other data in its cluster. $b(i)$ is the average distance between data point i and the data points that belong to i's neighbouring cluster. The neighbouring cluster for any data point is the cluster with the lowest average distance to that data point other than the cluster to which the data point belongs. For any clustering, $-1 \leq S(i) \leq 1$.

Silhouette is a standard measure of quality for clustering (see, for example, [169]). Using the silhouette measure is less time consuming than using the cross-validation accuracy, with a time complexity of $O(|\mathbf{S}|^3)$ where $|\mathbf{S}|$ is the set of shapelets in the original transformed dataset. It is still necessary to check every clustering from $|\mathbf{S}| - 1$ clusters to 1, meaning that every use of the algorithm has the worst-case time complexity.

5.6.2 Hierarchical Clustering via Minimum Description Length

Clustering using silhouette or cross-validation accuracy is time consuming because it requires every clustering to be assessed. A more efficient method would be to stop the clustering at some estimated optimum number of clusters, mitigating the need to examine the clusterings beyond that point.

The *Minimum Description Length*(*MDL*) framework can be used to cluster time series [142], for dimensionality reduction [92], and for stopping in semi-supervised clustering [16].

Clustering by MDL is described in detail in [142]. The technique used is aimed at clustering time-series subsequences from a streaming time series. Their algorithm inspects each pair of subsequences and stores the *bit save* made by clustering the pair together. A bit save is the saving made by reducing the number of bits necessary to store the data. If two subsequences are similar, it should be possible to make a saving, judged in terms of MDL, by storing the centroid and the difference vectors of the two series (in fact, one series in every cluster need not be stored, as it can be recovered from the centroid and the other difference vectors). If the subsequences in the cluster are all similar to the centroid (as they should be in a good cluster) then the difference vectors will be close to straight lines with gradient 0. This is very cheap to store in terms of MDL, and hence represents a substantial saving over storing the subsequences separately.

The method proposed in [142] is for unsupervised clustering of time-series subsequences from a streaming time series. It is not designed for clustering shapelets, which have an associated class, and must visually resemble one another to be considered a good match. We make a number of modifications to Rakthanmanon *et. al*'s method to optimise the approach for clustering shapelets (see Sections 5.6.3 to 5.6.5).

The first stage in using MDL for clustering shapelets is to discretise the shapelets to six-bit precision. Six bits is an arbitrary level of precision; experiments performed by Hu *et al.* [92] on the UCR datasets showed that transforming to six-bit precision from double precision made no substantial difference to classification accuracy. Once the shapelets are clustered, we use the original shapelets, not the discretised shapelets, further minimising any impact the discretisation might have.

We perform the discretisation using the procedure outlined in [92]. The minimum and maximum values are found by examining every shapelet. The following formula is used to discretise the series:

$$Discretisation_b(T) = round\left(\frac{T - min}{max - min} \cdot (2^b - 1)\right) - 2^{b-1}. \qquad (5.6.2)$$

max and min are found over the entire set of shapelets, rather than found separately for the individual shapelets; b is the number of bits we discretise the series to, in this case, six.

Once the shapelets are discretised, we can define the description length of a shapelet. The entropy of a time series is a lower bound on the average code length from any encoding of that series [142]. Hence, we can use the entropy of the shapelet as its description length.

To calculate the entropy of a discretised shapelet, S, we create the set of unique values that occur in that shapelet, $V_S = \{v : v \text{ is the value of a point in } S\}$. Using V_S and S, we can define a probability, $P(v)$ for each value v in V_S. $P(v)$ is the probability that a point, s in the shapelet S, takes the value v; it is calculated as follows:

$$P(v) = \frac{|\{s : s \in S \land s = v\}|}{|S|}. \qquad (5.6.3)$$

For each value v in the set V_S, we calculate $P(v)$. This allows us to calculate the entropy, $H(S)$, of the discretised shapelet S:

$$H(S) = -\sum_{v \in V_S} P(v) \cdot \log_2 P(v). \qquad (5.6.4)$$

We define the description length for a length m Shapelet S as follows [142]:

$$DL(S) = m \cdot H(S). \tag{5.6.5}$$

In order to cluster shapelets using MDL, we must define the description length of a cluster. To do this, we calculate the description length of the centroid of the cluster using the formula above. For each member of the cluster, we create a difference vector, which is equal to the difference between each point in the member and the centroid. The total description length of the cluster is equal to the description length of the centroid plus the sum of the description lengths of the difference vectors of the members, minus the difference vector of the member with the largest description length (as this information can be recovered from the centroid and the other members). For a cluster, $\mathbf{C}$, with a centroid, C_{cent}, we denote the difference vector between a member of the cluster, c, and the centroid as C_{cent-c}. The description length is calculated as follows:

$$DL(\mathbf{C}) = DL(C_{cent}) + \left(\sum_{c \in \mathbf{C}} DL(C_{cent-c}) \right) - \max_{c \in \mathbf{C}} (DL(C_{cent-c})). \tag{5.6.6}$$

The shapelets we cluster may be of different lengths, which means that the centroid cannot be a simple point-by-point average. In [142], the authors allow any possible offset between the centroid and the members of the cluster. Figure 5.13 shows an example of this, using different offsets of shapelets from the GunPoint dataset. The approach in [142] makes sense for unsupervised clustering of subsequences from a streaming time series; it is possible for the algorithm to identify two separate parts of a longer series. For shapelets, however, the approach makes less sense. In Figure 5.13, only the centroid at offset $= 0$ closely resembles the members of the cluster. Given that matching shapelets are, by definition, visually similar, we should not allow such centroids to be created. Hence, we restrict the length of the centroid to the length of the longest member, and select the offset by sliding the shorter shapelet along the longer shapelet.

When two shapelets are clustered (or a shapelet is added to an existing cluster, or two clusters are merged), all allowable offsets of the shapelets (or centroids for existing clusters) are tested, and the offset giving the smallest total description length is selected. For two shapelets, S_i and S_j, of lengths L_i and L_j respectively, where $L_i \geq L_j$, the possible offsets range from 0 to $i - j$, where 0 indicates that the first indexes of the shapelets are aligned, and a positive integer indicates the index of S_i aligned with the first point in S_j.

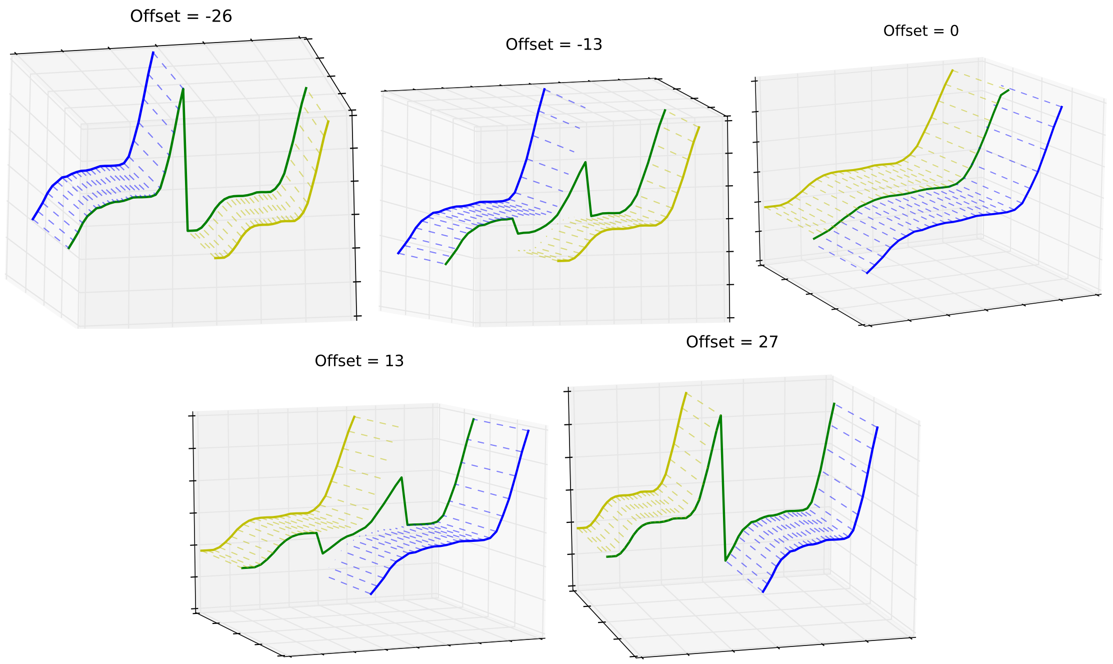

Figure 5.13: The effect of different offset values on the centroid formed by clustering two shapelets from the GunPoint dataset. The shapelets (blue and yellow) and the centroid (green) are offset on the z-axis for ease of presentation. Because the shapelets represent two instances that belong to the same cluster, the centroid is very similar to the members of the cluster at offset=0. As the offset moves away from zero in either direction, the centroid becomes less like the members of the cluster. This is accompanied by an increase in the description length of the cluster, allowing the algorithm to select the appropriate offset.

A good cluster will contain members that are very similar to one another, and hence very similar to the centroid. If all members of a cluster are very similar to the centroid of that cluster, the difference vectors will approximate straight lines of 0 gradient. In this situation, the total description length of the cluster will be small, as the difference vectors will have very small description lengths.

A *bit save* is achieved when storing shapelets in a cluster results in a smaller total description length than storing the shapelets individually.

We adopt the basic MDL framework, and use it to create two novel shapelet clustering methods.

5.6.3 Hierarchical clustering using MDL

Our first MDL-based clustering method uses the greedy algorithm described in [142] to cluster the shapelets exactly as time-series subsequences are clustered in that paper (barring our restrictions on offsets). The algorithm operates as follows:

- Each shapelet and cluster is tested against every other shapelet and cluster. The best bit save is recorded.

- If the best bit save is greater than 0, the operation with that bit save is carried out: two shapelets are merged to form a new cluster, a shapelet is merged into an existing cluster, or two clusters are merged. After performing the operation, the algorithm returns to the previous step. If the best bit save is 0 or less, the algorithm terminates.

One issue with using this form of clustering for shapelets is that the centroid need not be visually similar to the members of the cluster for the clustering to achieve a high bit save. Consider Fig 5.14 left, which shows two shapelets from the ItalyPowerDemand dataset that are clustered together by MDL clustering. Visually, they are opposites. The blue shapelet has a local maximum in the middle, whereas the yellow shapelet has a local minimum. The centroid, which is the average of the two shapelets, has a horizontal line in the same position; it resembles neither member of the cluster. This is a problem, because shapelets that match are different instances of the same shape, and should, therefore, be similar under visual inspection. The following example demonstrates how such pathological clusterings can occur during the first stage of the algorithm.

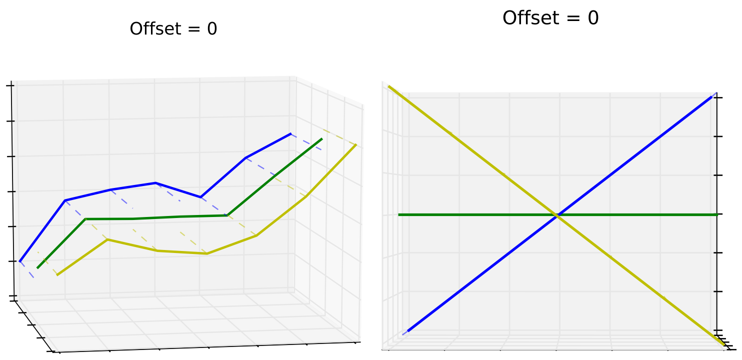

Figure 5.14: Left: two shapelets from the ItalyPowerDemand dataset. The blue and yellow shapelets are inverses in the middle third, resulting in a cheap-to-store flat line in the centroid and an incorrect clustering. Right: an extreme example showing two completely opposite shapelets that create a large bit save when clustered because the centroid is a horizontal line.

Suppose we test the bit save offered by merging two shapelets, S_1 and S_2, into a cluster C. They are a perfect match, so the centroid will be identical to the two shapelets; because of this, $DL(C_{cent}) = DL(S_1) = DL(S_2)$. The shapelets are identical to the centroid, meaning the difference vectors are horizontal lines; hence $DL(C_{cent-S_1}) = DL(C_{cent-S_2}) = 0$. The overall bit save offered by clustering S_1 and S_2 is:

$$DL(S_1) + DL(S_2) - DL(C_{cent}) - DL(C_{cent-S_1}) - DL(C_{cent-S_2}) + \max_{i=1,2}(DL(C_{cent-S_i})), \tag{5.6.7}$$

which is equal to the description length of one of the shapelets, or that of the centroid.

Now suppose we test the bit save offered by merging S_1 with its mirror opposite, S_{inv}. $DL(S_{inv}) = DL(S_1)$ as they are mirror opposites. C_{cent} will be a horizontal line; $DL(C_{cent}) = 0$. Because the centroid stores no information, the difference vectors must store the entire shapelet, and $DL(C_{cent-S_1}) = DL(C_{cent-S_{inv}}) = DL(S_1) = DL(S_{inv})$. In this case, the bit save is:

$$\begin{aligned} DL(S_1) + DL(S_{inv}) - DL(C_{cent}) - DL(C_{cent-S_1}) - DL(C_{cent-S_2}) \\ + \max(DL(C_{cent-S_1}, DL(C_{cent-S_{inv}})). \end{aligned} \tag{5.6.8}$$

Here we find exactly the same bit save as in the previous example; the bit save offered by clustering mirror opposites is the same as that offered by clustering perfect matches. The same is true for approximate matches and approximate mirror opposites.

We conclude that the clustering proposed in [142] can produce clusters that are not appropriate for shapelets, which require visual similarity to be considered a match. We aim to mitigate this problem by proposing a different form of clustering that makes use of MDL as a stopping criterion, but does not use it for the clustering.

5.6.4 Hierarchical clustering based on *sDist* with MDL stopping criterion

To mitigate the problems that may occur from using the clustering by MDL method presented in [142] to cluster shapelets, while keeping the key intuition that MDL is a principled way to select the correct number of clusters, we modify the algorithm to create a new form of clustering, *MDLStop*.

MDLStop proceeds in the same fashion as the hierarchical clustering proposed in Section 5.6.1. The distance map of the *sDists* between each pair of shapelets or cluster of shapelets is searched, and the shapelet or cluster with the smallest (average) distance to another shapelet or cluster is tested. The test ascertains the bit save made from merging the shapelets or clusters in question. If the bit save is positive, the merge is enacted, and the algorithm loops and resumes. If the bit save is zero or less, the algorithm terminates.

The time complexity of MDLStop in the worst case is $O(L^2|\mathbf{S}|)$, where L is the shapelet length and $|\mathbf{S}|$ is the size of the set of shapelets. In the best case, the algorithm will terminate after a single check requiring $O(L)$ operations. In contrast, both the silhouette and CV assessment methods have the same time complexity in every case: silhouette is $O(|\mathbf{S}|^3)$ where $|\mathbf{S}|$ is the size of the set of shapelets, and the CV method is $O(|\mathbf{S}|^2|\mathbf{T}|^2)$ where $|\mathbf{T}|$ is the size of the training set. Hence, in the average case, MDLStop is faster than the other two methods. In the worst case, MDLStop and silhouette are both of cubic complexity; however, the number of shapelets is less than the maximum shapelet length in only 8 out of 75 datasets, and the difference is an order of magnitude in only one (Cin_C_ECG_torso). For the majority of datasets, the number of shapelets is much larger than the maximum shapelet length, meaning that MDLStop will be faster than silhouette, even in the worst case. We conclude that, in the vast majority of cases, MDLStop offers a substantial speed up over the silhouette and CV assessment methods. Our experience of using the different clustering methods confirms this, as MDLStopCE is able to complete on all 75 datasets; the silhouette and cross-validation methods would not finish for the largest datasets within a reasonable period of time (five days, which is the default limit on the high-performance computing facility we use for our experiments).

5.6.5 Class enforcement for MDL techniques

The MDL-based clustering proposed in [142] is intended for the unsupervised task of finding repeated time-series patterns in a long time series; MDL provides a parameter-free method for clustering similar subsequences. A pair of subsequences (or a subsequence and a centroid) may be clustered if doing so offers a positive bit save. This creates a problem when we use bit save as a stopping criterion for hierarchical clustering of shapelets: clustering will continue for as long as a positive bit save can be found. If the shapelets of different classes are sufficiently similar, they will be clustered.

Consider the shapelets shown in Fig. 5.15. The shapelets of different classes are distinguishable only by minor differences. Such differences are enough to prevent them being clustered together until very late in the clustering process. With silhouette or CV as the assessment measure, there will be a sharp drop in the relevant metric when the clustering begins to merge shapelets of different classes, enabling us to select an earlier clustering in the hierarchy. If MDL is used, however, we may still achieve a positive bit save, resulting in a very poor clustering. Dissimilarities between the shapelets of different classes may be pronounced in terms of $sDist$, but small enough to allow the bit save from clustering them to be positive.

To mitigate this problem, we further adapt MDL-based clustering for shapelets by allowing two shapelets to be clustered only if they come from series of the same class. In the next section, we test whether class enforcement significantly improves the accuracy of the ensemble on data clustered using the MDL and MDLStop clustering methods.

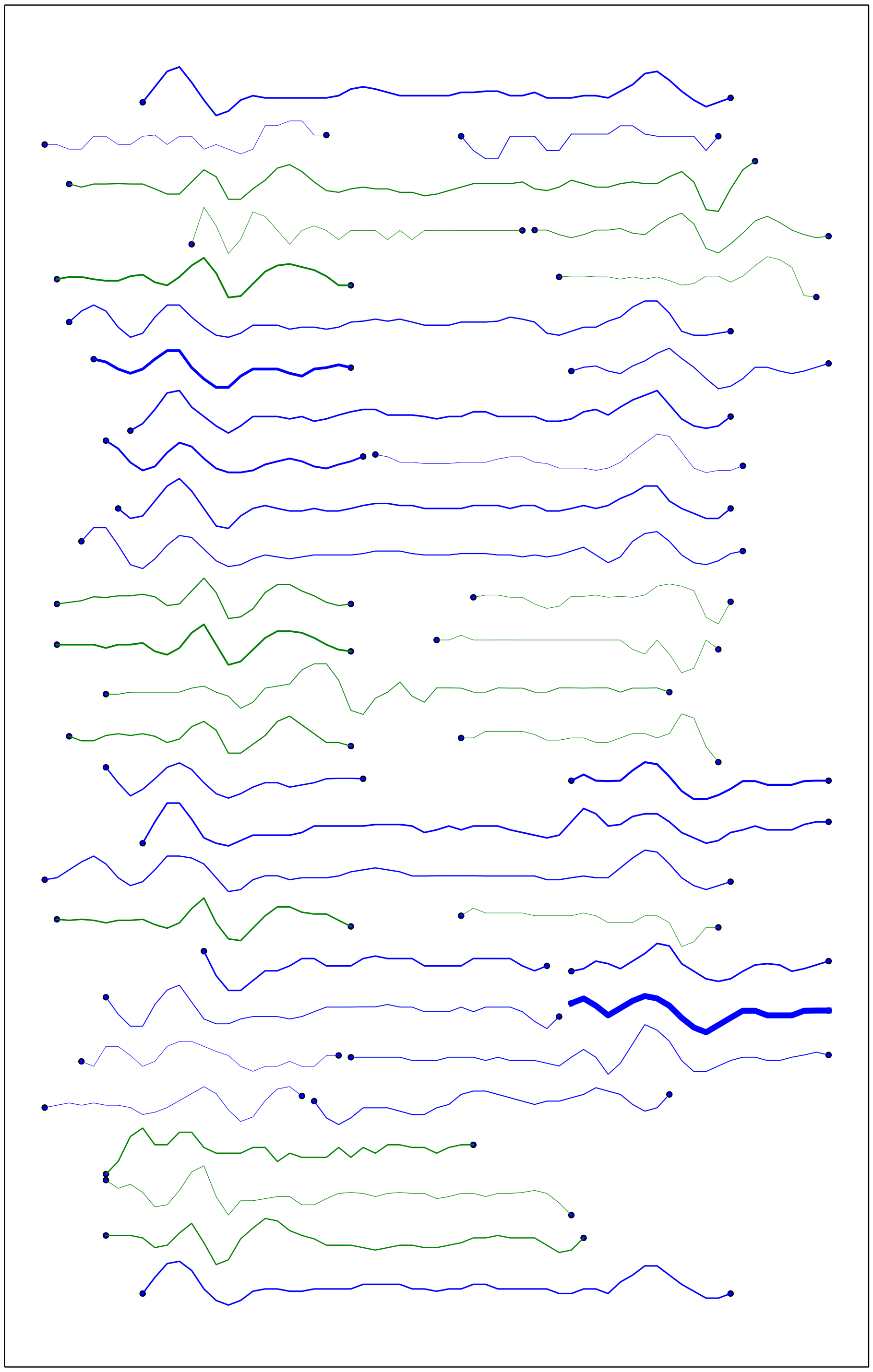

Figure 5.15: Shapelets from the filtered SonyAIBORobotSurfaceII dataset, arranged on the x-axis by where they appear in their respective series, and on the y-axis by the ID of the series they are taken from. The quality of the shapelet is indicated by the thickness of the lines, the class by the colour.

5.7 Clustering results

We compare the different clustering methods over 50 time-series datasets in terms of the accuracy of the ensemble on that data. First, we show that class enforcement significantly improves our novel clustering approach, MDLStop. Then we show that MDLStop with class enforcement is superior to the MDL clustering with class enforcement. Finally, we show that MDLStop with class enforcement produces datasets on which the ensemble is no less accurate than on the unclustered data. The acronyms we use are shown in Table 5.3.

5.7.1 Effects of class enforcement for MDL-based clustering

Our first experiment is a pairwise test of the effect of class enforcement on the MDL-based clustering methods. We hypothesise that enforcing the class distinction for the MDL and MDLStop clustering methods will improve their accuracy. We use a Wilcoxon Signed Rank test with a significance level of 0.01 on the accuracies of the ensemble on fifty datasets.

For MDL clustering, we find that there is no significant difference between MDL and MDLCE. Class enforcement has a strong effect on MDLStop, however; the shapelet ensemble is significantly more accurate on shapelet-transformed data clustered with MDLStopCE than that clustered with MDLStop. The p-value is 5.55×10^{-5}. Table 8.3 (see appendix) presents the full set of results.

Table 5.3: Acronyms used for clustering methods.

Acronym	Full name
Unclustered	Ensemble classifier on shapelet-transformed data.
CV	Hierarchical clustering with assessment by cross-validation accuracy.
Sil	Hierarchical clustering with assessment by silhouette.
MDL	Hierarchical clustering based on bit save with MDL-based stopping criterion.
MDLStop	Hierarchical clustering based on *sDist* with MDL-based stopping criterion.
MDLCE	MDL with class enforcement.
MDLStopCE	MDLStop with class enforcement.

We select the class-enforced versions of the MDL-based clustering methods. Class enforcement makes no significant difference to the performance of MDL clustering, so we do no harm to the method by using it, and offers a significant improvement for MDLStop clustering. We feel that enforcing class is sensible with regard to the way shapelets are found and used, and that it represents a move away from the unsupervised roots of MDL-based clustering, and toward the supervised task of time-series classification, which is our area of interest.

5.7.2 MDLStopCE vs MDLCE

Our next experiment is a pairwise comparison of the two MDL-based clustering methods. We test the differences using a Wilcoxon Signed Rank test with a significance level of 0.01 on the accuracies of the ensemble classifier on 50 datasets.

Our experiment shows that the ensemble classifier is more accurate on shapelet-transformed data clustered using MDLStopCE. The p-value is 2.138×10^{-3}. We display these results graphically in Figure 5.16.

MDLStopCE is, therefore, our preferred MDL-based clustering method. This result makes intuitive sense, as MDLStopCE is engineered for the shapelet approach, and is very different to the original method proposed in [142] for unsupervised clustering of subsequences from streaming time series.

5.7.3 Comparison of clustering methods

We compare the three clustering methods (silhouette, cross-validation, and MDL-StopCE) to the unclustered shapelet-transformed data by measuring the accuracy of the ensemble classifier over 50 datasets transformed using each method. The results are presented in the form of a critical-difference diagram in Figure 5.17, which is based on the Friedman test.

As can be seen from Figure 5.17, there is no significant difference in the ensemble's

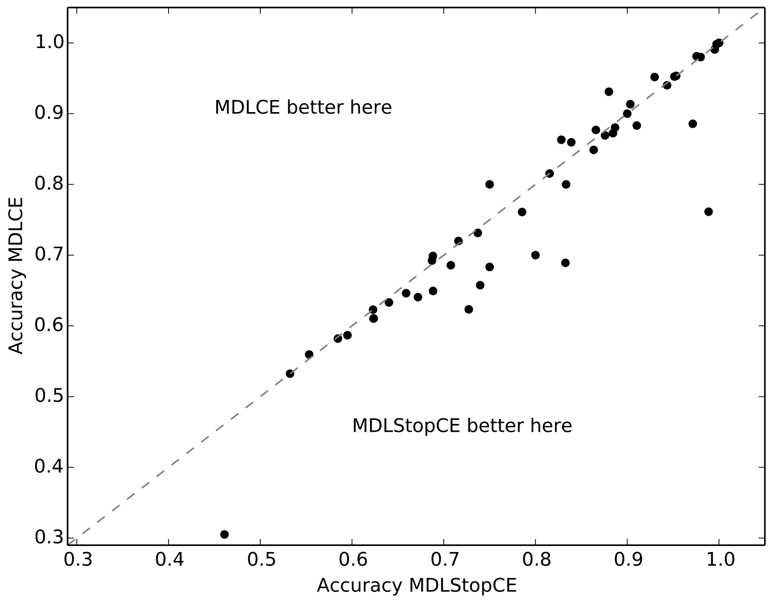

Figure 5.16: Comparison of accuracy over 50 datasets between the ensemble classifier on MDLCE-clustered shapelet-transformed data and MDLStopCE-clustered shapelet-transformed data. MDLStopCE is better on 28 datasets, MDLCE is better on 14.

accuracy between the unclustered shapelet sets, those where the number of clusters have been selected using cross-validation, and those where the number of clusters have been selected by MDLStopCE, as all three belong to the same clique. The ensemble is significantly less accurate if the number of clusters is selected using the silhouette method.

Classification using MDLStopCE-clustered data is no less accurate than classification on the unclustered shapelet-transformed data or the CV-clustered data. MDLStopCE also offers faster clustering than using CV, and better interpretability than using the unclustered shapelet-transformed data.

The time complexity of MDLStopCE ($O(L^2|\mathbf{S}|)$ in the worst case; much faster in the average case) is better than that of cross validation ($O(|\mathbf{S}|^2|\mathbf{T}|^2)$ in every case). MDLStopCE is viable on all 75 datasets, whereas the cross-validation method cannot be used on the largest datasets in a reasonable time frame.

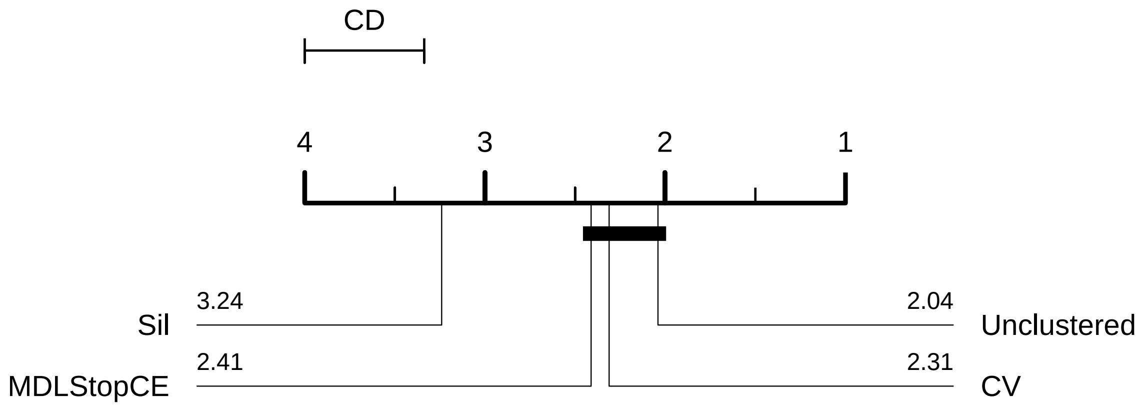

Figure 5.17: Critical-difference diagram of ranked differences in accuracy of the ensemble between the different clustering methods and the unclustered shapelet-transformed data on 50 datasets.

MDLStopCE-clustered datasets are also more interpretable than those with 10N shapelets, as, in the general case, there are fewer shapelets, and fewer repeated instances of the same shaplet. An example of this increase in interpretability comes from SimulatedSet, our data created to be optimal for the shapelet approach (see Chapter 4). The shapelet transform finds 1000 shapelets in SimulatedSet. Using the full 1000, the accuracy of the ensemble is 0.913. After clustering with MDLStopCE, only two shapelets remain (Figure 5.18), and the accuracy of the ensemble is 0.88. There is an enormous reduction in dimensionality, and the data are much more interpretable, as each series is represented by two distances, one to each shapelet; the only cost is a small reduction in accuracy.

Table 8.7 (see appendix), shows the number of shapelets in each dataset before and after MDLStopCE clustering.

Our next step in increasing the interpretability of shapelet-transformed data is to replace the distances that make up the features of the transformed data with binary values reflecting the presence or absence of a given shapelet.

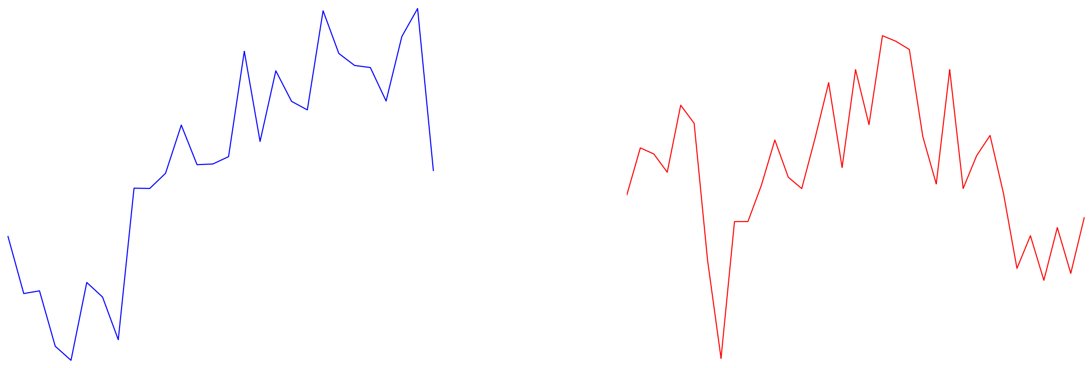

Figure 5.18: The two shapelets selected from the SimulatedSet shapelet transform by MDLStopCE clustering. Left: a spike from class 0. Right: a triangle from class 1. These shapelets are the highest-quality members of their clusters.

5.8 Binary discretisation

Shapelet-transformed data, including the data we create using the MDLStopCE clustered shapelets, encodes each instance of a time-series dataset as a set of real-valued distances. These distances are the $sDist$ between each shapelet and the instance in question, with a smaller distance indicating a closer match between the series and the shapelet. The intuition is that, given the shapelets have been selected from the training set to be maximally discriminative of class, those instances with smaller $sDist$ to a shapelet from a training instance of class c_1 are more likely to be of class c_1.

We use IG to find the optimum distance at which a given shapelet splits the training data by class, then transform each set of shapelet-transformed data into binary data by assessing whether a given $sDist$ is lower or higher than the optimum splitting distance. For instance i and shapelet S, if the information gain optimum splitting point of S is found to be S_{sp} on the training data, and $S_i < S_{sp}$, in the binary-transformed data, the equivalent attribute, B_i, will be given the value 0. If $S_i \geq S_{sp}$, B_i will be given the value 1.

$$B_i = \begin{cases} 0 & \text{if } S_i < S_{sp} \\ 1 & \text{if } S_i \geq S_{sp} \end{cases}$$

The benefits offered by binary-transformed data include increased interpretability and faster classification. Interpretability is increased because each value indicates the presence or absence of the shapelet in the series, rather than a real-valued distance. Classification is faster because calculations are faster on binary values than on real values.

We would expect the accuracy of our ensemble classifier to decrease, as the classifiers have less information available after the binary transformation. The results of our experiments to assess this effect are shown in Section 5.9. We compare two different methods of performing the binary transform, a method based on finding the best binary split point, and a novel method that transforms the class labels before finding the split, motivated by the idea that a shapelet discriminates only a single class from the other classes.

5.8.1 Standard binary transform

We refer to the straightforward identification of a binary splitting point for each shapelet as the Standard Binary Transform.

The weakness of this approach is evident when we consider multi-class problems. A given shapelet is selected because it discriminates one class from every other class. When we find the split point for a shapelet in a dataset with more than two classes, the split can end up in what is intuitively the wrong place for that shapelet.

Consider Figure 5.19, which shows data from the CBF training set, plotted by distance from shapelet 0 (x-axis) and shapelet 1 (y-axis). Shapelet 0 best discriminates instances of class 1, and shapelet 1 best discriminates instances of class 2. The optimum splitting point for each shapelet is represented with a dotted line along the appropriate axis (vertical for shapelet 0, Fig. 5.19 left, horizontal for shapelet 1, Fig. 5.19 right).

Intuitively, we can see from Fig. 5.19 right that the correct splitting point has been selected for shapelet 1. The blue points, which represent instances of class 2, have been cleanly delimited from the other points. Fig. 5.19 left, however, shows that the optimum splitting point for shapelet 0 does not match our intuition. Shapelet 0 is the shapelet that best discriminates training instances of class 1, represented by the red points, from those of classes 2 and 3 (blue and green points respectively). The optimum splitting point does not linearly separate the red points from the others. Instead, it separates the blue points from the other points. This occurs because the information gain method for selecting a binary split will find the optimum split to be the one that gives the highest information gain. In this case, a clean split can be made between the blue points and the other points, giving a greater information gain to an incorrect (as we would see it intuitively) split, especially as instances of class 2 are slightly more numerous in the training data.

The optimum splitting point for shapelet 2 (excluded to avoid redundancy) also separates the blue points from the other points. Hence, from a starting point of having three shapelets after clustering, we are left with three binary shapelets that all split the data in the same way, which results in three attributes that are perfectly correlated. Not only is the incorrect to visual inspection, it also destroys information, as all three shapelets become capable only of distinguishing class 2 from classes 1 and 3. This greatly decreases the predictive power of the shapelets. We address this problem in the next section.

5.8.2 Class transform

To mitigate this weakness we transform the class labels prior to searching for the optimum split point. This process is detailed in Algorithm 12.

For each shapelet, Algorithm 12 uses the training set to estimate which class the

Algorithm 12 `binaryTransformClass`($\mathbf{D}$)

```
1: $listOfSplits \leftarrow <>$
2: for all Shapelets $S$ in $\mathbf{D}$ do
3:    $bestRank \leftarrow \infty$
4:    $bestClass \leftarrow \emptyset$
5:    $\mathbf{D}_{copy} \leftarrow \texttt{sort}(\mathbf{D}, S)$
6:    for all Classes $C$ in $\mathbf{D}$ do
7:       $rank \leftarrow \texttt{rank\_avg\_test}(\mathbf{D}, S, C)$
8:       if $rank < bestRank$ then
9:          $bestRank \leftarrow rank$
10:         $bestClass \leftarrow C$
11:   for all Instances $I$ in $\mathbf{D}_{copy}$ do
12:      if $I.\texttt{class} \leftarrow bestClass$ then
13:         $I.\texttt{class} \leftarrow 0$
14:      else
15:         $I.\texttt{class} \leftarrow 1$
16:   $bestGain \leftarrow -\infty$
17:   $bestSplit \leftarrow \texttt{null}$
18:   for $i \leftarrow 2$ to $|\mathbf{D}_s|$ do
19:      $split \leftarrow (\mathbf{D}_{s,i} + \mathbf{D}_{s,i-1})/2$
20:      $gain \leftarrow \texttt{calculateGain}(\mathbf{D}_s, split)$
21:      if $gain > bestGain$ then
22:         $bestGain \leftarrow gain$
23:         $bestSplit \leftarrow split$
24:   $listOfSplits.\texttt{add}(bestSplit)$
25: return $listOfSplits$
```

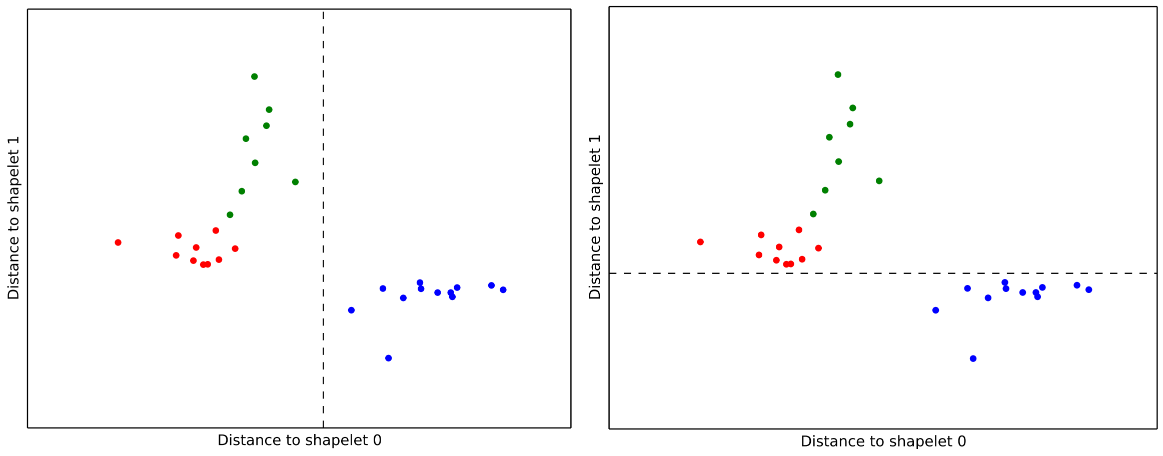

Figure 5.19: Data from the CBF training set, distributed by *sDist* from shapelets 0 and 1. The different coloured points represent the three classes of data; the dotted line shows the optimum splitting point by IG. The left graph shows the optimum splitting point for shapelet 0. The right graph shows the optimum splitting point for shapelet 1.

shapelet discriminates using a *rank average test*. This is performed as follows. The class labels of the training set are sorted by the distance between the corresponding instance and the shapelet, from smallest to largest. The average position of each set of class labels is calculated, and the class label with the lowest average rank is taken to be the class label that particular shapelet discriminates. Instances of that class have their label transformed into 0; all other instances have their class label transformed into 1 (this is only for the purposes of finding the appropriate splitting point).

The intuition behind transforming the data in this way prior to finding the split point is that shapelets discriminate one class from the other classes. They do not necessarily make a multi-class split. By emphasising the target class and pooling the data in the non-target classes, we should find a split that better reflects the way the shapelet divides the data.

Referring back to the graphical example given in Section 5.8.1, on the CBF dataset, finding the split points using the naive method results in each shapelet having the same splitting point relative to the data, as the (marginally) larger class 2 is split from classes 1 and 3. By performing the class transform, we find that each shapelet now splits the data in a way that is both more congruent with our intuitions about the data, and that gives us three different splits. The optimum splitting points for shapelets 0 and 1, found using the class transform method, are shown in Fig. 5.20. In each of the two graphs, the instances identified as being of the target class by the class transform algorithm are shown in red, while all other points are shown in blue.

As can be seen from Fig. 5.20, the split for shapelet 1 (Fig. 5.20 right) is the same as that found using the naive method. The split for shapelet 0, however, now matches where we would place the split intuitively, as it separates the red points (those of class 1) from all but one of the points of other classes (we exclude the graph for shapelet 2; it shows that the splitting point is different to that found by the standard method, and distinguishes the instances of class 3).

The class transform algorithm allows us to find optimum splitting points that more closely match our intuitions, and that take advantage of the innate structure of the shapelet space (i.e. that shapelets distinguish one class only).

We compare the two methods of transforming the data against one another in Section 5.9. Before we test them, we use a simple filter to remove attributes that are highly correlated after being transformed into binary data. This filter is described in the next section.

5.8.3 Correlation filtering

After performing the binary transform, we use a simple filter to remove shapelets (i.e. attributes) that are entirely positively or negatively correlated. Such correlations occur because the binary transform can smooth out differences between shapelets. Correlated attributes can result in less accurate classification, depending on the type

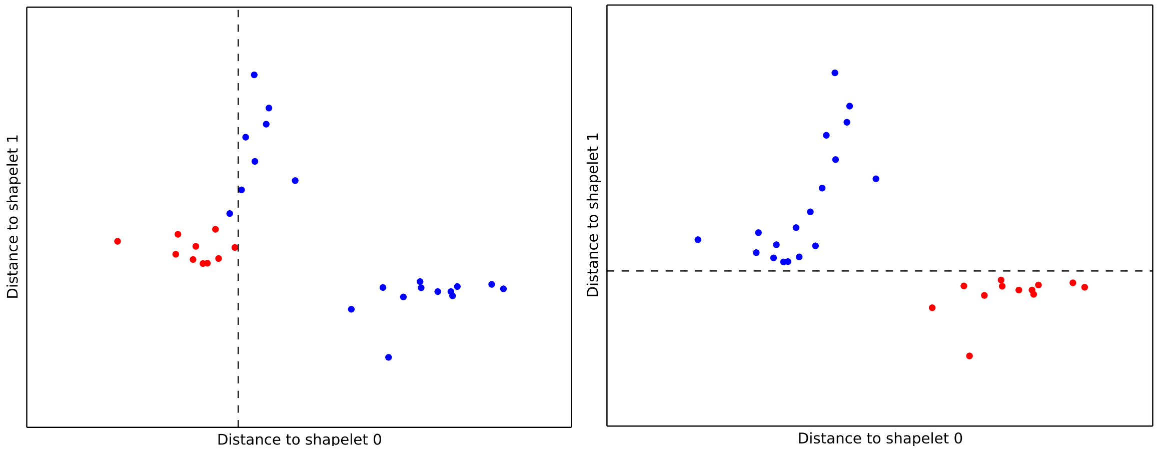

Figure 5.20: Data from the CBF training set, distributed by *sDist* from shapelets 0 and 1. The different coloured points represent the newly transformed two classes of data. The dotted line shows the optimum splitting point by IG, using the class transform algorithm. The left graph shows the optimum splitting point for shapelet 0 (points of class 1 are shown in red, points not of class 1 in blue). The right graph shows the optimum splitting point for shapelet 1 (points of class 2 are shown in red, points not of class 2 in blue).

of classifier. The process used to filter binary-shapelet-transformed data is shown in Algorithm 13.

Algorithm 13 `simpleCorrelationFilter`($\mathbf{S}$,$\mathbf{D}$)

```
1: for $i \leftarrow 1$ to $|\mathbf{S}|$ do
2:     for $j \leftarrow i+1$ to $|\mathbf{S}|$ do
3:         if checkCorrelation($i, j, \mathbf{D}$) then
4:             $\mathbf{S} \leftarrow \mathbf{S} - \mathbf{S_j}$
5:             $j - -$
6: return $\mathbf{S}$
```

We remove only those attributes that have an entirely positive correlation (1) or entirely negative correlation (-1). Such attributes add no information, and can lower classification accuracy. Once the training set has been transformed, our filtering algorithm searches the attributes in order, removing attributes that are entirely correlated with an earlier attribute. For many of the datasets we use, the training set is much smaller than the test set. This suggests that it could compromise accuracy if

Algorithm 14 `checkCorrelation`(i,j,$\mathbf{D}$)

```
1: $positive \leftarrow$ FALSE
2: $negative \leftarrow$ FALSE
3: for all Records $\mathbf{r}$ in $\mathbf{D}$ do
4:     if $\mathbf{r}_i = \mathbf{r}_j$ then
5:         $positive \leftarrow$ TRUE
6:     else
7:         $negative \leftarrow$ TRUE
8:     if $positive \wedge negative$ then
9:         return FALSE
10: return TRUE
```

we removed attributes that have any correlation other than 1 or -1. A difference in a single record in the training set could be much more prevalent in the test set; hence, we restrict correlation filtering to perfect positive and negative correlations.

Table 8.7 (see appendix) shows the number of shapelets for each dataset and each transform.

5.9 Binary results

For the first two experiments, we use the same fifty datasets we use for the clustering experiments. For the remaining experiments in this chapter, we use the same datasets used in the initial testing of the shapelet transform (Section 5.3).

5.9.1 Comparison of standard binary transform to binary class transform

Our first experiment is a pairwise comparison of the accuracy of the ensemble classifier on binary shapelet data transformed with either the standard binary transform or the binary class transform. Our alternative hypothesis is that the ensemble's accuracy will be higher on the class-transformed data. We test the hypothesis using a Wilcoxon Signed Rank test at a significance level of 0.01 on the accuracies of the ensemble over fifty datasets.

The results of the test show that the ensemble is significantly more accurate on the class-transformed binary data than on the standard binary data. The p value is 0.00828. We proceed using the class-transformed data. Figure 5.21 displays the results graphically.

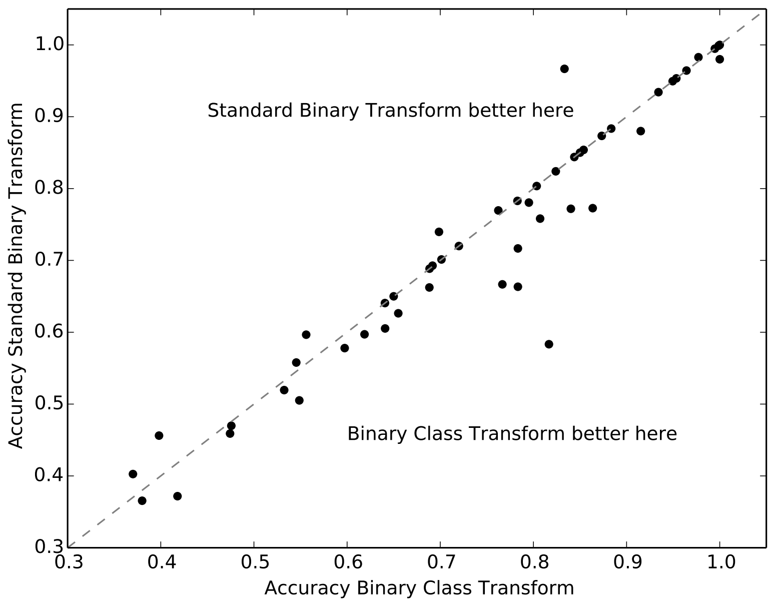

Figure 5.21: Comparison of accuracy over 50 datasets between ensemble classifier on binary-class-transformed shapelet data and standard binary-transformed data. The ensemble on binary-class-transformed shapelet data wins 21 times, the ensemble on standard binary-shapelet data 9 times.

5.9.2 Comparison to clustered data

Our next experiment compares the ensemble's performance on the class-transformed binary data to its performance on the MDLStopCE clustered data. We use a Wilcoxon Signed Rank test at a significance level of 0.01 on the accuracies of the ensemble over fifty datasets.

The p-value is 0.00272; the ensemble's accuracy is significantly worse on the binary data than on the clustered data.

5.9.3 Analysis

Examining the differences in the ensemble's accuracy on the two types of data reveals that for 40 of the 50 datasets, the difference in accuracy is 0.1 or less. We summarise these results in Figure 5.22. For these datasets, the differences in accuracy are relatively small; the ensemble appears to be slightly more accurate on the clustered data, but the effect is not a strong one.

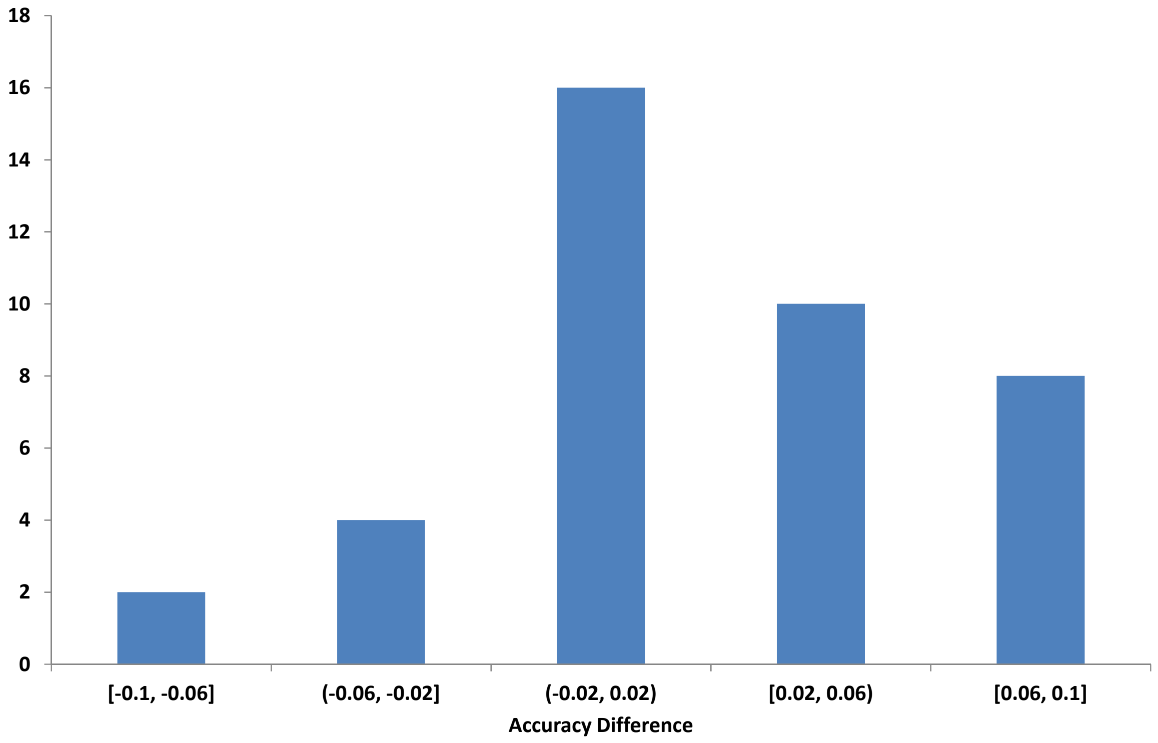

Figure 5.22: Difference in accuracy of the ensemble classifier on forty datasets before and after transformation using the binary-class transform. Positive differences indicate that accuracy was higher before transformation, negative differences that it was higher after transformation.

The problem for the binary data arises with the remaining ten datasets, where the binary transform has caused a large reduction in the ensemble's accuracy. This suggests that there are certain problems for which the binary transform is unsuitable. We believe there are three main reasons why the binary transform may be unsuitable for a dataset.

First, it may be the case that the binary transform smooths away some important differences, destroying information. This can be detected using the correlation filter; if a large number of shapelets are being removed, it suggests that information that was preserved over clustering is being destroyed. An example of this is the Bird/Chicken dataset, which loses over forty percent of its shapelets in the correlation filter (note that the information is lost over the binary transform, not during correlation filtering - the filter merely removes redundant shapelets).

Second, if the training set is very much smaller than test set, then the small amount of information lost over the binary transform may scale up to a large loss of accuracy on the test set. The CBF training set is 30 instances; the test set is 900 instances. By sacrificing a small amount of information to increase interpretability, we greatly reduce accuracy on the much larger test set.

The third reason that the binary transform may be unsuitable for a dataset is if there are a large number of classes. There is a strong correlation between the datasets where the accuracy loss has been large and those with a large number of classes (Figure 5.23). Out of fifty datasets, ten suffer from severe accuracy loss after binary transformation. Seven of these ten are members of the ten datasets with the highest number of classes. The binary transform is not unsuitable for all problems with numerous classes: three of the top ten do not show large accuracy losses, including the datasets with the first and third highest number of classes (fiftywords and SwedishLeaf). There is a correlation, however; in Chapter 6, we explore nugget discovery as a way to improve accuracy on problems with poorly predicted classes. We believe that the binary transform can reduce the accuracy of the ensemble in cases with many classes because it destroys information that can be used by the classifiers to distinguish more than one class from a single shapelet. The greater the number of classes in a dataset, the more likely it is that the shapelet distances will interact in complex ways to identify the different classes. When we perform the binary transform, we destroy any multi-class features that may have been present in the shapelet distances. The facility to make use of such information is a strength of the shapelet transform; care is needed when deciding whether to use the binary transform on problems with many classes.

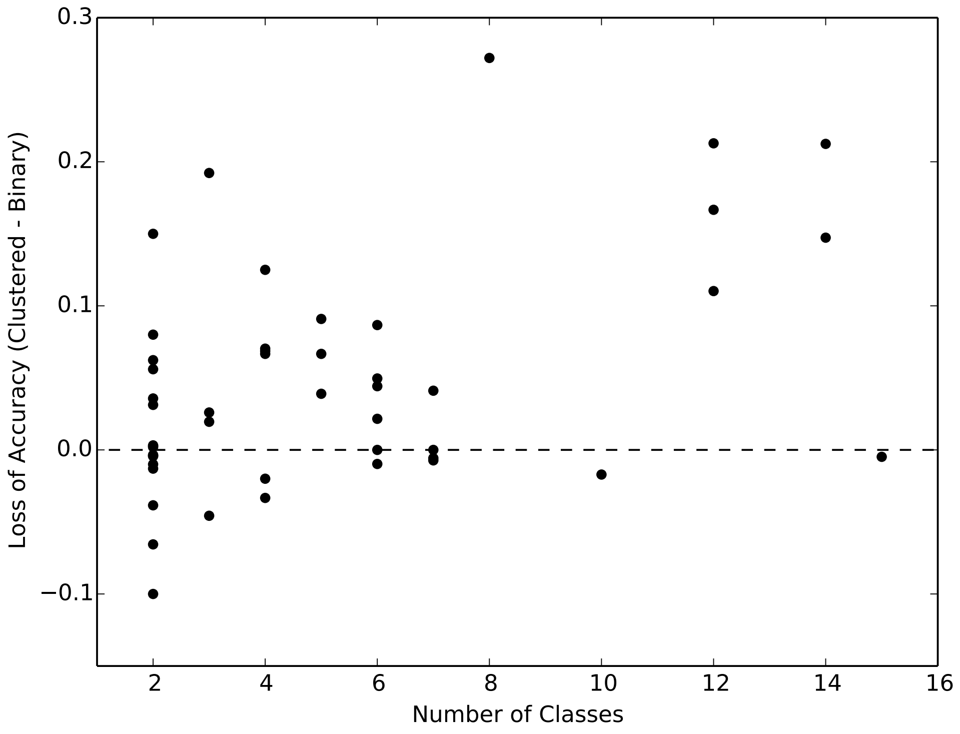

Figure 5.23: Loss of accuracy from binary transformation by number of classes. Two points are excluded from the graph to aid visual comprehension: (25,0.155) and (50,0.0527).

5.9.4 Comparison to 1NNDTW

1NNDTW is the benchmark for TSC; as such we compare the accuracy of the ensemble on binary-class-transformed data to that of 1NNDTW on the raw data over 75 datasets. We use a Wilcoxon Signed Rank test at a significance level of 0.01. According to the test, there is no significant difference in accuracy between the two classification methods. This result is shown graphically in Figure 5.24.

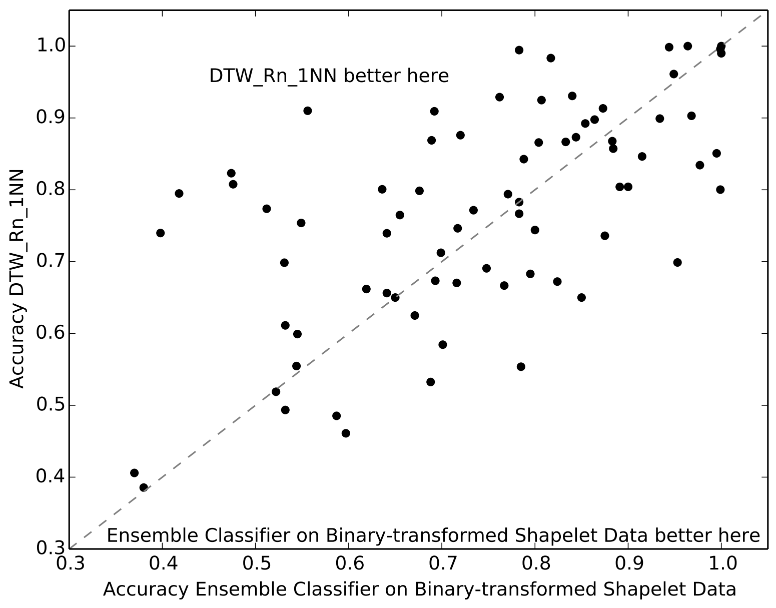

Figure 5.24: Comparison of accuracy over 75 datasets between ensemble classifier on binary-class-transformed shapelet data and 1NNDTW. 1NNDTW wins 41 times, the ensemble classifier 31 times.

Discretising the shapelet-transformed data reduces classification accuracy, but the ensemble is still as accurate as the benchmark method, as well as providing highly interpretable results.

5.9.5 Comparison to Logical Shapelets

We compare the accuracy of the ensemble on binary-class-transformed data to the the accuracy of the Logical Shapelets algorithm over 31 datasets using a Wilcoxon Signed Rank test at a significance level of 0.01. The ensemble on binary-class-transformed data is significantly more accurate than the Logical Shapelets algorithm. The p value is 6.60×10^{-3}.

Despite the loss of accuracy that occurs when the shapelet data is discretised, the ensemble is still more accurate than the Logical Shapelets algorithm. This result is shown graphically in Figure 5.25.

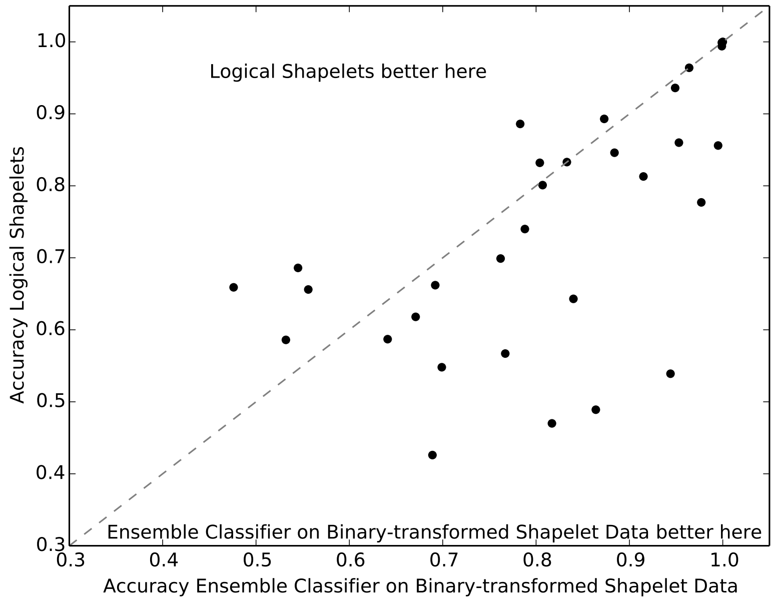

Figure 5.25: Comparison of accuracy over 31 datasets between ensemble classifier on binary-class-transformed shapelet data and Logical Shapelets. The ensemble wins on 22 datasets, Logical shapelets on 8.

5.9.6 Comparison to Fast Shapelets

We compare the accuracy of the ensemble on binary-class-transformed data to the accuracy of the Fast Shapelets algorithm over 44 datasets. We test for significant difference using a Wilcoxon Signed Rank test at a significance level of 0.01. The test reveals that there is no significant difference in accuracy between the two methods.

5.9.7 Assessment

The appropriate transform depends on the goal of the data-mining project. If accuracy is the only concern, we recommend using the shapelet transform with filtered data. If a blend of accuracy and interpretability is required, MDLStopClustered data is the preferred option. If interpretability is the prime goal, we recommend using binary-class transformed data; the analyst must be careful when applying this to problems with very small training sets, with a very large number of classes (though this can be mitigated with nugget discovery, see Chapter 6), or where the correlation filter removes many shapelets. In these cases, you may lose a lot of classification accuracy; however, the approach is no less accurate than two other leading TSC algorithms, 1NNDTW and Fast Shapelets, and is more accurate than Logical Shapelets. The accuracy of the ensemble on binary data is low only when compared to its highly accurate performance on the clustered and unclustered shapelet-transformed data.

5.10 Conclusions

We have made several novel contributions to TSC. The shapelet transform uses shapelets in a way that allows for more accurate classification than previous shapelet-based algorithms, and which is significantly more accurate than the benchmark algorithm, 1NNDTW. We have proposed a novel method for parameterless clustering of shapelets, which uses MDL as a stopping criterion for hierarchical clustering based on *sDist*. The ensemble is no less accurate on this data, and many datasets show a substantial reduction in the number of duplicate shapelets, in some cases down to one shapelet per class, which makes them highly interpretable. Our third contribution maximises the interpretability of the shapelet-transformed data by using a bespoke algorithm to discretise the data into a series of binary features indicating the presence or absence of a given shapelet. We have identified the types of problems for which the binary data might not be suitable. The binary-shapelet-transformed data is highly interpretable, and can provide insight into the problem domain while still offering good accuracy on most datasets.

Other research groups have referenced the shapelet transform in recently published papers, including [75, 5, 83].

Chapter 6

Rule Induction from Binary Shapelets

The work presented in this chapter on the BruteSuppression algorithm is published in:

J. Hills, A. Bagnall, B. De La Iglesia, and G. Richards

BruteSuppression: a Size Reduction Method for Apriori Rule Sets

Journal of Intelligent Information Systems 40 (3), pages 431–454, Springer: 2013.

6.1 Introduction

The ensemble classifier on shapelet-transformed data is the most accurate way of using the shapelet approach. It is also competitive with the best approaches to TSC. One weakness, however, is that it can often perform poorly for certain classes on multi-class problems. This is not a weakness specific to our ensemble classifier on shapelet-transformed data, but one that afflicts classification in general. Our proposed solution is tailored to the shapelet approach, and involves using association rules to improve classification accuracy on poorly predicted classes.

We use *Apriori* [3], to discover association rules between shapelets that target a particular class of interest. This process is called *nugget discovery* or *partial classification* [43]. We focus on predicting classes on which the ensemble performs poorly. Association rules are highly interpretable, and can also predict the occurrence of a class of interest with significantly better accuracy (measured using F1) than the ensemble, in cases where the ensemble performs poorly on a particular class.

Rule sets can contain tens of thousands of rules. This makes the rule sets incomprehensible, and can also be detrimental to classification accuracy. We propose a novel algorithm, BruteSuppression, that reduces the size of rule sets by deleting overlapping rules (Section 6.3). We show that using BruteSuppression can greatly reduce the number of rules in a rule set (Section 6.3) without compromising predictive accuracy (Section 6.5), or removing potentially interesting rules (Section 6.7).

Apriori is exponentially complex in the number of shapelets, which necessitates some form of dimensionality reduction, even after the clustering described in Chapter 5. We test a variety of different methods to reduce the dimensionality of the data. Dimensionality reduction decreases the effectiveness of nugget discovery. This suggests that the number of shapelets found using MDLStopCE clustering is approximately correct for that data, as reducing the size substantially reduces accuracy.

We conclude (Section 6.8) that nugget discovery significantly increases accuracy for shapelet data with twenty or fewer shapelets, for classes where the ensemble has performed very poorly. It can be used for higher-dimensional data, but the accuracy improvement is dependent on the dataset; it is best used where the accuracy of the ensemble is very low. It can also be used for exploratory data analysis, as the suppressed rule sets are generally small enough to be comprehensible.

6.2 Nugget discovery

The ensemble classifier performs well on the various shapelet-transformed datasets. One weakness of the approach can be seen by examining the confusion matrices for some multi-class problems. For example, consider Table 6.1, which shows the confusion matrix for the ensemble classifier on the Cricket_Y dataset. For some classes, the count of instances for which predicted = actual (the diagonal, shown in bold) approaches the total number of instances of that class, for example class 1, where 26 of the 31 instances are predicted correctly. The performance for some other classes is much worse; for example class 5, where only 4 of the 33 instances are predicted correctly. Situations like this are fairly common for multi-class classification.

This is potentially a problem for data-mining tasks that focus on one particular class of interest, or where we require accurate prediction of all instances. We mitigate this problem using rule-based nugget discovery. Nugget discovery is the discovery of classification rules that apply to a single target class [43].

6.2.1 Motivation

Nugget discovery involves mining association rules that concern one particular class of interest, then using those rules to classify the test data. Nugget discovery is particularly suited for shapelet-transformed data for two reasons. First, nugget discovery

Table 6.1: Confusion matrix for the ensemble classifier on Cricket_Y dataset. The diagonal values (in bold) show the counts for each class where the predicted class matches the actual class.

	Predicted Class												
Actual Class	0	1	2	3	4	5	6	7	8	9	10	11	Total
0	**14**	0	0	0	0	0	0	4	13	1	1	4	37
1	0	**26**	1	0	0	1	0	0	0	0	0	3	31
2	0	7	**15**	0	0	0	0	0	0	0	0	7	29
3	0	0	0	**27**	1	0	2	1	1	1	2	0	35
4	0	0	0	0	**20**	0	2	0	0	0	0	7	29
5	0	1	0	0	0	**4**	2	3	0	0	1	22	33
6	0	0	0	18	0	0	**7**	2	0	6	0	0	33
7	0	0	0	13	0	1	2	**8**	4	3	1	0	32
8	9	0	0	1	0	0	0	1	**11**	3	1	8	34
9	0	0	0	24	0	0	0	2	2	**4**	1	0	33
10	2	2	1	0	0	1	4	5	6	2	**7**	3	33
11	1	1	2	0	0	4	1	1	1	0	0	**20**	31
Total	26	37	19	83	21	11	20	27	38	20	14	74	390

allows us to use the existing shapelets to target the class of interest, as shapelets are class specific. Rule discovery in time-series data (e.g. [39]) generally requires that the time series be represented in terms of their subsequences (the success of this approach, particularly with regard to a failure to prevent trivial matching, is discussed in [100]). The shapelet transform already represents time series in terms of subsequences, negating any need for pre-processing before rule induction.

The second reason that nugget discover is suited to shapelet-transformed data is interpretability. Rule-based approaches are interpretable, which coheres well with the shapelet approach, and with our highly interpretable shapelet-transformed binary data. If...then rules that operate on binary shapelet data are highly comprehensible to non-experts.

An alternative approach to improve performance on a given class would involve transforming the data into a two-class problem where the instances of the class of interest are distinguished from the other instances. This approach has a number of weaknesses. First, it requires the shapelet transform to be applied for each class of interest. To ensure that each class in a multi-class problem is classified accurately, we may have to perform the transform many times, greatly increasing the time complexity of the classification process. This effect is magnified if we use the ensemble, as each new problem requires all eight classifiers to be trained, and cross-validation accuracies to be obtained. The time overhead for this may be feasible in cases where we know in advance that we are only interested in one class, but if we aim to improve accuracy on poorly predicted classes post classification, it is much more time efficient to use rule-based nugget discovery with the existing shapelets. The second weakness is that minority classes may *still* be predicted poorly by the ensemble classifier, especially very small minority classes. Such classes are common among those that are difficult to predict.

6.2.2 $F1$

We calculate classification performance for individual classes using the $F1$ measure.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}, \tag{6.2.1}$$

where

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{6.2.2}$$

$$Recall = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6.2.3}$$

		Actual	
		0	1
Predicted	0	TP	FP
	1	FN	TN

Figure 6.1: Confusion matrix for class 0; TP, FP, FN and TN stand for true positive, false positive, false negative, and true negative respectively.

Precision and recall are calculated via a confusion matrix (Figure 6.1). We assign class 0 to our class of interest, and class 1 to all other classes, and sum the predictions that fall in each of the four sections of the matrix. Where the number of true positives is zero, we take $F1$ to be zero.

We use $F1$ rather than accuracy because it can provide a nuanced assessment of the performance of a classifier on a single class. $F1$ takes into account false positives as well as false negatives. False positives can be a serious problem for certain tasks, especially if only one class is of interest.

6.2.3 Data

Table 6.2 shows the datasets and classes we use for nugget discovery. We select those classes for which the highly accurate ensemble classifier on shapelet-transformed data performs relatively poorly. We calculate the $F1$ score of the ensemble for each class, and then select the classes where $F1$ is low relative to the best $F1$ of the ensemble on any class of that dataset, and to the overall $F1$ of the ensemble on that dataset. Two of the classes we select, ToeSegmentation2 class 1 and MiddlePhalanxOutlineAgeGroup class 2, are not minority classes. The ensemble performs poorly on these classes, however, so we include them in our experiments.

We recognise that our selection is biased; however, we consider it to be a more accurate representation of how nugget discovery should be used: as a support routine alongside the ensemble classifier. For completeness, we also examine the performance of nugget discovery over all classes of the lower-dimensionality data.

6.2.4 Rule induction approach

Our nuggest-discovery approach uses the Apriori association rule algorithm [2]. We generate rule sets with a minimum confidence equal to the incidence of the class of interest in the training data, a minimum support of one record, and a fixed consequent

Table 6.2: Datasets and classes used for nugget discovery with F1 score of ensemble for that class, and the proportion of records of that class in the dataset.

Dataset	Target Class	F1	Proportion in training set
ToeSegmentation2	1	0.537	0.500
WormsTwoClass	0	0.531	0.420
CBF	2	0.530	0.267
FaceFour	2	0.700	0.125
OSULeaf	5	0.077	0.075
SyntheticControl	2	0.622	0.167
MALLAT	7	0.000	0.127
MALLAT	0	0.394	0.109
MALLAT	2	0.417	0.109
UWaveGestureLibrary_X	5	0.050	0.124
UWaveGestureLibrary_X	4	0.154	0.142
UWaveGestureLibrary_X	0	0.255	0.136
Cricket_Y	9	0.151	0.082
Cricket_Y	5	0.182	0.082
Cricket_Y	6	0.264	0.082
Cricket_Z	5	0.057	0.092
Cricket_Z	11	0.342	0.092
Cricket_Z	9	0.371	0.079
Cricket_X	0	0.069	0.097
Cricket_X	5	0.146	0.087
Cricket_X	11	0.146	0.082
FacesUCR	10	0.204	0.020
FacesUCR	3	0.454	0.110
FaceAll	5	0.271	0.071
FaceAll	10	0.273	0.071
FaceAll	12	0.227	0.071
UWaveGestureLibrary_Y	5	0.241	0.124
UWaveGestureLibrary_Y	3	0.326	0.123
CinC_ECG_torso	0	0.604	0.125
Symbols	3	0.727	0.120
OliveOil	2	0.667	0.133
UWaveGestureLibrary_Z	3	0.380	0.123
Beef	2	0.667	0.200
Beef	3	0.667	0.200
Worms	3	0.333	0.177
Lightning7	0	0.571	0.114
PtNDevices	10	0.178	0.071
PtNDevices	9	0.187	0.071
Herrings	0	0.465	0.391
Car	3	0.667	0.283
PtNDeviceGroups	0	0.489	0.143
DistalPhalanxTW	1	0.154	0.070
DistalPhalanxTW	4	0.182	0.045
InlineSkate	6	0.203	0.110
RefrigerationDevices	2	0.486	0.333
Haptics	0	0.268	0.116
MiddlePhalanxOutlineAgeGroup	2	0.389	0.593
ProximalPhalanxTW	0	0.000	0.040
ProximalPhalanxTW	4	0.000	0.043

Table 6.2: Datasets and classes used for nugget discovery with F1 score of ensemble for that class, and the proportion of records of that class in the dataset.

Dataset	Target Class	F1	Proportion in training set
DistalPhalanxOutlineAgeGroup	0	0.200	0.075
MiddlePhalanxTW	4	0.000	0.063
MiddlePhalanxTW	3	0.167	0.075
ProximalPhalanxOutlineAgeGroup	0	0.500	0.180
MedicalImages	7	0.000	0.016
MedicalImages	3	0.057	0.042
Earthquakes	1	0.054	0.180
SwedishLeaf	0	0.681	0.058
ChlorineConcentration	1	0.420	0.195
PhalangesOutlinesCorrect	0	0.604	0.349
WordSynonyms	6	0.000	0.015
WordSynonyms	10	0.000	0.022
WordSynonyms	14	0.000	0.015
WordSynonyms	19	0.000	0.026
NonInvasiveFatalECG_Thorax1	2	0.474	0.025
NonInvasiveFatalECG_Thorax2	16	0.333	0.029
Adiac	30	0.000	0.023
Adiac	21	0.000	0.026
Adiac	4	0.000	0.010
fiftywords	49	0.000	0.004
fiftywords	48	0.000	0.004
fiftywords	47	0.000	0.011
fiftywords	43	0.000	0.013
fiftywords	42	0.000	0.009
fiftywords	41	0.000	0.004
fiftywords	40	0.000	0.002
fiftywords	34	0.000	0.013
fiftywords	29	0.000	0.013
fiftywords	24	0.000	0.004
fiftywords	20	0.000	0.016
fiftywords	16	0.000	0.016

of the class of interest. We assess the performance of the rule sets in terms of $F1$. A rule set has a number of different rules that have been found on the training set, and a target class, which is the class of interest.

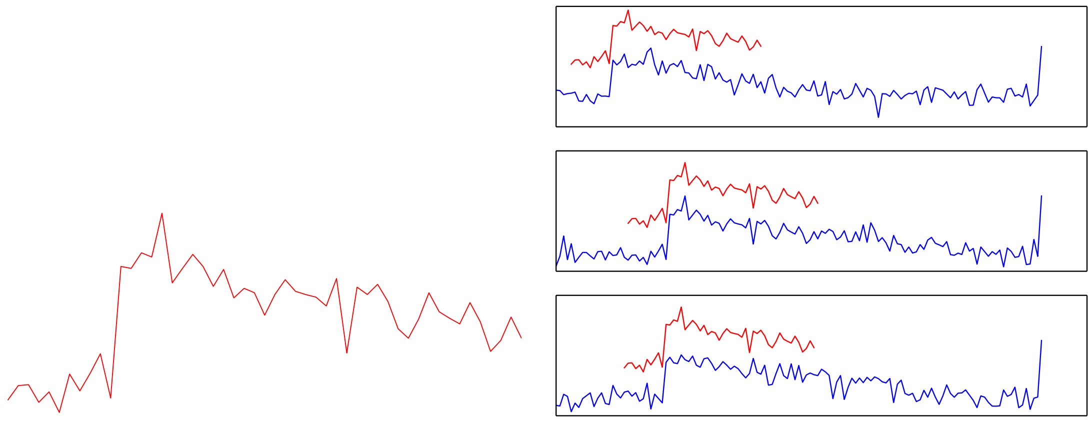

Figure 6.2: Left: $Shapelet_2$ from the CBF clustered shapelet set. Right: $Shapelet_2$ superimposed (offset on the y-axis) over three series of class 3.

As an example, the following rule is found by using Apriori on the CBF dataset, with class 3 as the target class: `IF` $Shapelet_2 = \{$`True`$\}$ `THEN` $Class = \{3\}$. Figure 6.2 shows $Shapelet_2$ alongside three series from CBF of class 3. As indicated by the rule above, the presence of $Shapelet_2$ is a predictor of membership in class 3.

6.2.5 Rule set size

A major weakness of Apriori is that it generates rule sets that may contain hundreds, thousands, or even tens of thousands of rules. A rule set of this size is difficult to interpret, and interpretability is a key feature of our approach. There can also be significant overlap between rules, which may compromise classification accuracy for a large rule set. It is important, therefore, to minimise the size of the rule sets where it can be done without a deterioration in performance.

The standard approach to reducing association rule set size (for example [3]) involves eliminating rules with parameter values (such as support and confidence) that are below certain thresholds. This form of rule set size reduction has two major drawbacks. First, deleting rules purely because, for example, they do not cover enough instances, can mean that interesting rules describing niches in the space of possible cases are removed. The second disadvantage is that the structure of the rule set is highly dependent on the parameter thresholds, and the user has no *a priori* guidance on which values to choose. Recent research has focused on finding other methods to reduce rule set size, either by adjusting the rule induction algorithm [156, 177], incorporating new interestingness measures [153], or by pruning the rule set [120]. We propose and test an algorithm, *BruteSuppression* (Section 6.3), for reducing rule set size by eliminating overlapping rules. By using BruteSuppression on our rule sets, we aim to increase interpretability and improve the performance of our nugget discovery.

6.2.6 Classification

Algorithm 15 `classifyByRuleSet`(Instance I, Rule set $\mathbf{R}$)

```
$fired \leftarrow 0$
$unfired \leftarrow 0$
**for all** rules $R$ in $\mathbf{R}$ **do**
  **if** $R$.fires($I$) **then**
    $fired \leftarrow fired + R$.conf
  **else**
    $unfired \leftarrow unfired + R$.conf
**if** $fired > unfired$ **then**
  **return** True
**else**
  **return** False
```

Algorithm 15 shows the procedure by which we classify an instance using a rule set. For each rule in the rule set, we test whether the rule fires for that instance. A rule, R, fires for an instance, I, if the antecedent conditions of the rule are satisfied by the instance, that is, if the conjunction of attribute tests in the antecedent of R is true of the instance I.

If R fires for I, the confidence of the rule is added to the variable $fired$. If R does not fire, the confidence is added to $unfired$. If the value of $fired$ is greater than the value of $unfired$, the algorithm returns true, indicating that the instance is of the class that the rule set targets. Otherwise, it returns false, indicating that the instance is not of the target class.

6.3 The BruteSuppression algorithm

We propose a novel algorithm for reducing the size of rule sets by deleting rules that overlap with other rules. The intuition is that if two rules are very similar in the cases they cover, then it is safe to delete one of them. In contrast, a rule that uniquely covers a set of cases is worth retaining even if it has lower support and confidence than other rules in the rule set.

We assess the overlap between rules using a measure based on *suppression* [68]. We construct the reduced rule set through a specifically tailored comparative enumeration of the rules in the original set.

We evaluate our algorithm in three stages. First, we show that BruteSuppression substantially reduces rule set size for rule sets generated on a wide variety of problems and with a wide range of sizes. Second, we apply the algorithm to the rule sets we use for nugget discovery, and compare the performance of the suppressed sets to that of the original sets (Section 6.6). Finally, to demonstrate that the approach is applicable to a variety of cases, we use qualitative analysis to show that using BruteSuppression to reduce rule set size yields a rule set much more similar to the original, larger rule set than if the confidence or support parameter settings are used.

6.3.1 Redundancy of rules

Our algorithm requires a measure to assess the redundancy of a rule. Trivially, if $A \Rightarrow C$ and $A \wedge B \Rightarrow C$ have the same confidence, the second rule is redundant with relation to confidence, as the extra AT adds no predictive power to the rule. Equally (see [10]), the rule $A \wedge B \Rightarrow C$ is redundant with respect to rule $A \Rightarrow B \wedge C$ and also rule $A \wedge B \Rightarrow C \wedge D$ (assuming equivalent levels of confidence). Apriori allows the user to set minimum support and minimum confidence thresholds; rules that fall below these thresholds do not appear in the rule set. This form of rule set size reduction does not take into account any information about the records that the rules cover, only the support and confidence counts for each rule. Hence, rules covering a unique set of records may be eliminated, while rules that cover very similar sets of records may be retained.

Here, we take redundant rules to be those rules that overlap to a large degree, in terms of the records they cover, with rules of greater confidence. That is, if two rules cover the same records (to some specified degree), then the rule with lower confidence is redundant. This is an instance-based form of redundancy, in contrast to the count-based method used to eliminate rules in Apriori. There are a number of measures proposed in the literature to calculate this form of redundancy; see, for example, [68, 37, 147]. An intuitive measure of the overlap of two rules (R and Q) in terms of the records they cover (D_R and D_Q) is:

$$O(R,Q) = \frac{|D_R \cap D_Q|}{|D_R \cup D_Q|}. \tag{6.3.1}$$

This represents the size of the intersection of the two sets of records divided by the size of the union of the two sets.

The suppression function [68] calculates whether one rule is redundant relative to another rule. Rule R suppresses rule Q if:

$$V(Q) < (1+\epsilon) \times [S(R,Q)] \times V(R). \tag{6.3.2}$$

The function requires some measure of rule interestingness (denoted V), a parameter for determining the intensity of the suppression (denoted as ϵ), and some affinity function (denoted $S(R,Q)$) to measure the similarity of the rules. We use 0.1 for ϵ (this is the most intense suppression recommended in [68]), and $O(R,Q)$ as our affinity function, as we wish to measure similarity in terms of overlapping coverage of records. We use confidence for V, as it is the standard measure of the quality of a rule (see Section 2.10). Our suppression function is as follows. Rule R suppresses rule Q if:

$$Conf(Q) < (1+0.1) \times \frac{|D_R \cap D_Q|}{|D_R \cup D_Q|} \times Conf(R), \tag{6.3.3}$$

where $|D_R \cap D_Q|$ is the number of records covered by both rule R and rule Q, and $|D_R \cup D_Q|$ is the number of records covered by either rule R or rule Q.

The algorithm we use to apply the suppression function and indicate redundant rules is given in the next section. Although we test the algorithm on rule sets generated by Apriori, it can be adapted to other association rule mining algorithms such as *Dense Miner* ([14]) and *All Rules Algorithm* ([148]).

6.3.2 BruteSuppression

The BruteSuppression algorithm is shown below as Algorithm 2. The algorithm iterates through a rule set, testing pairs of rules with the suppression function and removing rules deemed to be redundant. Due to the nature of the suppression function, a given rule need only be checked against unsuppressed rules of higher confidence. Hence, the sooner a given rule is suppressed, the fewer total comparisons are required. It is important for efficiency to suppress redundant rules as early as possible. In the worst case, where no rules are suppressed, a rule set of n rules requires $\frac{n(n-1)}{2}$ comparisons (this is lower than the computational complexity of the Apriori algorithm itself). As rules are not tested against suppressed rules, however, in many cases far fewer comparisons are required than in the worst case. In practice, we may achieve a high level of suppression (e.g. 90+% of the rules are suppressed in most cases, see Table 6.3); we have implemented the algorithm to maximise the saving from suppressed rules. This is achieved as follows. The BruteSuppression algorithm iterates through a rule set ordered by confidence, beginning at rule r_2, and comparing each r_i against rule r_{i-1}, then r_{i-2}, and so on, until r_i is suppressed or has been compared to r_1. At this point the algorithm moves to the rule after the current rule. If r_i is suppressed, it will be removed from the rule set; the indexing updates accordingly.

Our implementation checks whether r_i is suppressed, rather than what r_i suppresses, on the following grounds. Empirically, we have observed that rules in Apriori rule sets are most often suppressed by the rules immediately preceding them in the confidence ordering, as overlapping rules tend to have very similar confidence values. It is more common for r_i to be suppressed by, say, r_{i-3}, than for it to be suppressed by r_{i-200}. If the overlapping rules in a rule set are distributed randomly, there is no benefit to any specific ordering. However, for the rule sets we have observed, there is a clear reduction in the number of comparisons if the rules immediately preceding a rule in the confidence ordering are the first rules to which that rule is compared.

Consider the following illustrative example. Assume we have a rule set with 50% suppression, where rule 1 suppresses rule 2, rule 3 suppresses rule 4, etc. If there are ten rules in the rule set, our approach requires 15 comparisons; testing the rules in descending order of confidence against r_i requires 25 comparisons (with no suppression, 45 comparisons are required). This reduction in the number of comparisons scales to larger rule sets where the overlapping rules cluster together, resulting in a substantial saving.

Algorithm 16 `bruteSuppression`($\mathbf{R}$, $\mathbf{D}$)

```
i ← 2 // Begin the process from the second rule
$\epsilon \leftarrow 0.1$
while $i \leq |\mathbf{R}|$ do
    $j \leftarrow i - 1$ // The first rule compared to rule i is the previous unsuppressed rule
    suppress ← FALSE
    while $j > 0$ & !suppress do
        if Confidence($r_j$)$\cdot\frac{|\mathbf{D}_{rj} \cap \mathbf{D}_{ri}|}{|\mathbf{D}_{rj} \cup \mathbf{D}_{ri}|} \cdot (1+\epsilon) \geq$Confidence($r_i$) then
            $\mathbf{R} \leftarrow \mathbf{R} - r_i$ // Rule i is removed from the rule set
            suppress ← TRUE
        $j - -$
    if !suppress then
        $i + +$ // The index is increased only if the previous rule was not suppressed
return R
```

6.4 Rule set size

We begin by assessing the effect of BruteSuppression on rule sets generated from binary-shapelet data. We do this in two ways. In this section, we assess how strongly BruteSuppression reduces rule set size. In Section 6.6, we test how well the suppressed sets have retained the predictive power of the original rule sets by measuring their performance at nugget discovery.

To assess how strongly BruteSuppression reduces rule set size for rule sets built on binary shapelet data, we suppress rule sets of a number of different sizes: 10 rules, 100 rules, 1,000 rules, and 10,000 rules. The results are shown in Table 6.3.

The results for datasets such as WormsTwoClass, ToeSegmentation2, and CBF do not vary as the maximum rule set size increases because Apriori cannot discover more than 10 rules regardless of the maximum size parameter. Hence, there is the same degree of suppression in each column. This is also the case for other datasets, for example FaceFour and OSULeaf, where the number of rules that can be generated

Table 6.3: Proportion of rules suppressed for each class of interest over variation in the maximum number of rules.

Dataset	Class	10 Rules	100 Rules	1,000 Rules	10,000 Rules
ToeSegmentation2	1	0.25	0.25	0.25	0.25
WormsTwoClass	0	0	0	0	0
CBF	2	0.4444	0.4444	0.4444	0.4444
FaceFour	2	0.7	0.4667	0.4667	0.4667
OSULeaf	5	0.1	0.59	0.4569	0.4569
SyntheticControl	2	0.9	0.8481	0.8481	0.8481
MALLAT	7	0.9	0.89	0.7618	0.7618
MALLAT	0	0.4	0.75	0.8195	0.8195
MALLAT	2	0.7	0.77	0.8517	0.8517
UWaveGestureLibrary_X	5	0	0.41	0.7117	0.7117
UWaveGestureLibrary_X	4	0.5	0.58	0.8676	0.8676
UWaveGestureLibrary_X	0	0	0	0.6747	0.6747
Cricket_Y	9	0.9	0.86	0.863	0.9853
Cricket_Y	5	0.7	0.7	0.573	0.887
Cricket_Y	6	0.7	0.84	0.854	0.9844
Cricket_Z	5	0.7	0.64	0.997	0.9736
Cricket_Z	11	0.7	0.82	0.77	0.9328
Cricket_Z	9	0.7	0.75	0.997	0.9542
Cricket_X	0	0.7	0.68	0.582	0.9857
Cricket_X	5	0.8	0.84	0.544	0.9195
Cricket_X	11	0.8	0.83	0.456	0.9654
FacesUCR	10	0.9	0.99	0.999	0.9619
FacesUCR	3	0.8	0.82	0.831	0.6246
FaceAll	5	0.5	0.71	0.707	0.6305
FaceAll	10	0.7	0.66	0.673	0.9955
FaceAll	12	0.9	0.86	0.779	0.9368
UWaveGestureLibrary_Y	5	0.6	0.99	0.998	0.9971
UWaveGestureLibrary_Y	3	0.9	0.94	0.926	0.9979
CinC_ECG_torso	0	0.9	0.99	0.999	0.9999
Symbols	3	0.9	0.99	0.999	0.9999
OliveOil	2	0.9	0.99	0.999	0.9999
UWaveGestureLibrary_Z	3	0	0.5	0.618	-
Beef	2	0.8	0.98	0.932	0.9999
Beef	3	0.9	0.97	0.977	0.9967
Worms	3	0.9	0.99	0.998	0.998
Lightning7	0	0.9	0.99	0.999	0.9996
PtNDevices	10	0.8	0.92	0.627	0.4652
PtNDevices	9	0.2	0.89	0.637	0.4889
Herrings	0	0.1	0.55	0.523	0.978
Car	3	0.9	0.99	0.999	0.9999

Table 6.3: Proportion of rules suppressed for each class of interest over variation in the maximum number of rules.

Dataset	Class	10 Rules	100 Rules	1,000 Rules	10,000 Rules
PtNDeviceGroups	0	0.4	0.53	0.571	0.4985
DistalPhalanxTW	1	0.9	0.99	0.999	0.9996
DistalPhalanxTW	4	0.9	0.99	0.999	0.9999
InlineSkate	6	0.7	0.96	0.964	0.9885
RefrigerationDevices	2	0.2	0.81	0.94	0.997
Haptics	0	0.9	0.99	0.999	0.9999
MiddlePhalanxOutlineAgeGroup	2	0.8	0.98	0.998	0.9998
ProximalPhalanxTW	0	0.9	0.99	0.999	0.9999
ProximalPhalanxTW	4	0.9	0.99	0.999	0.9999
DistalPhalanxOutlineAgeGroup	0	0.9	0.99	0.999	0.9999
MiddlePhalanxTW	4	0.8	0.94	0.994	0.9994
MiddlePhalanxTW	3	0.9	0.99	0.999	0.9999
ProximalPhalanxOutlineAgeGroup	0	0.9	0.99	0.994	0.996
MedicalImages	7	0.9	0.99	0.998	0.9998
MedicalImages	3	0.9	0.99	0.999	0.9999
Earthquakes	1	0	0	0.018	0.8008
SwedishLeaf	0	0.9	0.99	0.993	0.9747
ChlorineConcentration	1	0.9	0.99	0.995	0.9975
PhalangesOutlinesCorrect	0	0.1	0.99	0.996	0.999
WordSynonyms	6	0.9	0.99	0.999	0.9999
WordSynonyms	10	0.9	0.99	0.999	0.9999
WordSynonyms	14	0.9	0.99	0.999	0.9999
WordSynonyms	19	0.9	0.99	0.999	0.9999
NonInvasiveFatalECG_Thorax1	2	0.9	0.99	0.999	0.9999
NonInvasiveFatalECG_Thorax2	16	0.9	0.99	0.999	0.9997
Adiac	30	0.9	0.99	0.999	0.9999
Adiac	21	0.9	0.99	0.993	0.9993
Adiac	4	0.9	0.99	0.999	0.9999
fiftywords	49	0.9	0.99	0.999	0.9999
fiftywords	48	0.9	0.99	0.999	0.9999
fiftywords	47	0.9	0.99	0.999	0.9997
fiftywords	43	0.9	0.99	0.999	0.9999
fiftywords	42	0.9	0.99	0.999	0.9999
fiftywords	41	0.9	0.99	0.999	0.9999
fiftywords	40	0.9	0.99	0.999	0.9999
fiftywords	34	0.9	0.99	0.999	0.9999
fiftywords	29	0.9	0.99	0.999	0.9999
fiftywords	24	0.9	0.99	0.999	0.9999
fiftywords	20	0.9	0.99	0.998	0.9998
fiftywords	16	0.8	0.98	0.998	0.9998

is greater than ten but less than 100 for FaceFour, or 1,000 for OSULeaf, causing the suppression values to stabilise at that point. UWaveGestureLibrary_Z has no value where the maximum rule set size is 10,000 due to insufficient RAM. A particular characteristic of that dataset is that many, many rules can be generated when the parameter is set to a high value. We believe the missing value would be in line with the other results, as the proportion of suppressed rules increases with each increase in maximum rule set size.

The BruteSuppression algorithm generally performs consistently across different sizes of rule set. In most cases, it removes at least 90% of the rules in a rule set; for the larger sets, this reduction is much greater. We conclude that BruteSuppression is useful for reducing rule set size where rule sets have been found in binary-shapelet data.

6.4.1 Initial maximum rule set size

Before proceeding to analyse the effectiveness of nugget discovery, we must make a design decision: how many rules should be included in the original rule set. The issue is that Apriori discovers rules by lowering the support threshold until it reaches the maximum number of rules. The rules, however, are sorted by confidence, rather than support. Rules with higher support and lower confidence can be eliminated in large rule sets if lowering the support threshold creates many more rules than the maximum number. Hence, the larger rule sets tend to contain many rules with low support, whereas the smaller rule sets contain rules with higher support.

From our initial investigations, it appears that smaller rule sets with higher support rules are more predictive. This is not necessarily surprising; a rule with higher support is more likely to generalise than one with low support. The other problem for the larger rule sets is that they tend to make fewer positive predictions, simply because there are very many rules all picking out different specific niches. Hence, it is unlikely that any case will cause enough rules to fire to allow for a positive classification. For example, half of the rule sets containing 10,000 rules make no positive predictions for the 28 low-dimensionality datasets (see Section 6.5.1). In contrast, the rule sets with only ten rules make positive predictions for all 28 datasets (and provide very good accuracy, see Section 6.5.1).

The problems presented by the large rule sets are almost entirely mitigated by using BruteSuppression. For the large rule sets, the suppressed rule sets easily outperform the unsuppressed rule sets, as well as being very much smaller and far more interpretable. While it would be easy for us to use the larger rule sets to make a strong case for BruteSuppression, we feel this would be misleading, as someone using our method would simply reduce the rule set size until the sets performed well. Any comparison of the suppressed rule sets to the large rule sets would not reflect how the tools we have developed would be deployed. We use the best-performing rule sets, the sets with ten rules, for our experiments, though these are the rule sets for which BruteSuppression adds the least value. The suppressed sets tend to perform very similarly, regardless of the initial number of rules, and this way, we get a fair comparison against the original rule sets.

6.5 Performance of nugget discovery

We present the results for a number of different experiments, divided into two stages: first, testing the effectiveness of nugget discovery; second, testing the effect of using BruteSuppression to reduce rule set size.

We begin by comparing the performance of the rule set against the performance of the ensemble classifier on poorly predicted classes, for datasets where there are 20 or fewer shapelets (we refer to these datasets as *low-dimensionality* datasets). We then present results for all classes of these datasets.

Our next experiment compares three different methods of reducing dimensionality for problems with more than twenty shapelets. We compare truncation, clustering, and class-specific truncation, and conclude that, for poorly-predicted classes, there is no significant difference between the methods. We select truncation as it is simpler.

We compare the performance of the rule set on truncated data against that of the ensemble on poorly predicted classes, and conclude that there is no significant difference, though nugget discovery can offer substantial improvements in some cases.

Finally, we compare our nugget-discovery approach to the ensemble on poorly-predicted classes for datasets with more than twenty classes, using a modified form of truncation to reduce the dimensionality. We find that there is no significant difference.

For the second stage of our experiments, we compare the rule sets we have used for the previous experiments with rule sets that have been suppressed using Brute-Suppression. We compare the suppressed rule sets to the original rule sets on: low-dimensionality data poorly predicted classes, low-dimensionality data all classes, truncated data poorly predicted classes, and high-dimensionality data poorly predicted classes. We find no significant differences between the suppressed rule sets and the unsuppressed rule sets, and conclude that suppression can be used to reduce rule set size substantially without compromising performance.

6.5.1 Low-dimensionality experiments on poorly predicted classes

Our first experiment compares the performance of the rule set against that of the ensemble for those classes where the ensemble predicts poorly. This is the most obvious use of nugget discovery - it performs a supplemental role to the overall classifier in cases where the classifier performs poorly for some class of interest. We restrict our investigation to datasets with twenty or fewer shapelets in the binary-transformed, correlation-filtered datasets. We experiment with higher-dimensionality data in Sections 6.5.3 and 6.5.4.

We test the difference in $F1$ score between the ensemble and the rule set using a Wilcoxon Signed Rank test at a significance level of 0.01; there are 28 poorly predicted classes with 20 or fewer shapelets.

Our test shows that nugget discovery is significantly better than the ensemble, in terms of $F1$, for poorly predicted classes. The p value is 1.349×10^{-3}. Where one or more classes have lower $F1$ than the other classes with the ensemble classifier, and there are twenty or fewer shapelets, nugget discovery is significantly better than the ensemble. We illustrate this in Figure 6.3.

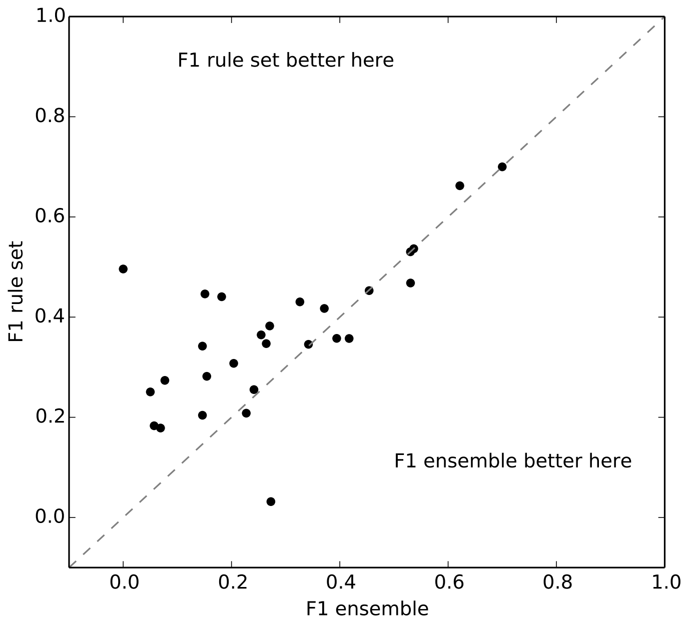

Figure 6.3: Comparison of $F1$ score between rule set and ensemble on poorly predicted classes. Points above the dotted line represent classes where the rule set outperforms the ensemble. Points below the dotted line represent classes where the ensemble outperforms the rule set. The rule set is better on 19 classes, the ensemble is better on 6.

6.5.2 Low-dimensionality experiments on all classes

Our second experiment examines the performance of nugget discovery over all classes of the low-dimensionality data, rather than just those on which the ensemble predicts poorly. Nugget discovery is generally used to improve predictive accuracy on specific classes where predictions have been poor, but for completeness, it is worthwhile to examine the general performance of the rules against the ensemble classifier.

We use a Wilcoxon Signed Rank test with a significance level of 0.01 to test the differences in $F1$ between the ensemble and the rule set over all classes of all datasets with 20 or fewer shapelets.

The test shows that there is no significant difference between the $F1$ scores of the ensemble classifier and the rule set, considered over every class of the low-dimensionality data. This is an interesting discovery; as the accuracy of the nugget discovery is not significantly worse than the ensemble, the rule sets discovered for any class may be useful, interpretable ways of understanding how to predict that class.

Figure 6.4 displays this result graphically.

The main problem with the Apriori-based nugget-discovery approach is that it is intractable for high-dimensionality data; the Apriori algorithm is exponentially complex in the attribute space. In the next section, we explore ways to reduce dimensionality to make the datasets with more than 20 shapelets tractable.

6.5.3 Medium-dimensionality experiments

For this set of experiments, we focus on poorly predicted classes for datasets with twenty or fewer classes but more than twenty shapelets. The reason for restricting our interest to datasets with twenty or fewer classes is the relationship between the number of shapelets and the number of classes. With twenty or fewer classes, we can retain at least one shapelet from each class, whereas we must lose shapelets

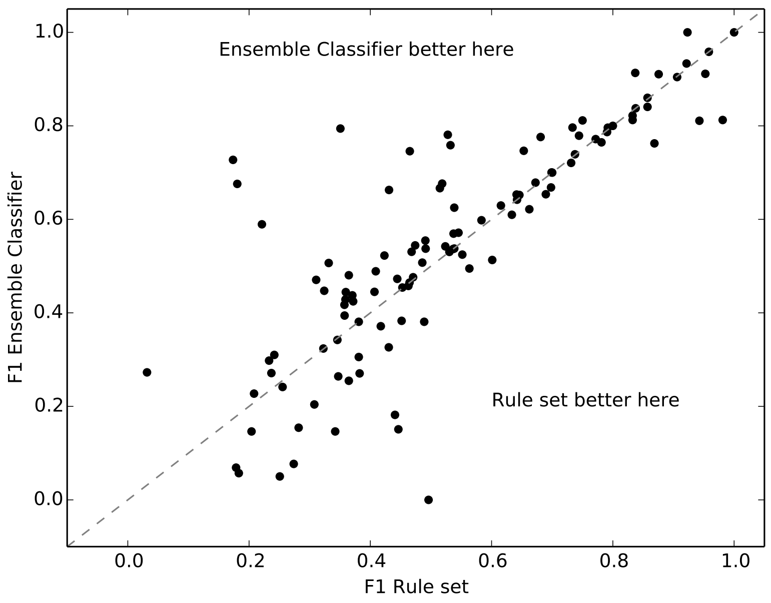

Figure 6.4: Comparison of $F1$ score between rule set and ensemble classifier on all classes of low-dimensionality data. The ensemble is better on 59 of the 100 classes, the rule set is better on 41.

representing some classes if there are more than twenty classes. We examine such cases in Section 6.5.4.

The traditional way to make Apriori tractable on larger datasets is to increase the minimum confidence and support constraints. We use a single record as the minimum support, and the base incidence rate of the class of interest in the training set as the minimum confidence. As shown in Section 6.7, using a higher minimum confidence and support alters the character of the rule set, and can eliminate rules that may be of interest, such as high confidence, low support exception rules. Such rules are particularly important for nugget discovery, as rules that target a minority class are very likely to be exception rules, simply because the number of records of that class is low. *Prima facie*, this method does not seem appropriate for our approach, as we are interested in classes that are difficult to predict. By using higher minimum support and confidence values, we may miss the rules we are looking for (see Section 6.7). Regardless, as a comparison method, we attempt to use the built-in constraints of

Apriori to create rule sets on higher-dimensional data by adjusting the parameter settings.

The first problem we encounter is that there appears to be no principled way to set a minimum support value, beyond trying a range of values. Different datasets have different itemsets for different classes; a support value that works for one class of a dataset may not work for another. Finding parameter settings that work is a time-consuming process, and one that may not result in the best rule set.

The second problem is that there is very little difference between parameter settings that will create an empty rule set, and settings that will deliver an explosion in time or space usage by the algorithm. Apriori was not designed to work with very high-dimensionality data, and small changes in the minimum support can have large effects on how the algorithm operates. For example, the DistalPhalanxTW dataset has 912 shapelets. If the minimum support is set to 19 records, no rules are generated. If the minimum support is decreased by one record, the smallest granularity possible, there is an explosion in the space requirement, and the software crashes due to inadequate RAM. This is the case even when the minimum confidence is set to 1. It may be possible to produce a rule set on the DistalPhalanxTW dataset by taking advantage of high-performance computing facilities with much greater quantities of RAM, but it seems likely that time would be a factor even with sufficient space. This is a consequence of the Apriori algorithm being exponentially complex in the attribute space.

Because of these problems, we do not make use of Apriori's built-in constraints to deal with high-dimensionality data. Instead, we experiment with three methods of reducing the dimensionality of the data: truncation, class-specific truncation, and clustering. For each of these methods, we reduce the dimensionality to 20 shapelets, a size that is tractable for every problem we use.

The first dimensionality-reduction method is based on truncating the shapelet data to the first 20 shapelets. The attributes in the binary shapelet data are ordered, with the first shapelet being the most discriminative. To create the truncated data, we first ensure that at least one shapelet from each class is included. Then we add those shapelets that are higher in the order until 20 shapelets have been included.

The second dimensionality-reduction method we test involves keeping only those shapelets that correspond to the class we are attempting to predict. Again, we restrict the data to 20 shapelets.

The final approach we try uses our existing clustering method to reduce the dimensionality of the data. The datasets are clustered using MDLStopCE clustering (see Section 5.6.2), a method that does not require any parameters. To reduce the dimensionality of the datasets with more than 20 shapelets, we enforce hierarchical clustering until there are twenty clusters, and select the best shapelet from each cluster to represent the cluster. We perform the binary transform on the data using the class transform approach, and use the correlation filter to remove any attributes that are entirely positively or negatively correlated.

We compare the three dimensionality-reduction methods using a Friedman test at a significance level of 0.01. The results are shown in Figure 6.5. There is no significant difference between the three methods of reducing dimensionality. We continue our experiments and evaluation using truncation, as it is the simplest of the three methods.

We test the performance of nugget discovery on truncated data by comparing the $F1$ values with those of the ensemble (the classes we use are all poorly predicted by the ensemble). We use a Wilcoxon Signed Rank test with a significance level of 0.01. The test shows that there is no significant difference between the performance of the ensemble and the performance of the rule set.

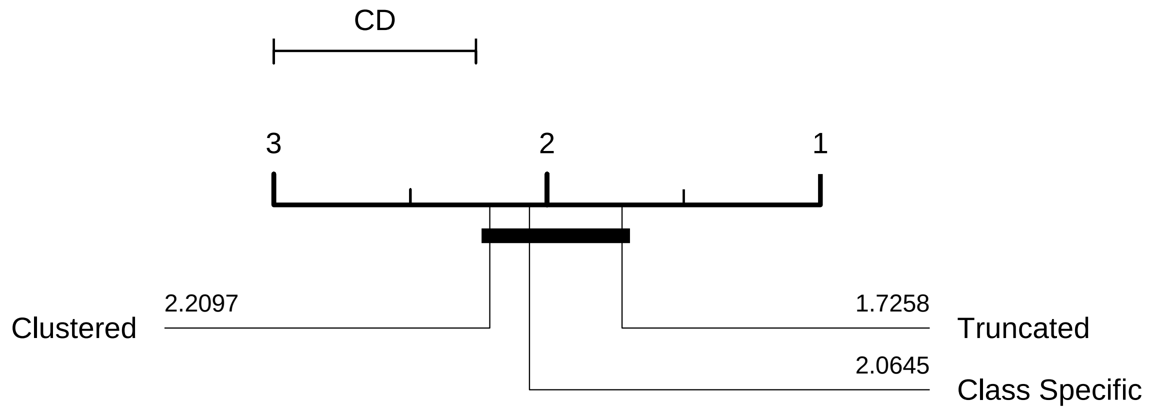

Figure 6.5: Critical-difference diagram comparing three methods of reducing dimensionality in terms of the $F1$ score of the ensemble on the medium-dimensionality datasets reduced using each method. There is no significant difference between the methods.

Figure 6.6 shows the differences sorted by the original $F1$ score of the ensemble on that class. We see that nugget discovery performs better than the ensemble (indicated by negative values) where the initial $F1$ score is very poor. As the performance of the ensemble increases, the performance of nugget discovery on truncated data decreases. Interestingly, the relationship is stronger when the absolute performance of the ensemble is considered, rather than the performance of the ensemble relative to the base incidence of the class in the training data.

These findings suggest that nugget discovery on truncated data, despite not being significantly better than the ensemble, may still be useful in situations where performance is especially poor. A good example of this is the Earthquakes dataset, where the ensemble performs very poorly on class 1, achieving an $F1$ score of only 0.0541, while nugget discovery performs very well, scoring 0.444. In cases like this, nugget discovery may be a useful way to improve predictive accuracy on a minority class, even where the dimensionality of the shapelet data has been severely restricted (in the case of Earthquakes, from 2807 shapelets to 20).

The best course of action for using nugget discovery for medium-dimensionality

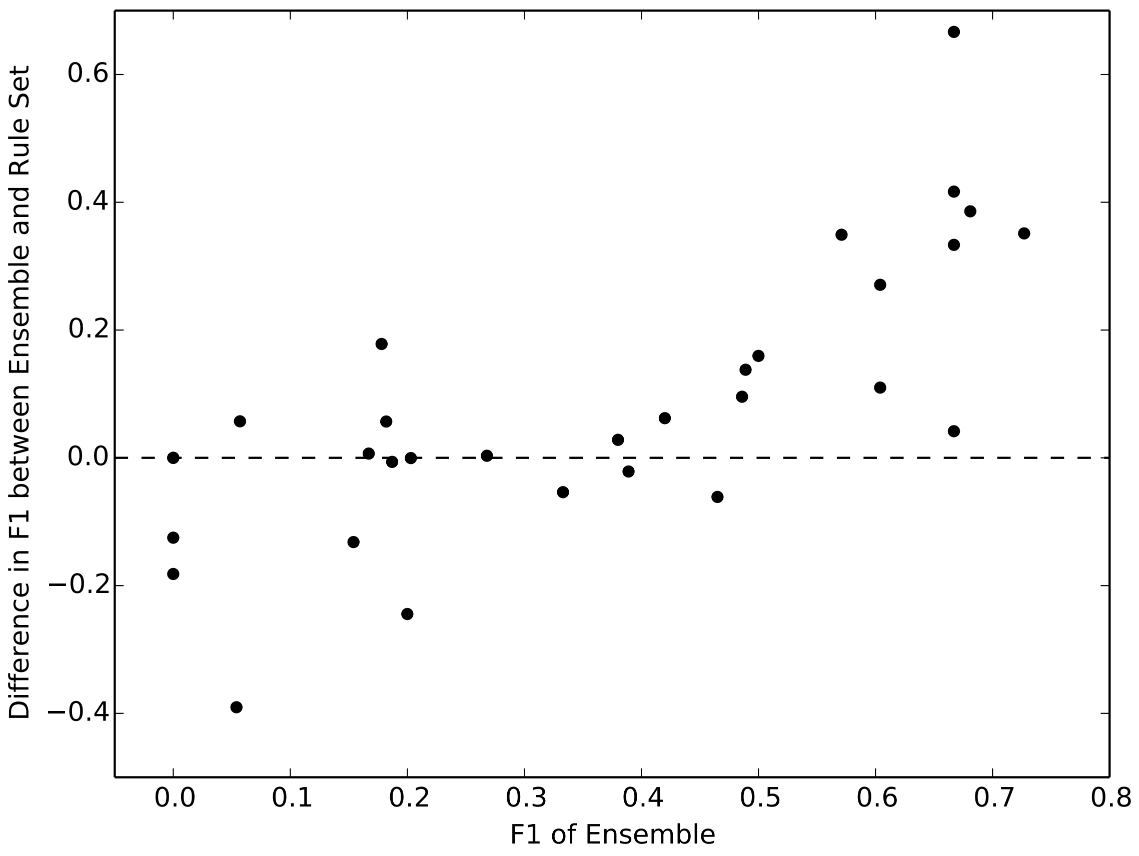

Figure 6.6: Differences in $F1$ between ensemble and nugget discovery on poorly predicted classes of medium-dimensionality data, sorted by the $F1$ of the ensemble on that class.

shapelet data is to restrict it to cases where the performance of the ensemble is very poor in absolute terms, as nugget discovery is unlikely to offer any greater accuracy than the ensemble if the accuracy is not very low, even if the ensemble is performing poorly relative to the incidence of the class in the training data. For the datasets we examine, no classes on which the ensemble has an $F1$ greater than 0.5 benefit from nugget discovery, and the general trend is an increasing improvement offered by nugget discovery as the ensemble $F1$ score decreases.

6.5.4 High-dimensionality experiments

Our final nugget discovery experiment targets those datasets with more than twenty classes. The MDLStopCE clustered data has a minimum number of shapelets equal to the number of classes in the data. Five datasets have more than 20 classes (see Chapter 4), which means that we cannot retain a shapelet of every class and still have the data tractable to Apriori. For these datasets, we adapt the truncation method. We truncate the data first by selecting the best shapelet from each class, as determined by quality measure. The weakest shapelets are eliminated until 20 shapelets remain. The shapelet from the class of interest is excluded from the elimination.

Truncation yields mixed results. For the WordSynonyms dataset, which has only 25 shapelets in MDLStopCE clustered form, nugget discovery offers an improvement over the ensemble for every class we examine. It seems likely that this success stems from the low number of shapelets that have been eliminated. Almost all of the information has been retained, and that is reflected in the performance of the rule set. On the fitywords dataset, only two of the 12 classes show improvement from nugget discovery. Fiftywords has over 350 shapelets in the clustered data, so many shapelets have been eliminated that might have benefited the nugget discovery process. The results are even worse for the other three datasets: nugget discovery offers no improvement in these cases. A huge number of shapelets have been eliminated, making nugget discovery very difficult. This trend is much less evident with the truncated data with fewer than 20 classes. This may be because, where there is a wide variety of shapelets for each class, as is the case for the larger datasets, eliminating whole classes makes discrimination much more difficult. We are forced to eliminate shapelets of some classes, and they might be necessary to distinguish instances of our class of interest from instances of another class, especially if we have multiple shapelets for each class that are no longer included in the data.

One conclusion we may draw from our investigation into nugget discovery is that MDLStopCE clustering is latching onto some feature of the data for its parameterless shapelet clustering. In many cases, nugget discovery performs poorly where there are many shapelets, and the dimensionality of the data must be reduced further to make Apriori tractable. It performs very well in cases where the number of shapelets naturally clusters (according to MDLStopCE) to fewer than 20 shapelets. This suggests that MDLStopCE clustering is finding an appropriate number of clusters without requiring parameterisation.

6.6 Results: comparing suppressed rule sets to unsuppressed rule sets

Our final experiments compare the performance of the rule set to the performance of the same rule set after it has been suppressed using the BruteSuppression algorithm.

6.6.1 Rule set performance

We perform four experiments to measure the effects of suppression on $F1$. The experiments mirror those used in the previous section to compare the performance of nugget discovery to that of the ensemble. The experiments are as follows:

1. Comparison of $F1$ between original rule set and suppressed rule set on poorly predicted classes from datasets with 20 or fewer shapelets.

2. Comparison of $F1$ between original rule set and suppressed rule set on all classes from datasets with 20 or fewer shapelets.

3. Comparison of $F1$ between original rule set and suppressed rule set on poorly predicted classes from truncated datasets where the original number of shapelets is greater than 20, and the number of classes is fewer than 20.

4. Comparison of $F1$ between original rule set and suppressed rule set on poorly predicted classes from truncated datasets where the number of classes is greater than 20.

For each experiment, we compare the $F1$ values using a Wilcoxon Signed Rank test at a significance level of 0.01. The four tests all have the same result: there is no significant difference in $F1$ score between the suppressed rule sets and the original rule sets. We illustrate this in Figure 6.7. The suppressed rule sets perform almost identically to the original rule sets, despite the large reduction in the number of rules (Table 6.3).

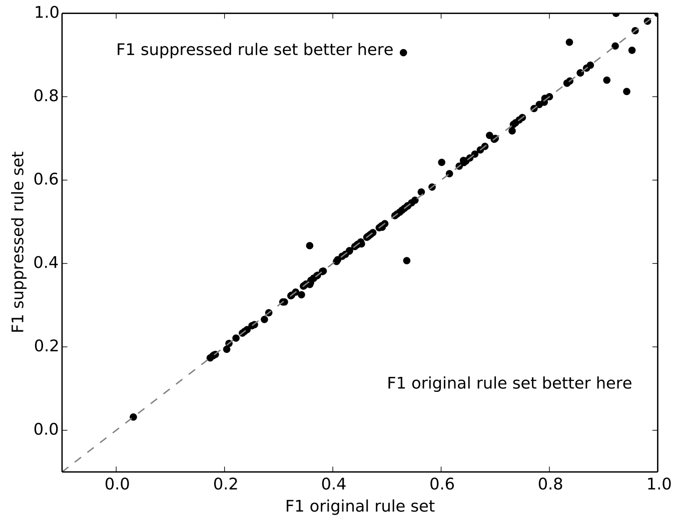

Figure 6.7: Comparison of $F1$ score between original rule set and suppressed rule set on all classes of low-dimensionality data. Points above the dotted line represent classes where the suppressed rule set outperforms the original rule set. Points below the dotted line represent classes where the original rule set outperforms the suppressed rule set.

6.6.2 Analysis

The BruteSuppression algorithm produces rule sets that are very much smaller than the original rule sets (see Table 6.3), and hence more interpretable and comprehensible. This reduction in size causes no loss in $F1$. The greater the size of the original rule set, the more marked the reduction in size (Table 6.3). The results shown in this section compare the suppressed rule sets to the best performing original rule sets, those with ten rules. As the maximum number of rules in the rule set increases, the performance of the original rule sets deteriorates; the performance of the suppressed rule sets remains consistent. We conclude the using BruteSuppression to reduce rule set size is a worthwhile step in the nugget discovery process, particularly if the original rule sets are very large.

In the next section, we show that BruteSuppression can be applied effectively to rule sets other than our binary-shapelet transformed data. Rather than assess the performance of BruteSuppression in terms of $F1$, we take a broader view, showing qualitatively that BruteSuppression can reduce rule set size without negatively affecting the distribution of the rules in the rule set.

6.7 Qualitative analysis of performance of BruteSuppression on different data

In this section, we perform detailed qualitative analysis of the performance of BruteSuppression on data other than our binary-shapelet transformed data. We show that BruteSuppression can be applied to the general problem of reducing rule set size without negatively affecting particular types of rule by examining in detail how the algorithm affects the *Adult* [164] dataset.

We analyse rule sets discovered from the Adult dataset, which represents United States census data. We use Adult because it is a large, well-known dataset that has been studied extensively in machine learning (details are available at [164]), and use it to demonstrate that our method to reduce the size of rule sets is applicable in the general case, not just to rule sets discovered from binary shapelet data. We have performed similar analysis with three other datasets; the results are very similar, and we exclude them for brevity. Details can be found in [86].

We analyse the rule sets in terms of confidence, support, and two novel interestingness measures, *swing*, and *swing surprisingness*. We show that suppression removes only redundant rules, leaving the intrinsic structure of the rule set intact.

6.7.1 Assessing rule set character: novel interestingness measures

For our qualitative analysis of the effects of using BruteSuppression, we propose two novel interestingness measures: swing (an adaptation of *relative surprisingness* [93] and *confidence gain* [160]) and swing surprisingness (an adaptation of *attribute surprisingness* [58]).

For any rule R, let $AT_i \Rightarrow C$ be the rule where the antecedent is the ith AT of R, and the consequent (C) is the consequent of R.

We define swing as follows:

$$Swing(R) = \frac{Conf(R) \times n}{\sum_{i=1}^{n} Conf(AT_i \Rightarrow C)}, \tag{6.7.1}$$

where rule R has n ATs. Swing focuses on the difference in confidence between the ATs in the antecedent taken singly and the rule taken as a whole.

We define swing surprisingness, SS, as follows:

$$SS(R) = \frac{n}{\sum_{i=1}^{n} Conf(AT_i \Rightarrow C)}, \tag{6.7.2}$$

where rule R has n ATs. Swing surprisingness is inversely proportional to the mean confidence of the ATs that make up the antecedent of the rule. The measure assigns rules a higher value if they have less predictive ATs, irrespective of the confidence of the rule.

Swing and swing surprisingness are closely related measures; as can be seen, $Swing(R) = SS(R) \times Conf(R)$. Hence, a rule of moderate confidence will have lower swing than a more confident rule composed of equivalently good predictors. This is not the case with swing surprisingness.

The particular intuition we wish to capture with these measures is that good rules are rules that improve on the individual predictive power of the ATs in their antecedent. If such rules are being eliminated, then we know that the BruteSuppression algorithm is not working to preserve the predictive power of the rule set, regardless of its effect on confidence and support.

We use these measures because BruteSuppression uses confidence as a measure of rule quality; as we demonstrate in Chapter 2, many commonly used interestingness measures are monotonic with respect to confidence, so there is no benefit in using any of them rather than confidence. We want to ensure, however, that we are not eliminating rules that may have other qualities useful for prediction.

We do not test the effects of suppression merely in terms of confidence or support, as this is a relatively narrow definition of interestingness that may not capture what makes a rule predictive. We examine the effect of suppression with confidence and support, and also in terms of swing and swing surprisingness; our aim is to identify whether BruteSuppression deletes rules that may be interesting or predictive in a way that might not be obvious in terms of count-based interestingness measures.

6.7.2 Methodology

We compare three reduced rule sets with the original rule set generated from the Adult dataset (for brevity, we focus on Adult; see [86] for qualitative analysis of a number of other datasets). We reduce them to approximately the same size using either the minimum confidence parameter, the minimum antecedent support parameter, or the BruteSuppression algorithm. The effects of reduction are assessed in two ways. First, we compare the distribution of rules using a chi-squared test. We take it that a reduced rule set should have a similar distribution of rules to the original set, as this suggests that no particular classes of potentially predictive rules are being removed. Second, we examine the distribution of rules visually. This enables us to discover which rules are being eliminated when the rule set is reduced. We take it that a good reduction should leave the same shaped distribution but with fewer rules, rather than a completely different shaped distribution.

The Adult dataset has 30,162 records in the training set and 12,435 records in the test set. It has 14 attributes, of which six are continuous. The target class is > 50K; the base incidence rate of this class is 0.249 in the training data and 0.236 in the test data. We generate the original rule set using a minimum support of 2% and minimum confidence of 0.25. The minimum confidence is the base incidence rate of the target class (we assume that rules with lower confidence than this are uninteresting). The minimum support is selected to allow Apriori to generate the rule set in a reasonable time frame. We feel that this has not compromised the results, as the setting is what a user might select, giving us a more realistic idea of the effectiveness of our algorithm. We have tested the algorithm on a single unconstrained rule set with similar results to the constrained rule sets.

We select the parameter settings for the increased minimum confidence and increased minimum antecedent support rule sets to produce rule sets similar in size to the suppressed set. This way, a fair comparison can be made between the effects of the BruteSuppression algorithm and the effects of the same degree of reduction from the parameter settings. We use a minimum confidence of 0.69 for the increased minimum confidence rule set, and a minimum support of 8% for the increased minimum support rule set (in each case, the other parameter remains unchanged). All of the rule sets are assessed on previously unencountered test sets; the various measures we use are assessed on the rules' performance on the test set, rather than the training set.

We initially investigate the effect of the reduction methods on the distribution of rules in a rule set using the chi-squared statistic. If two sets of values are drawn from the same distribution, the chi-squared statistic obtained by comparing them is likely to be small. Large chi-squared values are indicative of the rule sets having different distributions of the qualities in question. We rank the different methods by their chi-squared values (see Table 6.4) for rule confidence, coverage, swing, and swing surprisingness.

We also visually examine how suppression affects the distribution of rules in terms of coverage (see Chapter 2), confidence, swing, and swing surprisingness. In particular, we are interested in whether certain classes of rules are eliminated by suppression. We can divide rules into four classes: strong (high coverage and high confidence), general (high coverage, low confidence), exception (low coverage, high confidence, see [121, 93]), and weak (low coverage, low confidence). We do not discuss the effect of suppression on weak rules, as these are unlikely to be of interest, but the other three classes should be reduced equally to yield a rule set that is likely to be predictive. We are also interested in rules with high swing/swing surprisingness; eliminating these rules is likely to compromise the predictiveness of the rule set.

Table 6.4: Chi-squared values for the suppressed rule set, rule set reduced using increased minimum confidence, and rule set reduced using increased minimum antecedent support. A lower chi-squared value indicates a more similar distribution of rules relative to the original rule set.

Measure	Suppressed	Min. Confidence	Min. Antecedent Support
Confidence	**43.643 (1)**	360.815 (3)	105.867 (2)
Coverage	**9.133 (1)**	66.599 (2)	234.239 (3)
Swing	**5.098 (1)**	309.183 (3)	63.489 (2)
Swing Surprisingness	24.653 (2)	150.697 (3)	**18.431 (1)**
Avg. Rank	**1.25**	2.75	2

6.7.3 Comparing rule distribution using the chi-squared statistic

A summary of the results of our chi-squared distribution tests is shown in Table 6.4. For each reduced rule set, we generate a chi-squared statistic for each of the four properties by comparing the reduced rule set to the original set. The lower the chi-squared statistic (for a given original rule set), the more closely the distribution of rules in the reduced set matches that of the rules in the original set for that property. For each original set, we ranked the three reduced rule sets by their chi-squared statistic, assigning rank one to the reduced rule set with the lowest chi-squared statistic, and hence the greatest resemblance to the original rule set. Table 6.4 shows the number of instances in which each type of reduced rule set is ranked first, second, and third, and the average rank.

As can be seen in Table 6.4, the suppressed rule sets have consistently lower chi-squared values when compared to the original rule set than rule sets reduced by the other two methods. In the single case where the suppressed rule set does not have the lowest chi-squared statistic (swing surprisingness), the value is very close to the lowest value. We take this as good evidence that suppressed rule sets retain the distribution of the original rule sets. To expand on this analysis, we visually examine the distribution of rules in the next section.

6.7.4 Visual analysis of rules from the Adult dataset

Coverage/confidence distribution

The suppressed rule set has maintained the shape of the original distribution (see Figure 6.8). The rule set constrained with increased minimum confidence has lost every general rule and a large number of exception rules, changing the distribution substantially. The rule set constrained with increased minimum antecedent support has lost all of the higher confidence rules (both exception rules and strong rules). The coverage/confidence distributions suggest that increasing the minimum confidence or antecedent support parameters to constrain rule set size negatively affects the rule set, relative to a suppressed set of similar size.

Confidence/swing distribution

The suppressed rule set maintains the shape of the confidence/swing distribution to a high degree (see Figure 6.9). This suggests that swing is unlikely to be negatively affected by suppression.

Neither of the sets constrained with the Apriori parameters have maintained the shape of the confidence/swing distribution. Many rules with high swing relative to their confidence have been eliminated, suggesting that the rule sets are compromised in terms of swing relative to the original set and the suppressed set.

Confidence/swing surprisingness

The suppressed rule set generated on Adult maintains the shape of the confidence/swing surprisingness distribution of the original rule set (Figure 6.10). The rule set constrained with increased minimum confidence has many rules of high swing surprisingness missing, relative to the original set. The rule set constrained with increased

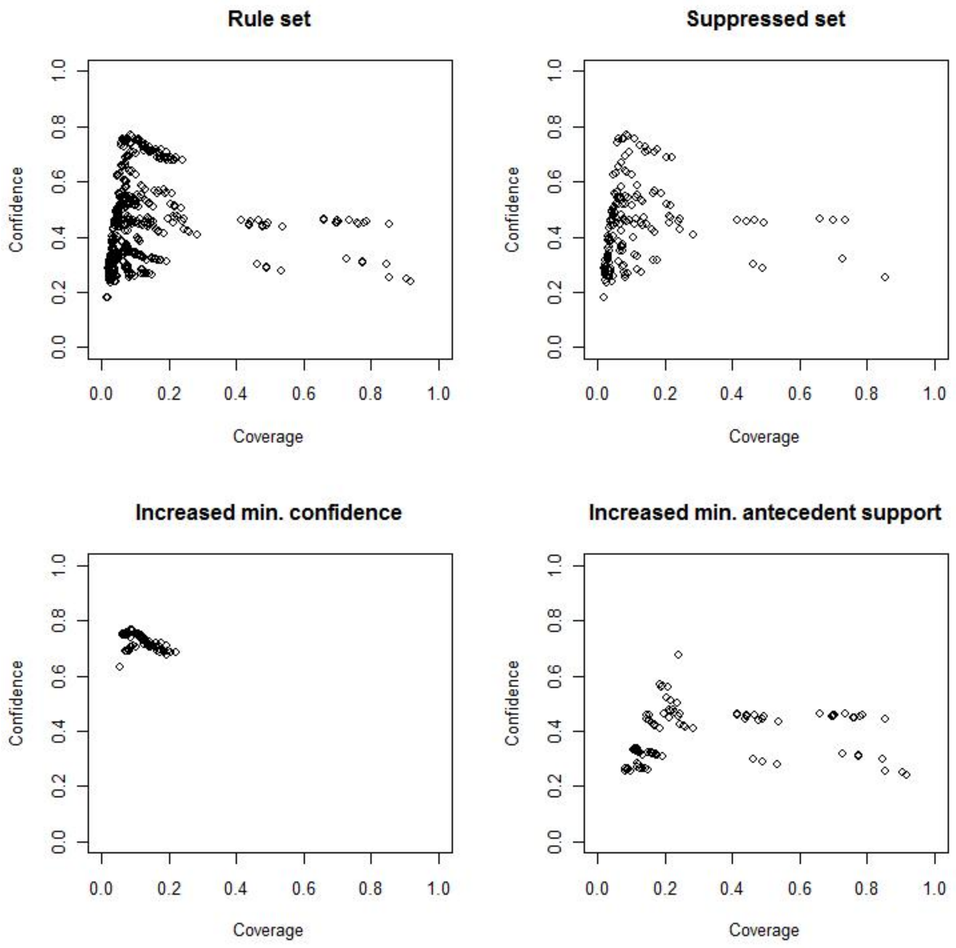

Figure 6.8: Coverage/confidence rule distribution for rule sets generated from Adult.

minimum antecedent support is more similar to the original rule set, but has still lost many rules that have been retained in the suppressed rule set. This shows that using these measures to constrain rule set size may adversely affect rule sets.

6.7.5 Assessment

We have shown that BruteSuppression can be used to reduce rule set size without altering the distribution of the rule set in terms of a number of different interestingness measures. In contrast, using the minimum confidence or minimum antecedent support

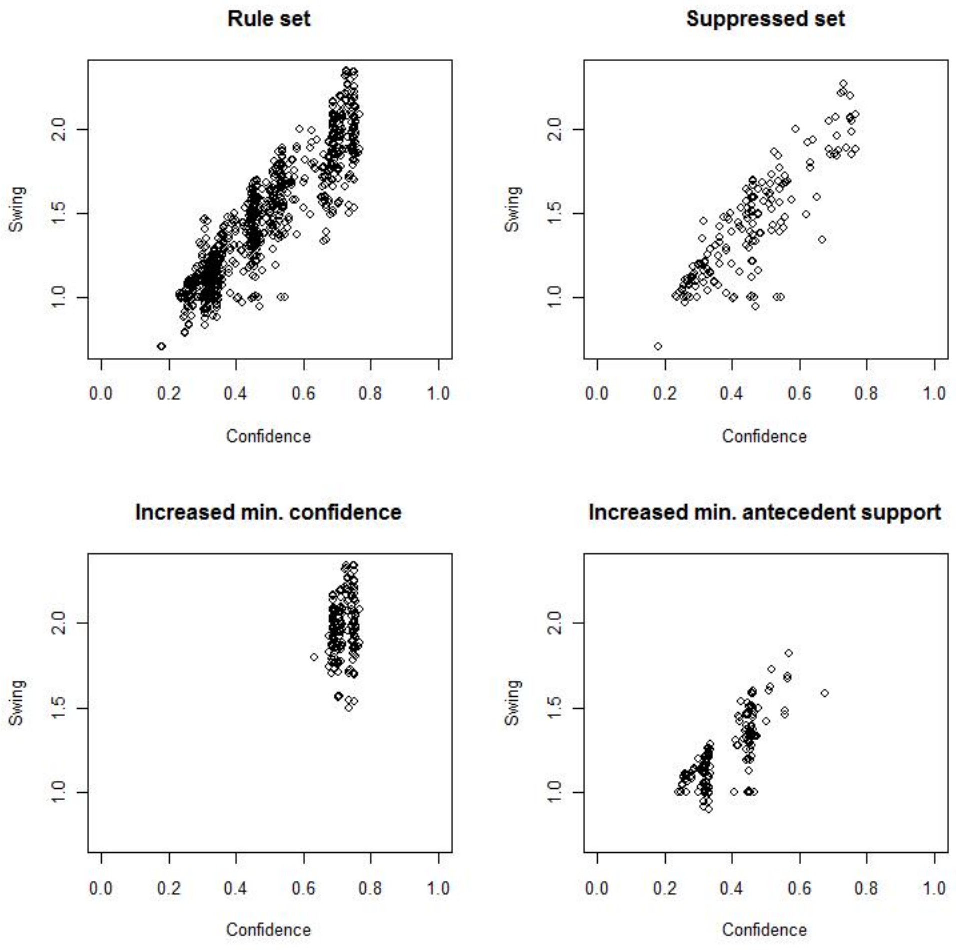

Figure 6.9: Confidence/swing rule distribution for rule sets generated from Adult.

parameters built into Apriori changes the distribution of the rules considerably. We take this as evidence that BruteSuppression is applicable to the general problem of reducing rule set size, as well as to the specific problem of reducing the size of our rule sets without harming the partial classification performance of the rule set.

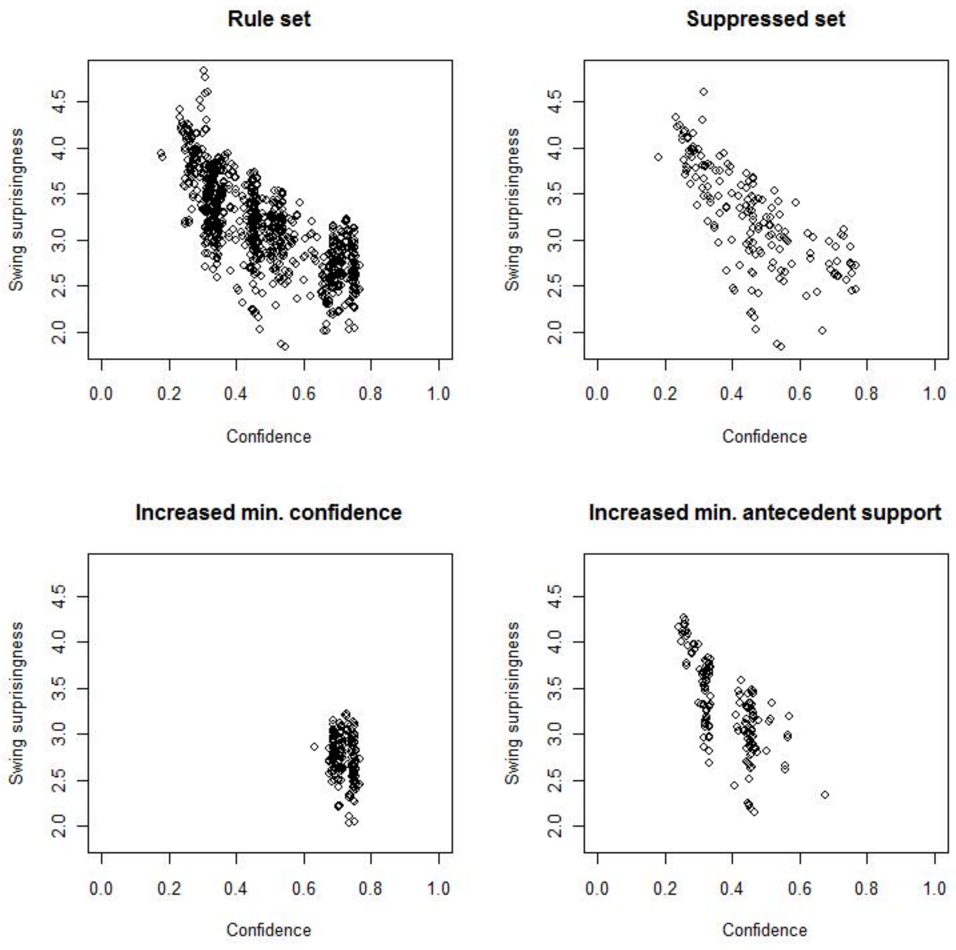

Figure 6.10: Confidence/swing surprisingness rule distribution for rule sets generated from Adult.

6.8 Conclusions

Nugget discovery can be used to improve classification accuracy on a given class, as well as providing interpretable rules that can provide insight into the problem domain. It combines well with binary-shapelet data, which is designed to be interpretable.

We make two novel contributions: the use of nugget discovery with binary shapelet data to yield accurate, interpretable partial classification for classes where the ensemble performs poorly, and BruteSuppression, an algorithm for reducing rule set size to improve interpretability while retaining both the character of the rule set and classification performance.

We show that nugget discovery outperforms the ensemble on poorly predicted classes for shapelet data with 20 or fewer shapelets. Where there are more than 20 shapelets, there is no significant difference in accuracy, though in individual cases the rule sets may provide considerable increases in accuracy. We also show that there is no significant difference in accuracy between a suppressed rule set and an unsuppressed rule set, though in general the suppressed sets are very much smaller. As rule set size increases, performance of the suppressed sets remains constant, while performance of the unsuppressed rule sets degrades. Finally, we show qualitatively that BruteSuppression can be applied more generally as a method to reduce rule set size without affecting the character of the rule set.

We conclude that nugget discovery with BruteSuppression can be used to increase accuracy where dimensionality is low and the ensemble has performed poorly. We recommend its use as a supplement to the ensemble classifier. As well as accuracy, the suppressed rule sets combine well with the binary shapelet data to offer comprehensible partial classification that could provide insight into the problem domain.

Chapter 7

Unsupervised Learning with Localised Shapes

7.1 Introduction

Finding motifs (approximately recurring subsequences) in time series is an unsupervised data-mining problem that mirrors the supervised shapelet approach. Rather than finding the subsequences that are most discriminative of classes by mining a labelled dataset, the problem is to find sets of subsequences that approximately match in a longer time series. This represents a different application of the idea of local-shape-based similarity.

Two significant papers on exact discovery of motifs are [131, 132]. Their emphasis is on finding best-matching pairs of subsequences. Our contribution is directed at exact discovery of frequently occurring subsequences, rather than best-matching pairs. We propose three algorithms for this purpose. Scan MK and Cluster MK (Sections 7.2.1 and 7.2.2 respectively) use the *Mueen-Keogh* (MK) pair-matching algorithm (Section 3.5.3) as a subroutine (although any pair-finding algorithm could be used) for building motif sets, either by scanning and condensing the subsequences around the matching pairs (Scan MK), or by hierarchically clustering the subsequences and centroids. Our third algorithm, Set Finder, finds motif sets based on whole set quality,

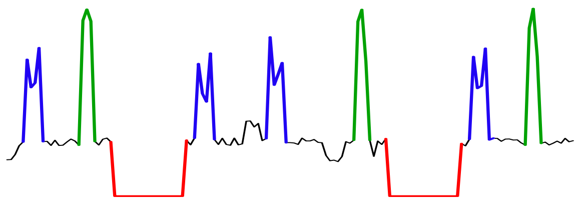

Figure 7.1: A simulation of an electricity demand profile, with three motif sets. The red pattern represents a period of no usage, the blue pattern approximates the usage pattern of a dishwasher, and the green pattern approximates the usage pattern of an immersion heater.

rather than pair matching (Section 7.2.3). We assess their performance on both real and synthetic data, described in Chapter 4.

The real-world problem we study is finding motifs in household electricity-usage profiles. The problem is as follows: given a time series of household electricity usage, taken at 15-minute intervals over the study period, find repeating patterns of usage and relate these patterns to devices. If we can solve this problem, we can use the motifs in further analysis. For example, we can attempt to deconstruct the household usage into its constituent parts, disaggregating the data in terms of devices [64], or use the devices detected within a household to profile and cluster customers. Figure 7.1 shows a manufactured example of the type of time series in which we wish to find motifs.

The results, presented in Section 7.3, show that Set Finder and Cluster MK find motif sets more accurately than Scan MK on the synthetic data; however, Cluster MK is more sensitive to parameter settings and much slower than the other two algorithms. Scan MK and Cluster MK are faster than Set Finder on the electricity data. Our qualitative analysis, however (see Section 7.4), shows that Set Finder produces motif sets with more meaning in relation to the problem domain. In Section 7.5, we conclude that, for the type of problem we have studied, Scan MK does not find motif sets as accurately as Set Finder, although it is much faster on smoother data; the more sophisticated approach of Cluster MK is promising in terms of accuracy, but is not suitable for noisy data in terms of speed.

7.2 Finding motif sets

7.2.1 Scan MK

Mueen *et al.* [132] define frequently occurring patterns as *range motifs*, and claim that once the best-matching pair is found with MK, finding the range motif is trivial through a linear scan. The process is not trivial, however, because of the necessity of avoiding trivial matching and ensuring that members of the motif set are a minimum distance apart. The linear scan discovers every matching subsequence to the original pair; the real difficulty lies in selecting the appropriate subsequences from that set to give the best motif set while satisfying the constraints.

Our contribution to solving this problem is extending the method for finding the range motif outlined in [132] to find approximate K-motif sets. We iterate the process of finding closest pairs and their matches, adding them to a motif set and removing members and their trivial matches from the list of candidates after each iteration. The algorithm is described in Algorithm 17. We assume a distance function $d(S_i, S_j)$ is defined (we use Euclidean distance for all experiments). r is approximately the cluster radius; for Scan MK, $2r$ is the maximum distance permitted between two members of the same cluster.

MK is used to find the best-matching pair of subsequences in S (line 6); if the distance between them is greater than $2r$, the algorithm terminates. Otherwise, the best-matching pair is added to a motif set, the trivial matches of the best-matching pair are removed from S (lines 12-15), and the remaining subsequences are scanned. Any subsequences within $2r$ of both members of the best-matching pair are added to the motif set (lines 16-22).

The `condense` function operates as follows. For each set of contiguously-indexed subsequences, the non-trivial match is taken to be the subsequence with the smallest total distance between it and each of the members of the motif set. The contiguously-indexed subsequences are taken to be trivial matches, and are excluded from the motif set. The motif set is further condensed by removing one subsequence of any pair whose members are more than $2r$ apart. We choose which subsequence to exclude based on the number of clashes. For example, if a subsequence that clashes with three others is greater than $2r$ from a subsequence that clashes with two others, the first subsequence is removed, maximising the cardinality of the set. Ties are decided based on average linkage; the subsequence with the shortest total distance to the other members of the set is retained. Once the motif set is established, its members and their trivial matches are removed from the candidate set, and the process is repeated until no more subsequences are within $2r$ of each other.

7.2.2 Cluster MK

The second algorithm we propose is based on hierarchical clustering of best-matching pairs. Hierarchical clustering is a widely used clustering approach (see, for example, [100]), based on finding best-matching pairs of series. We use MK to find the pairs, and an adapted form of bottom-up hierarchical clustering described in Algorithm 18.

We find the closest pair of subsequences, then merge this pair to form a new cluster (motif set). The cluster is represented by a new subsequence found by averaging the input subsequences, weighted by the number of subsequences that have already been combined. This ensures the cluster centre accurately reflects the members of the cluster. The process is repeated until the distance between the best-matching pair is greater than r. At this point the subsequence set S will contain the motifs, and the motif sets can be recovered from the clustering data structure.

Algorithm 17 `scanMK`($\mathbf{F}$, the set of all length n subsequences, r)

```
1: $M \leftarrow \emptyset$
2: $S \leftarrow \mathbf{F}$
3: $k \leftarrow 0$
4: while $end =$ FALSE do
5:    $end \leftarrow$ TRUE
6:    $\{L_1, L_2\} \leftarrow$ MK($S$) $\{L_1$ and $L_2$ are indexes in $\mathbf{F}\}$
7:    if $d(\mathbf{F}_{L_1}, \mathbf{F}_{L_2}) \leq 2r$ then
8:       $end \leftarrow$ FALSE
9:       $k \leftarrow k + 1$
10:      $M_k \leftarrow \{\mathbf{F}_{L_1}, \mathbf{F}_{L_2}\}$
11:      $D \leftarrow \emptyset$
12:      for $i \leftarrow 1$ to $|S|$ do
13:         if trivialMatch($\mathbf{F}_{L_1}, S_i$) $\vee$ trivialMatch($\mathbf{F}_{L_2}, S_i$) then
14:            $D \leftarrow D \cup S_i$
15:      $S \leftarrow S - D$
16:      for $i \leftarrow 1$ to $|S|$ do
17:         if $d(\mathbf{F}_{L_1}, S_i) \leq 2r \wedge d(\mathbf{F}_{L_2}, S_i) \leq 2r \wedge S_i \notin D$ then
18:            $M_k \leftarrow M_k \cup S_i$
19:            for $j \leftarrow 1$ to $|S|$ do
20:               if trivialMatch($S_i, S_j$) then
21:                  $D \leftarrow D \cup S_j$
22:      $S \leftarrow S - D$
23:      $M_k \leftarrow$ condense($M_k, r$)
24: if $k > 0$ then
25:    $M \leftarrow \{M_1, ..., M_k\}$
26:    sort($M$)
27: return $M$
```

Algorithm 18 `clusterMK`($\mathbf{F}, r$)

```
1: $S \leftarrow \mathbf{F}$
2: while $end =$ FALSE do
3:    $end \leftarrow$ TRUE
4:    $\{L_1, L_2\} \leftarrow$ MK($S$)
5:    if d($\mathbf{F}_{L_1}, \mathbf{F}_{L_2}$) $\leq r$ then
6:       $end \leftarrow$ FALSE
7:       $S \leftarrow S - \{\mathbf{F}_{L_1}, \mathbf{F}_{L_2}\}$
8:       $c \leftarrow merge(\mathbf{F}_{L_1}, \mathbf{F}_{L_2})$
9:       $S \leftarrow S \cup c$
10: return $S$
```

7.2.3 Set Finder

We propose an algorithm to find the K-motif sets directly, based on counting and separating (Algorithm 19). Each subsequence is compared to every other subsequence, and the non-trivial matches are counted. The set of counts is sorted. The sorted set is then input to the function `separate`, which checks each subsequence with a non-zero count in order to ensure that it is at least $2r$ apart from subsequences with a greater number of matches. Subsequences that fail the test are removed from the set. An early abandon based on the value of r is built into the distance function to speed up the algorithm.

Algorithm 19 `setFinder`$(\mathbf{F}, r)$

```
1: $C \leftarrow < 0, \ldots, 0 >$
   {$C$ is counts vector of length $|\mathbf{F}|$ initialised to 0}
2: for $i \leftarrow 1$ to $|\mathbf{F}|$ do
3:    for $j \leftarrow i + 1$ to $|\mathbf{F}|$ do
4:       if $\mathtt{d}(\mathbf{F}_i, \mathbf{F}_j) \leq r \wedge$ trivialMatch$(\mathbf{F}_i, \mathbf{F}_j)=$ FALSE then
5:          $C_i \leftarrow C_i + 1$
6:          $C_j \leftarrow C_j + 1$
7: sort$(C, \mathbf{F})$
8: $M \leftarrow$ separate$(C, \mathbf{F})$
9: return $M$
```

The storing and recovery of the motif sets is omitted for clarity, but is easily achieved by retaining references to subsequences in addition to count data.

7.3 Synthetic data results

7.3.1 Timing experiments

Our primary aim is to assess how accurately the algorithms discover motif sets; for completeness, we also include timing comparisons. As can be seen in Figure 7.2, Cluster MK performs very poorly on the synthetic data; its times increase exponentially with the range parameter r. There is a relatively small difference between Set Finder

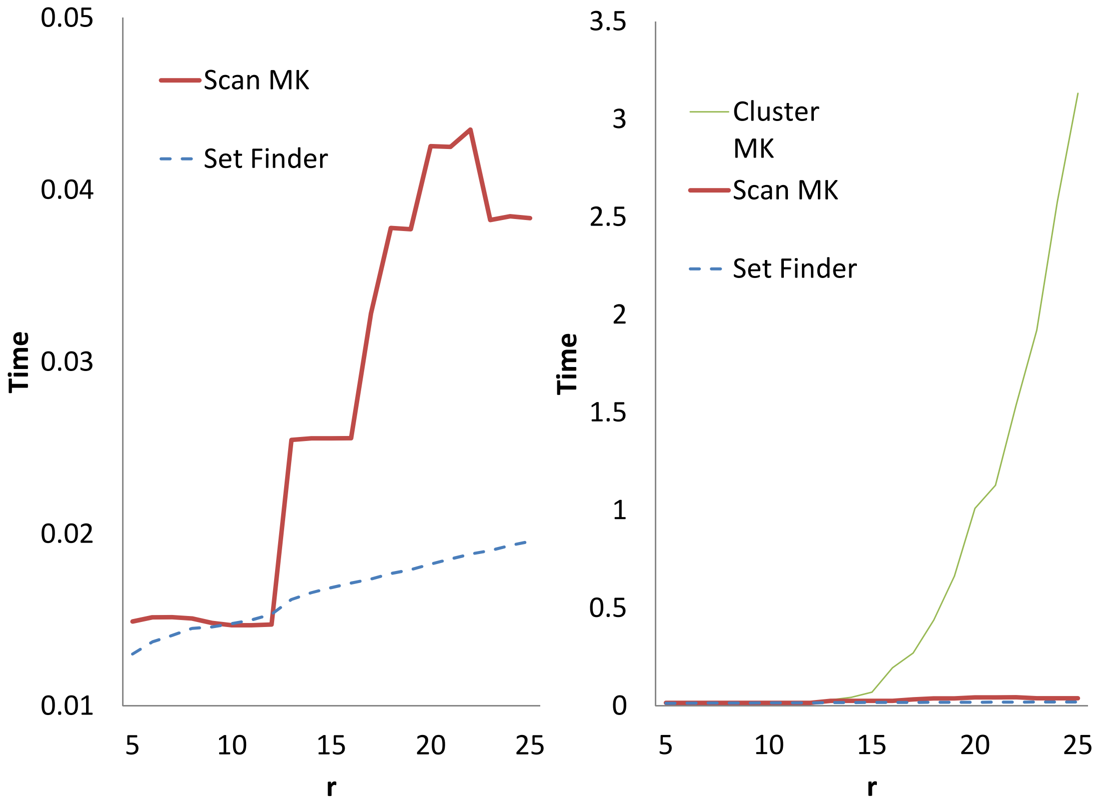

Figure 7.2: Time in seconds averaged over 100 runs for Set Finder, Scan MK, and Cluster MK for varying values of r on synthetic data (left, Set Finder and Scan MK, right, all three algorithms).

and Scan MK, with noticeable increases in time taken by the latter when the MK algorithm must be called multiple times. The time taken by Set Finder increases linearly with the value of r (this is true for both the synthetic and the electricity data), while the time taken by Scan MK remains constant until the increase in r causes additional calls to the MK algorithm. It should be noted that the synthetic data we have used is a worst-case scenario for MK; we attribute the performance of Set Finder on this data to the aggressive early abandon it employs and that it performs only one pass through the data. Since the other two algorithms must call MK multiple times for higher values of r, they perform more slowly.

For the electricity data (Figure 7.3), Scan MK and Cluster MK are much more impressive. This data is well suited to the MK algorithm, and the times for both

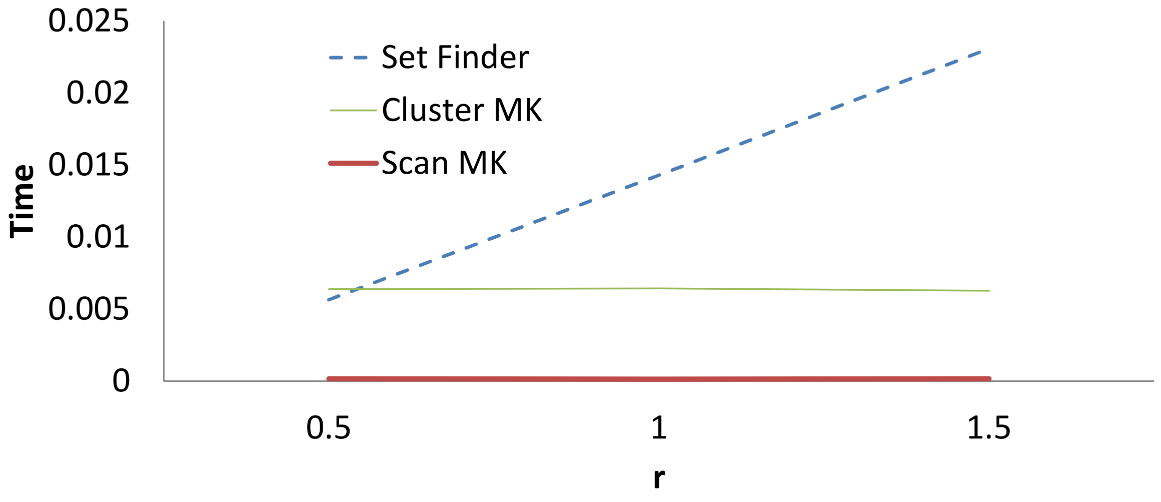

Figure 7.3: Time in seconds averaged over 100 runs for Set Finder, Scan MK, and Cluster MK for varying values of r on electricity data.

remain fairly constant, and lower than Set Finder, across all r values. Scan MK is very fast on this data. Set Finder performs adequately, but is clearly the slowest of the three algorithms.

From our timing experiments, we conclude that Cluster MK may be infeasibly slow for large, noisy datasets; Scan MK and Set Finder take a roughly equivalent amount of time for this type of data, assuming the value of r is not too high. Cluster MK is a better choice for smoother datasets such as our electricity-usage data; Scan MK is very fast and may be the best choice for very large, smooth datasets. Set Finder is the slowest of the three algorithms on this data, but its speed is more robust to increasing noise than that of the other two algorithms.

7.3.2 Performance evaluation

To assess performance, we measure how well the algorithms find labelled motif sets. The performance of the algorithms is assessed on two criteria: the proportion of the discovered motif set that is correct (precision) and the proportion of the labelled motif sets that are found by the algorithm (sensitivity). We calculate these as follows. An index to a subsequence returned by an algorithm is considered a *true positive* (TP) if it is within $\frac{n}{2}$ of the index of the shape in the data. It is considered a *false positive* (FP) if it is not within $\frac{n}{2}$ of the index of any shape in the data. A *false negative* (FN) is any shape contained in the data with an index that is not within $\frac{n}{2}$ of any index returned by the algorithm. We calculate two measures of accuracy:

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}, \tag{7.3.1}$$

and

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP+FN}}. \tag{7.3.2}$$

For synthetic data containing two motif sets, the sets of indexes generated by the algorithm are paired with the indexes of the motif sets in the data, giving a combination we refer to as a *matching*. A score is calculated for each matching as follows. Each index that is not paired with another index adds the value of n (the length of the shape in the data) to the score. In our experiments, n is fixed at 29. For each pair of indexes, the absolute value of the difference between the two is calculated, with a ceiling fixed at the value of n. Hence, pairing an index of 13 with an index of 21 gives a score of 8. The scores are tallied for each possible combination of indexes within sets, and of matchings between sets. Our measure rewards close matches, and punishes false negatives and false positives equally. Thus, if the data contain two shapes, A and B, with the associated index sets $A = \{13, 240, 500\}$ and $B = \{456, 708, 865\}$, and the algorithm has returned a single set of indexes $ALGO = \{447, 700, 870\}$, we tally scores for each of the two possible pairings of sets $\{A, ALGO\}$ and $\{B, ALGO\}$, by scoring each of the ways of pairing the indexes within those sets. The lowest score is the best match, and is returned as the score of the algorithm on that data.

To assess significance for differences between algorithms at a given value of r, we perform a two-sample T-test with an alpha value of 0.05.

We compare the algorithms on two problems: finding a single set of shapes inserted into random noise, and finding two sets of shapes inserted into random noise. The second problem is more complex, and more representative of real-life applications. We use a range of r values for two reasons. First, we are interested in discovering the value of r that is appropriate for various situations. Second, we are interested in how the algorithms compare to one another over a range of values; if we used a single value for r, our results might be misleading.

7.3.3 r value

For the synthetic data, all algorithms perform best when r is around $\frac{n}{2}$ for motif length n. We speculate that the high level of noise in the data prevents successful discovery of motifs with smaller values of r. For noisy data, we recommend setting r at this level as a heuristic. If r is higher than $\frac{n}{2}$ on our synthetic data, performance begins to degrade.

The electricity data is much less noisy than the synthetic data. On that data, we achieve good results with $r = \frac{n}{8}$. This equates to an r value of 3.6 for the synthetic data, a value at which the sensitivity is 0. The synthetic data is too noisy to tolerate such a low value of r. As a general rule, we suggest indexing r to n, and decreasing the value of r as the level of noise in the data decreases.

7.3.4 Problems with a single motif set

We find the precision and sensitivity for a range of values of r. In the limit, we would expect precision to approach unity as r decreases, and sensitivity to approach unity as r increases (for example, an algorithm returning the indexes of all subsequences would have sensitivity 1 and precision approaching 0).

For the experiments using a single shape, we tested the three algorithms over 1000 datasets containing three to five instances of the shape, using different values

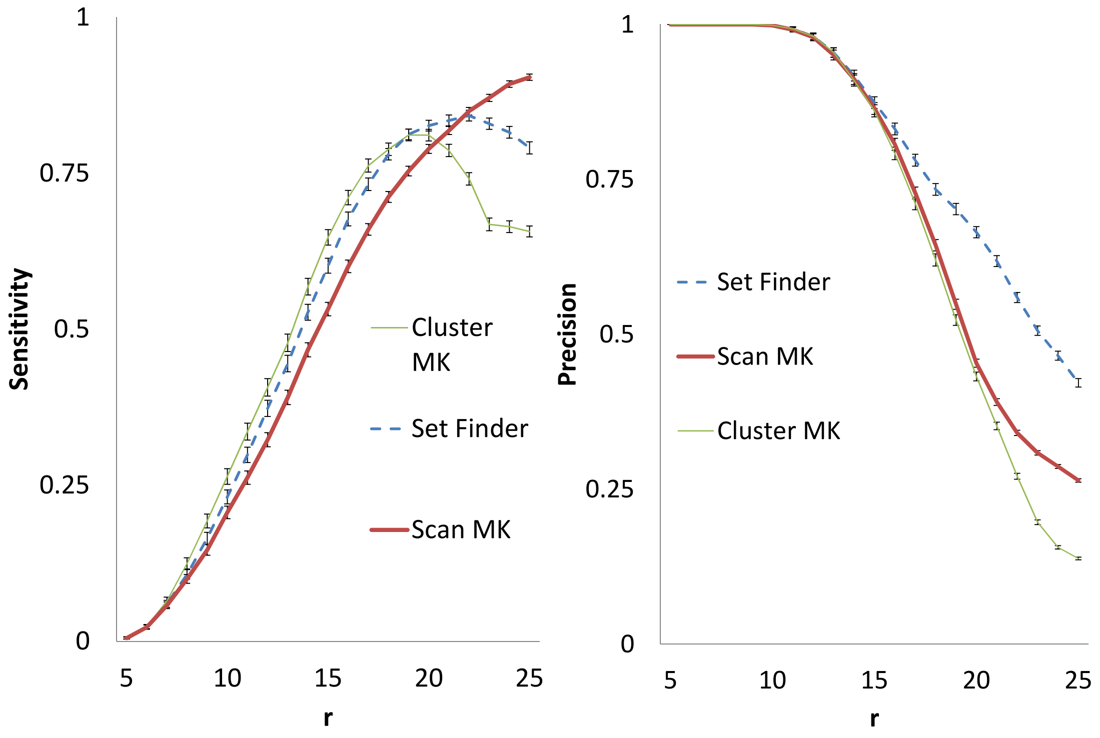

Figure 7.4: The mean sensitivity (left) and precision (right), with standard error, over a random sample of 1000 instances of the single shape datasets for Scan MK, Cluster MK, and Set Finder, with values of r in the range 5 to 25.

of r in the range $r = [5, 25]$. Figure 7.4 shows the results of these experiments. The statistically significant differences are listed below.

In terms of sensitivity, the Set Finder algorithm outperforms the Scan MK algorithm significantly in the range $r = [10, 20]$. Cluster MK has better sensitivity than either of the other algorithms in the range $r = [9, 17]$. The sensitivity score for Cluster MK is the worst for values of r greater than 20, suggesting that the algorithm requires accurate tuning to return useful results. For values of r greater than 22, Scan MK outperforms Set Finder. However, this appears to be because the Scan MK algorithm returns many more results for r in this range, and consequently has a greater number of both true and false positives, an inference that is supported by the precision results. For values of r above 15, the Set Finder algorithm has significantly better precision than the Scan MK or Cluster MK algorithms. The precision of the

Scan MK algorithm diminishes more quickly than that of Set Finder because the algorithm generates more false positives for higher values of r. The precision of Cluster MK is the worst for values of r greater than 16; although the algorithm generates many false positives, it also has low sensitivity at high values of r because it misses true positives. Again, this is suggestive of a need for careful parameter setting when using Cluster MK.

We conclude that Cluster MK and Set Finder are more accurate than Scan MK; Cluster MK is better for finding motifs, and Set Finder is better for avoiding false positives. Cluster MK has the disadvantage that it is very sensitive to the value of r, and its performance degrades quickly outside of the optimum range. Scan MK is significantly more sensitive at high values of r, but the concomitant loss of precision suggests that it will give many false positives in this range, which may be unsuitable for certain tasks.

7.3.5 Problems with two motif sets

Finding multiple motif sets is more complex, not least because the algorithms must distinguish between the subsequences that belong to different sets (see Section 7.3.2). For the single shape problem, we permit multiple sets to be aggregated; for the two-shape problem, each set returned by the algorithm is assigned to at most one of the motif sets in the data. Hence, an output of a single set containing all of the instances of both shapes is rewarded only for finding one motif set, and punished for missing the other set and for false positives. Equally, if the algorithm finds all instances of both motif sets, but splits them into many different sets, it is punished accordingly.

The results of the two-shape experiments are shown in Figure 7.5.

The Set Finder algorithm performs significantly better than the Scan MK algorithm in the range $r = [11, 15]$. Outside of this range, there is no significant difference

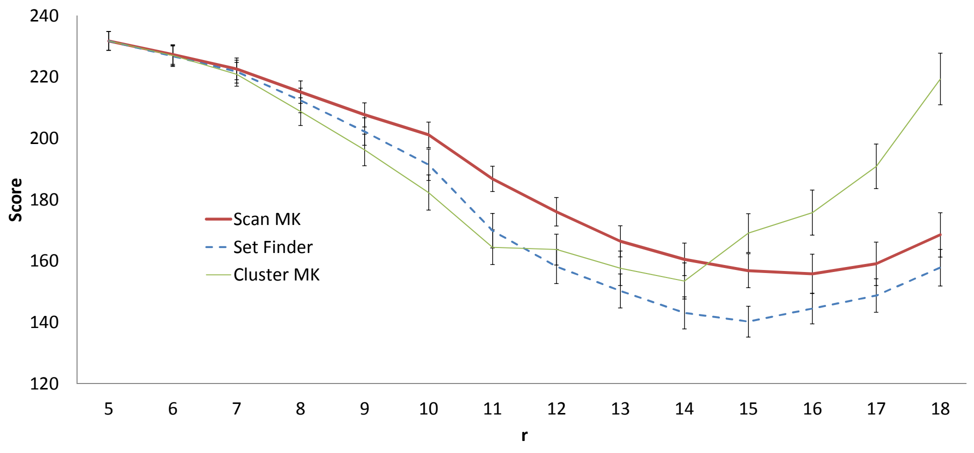

Figure 7.5: The mean match score (as defined in Section 7.3.2) and standard error over a random sample of 100 instances of the two shape datasets for Scan MK, Cluster MK, and Set Finder with values of r in the range 5 to 18. Lower scores indicate better performance.

between the two algorithms. For values of r greater than 14, Set Finder outperforms Cluster MK; at lower values, there is no significant difference in performance. In the range $r = [9, 12]$, Cluster MK outperforms Scan MK; this result is reversed in the range $r = [16, 18]$, where Cluster MK has the worst performance of the three algorithms. In accordance with the results of the single-shape problem, it seems that Cluster MK is more sensitive to the value of r than the other two algorithms. The best values for all of the algorithms occur in the range $r = [14, 16]$. This is also approximately the optimal range for precision and sensitivity in the single-motif problem. It is our opinion that experiments performed with a greater number of datasets (say, 1000, rather than 100) would show Set Finder to be significantly better over a greater range of values, but would also show Cluster MK to be superior at lower values of r.

Our results suggest that Set Finder is the most accurate algorithm when r is approximately $\frac{n}{2}$, which we suggest is an appropriate value for noisy data. Once again, Cluster MK is very sensitive to the value of r, which is a weakness of the algorithm, as small differences in r (which is difficult to estimate precisely), can cause the algorithm's performance to deteriorate.

7.4 Electricity-usage data

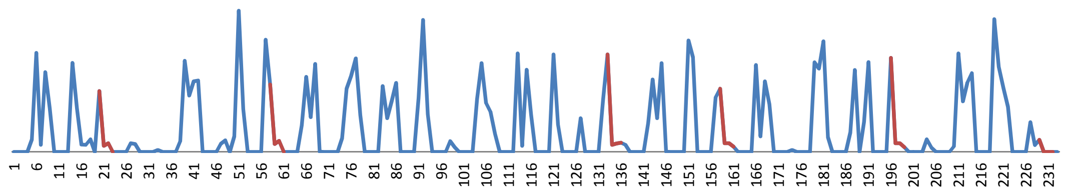

Figure 7.6: A household electricity-usage profile; the subsequences returned by the Set Finder algorithm as members of the two-motif set are highlighted in red, and represent the washing machine device.

In this section, we present qualitative analysis on the performance of Set Finder and Scan MK on household electricity-usage data. We use a window size of 4, as this represents one hour. Our results show that analysis in terms of motif sets is likely to be fruitful for profiling device usage.

We first analyse the performance of the Set Finder algorithm on a usage profile. The usage profile contains usage instances of three devices: a dishwasher, a washing machine, and an oven. Using the values $r = 0.5$ and $n = 4$, the algorithm returns two sets of indexes. Unsurprisingly, the larger set contains all instances of four consecutive zeros, representing the instances of no device usage. More interestingly, the other set consists of indexes that closely resemble the usage profile of the washing machine. Figure 7.6 shows the usage profile with the discovered motif set highlighted in red. The indexes are shown in Table 7.1. The 2-motif set found by the algorithm correctly identifies all instances of the washing machine in this usage profile, and no other devices.

The precision is 1 (no false positives), and for the washing machine device, sensitivity is also 1 (no false negatives). The sensitivity measured over all devices is 0.21; while this may appear to be fairly poor, it is better than some of the results obtained on the synthetic data.

The electricity-usage problem has an added level of complexity because the motif

Table 7.1: Indexes for the 2-motif, returned by the Set Finder algorithm on the data shown in figure 7.6, and the starting positions of the washing machine device.

Indexes
2-Motif: {19, 57, 132, 157, 195, 228}
Washing machine: {13, 56, 131, 156, 195, 226}

sets representing different devices contain subsequences of different lengths; this necessitates variation in the values of n and r, and explains why the algorithm found one device perfectly, while missing the others. An appropriate approach for such data would involve producing output for many values of n and r, and post-processing the discovered motif sets to find the set of devices in the data.

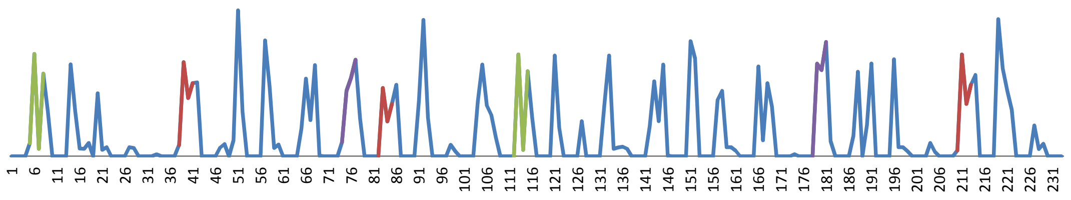

Figure 7.7: A household electricity-usage profile; the subsequences returned by the Scan MK algorithm are highlighted as follows: 4-motif set (red) and 5-motif set (green), are instances of the dishwasher device. The 6-motif set (purple) is two instances of the oven device.

We turn now to the performance of the Scan MK algorithm. Again, we use fixed values of $n = 4$ and $r = 0.5$. As identified by the algorithm, the 1-motif set is largely 0 elements. As with the Set Finder algorithm, we disregard this set. We also disregard the 2-motif set and 3-motif set, as they contains very similar data that would be post-processed as belonging with the 1-motif set. The other sets of indexes returned by the algorithm (see Table 7.2) are interpretable as follows. The algorithm has been reasonably successful at finding the dishwasher device, although post-processing would be required to combine the 4-motif set (highlighted in red on Figure 7.7) and the 5-motif set (highlighted on Figure 7.7 in green). The precision of the combined set is 1 (no false positives); the sensitivity for the dishwasher is 0.56. The overall

Table 7.2: Indexes for the sets returned by the Scan MK algorithm on the data shown in figure 7.7, and the starting positions of the dishwasher device.

Indexes
4-Motif: {37, 81, 209}
5-Motif: {4, 111}
6-Motif: {73, 177}
Dishwasher:{4, 37, 64, 82, 112, 141, 165, 186, 209}

sensitivity is 0.24; this value includes the two oven devices identified as the 6-motif set (highlighted in purple in Figure 7.7). It should be noted that the device-specific sensitivity of Scan MK was lower than that of Set Finder, even though Scan MK benefited from generous post-processing.

7.5 Conclusion

Finding motif sets in time series is essentially a form of clustering, and it is necessary to define a heuristic search technique to find motif sets, as the problem is NP-complete. We have proposed and compared three such algorithms for motif-set discovery: Scan MK, Cluster MK, and Set Finder. Extensive experimentation shows that Set Finder is significantly more accurate than Scan MK on synthetic data containing one and two shapes, for the values of the range parameter r for which the algorithms perform best. Cluster MK is competitive providing that appropriate values of r are used; however, it is very sensitive to the value of r.

We have extended our experiments to investigate the problem of profiling device usage from household electricity-consumption data. We found the motif set approach showed promise for identifying specific devices from data; we can reasonably expect to improve this performance dramatically on less aggregated data, and by using varying values of n and r followed by post-processing.

Our goal is to investigate the appropriateness of a representation based on local similarity of shape for analysis of repeated patterns in time series. We show the effectiveness of such a representation for the supervised tasks of classification and partial classification; an extension to unsupervised clustering of patterns is a logical progression. There is considerable overlap in the techniques required to use local similarity of shape in either context. For example, eliminating self-similar shapelets is closely related to avoiding trivially matched motifs. We consider shapelets and motifs to be two examples of the same conceptual insight applied to data-mining problems: that of features based on local similarity of shape. They involve overcoming many of the same difficulties, and, we argue, represent two separate applications of the same methodology.

Chapter 8

Conclusions and Future Work

We show that time-series data mining using local similarity of shape offers both improved performance for data-mining tasks, and highly interpretable models.

We discuss several methods for mining time-series data using an approach based on local similarity of shape. We show that classification using local similarity of shape, through our shapelet transform, offers improved accuracy over the benchmark method, 1NNDTW. This accuracy is improved for certain classes by using partial classification based on association rules mined from the shapelet-transformed data. Approximately repeated patterns can be mined from time-series data using a similar intuition to our shapelet approach, one that focuses on subsequences of the data.

Throughout the thesis, we focus not only on quantitative measures like accuracy, but also on improving the interpretability of our models, something that, for time-series data, is best offered by local similarity of shape. We cluster shapelets to reduce dimensionality, transform the clustered data into binary data to aid comprehension, and reduce rule set size to offer the most compact and interpretable model possible.

8.1 Evaluation

In Section 1.1, we pose the overall objective of the thesis in terms of the following question: how best can we use methods based on local similarity of shape for mining time-series data? We answer this question in terms of the objectives listed in Section 1.1.1:

1. *Develop and test subsequence-based representations for time-series.* We have developed and tested two subsequence-based representations for time series: a shapelet transform for time series classification, and a representation based on frequently-recurring subsequences (motifs) for unsupervised mining of patterns from time series.

2. *Create and refine algorithms and methods for discovering and extracting subsequences to represent locally-similar features.* We make a number of refinements to the shapelet transform, and create and test three algorithms for extracting motifs from time-series data.

3. *Implement and test approaches to make best use of the representations for solving specific data-mining problems.* We propose and thoroughly test an ensemble classifier that provides highly accurate classification of time-series data, and improve it with the addition of partial classification for poorly predicted classes. We also compare our three motif-finding algorithms in terms of speed and performance on both synthetic and real data.

4. *Design methods and representations that maximise interpretability without substantially diminishing performance.* We propose, implement, and test dimensionality-reduction methods, including filtering shapelets, clustering shapelets, and Brute-Suppression of rule sets, that improve interpretability without compromising

accuracy. We also propose a representation, binary shapelets, that maximises interpretability with only minor loss of accuracy in most cases. Our partial classification rule-based approach offers a highly comprehensible model that is no less accurate than the ensemble classifier for predicting membership of individual classes for low-dimensionality problems.

In achieving the objectives of the thesis, we overcome the three challenges described in Section 1.1.2; our new methods offer improved accuracy (Challenge 1; for example, shapelet transform with ensemble classifier, partial classification, Set Finder algorithm), improved interpretability (Challenge 2; for example, MDLStopCE clustering, binary shapelets, BruteSuppression, partial classification), and acceptable time complexity (Challenge 3; for example, shapelet transform with ensemble classifier, MDLStopCE clustering, BruteSuppression).

In answering the overriding research question, we make a number of novel contributions, listed in the next section.

8.2 Novel contributions

We have made a number of novel contributions to time-series data mining using local-shape-based similarity:

- We have proposed and extensively tested a state-of-the-art ensemble classifier on shapelet-transformed data, which provides better accuracy than the benchmark for TSC (1NNDTW), and existing methods that use the shapelet approach.

- We have proposed and tested a novel, parameterless method of clustering shapelets, MDLStopCE, that reduces the dimensionality of shapelet-transformed data without compromising accuracy.

- We have proposed a binary transform for shapelet-transformed data to enhance interpretability.
- We have used association rules for partial classification of shapelet-transformed data, improving accuracy for poorly predicted classes of lower-dimensionality data, and providing a highly interpretable model.
- We have shown that twelve commonly used interestingness measures are redundant for partial classification, as they impose the same ordering on a rule set as confidence.
- We have proposed and tested an algorithm, BruteSuppression, that substantially reduces the size of partial classification rule sets without negatively affecting the performance of the rule set, or causing large alterations in the distribution of rules in the rule set.
- We have described three novel algorithms for mining sets of approximately repeated patterns in time-series data, and tested them on synthetic data and on a real-world device disambiguation problem from electricity-consumption data.

8.3 Future directions

8.3.1 Extending the shapelet transform

There are a number of extensions that could be made to the shapelet transform. For example, k, the maximum number of shapelets used, could be indexed to the training data. The algorithm could also be modified to ensure that it delivers shapelets taken from each class; the proportion of shapelets from a given class could be weighted to the base incidence rate in the training data, or selected depending on other considerations (a particular class of interest could be allotted a greater proportion of the shapelets, for example). Approximate shapelets are another area that might offer considerable improvement; combining the shapelet transform with a bespoke search method for discovering good (but not optimal) shapelets could grant enormous speed increases without being overly detrimental to accuracy.

8.3.2 Extending shapelet clustering

Our MDL-based approach for clustering finds, we claim, the correct number of shapelets for a dataset. We feel that this clustering method could be extended to other time-series clustering problems, perhaps in conjunction with a repeated pattern mining approach. It is applicable to any problem where time series are clustered, and would be interesting to compare to existing time-series clustering methods.

8.3.3 Extending the partial classification framework

We have shown that partial classification with association rules can improve accuracy on poorly predicted classes and provide a comprehensible model of the relationship between the binary shapelets and the class label. The major weakness of the approach is that Apriori cannot be used easily with high-dimensionality data, and removing large numbers of shapelets is detrimental to performance. Hence, one direction in which the approach could be improved is by testing different rule induction algorithms with better time complexity in the attribute space, or developing a bespoke algorithm for use with shapelet-transformed data. We suspect that the results found on the low-dimensionality data could be extended productively to all shapelet-transformed datasets.

Another useful extension to the approach would be to employ an implementation of Apriori, or any rule induction algorithm, that could use real-valued data. This would sacrifice some interpretability; the interesting question is whether the performance of the partial classification would be improved by the additional information contained in the data.

8.3.4 Extending motif discovery

There are two directions in which the motif discovery research we have conducted could be extended. One extension would be to make use of the motifs as primitives in some other process, and use the results of that process to assess how well the algorithms are finding motifs. For example, after finding the motifs in a time series, rule induction could be used on the motifs to predict future behaviour.

Another potential extension would be to apply the algorithms to real data from a wider variety of domains. Noise makes an enormous difference to the relative speeds of the algorithms, and it would be interesting to examine this effect further. In addition, a wider variety of problems would allow us to extend what we found on synthetic data to real-world data, and see if the differences between the algorithms are similar for the different types of problem.

Bibliography

[1] *C. elegans* behavioural database. `http://wormbehavior.mrc-lmb.cam.ac.uk/`.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

[3] R. Agrawal, R. Srikant, et al. Fast Algorithms for Mining Association Rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499, 1994.

[4] K. Ali, S. Manganaris, and R. Srikant. Partial Classification using Association Rules. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 115–118, 1997.

[5] M. Alonso, S. González, J. Villar, J. Sedano, J. Terán, E. Ordax, and M. Coma. Data Analysis for Detecting a Temporary Breath Inability Episode. In *Intelligent Data Engineering and Automated Learning–IDEAL 2014*, pages 126–133. Springer, 2014.

[6] A. Bagnall and L. Davis. Predictive Modelling of Bone Age through Classification and Regression of Bone Shapes. *arXiv preprint arXiv:1406.4781*, 2014.

[7] A. Bagnall, L. Davis, J. Hills, and J. Lines. Transformation based ensembles for time series classification. In *SDM*, volume 12, pages 307–318. SIAM, 2012.

[8] A. Bagnall and G. Janacek. Clustering Time Series from ARMA Models with Clipped Data. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 49–58. ACM, 2004.

[9] A. Bagnall, J. Lines, J. Hills, and A. Bostrom. Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles. `https://ueaeprints.uea.ac.uk/id/eprint/49614`.

[10] J.L. Balcazar. Confidence Width: An Objective Measure for Association Rule Novelty. In *Workshop on Quality Issues, Measures of Interestingness and Evaluation of Data Mining Models QIMIE*, volume 9, pages 5–16, 2009.

[11] E. Baranauskas and P. Maginde. Time Series Classification Problems from Smart Metering Devices. Master's thesis, School of Computing Sciences, University of East Anglia, 2013.

[12] G. Batista, X. Wang, and E. Keogh. A complexity-invariant distance measure for time series. In *SDM*, volume 11, pages 699–710. SIAM, 2011.

[13] E. Bauer and R. Kohavi. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning*, 36(1-2):105–139, 1999.

[14] R.J. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based Rule Mining in Large, Dense Databases. *Data Mining and Knowledge Discovery*, 4(2):217–240, 2000.

[15] R.J. Bayardo Jr and R. Agrawal. Mining the Most Interesting Rules. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 145–154. ACM, 1999.

[16] N. Begum, B. Hu, T. Rakthanmanon, and E. Keogh. Towards a Minimum Description Length Based Stopping Criterion for Semi-supervised Time Series Classification. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, pages 333–340. IEEE, 2013.

[17] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic Classification on the Fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[18] D. Berndt and J. Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In *KDD Workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[19] S. Bian and W. Wang. On Diversity and Accuracy of Homogeneous and Heterogeneous Ensembles. *International Journal of Hybrid Intelligent Systems*, 4(2):103–128, 2007.

[20] J. Blanchard, F. Guillet, and P. Kuntz. Semantics-based Classification of Rule Interestingness Measures. *Post-mining of Association Rules: Techniques for Effective Knowledge Extraction*, page 56, 2009.

[21] M. Bober. MPEG-7 Visual Shape Descriptors. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(6):716–719, 2001.

[22] L. Breiman. Bagging Predictors. *Machine learning*, 24(2):123–140, 1996.

[23] L. Breiman. Bias, Variance, and Arcing Classifiers. *Tech. Rep. 460, Statistics Department, University of California, Berkeley, CA, USA*, 1996.

[24] L. Breiman. Random Forests. *Machine learning*, 45(1):5–32, 2001.

[25] L. Breiman, J. Friedman, C. Stone, and R. Olshen. *Classification and Regression Trees*. CRC press, 1984.

[26] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In *ACM SIGMOD Record*, volume 26, pages 255–264. ACM, 1997.

[27] A. Brown, E. Yemini, L. Grundy, T. Jucikas, and W. Schafer. A Dictionary of Behavioral Motifs Reveals Clusters of Genes Affecting *Caenorhabditis elegans* Locomotion. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 10(2):791–796, 2013.

[28] J. Buhler and M. Tompa. Finding Motifs using Random Projections. *Journal of Computational Biology*, 9(2):225–242, 2002.

[29] K. Buza. *Fusion Methods for Time-Series Classification.* PhD thesis, University of Hildesheim, Germany, 2011.

[30] J. Caiado, N. Crato, and D. Peña. A Periodogram-based Metric for Time Series Classification. *Computational Statistics & Data Analysis*, 50(10):2668–2684, 2006.

[31] S.E. Campana and J.M. Casselman. Stock Discrimination using Otolith Shape Analysis. *Canadian Journal of Fisheries and Aquatic Sciences*, 50(5):1062–1083, 1993.

[32] D. Carvalho, A. Freitas, and N. Ebecken. A Critical Review of Rule Surprisingness Measures. In *Proc. Data Mining IV-Int. Conf. on Data Mining*, volume 7, pages 545–556. WIT Press, 2003.

[33] D. Carvalho, A. Freitas, and N. Ebecken. Evaluating the Correlation between Objective Rule Interestingness Measures and Real Human Interest. *Knowledge Discovery in Databases: PKDD 2005*, pages 453–461, 2005.

[34] K.W. Chang, B. Deka, W.M. Hwu, and D. Roth. Efficient Pattern-Based Time Series Classification on GPU. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 131–140. IEEE, 2012.

[35] P. Clark and R. Boswell. Rule Induction with CN2: Some Recent Improvements. In *Machine learning–EWSL–91*, pages 151–163. Springer, 1991.

[36] F. Coenen, G. Goulbourne, and P. Leng. Tree Structures for Mining Association Rules. *Data Mining and Knowledge Discovery*, 8(1):25–51, 2004.

[37] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J.D. Ullman, and C. Yang. Finding Interesting Associations without Support Pruning. *Knowledge and Data Engineering, IEEE Transactions on*, 13(1):64–78, 2001.

[38] C. Cortes and V. Vapnik. Support-vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[39] G. Das, K.I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule Discovery from Time Series. *Knowledge Discovery and Data Mining*, pages 16–22, 1998.

[40] L. Davis, B.J. Theobald, and A. Bagnall. Automated Bone Age Assessment using Feature Extraction. In *Intelligent Data Engineering and Automated Learning-IDEAL 2012*, pages 43–51. Springer, 2012.

[41] L. Davis, B.J. Theobald, J. Lines, A. Toms, and A. Bagnall. On the Segmentation and Classification of Hand Radiographs. *International Journal of Neural Systems*, 22(05):1250020–1 – 1250020–16, 2012.

[42] L. Davis, B.J. Theobald, A. Toms, and A. Bagnall. On the Extraction and Classification of Hand Outlines. In *Proceedings of the 12th International Conference on Intelligent Data Engineering and Automated Learning*, pages 92–99. Springer-Verlag, 2011.

[43] B. De La Iglesia, M. Philpott, A. Bagnall, and V. Rayward-Smith. Data Mining Rules using Multi-objective Evolutionary Algorithms. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 3, pages 1552–1559. IEEE, 2003.

[44] B. De La Iglesia, G. Richards, M. Philpott, and V. Rayward-Smith. The Application and Effectiveness of a Multi-objective Metaheuristic Algorithm for Partial Classification. *European Journal of Operational Research*, 169(3):898–917, 2006.

[45] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[46] H. Deng, G. Runger, E. Tuv, and M. Vladimir. A Time Series Forest for Classification and Feature Extraction. *Information Sciences*, 239:142–153, 2013.

[47] D.A. DeVries, C.B. Grimes, and M.H. Prager. Using Otolith Shape Analysis to Distinguish Eastern Gulf of Mexico and Atlantic Ocean Stocks of King Mackerel. *Fisheries Research*, 57(1):51–62, 2002.

[48] V. Dhar and A. Tuzhilin. Abstract-driven Pattern Discovery in Databases. *IEEE Transactions on Knowledge and Data Engineering*, pages 926–938, 1993.

[49] G. Di Fatta, S. Leue, and E. Stegantova. Discriminative Pattern Mining in Software Fault Detection. In *Proceedings of the 3rd International Workshop on Software Quality Assurance*, pages 62–69. ACM, 2006.

[50] T. Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40(2):139–157, 2000.

[51] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008.

[52] P. Domingos. A Unified Bias-variance Decomposition for Zero-one and Squared loss. *AAAI/IAAI*, 2000:564–569, 2000.

[53] P. Duarte-Neto, R. Lessa, B. Stosic, and E. Morize. The Use of Sagittal Otoliths in Discriminating Stocks of Common Dolphinfish (*Coryphaena hippurus*) off Northeastern Brazil using Multishape Descriptors. *ICES Journal of Marine Science: Journal du Conseil*, 65(7):1144–1152, 2008.

[54] R. Durbin. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[55] A. Edgcomb and F. Vahid. Automated Fall Detection on Privacy-enhanced Video. In *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE*, pages 252–255. IEEE, 2012.

[56] U. Fayyad, C. Reina, and P.S. Bradley. Initialization of Iterative Refinement Clustering Algorithms. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 194–198, 1998.

[57] A. Frank, A. Asuncion, et al. UCI Machine Learning Repository. 2010.

[58] A. Freitas. On Rule Interestingness Measures. *Knowledge-based Systems*, 12(5-6):309–315, 1999.

[59] Y. Freund and R. Schapire. Experiments with a New Boosting Algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[60] J. Friedman. On Bias, Variance, 0/1 Loss, and the Curse-of-Dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[61] M. Friedman. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, 32(1):675–701, 1937.

[62] M. Friedman. A Comparison of Alternative Tests of Significance for the Problem of m Rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[63] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[64] J. Froehlich, E. Larson, S. Gupta, G. Cohn, M. Reynolds, and S. Patel. Disaggregated End-use Energy Sensing for the Smart Grid. *IEEE Pervasive Computing*, 10(1):28–39, 2011.

[65] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data Mining using Two-dimensional Optimized Association Rules: Scheme, Algorithms, and Visualization. In *ACM SIGMOD Record*, volume 25, pages 13–23. ACM, 1996.

[66] J. Furnkranz and P.A. Flach. Roc 'n' Rule Learning – Towards a Better Understanding of Covering Algorithms. *Machine Learning*, 58(1):39–77, 2005.

[67] J.G. Ganascia. Deriving the Learning Bias from Rule Properties. In *Machine intelligence 12*, pages 151–167. Clarendon Press, 1991.

[68] F. Gebhardt. Choosing among Competing Generalizations. *Knowledge Acquisition*, 3(4):361–380, 1991.

[69] L. Geng and H.J. Hamilton. Interestingness Measures for Data Mining: A Survey. *ACM Computing Surveys (CSUR)*, 38(3):9, 2006.

[70] P. Geurts, D. Ernst, and L. Wehenkel. Extremely Randomized Trees. *Machine Learning*, 63(1):3–42, 2006.

[71] M. Ghalwash, V. Radosavljevic, and Z. Obradovic. Extraction of Interpretable Multivariate Patterns for Early Diagnostics. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 201–210. IEEE, 2013.

[72] L.A. Goodman and W.H. Kruskal. Measures of Association for Cross Classifications. *Journal of the American Statistical Association*, 49(268):732–764, 1954.

[73] D. Gordon, D. Hendler, and L. Rokach. Fast Randomized Model Generation for Shapelet-Based Time Series Classification. *arXiv preprint arXiv:1209.5038*, 2012.

[74] T. Górecki and M. Łuczak. Using Derivatives in Time Series Classification. *Data Mining and Knowledge Discovery*, 26(2):310–331, 2013.

[75] J. Grabocka and L. Schmidt-Thieme. Invariant Time-series Factorization. *Data Mining and Knowledge Discovery*, pages 1–25, 2014.

[76] Y. Grandvalet, S. Canu, and S. Boucheron. Noise Injection: Theoretical Prospects. *Neural Computation*, 9(5):1093–1108, 1997.

[77] M.P. Griffin and J.R. Moorman. Toward the Early Diagnosis of Neonatal Sepsis and Sepsis-like Illness using Novel Heart Rate Analysis. *Pediatrics*, 107(1):97–104, 2001.

[78] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA Data Mining Software: an Update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[79] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2006.

[80] L. Hansen and P. Salamon. Neural Network Ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[81] B. Hartmann and N. Link. Gesture Recognition with Inertial Sensors and Optimized DTW Prototypes. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 2102–2109. IEEE, 2010.

[82] B. Hartmann, I. Schwab, and N. Link. Prototype Optimization for Temporarily and Spatially Distorted Time Series. In *The AAAI Spring Symposia*, pages 15–20.

[83] G. He, Y. Duan, R. Peng, X. Jing, T. Qian, and L. Wang. Early Classification on Multivariate Time Series. *Neurocomputing*, pages 777–787, 2014.

[84] Q. He, Z. Dong, F. Zhuang, T. Shang, and Z. Shi. Fast Time Series Classification Based on Infrequent Shapelets. In *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, volume 1, pages 215–219. IEEE, 2012.

[85] J. Hills. `https://sites.google.com/site/miningtimeseriesdata/`.

[86] J. Hills, A. Bagnall, B. De La Iglesia, and G. Richards. BruteSuppression: a Size Reduction Method for Apriori Rule Sets. *Journal of Intelligent Information Systems*, 40:431–454, 2013.

[87] J. Hills, L. Davis, and A. Bagnall. Interestingness Measures for Fixed Consequent Rules. In *Proceedings of the 13th international conference on Intelligent Data Engineering and Automated Learning*, pages 68–75. Springer-Verlag, 2012.

[88] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall. Classification of Time Series by Shapelet Transformation. *Data Mining and Knowledge Discovery*, 28(4):851–881, 2014.

[89] T. Ho. The Random Subspace Method for Constructing Decision Forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, 1998.

[90] F. Höppner. Discovery of Temporal Patterns. *Principles of Data Mining and Knowledge Discovery*, pages 192–203, 2001.

[91] B. Hu, Y. Chen, and E. Keogh. Time series classification under more realistic assumptions. pages 578–586, 2013.

[92] B. Hu, T. Rakthanmanon, Y. Hao, S. Evans, S. Lonardi, and E. Keogh. Using the Minimum Description Length to Discover the Intrinsic Cardinality and Dimensionality of Time Series. *Data Mining and Knowledge Discovery*, pages 1–42, 2014.

[93] F. Hussain, H. Liu, E. Suzuki, and H. Lu. Exception Rule Mining with a Relative Interestingness Measure. *Knowledge Discovery and Data Mining. Current Issues and New Applications*, pages 86–97, 2000.

[94] X.H. Huynh, F. Guillet, and H. Briand. Evaluating Interestingness Measures with Linear Correlation Graph. *Advances in Applied Artificial Intelligence*, pages 312–321, 2006.

[95] I.B.M. IBM Intelligent Miner User's Guide, Version 1 Release 1. Technical report, SH12-6213-00 edition, July 1996, 1996.

[96] Image Processing and Informatics Lab, University of Southern California. The Digital Hand Atlas Database System. `http://www.ipilab.org/BAAweb/`.

[97] G. James. Variance and Bias for General Loss Functions. *Machine Learning*, 51(2):115–135, 2003.

[98] G. Janacek, A. Bagnall, and M. Powell. A likelihood ratio distance measure for the similarity between the fourier transform of time series. In *Advances in Knowledge Discovery and Data Mining*, pages 737–743. Springer, 2005.

[99] S. Jeong, M. Jeong, and O. Omitaomu. Weighted Dynamic Time Warping for Time Series Classification. *Pattern Recognition*, 44(9):2231–2240, 2011.

[100] E. Keogh and J. Lin. Clustering of Time-series Subsequences is Meaningless: Implications for Previous and Future Research. *Knowledge and Information Systems*, 8(2):154–177, 2005.

[101] E. Keogh and M. Pazzani. Derivative Dynamic Time Warping. In *SDM*, volume 1, pages 5–7. SIAM, 2001.

[102] M.H. Ko, G. West, S. Venkatesh, and M. Kumar. Online Context Recognition in Multisensor Systems using Dynamic Time Warping. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005. Proceedings of the 2005 International Conference on*, pages 283–288. IEEE, 2005.

[103] W.H. Kruskal. A Nonparametric Test for the Several Sample Problem. *The Annals of Mathematical Statistics*, 23(4):525–540, 1952.

[104] L. Kuncheva. A Theoretical Study on Six Classifier Fusion Strategies. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):281–286, 2002.

[105] L. Kuncheva. Diversity in Multiple Classifier Systems. *Information fusion*, 6(1):3–4, 2005.

[106] L. Kuncheva, C. Whitaker, C. Shipp, and R. Duin. Limits on the Majority Vote Accuracy in Classifier Fusion. *Pattern Analysis & Applications*, 6(1):22–31, 2003.

[107] D. Lagun, M.l Ageev, Q. Guo, and E. Agichtein. Discovering Common Motifs in Cursor Movement Data for Improving Web Search. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, pages 183–192. ACM, 2014.

[108] L.J. Latecki, R. Lakamper, and T. Eckhardt. Shape Descriptors for Non-rigid Shapes with a Single Closed Contour. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 424–429. IEEE, 2000.

[109] N. Lavrač, P. Flach, and B. Zupan. Rule Evaluation Measures: a Unifying View. *Inductive Logic Programming*, pages 174–185, 1999.

[110] P. Lenca, P. Meyer, B. Vaillant, and S. Lallich. On Selecting Interestingness Measures for Association Rules: User Oriented Description and Multiple Criteria Decision Aid. *European Journal of Operational Research*, 184(2):610–626, 2008.

[111] P. Lenca, B. Vaillant, P. Meyer, and S. Lallich. Association Rule Interestingness Measures: Experimental and Theoretical Studies. *Quality Measures in Data Mining*, pages 51–76, 2007.

[112] W. Li, J. Han, and J. Pei. CMAR: Accurate and Efficient Classification Based on Multiple Class-association Rules. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 369–376. IEEE, 2001.

[113] Z. Li, C. Lin, B. Ding, and J. Han. Mining Significant Time Intervals for Relationship Detection. *Advances in Spatial and Temporal Databases*, pages 386–403, 2011.

[114] J. Lin, E. Keogh, S. Lonardi, and P. Patel. Finding Motifs in Time Series. *Proc. of the 2nd Workshop on Temporal Data Mining*, pages 53–68, 2002.

[115] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a Novel Symbolic Representation of Time Series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[116] J. Lines and A. Bagnall. Alternative Quality Measures for Time Series Shapelets. In *Intelligent Data Engineering and Automated Learning (IDEAL)*, volume 7435 of *Lecture Notes in Computer Science*, pages 475–483. 2012.

[117] J. Lines and A. Bagnall. Time Series Classification with Ensembles of Elastic Distance Measures. *Data Mining and Knowledge Discovery*, pages 1–28, 2014.

[118] J. Lines, A. Bagnall, P. Caiger-Smith, and S. Anderson. Classification of Household Devices by Electricity Usage Profiles. In *Intelligent Data Engineering and Automated Learning-IDEAL 2011*, pages 403–412. Springer, 2011.

[119] J. Lines, L Davis, J. Hills, and A. Bagnall. A Shapelet Transform for Time Series Classification. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 289–297. ACM, 2012.

[120] H. Liu, L. Liu, and H. Zhang. A Fast Pruning Redundant Rule Method using Galois Connection. *Applied Soft Computing*, 11(1):130–137, 2011.

[121] H. Liu, H. Lu, L. Feng, and F. Hussain. Efficient Search of Reliable Exceptions. *Methodologies for Knowledge Discovery and Data Mining*, pages 194–204, 1999.

[122] J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uWave: Accelerometer-based Personalized Gesture Recognition and its Applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.

[123] R. Maclin and D. Opitz. Popular Ensemble Methods: an Empirical Study. *arXiv preprint arXiv:1106.0257*, 2011.

[124] J.A. Major and J.J. Mangano. Selecting among Rules Induced from a Hurricane Database. *Journal of Intelligent Information Systems*, 4(1):39–52, 1995.

[125] A. McGovern, D.H. Rosendahl, R.A. Brown, and K.K. Droegemeier. Identifying Predictive Multi-dimensional Time Series Motifs: an Application to Severe Weather Prediction. *Data Mining and Knowledge Discovery*, 22(1):232–258, 2011.

[126] S. McMillan, C. Chia, A. Van Esbroeck, I. Rubinfeld, and Z. Syed. ICU Mortality Prediction using Time Series Motifs. *Computing in Cardiology 2012 Krakow, Poland*, 39:265–268, 2012.

[127] R. Meir and G. Rätsch. An Introduction to Boosting and Leveraging. In *Advanced Lectures on Machine Learning*, pages 118–183. Springer, 2003.

[128] A.M.F. Mood. Introduction to the Theory of Statistics. 1950.

[129] Y. Morimoto, T. Fukuda, H. Matsuzawa, T. Tokuyama, and K. Yoda. Algorithms for Mining Association Rules for Binary Segmentations of Huge Categorical Databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 380–391. Citeseer, 1998.

[130] A. Mueen, E. Keogh, and N. Young. 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. ACM, 2011.

[131] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover. Exact Discovery of Time Series Motifs. In *Proceedings of the SIAM International Conference on Data Mining*, pages 473–484. Citeseer, 2009.

[132] A. Mueen, E. Keogh, Q. Zhu, S.S. Cash, M.B. Westover, and N. Bigdely-Shamlo. A Disk-aware Algorithm for Time Series Motif Discovery. *Data Mining and Knowledge Discovery*, 22(1):73–105, 2011.

[133] P. Nemenyi. Distribution-free Multiple Comparisons. In *Biometrics*, volume 18, page 263. International Biometric Society, 1962.

[134] T. Oates, M. Schmill, and P. Cohen. A Method for Clustering the Experiences of a Mobile Robot that Accords with Human Judgments. In *AAAI/IAAI*, pages 846–851, 2000.

[135] M. Ohsaki, H. Abe, S. Tsumoto, H. Yokoi, and T. Yamaguchi. Evaluation of Rule Interestingness Measures in Medical Knowledge Discovery in Databases. *Artificial Intelligence in Medicine*, 41(3):177–196, 2007.

[136] M. Ohsaki, S. Kitaguchi, K. Okamoto, H. Yokoi, and T. Yamaguchi. Evaluation of Rule Interestingness Measures with a Clinical Dataset on Hepatitis. *Knowledge Discovery in Databases: PKDD 2004*, pages 362–373, 2004.

[137] P. Owen. Powering the Nation; Household electricity-using habits revealed. A report by the Energy Saving Trust, the Department of Energy and Climate Change (DECC), and the Department for Environment. *Food and Rural Affairs (Defra)*, 2012.

[138] G. Piatetsky-Shapiro and W. Frawley. *Knowledge Discovery in Databases*. AAAI press, 1991.

[139] F.J. Provost and J.M. Aronis. Scaling up Inductive Learning with Massive Parallelism. *Machine Learning*, 23(1):33–46, 1996.

[140] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 262–270. ACM, 2012.

[141] T. Rakthanmanon and E. Keogh. Fast Shapelets: a Scalable Algorithm for Discovering Time Series Shapelets. *Proc. 13th SDM*, pages 668–676, 2013.

[142] T. Rakthanmanon, E. Keogh, S. Lonardi, and S. Evans. Time Series Epenthesis: Clustering Time Series Streams Requires Ignoring Some Data. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 547–556. IEEE, 2011.

[143] T. Rakthanmanon, Q. Zhu, and E. Keogh. Mining Historical Documents for Near-Duplicate Figures. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 557–566. IEEE, 2011.

[144] C. Ratanamahatana and E. Keogh. Three Myths about Dynamic Time Warping Data Mining. In *Proceedings of SIAM International Conference on Data Mining (SDM05)*, pages 506–510. SIAM, 2005.

[145] G. Rätsch, T. Onoda, and K.R. Müller. Soft Margins for AdaBoost. *Machine Learning*, 42(3):287–320, 2001.

[146] A. Reiss, M. Weber, and D. Stricker. Exploring and Extending the Boundaries of Physical Activity Recognition. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 46–50. IEEE, 2011.

[147] A. Reynolds and B. De La Iglesia. Rule induction using Multi-objective Metaheuristics: Encouraging Rule Diversity. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 3343–3350. IEEE, 2006.

[148] G. Richards and V. Rayward-Smith. Discovery of Association Rules in Tabular Data. In *Proceedings of the IEEE International Conference on Data Mining*, page 465. IEEE Computer Society, 2001.

[149] G. Richards and V. Rayward-Smith. The Discovery of Association Rules from Tabular Databases Comprising Nominal and Ordinal Attributes. *Intelligent Data Analysis*, 9(3):289–307, 2005.

[150] J. Rodriguez and C. Alonso. Support vector machines of interval-based features for time series classification. *Knowledge-Based Systems*, 18(4):171–178, 2005.

[151] J. Rodriguez, L. Kuncheva, and C. Alonso. Rotation Forest: a New Classifier Ensemble Method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1619–1630, 2006.

[152] S.L. Salzberg. On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. *Data mining and Knowledge Discovery*, 1(3):317–328, 1997.

[153] P.K.D. Sarma and A.K. Mahanta. Reduction of number of association rules with inter itemset distance in transaction databases. *International Journal of Database Management Systems*, 4(5):61–82, 2012.

[154] R. Schapire. Theoretical Views of Boosting and Applications. In *Algorithmic Learning Theory*, pages 13–25. Springer, 1999.

[155] M. Sebag and M. Schoenauer. Generation of Rules with Certainty and Confidence Factors from Incomplete and Incoherent Learning Bases. In *Proc. of EKAW*, volume 88, pages 1–28, 1988.

[156] I.N.M. Shaharanee, F. Hadzic, and T.S. Dillon. Interestingness Measures for Association Rules Based on Statistical Validity. *Knowledge-based Systems*, 24(3):386–392, 2011.

[157] T. Shajina and P.B. Sivakumar. Human Gait Recognition and Classification Using Time Series Shapelets. In *Advances in Computing and Communications (ICACC), 2012 International Conference on*, pages 31–34. IEEE, 2012.

[158] C.E. Shannon, W. Weaver, R.E. Blahut, and B. Hajek. *The Mathematical Theory of Communication*. University of Illinois press Urbana, 1949.

[159] C. Stransky. Geographic Variation of Golden Redfish (*Sebastes marinus*) and Deep-sea Redfish (*S. mentella*) in the North Atlantic Based on Otolith Shape Analysis. *ICES Journal of Marine Science: Journal du Conseil*, 62(8):1691–1698, 2005.

[160] R. Tamir and Y. Singer. On a Confidence Gain Measure for Association Rule Discovery and Scoring. *The VLDB Journal*, 15(1):40–52, 2006.

[161] P.N. Tan, V. Kumar, and J. Srivastava. Selecting the Right Interestingness Measure for Association Patterns. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 32–41. ACM, 2002.

[162] E.K. Tang, P.N. Suganthan, and X. Yao. An Analysis of Diversity Measures. *Machine Learning*, 65(1):247–271, 2006.

[163] R. Tibshirani. *Bias, Variance and Prediction Error for Classification Rules.* Citeseer, 1996.

[164] UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml/datasets.html`.

[165] UCR Time-series Classification/Clustering Page. `http://www.cs.ucr.edu/ eamonn/time_series_data/`.

[166] D. Vail and M. Veloso. Learning from Accelerometer Data on a Legged Robot. In *Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, 2004.

[167] G. Valentini and T. Dietterich. Bias-variance Analysis of Support Vector Machines for the Development of SVM-based Ensemble Methods. *The Journal of Machine Learning Research*, 5:725–775, 2004.

[168] R. Walpole, R. Myers, S. Myers, and K. Ye. *Probability and Statistics for Engineers and Scientists.* Macmillan New York, 1993.

[169] L. Warren. Clustering of Time Series Data - a Survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[170] G. Webb. Multiboosting: a Technique for Combining Boosting and Wagging. *Machine Learning*, 40(2):159–196, 2000.

[171] L. Wei, E. Keogh, and X. Xi. SAXually Explicit Images: Finding Unusual Shapes. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 711–720. IEEE, 2006.

[172] F. Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, pages 80–83, 1945.

[173] Y. Wu, D. Agrawal, and A. El Abbadi. A Comparison of DFT and DTW Based Similarity Search in Time-series Databases. In *Proc. 9th ACM CIKM*, pages 488–495, 2000.

[174] X. Xi, E. Keogh, C. Shelton, L. Wei, and C.A. Ratanamahatana. Fast Time Series Classification using Numerosity Reduction. pages 1033–1040, 2006.

[175] Z. Xing, J. Pei, P. Yu, and K. Wang. Extracting Interpretable Features for Early Classification on Time Series. *the Proceedings of SDM*, pages 247–258, 2011.

[176] Z. Xing, J. Pei, and P.S. Yu. Early Classification on Time Series. *Knowledge and Information Systems*, 31(1):105–127, 2012.

[177] Y. Xu, Y. Li, and G. Shaw. Reliable Representations for Association Rules. *Data & Knowledge Engineering*, 70(6):555–575, 2011.

[178] J. Yao and H. Liu. Searching Multiple Databases for Interesting Complexes. *KDD: Techniques and Applications, World Scientific, Singapore*, 482:484–485, 1997.

[179] Y. Yao and N. Zhong. An Analysis of Quantitative Measures Associated with Rules. *Methodologies for Knowledge Discovery and Data Mining*, pages 479–488, 1999.

[180] L. Ye and E. Keogh. Time Series Shapelets: a New Primitive for Data Mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 947–956. ACM, 2009.

[181] L. Ye and E. Keogh. Time Series Shapelets: a Novel Technique that Allows Accurate, Interpretable and Fast Classification. *Data Mining and Knowledge Discovery*, 22(1):149–182, 2011.

[182] E. Yemini, T. Jucikas, L. Grundy, A. Brown, and W. Schafer. A Database of *Caenorhabditis elegans* Behavioral Phenotypes. *Nature Methods*, 10:877–879, 2013.

[183] J. Zakaria, A. Mueen, and E. Keogh. Clustering Time Series using Unsupervised-shapelets. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 785–794. IEEE, 2012.

Appendix

Table 8.1: Accuracies of ensemble classifier on shapelet-transformed data and 1NNDTW with warping window set by cross-validation on raw data, over 75 datasets.

Dataset	Ensemble classifier on shapelet-transformed data	1NNDTW
Adiac	0.565	0.611
ArrowHead	0.771	0.783
Beef	0.833	0.667
BeetleFly	0.750	0.650
BirdChicken	0.750	0.650
Car	0.733	0.767
CBF	0.997	0.994
ChlorineConcentration	0.700	0.625
CinC_ECG_torso	0.846	0.929
Coffee	1.000	1.000
Computers	0.700	0.876
Cricket_X	0.782	0.754
Cricket_Y	0.764	0.795
Cricket_Z	0.772	0.823
DiatomSizeReduction	0.876	0.925
DistalPhalanxOutlineAgeGroup	0.741	0.799
DistalPhalanxOutlineCorrect	0.736	0.746
DistalPhalanxTW	0.633	0.662
Earthquakes	0.734	0.691
ECGFiveDays	0.999	0.800
FaceAll	0.737	0.808
FaceFour	0.943	0.898
FacesUCR	0.913	0.909
fiftywords	0.719	0.765
fish	0.977	0.834
FordA	0.927	0.794
FordB	0.789	0.670
GunPoint	0.980	0.913
Haptics	0.477	0.406
Herrings	0.672	0.656
InlineSkate	0.385	0.385
ItalyPowerDemand	0.952	0.961
LargeKitchenAppliances	0.883	0.736
Lightning2	0.656	0.869
Lightning7	0.740	0.712
MALLAT	0.940	0.910
MedicalImages	0.604	0.739
MiddlePhalanxOutlineAgeGroup	0.630	0.461

Table 8.1: Accuracies of ensemble classifier on shapelet-transformed data and 1NNDTW with warping window set by cross-validation on raw data, over 75 datasets.

Dataset	Ensemble classifier on shapelet-transformed data	1NNDTW
MiddlePhalanxOutlineCorrect	0.725	0.801
MiddlePhalanxTW	0.539	0.494
MoteStrain	0.891	0.866
NonInvasiveFatalECG_Thorax1	0.900	0.804
NonInvasiveFatalECG_Thorax2	0.903	0.868
OliveOil	0.900	0.867
OSULeaf	0.715	0.599
PhalangesOutlinesCorrect	0.748	0.772
Plane	1.000	1.000
ProximalPhalanxOutlineAgeGroup	0.854	0.873
ProximalPhalanxOutlineCorrect	0.900	0.804
ProximalPhalanxTW	0.771	0.683
PtNDeviceGroups	0.798	0.554
PtNDevices	0.524	0.519
RefrigerationDevices	0.557	0.485
ScreenType	0.533	0.555
SimulatedSet	0.919	0.672
SmallKitchenAppliances	0.773	0.744
SonyAIBORobotSurface	0.933	0.699
SonyAIBORobotSurfaceII	0.885	0.857
StarLightCurves	0.976	0.903
SwedishLeaf	0.907	0.846
Symbols	0.886	0.931
SyntheticControl	0.983	0.983
ToeSegmentation1	0.956	0.899
ToeSegmentation2	0.854	0.892
Trace	0.980	0.990
TwoLeadECG	0.996	0.851
TwoPatterns	0.941	0.999
UWaveGestureLibrary_X	0.784	0.774
UWaveGestureLibrary_Y	0.697	0.698
UWaveGestureLibrary_Z	0.727	0.673
wafer	0.998	0.996
WordSynonyms	0.597	0.740
Worms	0.701	0.532
WormsTwoClass	0.766	0.584
yoga	0.805	0.843

Table 8.2: Accuracies of ensemble classifier on shapelet-transformed data, filtered shapelet-transformed data, and MDLStopCE clustered shapelet-transformed data, over 50 datasets.

Dataset	Shapelet Transform	Filtered data	MDLStopCE
ArrowHead	0.771	0.789	0.737
Beef	0.833	0.833	0.833
BeetleFly	0.750	0.750	0.750
BirdChicken	0.750	0.800	0.800
Car	0.733	0.733	0.750
CBF	0.997	0.996	0.976
CinC_ECG_torso	0.846	0.833	0.833
Coffee	1.000	1.000	1.000
Computers	0.700	0.716	0.716
Cricket_X	0.782	0.785	0.659
Cricket_Y	0.764	0.767	0.585
Cricket_Z	0.772	0.772	0.687
DiatomSizeReduction	0.876	0.876	0.876
DistalPhalanxTW	0.633	0.647	0.640
ECGFiveDays	0.999	0.992	0.995
FaceAll	0.737	0.740	0.688
FaceFour	0.943	0.932	0.989
FacesUCR	0.913	0.918	0.839
fiftywords	0.719	0.710	0.708
fish	0.977	0.971	0.971
GunPoint	0.980	0.980	0.953
Haptics	0.477	0.451	0.461
Herrings	0.672	0.688	0.672
InlineSkate	0.385	0.385	0.373
ItalyPowerDemand	0.952	0.957	0.951
Lightning2	0.656	0.623	0.623
Lightning7	0.740	0.726	0.740
MALLAT	0.940	0.933	0.828
MedicalImages	0.604	0.609	0.624
MiddlePhalanxOutlineAgeGroup	0.630	0.623	0.623
MiddlePhalanxTW	0.539	0.532	0.532
MoteStrain	0.891	0.887	0.866
OliveOil	0.900	0.900	0.900
OSULeaf	0.715	0.707	0.595
Plane	1.000	1.000	1.000
ProximalPhalanxOutlineAgeGroup	0.854	0.863	0.863
ProximalPhalanxTW	0.771	0.785	0.785
SimulatedSet	0.919	0.913	0.880
SonyAIBORobotSurface	0.933	0.938	0.943
SonyAIBORobotSurfaceII	0.885	0.880	0.887
SwedishLeaf	0.907	0.894	0.910
Symbols	0.886	0.884	0.884
SyntheticControl	0.983	0.980	0.903
ToeSegmentation1	0.956	0.952	0.930
ToeSegmentation2	0.854	0.838	0.815
Trace	0.980	0.980	0.980
TwoLeadECG	0.996	0.997	0.997
WordSynonyms	0.597	0.605	0.553
Worms	0.701	0.727	0.727
WormsTwoClass	0.766	0.792	0.688

Table 8.3: Accuracies of ensemble classifier on shapelet-transformed data clustered using MDL, MDLCE, MDLStop, and MDLStopCE, over 50 datasets.

Dataset	MDL	MDLCE	MDLStop	MDLStopCE
ArrowHead	0.737	0.731	0.777	0.737
Beef	0.733	0.800	0.833	0.833
BeetleFly	0.800	0.800	0.750	0.750
BirdChicken	0.750	0.700	0.800	0.800
Car	0.667	0.683	0.717	0.750
CBF	0.948	0.981	0.899	0.976
CinC_ECG_torso	0.696	0.689	0.833	0.833
Coffee	1.000	1.000	1.000	1.000
Computers	0.716	0.720	0.708	0.716
Cricket_X	0.303	0.646	0.241	0.659
Cricket_Y	0.385	0.582	0.246	0.585
Cricket_Z	0.244	0.692	0.244	0.687
DiatomSizeReduction	0.889	0.869	0.876	0.876
DistalPhalanxTW	0.669	0.633	0.640	0.640
ECGFiveDays	0.995	0.991	0.990	0.995
FaceAll	0.682	0.699	0.240	0.688
FaceFour	0.705	0.761	0.614	0.989
FacesUCR	0.735	0.860	0.310	0.839
fish	0.926	0.886	0.971	0.971
fiftywords	0.448	0.686	0.211	0.708
GunPoint	1.000	0.953	0.960	0.953
Haptics	0.347	0.305	0.263	0.461
Herrings	0.688	0.641	0.672	0.672
InlineSkate	0.265	0.260	0.215	0.373
ItalyPowerDemand	0.949	0.952	0.948	0.951
Lightning2	0.656	0.623	0.623	0.623
Lightning7	0.712	0.658	0.726	0.740
MALLAT	0.786	0.863	0.342	0.828
MedicalImages	0.612	0.611	0.611	0.624
MiddlePhalanxOutlineAgeGroup	0.591	0.610	0.617	0.623
MiddlePhalanxTW	0.552	0.532	0.532	0.532
MoteStrain	0.875	0.877	0.839	0.866
OliveOil	0.900	0.900	0.900	0.900
OSULeaf	0.397	0.587	0.397	0.595
Plane	1.000	1.000	1.000	1.000
ProximalPhalanxOutlineAgeGroup	0.844	0.849	0.854	0.863
ProximalPhalanxTW	0.776	0.761	0.761	0.785
SimulatedSet	0.933	0.931	0.816	0.880
SonyAIBORobotSurface	0.932	0.940	0.953	0.943
SonyAIBORobotSurfaceII	0.887	0.880	0.884	0.887
SwedishLeaf	0.880	0.883	0.899	0.910
Symbols	0.744	0.872	0.884	0.884
SyntheticControl	0.927	0.913	0.833	0.903
ToeSegmentation1	0.952	0.952	0.934	0.930
ToeSegmentation2	0.892	0.815	0.892	0.815
Trace	0.980	0.980	0.980	0.980
TwoLeadECG	0.993	0.998	0.998	0.997
WordSynonyms	0.324	0.560	0.292	0.553
Worms	0.623	0.623	0.494	0.727
WormsTwoClass	0.727	0.649	0.727	0.688

Table 8.4: Value of k (maximum number of shapelets to cache) and number of shapelets found for each dataset.

Dataset	Max. Shapelets	Num. shapelets found	Dataset	Max. Shapelets	Num. shapelets found
Adiac	3900	3900	MiddlePhalanxOutlineCorrect	6000	6000
ArrowHead	360	258	MiddlePhalanxTW	3990	2329
Beef	300	300	MoteStrain	200	69
BeetleFly	200	200	NonInvasiveFatalECG_Thorax1	18000	18000
BirdChicken	200	200	NonInvasiveFatalECG_Thorax2	18000	18000
Car	600	600	OliveOil	300	300
CBF	300	55	OSULeaf	2000	376
ChlorineConcentration	4670	4670	PhalangesOutlinesCorrect	18000	16993
CinC_ECG_torso	400	78	Plane	1050	346
Coffee	280	280	ProximalPhalanxOutlineAgeGroup	4000	2358
Computers	2500	2218	ProximalPhalanxOutlineCorrect	6000	5648
Cricket_X	3900	582	ProximalPhalanxTW	4000	1977
Cricket_Y	3900	515	PtNDeviceGroups	17500	1357
Cricket_Z	3900	612	PtNDevices	17500	760
DiatomSizeReduction	160	160	RefrigerationDevices	3750	3750
DistalPhalanxOutlineAgeGroup	4000	2083	ScreenType	3750	3750
DistalPhalanxOutlineCorrect	6000	4890	SimulatedSet	1000	1000
DistalPhalanxTW	4000	1203	SmallKitchenAppliances	3750	2946
Earthquakes	3220	3220	SonyAIBORobotSurface	200	56
ECGFiveDays	230	74	SonyAIBORobotSurfaceII	270	42
FaceAll	5600	560	StarLightCurves	10000	342
FaceFour	240	173	SwedishLeaf	5000	3642
FacesUCR	2000	268	Symbols	250	92
fiftywords	4500	450	SyntheticControl	3000	355
fish	1750	1750	ToeSegmentation1	400	164
FordA	36010	2252	ToeSegmentation2	360	69
FordB	36360	3099	Trace	1000	266
GunPoint	500	183	TwoLeadECG	230	164
Haptics	1550	1550	TwoPatterns	10000	3664
Herrings	640	612	UWaveGestureLibrary_X	8960	912
InlineSkate	1000	995	UWaveGestureLibrary_Y	8960	902
ItalyPowerDemand	670	141	UWaveGestureLibrary_Z	8960	900
LargeKitchenAppliances	3750	3750	wafer	10000	3054
Lightning2	600	430	WordSynonyms	2670	267
Lightning7	700	491	Worms	1810	835
MALLAT	550	548	WormsTwoClass	1810	1612
MedicalImages	3810	2683	yoga	3000	3000
MiddlePhalanxOutlineAgeGroup	4000	2124			

Table 8.5: Accuracies of our classifier ensemble on shapelet-transformed data and Logical Shapelets [130] on raw data, over 31 datasets.

Dataset	**Ensemble classifier on shapelet-transformed data**	**Logical Shapelets on raw data**
Adiac	0.565 (2)	**0.586 (1)**
Beef	**0.833 (1)**	0.567 (2)
CBF	**0.997 (1)**	0.886 (2)
ChlorineConcentration	**0.700 (1)**	0.618 (2)
CinC_ECG_torso	**0.846 (1)**	0.699 (2)
Coffee	**1.000 (1)**	0.964 (2)
DiatomSizeReduction	**0.876 (1)**	0.801 (2)
ECGFiveDays	**0.999 (1)**	0.994 (2)
FaceAll	**0.737 (1)**	0.659 (2)
FaceFour	**0.943 (1)**	0.489 (2)
FacesUCR	**0.913 (1)**	0.662 (2)
fish	**0.977 (1)**	0.777 (2)
GunPoint	**0.980 (1)**	0.893 (2)
ItalyPowerDemand	**0.952 (1)**	0.936 (2)
Lightning2	**0.656 (1)**	0.426 (2)
Lightning7	**0.740 (1)**	0.548 (2)
MALLAT	**0.940 (1)**	0.656 (2)
MedicalImages	**0.604 (1)**	0.587 (2)
MoteStrain	**0.891 (1)**	0.832 (2)
OliveOil	**0.900 (1)**	0.833 (2)
OSULeaf	**0.715 (1)**	0.686 (2)
SonyAIBORobotSurface	**0.933 (1)**	0.860 (2)
SonyAIBORobotSurfaceII	**0.885 (1)**	0.846 (2)
SwedishLeaf	**0.907 (1)**	0.813 (2)
Symbols	**0.886 (1)**	0.643 (2)
SyntheticControl	**0.983 (1)**	0.470 (2)
Trace	0.980 (2)	**1.000 (1)**
TwoLeadECG	**0.996 (1)**	0.856 (2)
TwoPatterns	**0.941 (1)**	0.539 (2)
wafer	0.998 (2)	**0.999 (1)**
yoga	**0.805** (1)	0.740 (2)
Wins	**28**	3

Table 8.6: Accuracies of our classifier ensemble on shapelet-transformed data and Fast Shapelets [141] on raw data, over 44 datasets.

Dataset	**Ensemble classifier on shapelet-transformed data**	**Fast Shapelets on raw data**
Adiac	**0.565 (1)**	0.486 (2)
Beef	**0.833 (1)**	0.553 (2)
CBF	**0.997 (1)**	0.947 (2)
ChlorineConcentration	**0.700 (1)**	0.583 (2)
CinC_ECG_torso	**0.846 (1)**	0.826 (2)
Coffee	**1.000 (1)**	0.932 (2)
Cricket_X	**0.782 (1)**	0.472 (2)
Cricket_Y	**0.764 (1)**	0.480 (2)
Cricket_Z	**0.772 (1)**	0.438 (2)
DiatomSizeReduction	0.876 (2)	**0.883 (1)**
ECGFiveDays	**0.999 (1)**	0.996 (2)
FaceAll	**0.737 (1)**	0.589 (2)
FaceFour	**0.943 (1)**	0.910 (2)
FacesUCR	**0.913 (1)**	0.672 (2)
fiftywords	**0.719 (1)**	0.511 (2)
fish	**0.977 (1)**	0.803 (2)
GunPoint	**0.980 (1)**	0.939 (2)
Haptics	**0.477 (1)**	0.376 (2)
InlineSkate	**0.385 (1)**	0.266 (2)
ItalyPowerDemand	**0.952 (1)**	0.905 (2)
Lightning2	0.656 (2)	**0.705 (1)**
Lightning7	**0.740 (1)**	0.597 (2)
MALLAT	0.940 (2)	**0.967 (1)**
MedicalImages	**0.604 (1)**	0.567 (2)
MoteStrain	**0.891 (1)**	0.783 (2)
NonInvasiveFatalECG_Thorax1	**0.900 (1)**	0.766 (2)
NonInvasiveFatalECG_Thorax2	**0.903 (1)**	0.802 (2)
OliveOil	**0.900 (1)**	0.787 (2)
OSULeaf	**0.715 (1)**	0.641 (2)
SonyAIBORobotSurface	**0.933 (1)**	0.686 (2)
SonyAIBORobotSurfaceII	**0.885 (1)**	0.785 (2)
StarLightCurves	**0.976 (1)**	0.942 (2)
SwedishLeaf	**0.907 (1)**	0.731 (2)
Symbols	0.886 (2)	**0.932 (1)**
SyntheticControl	**0.983 (1)**	0.919 (2)
Trace	0.980 (2)	**0.998 (1)**
TwoLeadECG	**0.996 (1)**	0.910 (2)
TwoPatterns	**0.941 (1)**	0.887 (2)
UWaveGestureLibrary_X	**0.784 (1)**	0.707 (2)
UWaveGestureLibrary_Y	**0.697 (1)**	0.608 (2)
UWaveGestureLibrary_Z	**0.727 (1)**	0.627 (2)
wafer	**0.998 (1)**	0.996 (2)
WordSynonyms	**0.597 (1)**	0.437 (2)
yoga	**0.805 (1)**	0.751 (2)
Wins	**39**	5

Table 8.7: Numbers of shapelets for all datasets using all methods.

Dataset	Shapelet Transform	After F-stat Filtering	MDLStopCE Clustered	Binary with Correlation Filtering
Adiac	3900	3900	3895	3866
ArrowHead	258	244	234	194
Beef	300	300	300	273
BeetleFly	200	186	186	82
BirdChicken	200	182	182	104
Car	600	600	595	590
CBF	55	51	3	3
ChlorineConcentration	4670	4670	4670	4518
CinC_ECG_torso	78	63	63	50
Coffee	280	280	267	155
Computers	2218	2176	2176	1149
Cricket_X	582	582	12	12
Cricket_Y	515	515	12	12
Cricket_Z	612	612	12	12
DiatomSizeReduction	160	160	160	70
DistalPhalanxOutlineAgeGroup	2083	2077	2077	1779
DistalPhalanxOutlineCorrect	4890	4834	4834	3499
DistalPhalanxTW	1203	1202	1197	912
Earthquakes	3220	3220	3220	2807
ECGFiveDays	74	66	2	1
FaceAll	560	560	14	14
FaceFour	173	164	4	4
FacesUCR	268	268	14	14
fiftywords	450	450	355	352
fish	1750	1750	1750	1743
FordA	2252	2166	2	2
FordB	3099	3023	2	2
GunPoint	183	171	2	2
Haptics	1550	1550	1542	1276
Herrings	612	542	537	533
InlineSkate	995	995	979	937
ItalyPowerDemand	141	137	133	122
LargeKitchenAppliances	3750	3750	3750	1431
Lightning2	430	341	341	318
Lightning7	491	451	439	383
MALLAT	548	533	8	8
MedicalImages	2683	2680	2667	2523
MiddlePhalanxOutlineAgeGroup	2124	2092	2092	1369
MiddlePhalanxOutlineCorrect	6000	5908	5907	4230
MiddlePhalanxTW	2329	2229	2228	1842
MoteStrain	69	58	2	1
NonInvasiveFatalECG_Thorax1	18000	18000	17941	17906
NonInvasiveFatalECG_Thorax2	18000	18000	17960	17954
OliveOil	300	300	300	244
OSULeaf	376	376	6	6
PhalangesOutlinesCorrect	16993	16619	16619	11256
Plane	346	346	342	201
ProximalPhalanxOutlineAgeGroup	2358	2345	2338	2048
ProximalPhalanxOutlineCorrect	5648	5272	5272	3751
ProximalPhalanxTW	1977	1960	1960	1684

Table 8.7: Numbers of shapelets for all datasets using all methods.

Dataset	Shapelet Transform	After F-stat Filtering	MDLStopCE Clustered	Binary with Correlation Filtering
PtNDeviceGroups	1357	1357	1357	735
PtNDevices	760	760	760	448
RefrigerationDevices	3750	3750	3750	1128
ScreenType	3750	3750	3742	870
SimulatedSet	1000	1000	2	2
SmallKitchenAppliances	2946	2942	2942	726
SonyAIBORobotSurface	56	53	2	2
SonyAIBORobotSurfaceII	42	41	2	2
StarLightCurves	342	340	285	272
SwedishLeaf	3642	3642	3626	3623
Symbols	92	92	92	57
SyntheticControl	355	355	6	6
ToeSegmentation1	164	155	2	2
ToeSegmentation2	69	67	2	2
Trace	266	264	142	59
TwoLeadECG	164	132	128	107
TwoPatterns	3664	3617	3617	3569
UWaveGestureLibrary_X	912	912	8	8
UWaveGestureLibrary_Y	902	902	19	19
UWaveGestureLibrary_Z	900	900	261	261
wafer	3054	3019	3019	2311
WordSynonyms	267	267	25	25
Worms	835	822	479	349
WormsTwoClass	1612	1319	2	2
yoga	3000	3000	2996	2980