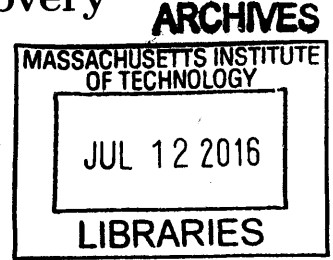


**Feature Flocks: Accurate Pattern Discovery
In Multivariate Signals**

by
Davis W. Blalock



Submitted to the Department of Electrical Engineering and Computer
Science

in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

© Davis W. Blalock, MMXVI. All rights reserved.

The author hereby grants to MIT permission to reproduce and to
distribute publicly paper and electronic copies of this thesis document
in whole or in part in any medium now known or hereafter created.

Author ... **Signature redacted**
Department of Electrical Engineering and Computer Science
Feb 29, 2016

Certified by. **Signature redacted**
John V. Guttag
Professor, Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ... **Signature redacted**
Leslie. A. Kolodziejski
Chair of Department Committee on Graduate Theses

Feature Flocks: Accurate Pattern Discovery In Multivariate Signals

by

Davis W. Blalock

Submitted to the Department of Electrical Engineering and Computer Science
on Feb 29, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Thanks to the rise of wearable and connected devices, sensor-generated time series comprise a large and growing fraction of the world's data. Unfortunately, extracting value from this data can be challenging, since sensors can only report low-level signals (e.g., acceleration), not the high-level phenomena that are typically of interest (e.g., gestures). We introduce a technique to bridge this gap by automatically learning to identify real-world events in low-level data with no human labeling.

By identifying “flocks” of features that repeat in the same temporal arrangement, we learn to recognize such diverse phenomena as human actions, power consumption patterns, and spoken words with up to 96% precision and recall. Our method is fast enough to run in real time and assumes only minimal knowledge of which variables are relevant or how long patterns are. Our evaluation uses numerous publicly available datasets and over 1 million samples of sensor data in which we manually labeled ground truth.

Thesis Supervisor: John V. Guttag

Title: Professor, Electrical Engineering and Computer Science

Acknowledgements

This thesis is possible because of the support and encouragement of many individuals. Foremost, I would like to thank my advisor, John Guttag. More than anyone else, John has changed how I see research, approach problems, and communicate ideas. Further, he had the confidence in me to remain supportive even when it wasn't clear how the problem in this thesis should be formulated, let alone solved. His combination of enthusiasm, affability, and insight make him as good a research coach as one could possibly ask for.

I would also like thank my labmates at MIT. I have learned a great deal about both machine learning and research from Anima Singh, Yun Liu, Guha Balakrishnan, Joel Brooks, Jen Gong, Amy Zhao, Matt Kerr, Maggie Makar, Mitchell Kates, and Nick Kaashoek.

Further, I owe a great deal to my supervisors in college, both at PocketSonic and at UVA. I do not believe I would be here at MIT were it not for John Lach, Ben Boudaoud, William Levy, Kevin Owen, Jeff Pompeo, Jim Tolle, and Jermaine Headley entrusting me with far more autonomy than I could possibly have had merited.

Perhaps most of all, I would like to thank my family and friends for supporting me and challenging me over the years. There is nothing I have accomplished that does not stem from lessons I have learned from these individuals.

Finally, I am grateful to the National Science Foundation for supporting my work under grant number 020772-00001, as well as to the donors who have funded the Harold E. Edgerton Fellowship.

Contents

1	Introduction	15
2	Definitions and Problem	21
2.1	Definitions	21
2.2	Problem Statement	21
2.3	Assumptions	22
2.4	Why the Task is Difficult	23
3	Related Work	25
3.1	Discretization	25
3.2	Closest Pairs	26
3.3	Pattern Refinement	27
3.4	Convolutional Neural Networks	27
3.5	Feature Representation	27
4	Method Overview	29
4.1	Unknown Instance Lengths	30
4.2	Dealing with Irrelevant Features	34
4.3	Generalizing to Multiple Dimensions	35
4.4	Finding Instances	36
5	Method Details	39
5.1	Structure Scores	39
5.2	Constructing the Feature Matrix	40

5.2.1	Categorical Variables	41
5.3	Generating Seed Windows	42
5.4	Scoring Sets of Windows	43
5.5	Recovering Instance Bounds	45
5.6	Runtime Complexity	45
5.6.1	Feature Matrix Construction	46
5.6.2	Instance Discovery	47
5.6.3	Total Runtime	48
6	Results	51
6.1	Datasets	51
6.1.1	TIDIGITS	52
6.1.2	Dishwasher	53
6.1.3	MSRC-12	53
6.1.4	UCR	54
6.2	Evaluation Measures	54
6.3	Comparison Algorithms	56
6.4	Instance Discovery Accuracy	57
6.5	Scalability	59
7	Theoretical Basis	63
7.1	Intuition	63
7.2	Objective Function	64
7.3	Objective Function Derivation	65
7.4	Binary Case	68
7.5	Relationship to Compression	70
7.6	Generalizing to Multiple Models	70
7.7	Generalizing to Time Series	73
7.8	Flock Filters	74
8	Discussion	75
8.1	Summary	75

8.2	Contributions	75
8.3	Limitations	76
8.4	Future Work	77
8.5	Conclusion	78

List of Figures

1-1	True instances of the dishwasher running (shaded). The task is to return the starts and ends of these pattern instances given only the raw data.	16
1-2	Even when told the length and number of pattern instances, the recent algorithm of [1] returns intervals with only the beginning of the pattern.	17
1-3	Our algorithm, Flock, returns accurate intervals with virtually no prior knowledge regarding the number, positions, lengths, or distinguishing features of the pattern.	18
4-1	a) A time series containing two pattern instances. b) Including extra data yields large distances (grey lines) between the windows i and j around the pattern instances. c) Comparing based on subsequences within these windows allows close matches between the pieces of the sine waves.	31
4-2	Feature matrix. Each row is the presence of a particular shape, and each column is a particular time (shown at reduced granularity). Similar regions of data contain the same shapes in roughly the same temporal arrangement.	32
4-3	Because the values in the feature matrix are independent of the window length, a window longer than the pattern can be used to search for instances.	32
4-4	Blurring the feature matrix. Despite the second sine wave being longer and warped in time, the two windows still appear similar when blurred.	33

4-5	Most irrelevant features are ignored after only two or three examples, since it is unlikely for them to repeatedly be in the same place in the window. A few “false positives” may remain.	34
4-6	New dimensions just add rows to the feature matrix. If these dimensions are not influenced by the pattern, they add irrelevant features that will be ignored with high probability.	35
4-7	Given one “seed” window containing an instance of the pattern, we find candidate windows that are spaced in time and have a large dot product with the seed window.	37
5-1	Greedy optimization. The score for a set of windows is based on the size of their intersection minus the portion that intersects with the best window that is excluded. <i>Top</i>) When the best candidate excluded resembles the candidates currently included as possible instances, the score is small. <i>Bottom</i>) When the best candidate excluded does not resemble them, the score is large.	44
5-2	We can recover the boundaries of instances within windows by finding a contiguous set of columns containing many nonzero feature weights.	45
6-1	Example time series from each of the datasets used. The Dishwasher datasets chronicles power consumption measures. The TIDIGITS dataset consists of repeated human utterances. The MSRC-12 dataset captures repeated instances of human actions or gestures. The UCR dataset is composed of real time series from the UCR archive embedded in random walks.	52
6-2	IOU measures the extent to which two regions overlap. The regions (100, 200) and (160, 240) have an IOU of .286.	55
6-3	The proposed algorithm is more accurate for virtually all “match” thresholds on all datasets. Shading corresponds to 95% confidence intervals.	58

6-4	Flock output on a TIDIGITS time series. <i>Top</i>) Original time series. <i>Bottom</i>) The feature matrix Φ . <i>Right</i>) The learned feature weights. These resemble a “blurred” version of the features that repeat each time the word is spoken.	60
6-5	The proposed algorithm is one to two orders of magnitude faster than comparisons.	61

List of Tables

5.1 Notation 40

6.1 Flock F1 Scores are High in Absolute Terms 59

Chapter 1

Introduction

The rise of wearable technology and the internet of things have made available a vast amount of sensor data, and with it the promise of improvements in everything from human health [2] to user interfaces [3] to agriculture [4]. Unfortunately, the raw sequences of numbers comprising this data are often insufficient to offer value. For example, a smart watch user is not interested in their arm’s acceleration signal, but in having their gestures or actions recognized.

Spotting such high-level phenomena using low-level signals is problematic. Given enough labeled examples of the phenomena taking place, one could, in principle, train a classifier for this purpose. Unfortunately, obtaining labeled examples is an arduous task [5–9]. While data such as images and text can be culled at scale from the internet, most time series data cannot. Furthermore, the uninterpretability of raw sequences of numbers often makes time series difficult or impossible for humans to annotate [6]. Perhaps worst of all, one may not even know what phenomena exist to be labeled—indeed, much of the literature on mining time series focuses on *pattern discovery* [1,2,10–16]. Since human labeling is expensive or impossible, what is needed is an unsupervised, *algorithmic* method of discovering and localizing patterns in raw data [3,5].

In this thesis, we describe such an algorithm. Specifically, given a multivariate time series in which certain time intervals contain a characteristic but unknown pattern, our algorithm returns these intervals and a learned model of the pattern.

As an illustration of our problem, consider Figure 1-1. The various lines depict the (normalized) current, voltage, and other power measures of a home dishwasher. Shown are three instances of the dishwasher running, with idleness in between. An ideal algorithm to solve this problem would return the start and end times of each of these instances.

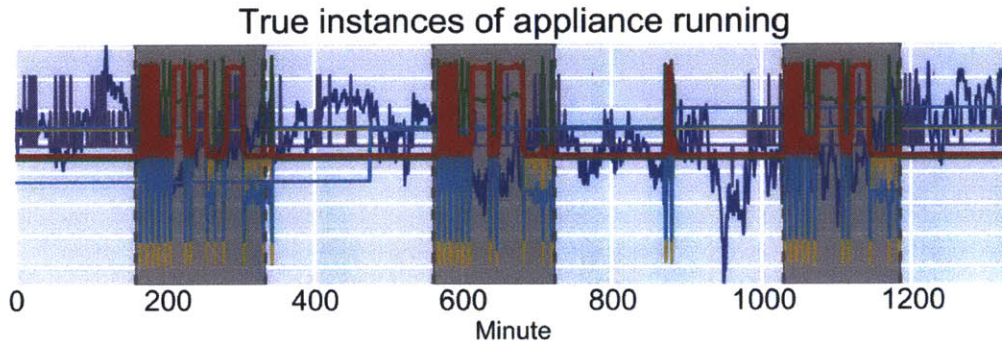


Figure 1-1: True instances of the dishwasher running (shaded). The task is to return the starts and ends of these pattern instances given only the raw data.

This is a challenging task, since the variables affected by the pattern, as well as the number, lengths, and positions of pattern instances, are all unknowns. Further, it is not even clear what objective should be maximized to find this pattern. For example, finding the nearest subsequences under the Euclidean distance yields the incorrect pattern returned by the algorithm of [1] (Fig 1-2). The approach of [17] also returns a nearly identical answer¹.

These errors are not limited to these particular techniques, but natural consequences of the traditional distance-based definition of “pattern” in the time series literature. That is, if pattern instances are defined as intervals of data with low distances to one another for some distance measure, then irrelevant variables and regions of noise can confound the algorithm. This is because the distance measure (absent *a priori* knowledge about which variables to ignore) must combine distances across all variables, and so can return large distances between would-be pattern instances thanks to variations in irrelevant variables. Further, these distances fail to account for

¹See Chapter 6 for descriptions of these algorithms.

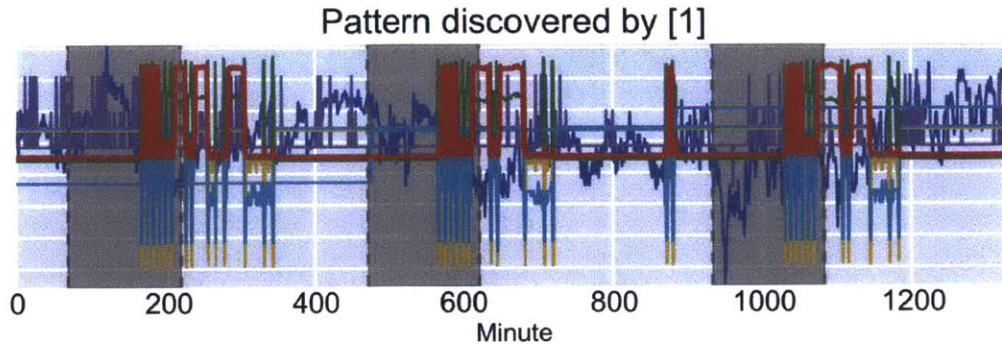


Figure 1-2: Even when told the length and number of pattern instances, the recent algorithm of [1] returns intervals with only the beginning of the pattern.

the probability of a given portion of the data being explained by other processes, such as noise. For example, the intervals shown above include a sizeable initial portion of flat signal—a human observer may know that this data is best explained as inactivity, but to a distance-based algorithm, it is instead an unusually similar section of data that likely indicates a pattern.

To avoid these pitfalls, we formulate pattern discovery not in terms of distance functions, but in terms of probabilities of the data resulting from a pattern or a noise process. At a high level, our approach leverages three observations:

1. Each subsequence of a time series can be seen as having representative features; for example, it may resemble different shapelets [18] or have a particular level of variance.
2. Repeating patterns will cause a disproportionate number of these features to occur together where the pattern happens. We term this set of features affected by the pattern a *flock*. In Figure 1-1, the flock would be a characteristic arrangement of spikes in a certain subset of the signals.
3. If we can identify this flock, we can locate each instance of the pattern with high probability. This holds even in the presence of irrelevant variables (which contribute features excluded from the flock) and unknown instance lengths (which can be inferred based on the interval over which flock features occur together).

As suggested in Figure 1-3 and evaluated rigorously in Chapter 6, this algorithm works well both in absolute terms and relative to existing techniques.

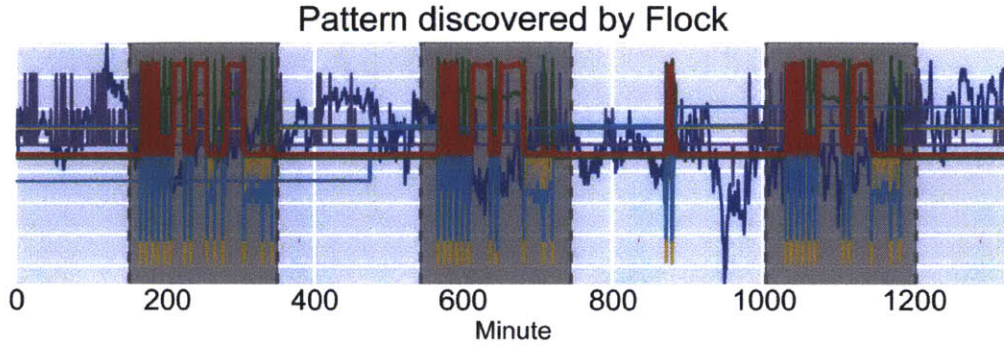


Figure 1-3: Our algorithm, Flock, returns accurate intervals with virtually no prior knowledge regarding the number, positions, lengths, or distinguishing features of the pattern.

Our contributions consist of:

- A formulation of time series motif (i.e., pattern) discovery under more relaxed assumptions than those of existing techniques. In particular, we allow multivariate time series, patterns that affect only subsets of variables, instances of varying lengths, and categorical variables.
- An $O(N \log(N))$ algorithm to discover patterns under this formulation. It requires less than 300 lines of code, but is considerably faster than similar existing algorithms [1, 17] and is fast enough to run on batches of data in real time. It is often much more accurate than existing algorithms as well, since many real-world datasets fail to satisfy these algorithms' assumptions. For example, we recognize instances of the above dishwasher pattern with an F1 score of over 90%, while the best-performing comparison [1] achieves under 30%.
- Open source code and labeled time series that can be used to reproduce and extend our work.

The remainder of this thesis is structured as follows. In Chapter 2, we formally describe our problem and why it is challenging. In Chapter 3, we explain how our problem and approach differ from related work. In Chapters 4 and 5, we give the

intuition for how and why our technique works, and then describe it in detail. In Chapter 6, we evaluate it on a battery of real and synthetic datasets and discuss the implications of our results. In Chapter 7, we describe the theoretical basis for our approach. Finally, in Chapter 8, we discuss the limitations of our work, suggest directions for future research, and conclude.

Chapter 2

Definitions and Problem

In this chapter, we formalize our problem and describe the assumptions it does and does not make about the data. We also discuss why the problem is challenging.

2.1 Definitions

To formalize our task, we introduce the following definitions.

Definition 2.1.1. *Time Series.* A D -dimensional time series T of length N is a sequence of real-valued vectors $t_1, \dots, t_N, t_i \in \mathbb{R}^D$. If $D = 1$, we call T “univariate” or “one-dimensional”, and if $D > 1$, we call it “multivariate” or “multi-dimensional”.

Definition 2.1.2. *Region.* A region R is a pair of indices $(a, b), a \leq b$. The value $b - a + 1$ is termed the **length** of the region, and the time series t_a, \dots, t_b is the **subsequence** for that region. If a region reflects an occurrence of the pattern, we term the region a pattern **instance**.

2.2 Problem Statement

We seek the set of regions that are most likely to have come from a shared “pattern” distribution rather than a “noise” distribution. This likelihood is assessed based on the subset of features maximizing how distinct these distributions are. The feature

representation is arbitrary as long as all features are valued between 0 and 1, but the present work considers features that encode the presence of different shapelets [18] in the data.

Formally, let x_1, \dots, x_K be binary feature representations of all K possible regions in a given time series and x_i^j denote feature j in the region i . We seek the optimal set of regions \mathcal{R}^* , defined as:

$$\mathcal{R}^* = \arg \max_{\mathcal{R}} \max_{\mathcal{F}} p(\mathcal{R}) \sum_{j \in \mathcal{F}} c_j (\log(\theta_{1j}) - \log(\theta_{0j})) \quad (2.1)$$

where θ_{0j} and θ_{1j} are the empirical probabilities for each feature j in the whole time series and the regions \mathcal{R} respectively, and c_j is the count of feature j , $\sum_{i \in \mathcal{R}} x_i^j$. \mathcal{F} is the set of features used, which we term the *flock*. The prior $p(\mathcal{R})$ is 0 if regions overlap or violate certain length bounds (see assumptions below) and is otherwise uniform.

Equation 2.1 says that we would like to find regions i and features j such that x_i^j happens both many times (so that c_j is large) and much more often than would occur by chance (so that $\log(\theta_{1j}) - \log(\theta_{0j})$ is large). In other words, the best flock \mathcal{F} is a large set of features that consistently occur together across many regions, and \mathcal{R}^* is these regions.

Given certain independencies, this objective is a MAP estimate of the regions and features. We defer the details to Chapter 7.

2.3 Assumptions

We do not make any of the following assumptions common in existing work:

- There is a known number of instances, or only a known number of them are needed. E.g., [10–13, 19] explicitly seek only one pair of instances.
- There is a known or constant length for instances. The authors of [12], for example, state that “we are assuming that we are given the lengths of the patterns-of-interest by domain experts.”

- There is a known set of characteristics shared by instances [20–22]. In particular, we do not assume that all instances have the same mean and variance, so we cannot bypass normalization when making similarity comparisons.
- There is only one dimension [10–13, 19].
- All dimensions are affected by the pattern, or the irrelevant dimensions are much lower variance. E.g., [23] converts multidimensional time series to one-dimensional time series via PCA, which only preserves pattern instances if all dimensions are relevant or the irrelevant dimensions have much smaller variance than the relevant ones.

So that the problem is well-defined, we do assume that:

- There is exactly one pattern in the data. While real data often contains multiple distinct patterns, we confine the present work to the one pattern case. Note that, with a technique to find single patterns, one can construct an algorithm to find multiple patterns [1, 17].
- There are at least two instances of the pattern, and no instances overlap in time.
- There exist bounds M_{min} and M_{max} , $M_{min} \geq M_{max}/2$ on the lengths of instances. These bounds disambiguate the case where pairs of adjacent instances could be viewed as single instances of a longer pattern.

We also do not consider datasets in which instances are rare (c.f. [12])—all time series used have instances that collectively comprise at least 10% of the data. This exact percentage is not significant; our algorithm merely requires sufficient “density” of pattern instances for an intelligent initial guessed instance to partially overlap with a true instance. Further, this requirement could be relaxed at the expense of increased runtime. See Chapter 5 for details.

2.4 Why the Task is Difficult

The lack of assumptions means that the number of possible sets of regions and relevant dimensions is intractably large. Suppose that we have a D -dimensional time series T

of length N and $M_{min} \leq M \leq M_{max}$. There are up to $O(N/M)$ instances, which can collectively start at (at most) $\binom{N}{N/M}$ positions. Further, each can be of $O(M)$ different lengths. Finally, the pattern may affect any of $2^D - 1$ possible subsets of dimensions. Altogether, this means that there are roughly $O(N^{N/M} \cdot M^{N/M} \cdot 2^D)$ combinations of regions and dimensions. This is far more combinations than can possibly be examined for even a short time series.

Worse, examining any incorrect combination may give us little or no information with which to move forward. If we look for the pattern in the wrong dimensions or in a set of regions in which it does not occur, we could easily miss it or mistakenly hone in on a false pattern. Further, even if we include many of the right regions and dimensions, dissimilarities in the other regions and dimensions can drown out the pattern and cause us to miss it. In short, it *appears* that solving this problem amounts to an intractable search task.

Chapter 3

Related Work

Finding repeating patterns (typically termed “motifs” [13]) is a common task in time series data mining. Most motif discovery algorithms use one of two approaches: discretization or closest-pair search.

3.1 Discretization

The first approach, discretization, involves quantizing the time series and then applying string mining techniques. Quantization is most often carried out via Symbolic Aggregate Approximation (SAX) [24] (or its variant iSax [25]). These schemes achieve roughly equiprobable discrete symbols by leveraging the observation that elements of z-normalized time series subsequences are approximately normally distributed.

Given the string representation of a time series, several tools can be used to locate patterns. When instances must match exactly, suffix trees [20,21] offer an efficient and exact solution. Another approach is to leverage downward closure [20] to efficiently prune the search space. When patterns can vary slightly, one might instead employ grammar induction [26] or random projections [19] for an approximate but fast answer.

While the string approach is fast, it often has trouble finding the full extents of patterns [26]. This is because longer patterns are less likely to yield (nearly) identical strings.

Moreover, it is not clear how the above string mining techniques can be applied

to multivariate signals. Doing so would require either mining multiple strings simultaneously or somehow reducing the data at a given time across all dimensions to a single symbol. The former approach has not been done under the banner of motif discovery to the best of our knowledge, while the latter would be susceptible to irrelevant dimensions. Specifically, in the latter case, irrelevant dimensions would pollute the representation and likely result in different strings for different pattern instances, even if the relevant dimensions were all in agreement. Further, even without irrelevant signals, the probability of D dimensions independently sharing similar values across several pattern instances decreases exponentially with D , so the strings encoding pattern instances would be even less consistent than in the univariate case.

3.2 Closest Pairs

The closest-pair approach to motif discovery is more common. It typically defines “closeness” in terms of Euclidean distance and formulates the task as a search problem. There are both fast approximate [19] and exact [10, 11] techniques to find this pair. However, both require specifying a particular length at which to search (or iterating over many lengths [1, 17]).

Relatively few of these works have considered the task of finding all pattern instances or finding their locations precisely. The techniques of [1] and [27] do so by finding closest pairs at different lengths and then extracting subsequences that are sufficiently similar to these under an entropy-based measure. Those of [17] and [14] do much the same, although with a distance-based generalization heuristic. These algorithms assume that pattern instances roughly share a single length. Most also assume that all dimensions are relevant. We discuss [1] and [17] in more detail in Chapter 6, where we use them as experimental comparisons.

3.3 Pattern Refinement

Orthogonal to these two discovery approaches are several techniques to refine motif results after-the-fact. These refinements include updating the lengths of the motifs returned [3, 15, 28] or combining different motifs [1, 3, 14, 17] (often those found by searching at different lengths). Refining lengths can be done using Hidden Markov Models [3, 15, 29], suffix trees [29], or a greedy heuristic based on Euclidean distances [28]. Reconciling different motifs is orthogonal to our work, since we consider the case of a single motif. However, several heuristics have been proposed, such as combining motifs that start at the same indices [17] or overlap in time [2, 14].

Refinements to a returned set of instances are compatible with our own technique, so we do not describe them in detail.

3.4 Convolutional Neural Networks

In addition to the traditional time series data mining literature, our approach is also inspired by modern machine learning techniques. In particular, our approach can be seen as a convolutional neural network in which the first layer of filters are sampled from the data and employ a threshold activation function, and the second layer is a single filter with a learned threshold activation function. Indeed, the learned pattern model is no more than a 2D filter (c.f. Section 7.8). This description is an oversimplification (see Chapters 5 and 7), but there is considerable conceptual overlap.

However, while convolutional neural networks require many training examples, many learned filters, many user-specified parameters, and (typically) labeled data, our approach discovers patterns with a single learned filter trained on a handful of unlabeled examples with virtually no user-specified parameters.

3.5 Feature Representation

In terms of approach, our work draws heavily from algorithms that use local features to represent or compare time series. Most notably, several of the most accurate

time series classifiers [30–32] work by transforming the data based on distances to representative sequences known as “shapelets.” Shapelets are also effective for clustering [8, 9], although these works treat entire time series, rather than regions of the series, as the objects to be clustered. The Data Dictionaries of [7] are also similar to our work, but are used for frame-level, rather than event-level, recognition; they also assume known classes and a user-specified query length. These works do not share our problem formulation, but do demonstrate that time series can be represented well by local features.

Chapter 4

Method Overview

In this chapter we offer a high-level overview of our technique and the intuition behind it, deferring details to Chapter 5.

As the reader may recall, the purpose of our algorithm is to find all the “instances” of an unknown pattern within a time series, where an instance is defined as a (start, end) index pair where the pattern has “happened.” In order to accomplish this task, the algorithm carries out the steps given in Algorithm 1. In step 1, we create a representation of the time series that is invariant to instance length and enables rapid pruning of irrelevant dimensions. In step 2, we find sets of regions that might contain pattern instances. In step 3, we refine these sets to estimate the optimal instances \mathcal{R}^* .

Since the main challenge overcome by our technique is the lack of information regarding instance lengths, instance start positions, and relevant dimensions, we elaborate upon these steps by describing how they allow us to deal with each of these unknowns. We begin with a simplified approach and build to a sketch of the full algorithm. In particular, we begin by assuming that time series are one-dimensional, instances are nearly identical, and all features are useful.

Algorithm 1 Flock

1. **Transform the time series T into a feature matrix Φ**
 - (a) Sample subsequences from the time series to use as shapes
 - (b) Transform T into Φ by encoding the presence of these shapes across time
 - (c) Blur Φ to achieve length and time warping invariance
2. **Using Φ , generate sets of “candidate” windows that might contain pattern instances**
 - (a) Find “seed” windows that are unlikely to have arisen by chance
 - (b) Find “candidate” windows that resemble each seed
 - (c) Rank candidates based on similarity to their seeds
3. **Infer the true instances within these candidates**
 - (a) Greedily construct subsets of candidate windows based on ranks
 - (b) Score these subsets and select the best one
 - (c) Estimate exact instance boundaries within the selected windows

4.1 Unknown Instance Lengths

Like most existing work, we find pattern instances by searching over shorter regions of data within the overall time series (Fig 4-1a). Since we do not know how long the instances are, the straightforward approach is to exhaustively search regions of many lengths [1, 17, 27] so that the instances are sure to be included in the search space. In contrast, our approach is to search over all regions of a *single* length M_{max} (the maximum possible instance length) and then refine these approximate regions. For the moment, we defer details of the refinement process. We refer to this set of regions searched as *windows*, since they correspond to all positions of a sliding window over the time series.

This single-length approach presents a challenge: since windows longer than the instances will contain extraneous data, only parts of these windows will appear similar. As an example, consider Figure 4-1. Although the two windows shown contain

identical sine waves, the noise around them causes the windows to appear different, as measured by Euclidean distance (the standard measure in most motif discovery work) (Fig 4-1b). Worse, because the data must be mean-normalized for this comparison to be meaningful [10], it is not even clear which portions of the regions are similar or different—because the noise has altered the mean, even the would-be identical portions are offset from one another.

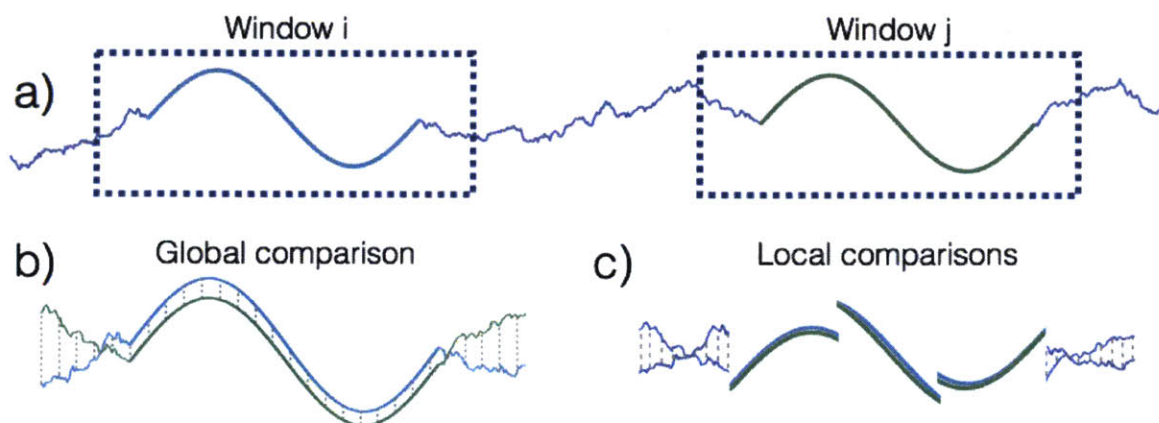


Figure 4-1: a) A time series containing two pattern instances. b) Including extra data yields large distances (grey lines) between the windows i and j around the pattern instances. c) Comparing based on subsequences within these windows allows close matches between the pieces of the sine waves.

However, while the windows appear different when treated as atomic objects, they have many sub-regions (namely, pieces of the sine waves) that are similar when considered in isolation (Fig 4-1c). This suggests that if we were to compare the windows based on *local* characteristics, instead of their *global* shape, we could search at a length longer than the pattern and still determine that windows containing pattern instances were similar.

To enable this, we transform the data into a sparse binary feature matrix that encodes the presence of various shapes at each position in the time series (Fig 4-2). Columns of the feature matrix are shown at a coarse granularity for visual clarity—in reality, there is one column per sample in the time series. We defer explanation of how these shapes are selected and how this feature matrix is constructed to the next section.

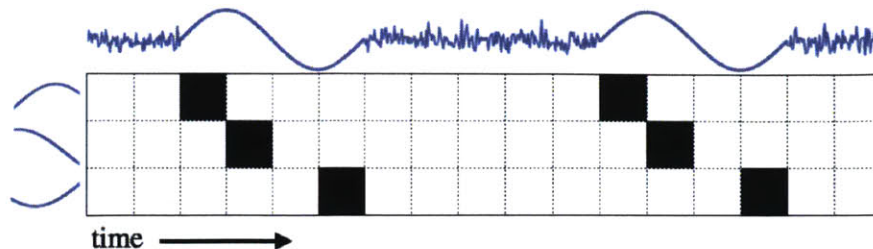


Figure 4-2: Feature matrix. Each row is the presence of a particular shape, and each column is a particular time (shown at reduced granularity). Similar regions of data contain the same shapes in roughly the same temporal arrangement.

Using this feature matrix, we can compare windows of data without knowing the lengths of instances. This is because, even if there is extraneous data at the ends of the windows, there will still be more common features where the pattern happens (Fig 4-3a) than would be expected by chance.

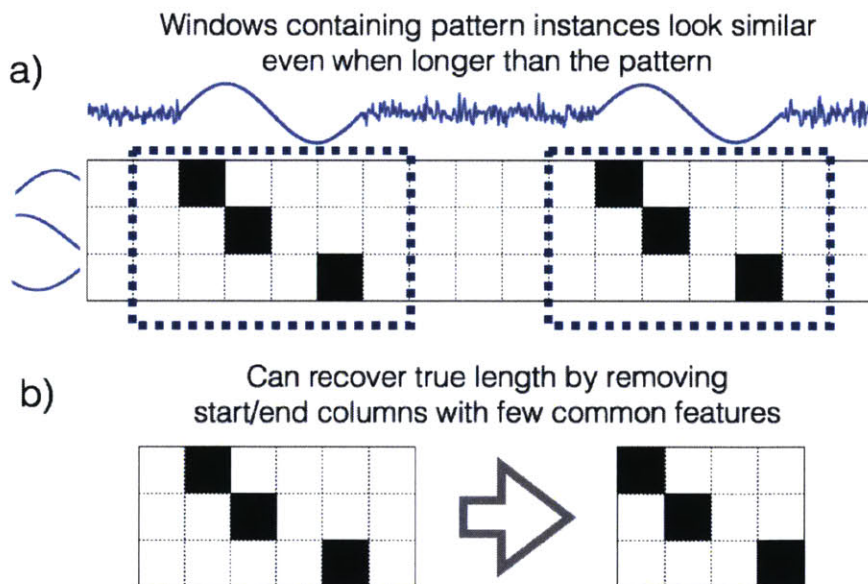


Figure 4-3: Because the values in the feature matrix are independent of the window length, a window longer than the pattern can be used to search for instances.

Once we identify the windows containing instances, we can recover the bounds of the instances by examining which columns in the corresponding windows look sufficiently similar—if a start or end column does not contain a consistent set of 1s across these windows, its features are likely not caused by the pattern, and we prune it (Fig 4-3b).

At present, however, this illustration is optimistic about the regularity of shapes within instances. In reality, a given shape will not necessarily be present in all instances, and a set of shapes may not appear in precisely the same temporal arrangement more than once because of either random variations in overall shape or deformation in time. The latter problem can take the form of uniform scaling [16] or time warping. The former entails the time axis being stretched or compressed by a constant factor in one instance relative to another. The latter entails the time axis being stretched and/or compressed throughout the instance, typically by different amounts at different locations.¹ We defer treatment of the first source of variability, overall shape differences, to a later section. The second, however, deformation of the time axis, can be remedied with a preprocessing step.

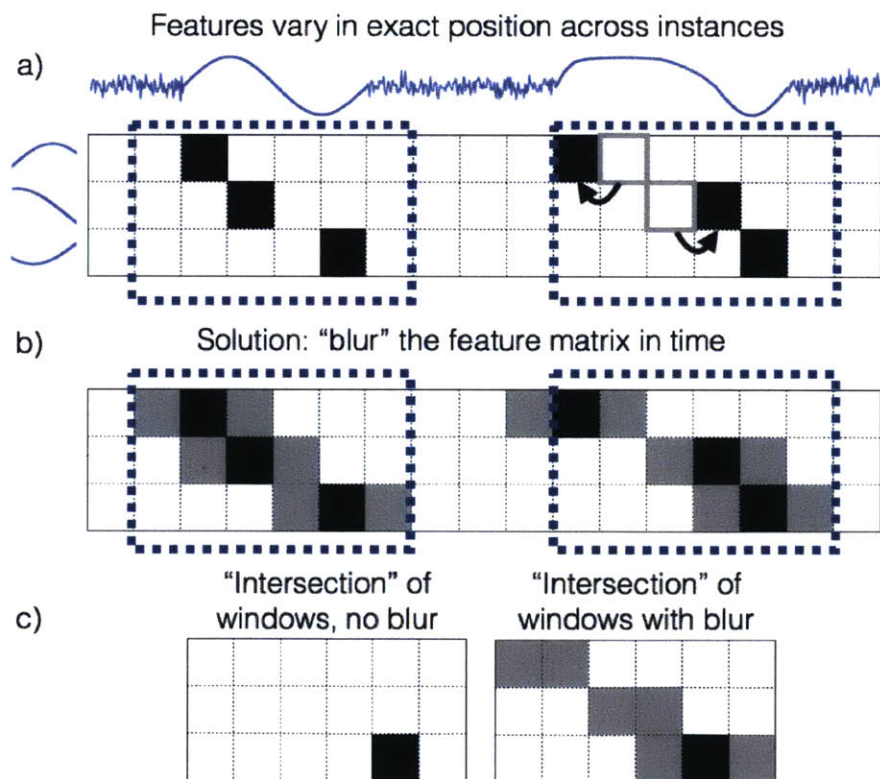


Figure 4-4: Blurring the feature matrix. Despite the second sine wave being longer and warped in time, the two windows still appear similar when blurred.

To handle both uniform scaling and time warping simultaneously, we “blur” the

¹Note that uniform scaling is just a special case of time warping wherein the deformations are uniform throughout the instance.

feature matrix in time. The effect is that a given shape is counted as being present over an interval, rather than at a single time step. This is shown in Figure 4-4, using the intersection of the features in two windows as a simplified illustration of how similar they are. Since the blurred features are no longer binary, we depict the “intersection” as the elementwise minimum of the windows.

4.2 Dealing with Irrelevant Features

This example has so far assumed that the shapes encoded in the matrix are all pieces of the pattern. In reality, we do not know ahead of time which shapes are part of the pattern, and so there will also be many irrelevant features.

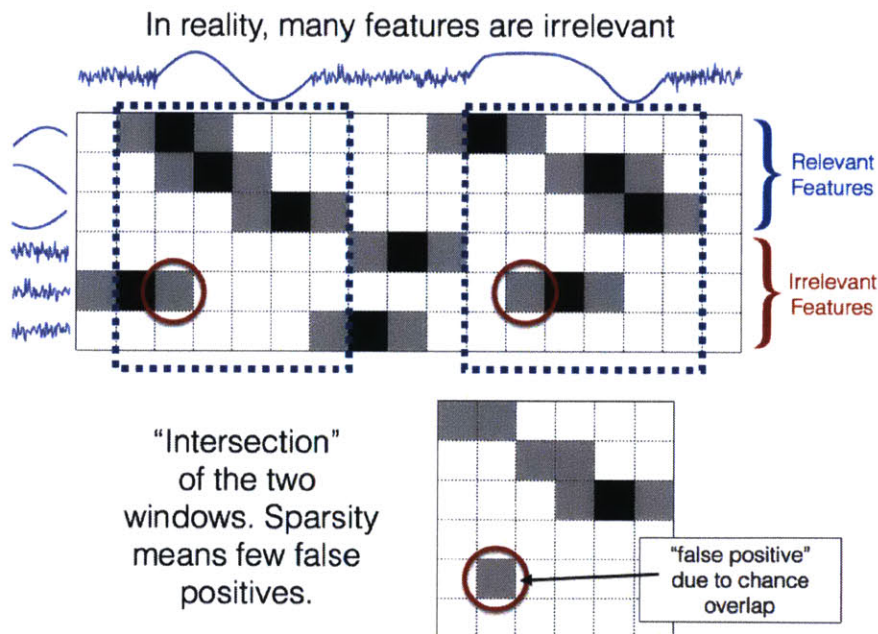


Figure 4-5: Most irrelevant features are ignored after only two or three examples, since it is unlikely for them to repeatedly be in the same place in the window. A few “false positives” may remain.

The combination of sparsity and our “intersection” operation causes us to ignore these extra features (Fig 4-5). To see this, suppose that the probability of an irrelevant feature being present at a particular location in an instance-containing window is p_0 . Then the probability of it being present by chance in k windows is $\approx p_0^k$. Feature

matrices for all datasets we examine are over 90% zeros², since a given subsequence can only resemble a few shapes. Consequently, p_0 is small (e.g., 0.05), and $p_0^k \approx 0$ even for small k .

4.3 Generalizing to Multiple Dimensions

The generalization to multiple dimensions is straightforward: we construct a feature matrix for each dimension and concatenate them row-wise (Fig 4-6). That is, we take the union of the features from each dimension. A dimension may not be relevant, but this just means that it will add irrelevant features. Thanks to the aforementioned combination of sparsity and the intersection operation, we ignore these features with high probability.

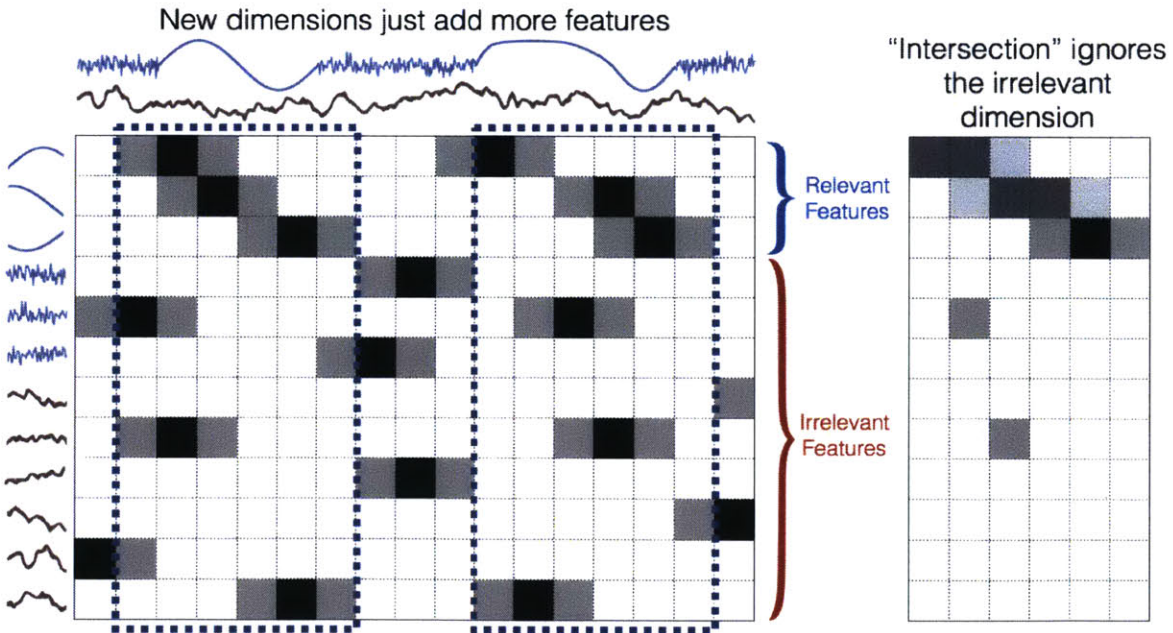


Figure 4-6: New dimensions just add rows to the feature matrix. If these dimensions are not influenced by the pattern, they add irrelevant features that will be ignored with high probability.

²And there is no *a priori* reason to expect other real datasets not to be sparse as well.

4.4 Finding Instances

The previous sections have described how we construct the feature matrix. In this section, we describe how to use this matrix to find pattern instances. A summary is given in Algorithm 2. The idea is that if we are given one “seed” window that contains an instance, we can generate a set of similar “candidate” windows and then determine which of these are likely pattern instances. Since we cannot generate seeds that are certain to contain instances, we generate many seeds and try each. We defer explanation of how seeds are generated to the next section.

Algorithm 2 *FindInstances*(\mathcal{S}, \mathcal{X})

```

1: Input:  $\mathcal{S}$ , “seed” windows;  $\mathcal{X}$ , all blurred windows
2:  $\mathcal{I}_{best} \leftarrow \{\}$ 
3:  $score_{best} \leftarrow -\infty$ 
4: for each seed window  $s \in \mathcal{S}$  do
5:   // Find candidate windows  $\mathcal{C}$  based on
6:   // dot product with seed window  $s$ 
7:    $P \leftarrow [s \cdot \tilde{x}_i, \tilde{x}_i \in \mathcal{X}]$ 
8:    $\mathcal{C} \leftarrow localMaxima(P)$ 
9:    $\mathcal{C} \leftarrow enforceMinimumSpacing(\mathcal{C}, M_{min})$ 
10:   $\mathcal{C} \leftarrow sortByDescendingDotProd(\mathcal{C}, P)$ 
11:  // assess subsets of  $\mathcal{C}$ 
12:  for  $k \leftarrow 2, \dots, |\mathcal{C}|$  do
13:     $\mathcal{I} \leftarrow \{c_1, \dots, c_k\}$  //  $k$  best candidates
14:     $score \leftarrow computeScore(\mathcal{I}, c_{k+1})$ 
15:    if  $score > score_{best}$  then
16:       $score_{best} \leftarrow score$ 
17:       $\mathcal{I}_{best} \leftarrow \mathcal{I}$ 
return  $\mathcal{I}_{best}$ 

```

The main loop iterates through all seeds s and generates sets of candidate windows for each. These candidates are the windows whose dot products with s are local maxima—i.e., they are higher than those of the windows just before and after. To prevent excess overlap, a minimum spacing is enforced between the candidates by only taking the best relative maximum in any interval of width M_{min} (the instance length lower bound). If s contains a pattern instance, the resulting candidates should be (and typically are) a superset of the true instance-containing windows. A toy example of the candidates that might be generated for a given seed window is shown

in Figure 4-7.

In the inner loop, we assess subsets of the candidates to determine which ones contain instances. Since there are $2^{|\mathcal{C}|} = O(2^{N/M_{min}})$ possible subsets, we use a greedy approach that tries only $|\mathcal{C}| = O(N/M_{min})$ subsets. Specifically, we rank the candidates based on their dot products with s and assess subsets that contain the k highest-ranking candidates for each possible k .

The final set returned is the highest-scoring subset of candidates for any seed. See Section 5.4 for an explanation of the scoring function.

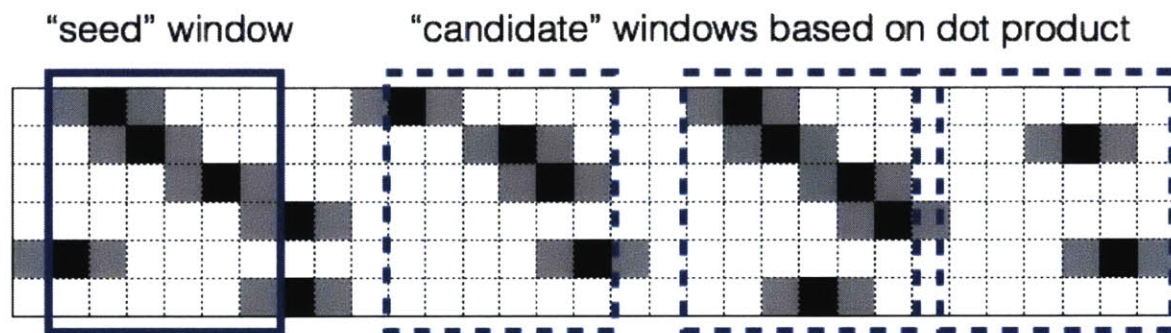


Figure 4-7: Given one “seed” window containing an instance of the pattern, we find candidate windows that are spaced in time and have a large dot product with the seed window.

Chapter 5

Method Details

We now describe how the ideas of the previous chapter translate into a concrete algorithm. Throughout this chapter, we will use the variables described in Table 5.1.

5.1 Structure Scores

We select shape features and seed regions that appear most likely to have been generated by some latent phenomenon. Since we lack domain-specific knowledge about what distinguishes such regions, we use the common approach of modeling “non-pattern” time series as random walks [1, 12]. Specifically, let T be a univariate time series of length N , and W_1, \dots, W_{100} be a collection of 100¹ Gaussian random walks of length N with $\sigma = \text{sqrt}(E[(t_i - t_{i-1})^2])$. We define:

$$\text{structure}(T) = \min_W \frac{1}{N} \sum_{i=1}^N ((t_i - \mu_T) - (w_i - \mu_W))^2 \quad (5.1)$$

where μ_T and μ_W are the means of the time series. The score is the minimum squared Euclidean distance to any of the random walks, normalized by mean and length. For multivariate time series, we sum the scores for each dimension. This is an approximation to the negative log likelihood of T being a random walk, using the optimal σ .

¹The exact number is unimportant; using larger values (e.g., 1000 or 10,000) has no effect.

Table 5.1: Notation

Variable	Meaning
T	A time series $T = \{t_1, \dots, t_N\}$, $t_i \in \mathbb{R}^D$
N	The length of T
D	The dimensionality of T
M_{min}	The lower bound on instance lengths
M_{max}	The upper bound on instance lengths
Φ	The feature matrix $\Phi = \{\phi_1, \dots, \phi_N\}$, $\phi_i \in \mathbb{R}^J$
$\tilde{\Phi}$	The blurred feature matrix $\tilde{\Phi} = \{\tilde{\phi}_1, \dots, \tilde{\phi}_N\}$, $\tilde{\phi}_i \in \mathbb{R}^J$
X	The windows of data in Φ , $X = \{x_1, \dots, x_{N-M_{max}+1}\}$, $x_i = \Phi_{:,i:i+M_{max}}$
\tilde{X}	The windows of data in $\tilde{\Phi}$, $\tilde{X} = \{\tilde{x}_1, \dots, \tilde{x}_{N-M_{max}+1}\}$, $\tilde{x}_i = \tilde{\Phi}_{:,i:i+M_{max}}$
S	The set of “seed” indices, $s_i \in \{1, \dots, N - M_{max} + 1\}$

5.2 Constructing the Feature Matrix

The first step in building the feature matrix is selecting the lengths of the shapes to use as features. Since we do not know what length is best, we employ all lengths that are powers of two within the interval $[8, M_{max}]$ samples. 8 is used because 2 or 4 samples are not enough to define a meaningful shape.

For each length M and each dimension, we select shapes by randomly sampling from the data. To limit the algorithm’s complexity to $O(N \log(N))$, we select $\log(N)$ subsequences. The probability of each subsequence being selected is proportional to its structure score.

For each shape j , we construct its row in the feature matrix Φ_j by sliding it over the data in its dimension and setting the value to 1 iff the distance between the shape and the subsequence centered at each position is less than some threshold. This threshold is fixed at 0.25 since this robustly rejects random walk data and consistently worked better than .125 or .5 in preliminary experiments.² If a shape has no subsequences within the threshold (other than itself), its row is discarded to

²On small sets of time series not used in the reported experiments

save computation. The row is also discarded if it is more than half nonzeros, which happens in constant signals. Distances are given by the mean-normalized Euclidean distance squared between the shape and subsequence, divided by their length and the variance of the shape. This is equivalent to the z-normalized Euclidean distance if the subsequence and shape have the same variance, but is larger if their variances differ.³

To construct the blurred feature matrix $\tilde{\Phi}$, we convolve each row with a Hamming filter of length M_{min} . We then divide each entry by the largest value within $M_{min}/2$ time steps in its row, so that the maximum value in $\tilde{\Phi}$ remains 1.

5.2.1 Categorical Variables

The above method of constructing the feature matrix is designed for real-valued variables. A similar technique can be employed to represent categorical variables. Specifically, one can augment Φ with one row for each value of each categorical variable and store a 1 wherever a given variable takes on a given value. In other words, one can encode the presence of a particular value for a categorical variable at each time step, instead of the presence of a particular shape. This is equivalent to concatenating a one-hot-encoded representation of the categorical variable to the feature matrix.

Alternatively, if the categorical variable has few possible values (e.g., genetic data with values $\{A, T, C, G\}$), one could encode the presence of *sequences* of particular values. For example, one might encode the presence of an *AGTT* subsequence beginning at time t instead of merely the presence of the value A at time t . We believe that this encoding might perform better since it would yield greater sparsity (since there are many more possible subsequences than individual values), and thus allow meaningful blurring. However, we defer exploration of this question to future work, since all time series considered in this thesis are real-valued.

³Technically, it is smaller if the Euclidean distance between the z-normalized objects is > 1 and the subsequence’s variance is less than that of the shape, but in this case the threshold will not be met anyway. Even in the limit of a zero-variance subsequence, the distance will still be at least 1.

5.3 Generating Seed Windows

We generate seeds by finding the start indices associated with the highest structure scores. Concretely, we score each start index t as the sum of the structure scores of the subsequences of all power-of-two lengths $M \in [8, M_{min}]$ that begin at t . We then take the two best start indices at least M_{min} apart. One could use any constant number of seeds without affecting the runtime, and we choose two because using more made little or no difference in preliminary experiments. Since these two seeds are unlikely to be perfectly centered on instances, we add 10 additional seeds⁴ on either side of each, spaced $M_{max}/10$ apart. This means that as long as one of the initial two seeds' windows includes at least the edge of an instance, one of the expanded set of seeds will be within $M_{max}/10$ of that instance's true start position. This procedure yields $2 + 2 \cdot 2 \cdot 10 = 42$ seeds total.

This seed generation scheme is a heuristic, but we found that it worked better in practice than other heuristics. In particular, using the two indices with the best structure scores yielded higher accuracy than using the indices of the closest pair of subsequences under the z-normalized Euclidean distance, as in [1, 17, 27].

If the data contained many subsequences that appeared to be non-random but were not instances, a different heuristic would be required. In practice, one would likely use regions with high variance, unusual frequency content, etc, based on basic knowledge of the domain. Using a motion sensor, for example, one could be confident that flat regions do not correspond to motion, and so high-variance regions should be used as seeds. One could also provide a single labeled instance to bypass the need for seed generation entirely. Finally, if one were unable to reliably generate seeds that contained part of a pattern instance through any scoring function, one could use uniformly spaced indices throughout the entire time series at the expense of increased runtime.

⁴Again, the precise number is irrelevant; the requirement is merely a dense and uniform sampling.

5.4 Scoring Sets of Windows

Recall that we evaluate sets of candidate windows using a scoring function. The function used is given in Algorithm 3. This returns the value of the objective function (Eq 2.1), with three alterations:

- We set the probabilities in the “pattern” model θ_1 using the blurred windows.
- We disallow features for which $\theta_1 \leq .5$ to prevent the learning of two or more unrelated but frequent sets of features. This resembles a soft “intersection” operation and can be seen as a prior $p(\mathcal{F}|\theta_1)$. The number .5 is the minimum value that prevents learning two disjoint patterns which occur with relative frequencies⁵ p and $1 - p$; for any p , one of these two quantities must be $\leq .5$, so only the features of one pattern can be learned.
- We subtract the log odds of the windows being generated by noise or a “rival” pattern exemplified by the best candidate excluded. See Chapter 7 for a probabilistic interpretation of this operation.

A simplified illustration of how sets of windows are scored is given in Figure 5-1. The score can be thought of (imprecisely) as the size of the intersection of the windows deemed to be instances minus the best non-instance candidate.

Algorithm 3 $\text{computeScore}(\mathcal{I}, \text{nextIdx})$

```

1:  $c \leftarrow \sum_{i \in \mathcal{I}} x_i$  // feature counts
2:  $\tilde{c} \leftarrow \sum_{i \in \mathcal{I}} \tilde{x}_i$  // blurred feature counts
3:  $\theta_1 \leftarrow \tilde{c} / |\mathcal{I}|$  // feature probabilities
4:  $\Delta \leftarrow \log \theta_1 - \log \theta_0$  // feature weights
5:  $\mathcal{F} \leftarrow \{j \mid \Delta_j > 0 \wedge \theta_{1j} > .5\}$  // optimal features
6:  $w \leftarrow \langle \Delta_j \text{ if } j \in \mathcal{F} \text{ else } 0 \rangle$  // feature weight vector
7:  $\text{odds}_{\text{pattern}} \leftarrow w \cdot c$ 
8:  $\text{odds}_{\text{next}} \leftarrow w \cdot x_{\text{nextIdx}} \cdot |\mathcal{I}|$ 
9:  $\text{odds}_{\text{noise}} \leftarrow \sum_{j \in \mathcal{F}} w_j \cdot E[\tilde{\Phi}] \cdot |\mathcal{I}|$ 
10: return  $\text{odds}_{\text{pattern}} - \max(\text{odds}_{\text{noise}}, \text{odds}_{\text{next}})$ 

```

In lines 1-3, we compute the counts of each feature and construct θ_1 based on the data in \mathcal{I} . In lines 4-5, we determine the optimal set of features to include assuming

⁵Where the total number of events is equal to the total number of instances of either pattern, not the total number of windows in the time series or candidates.

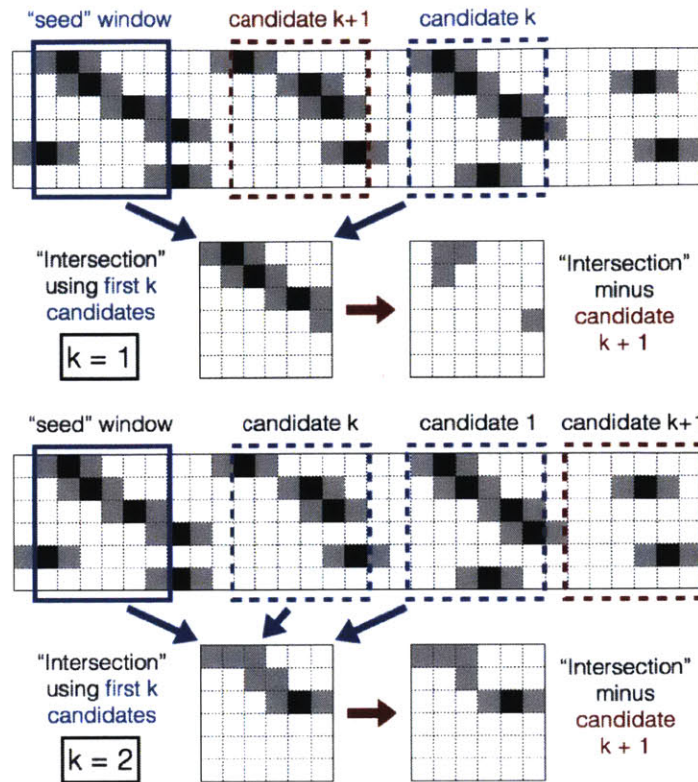


Figure 5-1: Greedy optimization. The score for a set of windows is based on the size of their intersection minus the portion that intersects with the best window that is excluded. *Top*) When the best candidate excluded resembles the candidates currently included as possible instances, the score is small. *Bottom*) When the best candidate excluded does not resemble them, the score is large.

irrelevant features are distributed according to θ_0 . In line 6, we construct a set of weights w for the features. These weights are 0 for features that are not in \mathcal{F} and equal to the difference between $\log(\theta_1)$ and $\log(\theta_0)$ for those that are. The introduction of w is merely a convenience so that $w \cdot c = \sum_{j \in \mathcal{F}} c_j (\log(\theta_{1j}) - \log(\theta_{0j}))$, the original objective function. Line 7 computes the value of this objective, which can be seen as the increase in log likelihood from generating the ones in \mathcal{I} using θ_1 instead of θ_0 . Line 8 computes this increase in odds for the next window excluded, instead of for the supposed instances. Line 9 computes the increase in odds for an average “noise” window.

The returned score corresponds to the log odds of a set of instances being generated by a pattern model versus either random noise or another pattern exemplified by the best candidate excluded. See Chapter 7 for a more detailed analysis.

5.5 Recovering Instance Bounds

Given an estimated set of instance-containing window positions \mathcal{I} , we compute \mathcal{R} by discarding columns in the windows that are no more similar than chance.

Let V be the feature weights w in the above algorithm associated with \mathcal{I} and reshaped to match the $J \times M_{max}$ shape of the window, where J is the number of rows in Φ . These weights are shown in Figure 5-2a.

We sum the entries in each column of V to produce a set of column scores (Fig 5-2b), and subtract from each score the number of ones that would be expected by chance (Fig 5-2c). This number is equal to $J \cdot E[\tilde{\Phi}]^{|\mathcal{I}|-1}$. We then extract the maximum subarray of the scores to find the start and end offsets of the “pattern” within v (Fig 5-2d). We add these offsets to the indices in \mathcal{I} to get \mathcal{R} .

This scheme is simple to implement, but not optimal in any formal sense.

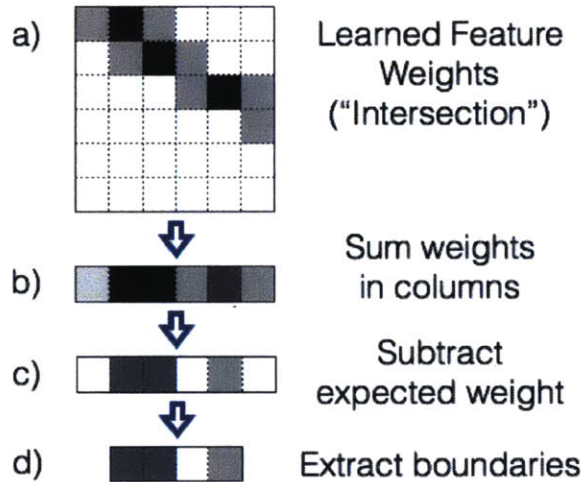


Figure 5-2: We can recover the boundaries of instances within windows by finding a contiguous set of columns containing many nonzero feature weights.

5.6 Runtime Complexity

In this section, we describe the time complexity of our algorithm. Since our algorithm consists of two sequential steps, constructing the feature matrix and optimizing the objective, we analyze each step separately. Wherever possible, we give the exact

runtime (i.e., $\Theta(\cdot)$), rather than the worst case (i.e., $O(\cdot)$).

5.6.1 Feature Matrix Construction

Lemma 1. *Computing the structure scores for all subsequences in a D dimensional signal requires $\Theta(DM_{max}N)$ time.*

Proof. The time to compute the score for a 1D subsequence of a given length M is $\Theta(M)$, and there are $\Theta(N)$ subsequences per dimension. Since the lengths used differ by factors of 2, the sum of all lengths is $\Theta(M_{max})$. The total time to compute all scores for one dimension is therefore $\Theta(M_{max}N)$. Since the total score is the sum of scores for each dimension, the total time is $\Theta(DM_{max}N)$. \square

Lemma 2. *Given the structure scores, selecting all of the shapes to use as features for a given dimension requires $\Theta(\log(M_{max}) \log(N)^2)$ time.*

Proof. There are $\Theta(\log(M_{max}))$ shape lengths used. For each length, we select $\Theta(\log(N))$ shapes. Since the shape selection is weighted (using the structure scores), it requires $\Theta(\log(N))$ time to select each successive index (by generating a number $\in [0, 1)$ and binary searching through the CDF of the probabilities). This is a total of $\Theta(\log(M_{max}) \log(N)^2)$ operations to select all the shapes for a given dimension. \square

Lemma 3. *Computing the row of the feature matrix for a shape of length M requires $\Theta(MN)$ time.*

Proof. Computing the distance between a shape of length M and an arbitrary subsequence (also of length M) requires $\Theta(M)$ operations. However, because consecutive subsequences share all but the first and last element, it is also possible to compute these distances by updating a set of sufficient statistics—namely, the mean, variance and dot product—at each time step [18]. The mean and variance can be obtained in $O(1)$ at each time step, but the dot product between the shape and subsequence requires $\Theta(M)$.⁶ It is therefore $\Theta(MN)$ to compute the distances to all subsequences.

⁶The dot products could instead be computed using a Fast Fourier Transform (FFT) in $O(N \log(N))$ time. However, because we do not use this approach (since it is not necessarily faster), we ignore it for purposes of complexity analysis.

Since each distance can be thresholded in $O(1)$, the total time to construct each row is also $\Theta(MN)$. □

Theorem 1. *Constructing the feature matrices Φ and $\tilde{\Phi}$ requires $\Theta(DM_{max}N \log(M_{max}) \log(N))$ time.*

Proof. For a given dimension, computing the structure scores requires $\Theta(M_{max}N)$ time (Lemma 1). Selecting the shapes for this dimension using these scores requires $\Theta(\log(M_{max}) \log(N)^2)$ (Lemma 2). This produces $\Theta(\log(M_{max}) \log(N))$ shapes. Computing the row for each shape requires $\Theta(NM)$ for shape length M (Lemma 3), but because the lengths differ by powers of two, the sum for all rows is only $\Theta(NM_{max} \log(N))$. The shapes and rows for each dimension are constructed independently, so the total time to construct the matrix Φ is $\Theta(DNM_{max} \log(N))$.

Blurring Φ into $\tilde{\Phi}$ requires convolving each row with a Hamming filter of length $\Theta(M_{max})$, which is $\Theta(M_{max}N)$ for each row⁷ and therefore $\Theta(DM_{max}N \log(M_{max}) \log(N))$ total. The total runtime is therefore equal to the time to blur the matrix, $\Theta(DM_{max}N \log(M_{max}) \log(N))$. □

5.6.2 Instance Discovery

Lemma 4. *Generating seed windows requires $\Theta(DM_{max}N)$ time.*

Proof. As described in Lemma 1, computing the structure scores for all subsequences in a D -dimensional signal requires $\Theta(DM_{max}N)$ time. Finding the two largest scores at least M_{min} apart requires $\Theta(N)$ time, and adding a constant number of seeds near this pair requires $O(1)$ time. □

Lemma 5. *Generating candidates for a given seed requires $\Theta(DM_{max}N \log(M_{max}) \log(N))$.*

Proof. Since there are $\Theta(D \log(M_{max}) \log(N))$ rows in the matrix and windows are of width M_{max} , the total number of elements in a window is $\Theta(DM_{max} \log(M_{max}) \log(N))$.

⁷Again assuming time-domain, rather than FFT-based, convolution

Computing the dot products between a seed window and all other windows therefore requires $\Theta(DM_{max}N \log(M_{max}) \log(N))$ time. Given the dot products, it requires only $\Theta(N)$ time to extract the relative maxima to use as candidates, so the total complexity is equal to that of computing the dot products. \square

Lemma 6. *Finding the best subset of a set of candidates requires $\Theta(DN \log(M_{max}) \log(N))$ time.*

Proof. In the worst case, the candidates are exactly M_{min} apart, and so the total number of elements in all of them is proportional to the total size of the feature matrix, $\Theta(DN \log(M_{max}) \log(N))$. Since the sufficient statistics (θ_1 , etc.) rely only on sums of elements in the candidates, and each subset considered is a superset of the previous one considered, each candidate's data need only be added in once. Ranking and sorting the candidates is only $\Theta(N/M_{min} \log(N/M_{min}))$, so the total time is proportional to the number of elements in all candidates which is $O(DN \log(M_{max}) \log(N))$. \square

Theorem 2. *Optimizing the objective given the feature matrix and seeds requires $\Theta(DM_{max}N \log(M_{max}) \log(N))$ time.*

Proof. Optimizing the objective consists of generating seed windows, generating candidate windows, and selecting the best subset of candidate windows. By Lemma 4, the first step requires $\Theta(DM_{max}N)$ time. Since there are a constant number of seeds, generating all candidates has the same complexity as generating the candidates for a single seed, $\Theta(DM_{max}N \log(M_{max}) \log(N))$ time (Lemma 5). Similarly, selecting from all of sets of candidates is the same complexity as selecting from one set of them, which is $\Theta(DN \log(M_{max}) \log(N))$ (Lemma 6). Adding up these complexities, the total time is $\Theta(DM_{max}N \log(M_{max}) \log(N))$. \square

5.6.3 Total Runtime

Theorem 3. *The entire algorithm requires $\Theta(DM_{max}N \log(M_{max}) \log(N))$.*

Proof. The entire algorithm consists of constructing the feature matrix and optimizing the objective. These steps execute sequentially, so their complexities add. By

Theorem 1, the first step requires $\Theta(DM_{max}N \log(M_{max}) \log(N))$ time. By Theorem 2, the second step requires $\Theta(DM_{max}N \log(M_{max})N \log(N))$ time. Summing these two, we arrive at $\Theta(DM_{max}N \log(M_{max}) \log(N))$ time. \square

Chapter 6

Results

We implemented our algorithm, along with existing algorithms from the literature [1, 17], using Scipy [33]. For the existing algorithms, we JIT-compiled the inner loops using Numba [34]. All code and raw results are publicly available [35]. We have endeavored to make our data cleaning code particularly simple and modular in the hopes that it will be useful to future researchers. Our full algorithm, including feature matrix construction, is under 300 lines of code.

6.1 Datasets

We used the datasets illustrated in Figure 6-1. These were selected on the basis that they were both publicly available and contained repeated instances of some ground truth pattern, such as a repeated gesture or spoken word.

Some of these datasets could be mined with simpler techniques than those considered here—e.g., an appropriately-tuned peak detector could find many of the instances in the TIDIGITS time series. However, the goal of our work is to find patterns *without* the knowledge that they resemble peaks, tend to be close together, etc. Thus, we allow these simplifying characteristics to be present, but refrain from exploiting them.

To aid reproducibility, we supplement the source code with full descriptions of our

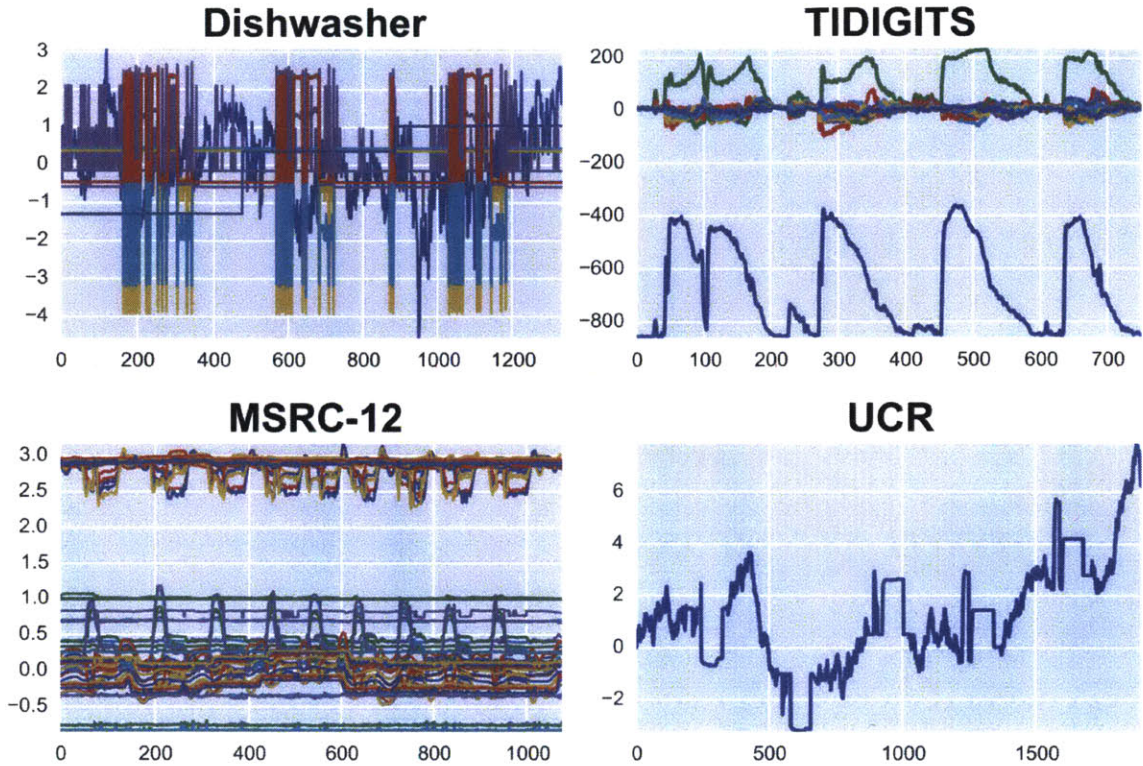


Figure 6-1: Example time series from each of the datasets used. The Dishwasher datasets chronicles power consumption measures. The TIDIGITS dataset consists of repeated human utterances. The MSRC-12 dataset captures repeated instances of human actions or gestures. The UCR dataset is composed of real time series from the UCR archive embedded in random walks.

preprocessing, randomization, etc, at [35], and omit the details here for brevity.

6.1.1 TIDIGITS

The TIDIGITS dataset [36] is a large collection of human utterances of decimal digits. We use a subset of the data consisting of all recordings containing only one type of digit (e.g., only “5”s or only “9”s). We randomly concatenated sets of these recordings to form 1604 time series in which multiple speakers utter the same word a total of 5-8 times. As is standard practice [15], we represented the resulting time series using Mel-Frequency Cepstral Coefficients (MFCCs) via Librosa [37], rather than as the raw speech signal. We use the first 13 MFCCs (as is common) extracted via a sliding window of length 10ms and a stride of 5ms. We then downsampled this

densely-sampled data by a factor of 2.

Because each recording brackets the word utterance with silence, we used as ground truth instance boundaries the first and last time steps in which the signal power exceeded 10% of its maximum value. This tends to be within 10-20% of the correct boundaries of each word.

6.1.2 Dishwasher

The Dishwasher dataset [38] consists of energy consumption and related electricity metrics at a power meter connected to a residential dishwasher. It contains twelve variables and two years worth of data sampled once per minute, for a total of 12.6 million data points.

We manually plotted, annotated, and verified pattern instances in all 1 million+ of its subsequences.¹

Because this data is 100x longer than what the comparison algorithms can process in a day [1], we followed much the same procedure as for the TIDIGITS dataset. Namely, we extracted sets of five to eight pattern instances and concatenated them to form shorter time series. We also extracted a random amount of data around each instance, with length in [25, 200] on either side (the patterns are of length ~ 150).

6.1.3 MSRC-12

The MSRC-12 dataset [39] consists of (x,y,z) human joint positions captured by a Microsoft Kinect while subjects repeatedly performed specific motions. Each of the 594 time series in the dataset is 80 dimensional and contains 8-12 pattern instances. To help the comparison algorithms (which are quadratic) run in a reasonable time frame, we downsampled each time series by a factor of 2.

Each instance is labeled with a single marked time step, rather than with its boundaries. Since the boundaries were not recorded, we use the number of marks in each time series as ground truth. That is, if there are k marks, we treat the first k

¹see the README in the datasets/ directory on our supporting website [35] for a more thorough explanation of how this was done and what the annotations mean.

regions returned as correct. This is a less stringent criterion than on other datasets, but favors comparison algorithms insofar as they often fail to identify exact region boundaries. We attempted to infer ground truth boundaries based on the positions of the marks, but found that this was not possible to do accurately. Similarly, we tried requiring each reported instance to encompass a mark in order to be correct, but found that the errors reported were dominated by the case where the reported instances happened to fall exactly between the marks. Thus, the scheme of assuming that each instance is in the correct place proved to be the most fruitful. Further, when manually examining incorrect outputs post-hoc, we found that this scheme likely introduced little error, since the dominant failure mode for all algorithms was failure to report some of the true instances, not detection of entirely incorrect patterns (and the employed scheme accurately detects the former).

6.1.4 UCR

Following [1], we constructed synthetic datasets by planting examples from the UCR Time Series Archive [40] in random walks. We took examples from the 20 smallest datasets,² as measured by the lengths of their objects. For each dataset, we created 50 time series, each containing five objects of one class. This yielded 1000 time series and 5000 instances. The random walk segments between the starts and ends of instances are $1.25\times$ the lengths of the instances.

Note that this is two orders of magnitude larger than the subset of datasets and instances used in [1], which may explain why our results using their algorithm are not as optimistic as their own.

6.2 Evaluation Measures

Let \mathcal{R} be the ground truth set of instance regions and let $\hat{\mathcal{R}}$ be the set of regions returned by the algorithm being evaluated. Further let $r_1 = (a_1, b_1)$ and $r_2 = (a_2, b_2)$

²Before the 2015 update to the archive in which additional datasets were added. See the source code for the exact datasets used.

be two regions.

Definition 6.2.1. $IOU(r_1, r_2)$. The **Intersection-Over-Union (IOU)** of r_1 and r_2 is given by $|r_1 \cap r_2| / |r_1 \cup r_2|$, where r_1 and r_2 are treated as intervals. See 6-2 for an illustration. This measures the extent to which two regions overlap, with an IOU of 1 implying perfect overlap and an IOU of 0 implying no overlap.³

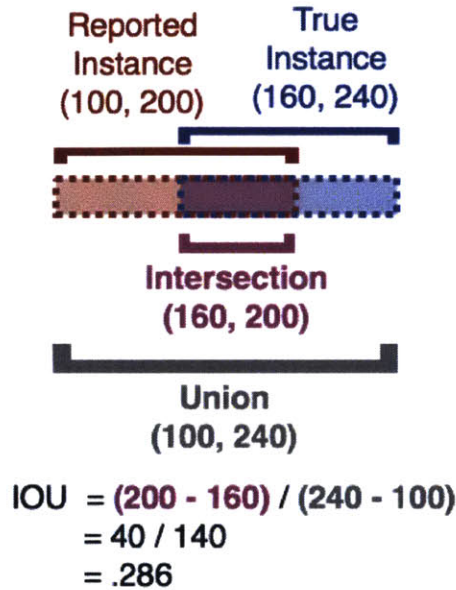


Figure 6-2: IOU measures the extent to which two regions overlap. The regions (100, 200) and (160, 240) have an IOU of .286.

Definition 6.2.2. $Match(r_1, r_2, \tau)$. r_1 and r_2 are said to **Match** at a threshold of τ iff $IOU(r_1, r_2) \geq \tau$.

Definition 6.2.3. $MatchCount(\hat{\mathcal{R}}, \mathcal{R}, \tau)$. The **MatchCount** of $\hat{\mathcal{R}}$ given \mathcal{R} and τ is the greatest number of matches at threshold τ that can be produced by pairing regions in $\hat{\mathcal{R}}$ with regions in \mathcal{R} such that no region in either set is present in more than one pair.⁴

³It is also equal to the Jaccard similarity coefficient between the two regions when the regions are treated as sets of indices.

⁴Since regions (and their possible matches) are ordered in time, this can be computed efficiently after sorting $\hat{\mathcal{R}}$ and \mathcal{R} .

Definition 6.2.4. *Precision, Recall, and F1 Score.*

$$Precision(\hat{\mathcal{R}}, \mathcal{R}, \tau) = MatchCount(\hat{\mathcal{R}}, \mathcal{R}, \tau) / |\hat{\mathcal{R}}| \quad (6.1)$$

$$Recall(\hat{\mathcal{R}}, \mathcal{R}, \tau) = MatchCount(\hat{\mathcal{R}}, \mathcal{R}, \tau) / |\mathcal{R}| \quad (6.2)$$

$$F1(\hat{\mathcal{R}}, \mathcal{R}, \tau) = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (6.3)$$

6.3 Comparison Algorithms

Among the 70+ techniques we reviewed in search of existing solutions to our problem, very few met our inclusion criteria for implementation as comparison algorithms. These criteria were as follows, and are mostly made difficult by the relaxed assumptions of our problem formulation:

- The algorithm was tested on at least two real-world datasets from different domains.
- The reported results define what it means to find the correct instances, and the datasets used have labeled “correct” instances to find.
- The algorithm operates on raw data, not the output of another algorithm. E.g., we exclude works whose contribution is to refine motif lengths after-the-fact. Such algorithms could operate on our own algorithm’s output, and so are orthogonal to our work.
- The algorithm can operate on multidimensional time series. This includes distance-based methods that assume univariate time series, since they can be generalized by summing distances across all dimensions.
- The work does not assume data with domain-specific characteristics. For example, we exclude works that neglect z-normalization of subsequences, since this is essential in many applications.
- The algorithm is no more than quadratic in the length of the time series.

Along with the fact that many works focus on accelerating motif discovery, rather than formulating it differently (e.g., [10, 11, 19]), these criteria result in extremely few

approaches to test. Specifically, we found only the two following approaches (and variations thereon) that could be generalized to solve our problem:

1. Closest-pair motif discovery to find two seed windows, followed by instance selection using a distance threshold [14, 17]. Distance is defined as the sum of z-normalized Euclidean distances across all dimensions. We find this pair efficiently using the lower bound of Mueen [17], and determine the threshold using Minnen’s algorithm [41]. We call this algorithm *Dist*.
2. The algorithm of [1], with distances and description lengths summed across dimensions. This amounts to closest-pair motif discovery to find two seeds, followed by instance selection using a minimum description length (MDL) criterion. We call this algorithm *MDL*.

In both cases, we consider versions of the algorithms that carry out searches at lengths from M_{min} to M_{max} and use the best result from any length. This means lowest distance in the former case and lowest description length in the latter. In other words, we give them the prior knowledge that there is exactly one pattern to be found, as well as its approximate length. This replaces the heuristics for determining the number of patterns described in the original papers.

We tried many variations of these two algorithms, and use the above approaches because they worked the best.

One modification we made to *MDL* is worth noting. We did not stop the search over lengths when the closest pair at a given length had a negative information gain—since this was *usually* the case on our real-world data—but instead stopped when the best information gain using any number of instances at a length was negative. Both methods are heuristics for when to stop searching longer lengths; our modification simply prevents it from stopping far too early.

6.4 Instance Discovery Accuracy

The core problem addressed by our work is the robust discovery of the locations of multiple pattern instances within an arbitrary time series. To assess our effectiveness

in solving this problem, we evaluated the F1 score on each of the four datasets, varying the threshold τ for how much ground truth and reported instances needed to overlap in order to count as matching. In all cases, M_{min} and M_{max} were set to 1/20 and 1/10 of the time series length. As shown in Figure 6-3, we outperform the comparison algorithms for virtually all match thresholds on all datasets.

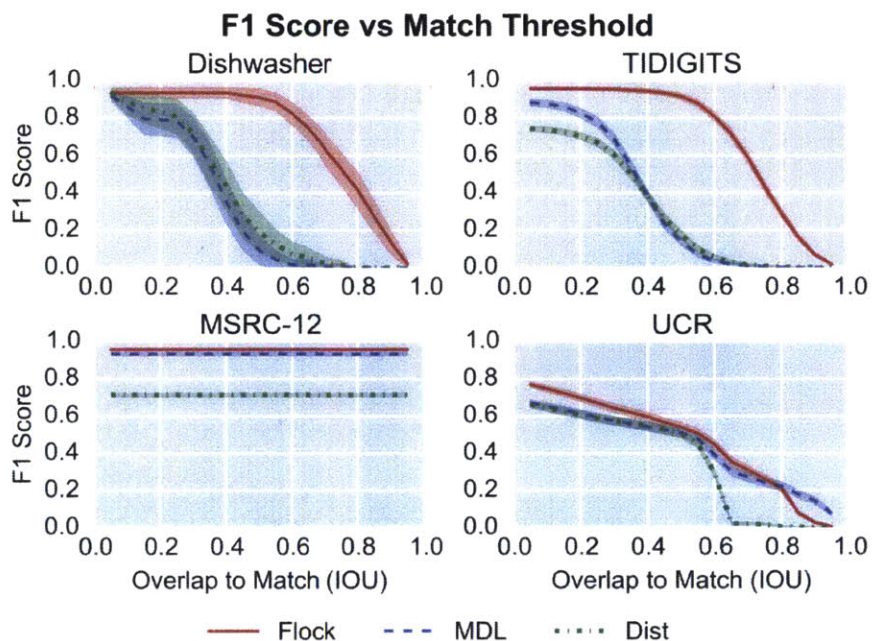


Figure 6-3: The proposed algorithm is more accurate for virtually all “match” thresholds on all datasets. Shading corresponds to 95% confidence intervals.

Note that MSRC-12 values are constant because region boundaries are not defined in this dataset (see 6.1.3). Further, the dataset on which we perform the closest to the comparisons (UCR) is synthetic, univariate, and only contains instances that are the same length. These last two attributes are what *Dist* and *MDL* were designed for, so the similar F1 scores suggest that *Flock*’s superiority on other datasets is due to its robustness to violations of these assumptions. Visual examination of the errors on this dataset suggests that all algorithms have only modest accuracy because there are often regions of random walk data that are more similar in shape than the instances are.

The dropoffs in Figure 6-3 at particular IOU thresholds indicate the typical

amount of overlap between reported and true instances. E.g., the fact that existing algorithms abruptly decrease in F1 score on the TIDIGITS dataset at a threshold near 0.3 suggests that many of their reported instances only overlap 30% with the true instances.

Our accuracy on real data is not only superior to the comparison methods, but also high in absolute terms (Table 6.1). Suppose that we consider IOU thresholds of 0.25 or 0.5 to be “correct” for our application. The former might correspond to detecting a portion of a gesture, and the latter might correspond to detecting most of it, with a bit of extra data at one end. At both of these thresholds, our algorithm discovers pattern instances with an F1 score of over 90% on real data.

Table 6.1: Flock F1 Scores are High in Absolute Terms

	Overlap $\geq .25$			Overlap $\geq .5$		
	Flock	MDL	Dist	Flock	MDL	Dist
Dishwasher	0.935	0.786	0.808	0.910	0.091	0.191
TIDIGITS	0.955	0.779	0.670	0.915	0.140	0.174
MSRC-12	0.947	0.943	0.714	0.947	0.943	0.714
UCR	0.671	0.593	0.587	0.539	0.510	0.504

An example of our algorithm’s output on the TIDIGITS dataset is shown in Figure 6-4. The regions returned (shaded) closely bracket the individual utterances of the digit “0.” The “Learned Pattern” is the feature weights V from Section 5.5, which are the increases in log probability of each element being 1 when the window is a pattern instance.

6.5 Scalability

In addition to being more accurate on both real and synthetic data, our algorithm is also much faster than the comparison algorithms. To assess this, we recorded the time our algorithm and the comparisons took to run on increasingly long sections of random walk data and the raw Dishwasher data.

In the first column of Fig 6-5, we vary only the length of the time series (N) and keep M_{min} and M_{max} fixed at 100 and 150. In the second column, we hold N constant

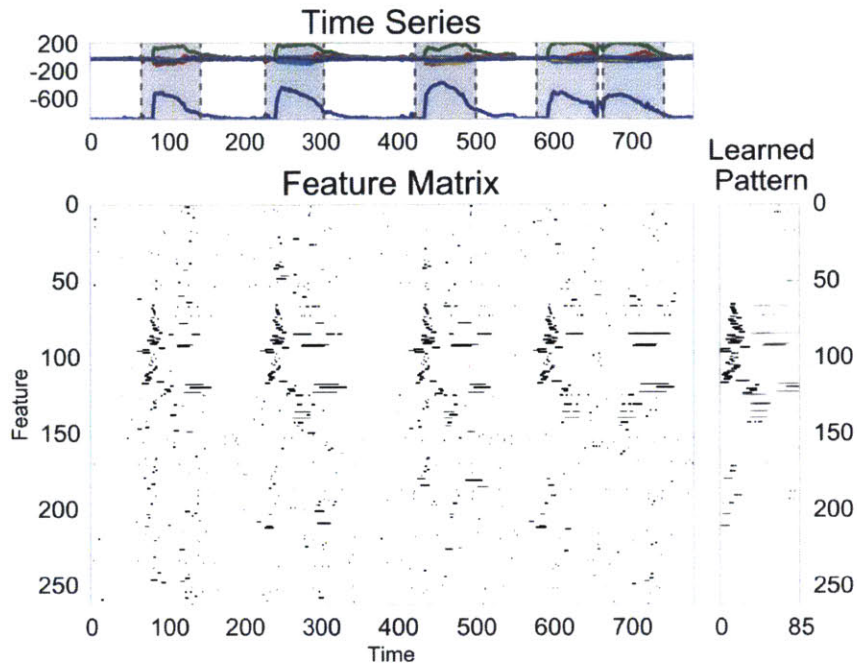


Figure 6-4: Flock output on a TIDIGITS time series. *Top*) Original time series. *Bottom*) The feature matrix Φ . *Right*) The learned feature weights. These resemble a “blurred” version of the features that repeat each time the word is spoken.

at 5000 and vary M_{max} , with M_{min} fixed at $M_{max} - 50$ so that the number of lengths searched is constant. In the third column, we fix N at 5000 and set (M_{min}, M_{max}) to $(150, 150), (140, 160), \dots, (100, 200)$.

As Figure 6-5 illustrates, our algorithm is at least an order of magnitude faster in virtually all experimental conditions. Further, it shows little or no increase in runtime as M_{min} and M_{max} are varied and increases only slowly with N . This is in line with what would be expected given our computational complexity, except with even less dependence on M_{max} . This deviation is because $D \log(M_{max}) \log(N)$ is an upper bound on the number of features used, and the actual number is lower since shapes that only occur once are discarded (c.f. 5.2.1). This is also why our algorithm is faster on the Random Walk dataset than the Dishwasher dataset; random walks have few repeating shapes, so our feature matrix has few rows.

Both $Dist$ and MDL sometimes plateau in runtime thanks to their early-abandoning techniques. $Dist$ even decreases because the lower bound it employs [17] to prune similarity comparisons is tighter for longer time series. Since they are $O(N^2)$, they are

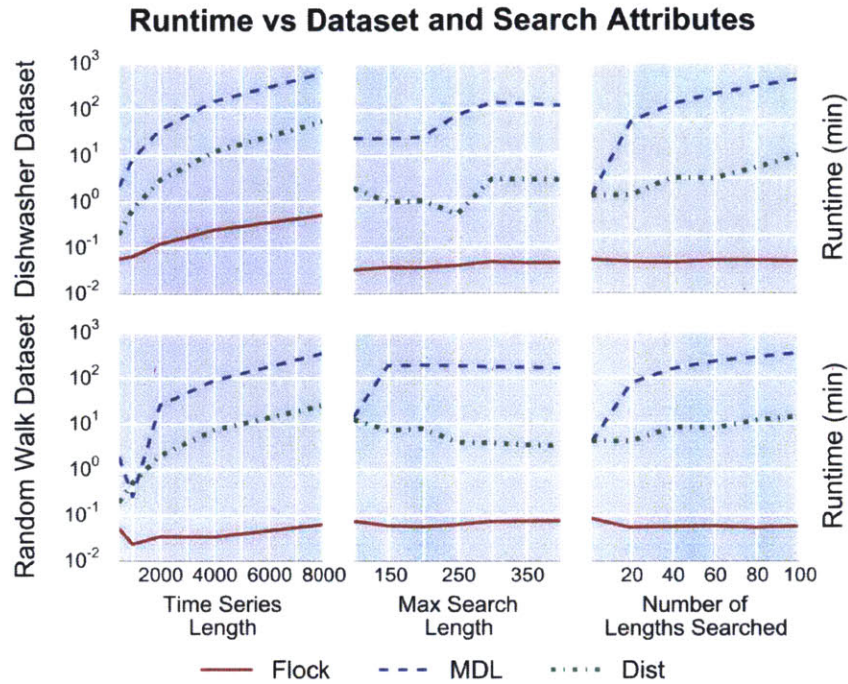


Figure 6-5: The proposed algorithm is one to two orders of magnitude faster than comparisons.

also helped by the decrease in the number of windows to check as the maximum window length M_{max} increases. This decrease benefits Flock as well, but to a lesser extent since it is subquadratic.

As with accuracy, our technique is not only better than the comparisons, but good in absolute terms—we are able to run the above experiments in minutes and search each time series in seconds (even with our simple Python implementation). Since these time series reflect phenomena spanning many seconds or hours, this means that our algorithm could be run in real time in many settings.

Chapter 7

Theoretical Basis

In this chapter, we describe the theoretical basis for Feature Flocks. Specifically, we show that the objective function 2.1 is a MAP estimate of the true pattern under particular assumptions about the generative model producing the data. We also describe connections between our objective and compression, as well as with digital filters.

To build the reader's intuition, we first consider a concrete and simpler problem. We then detail how this can be applied to time series and explore the aforementioned connections.

7.1 Intuition

Suppose that a gang of robbers has stolen something from your institution. Police have identified a group of suspects, but this group contains both robbers and random civilians (drawn iid from the general population in your city). As the local data scientist, it is up to you to determine who the robbers are. You cannot assume a certain number of robbers or anything about their characteristics (e.g., they do not necessarily have criminal records or other obvious features to look for). You can assume, however, that there are many more robbers in the group than would be expected by chance if you only sampled iid from the general population.

A priori, this is not necessarily solvable. If the robbers have nothing in particular

that distinguishes them, then there can be little hope of identifying them. However, it is probable that fellow robbers will have some sort of distinguishing features. For example, if several of the suspects all happen to have played on the same football team in high school, it is unlikely that this would happen in iid civilians, and so you might suspect that these are the robbers. If you can determine what these features are, then, you can recognize the robbers. So how could you identify these features? The above example seems sensible enough, but what is it about a given feature that makes it useful?

One rule is that the feature must be *distinguishing*—i.e., not common in the general populace. “Has brown hair” is probably a poor feature.

However, it is not enough for a feature to be unusual. Indeed, any person in the group is likely to have a number of features that are not just unusual, but unique. We cannot simply lock up, say, the one redhead.

However, if multiple people have the *same* unusual feature, we might suspect that this is not a coincidence. For example, if three of our suspects happen to be from the same neighborhood, or work for a particular company, this is unlikely under the assumption that they’re innocents drawn from the general population. This becomes even more unlikely the more suspects there are who have this trait in common—two people may work for a certain company by chance, but five almost certainly do not.

Further, if this same set of individuals has other features in common (e.g., graduating from the same high school, being the same age), this bolsters our conviction.

In short, we seek features with three properties:

1. **Rare** in general
2. **Common** in a certain subset of the people (whom we suspect are the robbers)
3. Shared across **many** (suspected) robbers

7.2 Objective Function

Formally, you are given a set X of feature descriptions of individuals, x_i , where x_i^j is the j th feature of the i th individual. Your task is to return the most probable

set of robbers, I^* , and distinguishing features, F^* . There are N individuals, at least two robbers ($|I^*| \geq 2$), and $D = |x_i|$ features. Your objective function is defined as follows:

$$I^*, F^* = \arg \max_{I, F} \prod_{i \in I} \prod_{j \in F} \frac{p(x_i^j | z_i = 1)}{p(x_i^j | z_i = 0)} \quad (7.1)$$

where z is an indicator variable describing whether $i \in I$, and different distributions are associated with each value of z . In words, we want to find a set of individuals and features such that the probability of getting all of those features across all of those individuals is much higher if the individuals are pulled from a shared “robber” distribution than the “random” general populace distribution. We will describe the family of distributions we use in solving our particular problem in following sections, but all that is required for the above objective to be sensible is that the “robber” and “random” distributions not be identical. Of course, for it to be meaningful, the “robber” distribution ought to vary with I and F and the “random” distribution ought to reflect statistics about the general populace.

This is a simplification of the time series problem in that there is no “overlap” among the x_i as there is with sliding windows in a time series. There is also no question regarding the “length” or boundaries of the x_i , since they are supplied directly as fixed-sized vectors.

7.3 Objective Function Derivation

The above objective function yields the maximum *a posteriori* (MAP) estimate of the set of robbers and their distinguishing features given a few simple assumptions.

By definition, the most probable set of objects and features is given by:

$$\arg \max_{I, F} p(I, F | X) \quad (7.2)$$

Using Bayes Theorem and dropping the denominator (since it does not affect the

argmax), we see that this is equivalent to:

$$\arg \max_{I,F} p(X|I, F)p(I, F) \quad (7.3)$$

Now suppose that our priors regarding the set of people and set of features are independent. We then have:

$$\arg \max_{I,F} p(X|I, F)p(I)p(F) \quad (7.4)$$

The first of these probabilities, $p(X|I, F)$ can be simplified further given a few reasonable assumptions. First, since we consider each object to have been drawn iid, we have:

$$p(X|I, F) = \prod_{i=1}^N p(x_i|I, F) \quad (7.5)$$

Now, we introduce the latent indicator variable z , where $z_i = 1 \iff i \in I$, to encapsulate the conditioning of x_i on I . This allows us to rewrite the previous equation as:

$$p(X|I, F) = \prod_{i \in I} p(x_i|F, z = 1) \prod_{i \notin I} p(x_i|F, z = 0) \quad (7.6)$$

$$= \prod_{i \in I} \frac{p(x_i|F, z = 1)}{p(x_i|F, z = 0)} \prod_{i=1}^N p(x_i|F, z = 0) \quad (7.7)$$

where we have multiplied in $\prod_{i \in I} p(x_i|F, z = 0)$ to get the second equation. Now let us define:

$$p(x_i|F, z = 0) = p(x_i|z = 0) \quad (7.8)$$

This reflects the idea that the probability of getting x_i from the “random” distribution is not dependent on which features we select as relevant for the “robber” distribution. E.g., if we determine that the robbers are likely to be from New York, this has no

bearing on the probability that a random person is from New York. This definition renders the second product in (7.6) independent of both I and F , allowing us to drop it without affecting our original argmax:

$$p(X|I, F) \propto \prod_{i \in I} \frac{p(x_i|F, z = 1)}{p(x_i|z = 0)} \quad (7.9)$$

We can reasonably make another assumption to simplify this even further. If only the features $j \in F$ are determined by an object's status as a robber, and others are determined by whatever processes produce the features of the general population, then one could reasonably factor the overall probability of an object into the probability of the robber and non-robber features. That is, we can define:

$$p(x|F, z) \equiv p(x^F|z)p(x^{-F}|z) \quad (7.10)$$

$$p(x^{-F}|z = 1) \equiv p(x^{-F}|z = 0) \quad (7.11)$$

where x^F and x^{-F} are the features of x in and outside of the set F , respectively. I.e., $x^F = \langle x^j \rangle, j \in F$ and $x^{-F} = \langle x^j \rangle, j \notin F$. Intuitively, this means that if brown hair has nothing to do with being a robber, then the probability of having brown hair is the same for robbers and non-robbers. Further, the relevant and irrelevant features are independent of one another conditioned on whether an individual is a robber. Using these properties, we can simplify 7.9 into:

$$p(X|I, F) \propto \prod_{i \in I} \frac{p(x_i^F|z = 1) p(x_i^{-F}|z = 1)}{p(x_i^F|z = 0) p(x_i^{-F}|z = 0)} \quad (7.12)$$

$$= \prod_{i \in I} \frac{p(x_i^F|z = 1)}{p(x_i^F|z = 0)} \quad (7.13)$$

Now consider what happens if we assume that $p(x_i^F|z)$ factorizes—i.e., that the individual features are independent when conditioned on z . This gives us:

$$p(X|I, F) \propto \prod_{i \in I} \prod_{j \in F} \frac{p(x_i^j|z = 1)}{p(x_i^j|z = 0)} \quad (7.14)$$

This yields the final objective function:

$$I^*, F^* = \arg \max_{I, F} p(I)p(F) \prod_{i \in I} \prod_{j \in F} \frac{p(x_i^j | z = 1)}{p(x_i^j | z = 0)} \quad (7.15)$$

If we assume uniform priors $p(I)$ and $p(F)$, so that these terms do not affect the argmax, we recover the original objective function in 7.1. Thus, this objective is a MAP estimate (or, ignoring the priors entirely, a maximum likelihood estimate).

7.4 Binary Case

The above analysis works for any distributions satisfying the stated independence assumptions. To identify the robbers, however, we must select distributions to fit to the data. Throughout the rest of this work, we will use *Bernoulli* distributions and binary features.

Specifically, we define $p(x_i^j | z = 0) \sim \text{Bernoulli}(\theta_{0j})$ and $p(x_i^j | z = 1) \sim \text{Bernoulli}(\theta_{1j})$, where θ_0 is any set of parameters computed independent of I and F , and θ_1 is the optimal set of parameters (in this case, simply the maximum likelihood estimate) for $\{x_i^j | i \in I\}$. We also drop $p(F)$ and $p(I)$ for now—the latter will be relevant when we generalize to time series. Taken together, these changes give the objective function:

$$I^*, F^* = \arg \max_{I, F} \prod_{i \in I} \prod_{j \in F} \frac{p(x_i^j | \theta_{1j})}{p(x_i^j | \theta_{0j})} \quad (7.16)$$

Expanding the two cases for each feature, we have:

$$\arg \max_{I, F} \prod_{i \in I} \prod_{j \in F} \frac{\theta_{1j}^{I\{x_i^j=1\}} (1 - \theta_{1j})^{I\{x_i^j=0\}}}{\theta_{0j}^{I\{x_i^j=1\}} (1 - \theta_{0j})^{I\{x_i^j=0\}}} \quad (7.17)$$

Letting c_j denote the number of times feature j is 1, $\sum_{i \in I} x_i^j$, and k denote $|I|$, this

can be rewritten as:

$$\arg \max_{I,F} \prod_{j \in F} \frac{\theta_{1j}^{c_j} (1 - \theta_{1j})^{(k-c_j)}}{\theta_{0j}^{c_j} (1 - \theta_{0j})^{(k-c_j)}} \quad (7.18)$$

Now, let us take the log of this objective function (which yields the same argmax, since the log is a monotonic function of its argument):

$$\arg \max_{I,F} \sum_{j \in F} [c_j \log(\theta_{1j}) + (k - c_j) \log(1 - \theta_{1j})] - \sum_{j \in F} [c_j \log(\theta_{0j}) + (k - c_j) \log(1 - \theta_{0j})] \quad (7.19)$$

Simplifying, this becomes:

$$\arg \max_{I,F} \sum_{j \in F} [c_j (\log(\theta_{1j}) - \log(\theta_{0j}))] \quad (7.20)$$

$$+ (k - c_j) (\log(1 - \theta_{1j}) - \log(1 - \theta_{0j}))] \quad (7.21)$$

Another simplification is possible if the data is sparse. In this case, it is not meaningful for a feature to be absent. E.g., if a feature is only present in 10% of the population, the fact that it is not present in three robbers is not a robust indication that being a robber affects it. For example, if none of the three robbers have been to Greenland, we should not take failure to visit this country as a sign of being a robber. This is crucial, since one could enumerate countless features of this nature and incorporating them would almost certainly drown out the features that were meaningful.¹ Dropping the term for feature absence in the above equation, we have:

$$\arg \max_{I,F} \sum_{j \in F} c_j (\log(\theta_{1j}) - \log(\theta_{0j})) \quad (7.22)$$

This equation says that we would like to find robbers x_i and features j such that

¹Note that it is not the fact that the feature is absent per se that causes us to ignore it, but that, given a sparse feature set, absence is far more common in the general populace. For particular features where this is not true and that may be predictive (e.g., lacking the “has a high school diploma” feature may be associated with being a robber), one could invert the feature in order to include it in the objective. I.e., one could encode a “lacks a high school diploma” feature.

x_i^j happens both many times (so that c_j is large) and much more often than would occur by chance (so that $\log(\theta_{1j}) - \log(\theta_{0j})$ is large). This is the objective function given in the problem definition section.

7.5 Relationship to Compression

The binary objective is closely related to the problem of compression. Suppose that θ_0 is set to the empirical probabilities across the whole dataset, so that $\theta_{0j} = p(x^j)$. Suppose further than we can only select features that are always present, so that $c_j = |I|$ and $\theta_1 = \mathbf{1}$. Then the objective becomes:

$$\arg \max_{I,F} \sum_{j \in F} -|I| * \log(\theta_{0j}) \quad (7.23)$$

$$= \arg \max_{I,F} \sum_{j \in F} -|I| * \log(p(x^j)) \quad (7.24)$$

This is the number of bits to encode k occurrences of all features j , with codeword length determined by the empirical probability $p(x^j)$. In other words, if each feature j would otherwise be encoded at a cost of $\log(p(x^j))$ when present, this is the number of bits saved by substituting for 1s in $x_i^F, i \in I$ a symbol composed of ones at the indices F . Thus, this objective is maximized when we find the largest set of ones that occur together the most times throughout the data.

7.6 Generalizing to Multiple Models

The analysis so far has considered only distinguishing between two models: a “robber” model and a “random” model. This can be generalized to many models. Specifically, while we are still interested only in identifying the robbers, non-robbers may be drawn from many distributions, rather than a naive aggregate one. This is a more realistic setup, but a more challenging one mathematically. It requires us to ensure that suspected robbers not only fail to appear random, but fail to resemble any competing distribution as well.

Formally, let $\mathcal{M} = \{\mathbf{m}_i\}$ be some set of possible models, and θ_1 continue to be the robber model. For ease of reference, further define $X_I \equiv \{x_i, i \in I\}$ and $X_{-I} \equiv \{x_i, i \notin I\}$. We seek to find:

$$\arg \max_{I,F} p(I)p(F) \frac{p(X|I, F, \theta_1)}{\sum_{\mathbf{m} \in \mathcal{M}} p(X|I, F, \mathbf{m})p(\mathbf{m})} \quad (7.25)$$

$$= \arg \max_{I,F} p(I)p(F) \prod_{i \in I} \frac{p(x_i|I, F, \theta_1)}{\sum_{\mathbf{m} \in \mathcal{M}} p(x_i|I, F, \mathbf{m})p(\mathbf{m})} \quad (7.26)$$

Since there can be many models, this quantity is challenging to optimize. Therefore, we will approximate it as follows, using the intuition that one model likely assigns X_I much greater probability than the others:²

$$\arg \max_{I,F} \min_{\mathbf{m} \in \mathcal{M}} p(I)p(F) \prod_{i \in I} \frac{p(x_i|I, F, \theta_1)}{p(x_i|I, F, \mathbf{m})} \quad (7.27)$$

That is, we approximate the sum with the maximum element and set its prior $p(\mathbf{m}) = 1$.

We must now define the set \mathcal{M} . One could use any number of possible sets of models, but we will use a simple instance-based approach. In our robber-finding example there are reasonable alternatives, since the general populace is large and one could learn a set of models for it. However, when generalizing to time series, we will assume limited data, and so it will be impossible to learn a set of parametric distributions (or even how many such distributions there should be) with accuracy.

Concretely, we take each object X_{-I} as the “centroid” of a model \mathbf{m} . This means that the probability of a suspected robber x_i coming from a distribution other than the robber distribution is based on its similarity to the most similar non-robber object. For ease of reference, we term this object the *nearest enemy*, and the suspected robber

²This need not be true, but it is reasonable to think the robbers might resemble, e.g., an “unmarried males” distribution much more than any other.

In the time series case, this approximation is even more likely to work well. Because our competing distributions will be based on other regions in the data and there is limited data, there are relatively few competing distributions possible. If there are k such distributions, then their summed probability is at most k times as great as that of the maximum probability in the worst case. This small factor is dominated by the differences in likelihood across distributions that arise from the presence of dozens or hundreds of features per object.

x_i the query.

But what defines this similarity, and how do we get a probability from it? One approach would be to have a kernel function that, say, returned a probability between 0 and 1 depending on the Hamming distance between the query and nearest enemy. This could work, but we would have to learn the mapping of distances to probabilities. We would also consider differences in all features equally significant³, while in reality, some features may be more variable than others.

What we do instead is recycle the learned distribution for the robbers, described by θ_1 . Specifically, letting x_k denote the nearest enemy, we define:

$$p(x_i|I, F, \mathbf{m}_k) \equiv p(x_k|I, F, \theta_1) \quad (7.28)$$

In words, we approximate the probability of the query coming from the distribution of the nearest enemy with the probability of the nearest enemy coming from the distribution of the query (i.e., the robber distribution). This yields the objective:

$$\arg \max_{I, F} \min_{k \notin I} p(I)p(F) \prod_{i \in I} \frac{p(x_i|I, F, \theta_1)}{p(x_k|I, F, \theta_1)} \quad (7.29)$$

$$(7.30)$$

Taking the log, this becomes:

$$\begin{aligned} \arg \max_{I, F} \min_{k \notin I} & \log(p(I)) + \log(p(F)) \\ & - |I| \log(p(x_k|I, F, \theta_1)) \\ & + \sum_{i \in I} \log(p(x_i|I, F, \theta_1)) \end{aligned}$$

In the binary case (using the previously described approximations and indepen-

³Assuming we don't learn the kernel, which, again, would add complexity and be challenging to do accurately with limited data

dencies), we have:

$$\begin{aligned} \arg \max_{I,F} \min_{k \notin I} \log(p(I)) + \log(p(F)) \\ + \sum_{j \in F} (c_j - |I|x_k^j) \log(\theta_{1j}) \end{aligned}$$

Unfortunately, this objective is degenerate—since $\log(\theta_{1j})$ is always negative, it is maximized when $I = F = \{\}$. Thus, we instead maximize:

$$\arg \max_{I,F} \min_{k \notin I} \log(p(I)) + \log(p(F)) \tag{7.31}$$

$$+ \sum_{j \in F} (c_j - |I|x_k^j) \log(\theta_{1j} - \theta_{0j}) \tag{7.32}$$

In other words, we continue looking for features that are better explained by the robber distribution than the noise distribution, but exclude those that are present in the nearest enemy distribution.

To handle the case when all objects are suspected of being robbers (and thus there is no object eligible to be the enemy), we add to the dataset a “dummy” enemy x_k such that $x_k^j = E[\theta_0]$. This object represents the “expected” enemy given features drawn from the “noise” distribution. Empirically, this object may also be chosen as the nearest enemy if the other possible enemies are extremely sparse or otherwise share few features with the suspected robbers.

7.7 Generalizing to Time Series

The above objective and analysis can be applied to time series via a straightforward reduction. Namely, instead of having each object x_i be a binary feature representation of an individual, it is a binary feature representation of a region (a, b) of the time series. This makes no difference mathematically—it simply changes what each object represents.

However, one mathematical alteration is necessary for the results to be meaningful.

Namely, we use the prior $p(I)$ to prevent overlapping, excessively long, or excessively short regions. That is:

- Given a minimum spacing M_{space} between the starts of regions, $p(I) = 0$ if $|a_1 - a_2| < M_{space}$ for any regions $(a_1, b_1), (a_2, b_2) \in I$. In practice we set $M_{space} = M_{min}$.
- Given minimum and maximum region lengths M_{min} and M_{max} , $p(I) = 0$ if $|b_1 - a_1| < M_{min} \vee |b_1 - a_1| > M_{max}$ for any $(a_1, b_1) \in I$.
- $p(I)$ is otherwise uniform.

This yields the final objective (without enemies):

$$\arg \max_{I, F} \log(p(I)) + \sum_{j \in F} c_j (\log(\theta_{1j}) - \log(\theta_{0j})) \quad (7.33)$$

With enemies, this becomes:

$$\arg \max_{I, F} \min_{k \notin I} \log(p(I)) + \sum_{j \in F} (c_j - |I| x_k^j) \log(\theta_{1j} - \theta_{0j}) \quad (7.34)$$

7.8 Flock Filters

In the time series case, maximizing the first of the above objectives (7.33) can be interpreted as learning a digital filter that selectively responds to pattern instances.

Let $\mathbf{w} = V(\log\theta_1 - \log\theta_0)$, where V is a binary diagonal matrix that zeros out all the features that aren't included in F . Then $\mathbf{w}^\top \mathbf{x}$ is the difference in log likelihoods of getting \mathbf{x} under the “pattern” distribution and “non-pattern” distribution, which is also the log odds of it being a pattern instance given equal noise and pattern priors. If the feature representation x_i is taken not as a vector but a 2D window, \mathbf{w} is a 2D digital filter that can be slid along the time dimension to assess each window of data. This is the mathematical meaning of the “Learned Pattern” in Figure 6-4.

Chapter 8

Discussion

In this chapter, we summarize our work, describe its limitations, and suggest directions for future research.

8.1 Summary

This thesis describes an algorithm to discover repeating patterns in multivariate time series. We formulate this problem in greater generality than has been done in related work, and introduce an efficient algorithm to solve this generalized problem. The key to our approach is a change of representation, which allows many possible signals and pattern lengths to be examined simultaneously. We evaluated our technique using several publicly available datasets and found that it achieved state-of-the-art speed and accuracy.

8.2 Contributions

Our contributions can be summarized as follows:

- We provide a formulation of time series pattern discovery under more relaxed assumptions than those of existing techniques. In particular, we allow multivariate time series, patterns that affect only subsets of variables, instances of varying lengths, and categorical variables.

- We describe and experimentally verify an efficient algorithm to discover patterns under this formulation. This algorithm is both faster than existing algorithms and fast enough to run in real time on batches of data. It is also highly accurate, both compared to these algorithms and in absolute terms. Finally, it requires fewer than 300 lines of code to implement.
- We provide a body of open source code and labeled time series that can be used to rigorously assess future algorithms for time series pattern discovery. It can also be used to reproduce or extend our own work.

8.3 Limitations

Our results demonstrate that the proposed algorithm can achieve high accuracy for problems with a certain structure. However, there are limits on what problems qualify. In particular, our problem formulation requires:

1. That there only be one pattern in the data.
2. That there exist known bounds M_{min} and M_{max} , $M_{min} \geq M_{max}/2$, on the lengths of all instances of this pattern.
3. That there are at least two instances of this pattern.

Further, our algorithm to solve this problem requires that:

1. Instances comprise an appreciable fraction of the data (at least 10% of the data consisted of instances in all time series used in our experiments).
2. There are few regions of data that appear highly dissimilar from noise, but are not pattern instances. E.g., there are not large but irrelevant spikes in the values of certain variables.
3. The feature representation captures the similarity of pattern instances. If the instances do not share more features than do non-instance regions, the algorithm cannot detect the instances.

8.4 Future Work

Future work could proceed in several directions:

1. **Discovering Multiple Patterns.** The current formulation assumes the presence of only a single pattern. It could be extended to find multiple patterns by iteratively extracting all instances of each single pattern, but this is likely to be error-prone. Because our objective function encourages explaining as much of the data as possible, early iterations are likely to attempt to explain multiple patterns. Further, if a given iteration fails to find all of its instances, those left over will likely confuse future iterations. In short, this approach is naive and any errors it makes are likely to compound as successive patterns are examined. Consequently, a unified approach that searches for multiple patterns simultaneously is desirable.
2. **Domain-specific features.** This work has described one way of constructing features, but many others could be used—the learning algorithm requires only a sparse feature matrix with values between 0 and 1. For maximum accuracy in a given domain, one could construct features based on variance, filter bank output, SAX words [24], or any number of other signal properties.
3. **Learning the features.** While the features described here are learned in the sense that shapes are extracted from the data, the features are not tied to the algorithm’s objective function. Greater accuracy might be possible with features that are selected or modified based on hypothesized patterns.
4. **Learning feature correlations.** The present objective function assumes that all features are independent. This is not necessarily the case, and one could likely obtain improved performance by learning joint distributions over the features.
5. **Compositional models.** While pattern instances can be understood as atomic entities, a more realistic model would allow them to be represented as the composition of many attributes. E.g., one could imagine representing not only the type of a gesture instance, but also that it was “slow,” “sloppy,” “left-handed,” or any number of other descriptors. Such compositionality could be achieved by stacking pattern detectors hierarchically, with lower-level detectors learning pattern primitives and

higher-level detectors learning entire patterns.

6. **Online learning.** The present learning algorithm examines all of the data in order to decide upon an optimal set of instances. This can be run on batches of data and still satisfy a real time constraint since the system is fast enough, but it would be simpler (and likely more efficient) to have a modified version of the algorithm that updated its predictions incrementally as each sample of data arrived.
7. **Leveraging existing knowledge.** The algorithm presently assumes no labels of any kind. However, in a real-world deployment, there would be phenomena for which the system already had labeled examples. Ideally, the algorithm should be able to leverage this existing knowledge to avoid returning known patterns and/or to recognize instances of these patterns. Moreover, given a set of labeled time series, it could be extended to selectively extract discriminative subsequences (c.f. [7]) in order to optimize a classification objective.
8. **Real-world deployment.** We plan to implement the algorithm in C++ and deploy it on mobile phones and fitness trackers in order to learn gestures, behaviors, and other actions in the real world. Our implementation will also support execution on remote servers in order to process many streams of user data simultaneously. We may also extend it to use GPUs for extra performance, likely in conjunction with other enhancements described in this section.

8.5 Conclusion

We have described an algorithm to efficiently and accurately locate instances of a pattern within a multivariate time series given virtually no prior information about the nature of this pattern. In particular, we assume no knowledge of where or how often the pattern occurs, what features distinguish it, or which variables it affects. Using a diverse group of publicly available datasets, we showed that this technique is fast and accurate both in absolute terms and compared to existing algorithms, despite its weaker assumptions.

In short, by applying our technique to low-level signals of various kinds, one

can discover patterns produced by high-level phenomena as diverse as spoken words, human actions, and household appliance usage without human labeling.

Bibliography

- [1] S. Yingchareonthawornchai and H. Sivaraks, “Efficient proper length time series motif discovery,” *Data Mining (ICDM)*, 2013.
- [2] M. Moazeni, B. Mortazavi, and M. Sarrafzadeh, “Multi-dimensional signal search with applications in remote medical monitoring,” *IEEE BSN 2013*, pp. 1–6, 2013.
- [3] D. Minnen, T. Starner, I. Essa, and C. Isbell, “Discovering Characteristic Actions from On-Body Sensor Data,” pp. 11–18, 2006.
- [4] Y. Chen, A. Why, G. Batista, A. Mafra-Neto, and E. Keogh, “Flying Insect Classification with Inexpensive Sensors,” *arXiv.org*, Mar. 2014.
- [5] Y. Hao, Y. Chen, J. Zakaria, B. Hu, T. Rakthanmanon, and E. Keogh, “Towards never-ending learning from time series streams,” in *KDD 2013*. New York, New York, USA: ACM Request Permissions, Aug. 2013, pp. 874–882.
- [6] H. T. Cheng, F. T. Sun, M. Griss, P. Davis, and J. Li, “Nuactiv: Recognizing unseen new activities using semantic attribute-based learning,” *ICMS 2013*, 2013.
- [7] B. Hu, Y. Chen, and E. Keogh, “Time series classification under more realistic assumptions,” in *SDM 2013*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2013, pp. 578–586.
- [8] J. Zakaria, A. Mueen, and E. Keogh, “Clustering time series using unsupervised-shapelets,” *Data Mining (ICDM)*, 2012.
- [9] L. Ulanova, N. Begum, and E. Keogh, “Scalable Clustering of Time Series with U-Shapelets,” *SDM 2015*, pp. 900–908, 2015.

- [10] A. Mueen, E. J. Keogh, Q. Zhu, S. Cash, and M. B. Westover, “Exact Discovery of Time Series Motifs.” *SDM*, 2009.
- [11] 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*, vol. 22, no. 1-2, pp. 73–105, Apr. 2010.
- [12] N. Begum and E. Keogh, “Rare time series motif discovery from unbounded streams,” *Proceedings of the VLDB Endowment*, vol. 8, no. 2, Oct. 2014.
- [13] J. Lin, E. Keogh, J. Lonardi, and P. Patel, “Finding motifs in time series,” *KDD 02*, 2002.
- [14] D. Minnen, C. Isbell, I. Essa, and T. Starner, “Detecting Subdimensional Motifs: An Efficient Algorithm for Generalized Multivariate Pattern Discovery,” in *ICDM 2007*. IEEE Computer Society, 2007, pp. 601–606.
- [15] D. Minnen, C. L. Isbell, I. Essa, and T. Starner, “Discovering multivariate motifs using subsequence density estimation and greedy mixture learning,” in *AAAI 07*. AAAI Press, Jul. 2007.
- [16] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan, “Detecting time series motifs under uniform scaling,” in *ACM SIGKDD 2007*. New York, New York, USA: ACM Request Permissions, Aug. 2007, p. 844.
- [17] A. Mueen, “Enumeration of Time Series Motifs of All Lengths,” in *ICDM 2013*. IEEE, 2013, pp. 547–556.
- [18] L. Ye and E. Keogh, “Time series shapelets: a new primitive for data mining,” in *ACM SIGKDD 2009*. New York, New York, USA: ACM Request Permissions, Jun. 2009, pp. 947–956.
- [19] B. Chiu, E. Keogh, and S. Lonardi, “Probabilistic discovery of time series motifs,” in *ACM SIGKDD 2003*. New York, New York, USA: ACM Request Permissions, Aug. 2003, pp. 493–498.

- [20] S. P. Dr and E. Ramanujam, “Naïve Bayes Classifier for ECG Abnormalities Using Multivariate Maximal Time Series Motif,” *Procedia - Procedia Computer Science*, vol. 47, pp. 222–228, 2015.
- [21] H.-L. Nguyen, W.-K. Ng, and Y.-K. Woon, “Closed motifs for streaming time series classification,” *Knowledge and Information Systems*, Jun. 2013.
- [22] A. Reinhardt and S. Koessler, *PowerSAX: Fast motif matching in distributed power meter data using symbolic representations*. IEEE, 2014.
- [23] Y. Tanaka, K. Iwamoto, and K. Uehara, “Discovery of Time-Series Motif from Multi-Dimensional Data Based on MDL Principle,” *Machine Learning*, vol. 58, no. 2-3, pp. 269–300, Feb. 2005.
- [24] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, “A symbolic representation of time series, with implications for streaming algorithms,” in *DMKD 2003*. New York, New York, USA: ACM, Jun. 2003.
- [25] J. Shieh and E. Keogh, “iSAX: disk-aware mining and indexing of massive time series datasets,” *Data Mining and Knowledge Discovery*, vol. 19, no. 1, Aug. 2009.
- [26] Y. Li and J. Lin, *Approximate variable-length time series motif discovery using grammar inference*. New York, New York, USA: ACM, Jul. 2010.
- [27] P. Nunthanid, V. Niennattrakul, and C. A. Ratanamahatana, *Parameter-free motif discovery for time series data*. IEEE, 2012.
- [28] Y. Mohammad, Y. Ohmoto, and T. Nishida, “G-SteX: Greedy Stem Extension for Free-Length Constrained Motif Discovery,” in *Advanced Research in Applied Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 417–426.
- [29] D. Minnen, T. Starner, J. A. Ward, P. Lukowicz, and G. Tröster, *Recognizing and Discovering Human Actions from On-Body Sensor Data*. IEEE, 2005.

- [30] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, “Learning time-series shapelets,” in *ACM SIGKDD 2014*. New York, New York, USA: ACM Press, 2014, pp. 392–401.
- [31] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, “Classification of time series by shapelet transformation,” *Data Mining and Knowledge Discovery*, vol. 28, no. 4, pp. 851–881, May 2013.
- [32] T. Rakthanmanon and E. Keogh, “Fast shapelets: A scalable algorithm for discovering time series shapelets,” in *SDM 2013*, Philadelphia, 2013.
- [33] E. Jones, T. Oliphant, and P. Peterson, “SciPy: Open source scientific tools for Python,” 2014.
- [34] T. Oliphant, “Numba python bytecode to LLVM translator,” in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2012.
- [35] Feature Flocks Homepage. [Online]. Available: <http://smarturl.it/featureFlocks>
- [36] R. G. Leonard and G. Doddington, “Tidigits speech corpus,” *Texas Instruments, Inc*, 1993.
- [37] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, “librosa: Audio and Music Signal Analysis in Python,” in *Proceedings of the 14th Python in Science Conference*, 2015.
- [38] S. Makonin, F. Popowich, L. Bartram, B. Gill, and I. V. Bajic, “AMPds: A public dataset for load disaggregation and eco-feedback research,” in *Electrical Power & Energy Conference (EPEC), 2013 IEEE*. IEEE, 2013, pp. 1–6.
- [39] S. Fothergill, H. M. Mentis, P. Kohli, and S. Nowozin, “Instructing people for training gestural interactive systems,” in *CHI*, J. A. Konstan, E. H. Chi, and K. Höök, Eds. ACM, 2012, pp. 1737–1746.

- [40] E. J. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei, and C. A. Ratanamahatana. (2011) The UCR Time Series Classification/Clustering Homepage. [Online]. Available: http://www.cs.ucr.edu/~eamonn/time_series_data/
- [41] D. Minnen, T. Starner, I. Essa, and C. Isbell, "Improving Activity Discovery with Automatic Neighborhood Estimation," pp. 1–6, Nov. 2006.