EARLY CLASSIFICATION ON TEMPORAL SEQUENCES

by

Zhengzheng Xing

M.Sc., University of Windsor, 2006

B.Eng, Beijing Institute of Technology, 2004

a Thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Fall 2010

APPROVAL

Name:	Zhengzheng Xing
Degree:	Doctor of Philosophy
Title of Thesis:	Early Classification On Temporal Sequences

Examining Committee: Dr. Hafer Lou
Chair

____

Dr. Jian Pei, Associate Professor, Senior Supervisor

____

Dr. Ke Wang, Professor, Supervisor

____

Dr. Martin Ester, Professor, SFU Examiner

____

Dr. Xingquan Zhu, External Examiner,
Associate Professor, Faculty of Engineering and Information Technology,
University of Technology, Sydney

Date Approved: November 30, 2010

Abstract

Early classification of temporal sequences has applications in, for example, health informatics, intrusion detection, anomaly detection, and scientific and engineering sequence data monitoring. Comparing to learning conventional sequence classifiers, learning early classifiers is a more challenging task and has not been systematically studied before.

In this work, we identify the problem of early classification and develop a series of classifiers for temporal sequence early classification. The proposed classifiers are designed for different types of temporal sequences including symbolic sequences and time series. Furthermore, the proposed classifiers have several desirable characteristics which are useful in different application scenarios. We evaluate our approaches on a broad range of real data sets and demonstrate that the classifiers can achieve competitive classification accuracies with great earliness. Also, the classifiers can extract interpretable features from sequences for better understanding.

To my family

Nothing in life is to be feared, it is only to be understood. Now is the time to understand more, so that we may fear less. — Marie Curie

Acknowledgments

I wish to express my deep gratitude to my senior supervisor, Dr. Jian Pei. I thank him for his continuous encouragement, confidence and support, and sharing with his knowledge and experience. Dr. Pei's guidance enables me to hurdle all the obstacles in the completion of this research work. Furthermore, he has taught me how to maximize creativity and stay persistent in the process of creating and implementing new ideas. I believe what he has trained me in graduate school will benefit my career all the time.

I am very thankful to my supervisor Dr. Ke Wang, my thesis examiner Dr. Martin Ester, and my external reader, Dr. Xingquan Zhu, for their insightful comments and advice for my research and thesis. I would also like to thank Dr. Lou Hafer to chair my thesis defense.

Part of this work is done in collaboration with Dr. Philip S. Yu and Dr. Guozhu Dong. I thank them for the knowledge and skills they imparted through the collaboration. I would also like to thank Dr. Eamonn Keogh who gave me constructive suggestions on my thesis.

I would also like to thank many people in our department, support staff and faculty, for always being helpful over the years. I thank my friends at Simon Fraser University for their accompany and help. A particular acknowledgement goes to Kathleen Tsoukalas, Guanting Tang, Bin Jiang, Bin Zhou, Ming Hua, Carrie (Cong) Wang, Luping Li, Zhenhua Lin, Yi Cui, Hossein Maserrat, Brittany Nielsen, Junqiang Liu and Rong Ge. They have made my graduate school enjoyable and memorable.

Last but not least, I am very grateful to my parents, my husband, my grandfather and my aunt who always support, encourage, and tolerate me without limits. Their love accompanies me wherever I go.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Classification is an important research area in machine learning and data mining. The task of classification is to first learn a prediction model from training examples, and then automatically classify unseen examples using the learned model [24]. Classification has various real world applications, such as spam email filtering, prostate cancer detection, and handwritten recognition [24].

Classification has been extensively studied on structured data where a training example is represented as a vector of attributes. The values of attributes are usually numerical or categorical. However, in many applications, data in more complex forms has been collected for classification. The data in complex forms may be unstructured or semi-structured, such as biological sequence data, time series data, text data, and multimedia data [22]. Generally, this thesis falls in the categories of classifying *sequence* data.

Simply, *a sequence is an ordered list of events.* The needs of classifying sequence data arise in many real-world applications, such as protein function classification, intrusion detection, and warning systems in critical care medicine. The broad range of sequence data and the different application scenarios keep posing various challenges for sequence classification. Driven by real world applications, in the thesis, we formulate and study a new sequence classification task, *early classification on temporal sequences.*

1.1 Motivation and Problem Definition

In many applications, the events in sequences follow a natural temporal order. We refer these sequences as *temporal sequences*. An event can be represented as a discrete value, a numerical value, or a complex data object. In this thesis, for a temporal sequence, if each event is a numerical value, we refer it as a *time series*; if each event is a discrete value, we refer it as a *symbolic temporal sequence*.

Most existing works on temporal sequence classification focused on *conventional off-line sequence classification*. A complete sequence is obtained before classification and is then classified based on the whole sequence. However, in some applications, in the process of receiving a temporal sequence online event by event, it is desired to monitor the temporal sequence and predict its class label as early as possible on the fly. We call this problem *early classification on temporal sequences*.

In a retrospective study of the infants admitted to a neonatal intensive care unit, it was found that the infants had abnormal heart beating time series patterns 24 hour before the doctor finally diagnosed them with sepsis [21]. Monitoring the heart beating time series data and classifying the time series data as early as possible may lead to earlier diagnosis and effective therapy. As another example, Bernaille *et al.* [7] showed that by only observing the first five packages of a TCP connection, the application associated with the traffic flow can be classified. The applications of online traffic can be identified without waiting for the TCP flow to end. In activity recognition [73], early classification may be used for detecting anomaly behaviors as early as possible. Sequence classification techniques have been applied for quality control and fault detection for semiconductor manufacture industry [8]. It is interesting to use an early classifier to constantly monitor the manufacture data and prevent serious faults ahead of time. Generally, early classification of temporal sequences has applications in anomaly detection, intrusion detection, health informatics, and scientific and engineering sequence data monitoring.

1.2 Challenges

To enable users to utilize the results of an early classifier, we argue that an early classifier is expected to meet two requirements.

- An early classifier should be able to affirm the *earliest* time of *reliable* classification so that the early predictions can be used for further actions.
- An early classifier should retain an accuracy comparable to that of a classifier using the full length temporal sequences or some user-specified accuracy thresholds.

Learning early classifiers for temporal sequences is a more challenging task than learning conventional sequence classifiers. Conventional sequence classifiers are built for optimizing classification accuracy and classifying unlabeled sequences off-line given the whole sequences. To learn an early classifier, we not only focus on learning an accurate model for classification, but also aim at building classifiers to achieve the earliness in classification. Conventional sequence classifiers may not achieve the goal for early classification. We use Example 1 to explain the reason.

Example 1 *In Figure 1.1, we list a sequence training data set which has two classes, the positive class and the negative class. The sequences are composed by events in alphabet =* $\{A, B, C, D\}$*. We suppose the sequences are temporal sequences. The order of events in the sequences is the temporal order. For conventional sequence classification, we can build a simple decision tree classifier. We consider all the possible length two subsequences as attributes of sequences. Each attribute has two values, absence and presence. To build a decision tree, we want to choose the attribute with the best information gain at the root. Subsequence* CD *has the best information gain since* CD *appears in all sequences in the negative class and is absent in all sequences in the positive class. As shown in Figure 1.1, a simple decision tree with only the root node is built. For a sequence to be classified, if it has a subsequence* CD*, it belongs to the negative class. Otherwise, it is classified into the positive class.*

By observing the training data, we can see that, actually, subsequence AB *is shared by* 75% *of sequences in the negative class but not any sequence in the positive class. Furthermore,* AB *always precedes* CD*. By using* AB*, we can classify some examples in the negative*

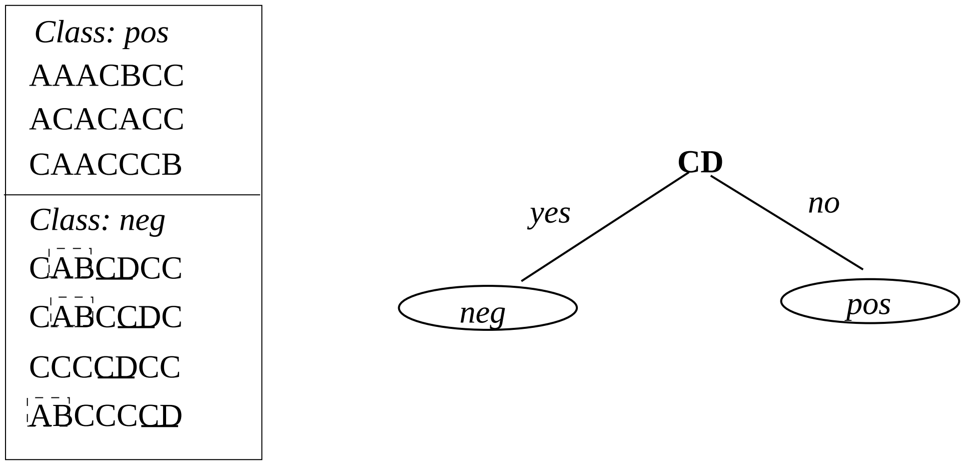


Figure 1.1: A decision tree example.

class earlier than using CD. However, in the decision tree, subsequence AB is not utilized. On the other hand, for the positive class, by using the decision tree, we cannot achieve early classification. Since without observing a whole sequence, we cannot know if the sequence contains CD or not. However, for the positive class, some features may be useful for early classification. For example, subsequence AA appears in $2/3 = 66.7\%$ examples in the positive class but never in the negative class, which may be used for early classification for the positive class. From the above example, we can see that conventional sequence classifiers may not well serve the purpose of early classification. ■

In the classification step, instead of doing off-line classification given a whole sequence, we do classification on the fly when new events in the sequence arrive. The challenge is on how to decide when to generate a reliable class label given the received events in a sequence. Can we just generate a class label whenever a new event is received? We argue that early classification should not only focus on how to generate a classification result on a prefix but should also learn from the data if a prefix contains sufficient information for a reliable classification and when is the earliest time for a reliable classification. Only a reliable early classification result can be trusted and used by users for further actions. We will explain this in more details in Section 2.1.

1.3 Contributions

In this thesis, we study the problem of early classification on temporal sequences. In particular, we make the following contributions.

- We formulate a new problem, early classification on temporal sequences. Although some previous works mentioned the concept of early classification [13, 9], they did not actually define and solve this problem. More details are provided in Section 2.1.
- We develop a series of classifiers for temporal sequence early classification. The proposed classifiers are designed for different types of temporal sequences including symbolic sequences and time series. Furthermore, the proposed classifiers have several desirable characteristics which are useful in different application scenarios.
- We evaluate early classifiers on a broad range of real data sets and demonstrate that the classifiers can achieve competitive classification accuracies with great earliness. Also, some interpretable features are extracted from sequences. The experiments validate that early classification is feasible and useful.

Some major components of this thesis were published as conference papers. Particularly, the major results in Chapter 3 were published in [68] and the major results in Chapter 4 were published in [70]. Part of contents of Chapter 2 were published as a survey paper in [69].

1.4 Organization of the Thesis

The remainder of thesis is structured as follows:

- In chapter 2, we review the related works and explain how they are related to this thesis.

- In chapter 3, we focus on early classification for symbolic sequences. A rule based classifier, SCR, and an extended decision tree, GSDT, are proposed for early classifying symbolic temporal sequences. The classifiers are built from selected features from sequences with good early classification utilities.

- In chapter 4, we present an instance based early classifier for time series. We extend the simple 1NN classifier by adding a learning phase for the instance based classifier. We aim to build an early classifier which is as accurate as 1NN classifier and achieves early classification.

- Although an instanced based early classifier has been developed for time series in Chapter 4, feature based early classifiers have better interpretability. In Chapter 5, we extract features from time series and build a rule based classifier for time series early classification. Competitive results and some meaningful features have been obtained in experiments.

- In Chapter 6, we summarize the thesis and discuss the future works.

Experiments are included in Chapter 3, 4, and 5 to demonstrate that the classifiers we proposed can achieve competitive classification accuracies with great earliness.

Chapter 2

Related Work

2.1 Early classification

To the best of our knowledge, Diez *et al.* [13] first mentioned the concept of early classification for time series. They proposed a time series classifier based on literals on temporal intervals and boosting. The classifier is not dedicated for early classification. They divided time series into several time intervals and described each interval by a predicate. For example, a predicate $always(Variable, Region, Begining, End)$ means that in the interval between *Beginning* and *End*, the *variable* is always in this *Region*. For two-class classification, a predicate is viewed as binary base classifier given the predicate is satisfied or not. Ada boost [19] was used to ensemble the base classifiers as a linear combination of them with different weights. They referred early classification as classifying partial examples which are the prefixes of the complete time series. To classify partial examples, they simply ignored the predicates on unavailable suffixes and only used the linear combination of the available predicates for classification.

Aníbal *et al.* [9] applied a case based reasoning method to classify time series to monitor system failures in a simulated dynamic system. The KNN classifier was used to classify uncompleted time series using various distances, such as Euclidean distance, DTW (Dynamic time warping) and Manhattan distance. The simulation studies showed that by using case based reasoning, the most important increase of classification accuracy occurs on the prefixes through thirty to fifty percent of the full length.

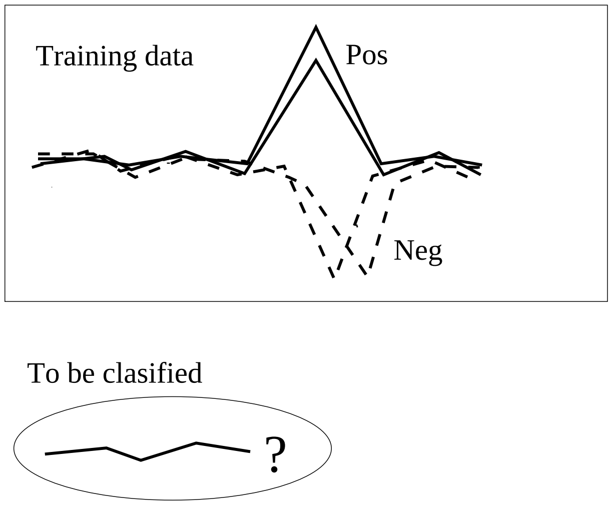


Figure 2.1: A two-class time series classification example.

Although in [7, 13], the importance of early classification on time series was identified and some encouraging results were shown, they only solved early classification as a problem of classifying prefixes of sequences. In Chapter 1, we pointed out the challenge of early classification is how to find out the earliest time for a reliable classification. The methods proposed in [7, 13] did not address the challenge. They only focused on generating a classification result given a prefix of time series but did not answer if the prefix contained enough information for a reliable classification. To classify a prefix, Diez *et al.* [13] simply ignored evaluating the predicates on unavailable suffixes. In another word, when they used the classifier, which is a linear combinations of predicates, to classify a prefix, they set the weights of the predicates on the unavailable intervals as zero. Since the weights represent the importance of intervals for classification, setting some of them as zero may generate unreliable classification results based only on some unimportant intervals. Aníbal *et al.* [9] classified a prefix for a time series by only comparing to its k nearest neighbors of the prefix.

Treating early classification as classifying prefixes oversimplifies the problem. We explain the reason using Example 3.

Example 2 *In Figure 2.1, we have two classes of time series. The two classes share similar patterns at the beginning and start to separate from each other in the later half of time series. For a time series to be classified, when we receive its prefix as shown in Figure 2.1, should* *we generate an early classification result based on the prefix? The classification result is actually meaningless since the prefix does not contain enough information to determine the class label. We should wait for more information to predict the class label.* ∎

Early classification should not only solve how to generate a classification result given a time series prefix but should also learn from the data if a prefix contains sufficient information for a reliable classification and when is the earliest time for a reliable classification. Only a reliable early classification result can be trusted and used by users for further actions.

2.2 Sequence classification

Generally, a sequence is an ordered list of events. An event can be represented as a symbolic value, a numerical real value, a vector of real values or a complex data type. In this thesis, we divide sequence data into the following sub-types.

- Given an alphabet of symbols $\{E_1, E_2, E_3, ..., E_n\}$, a *simple symbolic sequence* is an ordered list of the symbols from the alphabet. For example, a DNA sequence is composed of four animo acid A, C, G, T and a DNA segment, such as $ACCCCCGT$, is a simple symbolic sequence.
- A *complex symbolic sequence* is an ordered list of vectors. Each vector is a subset of the alphabet [37]. For example, for a sequence of items bought by a customer over one year, treating each transaction as a vector, a sequence can be $\langle(milk, bread)(milk, egg) \cdots (potatos, cheese, coke)\rangle$.
- A *simple time series* is a sequence of real values ordered in timestamp ascending order. For example,
$$\langle(t_1, 0.1)(t_2, 0.3) \cdots (t_n, 0.3)\rangle$$
is a simple time series recording the data from time stamp t_1 to t_n.
- A *multivariate time series* is a sequence of numerical vectors. For example,
$$\langle(t_1, \langle 0.1, 0.3, 05\rangle)(t_2, \langle 0.3, 0.9, 0.8\rangle) \cdots (t_n, \langle 0.3, 0.9, 0.4\rangle)\rangle$$
is a multivariate time series.

- In the above, the data types of the events are simple. In some applications, the data type of events can be arbitrarily complicated. For example, in a patient record data set (`http://www.informsdmcontest2009.org/`), each patient is represented by a longitudinal sequence of hospital visits. Each visit is an event and is described by multiple numerical measurements, categorical fields and text descriptions. A *complex event sequence* refers to the general form of sequences.

A sequence may carry a class label. Given L as a set of class labels, the task of *(conventional) sequence classification* is to learn a *sequence classifier* C, which is a function mapping a sequence $s \in S$ to a class label $l \in L$, written as $C : S \rightarrow L$.

In (conventional) sequence classification, each sequence is associated with only one class label and the whole sequence is available to a classifier before the classification. There are also other application scenarios for sequence classification. For example, for a sequence of symptoms of a patient over a long period of time, the health condition of the patient may change. For a streaming sequence, which can be regarded as a virtually unlimited sequence, instead of predicting one class label, it is more desirable to predict a sequence of labels. This problem is considered in [26, 27] as the *strong sequence classification* task.

Example 3 *EEG time series data has been used to classify human brain activities, such as "taking rest", "mental arithmetic", and "mental rotation" [36]. If we receive a streaming EEG time series of a human being who is switching among different brain activities, a strong classifier outputs a series of class labels which is the sequence of different brain activities performed by the human.* ■

In this thesis, we propose early classification on temporal sequences, which is different from (conventional) sequence classification. In early classification, a classifier receives the events in a temporal sequence online and classifies the temporal sequence as early as possible once the classifier is confident about the classification. For early classification, in this thesis, we do not consider strong sequence classification. Integrating early classification with strong sequence classification is left as future works.

There are three major challenges in sequence classification. First, most of the classifiers, such as decision trees and neural networks, can only take input data as a vector of features.

However, there are no explicit features in sequence data. Second, even with various feature selection methods, we can transform a sequence into a set of features, the feature selection is far from trivial. The dimensionality of the feature space for sequence data can be very high and the computation can be costly. Third, besides accurate classification results, in some applications, we may also want to get an interpretable classifier. Building an interpretable sequence classifier is difficult since there are no explicit features.

Most of the existing works focus on (conventional) sequence classification. The methods proposed for the (conventional) sequence classification handled the sequential nature of the data and are valuable for more advanced sequence classification tasks, such as strong sequence classification [27] and early classification. In the following, we give a brief review on existing sequence classification methods. In this chapter, when we mention sequence classification, we mainly refer to (conventional) sequence classification.

We categorize sequence classification methods into three large categories.

- The first category is *feature based classification*, which transforms a sequence into a feature vector and then applies conventional classification methods. Feature selection plays an important role in this kind of methods.
- The second category is *sequence distance based classification*. The distance function which measures the similarity between sequences affects the quality of the classification significantly.
- The third category is *model based classification*, such as using hidden Markov model (HMM) and other statistical models to classify sequences.

In the rest of this section, we will present some representative methods in the three categories. Some methods may ride on multiple categories. For example, we can use SVM by either extracting features (Category 1) or defining a distance measure (Category 2). Sequence classification using SVM will be summarized in Section 2.2.3.

2.2.1 Feature Based Classification

Conventional classification methods, such as decision trees and neural networks, are designed for classifying feature vectors [1]. One way to solve the problem of sequence classification is to transform a sequence into a vector of features through feature selections.

For a symbolic sequence, the simplest way is to treat each element as a feature. For example, a sequence $CACG$ can be transformed as a vector $\langle A, C, C, G \rangle$. However, the sequential nature of sequences cannot be captured by this transformation. To keep the order of the elements in a sequence, a short sequence segment of k consecutive symbols, called a k-gram, is usually selected as a feature. Given a set of k-grams, a sequence can be represented as a vector of the presence and the absence of the k-grams or as a vector of the frequencies of the k-grams. Sometimes, we also allow inexact matchings with gapped k-grams. By using k-grams as features, sequences can be classified by a conventional classification method, such as SVM [38, 39] and decision trees [11]. A summary of k-gram based feature selection methods for sequence classifications can be found in [17].

The number of all the possible k-grams may be huge. Since not all features are equally useful for classification, Chuzhanova *et al.* [11] used Gamma test to select a small informative subset of features from the k-grams. A genetic algorithm was used to find the local optimal subset of features.

In contrast to k-gram based feature selections, Lesh *et al.* [33, 37] proposed a pattern-based feature selection method. The features are short sequence segments which satisfy the following criteria (1) frequent in at least one class (2) distinctive in at least one class and (3) not redundant. Criterion (2) means a feature should be significantly correlated with at least one class. The intuition of Criterion (3) is that if two features are closely correlated to each other, they are redundant for classification. An efficient feature mining algorithm was proposed to mine features according to the criteria. After selecting the features, Winnow [45] and naive bayes classifiers were used. The experimental results in [33] showed that comparing to the method of considering each element as a feature, pattern-based feature selection can improve the accuracy by 10% to 15%.

The challenge of applying pattern-based feature selection on symbolic sequences is how to

[1]We also refer them as attribute vectors

efficiently search for the features satisfying the criteria. Ji *et al.* [25] proposed an algorithm to mine distinctive subsequences with a maximal gap constraint. The algorithm, which uses bit operations and a prefix growth framework, is efficient even with a low frequency threshold.

Time series data is numeric. The feature selection techniques for symbolic sequences cannot be easily applied to time series data without discretization. Discretization may cause information lost. Ye *et al.* [72] proposed a feature selection method which can be applied directly on numeric time series. Time series shapelet, the time series subsequence with the best classification ability, is considered as the feature for time series classification [72]. For a two-class classification task, given a learned distance threshold, a shapelet is a segment of time series which can be used to separate the training data into two parts according to the distance to the shapelet, and maximizes the information gain. The distance threshold and the shapelet were learned from the training data to optimize the information gain. To construct a classifier, the shapelet selection process was integrated with the construction of the decision tree.

Although subsequences are informative features, they can only describe the local properties of a long sequence. Aggarwal *et al.* [3] developed a method to capture both the global and local properties of sequences for the purpose of classification. They modified wavelet decomposition to describe a symbolic sequence on multiple resolutions. With different decomposition coefficients, the wavelet represents the trends in different ranges of intervals, from global to local. Using wavelet decomposition and a rule based classifier, the wavelet decomposition method outperforms the k-nearest neighbor classifier on a web accessing sequence data set and on a genomic sequence data set.

In summary, the existing methods differ from each other on the following aspects.

- Which criteria should be used for selecting features, such are distinctiveness, frequency, and length?
- In which scope does feature selection reflect the sequential nature of a sequence, local or global?
- Should matchings be exact or inexact with gaps?

- Should feature selection be integrated within the process of constructing the classifier or a separate pre-processing step?

2.2.2 Sequence Distance Based Classification

Sequence distance based methods define a distance function to measure the similarity between a pair of sequences. Once such a distance function is obtained, we can use some existing classification methods, such as K nearest neighbor classifier (KNN) and SVM with local alignment kernel [56] (to be discussed in Section 2.2.3), for sequence classification.

KNN is a lazy learning method and does not pre-compute a classification model. Given a labeled sequence data set T, a positive integer k, and a new sequence s to be classified, the KNN classifier finds the k nearest neighbors of s in T, $kNN(s)$, and returns the dominating class label in $kNN(s)$ as the label of s.

The choice of distance measures is critical to the performance of KNN classifiers. In the rest of this section, we focus on summarizing different distance measures proposed for sequence data.

For simple time series classification, Euclidean distance is a widely adopted option [29, 66]. The Euclidean distance usually requires two time series to have the same length. Keogh *et al.* [29] showed when applying 1NN classifier on time series, Euclidean distance is surprisingly competitive in terms of accuracy, compared to other more complex similarity measures.

Euclidean distance is sensitive to distortions in time dimension. Dynamic time warping distance (DTW) [31] was proposed to overcome this problem and does not require two time series to be of the same length. The idea of DTW is to align two time series and get the best distance by aligning. Xi *et al.* [67] showed that on small data sets, DTW can be more accurate than Euclidean distance. However, recent empirical results [16] strongly suggest that on large data sets, the accuracy of DTW converges with Euclidean distance.

For symbolic sequences, such as protein sequences and DNA sequences, alignment based distances are popular adopted [28]. Given a similarity matrix and a gap penalty, the Needleman-Wunsch algorithm [49] computed an optimum global alignment score between two sequences through dynamic programming. In contrast to global alignment algorithms, local alignment algorithms, such as the Smith-Waterman algorithm [60] and BLAST [5], measure the similarity between two sequences by considering the most similar regions but not enforcing the alignments on full length.

2.2.3 Support Vector Machine

SVM has been proved to be an effective method for sequence classification [46, 42, 38, 61, 62, 59, 12]. The basic idea of applying SVM on sequence data is to map a sequence into a feature space and find the maximum-margin hyperplane to separate two classes. Sometimes, we do not need to explicitly conduct feature selection. A kernel function corresponds to a high dimension feature space. Given two sequences, x, y, some kernel functions, $K(x, y)$, can be viewed as the similarity between two sequences [61]. The challenges of applying SVM to sequence classification include how to define feature spaces or kernel functions, and how to speed up the computation of kernel matrixes.

One of the widely used kernels for sequence classification is the *k-spectrum kernel* or *string kernel*, which transforms a sequence into a feature vector. Leslie *et al.* [38] proposed a k-spectrum kernel for protein classification. Given the protein animo acid alphabet of 20 elements $\langle A, R, N, D \cdots \rangle$, the k-spectrum is all possible sequences of length k that are composed by the elements in the alphabet. For example, if $k = 3$, the k-spectrum contains ARN, AND, DCN, and so on. Given the alphabet $\mathcal{A}$, a sequence x is transformed into a feature space by a transformation function

$$\Phi_k(x) = (\phi_a(x))_{a \in \mathcal{A}^k}$$

where $\phi_a(x)$ is the number of times a occurs in x. The kernel function is the dot product of the feature vectors,

$$K(x, y) = \Phi_k(x) \cdot \Phi_k(y)$$

By using a suffix tree algorithm [38], $K(x, y)$ can be computed in $O(kn)$ time. Lodhi *et al.* [46] proposed a string kernel for text classification. Leslie *et al.* [39] extended the k-spectrum kernel to handle mismatching. Sonnenburg *et al.* [62] proposed a fast k-spectrum kernel with mismatching.

One disadvantage of kernel based methods is that it is hard to be interpreted and hard for users to gain knowledge besides a classification result. Sonnenburg *et al.* [61] proposed a method to learn interpretable SVMs using a set of a string kernels. The ideas is to use a weighted linear combination of base kernels. Each base kernel uses a distinctive set of features. The weights represent the importance of the features. After learning the SVM, users can have an insight into the importance of different features.

String kernels or the k-spectrum kernel can be viewed as a feature based method. Saigo *et al.* [56] proposed a *local alignment kernel* for protein sequence classification which can be viewed as a distance based method. Although local alignment distance can effectively describe the similarity between two sequences, it cannot be directly used as a kernel function because it lacks the positive definiteness property [24]. Saigo *et al.* [56] modified the local alignment distance and formed a valid kernel called local alignment kernel, which mimics the behavior of the local alignment. The theoretical connection between the local alignment kernel and the local alignment distance was proved. Given two sequences x and y, the local alignment kernel $K(x, y)$ can be computed by dynamic programming.

Other kernels used for sequence classification include polynomial-like kernels [59], kernels derived from probabilistic model (Fisher's kernel) [59], and diffusion kernels [57].

2.2.4 Model Based Classification

One category of sequence classification methods is based on generative models, which assume sequences in a class are generated by an underlying model M. Given a class of sequences, M models the probability distribution of the sequences in the class. Usually, a model is defined based on some assumptions, and the probability distributions are described by a set of parameters. In the training step, the parameters of M are learned. In the classification step, a new sequence is assigned to the class with the highest likelihood.

The simplest generative model is the Naive Bayes sequence classifier [40]. It makes the assumption that, given a class, the features in the sequences are independent of each other. The conditional probabilities of the features in a class are learned in the training step. Due to its simplicity, Naive Bayes has been widely used from text classification [32] and genomic sequences classification [10].

However, the independence assumption required by Naive Bayes is often violated in practice. Markov Model and Hidden Markov Model can model the dependence among elements in sequences [18].

Yakhnenko *et al.* [71] applied a k-order Markov model to classify protein and text sequence data. In the training process, the model is trained in a discriminative setting instead of the conventional generative setting to increase the classification power of the generative model based methods.

Different from Markov Model, Hidden Markov Model assumes that the system being modeled is a Markov process with unobserved states. Srivastava *et al.* [63] used a profile HMM to classify biological sequences. A profile HMM usually has three types of states, inserting, matching and deleting. Aligned training examples are used to learn the transition probabilities between the states and emission probabilities. The learned HMM represents the profile of the training dataset. For each class, a profile HMM is learned. In the classification step, an unknown sequence is aligned with the profile HMM in each class by dynamic programming. An unknown sequence will be classified into the class which has the highest alignment score.

2.3 Temporal Sequence Classification

In sequences, if the events follow a temporal order, we call them temporal sequences. In this section, we summarize the existing works on temporal sequences in more details.

Time series data is an important type of temporal sequence data. In Time Series Data Library [2], time series data across 22 domains, such as agriculture, chemistry, health, finance,industry, are collected. UCR time series data archive [30] provides a set of time series datasets as a benchmark for evaluating time series classification methods.

For simple time series data, to apply feature based methods, the feature selection is a challenging task since we cannot do feature enumeration on numeric data. Therefore, distance based methods are widely adopted to classify time series [67, 29, 66, 55]. It was shown that comparing to a wide range of classifiers, such as neural networks, SVM and HMM, 1-nearest neighbor classifier is usually superior in classification accuracy [29, 67].

To apply feature based methods on simple time series, usually, before feature selection, time series data needs to be transformed into symbolic sequences through discretization or symbolic transformation [44]. Without discretization, Ye *et al.* [72] proposed a method to find time series shapelets and use a decision tree to classify time series. Comparing to distance based methods, feature based methods may speed up the classification process and be able to generate some interpretable results.

Model based methods are also applied to classify simple time series, such as HMM which is widely used in speech recognition [54]. Richard *et al.* [51] transformed time series data into a lower dimensional reconstructed phase space and then applied mixture gaussian models for classification.

Multivariate time series classification has been used for gesture recognition [27] and motion recognition [41]. The multivariate data is generated by a set of sensors which measure the movements of objects in different locations and directions. For multivariate time series classification, Kadous *et al.* [27] proposed a feature based classifier. A set of user-defined meta-features are constructed and a multivariate time series is transformed into a feature vector. Some universal meta-features included the features to describe the trends of increases and decreases and local max or min values. By using those features, multivariate time series with additional non-temporal attributes can be classified by a decision tree. One multivairate time series can be viewed as a matrix. Li *et al.* [34] proposed a method to transform a multivariate time series into a vector through singular value decomposition and other transformations. SVM is then used to classify the vectors.

For symbolic temporal sequences, Terran *et al.* [35] applied k-nearest neighbor classifiers to classify sequences of Unix shell command data of users for anomaly detection. Since a large amount of symbols are obtained from the command sequences, feature selection was conducted to limit the size the alphabet. A distance measure was designed for measure the similarities between users.

2.4 Relations to the thesis

In this chapter, we summarize the existing methods on sequence classification. We categorize sequence classification methods into three large categories, feature based classification, sequence distance based classification and model based classification. In this thesis, the methods proposed in Chapter 3 and Chapter 5 belong to feature based classification and the methods proposed in Chapter 4 fall into the category of sequence distance based classification.

We divide sequence data into five subtypes. In this thesis, the methods in Chapter 3 focus on classifying simple symbolic sequences and the methods in Chapter 4 and Chapter 5 are designed for simple time series data. For the other three subtypes, complex symbolic sequences, multivariate time series and complex event sequences, we may transform them into simple types or develop new methods to classify them, which are left as future works.

In Chapter 3, we develop feature based classifiers by extracting features from symbolic sequences. The criteria for feature extractions are built upon some previous sequence feature extraction methods [33, 37]. For early classification, we extend the existing methods by taking the earliness of features into consideration.

In Chapter 4, we extend nearest neighbor classifier for early classification. The methods proposed in Chapter 4 are related to nearest neighbor classifiers for time series [55, 67, 29, 66]. We introduce a novel learning step for nearest neighbor classifiers to achieve early classification.

In Chapter 5, we propose a feature selection method for time series early classification. The proposed method extends the ideas of distinguishing patterns for symbolic sequences [25] and Shapelets on time series [72].

Chapter 3

Early Classification On Symbolic Sequences

Symbolic temporal sequences are an important type of temporal sequences with real world applications. For example, Terran *et al.* [35] classified temporal sequences of Unix shell commands of users for anomaly detection. Each command is represented as a symbolic event. Furthermore, time series data or other complex forms of temporal sequences can be transformed into symbolic temporal sequences through discretization and other techniques [27].

In this chapter, we propose a framework for early classification on symbolic temporal sequences. The framework falls into the category of feature based classification. Features with good early classification utilities are extracted. A rule based early classifier and a decision tree based early classifier are proposed. The rest of the Chapter is organized as follows. In Section 3.1, we describe the problem of early classification on symbolic temporal sequences and introduce some notations. In Section 3.2, we develop a rule based early classifier, the sequential classification rule (SCR) method. In Section 3.3, we present a generalized sequential decision tree (GSDT) method for early classification. We report the empirical evaluations in Section 3.4. The chapter is summarized in Section 3.5.

3.1 Problem Description and Preliminaries

Let Ω be a set of symbolic events which is the alphabet of the sequence database in question. $s = a_1 \cdots a_l$ is a *sequence* if $a_i \in \Omega$ $(1 \leq i \leq l)$. The number of events in s is the *length* of the sequence, denoted as $\|s\| = l$. A sequence is a temporal sequence if the events in the sequence follow a temporal order.

Let $\mathcal{L}$ be a set of *class labels*. A *sequence database SDB* is a set of tuples (s, c) such that $s \in \Omega^*$ is a sequence and $c \in \mathcal{L}$ is a class label.

A *sequence classifier* is a function $C : \Omega^* \to \mathcal{L}$. That is, for each sequence $s \in \Omega^*$, C predicts the class label of s as $C(s)$.

There are many possible ways to construct a sequence classifier. We review some existing methods in Chapter 2. In order to make early prediction, in this Chapter, we want to build an *early sequence classifier*. An early sequence classifier reads a sequence from left to right in the temporal order, and makes prediction once it is confident about the class label of the input sequence based on the prefix read so far.

For sequence $s = a_1 \cdots a_l$, sequence $s' = a_1 \cdots a_{l'}$ $(1 \leq l' \leq l)$ is a *prefix* of s. We write $s' = s[1, l']$. If an early sequence classifier C classifies a sequence s on a prefix $s[1, l_0]$, the length l_0 is called the *cost* of the prediction, denoted by $Cost(C, s) = l_0$.

The performance of a classifier is often evaluated using a test database SDB'. The *accuracy* of an early sequence classifier C is

$$Accuracy(C, SDB') = \frac{\|\{C(s) = c \,|\, (s, c) \in SDB'\}\|}{\|SDB'\|}$$

Moreover, the *cost of the prediction* is

$$Cost(C, SDB') = \frac{\sum_{(s,c) \in SDB'} Cost(C, s)}{\|SDB'\|}$$

The cost of prediction describes the earliness of classification.

Ideally, we want to construct an early sequence classifier C such that for a sequence database to be classified, C can reach an expected classification accuracy p_0 and minimizes the expected prediction cost. p_0 is a user specified parameter, which is the expected classification accuracy by the user.

3.2 The Sequential Classification Rule Method

In this section, we develop a sequential classification rule (SCR) method for symbolic sequence early classification. The major idea is to first learn a set of features with good early classification utilities and then extract a set of sequential classification rules for early classification. Each rule hypothetically represents a set of sequences of the same class and sharing the same set of features. In this section, we first introduce some definitions in Section 3.2.1, and then present the feature selection and rule mining process in Section 3.2.2 and Section 3.2.3 respectively.

3.2.1 Sequential Classification Rules

A *feature* is a short sequence $f \in \Omega^*$. A feature $f = b_1 \cdots b_m$ *appears* in a sequence $s = a_1 \cdots a_l$, denoted by $f \sqsubseteq s$, if there exist $1 \leq i_0 \leq l-m+1$ such that $a_{i_0} = b_1$, $a_{i_0+1} = b_2$, $\ldots$, $a_{i_0+m-1} = b_m$. For example, feature $f = bbd$ appears in sequence $s = acbbdadbbdca$ twice. When a feature f appears in a sequence s, we can write $s = s'fs''$ such that $s' = a_1 \cdots a_{i_0-1}$ and $s'' = a_{i_0+m} \cdots a_l$. Generally, a feature may appear multiple times in a sequence. The *minimum prefix* of s where feature f appears is denoted by $minprefix(s, f)$. When $f \not\sqsubseteq s$, $minprefix(s, f) = s$, which means f does not appear in any prefix of s.

A *sequential classification rule* (or simply a *rule*) is in the form of $R : f_1 \rightarrow \cdots \rightarrow f_n \Rightarrow c$ where $f_1, \ldots, f_n$ are features and $c \in \mathcal{L}$ is a class label. For the sake of simplicity, we also write a rule as $R : F \Rightarrow c$ where F is the shorthand of a series of features $f_1 \rightarrow \cdots \rightarrow f_n$. The class label in a rule R is denoted by $\mathcal{L}(R) = c$.

A sequence s is said to *match* a sequential classification rule $R : f_1 \rightarrow \cdots \rightarrow f_n \Rightarrow c$, denoted by $R \sqsubseteq s$, if $s = s'f_1s_1f_2 \cdots s_{n-1}f_ns''$. That is, the features in R appear in s in the same order as in R. The *minimum prefix* of s matching rule R is denoted by $minprefix(s, R)$. Particularly, when $R \not\sqsubseteq s$, $minprefix(s, R) = s$, which means any prefix of s does not match R.

Given a sequence database SDB and a sequential classification rule R, the *support* of R in SDB is defined as

$$sup_{SDB}(R) = \frac{\|\{s | s \in SDB, R \sqsubseteq s\}\|}{\|SDB\|}$$

when $\|SDB\|$ is finite. When $\|SDB\|$ is infinite, $sup_{SDB}(R)$ is defined as the expectation of the percentage of sequences in SDB that match R.

The *confidence* of rule R on SDB is given by

$$conf_{SDB}(R) = \frac{\|\{(s,c)|(s,c) \in SDB, R \sqsubseteq s, c = \mathcal{L}(R)\}\|}{\|\{(s,c)|(s,c) \in SDB, R \sqsubseteq s\}\|}$$

when $\|SDB\|$ is finite. When $\|SDB\|$ is infinite, $conf_{SDB}(R)$ is the expectation of the percentage of sequences in SDB matching R that have the same class label $\mathcal{L}(R)$ as R.

The *cost of classification* of rule R on SDB is given by

$$cost_{SDB}(R) = \frac{\sum_{R \sqsubseteq s, s \in SDB} \|minprefix(s,R)\|}{||\{s|R \sqsubseteq s, s \in SDB\}||}$$

Let $\mathcal{R} = \{R_1, \ldots, R_n,\}$ be a set of sequential classification rules. For early prediction, a sequence tries to match a rule using a prefix as short as possible. Therefore, if a sequence s matches at least one rule in $\mathcal{R}$, the *action rule* is the rule in $\mathcal{R}$ which s has the shortest minimum prefix of matching, that is,

$$actionR(s, \mathcal{R}) = \arg\min_{R_i \in \mathcal{R}} \|minprefix(s, R_i)\|$$

Given a sequence database SDB. If a sequence s cannot match with any rule in $\mathcal{R}$, we can define that the action rule for sequence s is a default rule, $R_{default}$, which matches any sequences. The $\|minprefix(s, R_{default})\| = \|s\|$.

Given a rule set $\mathcal{R}$, the *cost of prediction* can be measured as the average cost per sequence, that is,

$$Cost(\mathcal{R}, SDB) = \frac{\sum_{s \in SDB} \|minprefix(s, action(s, \mathcal{R}))\|}{\|SDB\|}$$

To make the classifier accurate, we can confine that only sequential rules of confidence at least p_0 are considered, where p_0 is a user-specified accuracy expectation. Now, the problem is that how to mine a set of rules $\mathcal{R}$ such that each rule is accurate (i.e., of confidence at least p_0) and the cost $Cost(\mathcal{R}, SDB)$ is as small as possible.

3.2.2 Feature Selection

To form sequential classification rules, we need to extract features from sequences in the training data set. Particularly, we want to extract effective features for early classification.

Utility Measure for Early Prediction

We consider three characteristics of features for early prediction. First, *a feature should be relatively frequent.* A frequent feature in the training set may indicate that it is applicable to many sequences to be classified in the future. On the other hand, an infrequent feature may overfit a small number of training samples. Second, *a feature should be discriminative between classes.* Discriminative features carry the power of classification. Last, *we consider the earliness of features.* We prefer features to appear early in sequences in the training set.

Based on the above consideration, we propose a utility measure of a feature for early prediction. We consider a finite training sequence database SDB and a feature f.

The *entropy* of SDB is given by

$$E(SDB) = -\sum_{c \in \mathcal{L}} p_c \log p_c$$

where $p_c = \frac{\|\{(s,c) \in SDB\}\|}{\|SDB\|}$ is the probability that a sequence is in class c in SDB.

Let $SDB_f = \{(s,c) | (s,c) \in SDB, f \sqsubseteq s\}$ be the subset of sequences in SDB where feature f appears. The difference of entropy in SDB and SDB_f, $E(SDB) - E(SDB_f)$, measures the discriminativeness of feature f. When $E(SDB) - E(SDB_f) < 0$, we do not consider this feature.

To measure the frequency and the earliness of a feature f, we can use a weighted frequency of f. That is, for each sequence s where f appears, we use the minimum prefix of s where f appears to weight the contribution of s to the support of f. Technically, we have

$$wsup_{SDB}(f) = \frac{\sum_{f \sqsubseteq s, s \in SDB} \frac{1}{\|minprefix(s,f)\|}}{\|SDB\|}$$

Then, the *utility measure* of f is defined as

$$U(f) = (E(SDB) - E(SDB_f))^{\omega} wsup_{SDB}(f) \tag{3.1}$$

In the formula, we use a parameter $\omega \geq 1$ to determine the relative importance of information gain versus earliness and popularity. This parameter carries the same spirit of those used in some previous studies such as [50].

Top-k Feature Selection

Many existing rule-based classification methods such as [37, 43, 74] set some thresholds on feature quality measures like support, confidence, and discriminativeness, so that only those high quality (i.e., relatively frequent and discriminative) features are selected for classifier construction.

In our case, we may take a similar approach to set a utility threshold and mine all features passing the threshold from the training database.

However, we argue that such a utility threshold method is ineffective in practice. The utility values of effective features may differ substantially in various data sets. It is very hard for a user to guess the right utility threshold value. On the one hand, a too high utility threshold may lead to too few features which are insufficient to generate an accurate classifier. On the other hand, a too low utility threshold may lead to too many features which are costly to mine.

To overcome the problem, we propose a progressive approach. We first find top-k features in utility, and build a set of rules using the top-k features. If the rules are insufficient in classification, we mine the next k features. The progressive mining procedure continues until the resulting set of sequential classification rules are sufficient.

Now, the problem becomes how to mine top-k features effectively for sequential classification rule construction.

Given a finite alphabet set Ω, all possible features as sequences in Ω^* can be enumerated using a sequence enumeration tree $T(\Omega)$. We use an optional support threshold min_sup to prune features. A feature is not considered if its support is lower than min_sup. By default, the $min\_sup = 0$. We can also limit the maximal length of features [37]. The root of the tree represents the empty feature $\emptyset$. Each symbol $x \in \Omega$ is a child of the root node. Generally, a length-l sequence s is the parent of a length-$(l+1)$ sequence s' if $s' = sx$ where $x \in \Omega$. Figure 3.1 shows a sequence enumeration tree of alphabet $\Omega = \{a, b, c\}$.

To find the top-k features, we build a sequence enumeration tree. In order to avoid searching trough the complete enumeration tree, we use a utility bound to prune the tree.

As the first step, we select a set k features as the seeds. The set is denoted as $Seed$ in the following. We use a heuristic method to search a small portion of the enumeration tree

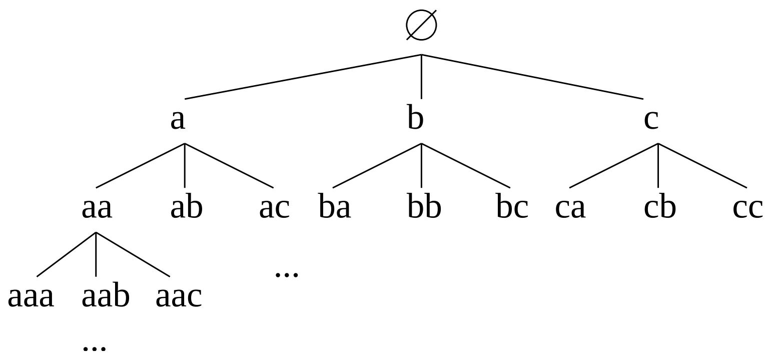


Figure 3.1: A feature enumeration tree.

and get the *Seed* which are the of top-k features in the portion of tree we searched. We first select k' length-1 features, where k' is a small number less then k. That is, we compute the utility values of all length-1 features, and select the best k' features into *Seed*. Then, for each of those length-1 features f in *Seed*, we search their length-2 children. Among all the k'^2 length-2 children, we select the best k' features and insert them into *Seed*. The selection procedure continues iteratively level by level until no longer features can be added into the seed set. As the last step, we choose the best k features in the set *Seed*, and remove the rest.

Once we obtain a set of k seed features, we can use the seeds to prune the search space. Let $U_{lb} = \min_{f \in Seed} U(f)$ be the lower bound of the utility values of the features in the set *Seed*. For a feature f in the sequence enumeration tree, if the utility of the descendants of f can be determined no greater than U_{lb}, then the subtree of f can be pruned.

Then, for a feature f in the sequence enumeration tree, how can we determine whether a descendant of f may have a utility value over U_{lb}?

Theorem 1 (Utility bound) *Let SDB be the training data set. For features f and f' such that f is a prefix of f',*

$$U(f') \leq \frac{E(SDB)^{\omega}}{\|SDB\|} \sum_{s \in SDB, f \sqsubseteq s} \frac{1}{minprefix(s, f) + 1}$$

Proof. In the best case, all sequences in $SDB_{f'}$ belong to the same class. In such a case, $E(SDB_{f'}) = 0$. Thus, the decreasing in entropy is no greater than $E(SDB)^{\omega}$.

Since f is a prefix of f', $\|f'\| \geq \|f\| + 1$. Thus,

$$wsup_S DB(f') \leq \frac{\sum_{s \in SDB, f \sqsubseteq s} \frac{1}{minprefix(s,f)+1}}{\|SDB\|}$$

Both the decreasing in entropy and the weighted support are non-negative. Thus, using Equation 3.1, we have the upper bound in the theorem. ■

If the descendants of f cannot be pruned by Theorem 1, we need to search the subtree of f. Once a feature whose utility value is greater than U_{lb} is found, we insert it into *Seed*, the set of seed features, and remove the feature in *Seed* whose utility value is the lowest. After inserting a better feature into *Seed* and removing a worse one from *Seed*, the lower bound U_{lb} of top-k utility values is increased. The new lower bound U_{lb} is used to pruned the search space.

When there are multiple nodes in the sequence enumeration tree whose utility values are over U_{lb} and whose subtrees need to be searched, we conduct a *best-first* search. That is, we search the subtree of the node of the highest utility value, since heuristically it may have a good chance to provide good features. The search procedure continues until we cannot extend any branches.

Since we search the sequence enumeration tree, which is the complete space of all possible features, and our pruning guarantees no feature which is promising in the top-k list in utility is discarded, we have the following claim.

Theorem 2 (Completeness of top-k features) *The top-k feature selection procedure described in this section selects top-k features in utility values.* ■

When we match a feature against a sequence, we do not consider gaps. Extracting features with gap constraints usually requires parameters to define in what extends the gap is allowed in matching. Furthermore, gap constraints increase the computational cost in constructing features [25]. Without allowing gaps, especially for long features, the supports of features may become low. But for early classification, we prefer early features. Short features usually have a better chance as earlier features than long features. Experimental results of length of features extracted by our methods are presented in Section 3.4. We build sequential classification rules by considering the feature combinations. In another word, we use a sequence of short features to represent long features with gaps.

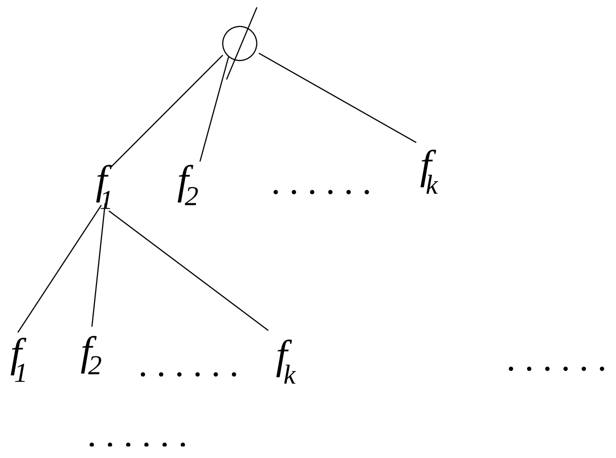


Figure 3.2: A rule enumeration tree.

3.2.3 Mining Sequential Classification Rules

Given a set of features $F = \{f_1, \ldots, f_k\}$, all possible rules using the features in F can be enumerated using a rule enumeration tree similar to a feature enumeration tree in spirit.

Figure 3.2 shows an example of a feature enumeration tree. The root of the tree is the empty set $\{\}$. Each node contains a feature $f_i \in F$. A node R_i represents a sequential rule $f_1 \rightarrow \cdots \rightarrow f_l \Rightarrow c$ ($c \in \mathcal{L}$), in which $f_1 \rightarrow \cdots \rightarrow f_l$ are the features on the path from root to node R_i. Node $R_j : f_1 \rightarrow \cdots \rightarrow f_l \rightarrow f_{l+1}$ is a child of R_i. For a training data set SDB composed by several classes, we use SDB_i to denote all sequences in class i. The class of the rule (right side of the rule) represented by node R_i is the class in which R_i has the largest support, $sup_{SDB_i}(R_i)$.

We want to learn a set of rules where each rule satisfies the minimum support threshold min_sup and the minimum accuracy threshold p_0. The min_sup is an optional parameter which is 0 by default. The accuracy threshold p_0 is the the user expected accuracy for the early classifier. To achieve the expected accuracy, we enforce that each rule satisfies this accuracy.

Furthermore, we want to construct a set of non-redundant sequential classification rules with low prediction cost. In order to avoid searching a large number of rules, we conduct a best-first search on the rule enumeration tree. In building the tree, we consider the rules with smaller cost before rules with larger cost.

A node in the rule enumeration tree can be in one of the four status: *active*, *chosen*, *pruned*, or *processed*.

We build the first level of node from the root using the top-K features we extracted in the feature selection step. Each node represents a single feature rule. All those nodes of single feature rules are set to active. At the beginning, the learned Rule set $\mathcal{R}$ is empty.

For node R, we define its *active support* against the learned rule set $\mathcal{R}$. The active support of a rule R is defined as

$$sup_{act}(R) = \|\{s | s \in SDB, R \sqsubseteq s, \nexists R' \in \mathcal{R} \; st \; minprefix(s, R') \leq minprefix(s, R)\}\|$$

The active support means if a sequence s can be matched by rule R, and the sequence can not be matched by any rule R' already in the rule set $\mathcal{R}$ with a lower cost, s contributes one vote for the active support of R. Otherwise, if a sequence s matching both R and R', and $minprefix(s, R') \leq minprefix(s, R)$, then s should not contribute to the active support of R.

Among the active nodes, we select a rule R of the lowest cost. If its confidence is at least p_0, its support is above *min_sup* and its active support is higher than 0, then the rule is chosen and added to the rule set $\mathcal{R}$. The status of R is set to chosen, and all descendants of R in the rule enumeration tree are set to pruned.

If the support of rule R is not higher than *min_sup*, R and its descendants are set to pruned since longer rules cannot satisfy the *min_sup* threshold.

If the active support of rule R is 0, R and its descendants are set to pruned since a longer rule cannot have a higher active support.

If the confidence of R is less than p_0, R satisfies the *min_sup* and its active support is higher than 0, we consider all children of R by adding one feature at the end of R. After the expansion, node R is set to processed and the children are set as active.

Once a rule R is chosen and added into the rule set $\mathcal{R}$, we need to adjust the active supports of the rules in the rule set $\mathcal{R}$. If a sequence s uses another rule R' as the action rule before R is chosen, and uses R as the new action rule after it is added, then, we should decrease the active support of R' by 1. After the adjustment, if the active support of R' becomes zero, R' will be removed from the rule set.

For each rule, by requiring its active support higher than 0, we remove redundant rules.

In another word, in the learning process, all the selected rules are the action rules for the training data. For early classification, it is important to keep a small number of rules. Since in the classification step, when a new event in a temporal sequence arrives, we need to decide whether there is a rule which matches the prefix we received so far. A large number of rules will increase the classification time. Furthermore, a small number of rules without redundancy can provide a succinct summarization of the early features in the data set.

The above best-first search will terminate because when rules are expanded, the supports monotonically decrease.

After building a set of rules by best-first search, there may still be some sequences in SDB which do not match any rules in $\mathcal{R}$. We need to find new features and new rules to cover them. To do so, we consider the subset $SDB_{\bar{\mathcal{R}}} \subseteq SDB$ which is the set of sequences not matching any rules in $\mathcal{R}$. We select features and mine rules on $SDB_{\bar{\mathcal{R}}}$ using the feature selection procedure and the rule mining procedure as described before. The only difference is that, when computing the utility of features, we still use $(E(SDB) - E(SDB_f))$ as the entropy difference, but use $wsup_{SDB_{\bar{\mathcal{R}}}}(f)$ as the weighted support. The reason is that the feature selected should be discriminative on the whole data set. Otherwise, some biased or over-fitting features may be chosen if we compute the entropy difference based on $SDB_{\bar{\mathcal{R}}}$.

Iteratively, we conduct the feature selection and rule mining until each sequence in the training data set SDB matches at least one rule in $\mathcal{R}$. The rule set $\mathcal{R}$ is used as the classifier.

When a set of rules $\mathcal{R}$ is used in classification, a sequence is matched against all rules in $\mathcal{R}$ simultaneously, until the first rule is matched completely. This earliest matched rule gives the prediction.

3.3 The Generalized Sequential Decision Tree Method

Decision tree [52, 53] is a popularly used classification model on attribute-value data which does not consider any sequential order of attributes. It is well recognized that decision trees have good understandability. Moreover, a decision tree is often easy to construct.

Unfortunately, the classical construction framework cannot be applied straightforwardly to sequence data for early classification. There are not natural attributes in sequences since sequence are not well structured attribute-value data. Thus, the first challenge is how to construct some “attributes” using features in sequences. As shown in Example 1 in Section 1.2, one way to construct an attribute is to use one feature as an attribute and each attribute has two values, feature presence and feature absence. However, we cannot use feature absence to achieve early classification since we need to wait to scan the whole sequence to determine whether the feature appears or not.

In this section, we propose a method to compose an attributes using a set of features. We integrate the early prediction utility in attribute composition. We develop a generalized sequential decision tree (GSDT for shot) method.

3.3.1 The GSDT Framework

As the critical idea of construction a GSDT, we use a set of features F to compose a node in the decision tree such that at least one feature in F likely appears in a sequence to be classified. In classification of a sequence s, once a feature in F is matched with s, s can be moved to the corresponding child of the node for further classification.

Ideally, we can choose a set of features F to compose an attribute in the decision tree such that each sequence to be classified matches one and only one feature in F. The feature composed node mimics the node in a classical decision tree perfectly. Unfortunately, in practice, we may not be able to get a set of features which perfectly partition the training data into several disjoint subsets.

To tackle the problem, in GSDT, we allow a sequence in the training set to contain more than one feature from F, and simultaneously ensure that each sequence in the training set contains at least one feature in F. By covering the training data set well, we hope that when the resulting decision tree is applied for classification of a unseen sequence, the sequence may likely match with at least one feature from F.

Figure 3.3 shows a GSDT example. At the root node, a set of features $\{f_1, f_2, f_3, f_4\}$ are chosen to form an attribute A. As will be explained in Section 3.3.2, we ensure that every sequence in the training set contains at least one feature from A, and allow more than one feature to appear in a sequence.

Attribute A divides the sequences in the training set into 4 subsets: SDB_{f_1}, SDB_{f_2}, SDB_{f_3}, and SDB_{f_4}. Each feature and the corresponding subset form a branch. Different

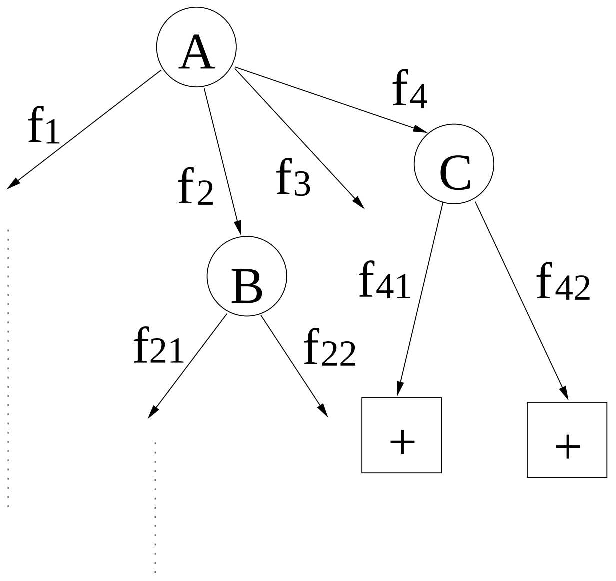


Figure 3.3: A GSDT.

from a classical decision tree, the four subsets are not disjoint. A sequence s in the training set is in both SDB_{f_1} and SDB_{f_2} if both features f_1 and f_2 appear in s. Therefore, GSDT allows redundancy in covering the training data set.

The redundancy in covering the training set in fact improves the robustness of GSDT. Due to the existence of noise, a feature may randomly appear or disappear in a sequence by a low probability. If a sequence is captured by more than one feature and thus more than one branch in GSDT, the opportunity that the sequence is classified correctly by GSDT is improved.

Although we select the features at the root node in a way such that each sequence in the training set contains at least one feature at the root node, it is still possible that an unseen sequence to be classified in the future does not contain any feature at the root node. To handle those sequences, we also store the majority class (i.e., the class of the largest population) in the training data set at the root node. If a sequence does not match any feature at the root node, the GSDT predicts its class using the majority class.

Once the root node is constructed, the subtrees of the branches of the root node can be constructed recursively. For example, in Figure 3.3, the branch of feature f_2 is further partitioned by choosing a set of features $B = \{f_{21}, f_{22}\}$.

If a branch is pure enough, that is, the majority class has a population of at least p_0 where p_0 is the user specified parameter to express accuracy expectation, a leaf node carrying the label of the majority class is added. In Figure 3.3, the branch f_4 is further split by attribute $C = \{f_{31}, f_{32}\}$, and two leaf nodes are obtained.

To avoid overfitting, similar to the classical decision tree construction, we stop growing a branch if the training subset in the branch has less than min_sup sequences, where min_sup is the minimum support threshold.

3.3.2 Attribute Composition

Now, the only remaining issue is how to select a set of features as an attribute.

Consider a training set SDB, we need to find a set of features which have high early prediction utility and cover all sequences in SDB. We also want the set of features as small as possible to avoid overfitting.

Given a set of features which cover all sequences in the training set, finding a minimal set to cover all sequences is an NP-complete problem which is actually the minimum set cover problem [20]

Here, we adopt a greedy approach. To construct the root attribute, let $A = \emptyset$ and $SDB' = SDB$ at the beginning. We consider the top-k features of the highest early prediction utility extracted using the method in Section 3.2.2. If a sequence in SDB' matches a feature in A, then the sequence is removed from SDB'. We iteratively add to A the feature which has the highest utility score in SDB'. The iteration continues until SDB' is empty. If the k features are used up but SDB' is not empty yet, another k features are extracted.

Each branch in the decision tree is actually a sequential classification rule. To construct a non-root node, the same procedure as constructing root node is performed on the subset of SDB which matches the branch to be extended.

In the classical decision tree construction, at each node, an attribute of the best capability to discriminate different classes is chosen. In GSDT, importantly, only the features of high early prediction utilities are considered for attribute composition. As explained in Section 3.2.2, the utility measure defined in Equation 3.1 captures the capability of discriminating different classes and the earliness of prediction simultaneously. Using features of high utility in attribute composition ensures the effectiveness of GSDT in classification accuracy and earliness.

3.4 Empirical Evaluation

In this section, we report an empirical study on our two methods using both symbolic sequence data sets and time series data sets. The time series data sets are transformed into symbolic sequences by discretization. All the experiments were conducted in a PC computer with an AMD 2.2GHz CPU and 1GB main memory. The algorithms were implemented in Java using platform JDK^{TM} 6.

3.4.1 Results on DNA Sequence Data Sets

DNA promoter data sets are suitable for testing our feature extraction techniques for early prediction, because there are explicit motifs on promoters. Also, the sequential nature of DNA sequences can be used to simulate the temporal orders. We use two promoter data sets of prokaryotes and eukaryotes respectively in our experiments.

Results on the E.Coli Promoters Data Set

The E. Coli Promoters data set was obtained from the UCI machine learning repository [6]. The data set contains 53 E. Coli promoters instances and 53 non-promoter instances. Every instance contains 57 sequential nucleotide ("base-pair") positions.

Along with the data set, the results of some previous classification methods [65] on it are also provided. The accuracy of those results are obtained by using "leave-one-out" methodology. For the comparison purpose, we also use "leave-one-out" in our experiments. In methods SCR and GSDT, we set the maximal length of a feature to 20. In each round of feature extraction, we extract the top-30 features with the highest utility scores. The results of in Table 3.1 is obtained when setting the accuracy parameter as $p_0 = 95\%$, minimal support threshold as $s_0 = 5\%$.

In Table 3.1, the results from our methods and previous methods are listed in the error rate ascending order. Our two methods can obtain competitive prediction accuracy using

Method	Error rate	Comments
KBANN	$\frac{4}{106}$	a hybrid machine learning system
SCR	$\frac{7}{106}$	Average prefix length for prediction: $\frac{21}{57}$
BP	$\frac{8}{106}$	standard artificial neural network with back-propagation using one hidden layer
GSDT	$\frac{10}{106}$	Average prefix length for prediction: $\frac{20}{57}$
O'Neill	$\frac{12}{106}$	ad hoc technique from the bioinformatics literature
3-NN	$\frac{13}{106}$	a nearest-neighbor method
ID3	$\frac{19}{106}$	[52]

Table 3.1: The accuracy of several methods on the E. Coli data set. Except for SCR and GSDT, all methods use the entire sequences in classification.

the parameter setting we described. Except for our two methods, the other methods all use every nucleotide position as a feature and thus cannot give early prediction. Our two methods use very short prefixes (up to 36.8% of the whole sequence on average) to give early prediction.

Please note that, when counting the error rate of SCR and GSDT in Table 3.1, if a testing sequence cannot be classified using a sequential classification rule or a path in the GSDT (i.e., the sequence does not match any feature in a node of a path), we treat the case as an error. In other words, we test the exact effectiveness of the sequential classification rules generated by SCR and the decision tree generated by GSDT. Thus, the error rate reported in Table 3.1 is the upper bound. In practice, the two methods can give make prediction on an unmatched sequence using the majority class.

The running time of GSDT and SCR are 294 and 340 milliseconds, respectively. SCR is slightly more accurate than GSDT but uses a slightly longer prefix on average. For the GSDT methods, we extracted 38 features and for the SCR method, we extracted 37 features. We list the rules generate by GSDT and SCR methods in Table 3.2. For GSDT, each rule is a branch in GSDT tree.

By comparing the rules of SCR and GSDT, we can see that 32 rules are shared by the two methods and GSDT generates more rules composed by feature combinations instead of single features than SCR.

Some rules generated by SCR and GSDT are interesting. For example, rules *tataa* $\Rightarrow$promoter, and *ataat* $\Rightarrow$promoter generated by both SCR and GSDT are highly meaningful in biology, since *tataat* is a highly conserved region on E. Coli promoters [23].

In section 3.2.2, we point out that for early classification, we prefer early features which are usually short features. From Table 3.2, we can see that the longest feature extracted is of length 7, which is $\frac{7}{57} = 12.28\%$ of the full length. Most rules are composed by single features. For some rules, several features composed a rule, such as $ctcaa \rightarrow ga \rightarrow ac \Rightarrow -$. As we discussed in section 3.2.2, we choose to extract and match features without allowing gaps for the sake of efficiency and avoiding parameters. We use combinations of a sequence of short features to represent long features with gaps.

Results on the Drosophila Promoters Data Set

The Berkeley Drosophila Genome Project provides a large number of drosophila promoter sequences and non-promoter coding sequences (CDS) [1]. We use a set of promoter sequences of 327 instances and a set of CDS of 527 instances provided in year 2002 by the project to test our two methods.

The length of each sequence is 300 nucleotide base pairs. Compared to the E. Coli data set, this data set has longer sequences and more instances. Drosophila (fruit fly) is a more advanced species than E. Coli (bacteria), which is expected to have more complex mechanics on promoters. Those differences make the feature extraction in this data set more challenging than that in the E. Coli data set.

10-fold cross validation is used to test the two methods. When setting the accuracy parameter as 95% and the minimal support threshold as $s_0 = 10\%$, the accuracy and the runtime of SCR and GSDT obtained are shown in Table 3.3.

SCR and GSDT reach similar accuracy, and both use a short prefix, around 28% of the whole sequence, in prediction. GSDT requires a longer running time than SCR. This is

Rule	SCR	GSDT
1	$aaaa \Rightarrow +$	$aaaa \Rightarrow +$
2	$aatt \Rightarrow +$	$aatt \Rightarrow +$
3	$caaa \Rightarrow +$	$caaa \Rightarrow +$
4	$ctacg \Rightarrow -$	$ctacg \Rightarrow -$
5	$ataat \Rightarrow +$	$ataat \Rightarrow +$
6	$accga \Rightarrow -$	$accga \Rightarrow -$
7	$actac \Rightarrow -$	$actac \Rightarrow -$
8	$tattt \Rightarrow +$	$tattt \Rightarrow +$
9	$tttta \Rightarrow +$	$tttta \Rightarrow +$
10	$tataa \Rightarrow +$	$tataa \Rightarrow +$
11	$gaccg \Rightarrow -$	$gaccg \Rightarrow -$
12	$agaaa \Rightarrow +$	$agaaa \Rightarrow +$
13	$gaacg \Rightarrow -$	$gaacg \Rightarrow -$
14	$gatc \Rightarrow +$	$gatc \Rightarrow +$
15	$tcgca \Rightarrow +$	$tcgca \Rightarrow +$
16	$gagag \Rightarrow -$	$gagag \Rightarrow -$
17	$ttcgt \Rightarrow -$	$ttcgt \Rightarrow -$
18	$tcaat \Rightarrow -$	$tcaat \Rightarrow -$
19	$ttaca \Rightarrow +$	$ttaca \Rightarrow +$
20	$gaggg \Rightarrow -$	$gaggg \Rightarrow -$
21	$aaact \Rightarrow +$	$aaact \Rightarrow +$
22	$cgctttg \Rightarrow -$	$cgctttg \Rightarrow -$
23	$taaacg \Rightarrow -$	$taaacg \Rightarrow -$
24	$gtgaac \Rightarrow -$	$gtgaac \Rightarrow -$
25	$ctttctt \Rightarrow -$	$ctttctt \Rightarrow -$
26	$caatgg \Rightarrow -$	$caatgg \Rightarrow -$
27	$cttta \Rightarrow +$	$cttta \Rightarrow +$
28	$acttgc \Rightarrow -$	$acttgc \Rightarrow -$
29	$tccggt \Rightarrow -$	$tccggt \Rightarrow -$
30	$ccca \Rightarrow +$	$ccca \Rightarrow +$
31	$ctcag \Rightarrow -$	$ctcag \Rightarrow -$
32	$attttt \Rightarrow +$	$attttt \Rightarrow +$
33	$aatt \Rightarrow +$	$cgtct \rightarrow tga \Rightarrow -$
34	$tttat \Rightarrow +$	$ctcaa \rightarrow aag \Rightarrow -$
35	$tgagg \Rightarrow +$	$cgtct \rightarrow gc \Rightarrow -$
36	$gagac \Rightarrow -$	$ctcaa \rightarrow ga \rightarrow ac \Rightarrow -$
37	$acgg \rightarrow agcct \Rightarrow -$	$ctcaa \rightarrow ctc \Rightarrow -$
38		$aacatt \Rightarrow -$

Table 3.2: List of rules learned by SRC and GSDT methods.

Method	Accuracy	Avg. prefix len	Runtime
SCR	87%	83.56 /300	142 seconds
GSDT	86%	85.64 /300	453 seconds

Table 3.3: Results on the Drosophila Promoters data set.

because GSDT performs more rounds of top-k extraction. When GSDT constructs a node in the decision tree, top-k feature selection is performed based on a subset of training data.

The Effects on Changing p_0

The user expected accuracy threshold, p_0, is an important parameter. Using the drosophila promoters data set as an example, we test the effects on changing parameter p_0 in terms of the accuracy and earliness in prediction. The results are shown in Figure 3.4 (for SCR) and Figure 3.5 (for GSDT).

In Figure 3.4 and Figure 3.5, we report the accuracies, the average length of prefix in percentage, the unmatch rates (i.e., the percentage of the test sequences which do not match any rule in SCR or any path in GSDT, and are assigned to the majority class) given different p_0. Figure 3.4 also shows the rule error rate which is the percentage of test sequences incorrectly classified by sequential classification rules without including the default rule. Correspondingly, Figure 3.5 shows the tree error rate which is the percentage of the test sequences incorrectly classified by the decision tree without including the default rule. The figure is obtained by using the setting described except for changing different p_0 value.

When p_0 increases, the rule error rate of SCR and the tree error rate of GSDT decrease. The reason is that the accuracy of each rule in SCR and each path in GSDT increases. However, the unmatch rate in both method increases, since the number of qualified rules and that of tree paths decrease. The overall effect is that the accuracy of SCR and GSDT is high when choosing a modest p_0. The accuracy is not good when p_0 is either too low or too high. When $p_0 = 0.9$, the SCR reaches the best accuracy which is quite close to the result when $p_0 = 0.95$. When $p_0 = 0.95$, the GSDT method reaches the best accuracy.

When p_0 increases, the average prediction length monotonically increases. We can see that there is an interesting trade-off between the expectations of earliness and accuracy. If the users expect a lower accuracy, they may gain in the earliness.

In those two methods, the support of the a rule is not required to be high. Usually, s_0 is set from 5% to 10%.

For some cases, it may not be easy for users to provide the expected accuracy, p_0. The benefit of enabling users to specify the thresholds of expected accuracy is to let users control the tradeoff between the accuracy and the earliness. The alternative way is to learn p_0 through cross-validation using the training data to optimize classification quality on the training data. The classification quality can be measured by a score function which considers the earliness, recall and precision of classification. The score function carries the same spirit as Equation 3.1. Given the classification score function, we can get a average classification score for a given p_0 through cross validation. For example, we can get the classification score using 10-folder cross validation when we set $p_0 = 85\%$. By setting p_0 to a range of different values, such as varying from $p_0 = 75\%$ to $p_0 = 90\%$ with a step of 5%, we can choose a p_0 with the highest classification score as the default parameter setting for unknown classification task.

The Effects on Changing ω

On the E.Coli dataset, we tested the effect of parameter ω in Equation 5.9 on the accuracy and the early prediction of SCR and GSDT using the setting described before except for varying ω. Figure 3.6 shows the accuracy of the two methods with respect to ω. When $\omega = 3$, both methods reach the highest accuracy. When $\omega > 3$, the accuracy is insensitive to ω. A similar trend is observed in Figure 3.7, which shows the average length of prefix with respect to ω. SCR and GSDT use the shortest prefix length on average in early prediction when ω is 2 and 3, respectively. When $\omega > 3$, the average length of prefix used in prediction is insensitive to ω. On the drosophila promoters data set, similar experiments are conducted, and the results are show in Figure 3.8 and Figure 3.9. We observe that, using a fixed setting of other parameters and only changing ω, the best prediction accuracy is reached for SCR when $\omega = 8$, and for GSDT when $\omega = 10$. Comparing with the optimal value of E.Coli dataset, it suggests that when the dataset is larger,ω is better to be set to a larger value.

Method	Nor.	Cyc.	Inc.	Dec.	Upward	Downward	Avg. prefix len.
SCR	0.96	0.96	0.80	0.76	0.36	0.46	33/60
GSDT	0.98	0.88	0.92	0.96	0.42	0.36	27/60
1NN	1	1	1	0.98	0.94	0.9	60/60
1NN (on prefix)	0.48	1	1	1	0.54	0.54	30/60
1NN (on prefix)	0.98	1	0.98	0.98	0.82	0.74	40/60
1NN (on prefix)	1	1	1	0.98	0.9	0.84	50/60

Table 3.4: Results on the Synthetic Control data set.

Method	Nor.	Cyc.	Inc.	Dec.	Avg. prefix len.
SCR	0.96	0.90	0.98	1.00	13/60
GSCT	0.96	0.92	0.98	1.00	15/60
1NN	1	1	1	1	60/60
1NN (on prefix)	0.84	1	0.92	0.96	20/60
1NN (on prefix)	1	1	1	1	30/60

Table 3.5: Results on the Synthetic Control data set without upward/downward shift sequences.

3.4.2 Results on Time Series Data Set

Time series is an important type of temporal sequence data. To apply our methods on time series, discretization needs to be performed. In this section, we report the results of SCR and GSDT on two time series data sets in details. Results of SCR methods on more time series data sets can be found in Table 4.3 in Chapter 4.

Time series data is continuous. Since our methods are designed for discrete sequences, we preprocess the time series as follows. We apply k-means discretization on the training data set to get the discretization thresholds, and transform the training data set into discrete sequences. The test data set is discretized using the thresholds learned from the training data set. So, in classification, the discretization is performed online. The results in the following session is obtained when we set $k = 3$ for the discretization.

Results on the Synthetic Control Data set

The synthetic control data set from the UCI machine learning repository [6] contains 600 examples of control charts synthetically generated by the process. There are six different classes of control charts: normal, cyclic, increasing trend, decreasing trend, upward shift, and downward shift. Every instance contains 60 time points.

We randomly split the data set into two subsets of the same size, i.e., 300 instances as the training examples, and the other 300 instances as the testing examples. We set $p_0 = 0.95$, $s_0 = 10\%$, $k = 30$, and the maximal length of feature as 20 to obtain the results in Table 3.4. In [29], it has been shown that for time series classification, one-nearest-neighbor (1NN) with Euclidean distance is very difficult to beat. In Table 3.4, the results of 1 NN classification with Euclidean distance using whole sequence and using prefix with different lengthes are shown for comparison.

From the results of 1NN classification, we can see that there is a big accuracy drop from using prefix of length 40 to using prefix of length 30. It indicates that the prefix till length 40 contains the important features for classification. In our GSDT and SCR, the average length used in prediction is 27 and 33. It demonstrates that our feature selection procedure can capture the key features within suitable length. GSDT and SCR methods perform better on sequences in classes normal, cyclic, increasing trend and decreasing trend than in class upward shift and downward shift. The reason is that upward shift and downward shift sequences are very similar to increasing and decreasing sequences, respectively, especially after discretization. To correctly classify the upward and downward shift sequences, a classifier has to capture the shift regions accurately. However, the shift regions are similar to some noise regions in other classes, which make the early prediction inaccurate.

If we remove the sequences in the classes of upward shift and downward shift, the performance of the two methods, as shown in Table 3.5, improve substantially. The accuracy is improved further and the average length of prefix is shortened remarkably. Our two method can not beat 1NN when the prefix is longer than 30. But when the prefix is reduced to 20, SCR and GSDT outperform $1NN$ by using average prefix of 13 and 15 respectively.

The features at the root node of a GSDT have good utility in early prediction. In the complete synthetic control data set (i.e., containing sequences of all 6 classes), on average, a sequence matches one of the features at the root node with a prefix of length 14.54, which is substantially shorter than the length of the whole sequence (60). Moreover, when the sequences in classes upward/downward shift are excluded, the average length of prefix matching one of the features in the root node is further reduced to 11.885.

Results on Physiology ECG Data Set

PhysioBank(http://physionet.org/physiobank/) provides a large amount of time series physiology data. A set of ECG data from physioBank is normalized and used in [66]. We downloaded the data set from http://www.cs.ucr.edu/∼wli/selfTraining/. The data set records the ECG data of abnormal people and normal people. Each instance in the normalized data set contains 85 time points. As same as in [66], we use 810 instances as the training examples and $1,216$ instances as the test examples.

In Figure 3.10, the prediction accuracy by using various length of prefix of 1NN is shown by the curve. From the result, we can see this dataset has highly distinguishing feature around the very beginning of the sequence. When $p_0 = 99\%$, $k = 30$ and $s_0 = 10\%$, SCR reaches a prediction accuracy as 99% using average prediction prefix of length 17.3 out of 85, which is the asterisk point in the figure. Under the same setting, GSDT reaches an accuracy of 98% by using an average prefix of length 11.6, which is the diamond point in the figure. Both methods can make prediction with competitive accuracy compared to 1NN and using a very short prefix on average. SCR takes 6.799 *sec* for the training and predication; while GSDT takes longer time of 491.147 *sec*.

3.5 Summary

In this Chapter, we propose methods, the sequential classification rule (SCR) method and the generalized sequential decision tree (GSDT) method, for early classification on temporal sequences. The utility in early prediction is used in selecting features and building classifiers to achieve early classification.

We report an empirical study on DNA sequence data sets and time series data sets. The results clearly show that SCR and GSDT can obtain competitive prediction accuracy using an often remarkably short prefix on average. Early prediction is feasible and effective. SCR tends to get a better accuracy, and GSDT often achieves earlier prediction. SCR is often faster than GSDT.

Although we made some good initial progress in early prediction, the problem is still far from being solved. For example, more accurate methods should be developed and methods on continuous time series are needed.

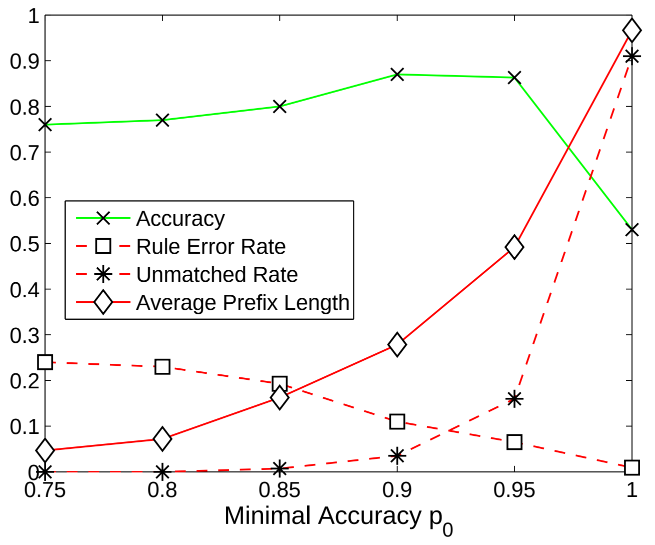


Figure 3.4: Performance of SCR with respect to p_0 on the Drosophila data set.

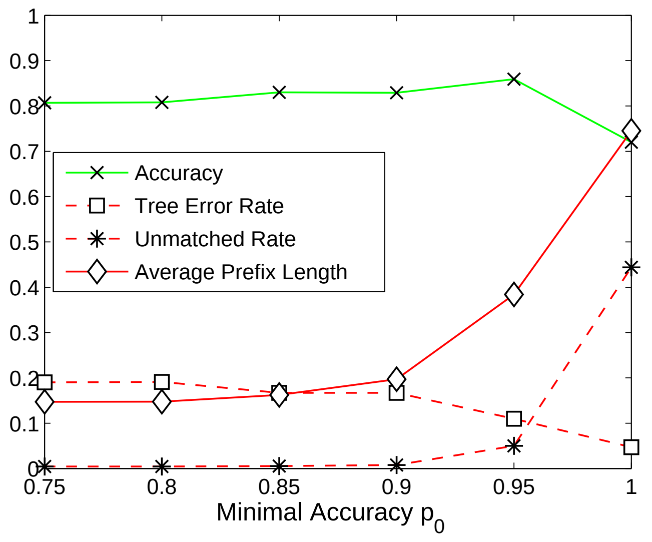


Figure 3.5: Performance of GSDT with respect to p_0 on the Drosophila data set.

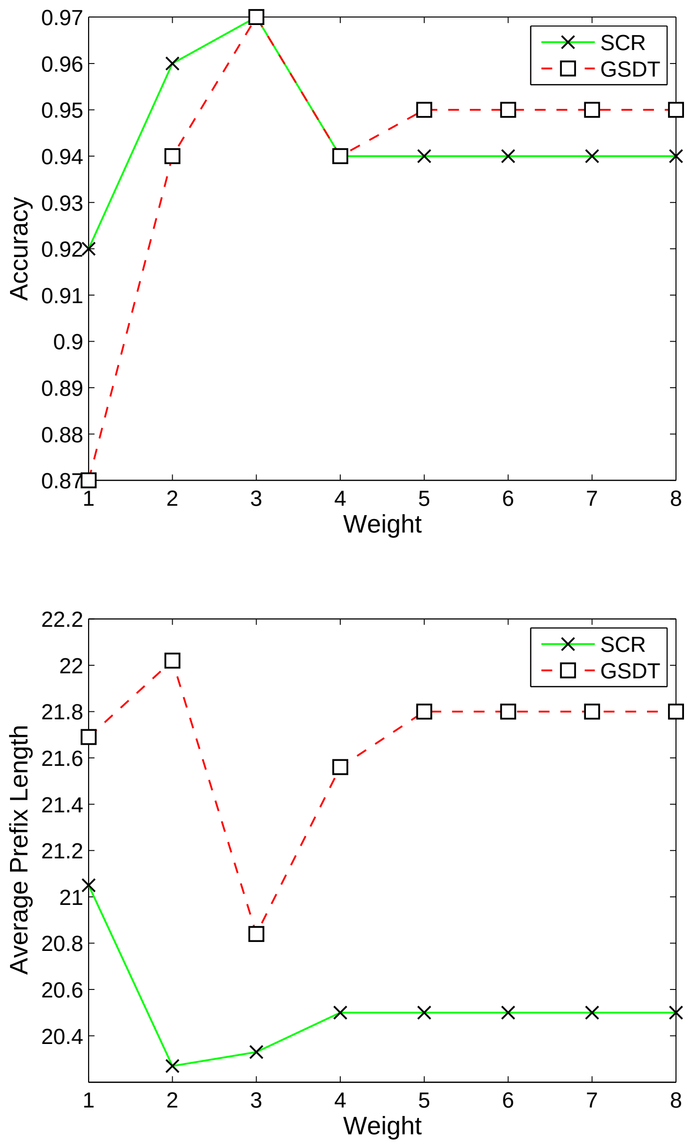


Figure 3.7: Average length of prefix with respect to weight.

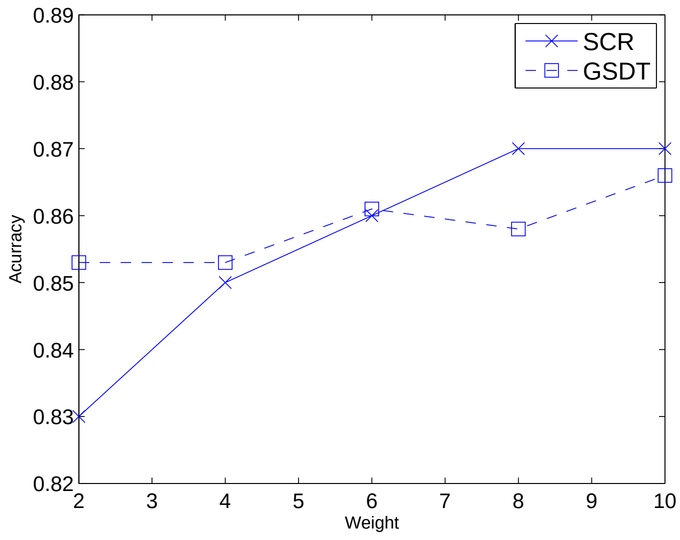


Figure 3.8: Accuracy with respect to weight on Drosophila.

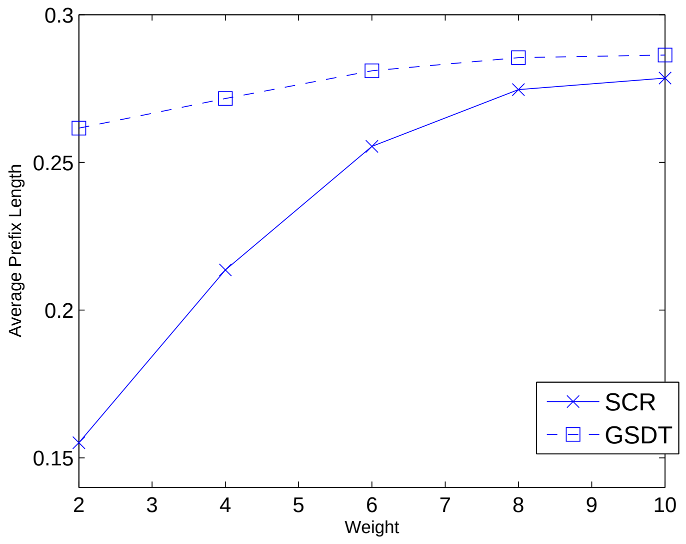


Figure 3.9: Average length of prefix with respect to weight on Drosophila.

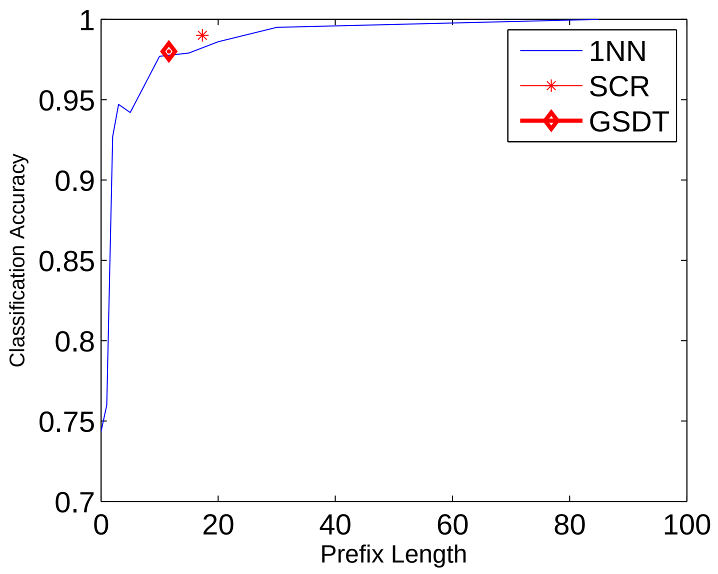


Figure 3.10: Results comparison on ECG Data.

Chapter 4

Instance Based Early Classifiers on Time Series

In Chapter 3, we develop an early classification framework to solve the problem of early classification on symbolic sequences. As demonstrated in Chapter 3, on some data sets, including time series and symbolic sequences, the methods are able to achieve competitive accuracies by using only less than half of the length of the full sequences. However, it is also shown that on some numeric time series data, such as the synthetic control data set, the symbolic methods do not work well. To apply the symbolic methods on time series, discretization need to conducted properly. Furthermore, in the classification step, online discretization is needed. However, appropriate discretization often heavily relies on good background knowledge. Moreover, affected by the discretization granularity, the discretization-based methods may lose important information in time series data. Thus, early prediction on time series data, a type of data prevalent in time-sensitive applications, remains open at large.

In this Chapter, we tackle the problem of early prediction on time series data and develop a native method. Specially, we adopt the 1-nearest neighbor (1NN) approach, which has been well recognized as an effective classification method for time series data. We introduce a novel concept of MPL (Minimum Prediction Length) and develop ECTS (Early Classification on Time Series), an effective 1-nearest neighbor classification method which makes prediction early and at the same time retains an accuracy comparable to that of a 1NN classifier using the full-length time series. Our empirical study using benchmark time series data sets shows that ECTS works well where 1NN classification is effective.

The rest of the Chapter is organized as follows. In Section 4.1, we formulate the problem. We present a basic 1NN early classifier in Section 4.2, and develop the ECTS method in Section 4.3. We evaluate ECTS empirically in Section 4.4. Section 4.5 concludes the paper.

4.1 Problem Definition

A *time series* s is a sequence of pairs $(timestamp, value)$. The data values are ordered in timestamp ascending order. We assume that all timestamps take positive integer values. We denote by $s[i]$ the value of time series s at timestamp i.

To keep our discussion simple, in this paper, we assume that all time series in question are of length L, i.e., each time series s has a value $s[i]$ at timestamp $1 \leq i \leq L$. L is called the *full length* of the time series.

For a time series s of length L, $s[i, j] = s[i]s[i + 1] \cdots s[j]$ $(1 \leq i < j \leq L)$ is the *subsequence* at timestamp interval $[i, j]$. Subsequence $s[1, l]$ $(l \leq L)$ is the *length-l prefix* of s.

For two time series s and s', $dist(s, s')$ denotes the *distance* between them. In this paper, we use the Euclidean distance

$$dist(s, s') = \sqrt{\sum_{i=1}^{L} (s[i] - s'[i])^2},$$

which is a simple yet effective and popularly adopted choice.

The set of all possible time series of length L is $\mathcal{R}^L$ and is called the *full-length space*, where $\mathcal{R}$ is the set of real numbers. The *prefix space of length-l*, denoted by $\mathcal{R}^l$, is the set of length-l prefixes of all possible time series.

In time series classification, a *training set* T contains a set of time series and a set of *class labels* $\mathcal{C}$ such that each time series $t \in T$ carries a class label $t.c \in \mathcal{C}$. The *time series classification problem* is to learn from T a classifier $C : \mathcal{R}^L \rightarrow \mathcal{C}$ such that for any time series s, C predicts the class label of s by $C(s)$. The performance of a classifier is often evaluated using a *test set* T', which is a set of time series such that each time series

$t' \in T'$ also carries a class label $t'.c \in \mathcal{C}$. The *accuracy* of a classifier C on the test set T' is $Accuracy(C, T') = \frac{|\{C(t')=t'.c | t' \in T'\}|}{|T'|}$. Often, we want the classifier C as accurate as possible.

For a time series s, an early classifier C can identify an integer l_0 and make classification based on $s[1, l_0]$. An early classifier is *serial* if $C(s[1, l_0]) = C(s[1, l_0 + i])$ for any $i > 0$. In other words, C can classify s based on only the prefix $s[1, l_0]$, and the classification remains the same by using any longer prefixes. An early classifier is preferred to be serial so that the early classification is reliable and consistent. The minimum length l_0 of the prefix based on which C makes the prediction is called the *cost* of the prediction, denoted by $Cost(C, s) = l_0$. Trivially, for any finite time series s, $Cost(C, s) \leq |s|$. The *cost* of the prediction on a test set T' is $Cost(C, T') = \frac{1}{|T'|} \sum_{t' \in T'} Cost(C, t')$.

Among many methods that can be used in time series classification, the 1-nearest neighbor (1NN) classifier has been found often accurate in practice [29, 66]. The 1NN classification method is parameter-free and does not require feature selection and discretization. Theoretically, Cover and Hart [64] showed that the error rate of the 1NN classifier is at most twice that of the optimal Bayes probability when an infinite sample set is used.

Due to the effectiveness and the simplicity of the 1NN classifier on time series data, in this Chapter, we focus on extending the 1NN classifier for early classification on time series data. We use the 1NN classifier on full length time series as the baseline for comparison. Ideally, we want to build a classifier which is as accurate as the baseline method and minimizes the expected prediction cost.

4.2 1NN Early Classification

1NN classifier is a lazy learning method and does not require any training. To extend 1NN classifier to achieve early classification, we need to add a training process to learn more information from the training data. In the training stage, we want to find out how early a time series can be used as the nearest neighbor to make accurate prediction.

Example 4 *Consider a training set of time series T in Table 4.1, where each time series, written as a sequence of values in timestamp ascending order, has a length $L = 3$.*

To classify a time series $s_1 = (1, 2, 1)$, using full length time series, the 1NN classifier

t-id	*time series*	*class*
t_1	$(1,1,1)$	c_1
t_2	$(2,1,2)$	c_1
t_3	$(5,2,1)$	c_1
t_4	$(6,1,2)$	c_1
t_5	$(5,8,7)$	c_2
t_6	$(5,9,9)$	c_2
t_7	$(6,8,9)$	c_2

Table 4.1: A training set T as the running example.

finds t_1 as the 1NN of s_1 in space $\mathcal{R}^3$, and outputs c_1, the class label of t_1.

The prediction on s_1 can be made much earlier, since t_1 is also the 1NN of prefixes $s_1[1] = (1)$ and $s_1[1,2] = (1,2)$. In other words, using the 1NN of the prefixes, we may make early prediction. ■

For a time series s and a training data set T, let

$$NN^l(s) = \arg\min_{t \in T}\{dist(s[1,l], t[1,l])\}$$

be the set of the nearest neighbors of s in T in prefix space $\mathcal{R}^l$. In Example 4, $NN^1(s_1) = \{t_1\}$ and $NN^2(s_1) = \{t_1\}$.

In the full length space $\mathcal{R}^L$, using the 1NN classifier, a time series s is classified by the dominating label in $NN^L(s)$. Consider prefix space $\mathcal{R}^{L-1}$. Let $T[1, L-1] = \{t[1, L-1] | t \in T\}$ be the set of length-$(L-1)$ prefixes of all time series in the training set T. If the time series in $NN^L(s)$ are still the nearest neighbors of $s[1, L-1]$ in $T[1, L-1]$, i.e., $NN^L(s) = NN^{L-1}(s)$, then we can use $NN^{L-1}(s)$ to make prediction on the class label of s at timestamp $(L-1)$ without compromise in accuracy. This immediately leads to early prediction, as observed in Example 4. The question is when the data points of s arriving in time ascending order, how we can know starting from which prefix length l, the nearest neighbors of s will remain the same.

Generally, the set of length-l prefixes ($1 \leq l \leq L$) of the time series in T is $T[1,l] = \{t[1,l] | t \in T\}$. For $t \in T$, the *set of reverse nearest neighbors in prefix space $\mathcal{R}^l$* of $t(1,l)$ is $RNN^l(t) = \{t' \in T \mid t \in NN^l(t')\}$. That is, $RNN^l(t)$ is the set of time series in T that treat t as its nearest neighbor. In Example 4, since $RNN^1(t_1) = RNN^2(t_1) = RNN^3(t_1) = \{t_2\}$, $RNN^1(t_2) = RNN^2(t_2) = RNN^3(t_2) = \{t_1\}$.

Suppose training set T is a sufficiently large and uniform sample of the time series to be classified. For a time series $t \in T$, if there exists a timestamp $l < L$ such that $RNN^l(t) = RNN^{l+1}(t) = \cdots = RNN^L(t)$, then we can use t to make prediction early at timestamp l without loss in expected accuracy, since every time series s which uses t in $\mathcal{R}^L$ to predict the class label also likely has t as the 1NN in prefix spaces $\mathcal{R}^l$, $\mathcal{R}^{l+1}$, …, $\mathcal{R}^{L-1}$. In Example 4, t_1 can be used to make prediction at timestamp 1 since $RNN^1(t_1) = RNN^2(t_1) = RNN^3(t_1)$.

Definition 1 (Minimum prediction length) *In a training data set T with full length L, for a time series $t \in T$, the* minimum prediction length *(*MPL *for short) of t, $MPL(t) = k$ if for any l ($k \leq l \leq L$), (1) $RNN^l(t) = RNN^L(t) \neq \emptyset$ and (2) $RNN^{k-1}(t) \neq RNN^L(t)$. Specifically, if $RNN^L(t) = \emptyset$, $MPL(t) = L$. We denote by $MPP(t) = t[1, MPL(t)]$ the* minimum prediction prefix *of t.* ■

Example 5 *In Example 4, $MPL(t_1) = 1$ and $MPP(t_1) = (1)$. Moreover, $MPL(t_2) = 1$, $MPL(t_3) = MPL(t_4) = 2$. $MPL(t_5) = MPL(t_6) = MPL(t_7) = 3$.* ■

Given a training set T, a simple *1NN early classification method* works as follows.

Training Phase For each time series $t \in T$, we calculate the minimum prediction length $MPL(t)$.

Classification Phase For a time series s to be classified, the values of s arrive in timestamp ascending order. At timestamp i, we return the dominating class label of time series in $NN^i(s)$ that have a MPL at most i. If no such a time series is found, we cannot make reliable prediction at the current timestamp, and have to wait for more values of s.

Example 6 *In Example 4, it is easy to verify that $s_1 = (1, 2, 1)$ is classified as c_1 using t_1 at timestamp 1. Consider $s_2 = (6, 2, 3)$. $NN^1(s_2) = \{t_4, t_7\}$, but the MPLs of t_4 and t_7 are greater than 1. Thus, s_2 cannot be classified at timestamp 1. $NN^2(s_2) = \{t_3, t_4\}$. The MPLs of t_3 and t_4 are 2. Thus, s_2 can be assigned label c_1 at timestamp 2.* ■

The intuition behind the simple RNN method can be understood from a classification model compression angle. A time series t in the training set T can be compressed as its minimum prediction prefix $MPP(t)$ since based on the information in T, any longer prefix of t does not give a different prediction on a time series that uses t as the nearest neighbor. Thus, those longer prefixes of t are redundant in 1NN classification and can be removed to achieve early prediction. We assume that the training set is a sufficiently large and uniform sample of the data to be classified. Then, for new time series to be classified, the early 1NN classifier can classify new time series as accurate as full length 1NN.

The 1NN early classification method has two drawbacks. First, to make early prediction using a time series t in a training set T, the RNNs of t must be stable after timestamp $MPL(t)$. This requirement is often too restrictive and limits the capability of finding shorter prediction length. In Example 4, the RNNs of t_5, t_6, t_7 are not stable, and the MPLs of them are all 3. We cannot use those time series to classify $s_3 = (5, 8, 8)$ at timestamp 2.

Second, the 1NN early classification method may overfit a training set. The MPP of a time series is obtained by only considering the stability of its RNN which often consist of a small number of time series. The learned $MPL(t)$ may not be long and robust enough to make accurate classification if the training set is not big enough or is not a uniform sample of the time series to be classified.

4.3 The ECTS Method

To overcome the drawbacks in the early 1NN classification method, we develop the ECTS method which extends the early 1NN classification method by finding the MPL for a cluster of similar time series instead of a single time series. In this section, we first develop the idea of the ECTS classifier. Then, we present the algorithm of ECTS method in detail. At last, we discuss a variation of computing MPLs in the framework of ECTS.

4.3.1 ECTS

When we cluster the training data in each prefix space, $\mathcal{R}^l$, where $l = 1, 2, \cdots, L$, we can observe the formations of clusters along the time. When l is small, the clusters in $\mathcal{R}^l$ may be very different from the ones in $\mathcal{R}^L$. When l increases and approaches L, some clusters in $\mathcal{R}^l$ may become stable and similar to some in $\mathcal{R}^L$. Those clusters can be used for early prediction.

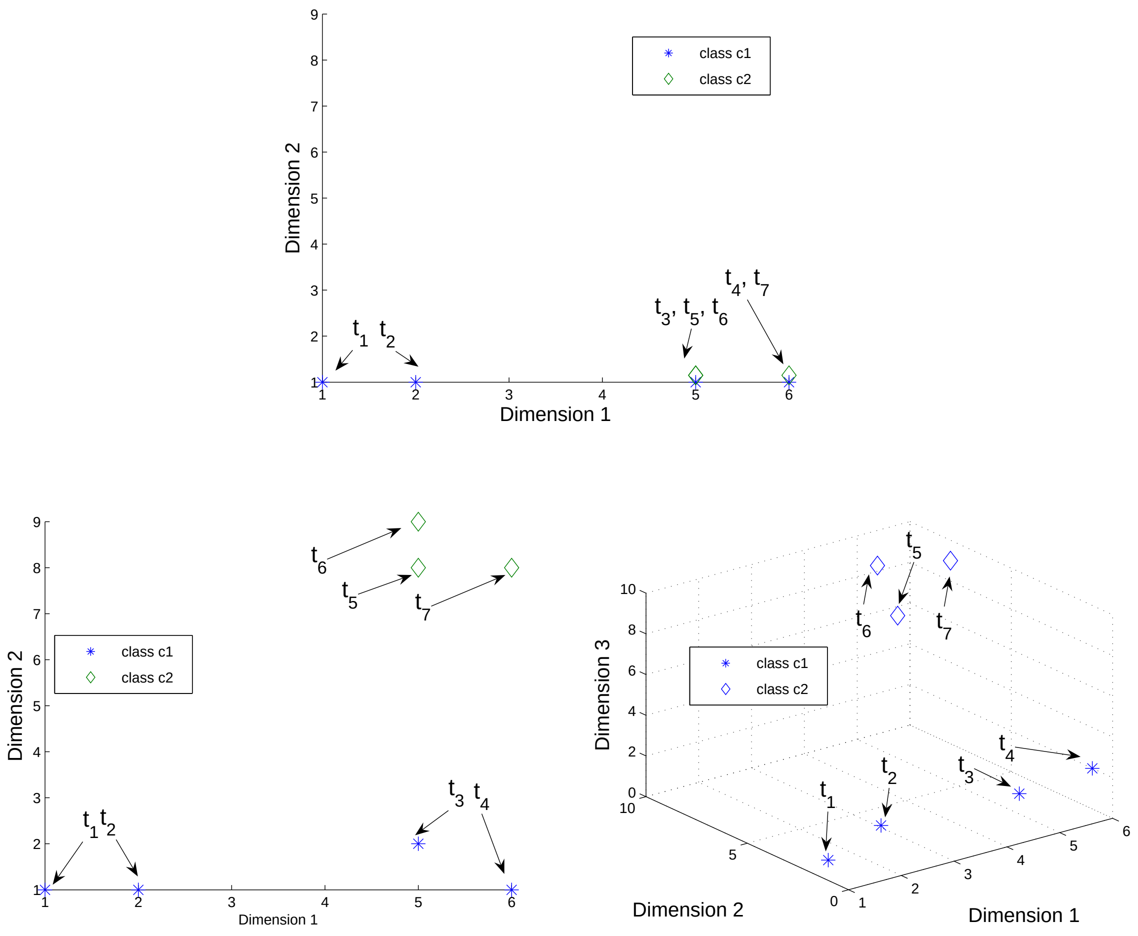


Figure 4.1: Plots of data in Table 4.1.

Prefix space	Clusters
$\mathcal{R}^1$	$S_1 = \{t_1, t_2\}$, $S_2 = \{t_3, t_4, t_5, t_6, t_7\}$
$\mathcal{R}^2$	$S_1 = \{t_1, t_2\}$, $S_2 = \{t_3, t_4\}$ $S_3 = \{t_5, t_6, t_7\}$
$\mathcal{R}^3$	$S_1 = \{t_1, t_2\}$, $S_2 = \{t_3, t_4\}$ $S_3 = \{t_5, t_6, t_7\}$

Table 4.2: The clusters in different prefix spaces.

Example 7 *Figure 4.1 shows the distribution of the time series in Table 4.1 in each prefix space, $\mathcal{R}^3$, $\mathcal{R}^2$ and $\mathcal{R}^1$. From Figure 4.1, we can observe the natural clusters in each prefix space. We summarize the clusters in Table 4.2. Cluster S_1 is stable in all three spaces, and thus can be used at early classification as early as timestamp 1. Clusters S_2 and S_3 are stable at timestamps 2 and 3 and thus can be used as early as at timestamp 2.* ■

We consider the clusters in the full length space $\mathcal{R}^L$ as the ground truth. To learn when a cluster can be used for early classification, we need to address two issues. First, we need to use a clustering approach to obtain the clusters in the full-length space. Second, we need to compute the MPLs of clusters.

We adopt single link MLHC [15], an agglomerative hierarchical clustering method to cluster the training data set in the full length space. It builds a hierarchical clustering tree level by level by merging all mutual nearest neighbor pairs of clusters at each level. Two clusters form a mutual nearest neighbor pair if they consider each other as the nearest neighbor. The distance between two clusters is measured by the minimum among all inter-cluster pair-wise distances. In the output hierarchical clustering tree (i.e., a dendrogram), a cluster represented by a leaf node is called a *leaf-cluster*, and a cluster represented by an internal node is called a *sub-cluster*. The whole training set is represented by the root, and thus is called the *root-cluster*.

A cluster S is called *1NN consistent* [14] if for each object (a time series in our case) $o \in S$, the 1NN of o also belongs to S. Immediately, we have the following.

Lemma 1 *The sub-clusters and the root-cluster generated by single link MLHC are 1NN consistent.*

[Proof] In the dendrogram generated by MLHC, we call a cluster represented by a leaf node a leaf-cluster, a cluster represented by an internal node a sub-cluster, and the cluster represented by the root the root-cluster.

A leaf-cluster contains only one time series. For a leaf-cluster s in the dendrogram generated by MLHC on space R^l, let $Sib(s)$ be the sibling cluster of s, which is another time series. Now, we show $NN^l(s) \cap Sib(s) \neq \emptyset$ by contradiction.

Suppose $NN^l(s) \cap Sib(s) = \emptyset$. Then, we can find $q \in NN^l(s)$. According to the assumption, $q \notin Sib(s)$. Then, $\forall s' \in Sib(s)$, we have $dist(s, s') > dist(s, q)$, which means

$dist(s, Sib(s)) > dist(s, q)$. Then, s cannot be merged with $Sib(s)$ in the process of MLHC. This contradicts with the assumption that $Sib(s)$ is the sibling of s.

For any sub-cluster S in the dendrogram, $\forall s \in S$, we have $Sib(s) \subseteq S$. Because $NN^l(s) \cap Sib(s) \neq \emptyset$, we have $NN^l(s) \cap S \neq \emptyset$. So, S is a 1NN consistent cluster. The lemma holds for the root cluster trivially since the root cluster contains all time series in question ■.

If all time series in a sub-cluster S carry the same label, S is called a *discriminative cluster*. We denote by $S.c$ the common class label of the time series in S. A discriminative cluster can be used in classification.

Example 8 *S_1, S_2 and S_3 in space $\mathcal{R}^3$ in Table 4.2 are discriminative clusters and can be used in classification. For example, $s_3 = (5, 8, 8)$ finds $t_5 \in S_3$ as the 1NN in $\mathcal{R}^3$. $S_3.c = c_2$ can be assigned to s_3.* ■

To explore the potential of a discriminative cluster S in early classification, we find the earliest prefix space in which S is formed and becomes stable since then. The corresponding prefix length is the minimum prediction length (MPL) of S.

One straight forward way is to apply MLHC on space $\mathcal{R}^{L-1}$ and check if S is preserved as a sub-cluster, and continue this process on the previous prefix spaces until the sub-cluster is not preserved. This can be very costly in computation.

An efficient way is to check if the 1NN consistence property holds for the cluster in the previous prefix spaces and the stability of the reverse neighbors of S.

For a sub-cluster S, in space $\mathcal{R}^l$ $(1 \leq l \leq L)$, we define the *reverse nearest neighbors* of S as $RNN^l(S) = \cup_{s \in S} RNN^l(s) \setminus S$. If S is well separated from other clusters, $RNN^l(S)$ is empty. Often, some sub-clusters in a training set may not be well separated from others. In such a case, $RNN^l(S)$ is the boundary area of S.

Definition 2 *[MPLs of clusters] In a training data set T with full length L, for a discriminative cluster S, $MPL(S) = k$ if for any $l \geq k$, (1) $RNN^l(S) = RNN^L(S)$; (2) S is 1NN consistent in space $\mathcal{R}^l$; and (3) for $l = k - 1$, properties (1) and (2) cannot be both satisfied.* ■

Example 9 *For discriminative clusters S_1, S_2 and S_3, $MPL(S_1) = 1$, $MPL(S_2) = 2$ and $MPL(S_3) = 2$. Take S_3 as an example, in spaces $\mathcal{R}^3$ and $\mathcal{R}^2$, S_3 is 1NN consistent and*

$RNN^3(S_3) = RNN^2(S_3) = \emptyset$. In space $\mathcal{R}^1$, S_3 is not 1NN consistent. Thus, $MPL(S_3) = 2$. ■

In a discriminative cluster S, there may be another discriminative cluster $S' \subset S$. $MPL(S)$ may be longer or shorter than $MPL(S')$. Moreover, for a time series $t \in T$, t itself is a leaf-cluster with $MPL(t)$ (Definition 1). Among all the discriminative or leaf clusters containing t, we can use the shortest MPL to achieve the earliest prediction using t.

As one drawback of the 1NN early classification method, the MPL obtained from a very small cluster may cause over-fitting. To avoid over-fitting, a user can specify a parameter *minimum support* to control the size of a cluster. We calculate the support of a cluster in a way that is aware of the population of the corresponding class.

Definition 3 (Support) *In a training set T, let $T_c = \{s \in T | s.c = c\}$ be the set of time series that have label c. For a discriminative cluster or a leaf cluster S such that $S.c = c$, $Support(S) = \frac{|S|}{|T_c|}$.* ■

We only use clusters passing the user-specified minimum support threshold for early prediction.

To summarize, given a training set T, the ECTS method works in two phases as follow.

Training Phase Given a *minimum support* threshold p_0, for a time series $t \in T$, let $SS = \{S | t \in S \wedge S \text{ is a discriminative cluster or a leaf cluster}\}$ be the set of discriminative or leaf clusters containing t.

$$MPL(t) = \min_{S \in SS, Support(S) \geq p_0} MPL(S).$$

The training process is to compute MPLs for all $t \in T$.

Classification Phase Same as the 1NN early classification method in Section 4.2.

Section 4.1 states that we want to build a classifier as accurate as the 1NN classifier using the full-length time series. The following result answers the requirement.

Theorem 3 *In a training data set T of full length L, assuming for any $t \in T$ and $1 \leq l \leq L$, $NN^l(t)$ contains only one time series*[1]*, ECTS has the same leave-one-out accuracy on T as*

[1] *If multiple 1NNs exist, we can select the 1NN of the smallest index.*

the 1NN classifier using the full-length time series. Moreover, ECTS is a serial classifier on the time series in T.

[Proof] In the leave-one-out classification, if a sequence $t \in T$ *is classified by sequence* q *on prefix* $t(1, k)$ *by the ECTS classifier, then* $MPL(q) \leq k$. *Since* t *is classified by* q, $t \in RNN^k(q)$ *in prefix space* $\mathcal{R}^k$. *From the learning process we know that there is a cluster* Q *containing* q, *and* $MPL(Q) = MPL(q)$. *Since* $t \in RNN^k(q)$, t *is either a member of cluster* Q *or a member of* $RNN^k(Q)$. *We consider the two cases one by one.*

If $t \in Q$, *because* $MPL(Q) \leq k$, *starting from length* k *to the full length* L, Q *is always 1NN consistent. Since* $t \in Q$, *in any prefix space* R^l, *where* $k \leq l \leq L$, *we have* $NN^l(t) \in Q$. *Because all members in* Q *have the same class label, by the ECTS classifier, we have* $C(t, k) = C(t, k + 1) = \cdots = C(t, L)$. *For* t, *the classifier is serial. Since* $C(t, L)$ *by ECTS is actually the 1NN classifier using the full length, so for* t, *ECTS gives the same prediction as the 1NN classifier using the full length time series.*

Let $t \in RNN^k(Q)$. *Since* $RNN^k(Q) = RNN^{k+1}(Q) = \cdots = RNN^L(Q)$, *we have* $t \in RNN^l(Q)$ *for any* $k \leq l \leq L$. *It means* $NN^l(t) \in Q$ *for any* $k \leq l \leq L$. *Because all members in* Q *have the same class label, by the ECTS classifier,* $C(t, k) = C(t, k + 1) = \cdots = C(t, L)$. *For* t, *the classifier is serial and ECTS gives the same prediction as the 1NN classifier using the full-length time series.*

Based on the above analysis, for any $t \in T$, *the class label predicted by ECTS will be the same as the class label predicted by the 1NN classifier using the full-length time series, and the ECTS is serial.* ∎

Theorem 3 indicates that the learned MPLs by ECTS are long enough to classify all time series in the training data set as accurately as the full length time series by a 1NN classifier. If the training data set is large enough and provides a good sample of the time series space, then, for an unlabeled time series, the ECTS classifier is expected to give the same accuracy as the full length 1NN classifier, and is expected to be serial.

We propose a framework of extending instance based classifiers for early classification through clustering. Particularly, we choose one nearest neighbor and single link hierarchical clustering. One of the disadvantages of using one nearest neighbor and single link hierarchical clustering is that they are both sensitive to noisy data. To overcome the sensitivity to noisy data, more robust clustering methods, such as hierarchical clustering using average link can be used. In this thesis, we focus on studying single link clustering and one nearest neighbor classifier because they together lead to the theoretical result shown in Theorem 3. Theorem 3 guarantees the leave-one-out accuracy of the proposed classifier to be the same as the one nearest neighbor classifier. It enables us to study in what extends the earliness can be achieved given a theoretical accuracy guarantee and validates the framework for early classification.

4.3.2 The Algorithm

To train an ECTS classifier, we integrate the computation of MPLs for clusters into the framework of MLHC. The algorithm of training the ECTS classifier is shown in Figure 5.6.

The training process requires many nearest neighbor queries and reverse nearest neighbor queries in various prefix spaces. To share the computation, we pre-compute the nearest neighbors for every time series $t \in T$ in all prefix space from $\mathcal{R}^1$ to $\mathcal{R}^L$ (line 1 in Figure 5.6). This pre-computation step takes time $O(|T|^2 \cdot L)$ where L is the full length.

Then, we apply the single link MLHC [15] to space $\mathcal{R}^L$. We implement the MLHC framework in lines 4-11. As stated before, in each iteration, MLHC merges all mutual nearest neighbor (MNN) pairs of clusters. When a new cluster S is formed in the process of MLHC, we compute its MPL (line 9). In line 10, we update the MPL of every time series in $s \in S$ if $MPL(S) < MPL(s)$. When the iteration terminates, the MPL of each time series s is the shortest MPL of all the clusters containing s.

All time series in a discriminative cluster should have the same label. In MLHC, if a cluster formed is not pure, that is, the time series in the cluster are inconsistent in class label, we do not need to merge the cluster further to other clusters in the next iteration. MLHC in our method terminates when no pure clusters are generated after the current round of iteration. In other words, our adaption of MLHC generates a cluster forest instead of a cluster tree.

To compute the MPL of a discriminative cluster S, according to Definition 2, we check whether the 1NN consistency and the RNN stability hold in space $\mathcal{R}^L$ and the prefix spaces $\mathcal{R}^l$ in prefix length descending order. If $MPL(S) = k$, to compute the MPL of S, we need to find $RNN^l(S)$ and check 1NN consistency in spaces $\mathcal{R}^l(L \geq l \geq (k-1))$. If the MPL of every discriminative cluster is computed from the scratch, it is very costly. Fortunately, we do not need to compute the MPL for every discriminative cluster from scratch. Instead, we can use the information provided by the clusters formed in the previous iterations. When computing the MPL of a discriminative cluster S, which is formed by merging clusters $S1$ and $S2$ in an iteration, two situations may arise.

First, $|S1| > 1$ and $|S2| > 1$. Without loss of generality, let us assume $MPL(S1) \leq MPL(S2)$. Cluster S must be 1NN consistent and RNN stable in spaces $\mathcal{R}^l$ for $L \geq l \geq MPL(S2)$. Thus, we only need to check if the 1NN consistency and RNN stability hold for S in space $\mathcal{R}^l$ for $MPL(S2) \geq l \geq 1$.

Second, if $|S1| = 1$ and $|S2| > 1$, S is always 1NN consistent in spaces $\mathcal{R}^l$ for $L \geq l \geq MPL(S2)$. For ECTS, to test the RNN stability of S, we only need to check if $RNN(S1) \backslash S2$ is stable in spaces $\mathcal{R}^l$ for $L \geq l \geq MPL(S2)$. This is because we already know that the RNNs of cluster $S2$ are stable in spaces $\mathcal{R}^l$ for $L \geq l \geq MPL(S2)$. If for some $l(L \geq l \geq MPL(S2))$ that S passes the test of l but fails for $l-1$, $MPL(S) = l$. If S passes the test, then $MPL(S) \leq MPL(S2)$. We need to check the 1NN consistency and the RNN stability in the prefix spaces of shorter prefix lengths.

The complexity of building the dendrogram in the full-length space by MLHC is $O(|T|^2)$. The worst case time complexity of ECTS is $O(|T|^3 L)$.

The advantage of the ECTS method is parameter free, and easy to understand, and more efficient comparing to the feature based method, the naive solution.

Our current method uses Euclidean distance which requires time series in the training data to be of the same length. For time series of different lengthes, instead of using Euclidean distances, we can use some distances, such as dynamic time warping (DTW) distances. For dynamic time warping distance, to learn the MPLs, we can first learn the clustering structures using full length time series with DTW distance. Then, we can shrink the individual time series by observing after which prefix, the time series changes its associated cluster comparing to the full length clusters.

```
Input: a training data set T;
Output: MPL(t) for each $t \in T$;
Method:
1:  pre-compute 1NNs for each $t \in T$ in all prefix spaces;
2:  compute the MPLs of leaf clusters and update the MPL
    for each $t \in T$;
3:  $n = |T|$, the number of time series in $T$;
4:  while $n > 1$
5:     compute the mutual nearest neighbor pairs;
6:     for each mutual nearest neighbor pair $(S_1, S_2)$
7:        merge $S_1$ and $S_2$ into a parent cluster $S$, $n = n - 1$;
8:        if all time series in $S$ carry the same label then
9:           compute the MPL of $S$;
10:          update the MPL for each time series in $S$;
          end if
       end for
11:    if no new discriminative clusters are generated in this
       round then break;
    end while
```

Figure 4.2: The algorithm of the training phase in ECTS.

4.3.3 Relaxed ECTS

If we apply the 1NN classifier in the full length space $\mathcal{R}^L$ to classify the training data T using leave-one-out method, we may get some time series miss classified. We denote $T_{mis} \subseteq T$ as the set of time series that are not classified correctly in the leave-one-out process. One important reason of misclassification is that some time series in T_{mis} are on the decision boundary. The nearest neighbors of those time series may have a considerable chance to fall into other clusters in the prefix spaces. To compute the MPL for a cluster, Definition 2 requires the RNN stability from prefix space $\mathcal{R}^{MPL(S)}$ till the full length space. The time series in T_{mis} may hinder many discriminative clusters from meeting the stability requirement. Can we relax the requirement of MPL by ignoring the instability caused by the time series in T_{mis}?

Definition 4 (Relaxed MPL) *In a training data set T with full length L, for a discriminative cluster S, $MPL(S) = k$ if for any $l \geq k$, (1) $RNN^l(S) \cap (T - T_{mis}) = RNN^L(S) \cap (T - T_{mis})$; (2) S is 1NN consistent in space $\mathcal{R}^l$; and (3) for $l = k - 1$, (1) and (2) cannot be both satisfied.*

Specifically, if S is a leaf cluster and $RNN^L(S) \cap (T - T_{mis}) = \emptyset$, then we define $MPL(s) = L$. ■

We call the ECTS classifier using the above relaxed MPL the relaxed version of ECTS. In the relaxed version, we relax the condition of RNN stability to partial stability. The relaxed ECTS has the following property.

Theorem 4 *In a training data set T of full length L, assuming for any $t \in T$ and $1 \leq l \leq L$, $NN^l(t)$ contains only one time series*[2]*, the relaxed ECTS has the same leave-one-out accuracy on T as the 1NN classifier using the full-length time series.*

[Proof] In the leave-one-out classification process, if a time series t is classified by sequence s on prefix $t(1, k)$ by the relaxed ECTS classifier, t either is a member of a cluster Q or belongs to $RNN^k(Q)$.

In the case where $t \in Q$, same as the proof in Theorem 3, we can prove the relaxed ECTS makes prediction as accurate as the 1NN classifier using the full length time series.

[2] *Again, if multiple 1NNs exist, we can select the 1NN of the smallest index .*

In the case that $t \in RNN^k(Q)$, we need to consider two subcases.

In the first subcase, $t \in RNN^k(Q) \cap (T - T_{mis})$, same as the proof in Theorem 3, we can proof the relaxed ECTS makes prediction as accurate as 1NN classifier using the full length time series.

In the second subcase, $t \in RNN^k(Q) \cap T_{mis}$. Although $C(c, k) = C(s, L)$ cannot be guaranteed, since t cannot be correctly classified by the 1NN classifier using the full length time series, the subcase does not make the accuracy of the relaxed ECTS lower than that of the 1NN classifier using the full length time series.

Based on the above analysis, we conclude, by using the relaxed ECTS, the leave-one-out accuracy is at least the same as using the 1NN classifier using the full length time series.

■

Comparing with ECTS, the average MPLs learned by the relaxed ECTS is expected to be shorter because for a cluster, we only require a subset of its RNN to be stable. When classifying unlabeled time series, the relaxed ECTS is expected to give the same prediction as ECTS and make the prediction on a shorter prefix. We will verify the expected property in our experiments.

4.4 Experimental Results

The UCR time series archive [30] provides 23 time series data sets which are widely used to evaluate time series clustering and classification algorithms. In each data set, the time series have a fixed length. Each data set contains a training set and a testing set. The 1NN classification accuracies using the Euclidean distance on the testing sets are provided in the archive as well.

Table 4.3 lists the results on all the 7 data sets in the archive where the full-length 1NN classifier using Euclidean distance achieves an accuracy of at least 85%. The 1NN classifier can be regarded effective on those data sets. The seven data sets cover cases of 2-class and more-than-two-class.

Table 4.3 compares 6 methods. We use the 1NN classifier using the full length (denoted by full 1NN) as the baseline. In addition to ECTS and relaxed ECTS, we also report the

Dataset		**ECTS**	RelaxedECTS	Early1NN	1NNFixed	**Full1NN**	SCR
Wafer	Accuracy	99.08%(−0.47%)	99.08%(−0.47%)	99.16%(-0.39 %)	99.32%(-0.23 %)	99.55%	93% (-6.58 %)
2 classes	Ave. len.	67.39 (44.34%)	67.39 (44.34%)	122.05 (80.30%)	110 (72.37%)	152	55.36 (36.42%)
1000 training inst.	Train. time	152.90 sec	161.08 sec	3.22 sec	1.58 sec	0 sec	164.59 sec
6174 testing inst.	Class. time	0.053 sec	0.038 sec	0.103 sec	0.004 sec	0.004 sec	0.204 sec
Gun-Point	Accuracy	86.67% (−5.1%)	86.67% (−5.1%)	87.3%(-4.41 %)	91.3%(-0.03 %)	91.33%	62.67%(−31.36%)
2 classes	Ave. len.	70.387 (46.92%)	70.387(46.92%)	107.24 (71.49%)	140 (92.67%)	150	116.17 (77.45%)
50 training inst.	Train. time	0.39 sec	0.406 sec	0.09 sec	<0.01 sec	0 sec	0.86 sec
150 testing inst.	Class. time	0.002 sec	0.003 sec	0.004 sec	0.002 sec	0.002 sec	0.11 sec
Two Patterns	Accuracy	86.48% (−4.97%)	86.35% (−5.11%)	87.25%(-4.12 %)	90.72%(-0.8%)	91%	94.75% (+4.12%)
4 classes	Ave. len.	111.097 (86.79%)	110.743 (86.52%)	113.24 (88.5%)	124 (96.88%)	128	86.13(67.29 %)
1000 training inst.	Train. time	48.76 sec	47.46 sec	2.18 sec	1.34 sec	0 sec	65.23 sec
4000 testing inst.	Class. time	0.101 sec	0.07 sec	0.077 sec	0.003 sec	0.003 sec	0.125 sec
ECG	Accuracy	89% (+1.17%)	0.89% (+1.17%)	89%(+1.17 %)	89%(+1.17 %)	88%	73%(−17.05%)
2 classes	Ave. len.	74.04 (77.13%)	57.71 (60.11%)	83.12 (86.58%)	92 (92.71%)	96	37.5 (39.06%)
100 training inst.	Train. time	0.83 sec	1.20 sec	0.1 sec	0.02 sec	0 sec	4.22 sec
100 testing inst.	Class. time	0.004 sec	0.004sec	0.004 sec	0.001 sec	0.001 sec	0.062 sec
Syn. Control	Accuracy	89% (+1.17%)	88.3%(+0.34%)	88.33%(+0.38 %)	88%(+0 %)	88%	58.33%(−33.72%)
6 classes	Ave. len.	53.98 (89.97%)	52.73(87.9%)	55.09 (91.82%)	60 (100%)	60	29.39 (50%)
300 training inst.	Train. time	4.64 sec	4.992 sec	0.12 sec	0.06 sec	0 sec	21.09 sec
300 testing inst.	Class. time	0.004 sec	0.006 sec	0.004 sec	0.001 sec	0.001 sec	0.02 sec
OliveOil	Accuracy	90% (+3.8%)	90% (+3.8%)	90%(+3.8 %)	83.33%(-3.92%)	86.7%	36.7%(−57.68%)
4 classes	Ave. len.	497.83 (82%)	497.83 (82%)	526 (92.28%)	406 (71.23%)	570	500 (87.72%)
30 training inst.	Train. time	0.22 sec	0.28 sec	0.08 sec	0.02 sec	0 sec	2.03 sec
30 testing inst.	Class. time	0.058 sec	0.0378 sec	0.043 sec	0.006 sec	0.006 sec	0.016 sec
CBF	Accuracy	85.2% (+0%)	85.2% (+0%)	86.89%(+1.98 %)	83.2%(-2.35%)	85.2%	55.22% (-35.18%)
3 classes	Ave. len.	91.73 (71.5%)	91.52 (71.50%)	103.20 (80.63%)	54 (42.19%)	128	46 (35.93 %)
30 training inst.	Train. time	0.09 sec	0.109 sec	0.02 sec	<0.01 sec	0 sec	0.24 sec
900 testing inst.	Class. time	0.001 sec	0.001 sec	0.001 sec	0.001 sec	0.001 sec	0.015 sec

Table 4.3: Results on seven datasets from UCR Time Series Archive

results of the 1NN early classification method (Section 4.2, denoted by 1NN Early), the 1NN fixed method which will be introduced in Section 4.4.2, and the SCR method [68]. ECTS and relaxed ECTS use only an optional parameter, *minimum support*, to avoid overfitting. All results of ECTS and relaxed ECTS in Table 4.3 are obtained by setting *minimum support*$= 0$.

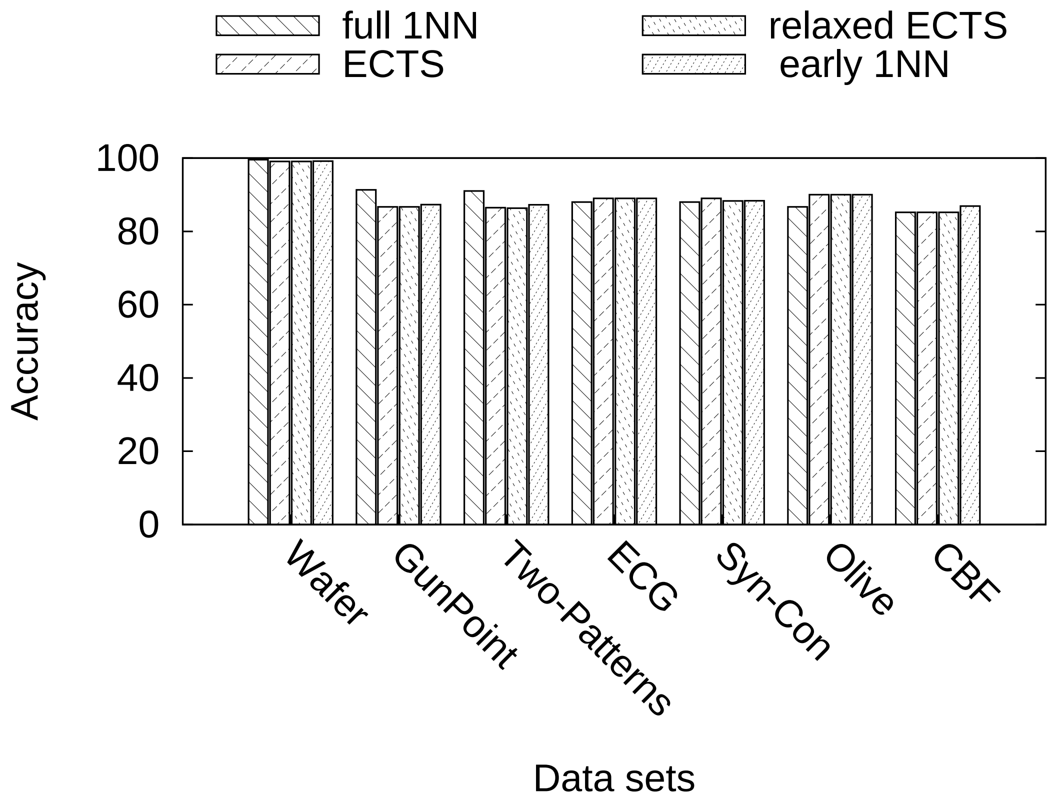
All the experiments were conducted using a PC computer with an AMD 2.2GHz CPU and 1GB main memory. The algorithms were implemented in C++ using Microsoft Visual Studio 2005.

4.4.1 Results of ECTS Methods

In Figure 5.12, we compare the performance of full 1NN, ECTS, relaxed ECTS and early 1NN in terms of classification accuracy and average prediction length ($Cost(C, T')$ defined in Section 4.1.

On data sets Wafer and Gun-point, the average prediction lengths of ECTS are shorter than half of the full lengths. On the other five data sets, the average prediction lengths of ECTS are from 71% to 89% of the full lengths. In terms of accuracy, except for data sets Gun-point and Two-patterns, ECTS has an accuracy almost the same or even slightly better than that obtained by the full-length 1NN method. On Gun-point and Two-patterns, ECTS is about 5% lower in accuracy. The results on the seven data sets show that ECTS can preserve accuracy consistently as the baseline method. At the same time, ECTS can achieve considerable saving in prediction length.

Similar to ECTS, the results of relaxed ECTS confirm that relaxed ECTS can also achieve early classification while retaining the accuracy as the baseline method. Especially, on data set ECG, relaxed ECTS uses an average prediction length of 57.71,(60.11% of the full length) and ECTS uses an average prediction length of 74.04 (77.13% of the full length). Relaxed ECTS achieves significantly earlier classification than ECTS. Also, on the Synthetic Control data set, relaxed ECTS uses shorter prediction than ECTS. The results are consistent with our discussion in Section 4.2. The relaxed ECTS may be able to use a shorter prefix length to obtain nearly the same accuracy as ECST. In Table 4.3, the runtime of the training algorithms in ECTS and the relaxed ECTS (i.e., computing the MPLs for



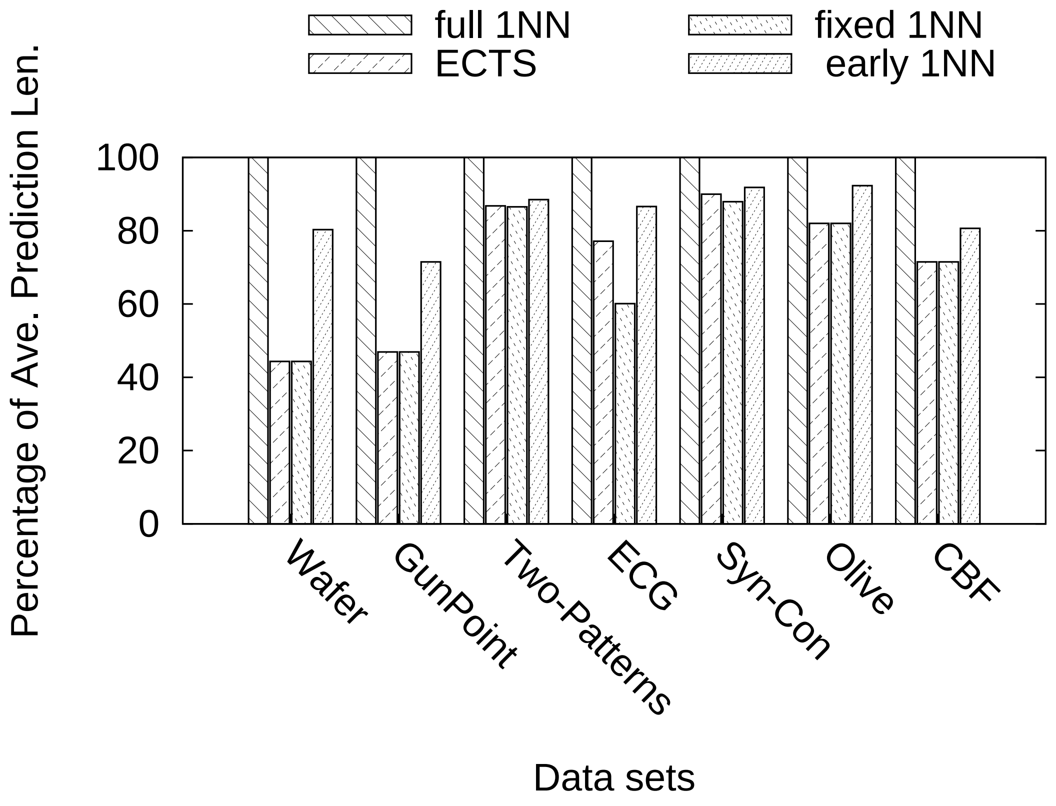


Figure 4.3: Comparison among full 1NN, ECTS, relaxed ECTS and early 1NN

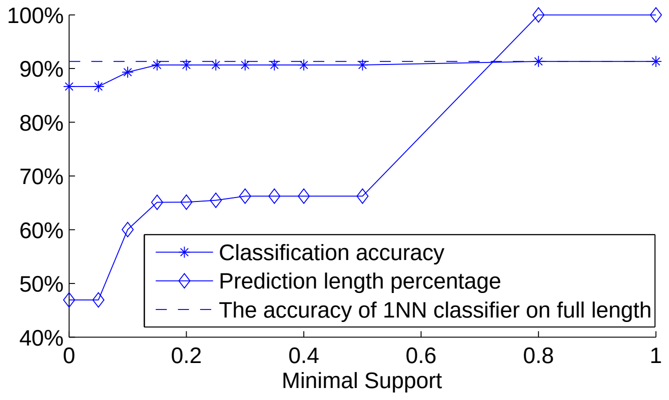


Figure 4.4: Accuracy and ave. length vs. minimum support.

all training time series) and the average runtime of the classification phase of the ECTS methods are also reported. The relaxed ECTS takes slightly longer time in training. Both methods are very fast in classification.

From Figure 5.12, we can also see that on all the 7 data sets, the early 1NN method achieves almost the same accuracies as the baseline method. In terms of accuracy, the early 1NN method, ECTS, and relaxed ECTS are all quite reliable. ECTS and relaxed ECTS always achieve a shorter average prediction length than the early 1NN method. This result confirms that finding MPLs through clusters helps to obtain shorter MPLs.

ECTS and relaxed ECTS have an optional parameter, *minimum support*. If the user does not set the parameter, the default value is 0. As shown in the experiments, in most cases, the performance of ECTS is satisfactory by using the default value. On data sets Gun-point and Two-patterns, ECTS and relaxed ECTS are about 5% lower in accuracy comparing to the full length 1NN. As explained before, when *minimal support*$= 0$, ECTS may over fit a training set and thus may slightly lose accuracy. By increasing *minimum support*, overfitting can be reduced. Figure 4.4 shows the effect on the Gun-point data set. When *minimum support* increases, the accuracy of ECTS approaches the accuracy of the full length 1NN classifier quickly. As the tradeoff, the average prediction length increases, too. By using parameter *minimum support*, ECTS can reduce overfitting effectively.

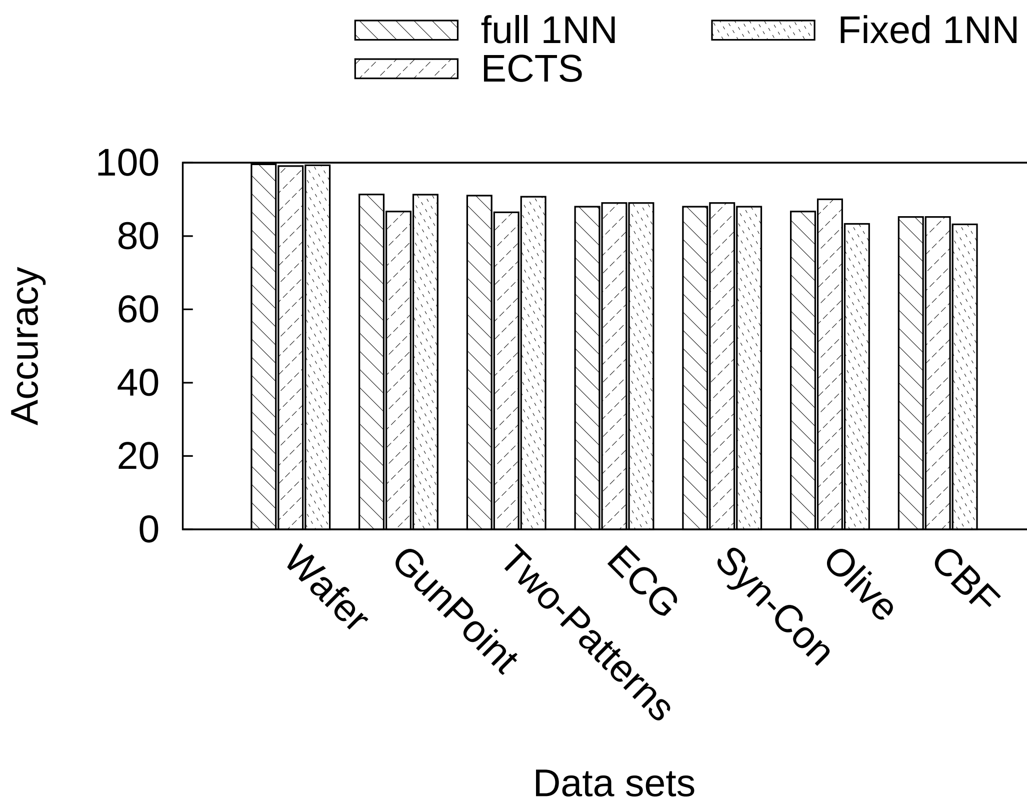
By comparing ECTS, relaxed ECTS and the early 1NN method against the baseline method, we can conclude that ECTS, relaxed ECTS and the early 1NN method have a common property that they can all make early classification while retain an accuracy comparable to using full length 1NN classifier. relaxed ECTS is the best at finding short prediction length. ECTS has the similar performance as relaxed ECTS in most cases. Early 1NN cannot achieves early classification as well as ECTS and relaxed ECTS, but it requires significantly less training time.

4.4.2 Comparison with 1NN Fixed

Is learning different MPLs for different time series necessary? Can we just learn a fixed MPL for all time series in a training set? Let us consider the following simple early classification method called *1NN fixed*. Given a training set T of full length L, we calculate the 1NN classification accuracy p in the full space $\mathcal{R}^L$. Then, we check the prefix spaces $\mathcal{R}^{L-1}, \mathcal{R}^{L-2}, \ldots$ until prefix space $\mathcal{R}^k$ such that the accuracies in spaces $\mathcal{R}^{L-1}, \ldots, \mathcal{R}^{k+1}$ are at least p, and the accuracy in space $\mathcal{R}^k$ is lower than p. We use $(k+1)$ as the MPL for all time series. That is, in classification, we always read the length-$(k+1)$ prefix of a time series s to be classified, and find the 1NNs of s among the length-$(k+1)$ prefixes of the time series in T to classify s.

Interestingly, 1NN fixed is a simplified special case of ECTS in 2-class situations under the assumption that each class forms one discriminative cluster in the full length space $\mathcal{R}^L$. However, since 1NN fixed does not consider the different early classification capabilities of the clusters in the hierarchy, it may use longer prefixes for classifications. Furthermore, when there are multiple classes or multiple large discriminative clusters, the 1NN fixed method may not work well since it requires the overall accuracy to be high and cannot identify clusters which are separated from other clusters earlier than the overall accuracy is satisfied.

We compare 1NN fixed with ECTS in Figure 4.5. On data sets Wafer, ECG and Synthetic-control, ECTS using *minimum support*$= 0$ achieves a shorter average prediction length than the 1NN fixed method. The two methods have similar accuracies. Especially, for the 6 classes synthetic control data set, the 1NN fixed method has to use the full length. ECTS uses a shorter prediction length. The saving mainly comes from the class cyclic, which is classified using an average length of 45. To handle multi-class situations, the 1NN



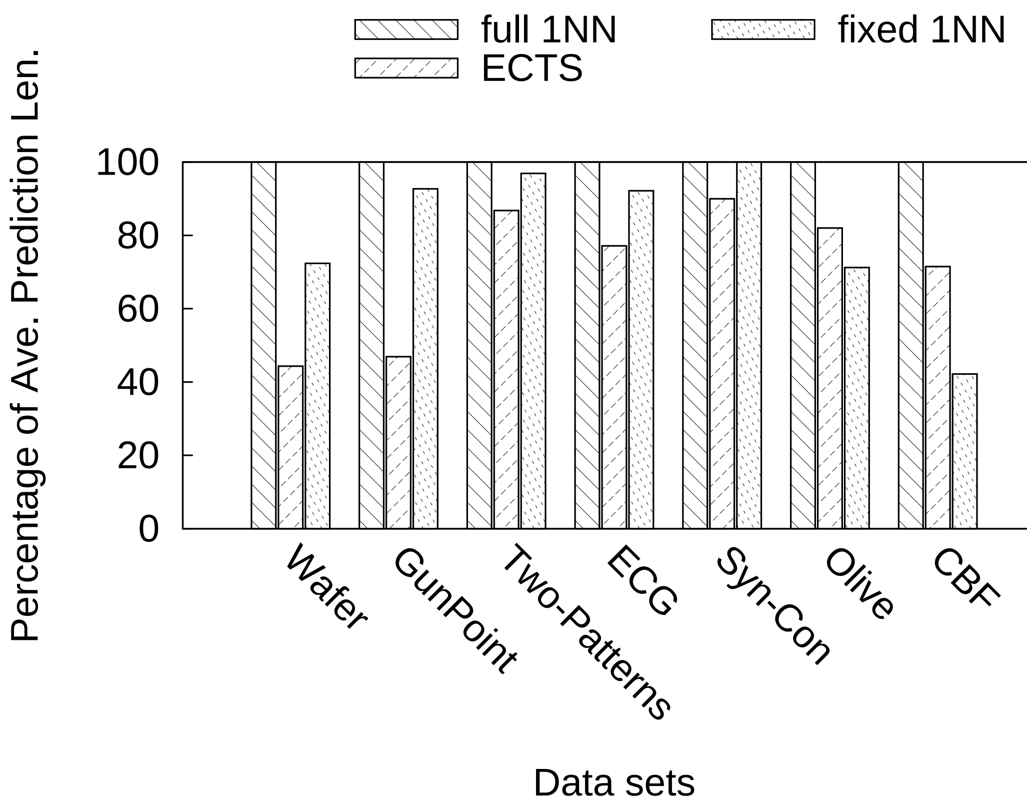


Figure 4.5: Comparison between ECTS and fixed 1NN

fixed method cannot classify one class earlier if the other classes are mixed together in the prefix spaces.

On data set Gun-point, by setting *minimum support*$=$ 15% to reduce overfitting, ECTS obtains an accuracy comparable to the 1NN fixed method, but uses a remarkably shorter average prediction length. Similar results are observed on data set Two-patterns.

From Table 4.3, we can see that on data sets OliveOil and CBF, interestingly, the 1NN fixed method is less accurate than ECTS but can obtain shorter average prediction length. For example, on data set OliveOil, ECTS obtains an accuracy of 90% and 1NN fixed method makes only 83.33%. The 1NN fixed method obtains an average prediction length of 406 and ECTS gives a length of 497.63. By analyzing the training set of 30 time series, we find that, training samples 7 and 9 are the cause of the dropping accuracy in the 1NN fixed method, which means the MPLs (406) of those two samples learned by the 1NN fixed method are not long enough to make accurate classification. In ECTS, the learned MPLs vary from 117 to 570 for the training samples. For samples 7 and 9, the learned MPLs are 567 and 570, respectively. Why does ECTS learn longer MPLs for samples 7 and 9? In the full length space, training sample 9 has an empty RNN set. The RNN set of training sample 7 consists of samples from two classes. Those RNN sets suggest that samples 7 and 9 are likely on the decision boundary. In contrast to the 1NN fixed method, ECTS can find longer MPLs for samples likely on the decision boundary to reduce the possible misclassification led by such a sample.

From the above analysis, we can conclude that in most cases, the ECTS method can use significantly shorter prediction to achieve very similar accuracy as the fixed 1NN method. ECTS is especially suitable for more-than-two-class early classification task.

4.4.3 Comparison with SCR

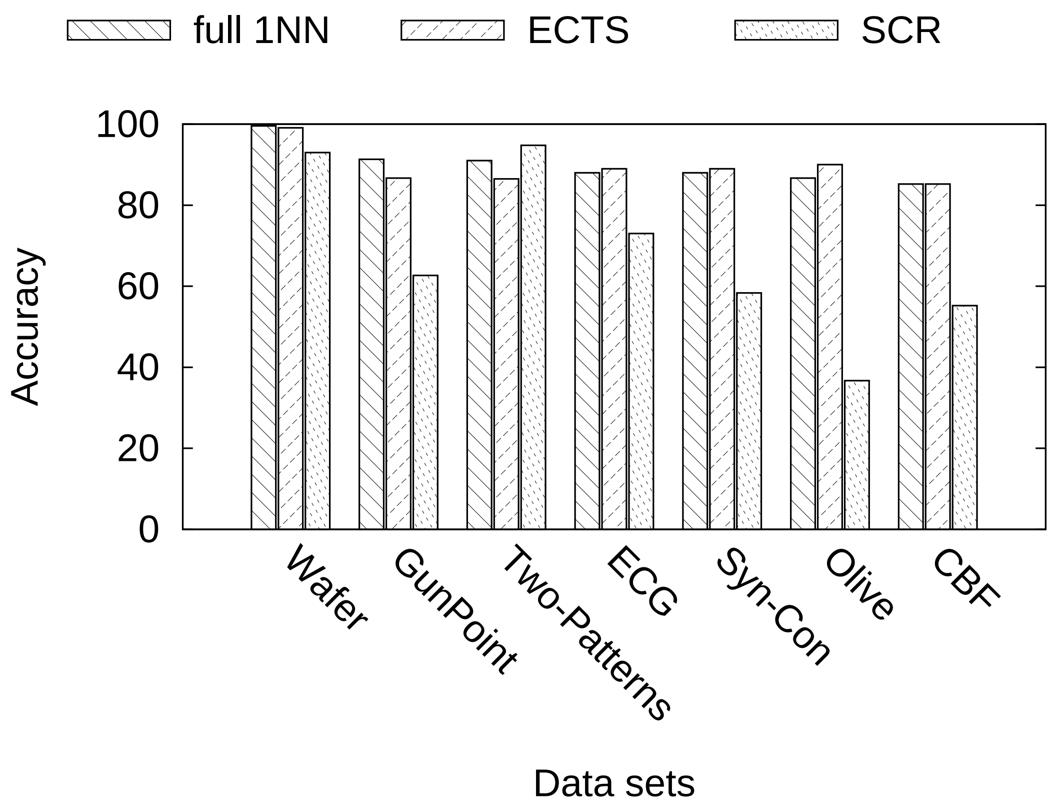
We also compare ECTS with our previous symbolic method SCR in Chapter 3, a rule based classifier proposed to solve the problem of early prediction for symbolic sequences.

Since SCR can only handle discrete values, k-means ($k = 3$) is used to discretize values in the time series into 3 values. In the classification, we do the online discretization using learned thresholds on the training data set. SCR requires a parameter, expected classification accuracy, which is set to the full length 1NN accuracy. The other parameter of SCR, *minimal support*, is set to 0.

We compare SCR with ECTS in Figure 4.6. Although SCR sometimes uses a shorter average prediction length, the accuracies are significant lower than the expected values. Comparing to SCR, ECTS makes early classification reliable in accuracy. In terms of efficiency, ECTS is much faster than SCR in training.

4.5 Summary

In this Chapter, we propose the ECTS classifier to tackle the problem of early classification of numerical time series data. ECTS extends the 1NN classifier to achieve early classification while retains nearly the same accuracy as that of the 1NN classifier using the full-length time series. In the experiments, we show that the ECTS methods are effective and superior to other existing early classification methods.



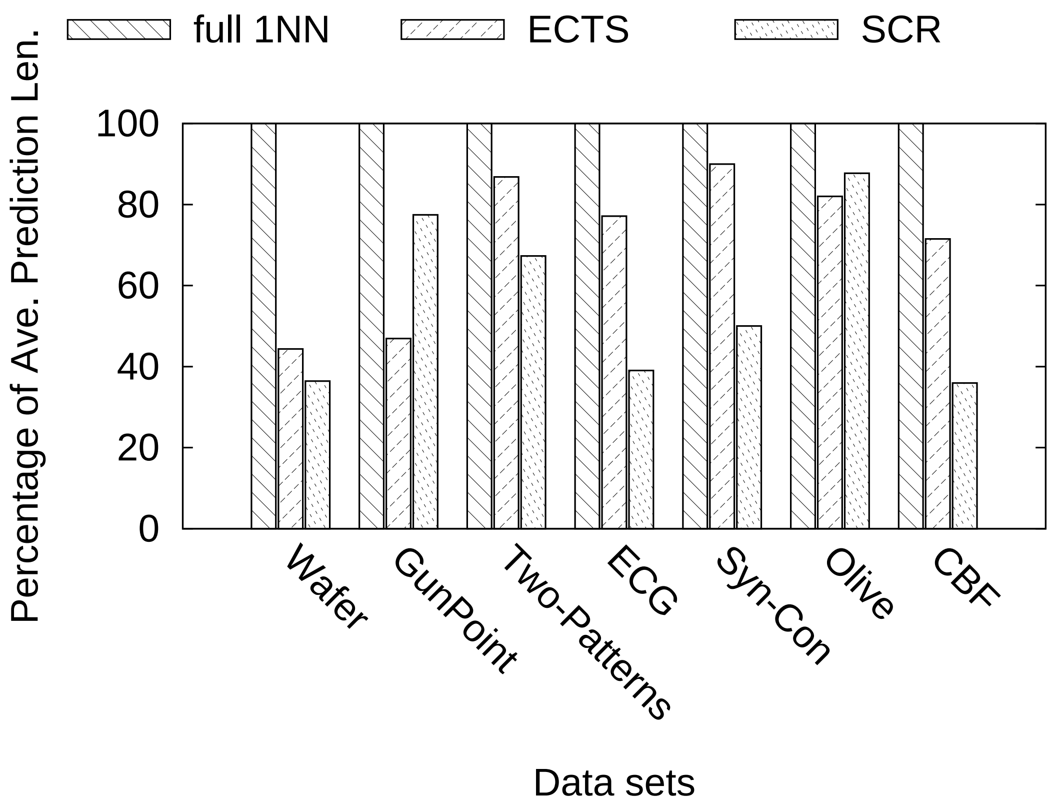


Figure 4.6: Comparison between ECTS and SCR

Chapter 5

Feature Extraction on Time Series Early Classification

5.1 Motivation and Problem Description

In Chapter 4, we propose an ECTS classifier for time series early classification. The ECTS classifier is a nearest neighbor based classifier. Ye *et al.* [72] pointed out that although the nearest neighbor classifier is a simple and effective classifier for time series classification, the time and space requirements in the classification step may limit its scalability. Furthermore, nearest neighbor classifiers only provide a classification result without extracting and summarizing patterns from the time series. It is not easy for users to interpret the classification result and gain insights into the data.

The interpretability of an early classifier is particularly important in early time series classification applications. A feature based early classifier, SCR, described in Chapter 3 extracts features from sequences and organizes features into sequential rules. The classifier provides useful information for users to understand the importance of features and the temporal relationship among features. But SCR is designed for symbolic sequences. When applying the method to time series, the classification quality is greatly constrained by the quality of discretization.

In this Chapter, we propose a new feature selection method to serve the purpose of monitoring time series online and detecting class label as early as possible. This feature selection method does not require discretization as the preceding step.

There has been some works on extracting features from sequence/time series data for sequence classification. In the following, we briefly discuss two methods which are closely related to the idea we propose.

Ji *et al.* [25] proposed a feature selection method on symbolic sequences to efficiently search for the distinguishing patterns. Distinguishing patterns are subsequences which are frequent in one class and infrequent in other classes. Distinguishing patterns intuitively represent the characteristics of a certain class of sequences and can be utilized for time series classification.

However, distinguishing patterns are defined on symbolic sequences. To extend the concept of distinguishing patterns to time series, we need to address the following challenges. First, how to represent a distinguishing pattern on time series and define pattern matching? For symbolic sequences, given a length k subsequence, exact matches or approximate matches can be defined. The matching is based on the portion and the positions of the identical symbols. But for time series, since it is numerical, we do not have the concept of the identical symbols. Therefore, we need to define the pattern matching on time series in another way. Second, how to generate distinguishing patterns for time series? For symbolic sequences, which are composed by a limited alphabet of symbols, the possible length k subsequences can be enumerated with various kinds of pattern mining techniques [25]. But for time series, the data is continuous and there is no such a finite alphabet. Therefore, we need to design new efficient methods to generate patterns from time series.

Ye *et al.* [72] proposed a feature selection method for time series classification, called shapelets on time series. "Informally, shapelets are time series subsequences which are in some sense maximally representative of a class" [72]. Shapelets provide an interesting way to represent features on time series and to define the matching. A feature on time series is represented as a pair (s, δ), where s is a time series subsequence, and δ is a distance threshold. A query time series subsequence s' is considered matching a feature (s, δ) if $distance(s, s') \leq \delta$. Ye *et al.* [72] used Euclidean distance as the distance measurement, and the matching is defined on the time series subsequences of the same length. The distance threshold δ is learned by maximizing the information gain. Among all the shapelet candidates (s, δ), the shapelet for distinguishing two classes is the one which separates the two classes with the best information gain. In the process of learning shapelet, maximizing information gain is the criterion used in both learning the thresholds for shapelet candidates and choosing the best shapelet. For multiple class classification, the shapelet selection process is integrated with the construction of a decision tree.

As shown in [72], using shapelets provides an effective approach for time series classification with good interpretability. However, we cannot directly apply shapelets for time series early classification. The shapelet method focuses on learning global features with the maximal information gain. Ideally, a perfect Shapelet of a class is the one representing all time series in one class but does not cover any time series in other classes. When building the decision tree, features with higher information gain will be the ancestors of features with lower information gain. However, for early time series classification, some local distinctive features, which do not have the best information gain, may be important for detecting the class labels online as early as possible. Here, a local distinctive feature is the one which represents a subset of time series in one class but very rare in other classes.

In Figure 5.1, we give an example to illustrate why the local distinctive feature is important for early classification. In Figure 5.1, we have two classes of time series, the diamond class and the star class. Feature A is shared by a subset of time series in the diamond class and does not appear in the star class at all. Feature B covers all time series in the diamond class but not any in the star class. Feature B is the shapelet of this data set since it has a higher information gain than feature A. Feature A can determine half cases in the diamond class but it does not cover as many cases as feature B, and we consider feature A as a local feature comparing to feature B. But if we want to determine class labels as early as possible, feature A represents half of the cases in the diamond class and feature A precedes feature B. Therefore, some local distinctive features are useful for early classification. In order to select local distinctive feature for early classification, we need new strategies to extract features from time series other than selecting features which maximize the information gain.

In this chapter, we propose a new feature selection method for building a feature based time series early classifier. The feature selection is particularly designed for detecting time series class labels as early as possible online. Since the proposed method extends the ideas

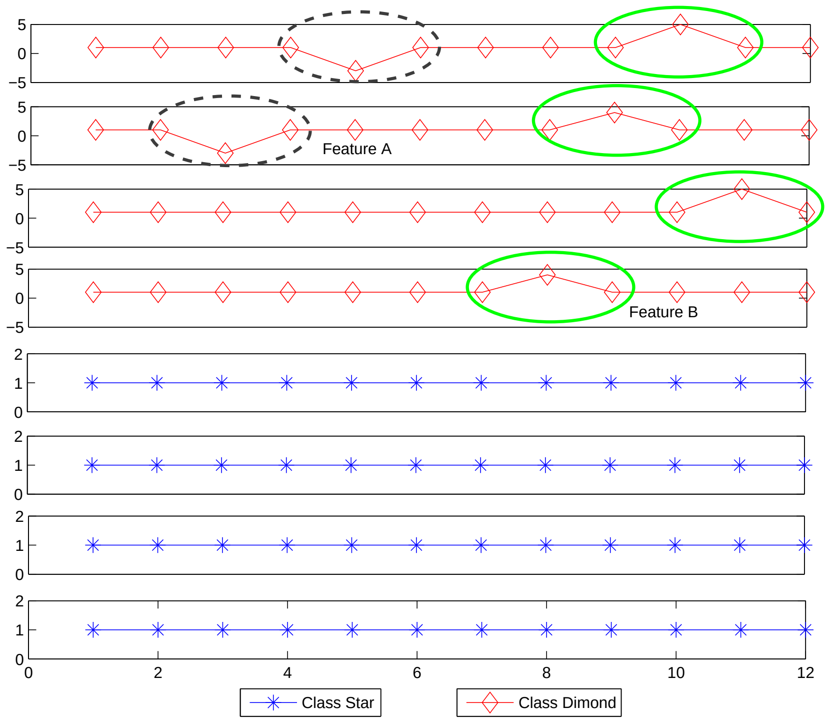


Figure 5.1: Local distinctive feature example.

of distinguishing patterns [25] and Shapelets on time series [72], in the following, we refer the method as *Early Distinctive Shapelets Classifier(EDSC)*.

5.2 EDSC

In this section, we propose the EDSC classifier for early classification on time series. We first define distinctive features on time series and introduce two feature learning methods in Section 5.2.1. Then, we propose a simple rule based classifier through feature selection in Section 5.2.2. In Section 5.2.3, we discuss some computational issues of the EDSC framework.

5.2.1 Feature Learning

Same as the representation of a shapelet [72], we represent a feature on time series as a time series subsequence and a distance threshold. We consider all the possible time series subsequences in the training data set as candidate features. Formally, we define a feature and feature matching as follows.

Definition 5 (Feature Representation) *Given a training data set T, a feature $f = (s, \delta, c)$ is a subsequence of a time time series $t \in T_c$ with a distance threshold δ. T_c is the subset of T containing all time series in class c. Before learning the distance threshold, a feature candidate is represented as $(s, ?, c)$. For a feature f, we refer class c as the target class, and the other classes as the non-target classes.* ■

Definition 6 (Feature Matching) *Given a feature $f = (s, \delta, c)$ and a time series t which has a subsequence s', where $distance(s', s) \leq \delta$, we say t has a match with feature f.* ■

The shapelet method [72] learns the distance threshold by maximizing the information gain. As we analyzed before, instead of only focusing on the global features, for early classification, we prefer some local features which are distinctive and early. In Chapter 3, we point out that the effective features for temporal sequences early classification are *distinctive*, *early*, and *frequent*.

In the step of learning the threshold for a candidate feature, we focus on learning distinctive features. We will consider the other two properties, the earliness and the frequency of features in the step of feature selection.

We define *Distinctive Features* on time series as follow,

Definition 7 (Distinctive Features) *Given a feature $f = (s, \delta, c)$, for any time series q which has a match with (s, δ, c), if q has a high probability belonging to class c, we call $f = (s, \delta, c)$ a distinctive feature.* ■

The above definition basically defines that if (s, δ, c) is a distinctive feature of a class c, for any time series which has the feature, it is very likely belonging to class c.

Given a candidate feature $(s, ?, c)$, we want to learn a distance threshold which makes it a distinctive feature. We know that some time series subsequences do not carry the characteristics to distinguish one class from other. For some feature candidates which are actually not distinctive, the learned threshold may be zero or an over-fitted threshold for the training data. The over-fitted features will be handled in the feature selection step.

Learning features through KDE

Given a candidate feature $f = (s, ?, c)$, we first compute its *best match vector* in the training data set T and then apply *kernel density estimation (KDE)* [47] on the best match vector to learn the threshold.

Definition 8 (Best Match Distance) *Given a time series subsequence s of length k and a time series t of length $L \geq k$, the best match distance of s and t is $BestMatchDist(s,t) = min\{Dist(s,s')|s'$ is any length k subsequence of t $\}$.* ■

Definition 9 (Best Match Vector) *Given a feature candidate $(s, ?, c)$ and the training data set T of N time series, the best match vector of $(s, ?, c)$ is $V = \langle d_1, d_2, ..., d_N \rangle$ where $d_i = BestMatchDist(s, t_i)$* ■

Kernel density estimation has been widely used in machine learning. Given a random sample $x_1, x_2, ..., x_N$ drawn from a probability density function $f(X)$. The kernel density of $f(X = x)$ can be computed by

$$\hat{f}(X = x) = \frac{1}{nh} \sum_{i=1}^{N} K(\frac{x - x_i}{h}) \tag{5.1}$$

where K is the kernel function and h is the smoothing factor [47].

We choose the gaussian kernel which is

$$K(\frac{x - x_i}{h}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_j)^2}{2h^2}} \tag{5.2}$$

To select an optimal bandwidth, given the gaussian kernel, we use a widely adopted approach to estimate the bandwidth [58],

$$h_{optimal} = 1.06\sigma n^{-\frac{1}{5}} \tag{5.3}$$

If we have J classes, by fitting the nonparametric density estimation $\hat{f}_k(x)$ separately for each class, we can compute the probability of sample x belonging to a certain class j, using

$$P(C(x) = j|X = x) = \frac{\pi_j \hat{f}_j(x)}{\sum_{k=1}^{J} \pi_k \hat{f}_k(x)} \tag{5.4}$$

where π_j is the class priors [47].

In the best match vector $V = \langle d_1, d_2, ..., d_N \rangle$ of a feature candidate $f = (s, ?, c)$, each item d_i in vector V corresponds to a training time series $t_i \in T$ and t_i is represented as its best match distance to s. Suppose we learned a threshold δ for f, a training time series t_i has a match with f if $d_i \leq \delta$. A training time series t_i does not have a match with f if $d_i > \delta$ since t_i cannot have a subsequence with a distance to s within δ.

To learn the distinctive feature, according to Definition 7, we want that for any time series t matches the feature, the probability of t belonging to the target class is high. One naive way is to interpret the class frequencies on the training data set as the expected probability. For example, we can learn a threshold δ for a feature candidate $f = (s, ?, c)$ such that for the training time series $t \in T$ which matches f, there are at least 90% of them are in the target class c. If there are multiple distance thresholds which satisfy the probability threshold, we pick the largest threshold as the learned threshold. By learning this threshold, we expect that for a query time series q to be classified, if q matches f, q has a probability of at least 90% belonging to class c. Does using class frequency as the expected probability lead to a robust threshold? Let us look at the following example.

Suppose we have a training data set T, which is composed of 11 negative training examples, and 9 positive training examples. We have feature candidate $f = (s, ?, N)$, which is a subsequence of the first training example with a label negative. The target class is the negative class. We compute the best match of f against each training example. The best match distance for each $t \in T$ to s and its class label are shown in Table 5.1. Note that since f is a subsequence of training example $ID = 1$, the best match distance is 0. Table 5.1 is actually the best match vector of f.

In Figure 5.2, we plot the best match distance vector along the horizontal axis. The star represents time series of the negative class and the diamond represents time series of the positive class. As we stated before, we want to learn a threshold δ for $f = (s, ?, N)$ such that

ID	distance	label
1	0	N
2	0.8914	N
3	2.5378	N
4	3.1134	N
5	3.2609	N
6	3.6260	N
7	4.2793	N
8	9.6961	N
9	15.2921	P
10	15.9861	N
11	16.9566	N
12	18.2847	N
13	18.5679	P
14	19.0240	P
15	19.2499	P
16	19.3602	P
17	20.0909	P
18	21.2128	P
19	22.5593	P
20	25.8332	P

Table 5.1: The example data set

for the training time series $t \in T$ which matches f, there are at least 90% of them are in the negative class. In another word, on the training data set, we want to learn a threshold for f to make its accuracy $\geq 90\%$. We pick the largest threshold which satisfies this accuracy as the learned threshold. In Figure 5.2, the learned threshold is the intersection of the horizontal axis and the solid vertical line and the accuracy is 91.67%. We can see that the threshold we learned actually cuts into a dense region of the non-target class (positive class) which makes it not a robust threshold for classifying unseen time series. When we learn the threshold, we only count the training examples in different classes but do not utilize the distance distribution. The actual distances in the best distance vector are not considered in computing the threshold but only the order of time series sorted by the distance is used.

To learn a more robust threshold for a feature candidate $(s, ?, c)$, we propose a KDE based method which takes the distance distribution in the best match vector into consideration. We first separate the best match vector V into two parts, the target class and the non-target class. The target class contains time series in class c and the non-target class contains time

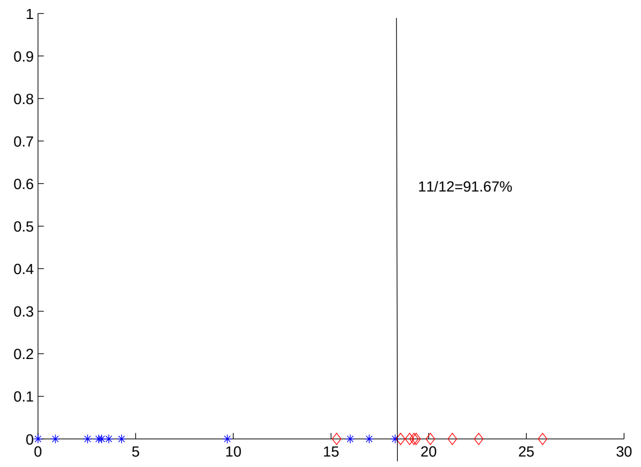


Figure 5.2: An example of learning distance threshold by class frequency.

series which do not belong to class c and may be composed by multiple classes. We then estimate the kernel density for the target class and the non-target class separately. At last, we estimate the class probabilities of any time series given its best match distance to s by Equation 5.4. Given the class probabilities, we can directly learn the distance threshold of $(s, ?, c)$ according to Definition 7.

In the following, we will illustrate the KDE threshold learning through the same example in Table 5.1. In Figure 5.3, we plot the distance vector along the horizontal axis. The dotted curve is the estimated density function, $\hat{f}_{negative}(x)$, for the target class (negative class) and the dashed curve is the estimated kernel density $\hat{f}_{positive}(x)$ for the non-target class (positive class). The solid curve is the estimated probabilities of samples belonging to the target class. The vertical solid lines are several distance thresholds learned by using different probability thresholds. We can see that when the probability threshold is high, the learned distance threshold protects a robust region of the negative class. But when the probability is not high enough, like only 50%, the learned distance threshold moves into the region where the two classes are mixed.

Formally, we summarize the KDE distinctive threshold as follow,

Definition 10 (Learning distinctive features by KDE) *Given a feature candidate* $(s, ?, c)$

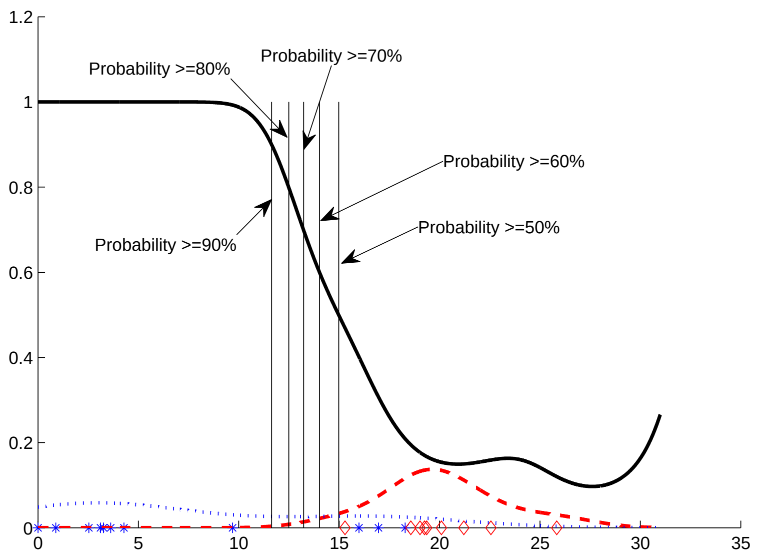


Figure 5.3: An example of learning distance threshold by KDE.

and its best match vector V, we learn a distinctive threshold δ of (s, δ, c), by satisfying $\forall x$, where $0 \leq x \leq \delta$, we have $P(C(x) = c|X = x) \geq \beta$ where β is a user defined probability threshold. $P(C(x) = c|X = x)$ is obtained by kernel density estimation based on V. ■

In the above definition, if x is not a value in the best match vector V, it can be viewed as a virtual time series with best match distance to s as x. There are various way to sample values in the range of vector V to learn the threshold. In our method, we sort the values in $V = \langle d_1, d_2, ..., d_N \rangle$ in ascending order, and then compute the target class probability of d_i in the ascending order. When we find the first breaking value, d_i, which does not satisfy the probability threshold β, we know that the distance threshold δ lies in the interval of $[d_(i-1), d_i]$. Although we can use the breaking value as the distance threshold, to learn a more refined threshold, we can generate a set of extra samples which uniformly distributed on the interval $[v_(i-1), v_i]$ and find the breaking value as the distance threshold. If the learned threshold $\delta = 0$, this feature will not be considered.

In the worst case, when the class distribution is extremely unbalanced, to compute the distance threshold, we may need to estimate the densities for almost all the values in V.

Given V, the worst case time complexity of learning the distinctive threshold is $O(N^2)$ where N is the number of training examples.

Learning feature by Chebyshev's ineqaulity

In the following, we propose an alternative way to learn the distinctive threshold with less computational cost. This method is based on estimating the probability distribution of the non-target class through *Chebyshev's inequality* [4].

Let X be random variable with a finite mean μ and finite variance σ^2, then for any real number $k \geq 0$, the one tail Chebyshev's inequality states [4]

$$Pr(|X - \mu| \geq k\sigma) \leq 1/(k^2 + 1) \tag{5.5}$$

In order to learn the distinctive threshold for a feature candidate $(s, ?, c)$, given its best match vector V, we can view the non-target class in V as a random variable and compute its mean and variance. Then, we can compute the range where the non-target class has a low probability to appear by Equation 5.5.

Formally, we summarize the distinctive threshold by Chebyshev's inequality as follow,

Definition 11 (Learning distinctive feature by Chebyshev's inequality) *Given a feature candidate $f = (s, ?, c)$ and its best match vector V, and we use $V_{nonTarget}$ to denote the time series in V in the non-target class, we have the threshold $\delta = Mean(V_{nonTarget}) - k * Variance(V_{nonTarget})$, where k is a parameter. $k \geq 0$ defines that that the probability of a non-target time series matches f is $\leq 1/(k^2 + 1)$.* ■

Using the same example in Table 5.1, in Figure 5.4, we show the different thresholds learned by using Chebyshev's inequality with $k = 2, 3, 4, 5$, respectively. For example, when $k = 3$, the probability of the non-target class appears in the left of the threshold is less than $1/(3^2 + 1) = 10\%$.

For a time series t matching feature $f = (s, \delta, c)$, by Bayesian theorem [48], the probability that t is in the target class is,

$$P(targetClass|t) = \frac{P(t|target)P(target)}{P(t|nonTarget)P(nonTarget) + P(t|target)P(target)} \tag{5.6}$$

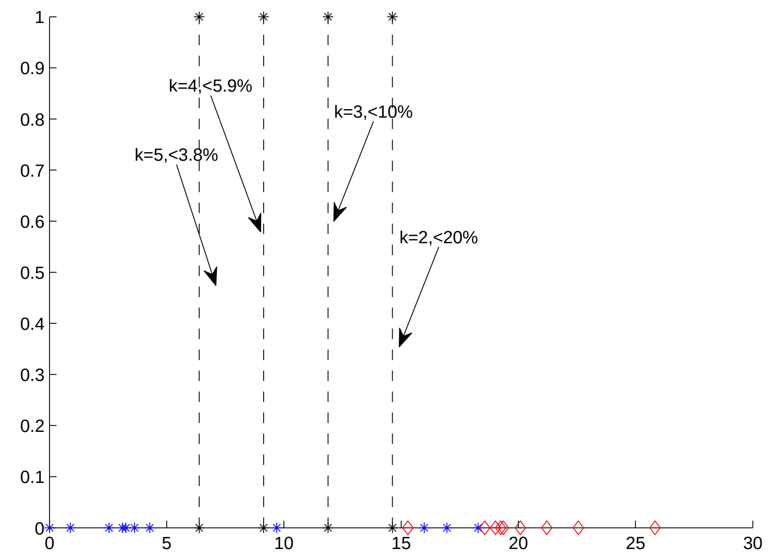


Figure 5.4: An example of learning distance threshold by Chebyshev's inequality.

The threshold learned by Chebyshev's inequality only considers the distance distributions of the non-target class in V. Therefore, it only enforces that $P(t|nonTarget)$ in Equation 5.6 is small but does not guarantee that $P(targetClass|t)$ is high. The learned threshold does not guarantee the feature has a high precision. Furthermore, we do not enforce the feature to be frequent since we focus on learning local distinctive features. In the feature selection step, we further selecting features which are frequent, distinctive and early. The benefit of estimating the probability $P(t|nonTarget)$ through Chebyshev's inequality is to prevent the learned threshold lies in the dense region of the non-target class.

To compute the threshold, we only need to compute the mean of and variance given the best match vector V. The worst case time complexity of learning the threshold is $O(N)$, where N is the number of training examples. This is more efficient than learning the threshold through kernel density estimation, whose cost is $O(N^2)$.

When we compute the threshold for a candidate feature $(s, ?, c)$, we learn the threshold based on its best match vector of s. Why do we use the best match distances instead of considering the distances of all the length $k = |s|$ subsequences in the training data to s? In the following, we refer distances of all the length $k = |s|$ subsequences to s as *all match* *distances*. Actually, the best match distances is a subset of all match distances. Or we can view the best matches distances as a sample from the all match distance space.

Given a feature candidate, for each time series t, among all the length k subsequences, we pick a representative subsequence to represent t. It is the subsequence closest to the feature candidate. Since we want to learn a threshold which separates the target class from the non-target class, we need to know how similar the subsequences in the non-target class are to s. In the non-target class, among all the length k subsequences, the best matches contain the most similar subsequences to s. In another word, in the 1-dimensional all match distance space, the best matches of the non-target class preserve the "left most" class boarder between the target and non-target classes. For learning the threshold, by using the best matches, the learning is biased on the "left most" samples in the non-target class. However, the "left most" samples contain the most important samples close the threshold we want learn.

By using the best matches instead of all matches, we can compute the threshold with less cost. For example, if we have a training data set of N time series of length L, for a feature candidate $(s, ?, c)$ of length k, we have in total $(L-k+1)N$ possible matches. To compute the threshold through KDE, by using the all match distances, the time complexity is $O((L-k+1)^2N^2)$. By using the best matches, the time complexity is $O(N^2)$. To apply the Chebyshev's inequality, the time complexity is $O((L-k+1)N)$ using all matches and is $O(N)$ using the best matches .

Best match vector is also used in [72] for learning the threshold for Shapelets. Note that in the classification step, to classify a time series t online, once we find a match in t to a feature (s, δ, c), which is within the threshold δ, we classify the time series based on this feature. Although the match may not be the best match of t to s given the complete time series, the best match distance must be within the threshold. Therefore, it is safe to classify a time series online once a match to a feature is found.

5.2.2 Feature Selection and a Rule Based Classifier

In the previous section, we discussed how to learn the threshold for a given feature candidate $(s, ?, c)$. In this section, we are going to discuss how to select a good set of features and build a rule based classifier for early classification.

As we discussed before, a good feature for early classification should have three properties, frequent, distinctive and early. Given a distinctive feature $f = (s, \delta, c)$, we propose an utility function in Equation 5.9 to measure the three properties.

The *precision* of $f = (s, \delta, c)$ given its best match vector $V = \langle d_1, d_2, ..., d_N \rangle$ is defined as

$$Precision(f) = \frac{|\{d_i | d_i \leq \delta \wedge label(i) = c\}|}{|\{d_i | d_i \leq \delta\}|} \tag{5.7}$$

The *weighted recall* of $f = (s, \delta, c)$ is defined as

$$WRecall(f) = \frac{\sum_{d_i \leq \delta} 1/\sqrt[\alpha]{BestMatchLength(i)}}{|TargetClass|} \tag{5.8}$$

The *utility measure* of $f = (s, \delta, c)$ is defined as

$$Utility(f) = \frac{2 \times Precision(f) \times WRecall(f)}{Precision(f) + WRecall(f)} \tag{5.9}$$

This utility function extends *f-measure* [22] by taking the earliness of a feature into consideration. Originally, the f-measure considers the precision and the recall of a feature. The precision describes how distinctive a feature is and the recall reflects how frequent a feature is. In order to consider the earliness, for each time series s, we use its best match length with respect to f to weight the contribution of s to the recall of f. We use a parameter $\alpha \geq 1$ to determine the relative importance of earliness. utility measure carries the same spirit as the utility measure proposed in Chapter 3.

Since we consider all the subsequences in the training data as candidate features, it leads to a large set of features which may have a great redundancy. Here, the redundancy means that features who represent the same set of training examples. The redundancy exists among features from different time series and lies in features which come from the same time series. For examples, a group of similar time series subsequences of the same label from different time series may represent each other with great overlaps. For a distinctive feature, its subsequences or super-sequences may be redundant to it.

In order to obtain good interpretability and reduce the number of features for efficient classification, we want to select a non-redundant set of features. Furthermore, as we analyzed, in the process of learning the threshold, we do not consider the earliness and the

```
Algorithm: Feature Selection
Input: $F^{'}$,
Output: $F$
Method:
1:    Let $B$ be the training examples covered by $F^{'}$
2:    $B' = \{\}$
3:    while $B' \neq B$
4:      Find $f \in F^{'}$ with the highest utility measure (Equation 5.9)
5:      If $f$ covers new training examples not in B'
6:        Add f to $F$ and remove it from $F^{'}$
7:        Update $B'$
8:      else
9:        Remove $f$ from $F^{'}$
10:     end if
11:   end while
```

Figure 5.5: Feature selection.

frequency of the features but only focus on the distinctiveness. In the feature selection process, we also want to prune features which do not have a good early classification utility.

In Figure 5.5, we propose an algorithm to select a non-redundant subset of features with overall good early classification utilities. It ranks the features based on its utilities, and selects features which can cover new training examples which are not covered by the features with higher utilities.

After feature selection, we build a simple rule based classifier. For each feature, we build a classification rule. For a time series t to be classified,

$$\text{If t matches}(s, \delta, c) ==> ClassLabel(t) = c$$

If some time series cannot be classified after a required length, a default rule is used to classify it. In our experiment, we set the default rule as the majority class, where

$$\text{Default Rule} : t ==> \text{Majortiy Class}$$

Generally, the default rule can be any time series classifier.

```
Algorithm: Learning EDSC
Input: a training data set T; MinL and MaxL;
Output: A rule set;
Method:
1:     $F = \{\}$
2:     for each class $c$
3:     $F_c = \{\}$
4:       for k=MinL:MaxL
5:         for each length k subsequence $s$ in $T_c$
6:           Compute the best match vector
7:           Learn the threshold
8:           if $threshold > 0$
9:           Add feature to $F_c$
10:          end if
11:        end for
12:      end for
13:      $F_c = FeatureSelection(F_c)$
14:      $F = F \cup F_c$
15:    end for
16:    Build the rules from $F$ and add the default rule
```

Figure 5.6: Learning EDSC.

In Figure 5.6, we summarize the framework of learning the EDSC classifier. The $MinL$ is the minimum length of a feature and $MaxL$ is the maximal length of a feature. In Figure 5.7, we describe the algorithm of online classification using EDSC.

What is the time complexity of building the classifier? For the feature learning step, we consider all the subsequences satisfying the $minL$ and $maxL$ requirements as the candidate features. For each candidate feature, we need to compute its best match vector and then learn the threshold. Suppose we have N time series in the training data set and each time series of average length L. The number of length k sub-sequences is $(L-k+1)*N$. For a length k feature candidate, to compute its best match vector against the training data set is $k*(L-k+1)*N$. To compute the best match vectors of all length k feature candidates, we need $O(k(L-k+1)^2N^2)$. For all the feature candidate between length 1 to L, the time complexity to compute the best match vectors is $\sum_{k=1}^{L} k(L-k+1)^2N^2 = O(N^2L^4)$.

Algorithm: Classification
Input: Rule Set R; a time series t to be classified
Output: class label of t;
Method:

```
1:  While the $i$th time point arrives
2:    for each rule $r$ in $R$ except for the default rule
3:      if $t[1:i]$ has a match with $r$
4:        classify $t$ using rule $r$;
5:        return
6:      end if
7:    end for
8:  end while
9:  if no new time point arrives
10:   classify $t$ by default rule and return
11: end if
```

Figure 5.7: Classification.

Given a best match vector, to learn the threshold, we need either $O(N)$ for the Chebyshev method or $O(N^2)$ for the KDE method. For all the feature candidates between length 1 to L, given their feature vectors, the time complexity to learn the threshold is $\sum_{k=1}^{L}(L-k+1)N^2 = O(N^2L^2)$ for the Chebyshev method and $O(N^3L^2)$ for the KDE method. In practice, for the KDE method, we often cannot reach this worst case complexity since we do not need to estimate the densities for all the training examples. We sort the best match vector and estimate the densities of the training examples in the sorted order until we found the breaking point, where the probability drops below the required probability threshold.

For the feature selection step, the complexity is bounded by sorting all the possible features which is $O(L^2Nlog(NL))$.

The computational cost of EDSC comes from two aspects. First, we consider a large number of feature candidates. Although this can be solved by using various heuristics to reduce the number of candidate features, our current method focuses on an exact approach for feature selection. We choose the exact approach to better evaluate the effectiveness of the proposed early classification framework. In the future, approximate solutions can be investigated. Second, same as [72], the computational bottleneck comes from computing

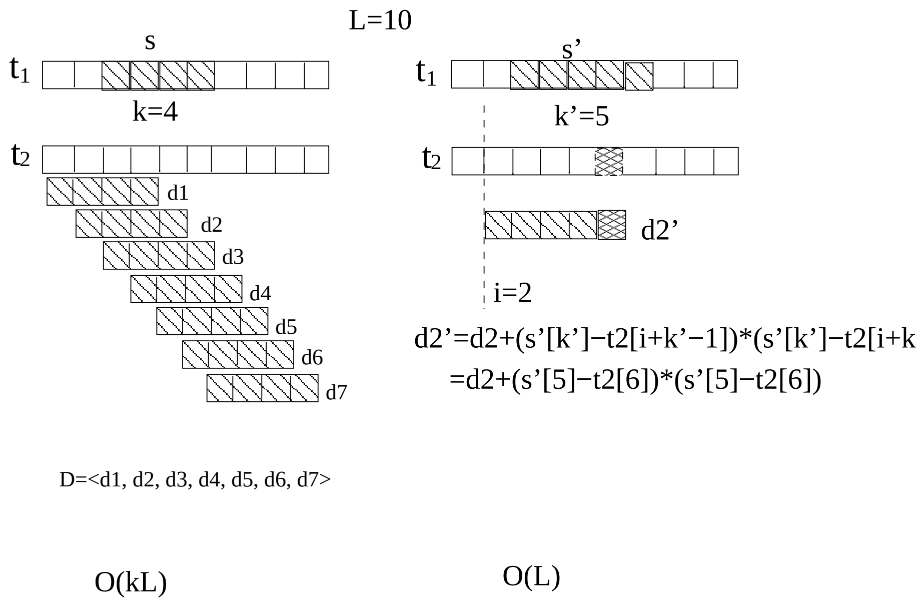

L=10
s
t1
k=4
t2
d1
d2
d3
d4
d5
d6
d7
D=<d1, d2, d3, d4, d5, d6, d7>
O(kL)
s'
t1
k'=5
t2
d2'
i=2
d2'=d2+(s'[k']−t2[i+k'−1])*(s'[k']−t2[i+k
=d2+(s'[5]−t2[6])*(s'[5]−t2[6])
O(L)

Figure 5.8: Share computing .

the best match vectors.

In the following section, we will discuss an improvement on computing the best match vectors and on feature selection.

5.2.3 Improving the Efficiency

When computing the best match vectors, actually, we can share the computation between different candidate features instead of computing it for each feature from scratch.

Given a time series t, for the subsequences of t starting from the same position, we can use them as a group to compute their best match vectors and share computing between them. In Figure 5.8, we illustrate the idea of sharing computation by storing additional information of the matching distances.

Suppose we have a time series t_1 of length L, and we use its subsequence s and s' as two feature candidates. The length of s is 4 and the length of s' is 5. s and s' start from the same position in t_1. In another word, s is a prefix of s' and s' contains one more time point at the end. When we compute the best match distance of s against another length L time series t_2, we need to compute the Euclidean distance of s with every sliding window in t_2 of length $|s|$, and the time complexity is $O(L|s|)$. We store all the squared Euclidean distances between s and each sliding window in t_2 in a vector D. when we compute s''s best match distance with t_2, we only need $O(L)$ instead of $O(L * |s'|)$ by using D. This is because to obtain s''s Euclidean distance with one length 5 sliding window of t_2, we only need to update the distance of last time point of s' to the last time point in a sliding window.

In Figure 5.9, we group features candidates generated from the same time series of the same starting position and compute a feature's best match vector by incrementally updating the stored sliding window distance array of its prefix feature. The updating matrix algorithm is listed in Figure 5.10. By using the above method, to compute the best match vectors for all features from length 1 to L, the time complexity is reduced to $O(N^2L^3)$ from $O(N^2L^4)$.

In Figure 5.11, we present the improved learning framework. The other improvement in this framework is on the feature selection. We do two rounds of feature selection. In the first round, we do feature selection for the features generated from the same time series (Line 7). In another world, instead of doing feature selection on all possible features, we divide the the set of possible features into several subsets and do feature selection on each of them. Each subset contains all the features generated from the same time series. In the second round, we do feature selection on the union of selected features in the first round (Line 9). By doing feature selection in two rounds, we can save space. We do not need to store all the possible features and then perform feature selection. We also reduce the time complexity. The time complexity is bounded by sorting the features generated from each time series, which is $O(NL^2 logL)$. This is more efficient than doing one round feature selection on all possible features, which is $O(NL^2 log(NL))$. Selecting features in two rounds by breaking down into several subsets generates the same result as selecting features on whole set. This is proved in Lemma 2.

Lemma 2 *For a set F of N features and a group of m exclusive subsets of F, F_1, F_2,...,F_m, where $F_1 \cup F_2, ..., \cup F_m = F$ we have,*
FeatureSelection(F) =FeatureSelection($\bigcup_{i=1}^{m}$FeatureSelection(F_i))

Proof. *Suppose we select a subset of features from F using the feature selection algorithm, and we denote the result set as $\widehat{F}$. We also do feature selection on each subset F_i and denote*

```
Algorithm: LFOT(Learn Features for One Training Time Series)
Input: a training data set T of length $L$; $t_i \in T$; MinL and MaxL;
Output: A set of feature $F = \{f | f = (s, \delta)\}$;
Method:
1:    $F = \{\}$
2:    for each $startP \leq L + 1 - MinL$
3:      $s = (t_i, startP, MinL)$
        (Note: $s$ is the length $MinL$ subsequence of $t_i$ starting from position $startP$)
4:      Initialize sliding window distance array $M$ for $s$
5:      Compute the best match vector $V_s$ for $s$
6:      Learn feature threshold and add to $F$
7:      for $MinL < l \leq min(L + l - startP, MaxL)$
8:        $s = (t_i, startP, l)$
9:        $V_s$=UpdateM(s)
10:       Learn feature threshold and add to $F$
11:     end for
12:   end for
```

Figure 5.9: LFOT.

the result set as $\widehat{F_i}$. We first prove that for any $\widehat{f} \in \widehat{F}$, there must exist one $\widehat{F_i}$, such that $\widehat{f} \in \widehat{F_i}$. Since if $\widehat{f}$ is not in any $\widehat{F_i}$, there must be another feature $\overline{f}$ which dominates $\widehat{f}$, which means $\overline{f}$ has a better utility score then $\widehat{f}$ and covers all the training examples which are covered by $\widehat{f}$. In this case, $\widehat{f}$ cannot appear in $\widehat{F}$ which contradicts with $\widehat{f} \in \widehat{F}$. It means that $\widehat{F}$ is a subset of $\bigcup_{i=1}^{m} \widehat{F_i}$. Then, we do second round feature selection on $\bigcup_{i=1}^{m} \widehat{F_i}$. We can know that the selected set on $\bigcup_{i=1}^{m} \widehat{F_i}$ is the same as $\widehat{F}$ since the features in $\widehat{F}$ dominate any other features in $\bigcup_{i=1}^{m} \widehat{F_i}$.

■.

5.3 Experiments

We evaluate our methods on seven data sets from UCR time series archive [30]. For comparisons, we conducted the experiments on the same data sets as in Chapter 4.

All the experimental results are obtained by using a PC computer with and AMD 2.2GHz

```
Algorithm: UpdateM
Input: M; T; $t_q \in T$; $MinL$ and $MaxL$; $s = (t_q, startP, l)$
Output: V
Method:
1:  for each row $i$ in $M$
2:      for each column $j = 1 : L - l + 1$ in $M$
3:          $M[i][j] = M[i][j] + (s[l] - t_i[j + l - 1])^2$
4:      end for
5:      $V[i] = sqrt(min(M[i]);$
6:  end for
```

Figure 5.10: Update M.

```
Algorithm: Improved Learning Framework
Input: a training data set T; MinL and MaxL;
Output: A set of feature $F = \{f | f = (s, \delta)\}$;
Method:
1:  $F = \{\}$
2:  for each class $C_i$
3:      $F_c = \{\}$
4:      for $t_i \in C_i$
5:          $F' = \{\};$
6:          $F'$=LFOT($t_i$)
7:          $F_c = F_c \cup FeatureReduction(F')$
8:      end for
9:      $F_c = FeatureReduction(F_c)$
10:     $F = F \cup F_c$
11: end for
```

Figure 5.11: The improved learning framework.

CPU and 3GB main memory. The algorithms were implemented in C++ using Microsoft Visual Studio 2005.

5.3.1 Results Overview

The results of the seven data sets are listed in Table 5.2(ECG), Table 5.3(Gun Point), Table 5.4(CBF), Table 5.5(Synthetic Control), Table 5.6(Wafer), Table 5.7(OliveOil), and Table 5.8(Two Patterns).

For each dataset, in its corresponding table, we describe the size, dimension, and number of classes for the training data set and testing data set in the first row. We compare four methods for each data set, EDSC-CHE (EDSC with Chebyshev's inequality threshold learning), EDSC-KDE (EDSC with the KDE threshold learning), 1NN-Full (Full length 1NN with Euclidean distance) and ECTS (Early classifier for time series proposed in Chapter 4). For the EDSC-CHE and EDSC-KDE classifiers, as we described before, the classifiers contain a default rule which is the majority class. Without using the default rule, the EDSC-CHE and EDSC-KDE may not cover all the time series to be classified. In the result tables, we also report the accuracy and coverage rate for the EDSC classifier without using the default rules, and they are referred as EDSC-CHE-NDefault and EDSC-KDE-NDefault.

All the results are obtained using the same parameter settings. For the EDSC-CHE method, we use the parameters as $MinLen = 5; MaxLen = L/2; k = 3$, and for the EDSC-KDE method, we use the parameters as $MinLen = 5; MaxLen = L/2; p = 95\%$ where L is the length of the full length time series.

In Figure 5.12, we summarize the performance of EDSC-CHE, EDSC-KDE, ECTS and full 1NN in terms of classification accuracy and average prediction length. From Figure 5.12, we can see that, in terms of earliness, EDSC-CHE and EDSC-KDE are always earlier than ECTS, and sometimes, significantly earlier, such as on the ECG, CBF and Synthetic control data sets. EDSC-CHE and EDSC-KDE have similar performance in terms of earliness. For the classification accuracy, on Gun-Point data set and CBF data, EDSC-CHE and EDSC-KDE are more accurate than FUll-1NN and ECTS. On Synthetic control and ECG data set, EDSC-KDE has a better accuracy than FULL-1NN. On the remaining three data sets, the EDSC-KDE and EDSC-CHE methods do not beat the accuracies of full length 1NN,

ECG: 2 classes; 100 training inst.; 100 testing inst.; L=96				
EDSC-CHE: $MinLen = 5; MaxLen = L/2; k = 3$ EDSC-KDE: $MinLen = 5; MaxLen = L/2; p = 95\%$				
	Accuracy	Coverage	Ave.Length	No. Features
EDSC-CHE-NDefault	86.67%	90%	16.87/96	32
EDSC-CHE	82%	100%	24.78/96	32
EDSC-KDE-NDefault	90.70%	86%	20.34/96	29
EDSC-KDE	88%	100%	30.93/96	29
1NN-FUll	88%	100%	96/96	N/A
ECTS	0.89%	100%	57.71/96	N/A

Table 5.2: Results of ECG data set

Gun-Point: 2 classes; 50 training inst.; 150 testing inst.; L=150				
EDSC-CHE: $MinLen = 5; MaxLen = L/2; k = 3$ EDSC-KDE: $MinLen = 5; MaxLen = L/2; p = 95\%$				
	Accuracy	Coverage	Ave.Length	No. Features
EDSC-CHE-NDefault	97.18%	94.67%	64.75/150	8
EDSC-CHE	94.67%	100%	69.3/150	8
EDSC-KDE-NDefault	95.74%	94%	64.80/150	9
EDSC-KDE	94%	100%	69.97/150	9
Shapelets [72]	93.3%	100%	150/150	1
1NN-FUll	91.33%	100%	150/150	N/A
ECTS	86.67%	100%	70.39/150	N/A

Table 5.3: Results of Gun-Point data set

CBF: 3 classes; 30 training inst.; 900 testing inst.; L=128				
EDSC-CHE: $MinLen = 5; MaxLen = L/2; k = 3$ EDSC-KDE: $MinLen = 5; MaxLen = L/2; p = 95\%$				
	Accuracy	Coverage	Ave.Length	No. Features
EDSC-CHE-NDefault	95.03%	89.44%	35.03/128	3
EDSC-CHE	87.89%	100%	44.84/128	3
EDSC-KDE-NDefault	94.94%	87.78%	35.12/128	3
EDSC-KDE	85.89%	100%	46.47/128	3
1NN-FUll	85.2%	100%	128/128	N/A
ECTS	85.2%	100%	91.73/128	N/A

Table 5.4: Results of CBF data set

Synthetic Control:6 classes; 300 training inst.; 300 testing inst.; L=60				
EDSC-CHE: $MinLen = 5; MaxLen = L/2; k = 3$ EDSC-KDE: $MinLen = 5; MaxLen = L/2; p = 95\%$				
	Accuracy	Coverage	Ave.Length	No. Features
EDSC-CHE-NDefault	94.49%	90.67%	30.62/60	38
EDSC-CHE	87.66%	100%	33.36/60	38
EDSC-KDE-NDefault	97.06%	90.67%	30.43/60	39
EDSC-KDE	0.9033	1	33.19/60	39
1NN-FUll	88%	100%	60/60	N/A
ECTS	89%	100%	53.98/60	N/A

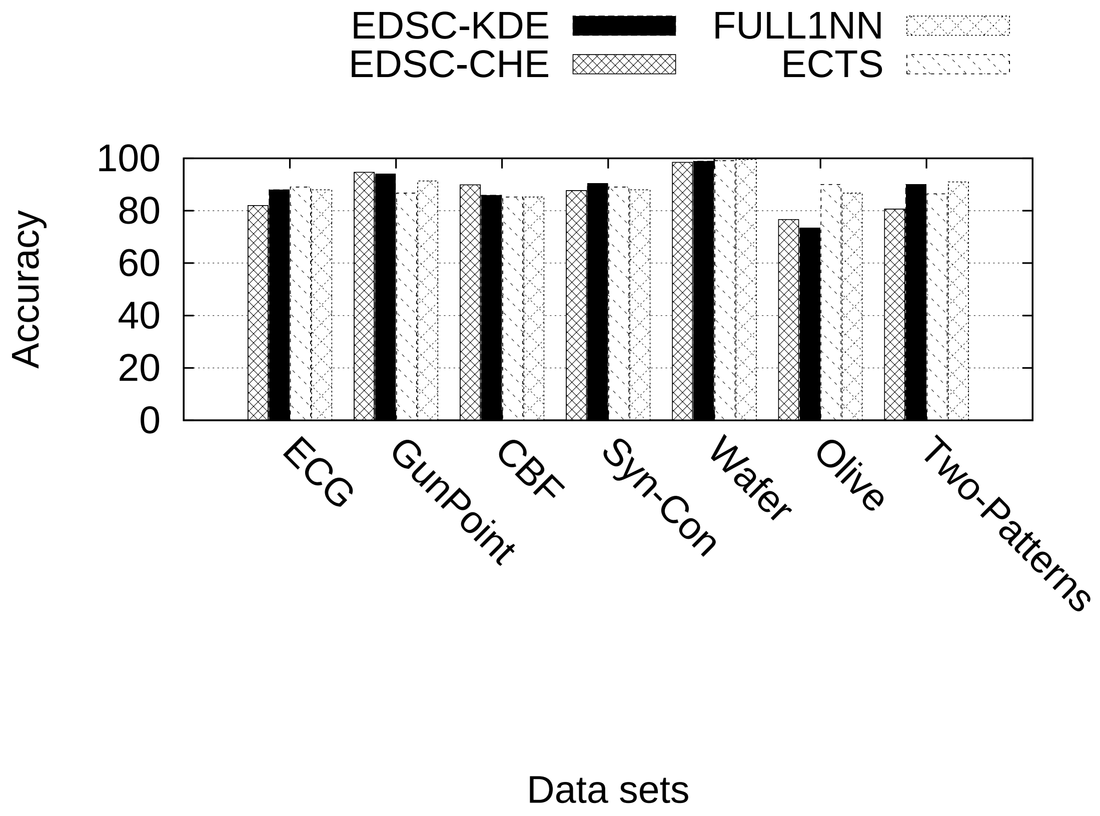
Table 5.5: Results of Synthetic Control data set

Wafer: 2 classes; 1000 training inst.; 6174 testing inst.; L=152				
EDSC-CHE: $MinLen = 5; MaxLen = L/2; k = 3$ EDSC-KDE: $MinLen = 5; MaxLen = L/2; p = 95\%$				
	Accuracy	Coverage	Ave.Length	No. Features
EDSC-CHE-NDefault	98.82%	99.51%	41.39/152	62
EDSC-CHE	98.49%	100%	41.93/152	62
EDSC-KDE-NDefault	99.15%	99.51%	38.42/152	52
EDSC-KDE	98.87%	100%	38.97/152	52
1NN-FUll	99.55%	100%	152/152	N/A
ECTS	99.08%	100%	67.39/152	N/A

Table 5.6: Results of Wafer data set

OliveOil: 4 classes; 30 training inst.; 30 testing inst.; L=570				
EDSC-CHE: $MinLen = 5; MaxLen = L/2; k = 3$ EDSC-KDE: $MinLen = 5; MaxLen = L/2; p = 95\%$				
	Accuracy	Coverage	Ave.Length	No. Features
EDSC-CHE-NDefault	85.16%	90%	174.07/570	13
EDSC-CHE	76.67%	100%	213.48/570	13
EDSC-KDE-NDefault	95.45%	73.33%	223.18/570	13
EDSC-KDE	73.33%	100%	315.67/570	13
1NN-FUll	86.7%	100%	570/570	N/A
ECTS	90%	100%	497.83 /570	N/A

Table 5.7: Results of OliveOil data set



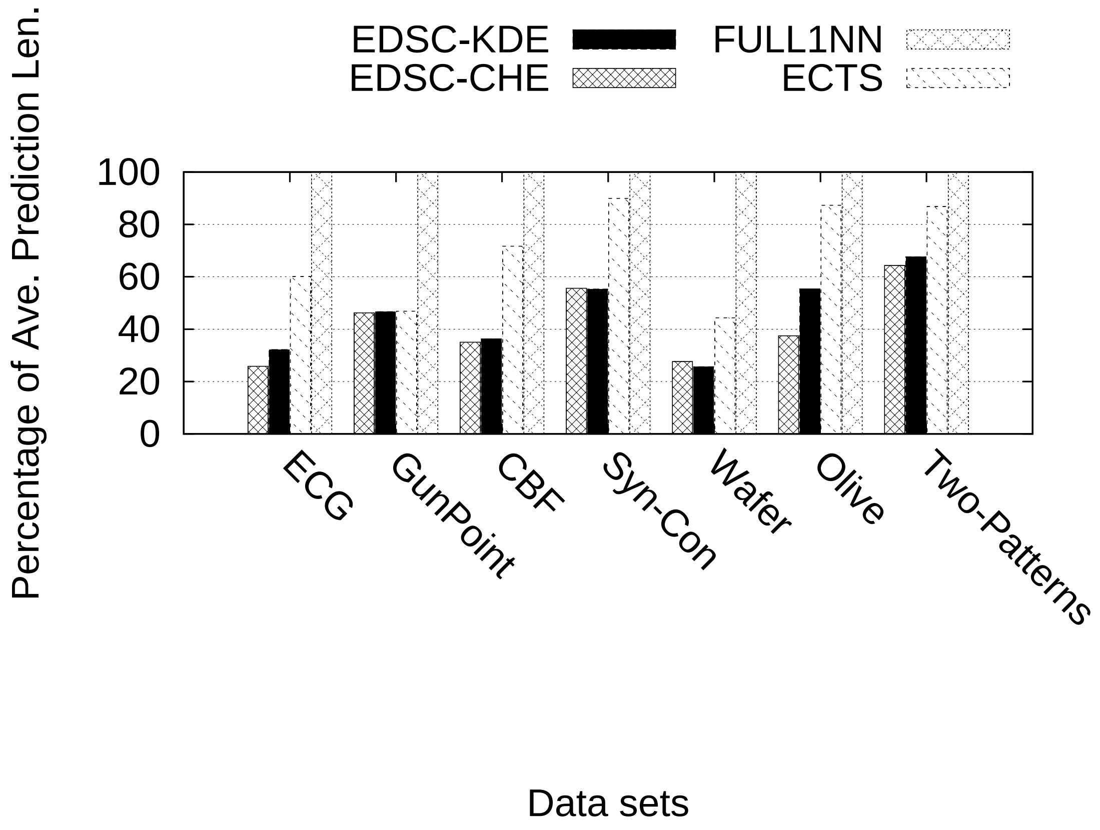


Figure 5.12: Comparison among EDSC, ECTS and Full-1NN

Two Patterns: 4 classes; 1000 training inst.; 4000 testing inst.; L=128				
EDSC-CHE: $MinLen = 5; MaxLen = L/2; k = 3$ EDSC-KDE: $MinLen = 5; MaxLen = L/2; p = 95\%$				
	Accuracy	Coverage	Ave.Length	No. Features
EDSC-CHE-NDefault	84.75%	93.75%	79.29/128	305
EDSC-CHE	80.6%	100%	82.33/128	305
EDSC-KDE-NDefault	94.94%	93.33%	83.62/128	279
EDSC-KDE	90%	100%	86.58/128	279
1NN-FUll	91%	100%	128	N/A
ECTS	86.48%	100%	111.10/128	N/A

Table 5.8: Results of Two Patterns data set

ECG	EDSC-CHE (length), EDSC-KDE(combination), ECTS(accuracy)
GunPoint	EDSC-CHE(length and accuracy)
CBF	EDSC-CHE (length and accuracy)
Syn-Control	EDSC-KDE (length and accuracy)
Wafer	EDSC-KDE (length), Full-Length 1NN (accuracy), ECTS
OliveOil	EDSC-CHE (length), ECTS (Accuracy)
TwoPatterns	EDSC-CHE(length), 1NN-Full(Accuracy), EDSC-KDE(combination)

Table 5.9: Dominating classifiers

but with similar accuracies except for the OliveOil data set. Generally, EDSC-KDE is more accurate than the EDSC-CHE. It is because the threshold learning quality of the EDSC-CHE is not good as EDSC-KDE in order to gain a faster computation.

On a data set, if classifier A has a shorter or equal prediction length and a higher or equal accuracy than classifier B, we say classifier A dominates classifier B on this data set. In Table 5.9, for each data set, we list the classifiers which are not dominated by any other classifiers. From Table 5.9 we can see that, on the five data sets, EDSC-CHE is among the dominating classifiers, mainly because its earliness. On the four data sets, EDSC-KDE is among the dominating classifiers. Interestingly, EDSC-CHE or EDSC-KDE is always among the dominating classifiers if not both. When either EDSC-CHE or EDSC-KDE is not among the dominating classifiers, they are dominated by each other, but never by ECTS and FUll-length 1NN classifier.

From the above analysis, we can see EDSC-CHE and EDSC-KDE can achieve competitive classification accuracies with great earliness.

5.3.2 Interpretability of Features

For the rule based classifier, besides a classification result, we also want to understand the features extracted by the classifier. In this section, we are going to exam the interpretability of the learned features.

CBF data set has three classes, named cylinder, bell, and funnel. In Figure 5.13, we plots the profiles of the three classes of the CBF data set in the left column. Both the EDSC-CHE method and EDSC-KDE methods extract 3 features for the CBF data set, 1 feature for each class. The features extracted by the two methods are quite similar. We use the features of the EDSC-CHE as examples. In the right column of Figure 5.13, the features of each class extracted by the EDSC-CHE are plotted. The features are the bold region in the time series it comes from. By observing the three classes, we can see the three classes are quite similar at the beginning. For example, in the interval between $[0 - 20]$ time points, the three classes mix together in the range of -2 to 0 along the vertical axis. When the time series grows longer, the features for each class start to distinguish different classes from each other. The features learned by EDSC-CHE well represent the characteristics of each class and they lie in the early phases of the time series when the classes start to separate from each other.

Let us take the Gun-Point data set as another example. The Gun-Point data set contains two classes, the Gun-Draw class and the Point class [55]. In Figure 5.14, the profile of the Gun-Draw class (left) and the Point class (right) are plotted in the first row. For the Gun-Draw class, the actors "draw a replicate gun from a hip-mounted holster, point it at a target for approximately one second, then return the gun to the holster [55]". For the Point class, "The actors have their hands by their sides. They point with their index fingers to a target for approximately one second, and then return their hands to their sides" [55]. The Gun-Draw class is different from the Point class by two actions, "draw a gun from a holster", and "return the gun to the holster" [55].

Using the EDSC-CHE, we learned 4 features for each class. In Figure 5.14, in the second row, we plot the features with the highest utility score for each class respectively. We can see that, for the GUN-Draw class, the feature captures the region of "draw a gun from the holster". It is the action to distinguish the two classes and it is an earlier feature than "return the gun to the holster". For the Point class, the feature happens to belong to a noisy signal. But actually, by plotting the best matches of other time series for this feature in bold (in the third row), we can see the feature represents the later moment of the "lifting the arm". The top features learned by the EDSC-KDE method are quite similar to the two features plotted.

The above two examples on CBF and Gun-Point data sets demonstrate that the features learned by our methods can capture the early characteristics of different classes.

5.3.3 Sensitivity of Parameters

The previous results are all generated by using $k = 3$ for the EDSC-CHE and using $p = 95\%$ for the EDSC-KDE methods. In learning the threshold, parameter k (in Definition 11) and p (in Definition 10) correspond to the degree of distinctiveness of learned features. In the following, we use the ECG data set as an example to show the effects of changing the parameters. The results on the ECG data with different values of p and k are shown in Figure 5.15 and Figure 5.16, respectively.

We can see for the EDSC-KDE, when we increase p, the features we learned should be more distinctive and the accuracy classified by the features should increase. In Figure 5.15, we see that the accuracies of EDSC-KDE-NDefault generally increase from 80% to 91% in the range of $p = 50\%$ to $p = 95\%$ and decrease after $p = 99\%$. When we set $p = 100\%$, no feature satisfies this probability threshold and the accuracy of EDSC-KDE-NDefault is 0. The trends show that when we increase p until a very high value, the features we learned are more accurate in classification. Since when p is higher, the distance threshold protects a safer region for the target class which has a larger margin to non-target class.

But when $p = 99\%$, the accuracy of EDSC-KDE-NDefault is lower than the accuracy at $p = 95\%$. By examining the features, we found that when $p = 99\%$, some features with very small recalls are used in classification and lead to wrong classification. Setting a very high threshold $p = 99\%$ may lead to selecting some features with very small recalls which are actually over-fitted features.

On the hand, when we increase p, the coverage rates of the EDSC-KDE-NDefault decrease. Because when we increase P, the threshold learned tends to be smaller. There is a trade-off between the accuracy and coverage when p increases. By using a default classification rule, EDSC-KDE reaches the highest accuracy of 88% when $p = 95\%$ as the best balance between accuracy and coverage. When $p = 90\%$, the EDSC-KDE has a similar accuracy as 87%. On the other data sets, we also observe that when the best results usually appear when setting $90\% \leq p \leq 95\%$, and the results are stable in this range.

When increase p, the earliness of classification is monotonic after $p > 65\%$. It suggests that a lower threshold may lead to a earlier classification.

In Figure 5.16, we present the result for the ECG data set using EDSC-CHE when changing the value of k. We can see when k increases, the accuracy of EDSC-CHE-NDefault increases and the coverage rate decreases. When $k = 2.5$, EDSC-CHE reaches the best accuracy as 85% due to a good balance between the accuracy and coverage of EDSC-CHE-NDefault . On other data sets, we observe that the best results usually appear when $2.5 \leq k \leq 3.5$. For the average prediction length, we observe that it monotonically increases after $k > 1$, which shares a similar behavior as the EDSC-KDE method.

5.3.4 Efficiency

In Table 5.10, we compare the training time using the algorithm in Figure 5.6 (EDSC-CHE(Nai.) and EDSC-KDE(Nai.)) and the improved algorithm in Figure 5.11 (EDSC-CHE(Impr.) and EDSC-KDE(Impr.)). Actually, in EDSC-CHE(Nai.) and EDSC-KDE(Nai.), in computing the best match vectors, we also incorporate the early stopping techniques proposed in [72].

From Table 5.10, we can see that EDSC-CHE is faster than EDSC-KDE. This is due to the different time complexity of learning the distance threshold. The improved algorithms, EDSC-CHE(Nai.) and EDSC-KDE(Nai.), can significantly reduce the training time comparing to the naive algrithms. As we analyzed, this is due to the reduced time complexity on computing best match vectors.

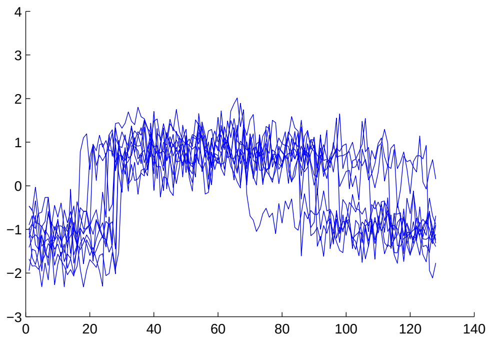
	EDSC-CHE(Nai.)	EDSC-CHE(Impr.)	EDSC-KDE(Nai.)	EDSC-KDE(Impr.)
CBF N=30, L=128	37.93 sec	9.52 sec	41.48 sec	13.05 sec
ECG N=100, L=96	123.42 sec	26.05 sec	137.45 sec	40.17 sec
SynCon N=300, L=60	252.83 sec	62.20 sec	332.60 sec	140.798 sec
GUN Point N=50, L=150	165.13 sec	25.6 sec	170.21 sec	30.38 sec
OliveOil N=30, L=570	6729.26 sec	1707.63 sec	6700. 1 sec	1728.91 sec
TwoPattern N=1000, L=128	37249.4 sec	10876.6 sec	40925.3 sec	14258.1 sec
Wafer N=1000, L=152	73944.4 sec	10841.5 sec	103298 sec	40082.2 sec

Table 5.10: Training time comparison

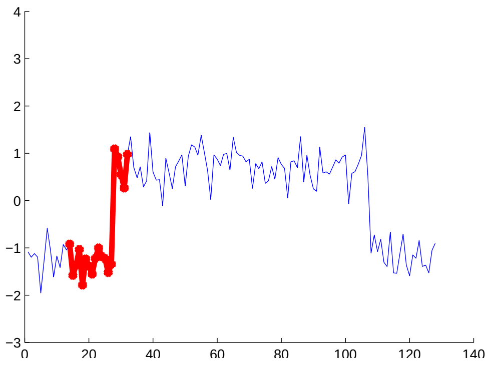
5.4 Summary

In this chapter, we propose an EDSC framework for building a rule based classifier for time series early classification. EDSC can achieve early classification and competitive accuracies with good interpretability. In the future, we are going to investigate different feature selection strategies and explore approximate or heuristic approaches to improve the efficiency of the proposed methods.

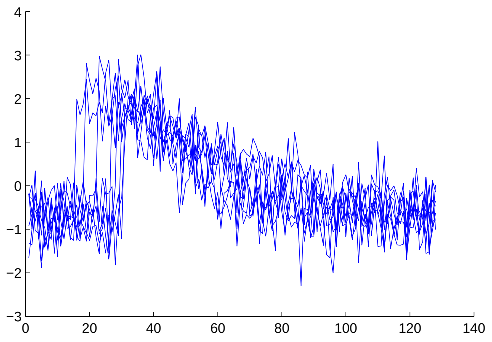
Figure 5.13: Features on the CBF Data Set



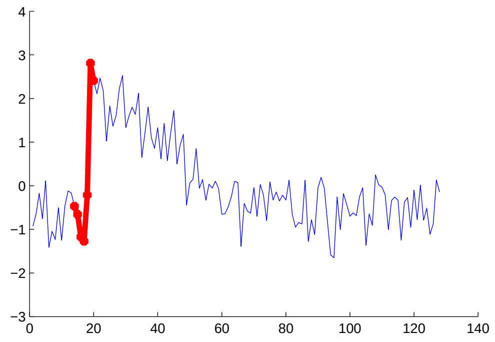
(a) Cylinder class



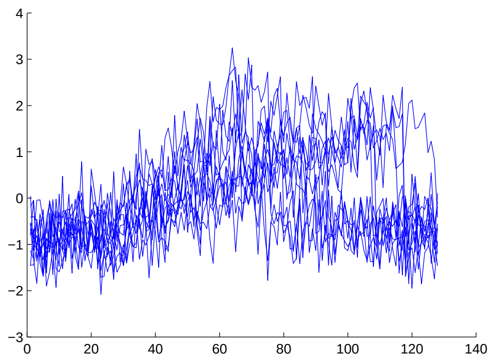
(b) A feature of cylinder class



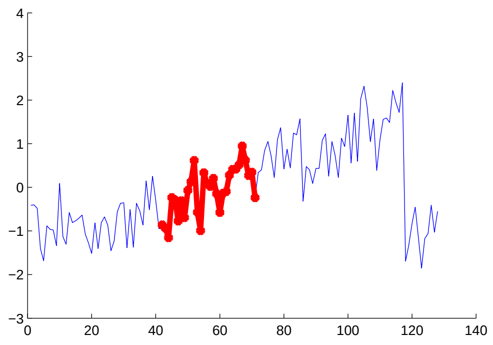
(c) Funnel class



(d) A feature of funnel class

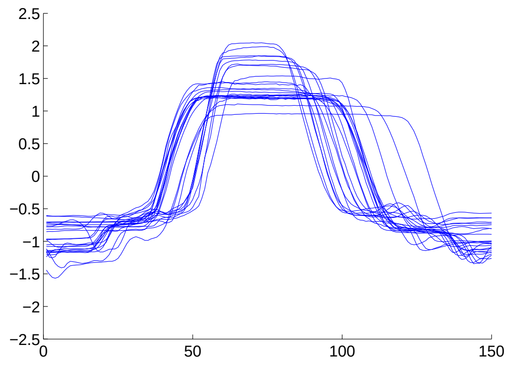


(e) Bell class

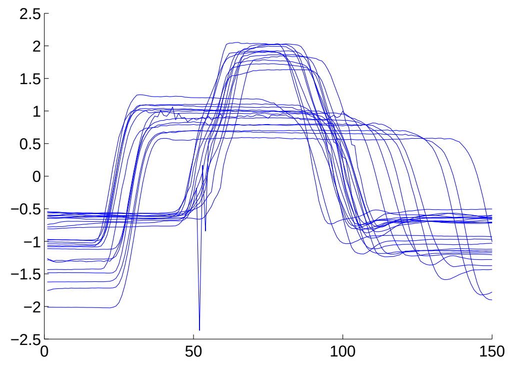


(f) A feature of bell class

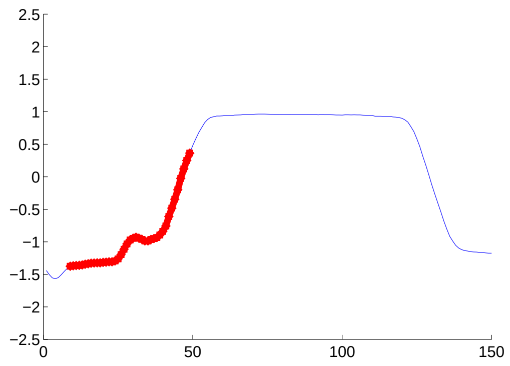
Figure 5.14: Features on the Gun-Point Data Set



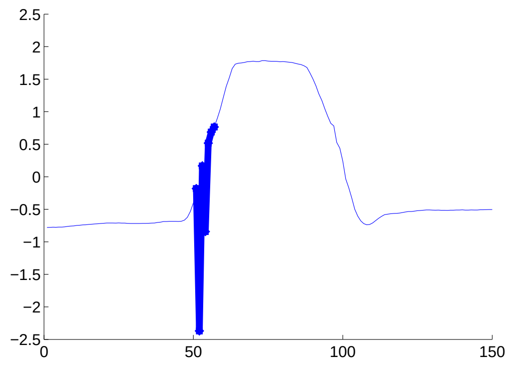
(a) Gun-Draw class



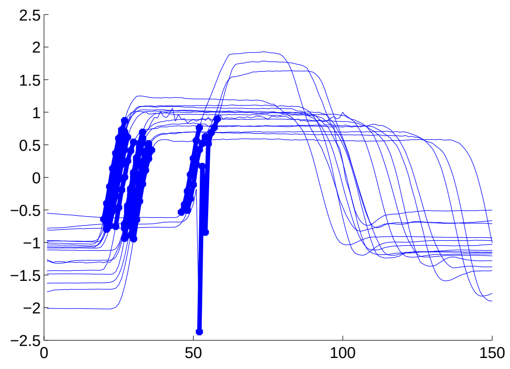
(b) Point class



(c) A feature of gun-draw class



(d) A feature of point class



(e) Best matchings of point class

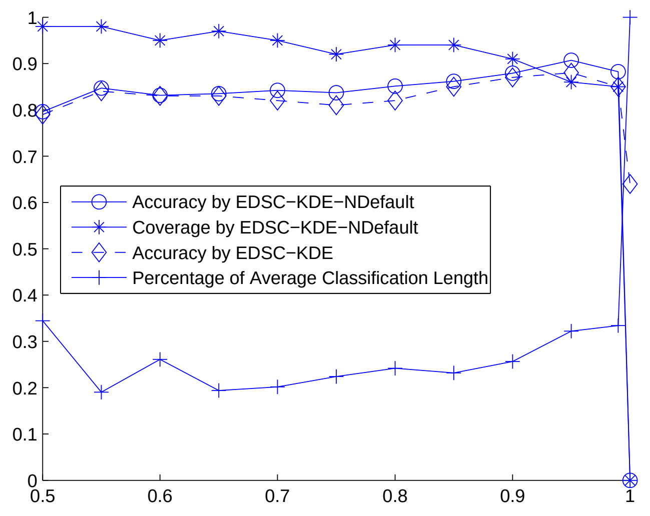


Figure 5.15: Results on ECG data set by varying p

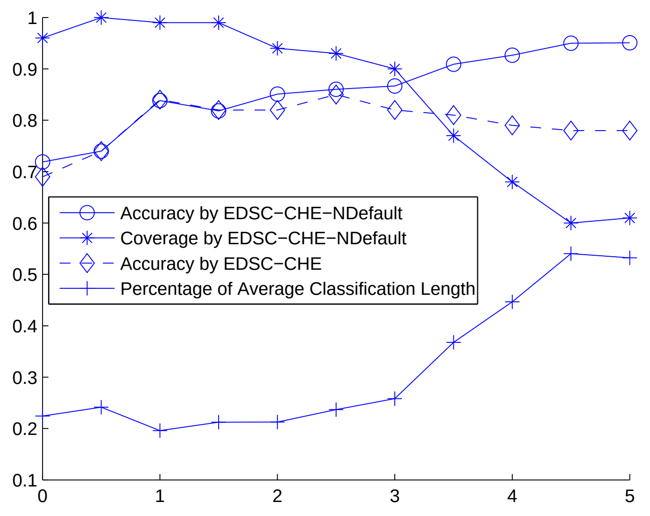


Figure 5.16: Results on ECG data set by varying k

Chapter 6

Conclusions and Future Works

In this thesis, we identify the problem of early classification on temporal sequences and analyze the challenges. We formulate early classification as a problem of monitoring temporal sequences online and predicting class labels as early as possible once classifiers are confident about the predictions. We develop a series of classifiers for temporal sequence early classification. The proposed classifiers are designed for different types of temporal sequences including symbolic sequences (Chapter 3) and time series (Chapter 4 and Chapter 5). Furthermore, the proposed classifiers have several desirable characteristics which are useful in different application scenarios, such as the interpretability. We evaluate our approaches on a broad range of real data sets and demonstrate that the classifiers can achieve competitive classification accuracies with great earliness. Also, some interpretable features are extracted from sequences. The experiments validate that early classification is feasible and useful.

In the future, we are going to explore the following directions.

- For early classification, one of the challenges is to develop scalable algorithms for learning eraly classifiers. Although in this thesis, we proposed various methods to achieve early classification, the efficiencies of those methods need to be improved. For example, for the feature extraction methods we proposed in Chapter 5, the time complexity is $O(N^2L^3)$, where N is the number of sequences and L is the dimensionality of the sequences. As we discussed in Chapter 5, one way to improve the efficiency is to the develop some heuristic methods to eliminate the number of candidate features. The challenges of developing efficient learning algorithms for early classifiers come from the following aspects. First, efficiently extracting features and building classifiers for conventional sequence classification is a challenging task [25, 72]. Furthermore, comparing to learning conventional sequence classifiers, learning early classifiers usually requires extracting and examining a larger number of features since we do not only focus on building an accurate classifier but also comparing features in terms of their utilities for the earliness in classification. For example, for the ECTS classifier we proposed in Chapter 4, its counterpart for the conventional time series classification is the lazy instance based classifier. Without requiring the earliness in classification, instance based classifiers do not have a training step. But in order to achieve early classification, we create the training step which increases the time complexity of training step from none.

- As mentioned in Chapter 2, temporal sequences can be summarized as five types, simple symbolic sequences, complex symbolic sequences, simple time series, multivariate time series and complex event sequences. In this thesis, we focus on the simple symbolic sequences and simple time series. In the future, we are going to explore early classification on multivariate time series, complex symbolic sequences and complex event sequences. We are going to explore the proper ways of transforming complex forms of sequences into simple types for early classification. We are also interested in developing new methods which are designed for specific types of data in different applications.

- For a streaming sequence, the label of the sequence may change. The problem of strong sequence classification [26, 27] is to predict a sequence of labels instead of predicting one class label. In the future, we are going to integrate early classification with strong sequence classification problem. We are interested in learning the earliest time to predict a reliable changing of labels. This prediction is not only based on features in the sequences, but also based the labels of the sequence detected before.

- In early classification and the strong sequence classification process, we may get feedbacks after giving predictions. We may also suggest domain experts to label some sequences which may be useful for early classifying unseen sequences. In the future, we are going to integrate early classification with active learning or semi-supervised learning.

Bibliography

[1] Berkeley drosophila genome project. `http://genomebiology.com/2002/3/12/research/0087.1`.

[2] Time series data library webpage: `http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/`.

[3] C. C. Aggarwal. On effective classification of strings with wavelets. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 163–172, 2002.

[4] A. O. Allen. *Probability, Statistics, and Queuing Theory with Computer Science Applications.* Acadimic Press, Inc., San Diago, CA, 1990.

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipmanl. Basic local alignment search tool. *J.Mol.Biol.*, 215:403–410, 1990.

[6] A. Asuncion and D. Newman. UCI machine learning repository. `http://archive.ics.uci.edu/ml/index.html`, 2007.

[7] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic classification on the fly. *Computer Communication Review*, 36(2):23–26, 2006.

[8] E. Bertino, B. Catania, and E. Caglio. Applying data mining techniques to wafer manufacturing. In *PKDD '99: Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery*, pages 41–50. Springer-Verlag, 1999.

[9] A. Bregón, M. A. Simón, J. J. Rodríguez, C. J. Alonso, B. P. Junquera, and I. Moro. Early fault classification in dynamic systems using case-based reasoning. In *CAEPIA*, pages 211–220, 2005.

[10] B. Y. Cheng, J. G. Carbonell, and J. Klein-Seetharaman. Protein classification based on text document classification techniques. *Proteins*, 1(58):855–970, 2005.

[11] N. A. Chuzhanova, A. J. Jones, and S. Margetts. Feature selection for genetic sequence classification. *Bioinformatics*, 14(2):139–143, 1998.

[12] M. Deshpande and G. Karypis. Evaluation of techniques for classifying biological sequences. In *PAKDD '02: Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 417–431, 2002.

[13] J. J. R. Diez, C. A. González, and H. Boström. Boosting interval based literals: variable length and early classification. *Intell. Data Anal.*, 5(3):245–262, 2001.

[14] C. Ding and X. He. K-nearest-neighbor consistency in data clustering: incorporating local information into global optimization. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 584–589, New York, NY, USA, 2004. ACM.

[15] C. H. Q. Ding and X. He. Cluster aggregate inequality and multi-level hierarchical clustering. In *PKDD*, pages 71–83, 2005.

[16] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. J. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB*, 1(2):1542–1552, 2008.

[17] G. Dong and J. Pei. *Sequence Data Mining*, pages 47–65. Springer US, 2007.

[18] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Chapter 3. Markov Chain and Hidden Markov Model. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, pages 47–65. Cambridge University Press, 1998.

[19] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[20] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[21] M. P. Griffin and J. R. Moorman. Toward the early diagnosis of neonatal sepsis and sepsis-like illness using novel heart rate analysis. *PEDIATRICS*, 107(1):97–104, 2001.

[22] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[23] C. harley and R. Reynolds. Analysis of e.coli promoter sequences. *Nucleic Acides Res.*, 15(5):2343–61.

[24] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning.* Springer, New York, NY, USA, second edition, 2009.

[25] X. Ji, J. Bailey, and G. Dong. Mining minimal distinguishing subsequence patterns with gap constraints. *Knowl. Inf. Syst.*, 11(3):259–286, 2007.

[26] M. W. Kadous. *Temporal classification: extending the classification paradigm to multivariate time series.* PhD thesis, 2002.

[27] M. W. Kadous and C. Sammut. Classification of multivariate time series and structured data using constructive induction. *Machine Learning*, 58(2-3):179–216, 2005.

[28] L. Kaján, A. Kertész-Farkas, D. Franklin, N. Ivanova, A. Kocsor, and S. Pongor. Application of a simple likelihood ratio approximant to protein sequence classification. *Bioinformatics*, 22(23):2865–2869, 2006.

[29] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 102–111, 2002.

[30] E. Keogh, X. Xi, L. Wei, and C. A. Ratanamahatana. The UCR time series classification and clustering homepage: `http://www.cs.ucr.edu/~eamonn/time_series_data/`, 2006.

[31] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 285–289, 2000.

[32] S.-B. Kim, K.-S. Han, H.-C. Rim, and S. H. Myaeng. Some effective techniques for naive bayes text classification. *IEEE Transactions on Knowledge and Data Engineering*, 18(11):1457–1466, Nov. 2006.

[33] D. Kudenko and H. Hirsh. Feature generation for sequence categorization. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 733–738, 1998.

[34] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, 2001.

[35] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Trans. Inf. Syst. Secur.*, 2(3):295–331, 1999.

[36] J. C. Lee and D. S. Tan. Using a low-cost electroencephalograph for task classification in hci research. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, pages 81–90, New York, NY, USA, 2006. ACM.

[37] N. Lesh, M. J. Zaki, and M. Ogihara. Mining features for sequence classification. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 342–346, 1999.

[38] C. S. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Pacific Symposium on Biocomputing*, pages 566–575, 2002.

[39] C. S. Leslie and R. Kuang. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5:1435–1455, 2004.

[40] D. D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *ECML' 98: The 10th European Conference on Machine Learning*, pages 4–15, 1998.

[41] C. Li, L. Khan, and B. Prabhakaran. Real-time classification of variable length multi-attribute motions. *Knowl. Inf. Syst.*, 10(2):163–183, 2006.

[42] M. Li and R. Sleep. A robust approach to sequence classification. In *ICTAI '05: Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence*, pages 197–201, 2005.

[43] W. Li, J. Han, and J. Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In *ICDM '01: Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 369–376, Washington, DC, USA, 2001. IEEE Computer Society.

[44] J. Lin, E. J. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15(2):107–144, 2007.

[45] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1987.

[46] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. J. C. H. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

[47] M. D. Marzio and C. C. Taylor. Kernel density classification and boosting: an l2 analysis. *Statistics and Computing*, 15(2):113–123, 2005.

[48] T. M. Mitchell. *Machine Learning.* McGraw-Hill, New York, 1997.

[49] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J.Mol.Biol.*, 48:443–453, 1970.

[50] M. Núńez. The use of background knowledge in decision tree induction. *Mach. Learn.*, 6(3):231–250, 1991.

[51] R. J. Povinelli, M. T. Johnson, A. C. Lindgren, and J. Ye. Time series classification using gaussian mixture models of reconstructed phase spaces. *IEEE Trans. on Knowl. and Data Eng.*, 16(6):779–783, 2004.

[52] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106.

[53] J. R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[54] L. Rabiner. A tutorial on HMM and selected applications in speech recognition. In *IEEE*, pages 257–286, 1998.

[55] C. A. Ratanamahatana and E. J. Keogh. Making time-series classification more accurate using learned constraints. In *SDM*, 2004.

[56] H. Saigo, J.-P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–1689, 2004.

[57] B. Schölkopf, K. Tsuda, and J.-P. Vert. *Kernel Methods in Computational Biology*, pages 171–192. The MIT press, 2004.

[58] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley, 1992.

[59] R. She, F. Chen, K. Wang, M. Ester, J. L. Gardy, and F. S. L. Brinkman. Frequent-subsequence-based prediction of outer membrane proteins. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 436–445, 2003.

[60] T. Smith and M. Waterman. Identification of common molecular subsequences. *J.Mol.Biol.*, 147:195–197, 1981.

[61] S. Sonnenburg, G. Rätsch, and C. Schäfer. Learning interpretable SVMs for biological sequence classification. In *RECOMB '05: The Ninth Annual International Conference on Research in Computational Molecular Biology*, pages 389–407, 2005.

[62] S. Sonnenburg, G. Rätsch, and B. Schölkopf. Large scale genomic sequence svm classifiers. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 848–855, 2005.

[63] P. K. Srivastava, D. K. Desai, S. Nandi, and A. M. Lynn. HMM-ModE-Improved classification using profile hidden Markov models by optimising the discrimination threshold and modifying emission probabilities with negative training sequences. *BMC Bioinformatics*, 8(104), 2007.

[64] T.M.Cover and P.E.Hart. Nearest neighbor pattern classification. *IEEE Transactions on Infomration Theory*, 13(1):21–27, 1967.

[65] G. G. Towell, J. W. Shavlik, and M. O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *In Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 861–866, 1990.

[66] L. Wei and E. Keogh. Semi-supervised time series classification. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 748–753, 2006.

[67] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana. Fast time series classification using numerosity reduction. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 1033–1040, 2006.

[68] Z. Xing, J. Pei, G. Dong, and P. S. Yu. Mining sequence classifiers for early prediction. In *SDM'08: Proceedings of the 2008 SIAM international conference on data mining*, pages 644–655, 2008.

[69] Z. Xing, J. Pei, and E. Keogh. A brief survey on sequence classification. *To appear in SIGKDD Explorations*.

[70] Z. Xing, J. Pei, and P. S. Yu. Early classification on time series: A nearest neighbor approach. In *IJCAI'09: Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1297–1302, 2009.

[71] O. Yakhnenko, A. Silvescu, and V. Honavar. Discriminatively trained markov model for sequence classification. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 498–505, 2005.

[72] L. Ye and E. Keogh. Time series shapeletes: A new primitive for data mining. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009.

[73] J. Yin, Q. Yang, D. Shen, and Z.-N. Li. Activity recognition via user-trace segmentation. *ACM Trans. Sen. Netw.*, 4(4):1–34, 2008.

[74] X. Yin and J. Han. CPAR: Classification based on predictive association rules. In *SDM*, 2003.