Mining and Classification of Multivariate Sequential Data

Ariella D. Richardson

Department of Computer Science

Ph.D. Thesis

Submitted to the Senate of Bar-Ilan University

Ramat-Gan, Israel

February, 2011

This work was carried out under the supervision of
Professor Sarit Kraus and Professor Gal A. Kaminka
(Department of Computer Science), Bar-Ilan University

Acknowledgements

This thesis would not have been possible without the help of many people along the way. First and foremost I would like to thank my supervisors Prof. Gal A. Kaminka and Prof. Sarit Kraus for their endless guidance, insight and encouragement. Gal and Sarit provided a unique environment, and like parents complete one other to create the perfect team. It was an honor and a pleasure to have them as my supervisors.

Many thanks are due to all members of the MAS group. Especially to Noa Agmon, Tammar Shrot and Galit Haim for their friendship and advise. To Yael Ejgenberg, Anat Sadeh-Or, Yoav Schwartz and Yael Blumberg for their assistance.

I would like to thank Prof. Patrice L. Weiss and Dr. Sara Rosenblum for introducing the world of handwriting deficiencies. It was a pleasure working together.

I gratefully acknowledge the financial support of the Israeli Ministry of Industry and Trade under the NEGEV project.

I would like to thank to my family. Thanks to my parents Joan and Robert, without their upbringing and support, I would never be who I am. To my one and only sister Tammy, who never fails to listen and always has something sensible to say. To my in-laws Rachel and Avi for their ongoing support. Many thanks to my loving husband Eitan who encouraged me to start my PhD. and proudly supported me throughout the process. Thanks to my children Elad Ido and Yael for giving it all a purpose. Finally I would like to dedicate this thesis to my beloved grandparents: I hope I make you proud.

Abstract

Multivariate sequence mining and classification are important and challenging tasks. They can be applied to numerous domains including medical diagnosis, handwriting deficiency diagnosis, identification of users for security or personalized TV services, and even transportation and traffic planning. The problem we address in this dissertation is classification of multivariate sequences. Multivariate sequences are sequences that have multiple attributes for each item in the sequence. Several attempts to address this problem exist, but none provide a full solution. One type of solution to this problem is to reduce the solution to a single attribute or non sequential problem while loosing valuable information. Other solutions address both the multivariate and the sequential aspect of the input but provide an unscalable solution.

In this dissertation we first present COACH (**C**umulative **O**nline **A**lgorithm for **C**lassification of **H**andwriting deficiencies). COACH is a classification algorithm for multivariate sequences that uses heuristics to combine several single attribute classifications. COACH is evaluated on real data obtained from children with poor handwriting using a digitizer tablet. Results show that COACH manages to successfully differentiate between poor to proficient handwriting. Integrating several single attribute classifications encouraged us to search for a solution that uses all the attributes together in the classification process.

The second part of the dissertation introduces frequent sequence mining. Frequent sequence mining, as well as being a challenging and interesting task, can be used for

classification as we will show in the third part of the dissertation. Many algorithms have been proposed to efficiently address frequent sequence mining. Most of them use support based mining to achieve this task. However, support based mining has been shown to suffer from a bias towards mining short sequences. We will show how resolving this bias produces better sequences than traditional support based mining.

We present REEF, a frequent sequence mining algorithm that resolves this length bias. We define *norm-frequency*, based on the statistical z-score of support, and use it to replace support based frequency. Unfortunately the use of *norm-frequency* hinders pruning. We address this issue and introduce a bound to perform pruning. Calculating the *norm-frequency* requires a preprocessing stage performed on a sample of the database. Values acquired from the sample suffer from a distortion. We analyze this distortion and correct it.

Experimental results on synthetic and real world data presented in this dissertation establish that REEF overcomes the short sequence bias successfully. Mining performed with REEF on textual data is used to demonstrate that the sequences mined with REEF are more meaningful than those mined with support based algorithms, indicating that REEF is better than traditional algorithms for producing interesting sequences.

Finally in the third part of the dissertation we use the new mining algorithm REEF to develop CUBS (**C**lassification **U**sing **B**ounded Z-Score with **S**ampling) a classification algorithm for multivariate sequences. CUBS uses the REEF mining to produce frequent subsequences, and then selects among them the statistically significant subsequences to compose a classification model. We evaluate the accuracy of CUBS on a synthetic dataset and on two real world dataset. CUBS provides a scalable classification algorithm for multivariate sequence classification that makes use of both the multiple attributes and the sequential nature of the data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The availability of data in the modern world poses a great challenge to those who try to tame it. Data is extracted in almost every domain imaginable. The computerization of systems, sensor usage combined with storage capabilities produce a vast sea of data waiting to be mined. The challenge is to sift through the data, to produce interesting and useful information.

Different types of tasks are associated with extracting information from data. Data mining is defined as the application of specific algorithms for extracting patterns from data [20]. These patterns are interesting in their own right, and also used for other learning tasks such as classification, prediction, and clustering.

The structure of the data from various sources is very diverse. Data can represent a single time point or be sequential. Data can be representative of a single attribute, or multivariate (multiple attributes). Obviously these structures influence the algorithms we use for learning, regarding both the input fed to the algorithms and the output we expect to obtain. The scale and diversity of the data raise difficult computational challenges.

The task we focus on is classification of multivariate sequential data. The algorithms we are interested in obtain data that have several categorical attributes and

are sequential. We mine the data and learn a model that is used for classification of unlabeled data. This task is of great interest since it can be applied to a broad set of domains such as medical diagnosis [47, 11], identification of users [46], or transportation and traffic planning [32].

1.1 Multivariate Sequential Data Classification

There have been several attempts to address multivariate sequence classification, but none present a full solution. One solution is to apply time-series classification to each attribute independently e.g. [34, 64, 58]. This allows the use of one attribute at a time and uses the full time series for this attribute. The main drawback of this solution is that there may be information in the combination of several attributes that does not show up when they are separated. A second approach is to perform feature selection that looks at several attributes at single time points, thus losing the temporal property, as in [11]. Another option is to perform a preprocessing stage that incorporates several attributes on several time points as done in [28]. This is a good solution if one knows what the interesting sets of attributes are, but for many cases, including the domains we investigated, this information is not available.

In the first part of this dissertation in Chapter 3 we introduce a heuristic method for classification of multivariate sequential data. Our algorithm initially performs classification of each attribute separately. We suggest and evaluate several heuristic methods for combining the single attribute classification into a classification based on multiple attributes. We evaluated this method on handwriting diagnosis data. Results on the experimental data proved to be very successful.

The drawback of this method is that it is domain specific. The feature extraction required extensive expert knowledge and a true understanding of the domain. We used information obtained from occupational therapists regarding handwriting

deficiencies in order to select the features used for classification. Not only is this type of information expensive to obtain, it is often nonexistent. Another downside of this algorithm is that the integration between attributes is only performed after classification. Valuable information that may be present in the fashion the different attributes correlate to each other is lost and cannot assist the classification process.

This motivates the algorithm presented in the second part of this dissertation (Chapters 4 and 5). Here we build a classifier that handles multivariate sequential data while allowing for both the sequential and the multivariate aspects of the data to be expressed. There are two parts to building the classifier. The first part involves mining the multivariate sequential data and finding frequent subsequences. The second uses the frequent subsequences to perform classification of the sequential data.

The frequent sequence mining is performed with an innovative frequent sequence mining algorithm that we developed (Chapter 4). We show how this algorithm improves existing mining algorithms. For domains where there is no expert knowledge for preprocessing and feature selection this algorithm is a good choice. This algorithm is not domain specific and we will demonstrate its application to a variety of data sets. We use both synthetic and real world data for evaluation.

The classification (Chapter 5) is performed by first generating frequent sequences. Then we select the statistically significant sequences from among the frequent sequences and use them as class representatives. Classification is performed by calculating the distance between unlabeled data to each class model.

In both parts of the dissertation we present classification algorithms that successfully handle sequential multivariate input. There are differences in the exact nature of the data, the expert knowledge available and the expected output, hence two different algorithms are suggested and evaluated. In the following sections we present motivation for these problems along with more specifics on the proposed approaches.

1.1.1 Classification using Heuristics

The first classification algorithm proposed in this dissertation is described in Chapter 3. In order to classify the multiple attributes sequences the data is dissected into several single attribute sequences. Each of these single attribute sequences is classified separately using standard classification algorithms. The results from the various classifications for each attribute are then combined using domain based heuristics.

This algorithm was evaluated in the domain of handwriting deficiency classification. Handwriting deficiencies are common among many people. They are caused by a variety of conditions, and are of great interest to the Occupational Therapy (OT) community. Diagnosis is usually performed by trained professionals an expensive an imprecise process. There is much interest arising from the OT field in applying AI learning to this domain in order to assist the diagnosis procedure. Simple application of existing single attribute classification algorithms to this problem does not suffice. Indications from the OT community imply that the deficiencies are characterized by a combination of attributes and using a single attribute for classification is not good enough. Obviously there is importance to changes made over time, thus using multiple attributes in a single time point is not an acceptable solution either.

Thus the need for an algorithm that combines the temporal dimension and the multiple attributes is necessary. Our approach is composed of two stages. In the first stage we single out one attribute at a time and perform classification of the handwriting based on this attribute. In the second stage we use various heuristics to combine the single attribute classification into a multivariate classification.

Once the single attribute classification is performed for several attributes we use domain specific heuristics to combine them and find the full multivariate classification. Our results show high success rates on this classification for two data sets of handwriting deficiencies. These results were embraced by the OT community and demonstrate how AI learning can be adapted for multivariate sequential classification

in real world domains.

Although these results from the handwriting domain proved very promising, and provide great assistance to specialists, the heuristics and the feature selection rely heavily on expert information. In this specific domain this assistance was available however this is not the case for all domains and we were motivated to find a classification algorithm that could use multivariate sequential data without the need for expert help. In the next section we describe how this is done.

1.1.2 Classification using Frequent Sequence Mining

The second part of the dissertation is composed of two chapters. Chapter 4 introduces a new algorithm to perform frequent sequence mining. This algorithm improves existing mining algorithms by enabling mining of more interesting patterns than traditional techniques as will be explained. Chapter 5 incorporates our mining algorithm into a classification algorithm that classifies multivariate sequences.

Frequent sequence mining is a broadly researched topic and has many applications. It was first introduced by Agrawal and Srikant [3] in the Apriori family of algorithms. The algorithms perform pattern mining in sequences of itemsets (events) and find frequent patterns in the input. A large variety of algorithms, similar to the Apriori algorithms, have been introduced such as SPADE [66], PrefixSpan [41], SPAM [8] and PRISM [22]. These algorithms all use the support measure for determining frequency. Support of a sequence is simply the proportion of entries in the data base that it appears in. i.e. its relative frequency.

It has been shown [26, 52] that the support based methods mine many more short sequences than long sequences. In other words there is a bias towards short sequences in the mining process. The reason this bias occurs is that short sequences are inherently more frequent than long sequences: every long sequence contains a number of short sequences.

However short sequences may not be interesting patterns to mine. They may be frequent by chance, for example if A is frequent and B is frequent AB are likely to show up often even though there is no meaning to the pattern AB. Several attempts have been made to address this bias. Long patterns can be forced by mining closed or maximal patterns as in BAMBOO [61], BIDE [60], TSP [59] and MSPS [37]. However we do not want to force long patterns any more than we want to force short patterns., We would like to give all lengths a fair chance to be chosen.

Seno and Karypis propose using a length decreasing support constraint in order to overcome the short sequence bias in SLPMiner [52]. They base their algorithm on the idea that in order for a short sequence to be interesting it must be very frequent (have a very high support). Long sequences on the other hand may be interesting with a lower support. This heuristic approach works well, however we provide a solution that is based on statistical methods rather than heuristics.

In contrast to these methods for solving the bias towards short subsequences Horman and Kaminka [26] proposed using a statistical normalization of support. The support measure is normalized in relation to sequence length. They showed how support normalization is very successful in finding frequent subsequences with different lengths in an unbiased fashion. However this solution suffers from a great scalability problem. There are two problems with the scalability of this algorithm. The first is that using the normalized support ruins the pruning in support based mining. This makes the algorithm unscalable. The second problem is that calculating the normalized support requires information obtained by enumerating all possible subsequence candidates yet again causing great scalability problems.

The mining algorithm we introduce in Chapter 4 named REEF addresses both scalability issues. In REEF we introduce a new definition of *norm-frequent* sequence mining based on the work of Horman and Kaminka [26]. We provide a new method for pruning that solves the first scalability problem. And introduce a sampling method to

obtain information used in the *norm-frequent* sequence mining to overcome the second scalability issue. REEF is a frequent sequence mining algorithm that is both unbiased to length and scalable. The experimental results we obtained while comparing REEF to standard support based methods show how the scalability is obtained, how we manage to mine sequences with a variety of lengths and how our sequences are more interesting than those mined with traditional methods.

Using the scalable and unbiased sequence mining algorithm we proceed to the classification algorithm named CUBS described in Chapter 5. This algorithm enables classifying the complex multiple attribute sequential data without separating the data to several single attribute sequences.

The classification algorithm CUBS begins with a set of frequent sequences in the data. From among these sequences we select those sequences that are statistically significant for one class in relation to others. these sequences compose the model for each class. In order to perform classification a distance function is used to determine the distance between the sequences in the model for each class to the sequences in an unclassified set. Essentially we are comparing the distributions of sequences in the test set to those in the model and selecting the closest fit.

This Classification algorithm can be applied to any multiple attribute sequential data set. We applied it to several real world data sets and one synthetic data set. The experimental results in Chapter 5 demonstrate how the classification is successfully obtained. Thus we have provided full classification while using multiple attribute and sequential data.

1.2 Publications

Subsets of the results that appear in this dissertation were published in the proceedings of the following refereed journals, conferences and workshops:

- Ariella Richardson, Sarit Kraus, Gal A. Kaminka. "REEF: Resolving Length Bias in Frequent Sequence Mining", Knowledge and Information Systems Journal, Submitted.

- Ariella Richardson, Sarit Kraus, Gal A. Kaminka. "CUBS : Multivariate Sequence Classification Using Bounded Z-score with Sampling", IEEE International Conference on Data Mining Workshops, pp 72-29, 2010.

- Ariella Richardson, Sarit Kraus, Patrice L. Weiss and Sara Rosenblum. "COACH - Cumulative Online Algorithm for Classification of Handwriting Deficiencies", Proceedings of the Twentieth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI 2008).

Chapter 2

Related Work

The related work for this dissertation is divided into two sections. In section 2.1 we describe the related work for classification algorithms that handle multivariate and sequential data. A survey of frequent sequence mining algorithms appears in section 2.2.

2.1 Multivariate Sequence Classification

The multivariate sequential data we are interested in classifying is complex as it is composed of two dimensions. One dimension is that of the attributes. The data is multivariate meaning that there are multiple attributes, and the number of attributes may vary throughout the data set. The second dimension of the data is the sequential dimension. The data is collected over time, and not necessarily at constant intervals.

The simple manner to perform classification on this type of data is to classify each dimension separately. An example to this type of dimension reduction is to remove the sequential aspect of the data and consider the attributes. This approach has been evaluated by Baxter et al. [11]. Each attribute is a medical test performed at various time intervals. They capture the behavior of a specific attribute (medical test) into a

single feature that is used for classification. In this fashion the sequential dimension is compressed into a single feature and we are left with a single feature for each time series. Knowing how to do this relies heavily on understanding how change in time defines the feature, and some important information on changes over time may be lost. For the handwriting data we actually tried at an early stage of our research compressing each time series into a single value before classification. However this did not achieve good classification results.

The other alternative to dimension reduction is separating attributes to create single attribute sequential data. These single attribute series can then be classified with standard algorithms such as SVM [15] or decision trees [43]. Using boosting algorithms such as AdaBoost [50] are also an option for combining attributes. Although the procedure is straightforward it is not a good solution for problem we are trying to solve. Information on how different attributes relate to each other is obviously lost.

A special type of sequential data are time series where the intervals between measurements are typically constant. Several algorithms handle multivariate sequential data by looking at a full time series as one unit. One example is Gwadera and Crestani [24] who discover significant patterns in multi-stream sequences. Another example is Liu et al. in [34] who perform causal analysis to find how one time series infers another. Using the full time series does not allow for finding dependencies or patterns that occur among segments of the various time series. These algorithms require processing the full time series in order to perform classification and therefore are problematic for online classification applications where we are interested in using small parts of the data for classification. We are more interested in online classification algorithms where we do not have the full time series, thus the search for a new solution.

An interesting method for time series prediction is presented by Alberg et. al. in [6]. Alberg suggests taking the time series and dividing it into intervals. Each interval

is transformed into two statistical moments, and these are used for prediction. In later work [5] they use a sliding window to decrease input parameters. Both these papers deal with stream data. One of the big challenges of stream data is that it appears online and usually in large amounts, and must be handled quickly. However each of the attributes is handled in a separate sequence. This is different to the domains we used, where the main difficulty did not arise from the streaming nature of the data, but rather from the relationships between the attributes.

A more advanced approach is taken by Kadous and Sammut [28] who use metafeatures to convert each series into several features and then perform classification. Kadous and Summut apply their algorithm to a form of sign language. They define a metafeature which is a feature composed of several attributes over several temporal values. In this metafeature they essentially capture the type of data unit that we are interested in using for our classification process. This is a unit that spans both the attribute dimension and the temporal dimension and captures the essence of the complex behavior. The metafeatures are then handled by a standard classifier. The drawback of this solution is the difficulty in defining these metafeatures. The definition requires domain knowledge and understanding of the structure of "good" features. In the domains we looked at, as in many other domains this knowledge was unavailable. Thus we need to search for a mining algorithm that can recover the interesting structures without the domain knowledge. Furthermore restricting the feature selection to these predefined metafeatures may result in loss of information.

Closer to our work are Silvescu et al. [54]. They model the data as multiple attribute sequences as we do and substitute feature selection by mining k-grams of sequences and using them as representatives. k-grams are composed of k items that appear together. In our solution we want to allow gaps in the mining and not impose full subsequences to be used. The k-grams algorithm does not allow for the breaking of the data as we do. For example in a sequence such as $A- > B- > C$ they would

obtain $A->B$, $B->C$ but not $A->C$ as we can. However this work of Silvescu leads to the idea of using frequent itemset or sequence mining for classification tasks.

Frequent itemset mining for classification has been suggested by several researches. For example in [17] Deshpande and Karypis perform itemset mining and use the frequent itemsets as input to an SVM classifier. The data that is mined is not sequential in their case. The items that are mined have no order and the frequent mining algorithm is simply looking for groups of items that co-occur. This enables use of the SVM (or any other standard) classifier after the mining phase. However in our scenario the data is sequential and even after we have mined the data for frequent sequences we are still dealing with sequences and cannot use them as input for a standard classifier.

The idea of expanding this type of frequent itemset mining (which is correlative to multivariate item mining) into frequent sequence mining (correlative to multivariate sequence mining) was introduced by Zaki and Lesh [67]. They use multivariate subsequences for failure detection in planning. They collect subsequences that are typical of successful and failed plans. Subsequences that appear only in failed plans are used for failure detection. This seems to be beneficial, however it seems that the pruning process can be refined and we improve on this in our work. Zaki and Lesh actually provide the basis to the research we performed in the second part of this work and we will enhance this algorithm in our work.

Lee and Tseng proposed univariate classification in [58], by searching for frequent patterns and then refining to find the ones that are good for classification. This is a similar idea to ours, but not easily extended to multivariate series. Recently Lee and Tseng proposed another algorithm that performs classification of multivariate sequential data [31]. They separate the data into single attribute sequences and then use a form of sequence mining to find rule based classification on the single attribute

sequences. Then the single attribute rules are combined to provide a multiple attribute output. This is similar to the approach that we published earlier in [47] and describe in Chapter 3. Their work differs from our in that the multiple attributes are mined separately and then combined to provide the full classification. In contrast in Chapters 4 and 5 we perform multivariate sequence mining and use both sequential and multivariate aspects of the data to assist in the classification.

2.2 Frequent Sequence Mining

A large portion of our research was devoted to improving the frequent sequence mining algorithm as described in Chapter 4. Therefor we chose to expand the survey of related work on this topic.

Support based algorithms for frequent sequence mining were first introduced by Agrawal and Srikant [3] where the algorithms AprioriAll, AprioriSome and DynamicSome were introduced. These algorithms naturally expand frequent itemset mining [1] to frequent sequence mining, for sequences of itemsets. The algorithms perform pattern mining in sequences of itemsets (events) and find frequent patterns in the input. The itemsets typically contain multiple items. Later they introduced the more efficient GSP [55] which has been broadly implemented and used since.

Since the search space for these mining problems is incredibly large other support based algorithms were introduced to improve the speed and efficiency of the mining process. SPADE [66], introduced by Zaki, is an algorithm for frequent sequence mining that belongs to the family of support based mining algorithms. SPADE outperforms GSP, due to the use of a vertical layout for the database and a lattice-theoretic approach for search space decomposition. Later Pei et. al proposed PrefixSpan [41]. By recursively projecting the database into sets of smaller databases and searching for the frequent patterns locally PrefixSpan is shown to perform better than both

SPADE and GSP.

An improved algorithm, SPAM [8], uses a depth first strategy with efficient pruning and outdoes previous algorithms. Using a directed acyclic graph (DAG) has been suggested by Loekito et. al in [35] to outperform the projection method of PrefixSpan for input sequences that are highly similar.

Zaki countered PrefixSpan and SPAM with PRISM [22], using the vertical approach from SPADE but adding prime block encoding and yet again performance was improved in relation to previous methods. These algorithms provide different database representations and different pattern growth techniques to try and obtain efficient and scalable algorithms. They are all based on the support measure for determining frequency. We will adopt the method presented in SPADE [66] and adapt it to use a normalized support for finding frequent sequences. Any of the support based algorithms could be used rather than SPADE, however since we are not interested in speed of the mining all the algorithms are equivalent.

They key idea in many of these support based algorithms is the generation of candidate sequences. The candidate sequences are subsequences of the input-sequences in the database. *Frequent* candidate sequences are both placed in the set of mined *frequent* sequences, as well as used to generate the next generation of candidates. First, 2-sequences (sequences of length 2) are generated, then they are used to create 3-sequences etc: Pairs of l-sequences with common prefixes are combined, to create an $l+1$-sequence.

Generating all possible candidate sequences is infeasible and results in an unscalable solution. Therefor a pruning is introduced to this process. Candidates that are *not frequent* are pruned. They are not used to generate the next generation of candidates. The reason this can be done is based on the anti-monotonic property of support. Support has a nice anti-monotonic property promising that it does not grow when a candidate sequence is expanded. This promises that candidate sequences that

are *not frequent* will never generate *frequent* sequences, and therefore can be pruned. Thus the anti-monotonic property is very important and ensures scalability of the mining.

Alongside the rich variety of support based mining algorithms Mannila et.al [38] proposed an algorithm for mining frequent episodes, a type of frequent sequence, in an input composed of a single long sequence. Frequent *episode* mining algorithms find frequent items that are frequent within a single sequence whereas frequent support based sequence mining searches for items that reoccur in multiple sequences. Similar is the MSDD algorithm [40] for finding patterns in multiple event sequences. MSDD focuses on finding patterns with high dependencies among the items. Recently Tatti and Cule [56] proposed mining closed episodes that are represented as DAGs. This algorithm cannot handle multivariate sequences, and although in theory it could be extended to the multivariate case this has not yet been done.

Although the problem of Frequent Sequence Mining has been solved, the *frequent* sequences found are often insufficient. Unfortunately support based mining methods suffer from a bias towards shorter sequences as has been shown in [26]. This means that in the frequent sequence mining, short sequences are found more often than long sequences. This is very problematic since these short sequences are often not very interesting.

Several attempts have been made to address this bias. One possibility is to force large patterns by searching for closed or maximal patterns for example BAMBOO [61], BIDE [60], TSP [59] and MSPS [37]. In BAMBOO only patterns that do not have larger frequent patterns containing then are mined. Later BIDE [60] was introduced by Wang and Han where an efficient mining of the complete set of closed frequent sequences is presented. Tzvetkov et. al present the TSP algorithm that also performs mining of closed sequential patterns. TSP finds the top-k closed patterns rather than using a minimum support threshold. The algorithm is based on repetitive automatic

adjustments of the minimum support. Mining the top-k patterns is shown to be preferable to setting the minimum support. However from the point of view of length bias it is similar to other closed pattern mining algorithms. Luo and Chung [37] propose an algorithm for mining maximal frequent sequences (MSPS) as an attempt to find longer subsequences and avoid the flooding of short subsequences. However mining closed or maximal patterns may not be the best approach to solve the short sequence bias. Using closed and maximal sequences ignores shorter partial sequences that may be of interest. We propose an algorithm that mines sequences of all lengths without a bias towards long or short sequences.

In LPMiner [51](itemset mining) and SLPMiner [52](sequence mining) Seno and Karypis introduce a length decreasing support constraint in order to overcome the short sequence bias. This is based on the observance that in order for a short sequence to be interesting it must be very frequent (have a very high support). Long sequences on the other hand may be interesting with a lower support. SLPMiner is a heuristic approach whereas in our work we attempt to find a general solution based on support normalization.

An alternative approach is taken by Yun and Legget in WSpan [65]. They introduce a weighted mining algorithm, for sequences with weighted items. Using weights in the mining process is very useful since it provides more input than using frequency alone. Unfortunately this is of no assistance in domains where there is no information on what weights to apply. Our solution requires no knowledge on what weights should be used and can be implemented in any domain.

The methods for solving the bias towards short subsequences suggested in [61, 60, 51, 52, 65] are heuristic. They are based on forcing long sequences to be mined. In contrast Horman and Kaminka [26] proposed using a statistical normalization of support. The support measure is normalized in relation to sequence length. They

showed how support normalization enables finding frequent subsequences with different lengths in an unbiased fashion. Using normalized support makes no assumptions on the relation between length to support, or on the relative weights of the items in a database as were made in the other methods.

Although Horman and Kaminka [26] successfully solve the statistical bias using normalization, their method suffers from a scalability problem. There are two problems with the scalability of this algorithm. The first is that using the normalized support ruins the anti-monotonic property used for pruning in support based mining. Unfortunately this makes pruning impossible and therefore the algorithm is unscalable. The second problem is that calculating the normalized support requires information obtained by enumerating all possible subsequence candidates yet again causing scalability problems.

With the scalability spoiled it seems there is a need to choose between a scalable algorithm that is heuristic to one that can fully overcome the short sequence bias but is unscalable. In this thesis we propose an algorithm that can do both. The algorithm we present uses normalized support to overcome the short sequence bias successfully while using a pruning method and a sampling unit to solve both scalability issues.

Chapter 3

Multivariate Sequence Classification using Heuristics

This chapter introduces our first method for performing multivariate sequence classification. The method used in this chapter involves extracting several single attribute sequences from the multivariate sequences. Each of the single attribute sequences is classified separately. Then heuristic methods are used to combine the separate classifications of the single attribute data to a single classification for the multivariate input. This method was explored in the domain of handwriting deficiency classification. We first introduce the domain, and then introduce the method.

Many people suffer from handwriting deficiencies of different kinds. These deficiencies can be of various origins and have many characterizations. The number of people with problems such as these is increasing all the time. The diagnosis of such problems is usually performed by trained occupational therapists using a set of Handwriting Evaluation tests such as described in [19]. There are many problems with this type of testing. The tests are limited to characteristics of the writing observable by humans. The testing is subjective and if performed by an unexperienced therapist may be wrong. Testing is very time consuming and expensive since it requires

professional evaluation and therefore is usually only performed once for diagnosis. An application that provides diagnosis would lower costs of evaluation, provide support to unexperienced therapists, enable re-evaluation throughout therapy to test for improvement and is therefore an important contribution to this domain.

Another problematic aspect of existing evaluation techniques is their inability to use information hidden from the human eye. This includes the pen's pressure used when writing, or the pen's tilt and azimuth. This data can add insight to what causes the difficulty and how to intervene in order to improve handwriting. Online data from the handwriting process are collected using a digitizing tablet and instrumented pen. These data provide much information on the handwriting process, but are complex to analyze since they have multiple attributes and are collected over time. We do not have enough expert information concerning the relative importance of the various attributes so we must look at all of them. We use learning and data mining from AI and applies them in an innovative fashion to handwriting deficiency classification.

Our first algorithm is COACH (a Cumulative Online Algorithm for Classification of Handwriting deficiencies) which is an online innovative classification algorithm that provides the user with immediate feedback on handwriting. COACH can be used as a diagnostic tool for subjects. In addition COACH can be used to test various handwriting interventions by the therapist or to practice alone after the correct intervention is found. The algorithm uses pressure, tilt and inAir (the time that the pen is not in contact with the surface) and can provide details on the proficiency of a writer for each attribute. COACH is trained on data collected from various writers both proficient and poor and can provide an online evaluation of new handwriting samples.

The most challenging aspect of this research is the structure of the data. We use real world data collected by occupational therapists for their ongoing handwriting research. To use the data with AI learning mechanisms we needed to pre-process the

data. This was a difficult task because of the nature of the data. The data is in the form of multivariate time series. Most classification algorithms are not capable of directly classifying multivariate time series.

Multivariate data that are not dependent on time such as a vector of information gathered on a subject (age, sex, height, temperature) are easy to classify [63]. Time series with a single attribute, e.g. temperature at different time intervals $\{temp(t_1), temp(t_2), ..., temp(t_n)\}$ can also be classified using available classifiers, for example Morabito and Versaci [39]. However, multivariate time series are difficult to classify.

Using common solutions [16] it is possible to choose a single attribute from among the set of attributes and use it for classification. However, this results in the loss of the information found in the other attributes. Alternatively, one can look at multiple attributes in separate time intervals, but then data from other time intervals are lost. In both cases the process could be repeated for each attribute or time interval and then the classifications could be integrated.

Another approach is to integrate data over time for each attribute, for example using averages and standard deviations on the whole time series for each attribute and creating a vector of the results for each subject [11].

We chose to separate the attributes and create time series for each attribute. This is performed for various time intervals in order to simulate online behavior. These time series are classified and the integration of the results for all the attributes and time intervals are combined to provide the current classification.

The problem of handwriting recognition is known to be a difficult task and much research has been conducted in this domain e.g. [9]. However previous research has not addressed the special characteristics of handwriting belonging to writers with various deficiencies. Most handwriting studies have been made on proficient writers and it is obvious even to the naked eye that the writing from deficient writers is

shaped differently to proficient writing. It therefore seems that standard techniques are not applicable.

Some exploration of handwriting deficiencies using computerized methods has already been done for example by Rosenblum, Parush, and Weiss [48, 49]; one of the unique issues in the present study is the cooperation between disciplines. The need for creating a handwriting classification tool evolved within the occupational therapy (OT) community. This interaction between OT and computer science provided us not only with real data, but also with focus on which points are of interest in this domain. Finding the attributes that contribute to good classification provides insight into understanding the mechanisms of poor handwriting. It seems that the OT community is ready to embrace new technologies to assist with the diagnosis of various conditions. COACH demonstrates potential for AI integration in OT. It shows how AI techniques can contribute to solving OT problems and why this cooperation seems promising.

One of the difficulties encountered during the adaptation of the classification algorithms to handwriting was the choice of the unit of text that was best suited for analyzing. In order to discuss this let us define the term stroke.

Definition 3.0.1 (Stroke) *A **stroke** is a continuous line that the subject draws on the paper.*

Some letters are typically composed of multiple strokes - such as the letter 'K' and others composed of a single stroke such as 'O'. Poor writers however, tend to break each letter into more strokes. In contrary to our initial assumption that letters (or even words) would be used as a basic text unit for handwriting deficiencies we discovered that our analysis can be based on strokes. The ability to extract information from strokes rather than letters is very important as it saves the need to use a text identification algorithm [9]. This may be important when dealing with poor writers who tend to form letters in unconventional ways. Using strokes also means that the

algorithm can be applied to character sets from many languages. We found that it is possible to perform classification after a very small amount of writing and this contributes to our ability to build an online system.

COACH may provide insight to many disabilities. Alzheimer's disease, multiple sclerosis, and other diseases affect handwriting abilities and it would be interesting to study these diseases through handwriting. Understanding handwriting mechanisms may provide a vantage point for understanding the broader area of all motor control.

COACH is also applicable to anomaly detection. Anomalies are observed events which deviate from what is expected. When sensors are used to measure the behavior of the system, it is easy to report problems when a sensor measurement exceeds a defined threshold. However in the case that this threshold is unknown, or some other unknown collective sensor behavior causes trouble it is not easy to find the anomaly. A similar process to the one described in COACH can be used to learn to classify the anomalous behavior. This opens a broad array of possible applications for example: evaluating robot performance and accessing security in computer systems.

3.1 Problem Description

Let M be a set of labeled matrices, one matrix per subject, where each matrix m_i^x is of the form:

$$m_i^x = \left\{ \begin{array}{cccc} a_{t_1}^1 & a_{t_1}^2 & ... & a_{t_1}^n \\ a_{t_2}^1 & a_{t_2}^2 & ... & a_{t_2}^n \\ . & . & ... & . \\ a_{t_l}^1 & a_{t_l}^2 & ... & a_{t_l}^n \end{array} \right\}$$

- $a_{t_j}^o$ is the value of attribute a^o sampled at time t_j.
- all subjects have the same number and type of attributes a^1 to a^n but the number of time samples t_l can vary for different subjects.

- x is the class label of the subject, X_1, X_2 ... X_K

We use the labeled data from the set of matrices in M to compose a model. This model will be used for classification. Given a new matrix m' we want to know for which k $m' \in X_k$, $1 < k < K$. It would also be beneficial to obtain some information on which attributes a^i contribute or affect the classification.

For example: in our domain M is a set of matrices of data collected on 9 yr old children, where each matrix m_i is the data from one subject. The subjects are labeled according to their handwriting abilities, $X1 = \{\text{proficient}\}$, $X2 = \{\text{poor}\}$. There are $n = 3$ attributes a^1 =pressure, a^2 =tilt, a^3 =inAir time. The data is sampled at times $t_1 = 0$, $t_2 = 0.01$ to $t_l = 10$. Given data from a new subject m' we would like to determine whether $m' \in X1$ (proficient) or $m' \in X2$ (poor).

3.2 Cumulative Online Algorithm for Classification of Handwriting deficiencies

The task of classifying handwriting consisted of several stages. First we analyzed the data off-line. This analysis determined the units of data used for classification, the classifier used, and the manipulations made on the data. Then we tested our algorithm. Once preliminary results were obtained we decided on heuristic improvements to be made to the algorithm and tested it again. Details on how this was done are presented below.

The first task was to determine which units of the handwriting need to be used. Initially it seemed natural to use letters as a basic unit for model building and classification. Letters have different lengths and shapes. Our assumption was that it would be meaningless to compare a long complicated letter to a short easy one since deficient writers are expected to have more trouble with complex letters. However segmenting

the data into letters is a non-trivial research issue [12], [9], [29]. Furthermore we have no interest in recognizing the letters but rather want to uncover characteristics of the handwriting style or deficiency.

We therefore we decided to use strokes (a continuous line drawn without lifting the pen). Using strokes is very helpful as it avoids the need to segment the data into letters; it is easy to extract strokes. Strokes may be used for many languages or perhaps even drawing.

We proceeded to choose a classifier. We decided to perform classification of a single attribute time series, for each attribute, and then integrate the results. Classification of the single attribute time series was performed using WEKA [63], a collection of machine learning algorithms for data mining tasks. COACH uses Decision Trees (C4.5) [43] for classification, with Leave-one-out cross-validation for evaluation.

After selecting the data units (strokes) and the classifier (C4.5) we explored the option of processing the data before performing the classification. We performed the classification both on the data in its raw form and also used various manipulations of the data. Our manipulations involved taking derivatives of the data. We also used means and standard deviations on the whole timeseries. The conclusion of these preliminary experiments was that it is best not to perform any manipulations on the data.

The next stage was to classify all the data for each attribute separately and build a model. The main aim of this is to provide input for the next stage of the classification process where we integrate the classifications obtained on different attributes. The important byproduct of this process is information on how each attribute contributes to the classification process. This in itself is valuable output for an occupational therapist researching handwriting. The single attribute classification provides information on the behavior of attributes that are typical of a deficiency.

Rather than building the model on all the data, the text is divided into N parts

1: $FinalClass \leftarrow \emptyset$, $C \leftarrow \emptyset$
2: **for all** Att **do**
3: Divide *text* into N parts
4: **for** $i \leftarrow 1$ to N **do**
5: $S_i \leftarrow$ first M strokes from part i
6: $C_{i,Att} \leftarrow ClassifyAttSet(S_i, Mod_{i,Att})$
7: $FinalClass \leftarrow CombineHeuristic(\{C_i\})$
8: **return** $FinalClass$

Figure 3.1: COACH(*text*)

and a model built for each part and for each attribute $Mod_{i,Att}$. There are two reasons for dividing the text into parts. The first is that this simulates an online classification. The second is that it is known that for poor writers writing usually deteriorates over time. It is therefore important to classify strokes of an unclassified writer with the model that corresponds to the same part of the writing task. Once we have found the classification for single attributes we want to create a final classification based on all attributes. For this we must find ways to combine the results obtained in the single attribute classification. We later suggest several heuristics along with experimental evaluations.

The classification of a new subject is performed using the $COACH$ algorithm (Algorithm 3.1). $COACH(text)$ is provided with the text belonging to a new subject that we wish to classify. For each attribute we divide the text into N parts, select the first M strokes from each part and classify them using $ClassifyAttSet(S_i, Mod_{i,Att})$. Once we have a classification for each ('attribute','text part') pair we use a heuristic to combine all the classifications.

We tried different heuristics for combining these into one multiple attribute classification that is performed in $CombineHeuristic()$ found on (line 8). The first heuristic uses a majority vote and choose the classification that was found most often in the single attribute classifications. The second uses one attribute on part of the text and

another for other parts. The third chooses the attribute we use to classify based on the models we find. Details on these heuristics along with some experimental results appear later on.

The $ClassifyAttSet(S_i, Mod_{i,Att})$ performs iterative single attribute classification. S_i is the set of strokes currently being classified. $Mod_{i,Att}$ is the model built from training data that corresponds to part i of the text for attribute Att (pressure, tilt or inAir). For each unclassified subject we classify single strokes from one part of the text and use a majority vote to determine the classification. In case of a tie we use random classification.

3.3 Experimental Results

3.3.1 Dataset

The data used was collected on a WACOM x-y digitizing tablet using a wireless electronic pen with pressure sensitive tip. At each time interval samples of the x and y coordinates, pressure, tilt and azimuth are taken, which creates a time series for each attribute. The pressure attribute measures the amount of pressure applied to the tablet with the pen. Tilt relates to the angle between the pen to the tablet. Azimuth is a property describing how the pen turns. We derived another attribute named inAir. InAir measure the time between strokes, in other words the time that passe between when the pen is lifted to when it touches the tablet again. We used two datasets. Both of them include children in elementary school. Each set has two groups of subjects, one including the poor writers, and the other of proficient writers (the control). The sets are

- **DCD: Developmental Coordination Disorders.** - 42 subjects, 22 poor (**DCD**), 20 proficient. **DCD** is a motor impairment that affects a subjects

ability to perform the skilled movements necessary for daily living and among other things affects handwriting proficiency.

- **Dysgraphic** - 94 subjects, 49 poor (**Dysgraphic**), 45 proficient. "Dysgraphia" is a learning disability resulting from difficulty in expressing thoughts in writing. The labeling was performed initially by teachers and only then updated by occupational therapists and may be unreliable. Thus, we are working with a noisy data-set.

Each subject was asked to write a few sentences that were predefined and are the same for all writers in a group. The data from each subject is labeled by a trained occupational therapist using a standardized evaluation tool as being poor or proficient. It must be noted that not all subjects classified as poor are the same, some may be more similar to proficient writers than others.

3.3.2 Evaluation

We have results on both the DCD and the Dysgraphic data sets. For both groups we ran ClassifyAttSet($S_i, Mod_{i,Att}$) with Att=tilt, Att=pressure and Att=inAir (the time between strokes). We chose $M = 10$ strokes from each part for our experiments, because this number provided good classification. When we used smaller numbers such as $M = 4$ success rates dropped, however using $M = 50$ did not improve success rates. We use $N = 5$ parts of text, this provided us with as many sections as possible while maintaining enough strokes in each section, if we set $M = 10$.

Results for Single Attribute Classification We first present the average results for the **DCD** data in (Fig. 3.2). The **success rate** is the percentage of subjects classified correctly. This is shown as *% success* on the Y axis in the graphs. The

classification is performed on five parts of the text. We use leave-one-out cross-validation. The results for each part use all strokes obtained from current and previous text parts ('cumulative'). This corresponds to the *text part* on the X axis.

For tilt and inAir the classification reaches over 60% on average. For pressure we obtain a success rate of 70% after the first text part (after only 10 strokes), and reach over 80% on average when using the entire text.

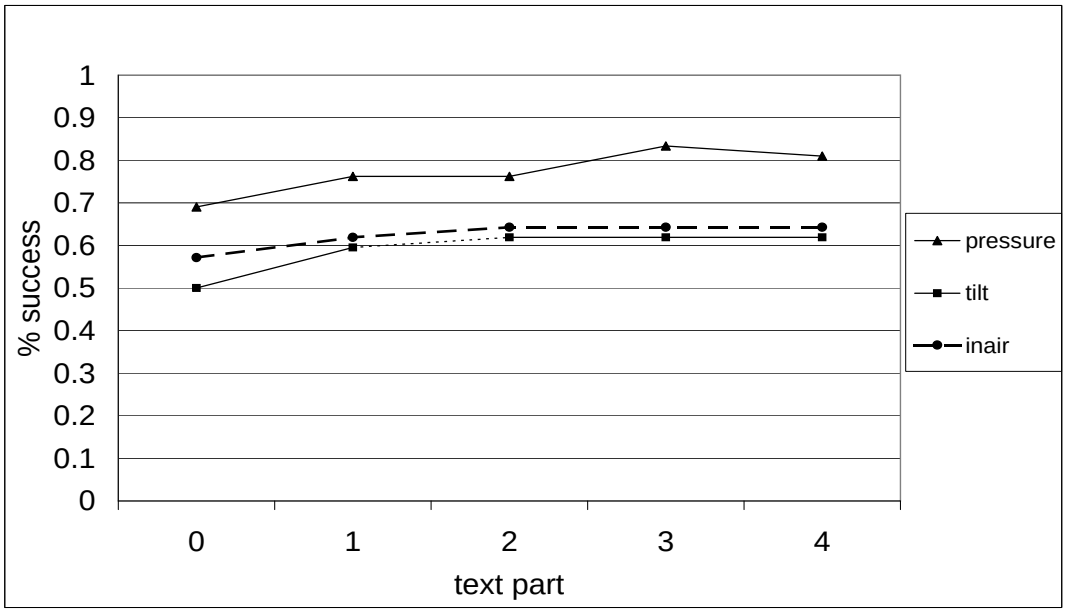


Figure 3.2: DCD results for classification.

We follow with the **Dysgraphic** results in Figure 3.3. The results for tilt are 60%, for inAir they reach 65%. For pressure we obtain a success rate of 75% for the entire text.

3.3.3 Heuristics for Multivariate Classification

As mentioned earlier once we have a single attribute classification we also experimented with some heuristics for combining the classifications of different attributes

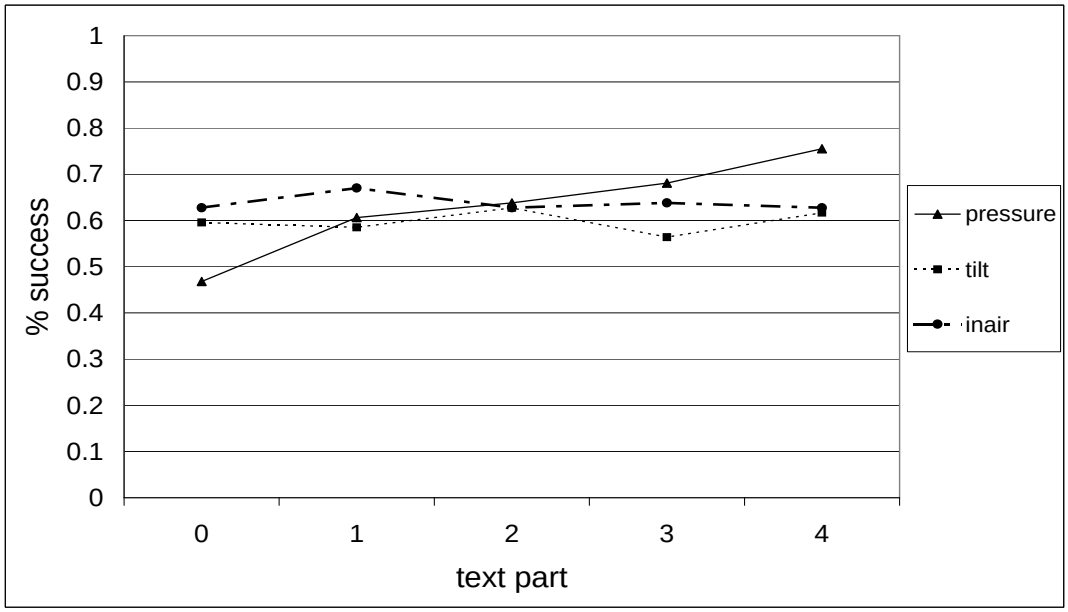


Figure 3.3: Dysgraphic results for classification.

into a single classification. The description of these heuristics together with experimental results follow:

- **TD-pres**: for **DCD** data the main attribute that contributes to classification is pen pressure. We noticed that the classification using tilt was not successful overall. However when the classification using tilt was '**DCD**' it was nearly always correct. This is shown in Fig. 3.4.

  We used this information to introduce the following heuristic: Use classification of tilt when it classifies as '**DCD**', otherwise use pressure classification.

  The results of this heuristic are shown in Fig. 3.5. This figure compares using pressure or tilt alone with using the **'TD-pres'** heuristic that combines both attributes. combining the attributes with our heuristic improves classification results and provides an 85% average success rate.

- **A-P**: for **Dysgraphic** data we noticed that the inAir attribute provides a success rate of over 60% after only 10 strokes have been written. The pressure attribute only starts contributing later (probably because the writers get tired - a known symptom of dysgraphia). We used this to derive the following heuristic: use classification obtained from inAir attribute for first parts of text and then transfer to using the pressure attribute as text proceeds.

  In Figure: 3.6 we present the improvement made by using the **'A-P'** heuristic. We show the classification results using pressure or inair alone as opposed to using the **'A-P'** heuristic. We show how the **'A-P'** heuristic is much better than using each attribute separately and manages to achieve a 76% average success rate.

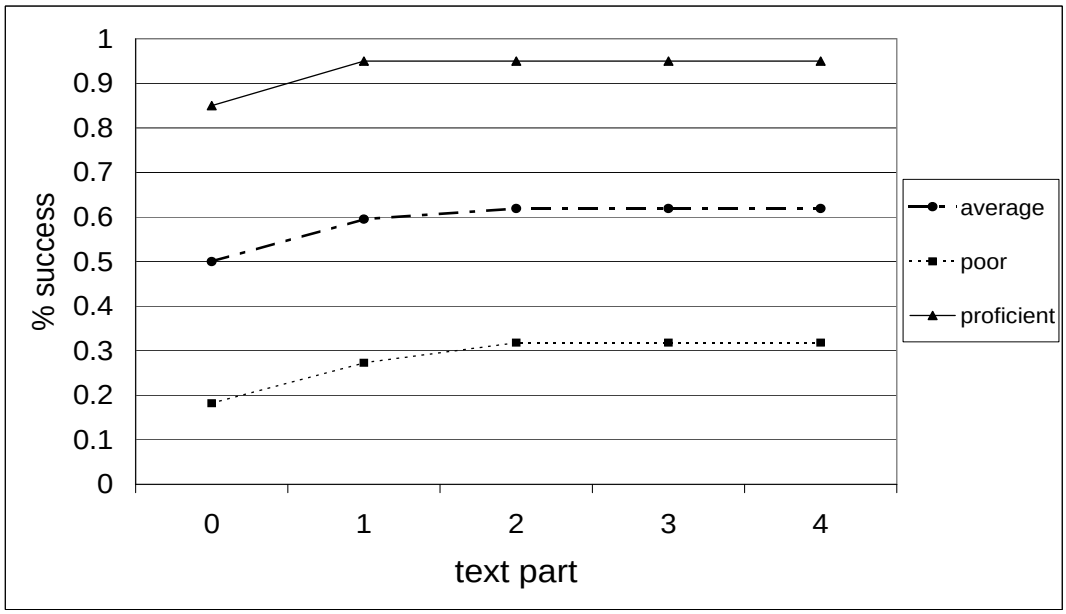


Figure 3.4: DCD results for classification using tilt.

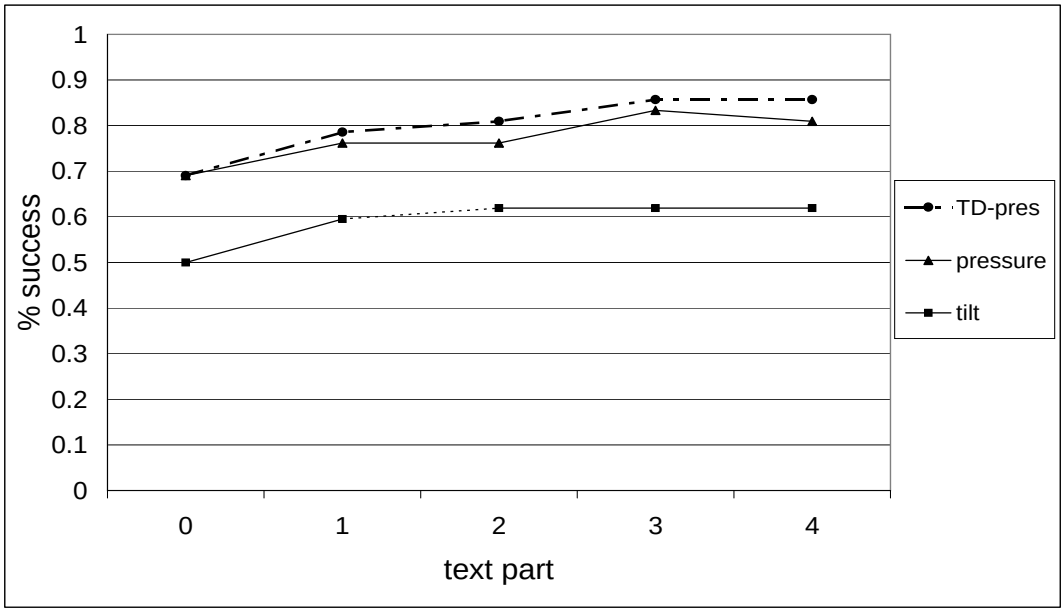


Figure 3.5: DCD classification using TD-pres heuristic.

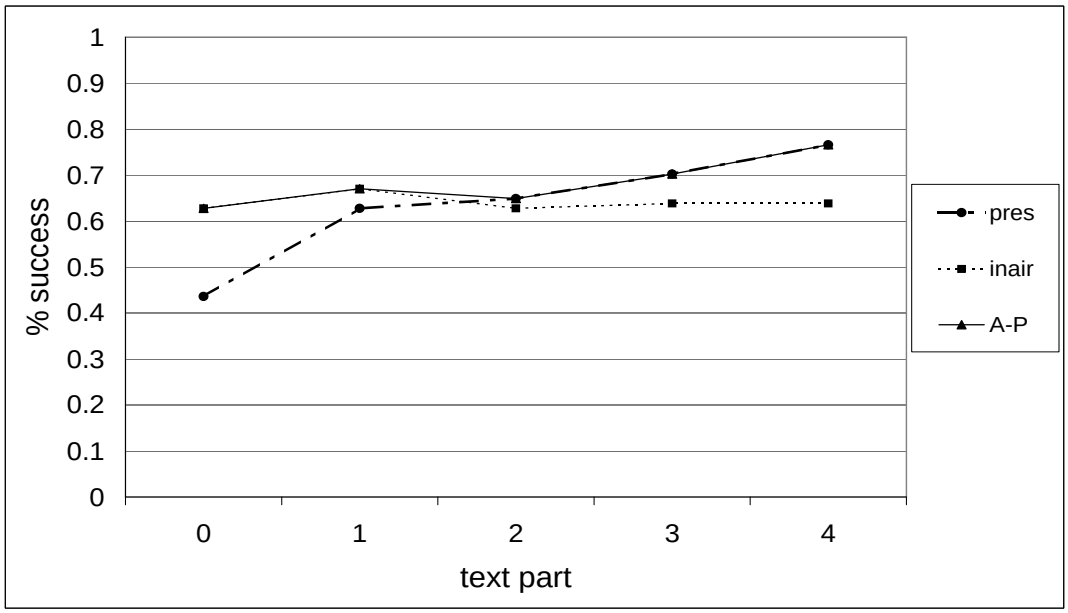


Figure 3.6: Dysgraphic classification using A-P heuristic.

3.4 Discussion

Applying COACH to the DCD group achieved success rates of 85%. These results are considered to be very good in this domain, as diagnosing writing deficiencies is not an exact science and perfect classification is not expected. Pen pressure was found to be a good attribute to use for single attribute classification. It is interesting to see how pressure affects the classification for DCD writers. The DCD writers have trouble with pen pressure right from the start because of the nature of their disability. This enables classification using pressure in the first parts of the writing.

The success rate when applying COACH to the Dysgraphic dataset was around 75%. These results are lower than the for the DCD set for a number of reasons. First, the labeling of the initial data for the Dysgraphic set is noisy, which in turn affects our ability to classify correctly. Another cause may be that writers labeled Dysgraphic may not be very different from proficient writers, furthermore Dysgraphia results from various causes and therefore is diverse making classification a difficult task.

For Dysgraphic writers the pressure attribute can only discriminate after a large proportion of the text is written. The reason is that Dysgraphic writers tend to tire over time. InAir is also especially important in the Dysgraphic set where it is the main attribute that is immediately distinguishable. This is of great importance to an online system where fast discrimination is desired. Hence even when one attribute is dominant there is information hidden in other attributes.

We believe that the results presented in this dissertation can be expanded to other cases such as Alzheimer's disease, multiple sclerosis, and other diseases, which affect handwriting abilities, would be interesting to study. We would like to expand our classification to differentiate between more than two classes in order to develop a deployed classification system that can be used for diagnosis and remediation of

deficiencies. Finally, in the future we plan to further explore multivariate sequence learning in a general fashion in order to classify data from many domains.

COACH has two major drawbacks. The first is that we relied heavily on expert information for selecting attributes and building the heuristics, this information is sometimes expensive and often unavailable.

The second drawback is that in separating the attributes and them merging the classification results we may have lost information hidden in the combination of the attributes. We would like to use all the attributes together in order to perform the classification. This is the task that we focus on in the next chapters.

Chapter 4

Scale Up in Multivariate Frequent Sequence Mining

Mining sequential data to find frequent sequential patterns [3, 1, 55, 66, 41, 8, 22] was first introduced by Agrawal and Srikant [3] and by Mannila et.al [38]. Frequent sequential pattern generation is traditionally based on selecting those patterns that appear in a large enough fraction of input-sequences from the database i.e. whose relative frequency is above a set threshold. This relative frequency is known as *support*, and the threshold is termed *minsup*. All sequences with a *support* higher than *minsup* are considered frequent.

Support based mining is known to suffer from a bias towards short patterns [26]: short patterns are inherently more frequent than long patterns. For example assume the data base has the following sequences :

$$(\{ABC\} \rightarrow \{BF\} \rightarrow \{DEF\})$$
$$(\{ABD\} \rightarrow \{CF\} \rightarrow \{DAF\})$$
$$(\{AB\} \rightarrow \{BEF\}$$

The support of $(\{ABC\} \rightarrow \{BF\})$ is 0.3 since it appears in one of the three sequences only. However the support of $(\{AB\})$ is 1 since it appears in all of the

sequences.

This bias creates a problem, since short patterns are not necessarily the most interesting patterns. Often, short patterns are simply random occurrences of frequent items. The common solution of lowering the *minsup* results in obtaining longer patterns, but generates a large number of useless short sequences as well. Using confidence measures lowers the number of output sequences but still results in short sequences.

Removing the short sequence bias is a key issue in finding meaningful patterns. Possible approaches to this issue along with their drawbacks were described in Chapter 2. We present REEF (**RE**solving l**E**ngth bias in **F**requent sequence mining). REEF is an algorithm for mining frequent sequences that normalizes the support of each candidate sequence with a length adjusted z-score. The use of the z-score in REEF eliminates statistical biases towards finding shorter patterns, as we will demonstrate. However, it raises two challenges to the scalability of the approach: First, z-score normalization lacks the anti-monotonic property used in support based measures, and thus supposedly forces explicit enumeration of every sequence in the database. This renders useless any pruning of candidate sequences, the basis for scalable itemset mining algorithms, such as SPADE [66]. Second, calculation of the necessary statistics for computation of the z-score (the average and standard deviation of the support, for each sequence length) seems to require computing the support for every sequence, again eliminating any benefit from pruning. REEF addresses both of these challenges.

First, in order to allow for pruning candidate sequences, we introduce a bound on the z-score of future sequence expansions. The z-score bound enables pruning in the mining process to provide scalability while ensuring closure. We use this bound with an enhanced SPADE-like algorithm to efficiently search for sequences with high z-score values, without enumerating all sequences. Our experimental evaluation indicates that this bound assists the speedup substantially.

Second, in order to remove the need for a pre-pruning pass over the entire database, we advocate using a sample of the database to estimate the necessary statistics for z-score calculations. Such sampling causes a distortion in the estimated values [68]. We analyzed this distortion, and present a formula for performing the correction.

The structure of this chapter is as follows: In Section 4.1 we present our notation and introduce the Norm-Frequent Sequence Mining Problem. We describe the scalability issues in Norm-Frequent Sequence Mining and explain how to resolve them. The REEF algorithm is introduced in Section 4.1.5, followed by experimental evaluation in Section 4.2, and discussion in Section 4.3.

4.1 Problem Description

This section will define the *norm-frequent* Sequence Mining problem. The notation and the traditional *Frequent* Sequence Mining problem are introduced in Section 4.1.1. We will define the *norm-frequent* Sequence Mining problem in Section 4.1.2. *Norm-frequent* Sequence Mining solves the short sequence bias present in traditional *Frequent* Sequence Mining. We explain why the scalability is hindered by the naive implementation of normalized support and how this is resolved. Section 4.1.3 addresses the first scalability difficulty by introducing a bound that enables pruning in the candidate generation process. Section 4.1.4 describes the use of a sampling process used to eliminate the second scalability problem. Finally in Section 4.1.5 we bring all parts together to compose the REEF algorithm.

4.1.1 Notation and Frequent Sequence Mining

We use the following notation in discussing Norm Frequent Sequence Mining.

event Let $I = \{I_1, I_2...I_m\}$ be the set of all *items*. An *event* (also called an *itemset*)

is a non-empty unordered set of *items* denoted as $e = \{i_1, ..., i_n\}$ where $i_j \in I$ is an item. Without loss of generality we assume they are sorted lexicographically. For example, $e = \{ABC\}$ is an event with items A B and C.

sequence A *sequence* is an ordered list of *events*, with a temporal ordering. The sequence $s = (e_1 \rightarrow e_2 \rightarrow ... \rightarrow e_q)$ is composed of q events. If event e_i occurs before event e_j, we denote it as $e_i \rightarrow e_j$. e_i and e_j do not have to be consecutive events and no two *events* can occur at the same time. For example in the sequence s=$(\{ABC\} \rightarrow \{AE\})$ we may say that $\{ABC\}$ occurs before $\{AE\}$.

sequence size and length The *size* of a sequence is the number of events in a sequence, e.g. $size(\{ABC\} \rightarrow \{ABD\}) = 2$. The *length* of a sequence is the number of items in a sequence including repeating items. A sequence with length l is called an *l-sequence*, e.g. $length(\{ABC\} \rightarrow \{ABD\}) = 6$.

subsequence and contain A sequence s_i is a *subsequence* of the sequence s_j, denoted $s_i \preceq s_j$, if $\forall e_k, e_l \in s_i, \exists e_m, e_n \in s_j$ such that $e_k \subseteq e_m, e_l \subseteq e_n$ and if $e_k \rightarrow e_l$ then $e_m \rightarrow e_n$, where all es refer to events. We say that s_j *contains* s_i if $s_i \preceq s_j$. For example, $(\{AB\} \rightarrow \{DF\}) \preceq (\{ABC\} \rightarrow \{BF\} \rightarrow \{DEF\})$.

database The database D used for sequence mining is composed of a collection of sequences.

support The *support* of a sequence s in database D is the proportion of sequences in D that *contain* s. This is denoted $supp(s, D)$.

This notation allows the description of multivariate sequence problems. The data is sequential in that it is composed of ordered events. The ordering is kept within the subsequences as well. The multivariate property is achieved by events being composed of several items. The notation enables discussion of mining sequences with gaps both

in events and in items, as long as the ordering is conserved. The mined sequences are sometimes called patterns.

In traditional support based mining, a user specified minimum support called *minsup* is used to define frequency. A *frequent* sequence is defined as a sequence with a support higher than *minsup*, formally defined as follows:

Definition 4.1.1 (Frequent) *Given a database D, a sequence s and a minimum support* minsup*. s is* frequent *if* $supp(s, \mathrm{D}) > minsup$*.*

The problem of frequent sequence mining is described as searching for all the *frequent* sequences in a given database. The formal definition is:

Definition 4.1.2 (Frequent Sequence Mining Problem) *Given a database D, and a minimum support* minsup*, find all the* frequent *sequences.*

In many support based algorithms such as SPADE [66], the mining is performed by generating candidate sequences and evaluating whether they are frequent. In order to obtain a scalable algorithm a pruning is used in the generation process. The pruning is based on the anti-monotonic property of support. This property ensures that support does not grow when expanding a sequence i.e. $supp(AB \rightarrow C) \geq supp(AB \rightarrow CD)$. This promises that candidate sequences that are *not frequent* will never generate *frequent* sequences, and therefore can be pruned.

Frequent sequence mining seems to be a solved problem with a scalable algorithm. However it suffers from a bias towards mining short subsequences. There is great interest at being able to mine sequences of all lengths. We do not want to force the mining of long sequences, or settle for short sequences. We would like to provide and algorithm that enables mining sequences of all lengths.

4.1.2 Norm-Frequent Sequence Mining

In this section we define the problem of *norm-frequent* Sequence Mining. *Norm-frequent* Sequence Mining resolves the short sequence bias present in *Frequent* Sequence Mining, by normalizing support of sequences for their length. Based on the normalized support we define a normalized frequency measure, and use it in the mining process.

We use the statistical z-score for normalization. The z-score for a sequence of length l is defined as follows:

Definition 4.1.3 (Z-score) *Given a database D and a sequence s. Let $l = len(s)$ be the length of the sequence s. Let μ_l and σ_l be the average support and standard deviation of support for sequences of length l in D. The* z-score *of s denoted $\zeta_l(s)$ is given by $\zeta_l(s) = \frac{supp(s)-\mu_l}{\sigma_l}$.*

We use the z-score because it normalizes the support measure relative to the sequence length. Traditional mining, where support is used to define frequency, mines sequences that appear often relative to **all** other sequences. This results in short sequences since short sequences always appear more often than long ones. Using the z-score normalization of support for mining finds sequences that are frequent relative to other **sequences of the same length**. This provides an even chance to sequences of all lengths to be found frequent.

Based on the definition of z-score for a sequence we define a sequence as being *norm-frequent* if the z-score of the sequence is among the top z-score values for sequences in the database. The formal definition follows:

Definition 4.1.4 (Norm-Frequent) *Given a database D, a sequence s of length l and an integer k. Let Z be the set of the k highest z-score values for sequences in D, s is* norm-frequent *if $\zeta_l(s) \in Z$. In other words, we perform top-K mining of the most norm-frequent sequences.*

We introduce the problem of *norm-frequent* Sequence Mining. This new problem is defined as searching for all the *norm-frequent* sequences in a given database. The formal definition follows:

Definition 4.1.5 (Norm-Frequent Sequence Mining Problem) *Given a database D and an integer n, find all the* norm-frequent *sequences.*

Unfortunately the z-score normalization test spoils the anti-monotonic property: We **cannot** determine that $\zeta_3(AB \rightarrow C) \geq \zeta_4(AB \rightarrow CD)$. Therefor pruning becomes difficult; we cannot be sure that the z-score of a candidate sequence with length l will not be improved in its extensions of length $l+1$ or in general $l+k$ for some positive k. Therefor we cannot prune sequences based on z-score and ensure we will find all *norm-frequent* sequences. This creates a problem since without pruning our search space becomes unscalable.

Another problem with performing *norm-frequent* Sequence Mining is that the values for μ_l and σ_l must be obtained for sequences of all lengths prior to the mining process. This imposes multiple passes over the database and completely spoils scalability.

These two important scalability issues are addressed and solved in Sections 4.1.3 and 4.1.4 resulting in a scalable frequent sequence mining algorithm that overcomes the short sequence bias.

4.1.3 Bound used for Pruning

As we explained in Section 4.1.2 using the z-score for *norm-frequent* mining prevents the use of the pruning methods used in *frequent* sequence mining such as SPADE [66]. We propose an innovative solution that solves the scalability problem caused by the inability to prune. Our solution is to calculate a bound on the z-score of sequences, that can be expanded from a given sequence. This bound on the z-score of future

expansions of candidate sequences is used for pruning. We define the bound and then explain how it is used.

Z-score was defined in definition 4.1.3. The bound on z-score is defined in definition 4.1.6, where $maxsupp(s)$ is the maximum support of sequences s' generated from s. We describe how it is obtained shortly.

Definition 4.1.6 (Z-score-Bound) *Given a database D and a sequence s. Let $l = len(s)$ be the length of the sequence s. Let μ_l and σ_l be the average support and standard deviation of support for sequences of length l in D. The* z-score-bound *of s denoted $\zeta_l^B(s)$ is given by $\zeta_l^B(s) = \frac{maxsupp(s)-\mu_l}{\sigma_l}$.*

We know that support is anti-monotonic, therefore as the sequence length grows support can only get smaller. Given a candidate sequence s of length l with a support of $supp(s)$ we know that for all sequences s' generated from s with length $l' > l$ the maximal support $maxsupp(s') = supp(s)$. We can calculate the bound on z-score, $\zeta_l^B(s)$, for all possible extensions of a candidate sequence. The real z-score of an extension may be lower than the bound, but cannot be higher. The ability to calculate this bound on possible candidate extensions is the basis for the pruning.

In order to mine *frequent* or *norm-frequent* sequences candidate sequences are generated and evaluated. In traditional *frequent* sequence mining there is only one evaluation performed on each sequence. If the sequence is found to be *frequent* it is both saved in the list of *frequent* sequences and expanded to generate future candidates, if it is not *frequent* it can be pruned (not saved and not used for generating candidates). For *norm-frequent* mining we perform two evaluations for each sequence. The first is to decide whether the proposed sequence is *norm-frequent*. The second is to determine if it should be expanded to generate more candidate sequences for evaluation. There are two tasks since z-score is not anti-monotonic and a sequence that is not *norm-frequent* may be used to generate *norm-frequent* sequences. This

second task is where the bound is used for pruning. The bound on future expansions of the sequences is calculated for all possible lengths. If the bound on the z-score for all possible lengths is lower than the top n z-scores then no possible expansion will be *norm-frequent* and the sequence can be pruned from the generation process. If for one or more lengths the bound is high enough to be *norm-frequent* the sequence is used to generate candidates that are then evaluated in order to determine if they really are *norm-frequent*.

Using the bound enables pruning of sequences that will definitely not be able to generate *norm-frequent* candidates. The pruning enabled by using the bound resolves the first scalability issue. What remains to describe is how the second scalability problem is resolved. The values for μ_l and σ_l are used both in the z-score calculation and the bound calculation. Section 4.1.4 describes how μ_l and σ_l are obtained while still ensuring a scalable algorithm.

4.1.4 Sampling for Norm-Frequent Mining

Norm-frequent mining uses the z-score defined in Definition 4.1.3 and the bound described in Definition 4.1.6. Both these measures make use of the average and standard deviation of support for each subsequence length (μ_l and σ_l). We must calculate these values prior to the sequence mining. The naive way to calculate these values would be to generate all possible subsequences and calculate these measures. However this is obviously irrelevant as making a full expansion completely defeats the purpose of mining with the z-score pruning.

Therefor we propose extracting a small sample of the database and calculating these values on the sample. For the sample, full expansion is feasible and generates the necessary measures while ensuring scalability.

However there is a problem that arises with the sampled measures. They do not reflect the full database measures correctly. It has been shown by [68, 36, 45, 37], that

there is a distortion, also termed overestimation, in the values of support calculated on a sample of a database relative to support calculated over a full database. Similarly the average and standard deviation of support suffer a distortion in the sampled data.

Effects of Sampling Distortion

We use Fig. 4.1 to demonstrate how the distortion affects sequence mining. The norm-freq sequences that are mined using z-score and calculated with statistics from the full database are displayed in column 1. The top most stripes (light) represent the most *norm-frequent* sequences and the bottom (dark) represent sequences that are *not norm-frequent* (rare). Column 3 shows the *norm-frequent* sequences discovered using averages and standard deviations for z-score calculation from the sampled database, the colors match the coloring in the full database, the ordering is based on sampled results. One notices that the order is confused and rare sequences in dark greys show up relatively high in the list. *Norm-frequent* (light) sequences are pushed down as rare. The black stripes at the side of the column represent sequences that didn't appear at all in the *norm-frequent* list when using the full data set and appeared when using the sampling. It is obvious that sequences are shifting around and *norm-frequent* sequences are being chosen as rare and vice versa. Therefor the distortion badly affects sequence mining. In column 2 we use the correction displayed in the next section and improve this shifting. The rare sequences show up further down with the correction than without, as do the candidates that didn't appear in the original list. Although this is just an example on one small set of data it conveys the effects of the distortion and the correction.

Chernoff Bounds and Hoeffding Inequalities

We would like to evaluate how far off the sampled statistics are from the real statistics. One might suggest using Chernoff bounds as in [68, 57] or Hoeffding inequalities as

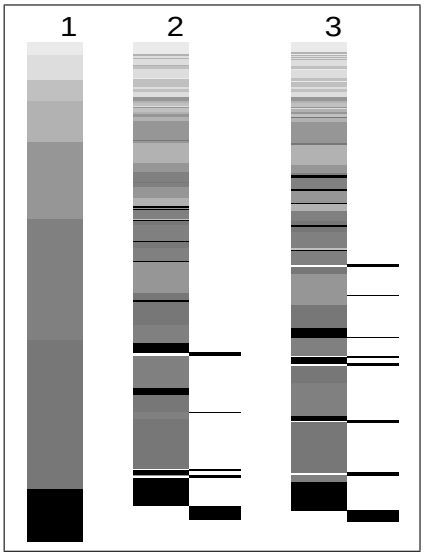


Figure 4.1: Sampling distortion effect.

in [45] for this task. In [68, 57, 45] the aim is to show how far off sampled support is from real support for a single subsequence. The appearance of a subsequence in each sequence in the sample is described as a random variable with a Bernoulli distribution. These random variables are **independent**, and the Chernoff bounds or Hoeffding inequalities can be used. The scenario we are using is different. Instead of looking at the accuracy of the support on the sampled data we are looking at the **average** and **standard deviation** of support for a subsequence of a specific length. Unlike the application of Chernoff and Hoeffding bounds in [68, 57, 45], where the random variable was independent, the random variable in our setting is the average of support of sequences for a given length. This random variable is strongly **dependent** and therefore the known bounds are problematic to apply. There are also situations where although Chernoff bounds can be applied, it is problematic to apply them because a very large sample of the database is needed, as in [68]. For cases where Chernoff and Hoeffding bounds cannot be applied, or situations where one chooses not to apply them we propose a method of distortion correction.

Sampling Distortion Correction Method

We introduce a method for correcting the distortion that can be used for any data set. This method finds the model of the distortion for various input sequence lengths and sample rates using the non-linear regression function *nlsfit()* in the R Project for Statistical Computing [44]. Once we have modeled the distortion, correcting it is immediate. The model provides an equation that determines the exact distortion value of average support or standard deviation for a given input sequence length and sample rate. A simple inverse multiplication provides the corrected value.

In order to perform the regression we must propose functions and then perform the non-linear regression to set the parameters. We list the equations we propose using based on our experimental experience as described in Section 4.2.2. The variables in the equations are len (length of the input-sequences) and smp (sampling rate). The coefficients that are determined in the non-linear regression are a, b, c, d. The functions we propose using are:

$$a \times (len - 1)^{b \times log(smp)} + c \times smp \tag{4.1}$$

$$a \times (len)^{b \times smp + c} + d \times smp \tag{4.2}$$

$$a \times (len - 1)^{b \times smp + c} + d \times smp \tag{4.3}$$

$$(len - 1)^a + e \times smp^b + c \times len + d \times smp \tag{4.4}$$

For the standard deviation of support we perform distortion correction in a similar fashion using the following equations for approximation.

$$a * (smp^b) \tag{4.5}$$

$$(smp^b) + a \tag{4.6}$$

A tool such as the R Project for Statistical Computing [44] is used to find the correct parameters for these equations as demonstrated in detail in Section 4.2.2.

Once we have found the equations that represent the distortion for average support and standard deviation of support, for a certain type of data set, correction of this distortion is simple. For new data in these sets we can select any sample rate and calculate the distortion correction of average and standard deviation for each possible sequence length. We multiply the sampled values by the inverse of the distortion and use the results as the average and standard deviation of support in the z-score calculation for *norm-frequent* mining.

We found the proposed equations to be general and provide good approximations for different data sets, as shown in in Section 4.2.2, and therefore suggest they can be used for other data sets as well. However for data sets where these equations do not provide good approximations the same method we used can be applied while using different equations.

Now that we have presented the full method for sampling the data set and calculating the values for μ_l and σ_l we have a complete scalable algorithm that mines *norm-frequent* sequences without a bias to short sequences, Section 4.1.5 puts all the pieces together and describes the full algorithm.

4.1.5 REEF Algorithm

In this section we combine all the components we have described in the previous sections. We describe the implementation of REEF (**RE**solving l**E**ngth bias in **F**requent sequence mining). The REEF algorithm is composed of several phases. The input to REEF is a database of sequences and an integer *'best'* determining how many *norm-frequent* sequences will be found. The output of REEF is a set of *norm-frequent* sequences. Initially a sampling phase is performed to obtain input for the later phases. Next we perform the candidate generation phase. First *norm-frequent* 1-sequences

and 2-sequences are generated. Once 2-sequences have been generated an iterative process of generating candidate sequences is performed. The generated sequences are evaluated and if found to be *norm-frequent* are placed in the output list of *norm-frequent* sequences. These sequences are also examined in the pruning process of REEF in order to determine if they should be expanded or not.

Sampling Phase The sampling phase is performed as a preprocessing of the data in order to gather statistics of the average and standard deviation of support for sequences of all possible lengths. This stage uses SPADE [66] with a *minsup* of 0 to enumerate all possible sequences in the sampled data and calculate their support. For each length the support average and standard deviation are calculated. These values are distorted and corrected values are calculated using the technique described in section 4.1.4. These corrected values provide the average support μ_l and standard deviation of support σ_l that are used in z-score calculation and the bound calculation.

Candidate Generation Phase The candidate generation phase is based on SPADE along with important modifications. As in SPADE we first find all 1-sequence and 2-sequence candidates. The next stage of the candidate generation phase involves enumerating candidates and evaluating their frequency.

We make two modifications to SPADE. The first is moving from setting a *minsup* to setting the *'best'* value. *'best'* determines the number of z-score values that *norm-frequent* sequences may have. Note that there may be several sequences with the same z-score value. The reason for this modification is that z-score values are meaningful for comparison within the same database but vary between databases. From our experience setting the *'best'* value is more intuitive than selecting a min-z-score threshold. The algorithm could be easily modified to use a min-z-score threshold.

The second and major change we make is swapping *frequency* evaluation with

```
1: for all x is a prefix in S do
2:    T_x = ∅,F_R = ∅
3: for all items A_i ∈ S do
4:    for all items A_j ∈ S, with j ≥ i do
5:       R = A_i ⋁ A_j (join A_i with A_j)
6:       for all r ∈ R do
7:          if ζ_l(r) > ζ(a seq s in F_R) then
8:             F_R = F_R ⋃ r\s //replace s with r
9:          for all k = l + 1 to k = maxlen do
10:            if ζ_k^B(r) > Zscore(a seq s in F_R) then
11:               if A_i appears before A_j then
12:                  T_i = T_i ⋃ r
13:               else
14:                  T_j = T_j ⋃ r
15:   enumerate-Frequent-Seq-Z-score(T_i)
16:   T_i = ∅
```

Figure 4.2: Enumerate-Frequent-Seq-Z-score(S). Where S is the set of input sequences we are mining for frequent subsequences (patterns). A set of *norm-frequent* subsequences is returned. F_R is a list of sequences with the top *'best'* z-scores

norm-frequency evaluation as appears in lines 7-8. In other words for each sequence s replace the test of is $supp(s, D) > minsup$ with the test of is $\zeta_l(s) \in Z$ where Z is the set of the *'best'* highest z-score values for sequences in D. This replacement of the frequency test with the *norm-frequency* test is the essence of REEF and our main contribution.

The improved version of sequence enumeration including the pruning is presented in Fig. 4.2 and replaces the enumeration made in SPADE. The joining of l-sequences to generate *l+1*-sequences ($A_i \bigvee A_j$ found in line 5) is performed as in SPADE [66].

Pruning Phase using Bound Obviously REEF cannot enumerate all possible sequences for *norm-frequency* evaluation. Furthermore as we discussed in Section 4.1.2 the z-score measure is not anti-monotonic and cannot be used for pruning while ensuring that *norm-frequent* candidates are not lost. In Section 4.1.3 we introduced

the bound on z-score that is used for pruning.

The pruning in REEF calculates $\zeta_{l'}^{B}(s')$ for all possible sequences s' with lengths $l' > l$ generated from s. The key to this process is the fact that this does not require generation of the extensions s'. It is enough to know the $supp(s)$ (as this is the $maxsupp(s')$ for all s' generated from s), μ_l and σ_l for all $l' > l$. If for any length $l' > l$ we find that $\zeta_{l'}^{B}(s) \in Z$ (in the list of *'best'* z-scores) we keep this sequence for candidate generation, if not then we prune it. Using the bound for pruning reduces the search space while ensuring closure or in other words ensuring all frequent sequences are found. The pruning is performed as part of the enumeration described in algorithm Fig. 4.2, in lines 9-10. This pruning is the key to providing a **scalable** *norm-frequent* algorithm.

4.2 Evaluation

In this section we present an evaluation of REEF on several data sets. We begin by describing the data sets we used in Section 4.2.1. Section 4.2.2 describes how we found the correct parameters for distortion correction in the sampling unit. The next three sections compare the performance of REEF to that of SPADE. In Section 4.2.3 we compare runtime of the algorithms and justify the use of the bound that was introduced in Section 4.1.5. Section 4.2.4 will show that *norm-frequent* mining overcomes the short sequence bias present in *frequent* mining algorithms. Finally in Section 4.2.5 we will provide evidence that the sequences mined with REEF are more interesting than sequences mined with SPADE.

4.2.1 Data Sets

The evaluation is performed on four data sets. One of these data sets is a synthetic data set, three use real world sequential data.

Syn is the synthetic data generated with the IBM QUEST Synthetic Data Generator [2]. QUEST generates data for various data mining tasks, including frequent sequence mining. We generated sequences with the following parameters: Number of customers in database = 1000, Average sequence length = 3, Average transaction length = 3, Number of items = 10, all other settings are the default settings. The tests in the evaluation are performed on 5 synthetic sets with these parameters.

TEXT is a corpus of literature of various types. We treat the words as sequences with letters as single item events. We removed all formatting and punctuation from text thus we get a long sequence of letters. Mining this sequential data for frequent sequences produces sequences of letters that may or may not be real words. The reason we chose to mine text in this fashion is to demonstrate how interesting the frequent sequences are in comparison to norm-frequent sequences by testing how many real words are discovered. In other words, we use real words from the text as ground truth against which to evaluate the algorithms.

We use three sets of textual data, one is from Lewis Carroll's "Alice's Adventures in Wonderland" [14], another is Shakespeare's "A Midsummer Night's Dream" [53] and the third is a Linux installation guide [21]. Evaluation is performed on segments of the corpus. Each test is performed on a five segments.

UPD (User Pattern Detection). We perform identification and authentication of a computer user based on mouse and keyboard activity. Identification and authentication often use only password and user ID as in [69] and [23] or a predefined or single task as in [62] [42]. Our setting involves using data from all applications, this increases the difficulty in building a good classifier but provides a better solution for intruder detection. Some detection algorithms use either mouse [4] or keyboard [27] data. Using both mouse and keyboard data, as we do, should provide a broader

and more secure solution, since it becomes harder for an imposter to imitate both patterns.

The UPD data set logs keyboard and mouse activity of users on a computer. Sequences mined from the UPD data can be used to model specific users and applied to security systems as in [4], [27] and [46]. The data is collected throughout the whole work session and not just at login. Each activity is logged along with the time and date it occurs. The data is then converted into the following events: pressing a key, time between key presses, key-name, mouse click, mouse double click, time between mouse movements. For each session the events are saved in sequences.

The data acquired regarding times is continuous data, and must be discretized before it can be fed to REEF. Time between key presses is discretized as being long (over 200 milliseconds) short (under 80 milliseconds) or medium. Times for between mouse clicks are discretized as long (over 100 milliseconds) short (under 50 milliseconds) or medium. The experiments are run on 11 user sessions. More details on this data set can be found in appendix A.

Zapping The ability to provide personalized television services is broadly discussed [10, 7, 25, 33, 13] and is of much interest to various content providers. This dataset is used to perform identification of the viewer watching TV in order to provide personalized content. This type of identification has been discussed both by academia and in industry. A patent had been issued by INVIDI [18] on this topic. INVIDI use the remote control usage to determine if someone is watching the TV. They only want to know if there is an active viewer but do not use this for user identification as we do. Another type of approach such as [30] use devices with sensors, fingerprint ID etc instead of a regular remote control to identify the current viewer. We do not require any active cooperation from the viewer in order to perform identification as these solutions do. Our solution requires software addition and no extra hardware in

order to perform the identification and is therefore easier to deploy.

The Zapping dataset is composed of data that we gathered on remote control usage. In each household members were asked to identify themselves as they begin watching TV, by pressing a designated button on the remote, and then the "zapping sequence" is saved, in other words the buttons they pressed on the remote while they were watching. This sequence is converted into the following events: Button pressed, Time passed since last activity and Time of day. Time of day is divided into 5 intervals: MORNING, AFTERNOON, EVENING, NIGHT and SLEEP. Time passed since last activity is discretized into 20 intervals, each interval is composed of 500 milliseconds.

Each zapping session generates a single long sequence. Evaluation is performed on 10 sets. More details on this data set can be found in appendix A.

Evaluation settings: For all these data sets the input is composed of long sequences. In order to use REEF these sequences are cut into smaller sequences using a sliding window thus creating manageable sequences for mining. The size of the sliding window is termed *input sequence length* in our results.

The comparison made between REEF to SPADE is delicate since SPADE uses *minsup* to define how many sequences to mine whereas REEF uses *'best'* as described in Section 4.1.5. Adjusting these settings changes the runtime and other output. Although these parameters are similar in nature they cannot be set to be exactly the same for experiments.

We consistently use a single setting of *minsup=1%* and *'best'=50* throughout all experiments. This allows an investigation of the quality and lengths of mined sequences achieved for specific runtime. Throughout all the experiments presented here we set a sample rate of 10% for the preprocessing sampling component.

4.2.2 Sampling Distortion Correction

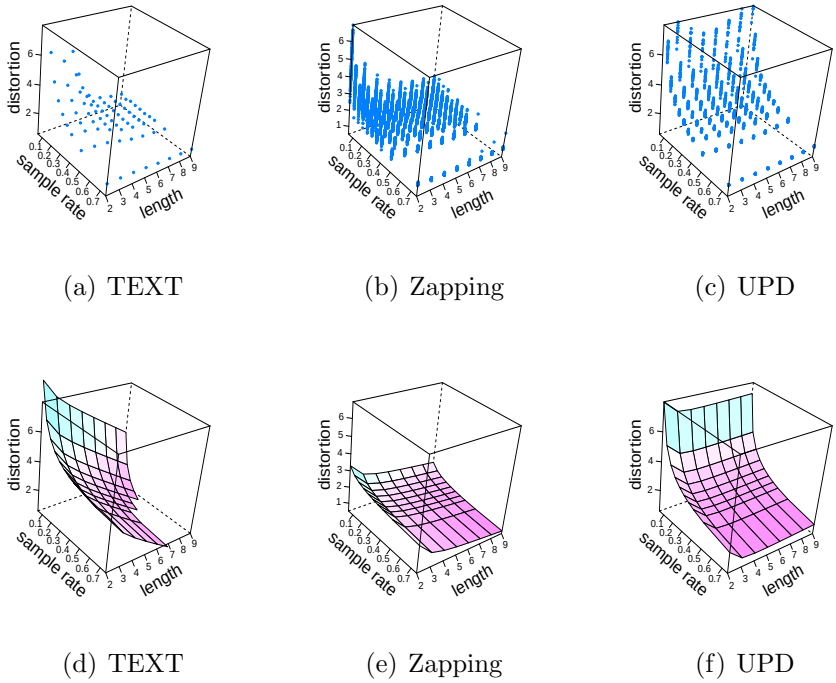
We will demonstrate how to perform the distortion correction described in Section 4.1.4 for several data sets. We found a single equation to model distortion for all the data sets we investigated. Although this does not imply that the same model fits all possible data sets, it is a strong indication that this may be the case. For data sets where this does not hold, the same method we used can be applied to find other models. The data sets we used are the TEXT data set the Zapping data set and the UPD dataset. These were all described in detail in Section 4.2.1.

Fig. 4.3 (a),(b),(c) displays the distortion ratio between sampled average support to full data average support on all three data sets. The data used for this analysis is excluded from the experimental evaluation performed in Sections 4.2.3, 4.2.4 and 4.2.5. We used approximately half the data for this analysis and half for the experimental evaluation. Each point is an instance of the dataset. The distortion is calculated on each instance for various input sequence lengths and sample rates. The distortion obviously has an orderly structure that we want to find.

We modeled the distortion using non-linear regression. We used R Project for Statistical Computing [44] in order to find a general formula for calculating the correction factor. There are two correction parameters that we are looking for, one for the average support, the other for the standard deviation of support.

We first describe the average support correction. We noticed that when we set the sample rate, the distortion ratio follows a nonlinear function of the length, shown in Fig. 4.4. On the other hand if we set the length then the distortion ratio follows a nonlinear function of the sample rate, shown in Fig. 4.5. Therefore the distortion of average support is dependent both on length and on sample rate and we are looking for a function $f(len, smp)$ where len is the length of a sequence and smp is the sample rate.

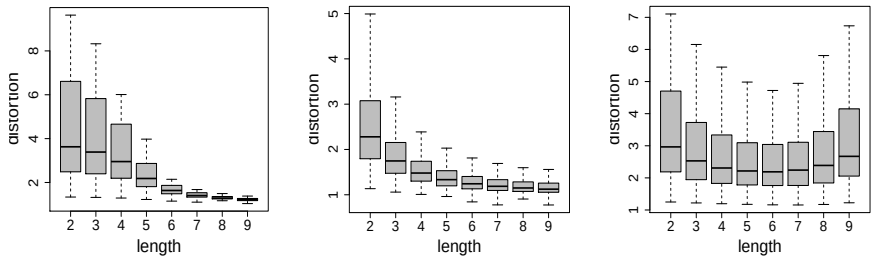
In previous research [46] we investigated the distortion on the Zapping data alone.



(a) TEXT (b) Zapping (c) UPD

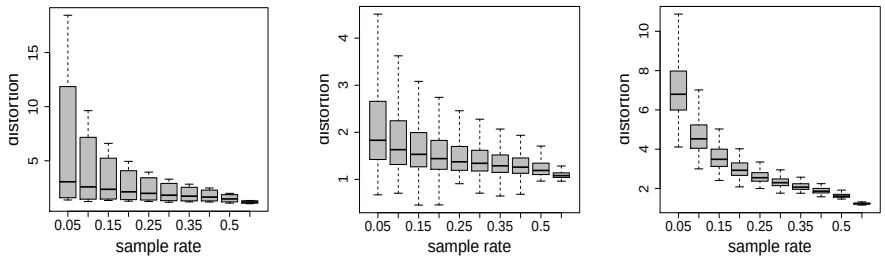
(d) TEXT (e) Zapping (f) UPD

Figure 4.3: Distortion ratios of average support on sampled data in (a), (b) and(c). Regression surfaces of equation 4.4 in (d), (e) and (f) .



(a) TEXT (b) Zapping (c) UPD

Figure 4.4: Length cross cut of distortion ratio for average support.



(a) TEXT (b) Zapping (c) UPD

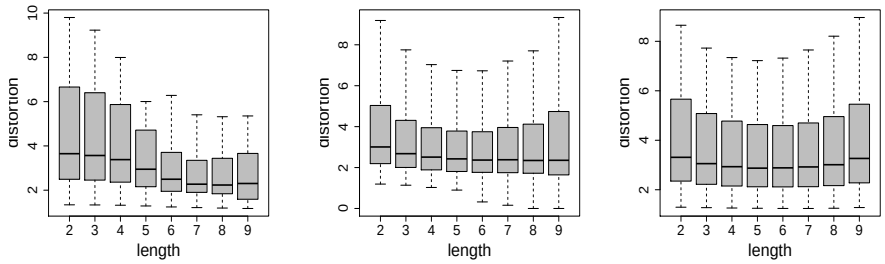
Figure 4.5: Sample rate cross cut of distortion ratio for average support.

We tried to build a combination of the power and logarithmic functions that we saw when looking at each variable, into a single function. This led us to investigating equations 4.1, 4.2 and 4.3 in Section 4.1.4. However when we tried performing regression for other data sets we discovered that for UPD these were not the best candidates, and did not even converge on the TEXT data. We suspect over-fitting of the regression on the Zapping data. Realizing that the shape of the distortion is reminiscent of a stretched paraboloid we tried regression with equation 4.4 in Section 4.1.4 and found that this best suits all three data sets and was therefore selected as the distortion model. The regression surfaces for Equation 4.4 in Section 4.1.4 appear in Fig. 4.3 (d),(e),(f). Values of the parameters for non linear least of squares regression appear in table 4.1.

Standard deviation of distortion is linear relative to length see Fig. 4.6, and is a nonlinear function of sample rate as shown in Fig. 4.7. Therefore the only variable involved is the sample rate. The sampling distortion correction we found for zapping in [46] fits the UPD and TEXT data as well. The equations we tested are Equations 4.5 and 4.6 in Section 4.1.4. We choose 4.5. Regression parameter values appear in table 4.2.

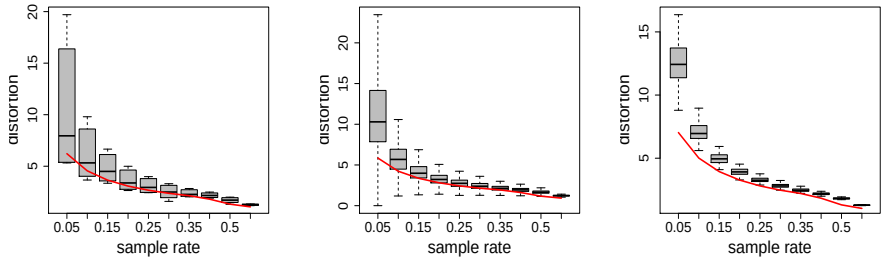
data set	func	RSE	a	b	c	d	e
Zapping	4.1	0.3975	3.561935	0.183471	-3.438119		
Zapping	4.2	0.3596	5.109377	0.751144	-0.528501	-5.712356	
Zapping	4.3	0.3391	3.789649	0.703963	-0.465705	-4.124942	
Zapping	4.4	0.3998	-1.664660	-0.229604	-0.067215	-0.087310	1.226132
UPD	4.1	1.21	4.935761	-0.009716	-6.576244		
UPD	4.2	1.102	8.852121	0.648309	-0.302588	-15.3049	
UPD	4.3	1.141	6.932194	0.476740	-0.220498	-10.9230	
UPD	4.4	0.5916	-2.417082	-0.665367	0.037497	-0.392893	0.928968
TEXT	4.4	1.899	-1.416	-0.409	-0.559	1.387	2.719

Table 4.1: Regression parameter values for average support



(a) TEXT (b) Zapping (c) UPD

Figure 4.6: Length cross cut of distortion ratio for standard deviation.



(a) TEXT (b) Zapping (c) UPD

Figure 4.7: Sample rate cross cut of distortion ratio for standard deviation.

num	func	RSE	a	b
Zapping	4.5	2	0.928601	-0.799307
Zapping	4.6	2	-0.101002	-0.776648
UPD	4.5	0.6055	1.027204	-0.835350
UPD	4.6	0.6057	0.043373	-0.843472
TEXT	4.5	2.053	1.059	-0.768

Table 4.2: Regression parameter values for standard deviation of support.

4.2.3 REEF Runtime and Bound Pruning

Although the main focus of REEF is not speed but rather output quality, we must show that REEFs' runtime is comparable with existing algorithms. We compare the runtime for two versions of REEF to SPADE. REEF refers to the full algorithm described in Section 4.1.5. NB-REEF refers to the same algorithm but without the use of the bound, or in other words without pruning. Results for all data sets appear in Fig. 4.8. There are four types of data sets as described in 4.2.1, however the TEXT dataset is composed of three sets of textual data , thus in the results in Fig. 4.8 there are 6 graphs. The x-axis represents input-sequence length. For the synthetic data we had full control over input sequence length and thus present results for all values. For the real data sets the input sequence length is controlled by the number of attributes in an event. This results in varying values along the x-axis for the results. The y-axis displays runtime of the algorithm in seconds. We tested the runtime is for various input-sequence lengths. Each point on the graph is the average of five runs.

The first important observation to make is the importance of the pruning bound. For all data sets the pruning noticeably reduces runtime and is an important component of REEF. This is particularly noticeable on the synthetic data in Fig. 4.8(d). This difference grows along with input sequence length and therefore becomes more and more important as input length grows.

The other important result is that the REEF runtime is comparable with that of SPADE. Although SPADE is faster than REEF they are close in runtime. The reason

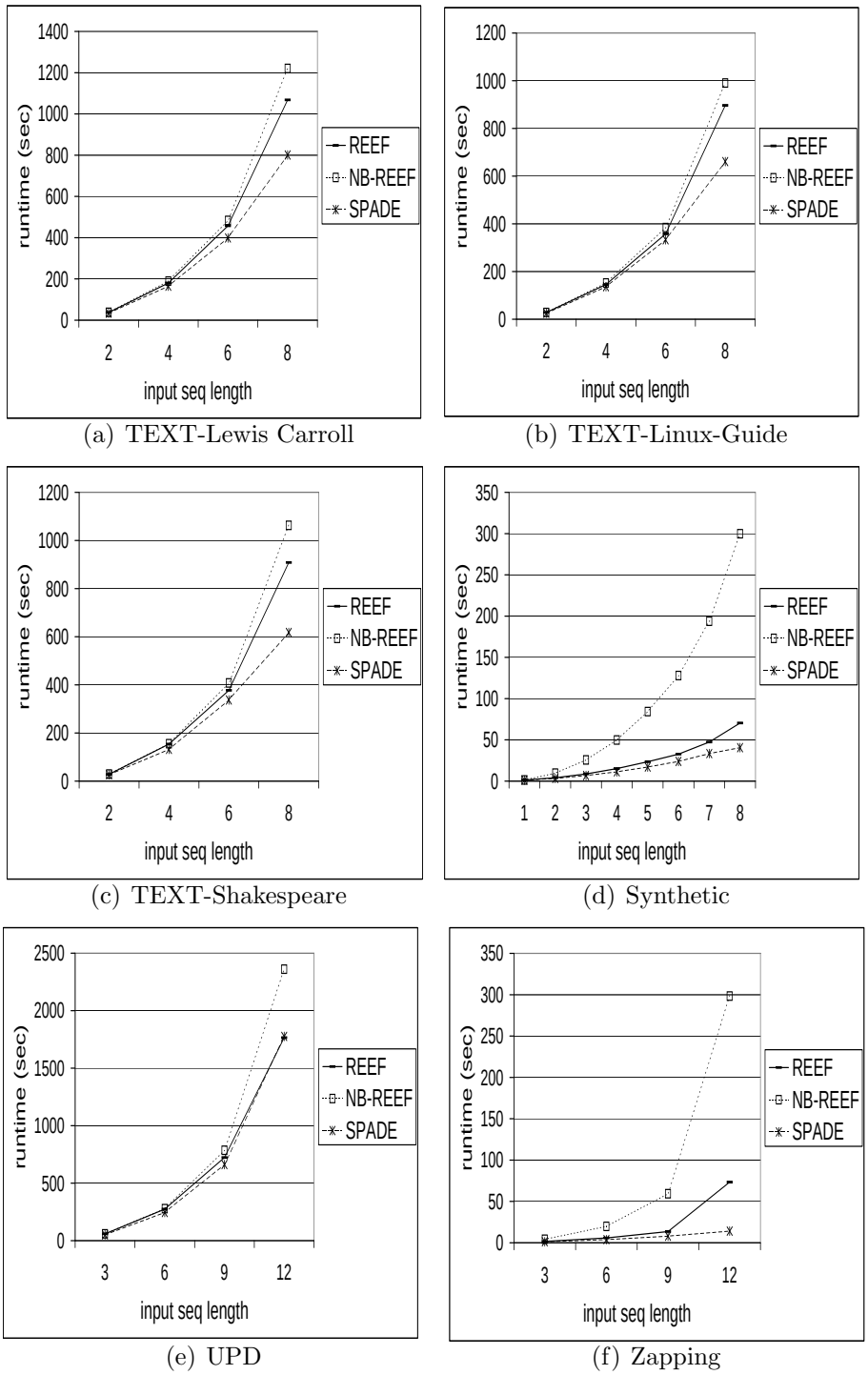
SPADE is often faster than REEF is because *minsup* provides a tighter pruning bound than the one we use in REEF. However faster may not be better. The tight pruning results in the creation of short sequences. In the next section we show that there is a tradeoff between runtime to the length of mined sequences, and show how REEF although slightly slower than SPADE has better performance. By overcoming the short sequence bias REEF produces a better distributed set of mined sequences.

4.2.4 Resolving Length Bias in Frequent Sequence Mining

In this section we establish how REEF successfully overcomes the short sequence bias that is present in the frequent sequence mining techniques. We performed *frequent* sequence mining with SPADE and *norm-frequent* sequence mining with REEF. We compared the lengths of the mined sequences for both algorithms. The results are displayed in Fig. 4.9. Results are shown for the Syn, UPD, Zapping and three TEXT data sets. The x-axis shows the lengths of the mined sequences. The y-axis displays the percentage of sequences found with the corresponding length. For each possible length we counted the percentage of mined sequences with this length.

The synthetic data set in Fig. 4.9(d) displays the clearest description of the algorithmic behavior. While SPADE outputs mainly sequences with a length of 2, some with a length of 3, very little with a length of 4 and no longer sequences, REEF outputs sequences with lengths varying from 2 to 6 and with a bell shaped distribution. REEF succeeds in capturing the real nature of the synthetic data and the correct distribution of sequence length.

In the TEXT data set it is known that the average length of words in English is around 5 letters. The text results on all three corpus show how SPADE mines mainly short sequences, while REEF manages to mine a broader range of sequence lengths much closer the known average of 5 letters in a word. This is displayed in in Fig. 4.9(a),(b) ,(c). In the next section we will count how many of these sequences are



(a) TEXT-Lewis Carroll (b) TEXT-Linux-Guide

(c) TEXT-Shakespeare (d) Synthetic

(e) UPD (f) Zapping

Figure 4.8: Runtime. Comparing REEF (with bound), NB-REEF (without bound) and SPADE.

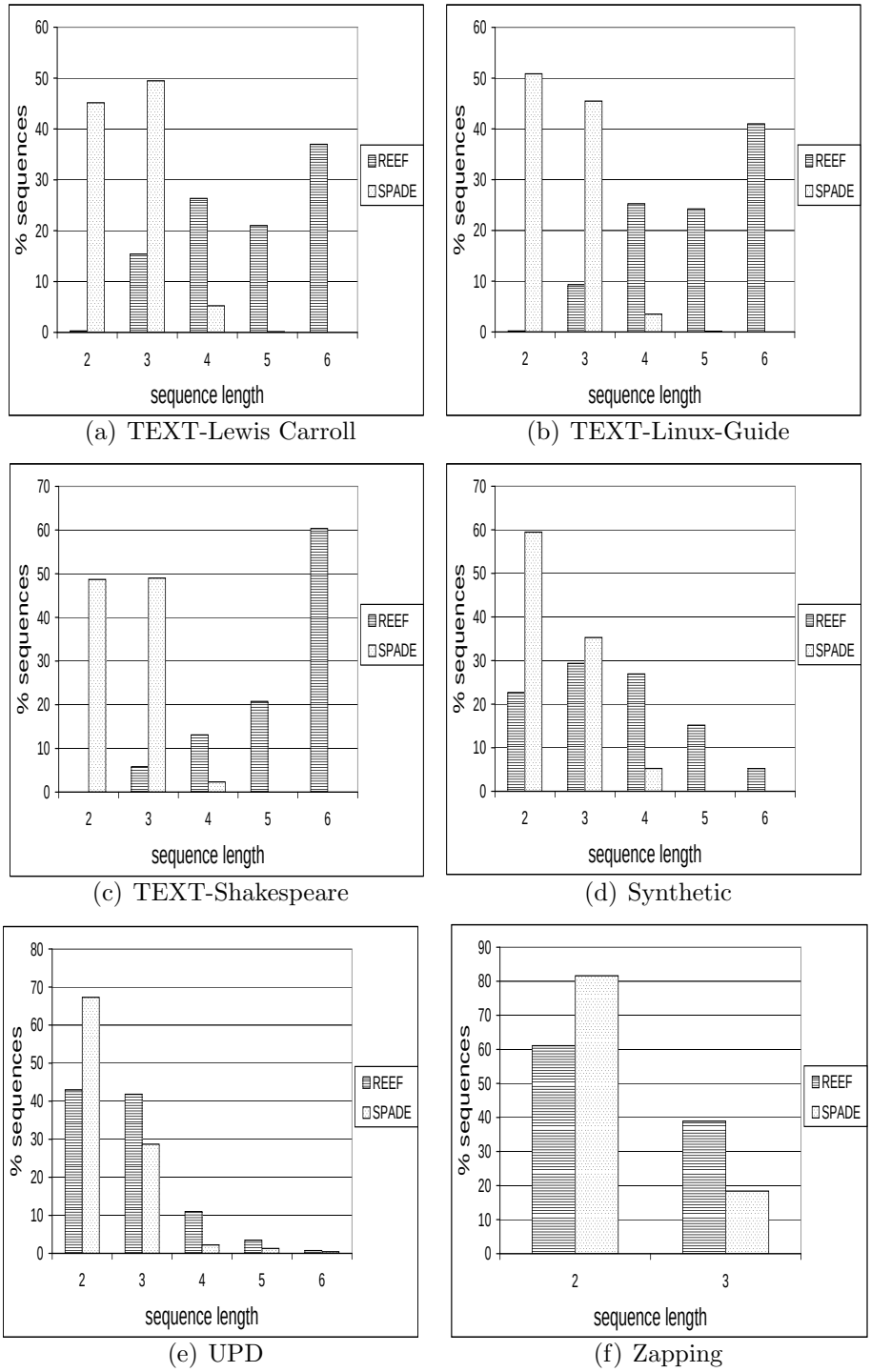
real words and illustrate yet again how the REEF output is superior.

For the Zapping and UPD data sets we do not know what the real underlying sequences are and what their distributions are. In the UPD set REEF again overcomes the short sequence bias and provides output sequences of all lengths in a more normal distribution than SPADE. This can be seen in in Fig. 4.9(e).

An interesting data set is the Zapping set. Although REEF allows for fair mining of all lengths the sequences found both with REEF and with SPADE are short, and there are no sequences with lengths higher than 3 as shown in in Fig. 4.9(f). This seems to imply that the frequent sequences in this set really are short. For this data set it would be more beneficial to use SPADE than REEF since there is not much quality to be gained from the slightly longer runtime with REEF. The Zapping set is different from all other three sets where the extra runtime is clearly worthwhile since the output sequences tend to be better representatives of the data set. Results on all four sets clearly show the tradeoff in the mining algorithms between time to sequence quality. Frequent sequence mining in support based algorithms such as SPADE generate short frequent sequences quickly. In contrast norm-frequent mining such as the one we presented in REEF takes slightly longer but generates better sequences as we show in 4.2.5, with a broader length distribution.

4.2.5 Mining Meaningful Sequences with REEF

The text domain was chosen in order to demonstrate the quality of the output sequences. We wanted a domain where the meaning of interesting sequences was clear. TEXT is obviously a good domain for this purpose since words are clearly more interesting than arbitrary sequences of letters. We hope to find more real words when mining text than nonsense words. Our evaluation is performed on three sets of text as described in Section 4.2.1. Results appear in Fig. 4.10. We compare results on *frequent* sequence mining using SPADE with *norm-frequent* sequence mining using



(a) TEXT-Lewis Carroll (b) TEXT-Linux-Guide

(c) TEXT-Shakespeare (d) Synthetic

(e) UPD (f) Zapping

Figure 4.9: Removal of length bias.

REEF. The x-axis shows different input sequence lengths (window sizes). For each input window size we calculated the percentage of real words that were found in the sequences that were mined. This is displayed on the y-axis.

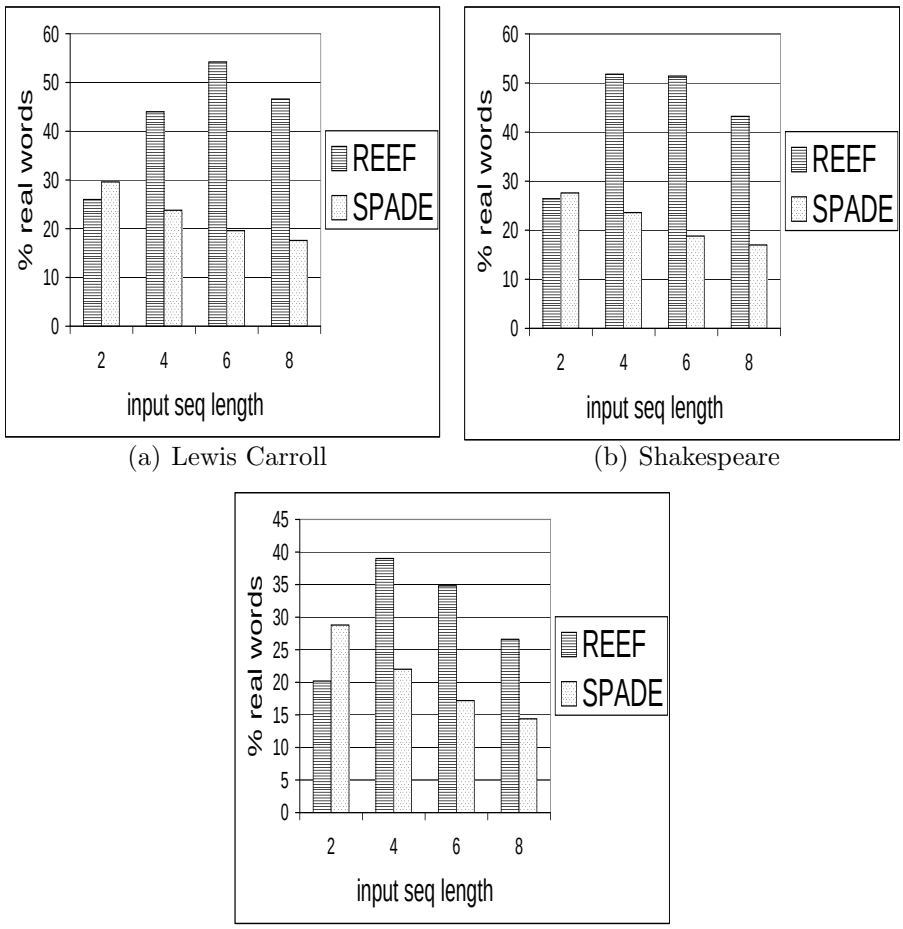
For all text sets REEF clearly outdoes spade by far. REEF manages to find substantially more words than SPADE for all input lengths. The short input-sequence sizes of 2 does not produce high percentages of real words for REEF or SPADE. Using longer input sequence lengths exhibits the strength of REEF in comparison to SPADE. For input lengths of 4,6 and 8 REEF manages to find a much higher percentage of words than SPADE. The average length of words in English is about 5 letters, thus the results from REEF are best for input sequences around this length. Clearly for text REEF performs much better mining than SPADE and the sequences mined are more meaningful.

Although the runtime for SPADE was shorter than for REEF the tradeoff between runtime and output quality is clearly demonstrated on the textual data. For many data sets, as for TEXT, it is worth spending more time to the more meaningful sequences in the mining process.

4.3 Discussion

We developed a new algorithm for frequent sequence mining named REEF that overcomes the short sequence bias present in many mining algorithms. We did this by defining *norm-frequency* and using it to replace support based frequency used in algorithms such as SPADE. In order to ensure scalability of REEF we introduced a bound used for pruning in the mining process. Making the runtime for REEF comparable to that of SPADE.

Sampling of the input data is performed for preprocessing in order to assist *norm-frequency* calculations. We addressed the issue of value distortion found in sampled



(a) Lewis Carroll (b) Shakespeare

(c) Linux Guide

Figure 4.10: Percentage of real words found among sequences.

values and provided a detailed method on how to fix it. Aside from the obvious contribution towards the mining in our work this method is beneficial for many algorithms that use sampling. Although the equations may vary for different data sets the method we presented can easily be applied to other problems where sampling is needed. In the future we plan to search for an analytic bound on the distortion and incorporate it into the distortion correction.

Our extensive experimental evaluation is performed on four different types of data sets. They are a mixture of synthetic and various real world data sets, thus providing a broad performance analysis of REEF. Results show without doubt that the bias is indeed eliminated. REEF succeeds in finding frequent sequences of various lengths and is not limited to short sequences. We demonstrated the scalability of REEF and addressed the tradeoff between runtime to quality of mined sequences.

We demonstrated that REEF produces a more variant distribution of output pattern lengths. We clearly showed how REEF mines more real words than SPADE. Therefor although REEF requires slightly longer runtime than SPADE the nature of the mined sequences makes this worthwhile. In the future we hope to improve the bound we use for mining and create an algorithm that is more efficient while still producing the high quality sequences we found in REEF.

Chapter 5

Multivariate Sequence Classification using Frequent Sequence Mining

It is important to classify multivariate temporal sequences. These are sequences of events, each of several attributes, with a temporal ordering. The task consists of finding sequences that can be used to classify different example classes. For instance in many domains it is important to observe behaviors of several people in a group, and then use these observations to determine online who is currently active. Examples include identifying the current user on a computer for security issues [69],[4] and identifying the current TV viewer for personalized TV services [18], [30].

Multivariate temporal sequence classification is a challenging task. There have been several attempts to address this problem, but none of the attempts present a full solution. One solution is to apply time-series classification to each attribute independently e.g. [34], [64] and [58]. This allows the use of one attribute at a time and uses the full time-series for this attribute. The main drawback of this solution is that there may be information in the combination of several attributes

that does not show up when they are separated. A second approach is to perform feature selection that looks at several attributes at single time points, thus losing the temporal property, as in [11]. Another option is to perform a preprocessing stage that incorporates several attributes on several time points as done in [28]. This is a good solution if one knows what the interesting events are, but for many cases, including the domains we investigated, this information is not available.

We present CUBS (**C**lassification **U**sing **B**ounded Z-Score with **S**ampling), an algorithm that finds patterns that span several attributes over several time points without any previous knowledge about the type of behaviors we are looking for. CUBS first uses the *norm-frequent* sequence mining algorithm REEF that was introduced in Chapter 4 to produce frequent subsequences. Next CUBS selects the statistically significant subsequences from among them as representatives to compose the model for classification.

CUBS provides a framework for classification of multivariate sequences for settings where we lack knowledge regarding the interesting features in the dataset, and have trouble reducing the problem for standard classifiers. CUBS achieves good results on several data sets. We perform evaluation on two real world data sets and one synthetic dataset. We demonstrate that using REEF as the mining component provides higher accuracy than using support based mining for classification of long sequences. We compare using CUBS to using an adaptation of decision trees for this classification and achieve good results in this task.

5.1 Problem Description

The notation we use was presented in Chapter 4 in Section 4.1.1. We now phrase the classification problem and the proposed approach.

Our input is a set of users $U = \{u\}$ where for each user we have a set of labeled

sequences $S_u = \{s_u\}$ where each s_u is a sequence $s_u = (e_1 \rightarrow e_2 \rightarrow ... \rightarrow e_q)$. We use these sequences to model the users. Given a new set of unlabeled sequences $S_{un} = \{s_{un}\}$ we want to determine which of the users $u \in U$ this set belongs to.

We select a set of subsequences $T_u = \{t_u | t_u \preceq s_u, s_u \in S_u\}$. The subsequences we place in T_u are frequent and statistically significant in relation to other users, and are used to model the user u. Our model M is represented by the subsequences selected for each user $M = \{T_u | u \in U\}$.

The classification is described as follows: Given a new unlabeled set of sequences S_{un} we use the model M to classify S_{un} by generating subsequences T_{un} from S_{un} and comparing them to T_u for each $u \in U$. We define a distance function in equation 5.1. The user u for whom $dist_{u,un}$ is smallest will be chosen as the classification for S_{un}. Intuitively we are comparing the sequence distributions in T_u and T_{un}.

5.2 CUBS Algorithms

We now describe the full CUBS classification Algorithm. Two main parts compose the algorithm. The first models the users, and the second classifies an unlabeled user. We describe building the model in Sec. 5.2.1. The model is built by integrating the REEF algorithm described in Section 4.1.5 with the significance component we will describe in 5.2.1. Section 5.2.2 presents the algorithm for classification.

5.2.1 Building the Model

The model we build uses the input sequences S_u. We choose representative subsequences T_u from among the input sequences S_u for each user u. There are two traits we look for in good candidate subsequences. One is that they are **frequent** - appear often enough to be useful, and the other is that they are **significant**- characteristic of only one user.

1: **for all** $u \in 1...n$ **do**
2: $Model_u \leftarrow 0$
3: **for all** $u \in 1...n$ **do**
4: $Data_u \leftarrow S_u$ broken into seq of *'size'*
5: $Freq_u \leftarrow \text{GetFrequent}(Data_u, best)$
6: **for all** $u \in 1...n$ **do**
7: $Sig_u \leftarrow \text{GetSig}(Freq_u, Freq_v, sig) \forall v \neq u$
8: $Model = Model \cup Sig_u$
9: **return** $Model$

Figure 5.1: BuildModel($\{S_1, , S_n\}, best, sig, size$). Where $\{S_1, , S_n\}$ are the input data sequences, *best* is the number of frequent subsequences to mine, *sig* is the number of significant subsequences to keep and *size* is the input window size. The function returns a Model.

The Algorithm *BuildModel*() in Fig. 5.1 describes the procedure for building the model. Where n is the number of users, and S_u the input for each user. The $best, sig, size$ parameters are defined shortly. The input to our algorithm for each user is in the form of several temporal sequences. Each temporal sequence s_u represents one session of activity and there are several sessions for each user $S_u = \{s_u\}$. For example assume we have two users u and v with two sessions each:

$$\begin{aligned} s_{u_1}: &\quad A, B \rightarrow C, D \rightarrow A, D \rightarrow A, B, C \\ s_{u_2}: &\quad A, B, C \rightarrow C \rightarrow A, D \rightarrow B, C \\ S_u: &\quad \{s_{u_1}, s_{u_1}\} \end{aligned}$$

$$\begin{aligned} s_{v_1}: &\quad B, C \rightarrow A, C \rightarrow B \rightarrow A, B, C \\ s_{v_2}: &\quad B, C \rightarrow D \rightarrow A, B, C \\ S_v: &\quad \{s_{v_1}, s_{v_1}\} \end{aligned}$$

The full sessions are too long to be processed, so we break these sessions into smaller sequences. We assume that in the full session there are many short events that reoccur and represent the user. We would like to break each session into shorter sequences that correlate to these events. However we do not know where one event ends and where another begins. We do not even know what these events are. Therefor

we use a sliding window of size *'size'* and create sequences from the input session covering all possible events. Using a window of *'size'* = 2 on our input S_u results in $Data_u$.

$$\begin{array}{lllll} Data_u: & A,B \rightarrow C,D & C,D \rightarrow A,D & A,D \rightarrow A,B,C \\ & A,B,C \rightarrow C & C \rightarrow A,D & A,D \rightarrow B,C \end{array}$$

Frequent Sequences

BuildModel next finds frequent subsequences $Freq_u \preceq Data_u$ in the input sequences for each user. The number of frequent subsequences found is determined by the *'best'* parameter. The algorithm used for finding the frequent sequences is the REEF algorithm from Section 4.1.5. The frequent sequences for our example are:

$$\begin{array}{llllll} Freq_u: & A,C & A,B,C & B,C & \\ & A,B & B \rightarrow A & B \rightarrow C & \\ Freq_v: & A,B,C & A,B \rightarrow C & A \rightarrow C & A,D \rightarrow B,C \\ & A,D & A,D \rightarrow B & A,D \rightarrow C & A \rightarrow B,C \end{array}$$

Notice that the subsequence A,B,C in our example appears in both sets of frequent subsequences, therefore it may not be a good separator, unless it appears very often in one group and rarely in the other. This leads us to selecting significant sequences from among the frequent sequences.

Significant Sequences

The next part of the CUBS classification algorithm selects statistically significant subsequences from the list of frequent sequences. This is performed using a χ^2 test. Calculating the χ^2 test on the subsequences in order to find the significant subsequences simply grades all subsequences. It is then necessary to decide how significant we want the subsequences to be, or in other words how many to use. One option is to use a threshold and select all subsequences with a χ^2 value above the threshold. The

problem with this technique is that different data sets have different χ^2 values and it is unclear how to set this threshold. The value is depends on the size of the data and each user has different number of representing sequences. By selecting a fixed number of subsequences to use for each user we promise a fixed size for the model. We simply select *sig* top candidates for each user.

Model

This frequent and significant sequence mining is performed for each user. The model consists of the union of the significant subsequences for all users $Model = \bigcup Sig_u$.

$$
\begin{array}{lllll}
Model_{u,v}: & A, D \rightarrow B, C & A, D & A \rightarrow C \\
 & A, B \rightarrow C & A, D \rightarrow B & A, D \rightarrow C \\
 & A \rightarrow B, C & B \rightarrow A, C & A, C \\
 & B, C & A, B & B \rightarrow A \\
 & B \rightarrow C
\end{array}
$$

The structure of the model is in the form of a list of subsequences, for each subsequence t we specify what the $\chi^2(t)$ grade was, and for each user u the probability that this subsequence appears $p(t|u)$ or does not appear $p(\neg t|u)$ in this user.

5.2.2 Classification

The classification unit of CUBS is described by the Classify() function in Fig. 5.2. At this stage we have a $Model$ and a new input data set S_{un}.

We want to classify S_{un} as belonging to one of the users in the $Model$. We break S_{un} into sequences to create the dataset $Data$, and find all possible subsequences $Data' \preceq Data$.

$$
\begin{array}{lllll}
S_{un}: & A, D \rightarrow A \rightarrow C \\
Data: & A, D \rightarrow A & A \rightarrow C \\
Data': & A, D & A \rightarrow A & D \rightarrow A & A \rightarrow C
\end{array}
$$

1: $Data \leftarrow x$ broken into sequences of *'size'*
2: $Data' \leftarrow \text{GetSubSeq}(Data)$
3: $f = \text{Clean}(Data', M)$
4: **for all** $t \in M$ **do**
5: **for all** $u \in 1...n$ **do**
6: $dist_{u,un} \leftarrow dist_{u,un} + dist_{u,un}(t)$
7: **return** u with $\text{minimum}(dist_{u,un})$

Figure 5.2: Classify$(x, Model, size)$. Where x is the data to classify, $Model$ is the Model to use for classification and $size$ is the window size to use on the data. The function returns a classification.

Next we clean the list $Data'$ using the $Model$. This stage discards all subsequences that do not appear in the $Model$. The reason for this is that we cannot say anything about subsequences that do not appear in the $Model$ so they are of no use to us. In our example we discard $A \rightarrow A$ and $D \rightarrow A$. Every subsequence $t \in Model$ that appears in the model is given a grade determined by the distance between the subsequence from the input $t \in Data'$ to the same subsequence $t \in Model$ in the model. The function that calculates the distance between two users, a user u from the model, and an unknown user un, for a subsequence t is described in Eq. 5.1.

$$dist_{u,un}(t) = \begin{cases} \frac{(p(t|u)-p(t|un))^2}{p(t|u)} & t \in un \\ \\ \frac{(p(\neg t|u)-p(\neg t|un))^2}{p(\neg t|u)} & t \notin un \end{cases} \tag{5.1}$$

We calculate the 'distance' between user u to user un by summing distances $dist_{u,un}(t)$ for all subsequences $t \in u$. The user u for which $dist_{u,un}$ is minimal is the classification. It should be noted that in our distance calculation we may have to divide by 0 when a sequence appears in the model but not in a specific user. In this case we return a large constant. In the example we are using this would result in our test being classified as u.

5.3 Evaluation

In this section we present the experiments used to evaluate CUBS. We use both synthetic data and two real data sets. We demonstrate the strength of CUBS relative to support based methods. We show how we compare to decision tress with a special version of CUBS.

5.3.1 REEF component

We begin by comparing the performance of the CUBS algorithm with two variations. CUBS-REEF that uses the REEF component for mining, and CUBS-SPADE that is the same as CUBS but replaces the REEF component with the SPADE algorithm. This compares the accuracy rates using *norm-frequent* sequence mining with the z-score normalization (CUBS-REEF) to *frequent* sequence mining using support based mining(CUBS-SPADE). We perform this comparison on the synthetic data set we used the IBM Quest Synthetic Data Generator [2], details on the data set appear in Section 4.2.1. With QUEST we generated one base class from which we derived 5 sets of classes. The base class is composed of items $\{1, 2, 3, 4, 5, 6, 7, 9\}$ without $\{8\}$. In the first set we added 8 to all sequences in one class, and nothing to the other class. This is trivial to separate and is used as a base for evaluation. In the second set we add 8 to 90% of the sequences in the base set for one class, and 8 to 10% of the sequences in the second class. The third set has 8 added to 80% and 20%, the fourth has 8 added to 70% and 30%, and the fifth set has has 8 added to 60% and 40% of the sequences.

The structure of the data for each set is as follows: for each class we have one training set, and five test sets. The classification of a class is given by the average accuracy over these five sets.

Fig. 5.3 shows how accuracy improves when using the the REEF component

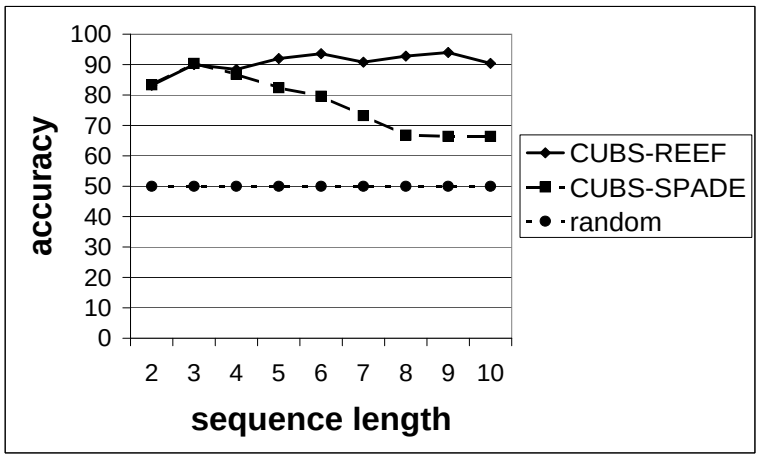


Figure 5.3: Accuracy using REEF component (Norm-Frequent) vs. SPADE component (Frequent)

relative to use of the SPADE component. Each point is an average of 5 runs using various settings for *'best'* and *'sig'* parameters described in Sec. 5.2.1, and over all five sets.

The results in Fig. 5.3 demonstrate that as sequence length grows the z-score version performs at a higher accuracy because it can handle long sequences, whereas the minimum support version is biased towards short sequences and performance drops as sequence length grows. This graph shows that the bias towards short sequences is removed by the z-score normalization, thus improving classification accuracy for long sequences.

Fig. 5.4 shows how the improved frequent mining algorithm incorporated in the CUBS classification performs on real data. UPD (User Pattern Detection) is another dataset we used for evaluation and is based on real world data. Details on the data set appear in Section 4.2.1, and in Appendix A. Our experiments are run on 15 pairs of users. The length of sessions varies between users. For each user we use 9 sessions for training and 1 for test. The *'best'* parameter is set to 50, and *'sig'* is set to 10. Fig. 5.4 shows that the best accuracy of 83% is achieved using a window *'size'* of 1,

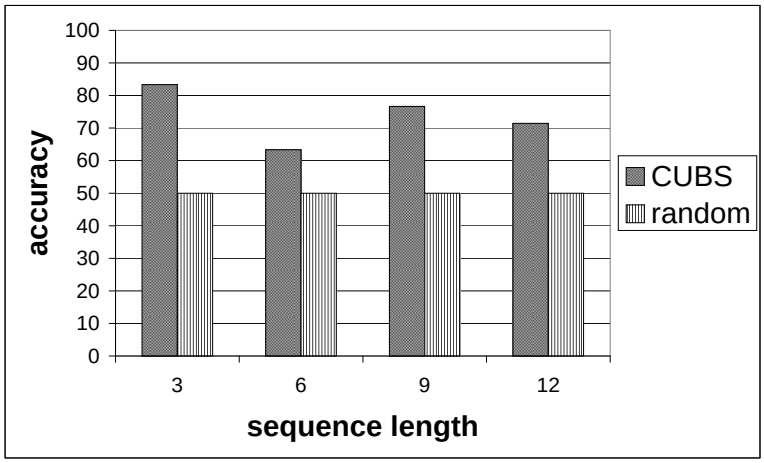


Figure 5.4: UPD: Accuracy vs. Size

correlating to a length of 3, meaning one itemset with 3 items.

5.3.2 Sampling Component

This section evaluates how the use of the sampling component affects the accuracy results of the classification. We ran these experiments on the data collected on the remote control Zapping data set. Details on the data set appear in Section 4.2.1, and in Appendix A. The amount of data for each family is limited, therefore instead of setting some data aside for testing we perform 5 fold cross validation. We used half the data for learning the sampling distortion correction and use the other half for results presented here.

Fig. 5.5 shows results of the classification when using statistics from full DB and statistics from a sampled DB with distortion correction. We sampled 10% of the database for the sampled set. The results obtained with the sampled data after performing distortion correction are very close to results obtained using the full dataset. Therefor we have shown that the distortion correction can be successfully used.

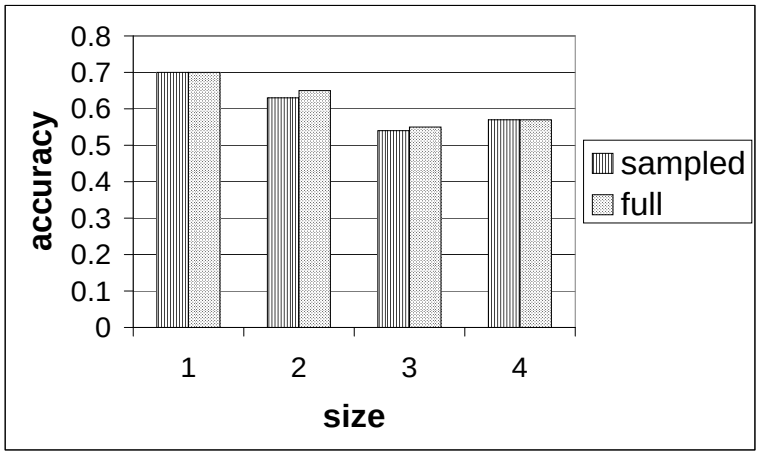


Figure 5.5: Zapping: Effect of Using Sampled Statistics

5.3.3 Parameter Setting

In the CUBS algorithm there are two parameters that need to be set. One is the *'best'* parameter that describes how many frequent sequences to mine. The other is the *'sig'* parameter that describes how many significant sequences to select from among the frequent sequences. We tested many combinations and present some of them in the following graphs.

The results for the synthetic data are presented in Fig. 5.6 and Fig. 5.7. Fig. 5.6 displays accuracy rates for various settings of *'best'* and *'sig'* over the different synthetic sets as described in 5.3.1. Each point is the result of several runs with different input lengths. Fig. 5.7 has the accuracy for various *'best'* and *'sig'* values as function of input sequence length. Each point is averaged over all 5 synthetic data sets.

Both figures show that it is of little importance which parameters are selected for this data set. In fact for Fig. 5.6 the different parameters results in the same accuracy rates and the lines in the graph are not differentiable. For the accuracy rates in Fig. 5.7 there are some differences for small input sequence lengths, but as input lengths grows, the various parameters provide similar accuracy rates.

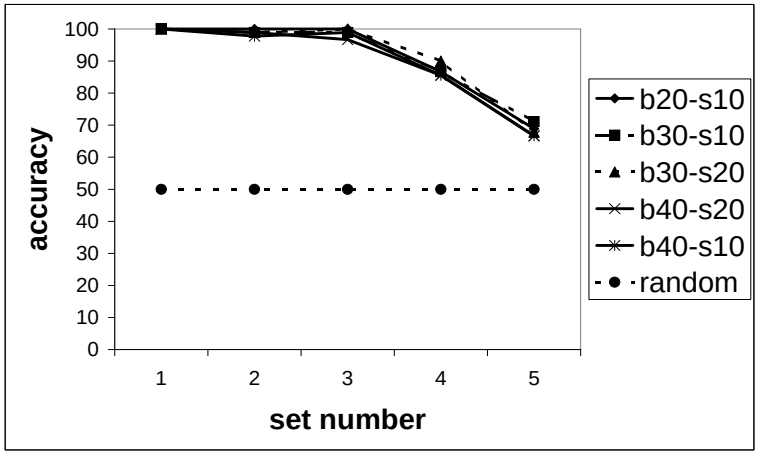


Figure 5.6: Accuracy for Synthetic data - various sets

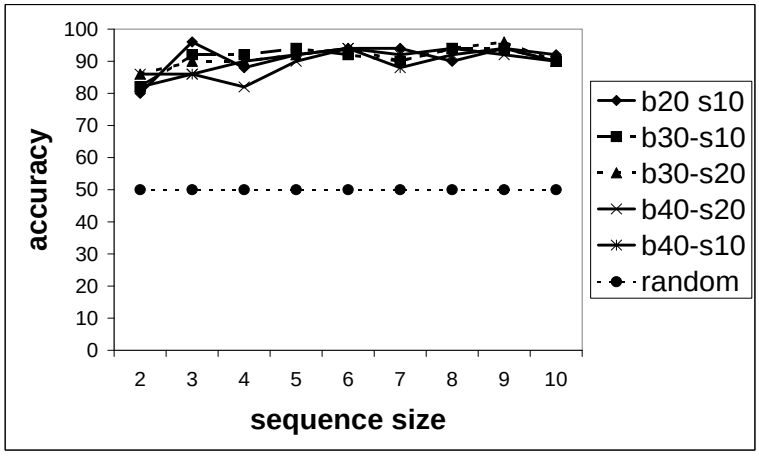


Figure 5.7: Accuracy for Synthetic data - various input sequence lengths

We explored the way various parameter settings affect the accuracy rates of CUBS for the Zapping data as well. In Fig. 5.8 we set *'sig'* = 20 and show results for various *'best'* values. Using higher *'best'* values, or in other words more frequent subsequence candidates generates better results. Fig. 5.9 complements this by setting *'best'* = 100 and varying *'sig'*. For most cases setting *'sig'* high produces the best results.

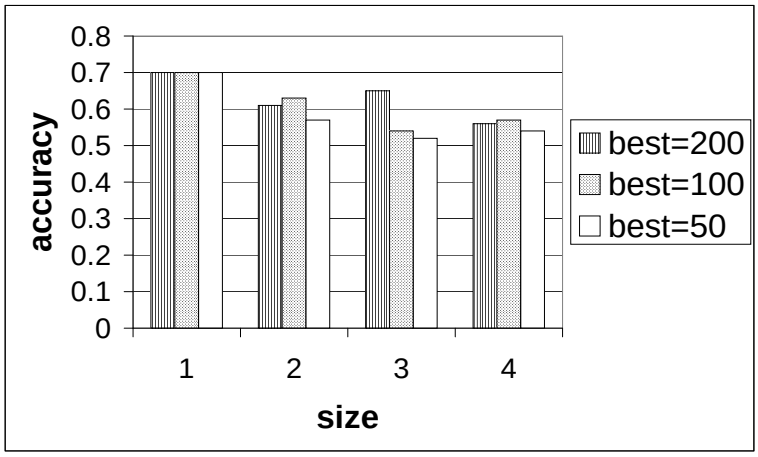


Figure 5.8: Zapping: 'sig'=20, various 'best'

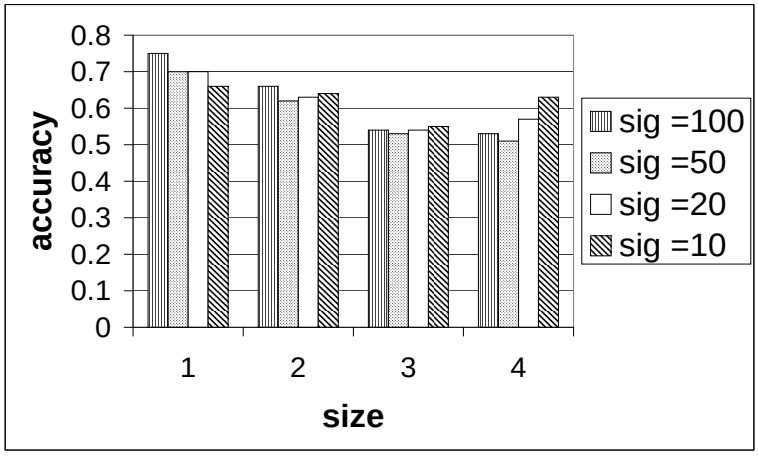


Figure 5.9: Zapping: 'best'=100, various 'sig'

5.3.4 Differentiability of Data

For the synthetic data set we checked how different levels of differentiability affect the accuracy rate. Fig. 5.10 demonstrates this effect. One can see that as the classes become harder to differentiate because they are more similar to each other, the accuracy drops as we expect. Set 1 is trivial to differentiate as one class has 8's in it and the other doesn't, in this case we achieve a classification rate of 100% as expected. At the other end of the spectrum in set 5 the classes are very similar and

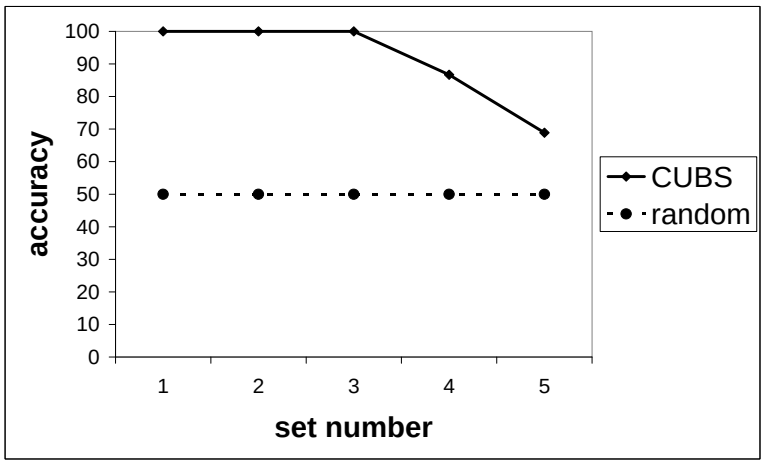


Figure 5.10: Accuracy vs. Set

the only difference is the frequency that the 8's appear that are 60% of the sequences in one class and 40% in the other, in this case we achieve an accuracy of around 70% still much higher than the random classification which is 50% for these two classes.

5.3.5 Comparison Decision Trees

In order to compare CUBS to state of the art classification algorithms we chose Decision Trees (DT). Fig. 5.11 displays the accuracy for CUBS relative to DT and random classification. In order to use DT for our multivariate temporal data, it must be preprocessed, since DT cannot handle the temporal sequential nature of the data. We adapted the data so that the time dimension is removed and each vector is composed of one item set.

CUBS achieves a 75% accuracy rate and performs slightly better than DTs with 71% accuracy on our data set. This is a small set and we are not confident that we can always outperform DTs. However CUBS is capable of handling data that DTs cannot handle. In order to use DTs for multivariate data one of the dimensions must be reduced and information is lost, whereas CUBS handles the multivariate input as is. In this sense CUBS is a stronger algorithm than DT.

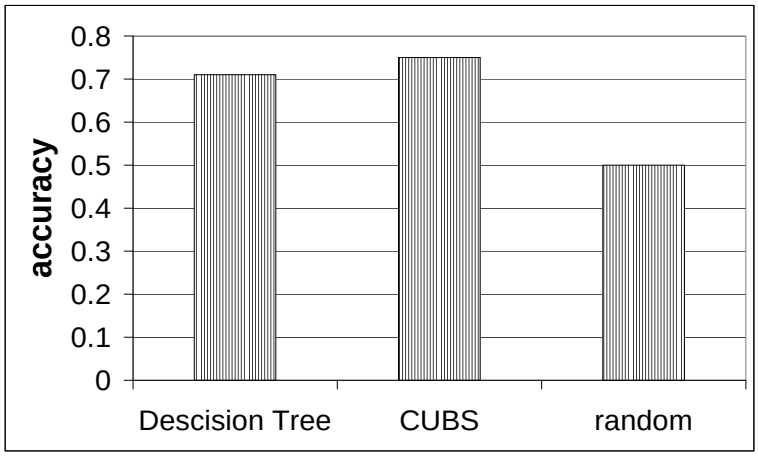


Figure 5.11: Zapping Classification

5.4 Discussion

This Chapter presented the innovative CUBS algorithm. CUBS provides a much needed framework for classification of multivariate temporal sequences. In CUBS we mine the sequential multivariate data for frequent subsequences and then select the statistically significant candidates and use them for classification. We have demonstrated the successful classification ability on both synthetic and real data. CUBS uses the novel REEF algorithm for frequent sequence mining that introduces *norm-frequent* mining by using a z-score normalization on support. Replacing the commonly used support measure with this z-score normalization shows evidence of fixing the bias towards short sequences in the mining. Fixing this bias significantly improves accuracy of classification.

Combining the z-score normalization with the pruning in REEF, based on the z-score bound, along with our novel sampling unit provides an improved, unbias and scalable algorithm for mining frequent multivariate subsequences. Using this algorithm as part of the CUBS classification algorithm results in a multivariate sequential classification algorithm. We used real world data that was collected in a totally free uncontrolled environment, and is therefore very noisy. Despite this CUBS achieves

high accuracy rates of 75%–83% and performs slightly better than Decision Trees. The beauty of CUBS is no doubt its ability to successfully classify multivariate and temporal sequences while exploring both the attribute dimension and the temporal dimension simultaneously.

Chapter 6

Conclusion and Future Work

In this thesis we researched classification and frequent sequence mining algorithms for multiple attribute sequences. The first chapter was motivated by the well known technique of reducing the multivariate problem into several single attribute problems and performing classification for each attribute separately. Since having several classifications for a single data set is of little use we introduced heuristic method for integrating the single attribute classifications into a single classification for the multivariate input. The algorithm, named COACH, was applied to handwriting deficiency classification. COACH is an algorithm that has been described by domain experts as successful in achieving high classification accuracy rates in comparison to what is acceptable in this domain. However the methods for creating the heuristics used for integration are extremely specific and dependent on input from domain specialists.

Unfortunately domain expert information is not always available. In many domains no expert information is available and classification must be performed without any understanding regarding the data. This motivates the work we performed in the second and third parts of this thesis. The second part of the thesis introduces an innovative algorithm for performing frequent sequence mining of multivariate data. Frequent sequence mining is an interesting problem in its own right. Our interest in

the frequent sequence mining algorithm was also for use as part of our classification algorithm described in the third part of this dissertation.

The frequent sequence mining algorithm REEF that we introduced in the second part of the dissertation introduced a new definition of frequency. The regular definition of frequency is based on support of sequences. We introduced the new *norm-frequency* that is based on the normalized value of support. The *norm-frequency* definition is used to address the short sequence bias that is present in support based frequent sequence mining. Aside from solving the short sequence bias we addressed two scalability issues that arise from the introduction of the *norm-frequent* measure.

The first scalability problem we solved was the use of a bound in the enumeration process of candidates for frequent sequences. The use of the bound enables pruning in the enumeration process. Without pruning the algorithm although correct is unscalable. The bound and the pruning are what make the algorithm scalable and relevant to real world data sets. Calculating the *norm-frequency* and the bound used for pruning both require a preprocessing stage that seems to require a full pass over the database. We addressed this second scalability issue by using a sample of the database to acquire these values. As values acquired from the sample suffered from a distortion, we analyzed this distortion and corrected it.

Our experimental results on the frequent sequence mining show that we have succeeded in solving the bias towards short sequences. We demonstrated on several data sets both with real world data and synthetic data how the algorithm is scalable and can be used for real data. Although the runtime of our algorithm is not as short as support based methods the richness of the mined sequences makes that the extra runtime worthwhile. We used textual data to show how REEF mined many more real words than support based methods implying that REEF mines more meaningful sequences.

In the last section we implemented CUBS, a classification algorithm that uses

REEF as a frequent sequence mining component. We evaluated CUBS on several data sets. On the synthetic data we established that using CUBS with the REEF component provided high success rates. As the input sequence length increased accuracy improved. For the synthetic data we know that the "real" sequences hidden in the data are long sequences, and as we expected using long sequences improves classification accuracy. For our real data we did not find the same patterns, increasing input sequence length did not always increase accuracy rates.

In the future we would like to investigate why accuracy does not always improve with input sequence length. We experimented with numerous values for CUBS parameters settings of *'best'* and *'sig'* and found the accuracy rates to be rather indifferent to these values. We conclude that it is not the parameter values that affect the accuracy rate. The assumption we made is that in the real data sets we used, longer sequences do not hold more information than shorter sequences. A possible direction for future work is to confirm that this is indeed the case.

Continuous data values in the input sequences were segmented into discrete values for both REEF and CUBS. We would like to provide an algorithm that performs the same type of mining and classification as REEF and CUBS but can handle continuous input directly.

This thesis provides a framework for mining and classifying multivariate sequential data. Although found to be scalable and solve the short sequence bias there is still room for improvement in the runtime of the mining process. Our mining algorithm does well with multiple attributes for data where there is a small number of values that can be assigned to the attributes. For continuous values or data with a large number of possible attribute values we still have trouble with scalability. We believe this is an interesting and beneficial area to approach the future.

Appendix A

Data Collection

The research carried out in this chapter uses two real world data sets. These data sets were very important to the evaluation process. Results on synthetic are always beneficial for understanding how an algorithm works. However the use of real world data is necessary in order to provide a complete evaluation. Real world data often causes the algorithm to react in unexpected ways and therefore is crucial to complete the evaluation process. As part of this work we have created two such real world data sets that are composed of multivariate sequential data. These data sets are a contribution to the field of multivariate sequence mining and classification since although many systems create this type of data there are few publicly available sets of this kind. We describe the collection manner and the properties of these data sets in the following sections.

A.1 ComMonitor-Simulation for Personal TV

In order to gather data from remote control usage we developed our ComMonitor device. In the real world data from remote control usage can be easily collected by TV providers. However since this data is highly personal there are many restrictions

on the distribution of this type of information. Therefor finding data sets of this type of data is impossible. This raised the need to create our own data set. In order to fulfill this task we built a device that logs the actions performed on the remote control by capturing the infrared signal from the remote control. This data is saved on a laptop and used to build our data set. Each user identifies himself when starting to use the remote control thus providing a labeled data set of remote control usage. The experiment was run in over 30 households.

When a person starts to use the remote control he presses a predefined button on the remote. Each member of the household has such a button. This is defined in an ini file as shown in Fig A.1, where Joan and Robert and Ariella are the household member names, "00 00 ff", "00 ff 00", and "ff 00 ff" the colors displayed on the interface, and "0C0C" the code for the remote control and 18, 68, e8 the designated buttons on the remote control for each user. The member name remains constant until another user is selected. This member is logged along with each action he performs. The setup of the system is described in Fig. A.2, a laptop is connected to the ComMonitor box. The ComMonitor box records remote control infrared signal, and passes information to the laptop where the information is logged. The ComMonitor and laptop sit next to the set-top box as to record the same signal recorded by the set-top box. There is no physical connection between them. The ComMonitor is accompanied by an interface that displays the current viewer, the active remote control and the last button selected as shown in Fig. A.3. The display includes both the members' name and a colored rectangle this enables small children to identify themselves as the active viewer, allowing them to contribute to the experiment.

The ComMonitor device collects and logs all actions performed on the remote control for use in the identifying process. The ComMonitor detects the infra red signal from the remote control and logs data concerning the signal. For each signal the data saved is:

```
COM8
Robert: 00 00 ff 0C0C 18
Joan: 00 ff 00 0C0C 68
Ariella: ff 00 ff 0C0C e8
```

Figure A.1: ComMonitor *.ini file

- Time of the signal.
- Code of the remote control.
- The code for the button pressed.
- The name of the current member using the remote.

An example of a part of a typical log file is shown in Fig. A.4.

The data collected with the ComMonitor device is very basic. We also developed an application that uses the data provided by the ComMonitor logs and converts it into events. Our software reads these logs and builds the sequences of events used for classification. The Events defined and collected on the zapping patterns include:

- Button pressed (number, help, data, etc...)
- Time passed since last activity (20 intervals describing different time lengths)
- Time of day (Morning, afternoon, evening, night)

Collecting data is an ongoing process. We have up to date collected data from over 30 families, and plan to continue as this data set can be used for further research. We made an effort to use families with adults and children in order to model different ages. We modeled children aged 4-13, with 3-6 members per household for a period of 2-3 weeks.

The process is described by most families as easy. They liked the fact that there was no direct intervention between the TV or the set top box and our equipment (so

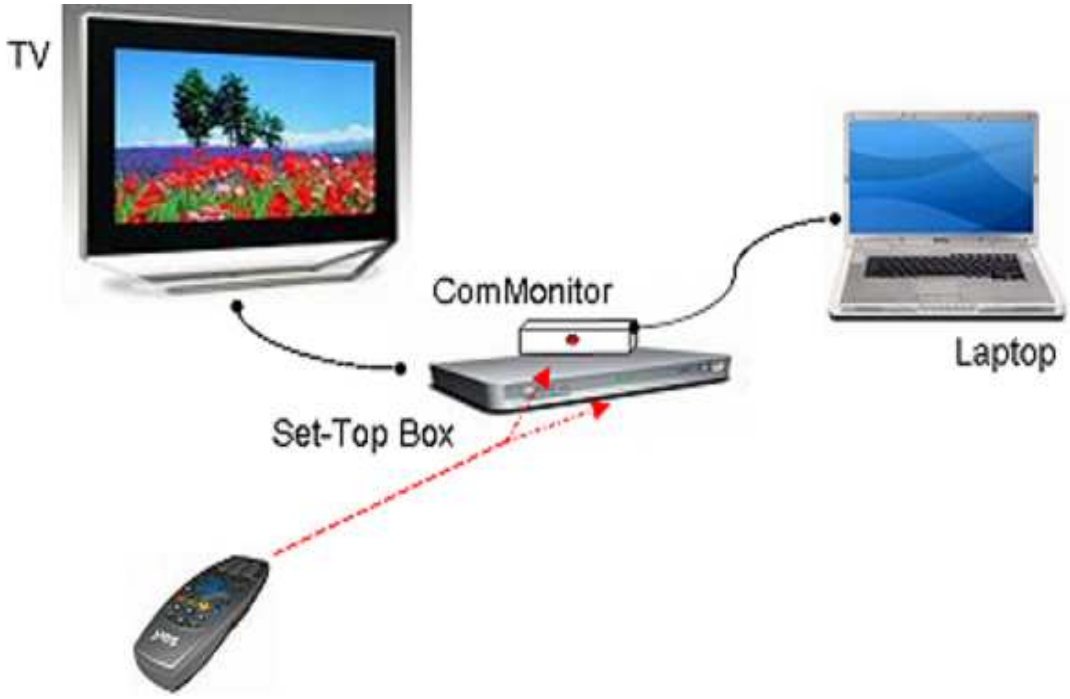


Figure A.2: ComMonitor Collection Setup

nothing could go wrong with the TV...). School aged children seemed to actually enjoy the involvement very much. We encountered some problems with data collection in a family where the knowledge of handling electronic equipment was very low. Our main problem was getting people to participate before they understood what was involved. People feared we would invade their privacy and track what they were watching etc. and many people felt uncomfortable with that even thought we explained that we had no interest in tracking what they watched, but were interested in what buttons they pressed.

Figure A.3: ComMonitor Interface

```
SysTime:22-11-2008 21:56:58.382 0C0C 44 Joan
SysTime:22-11-2008 21:56:59.494 0C0C 78 Joan
SysTime:22-11-2008 21:57:05.983 817E BC Joan
SysTime:22-11-2008 21:57:06.804 02FD 78 Joan
SysTime:22-11-2008 22:31:50.981 0C0C 18 Robert
SysTime:22-11-2008 22:31:55.818 0C0C 58 Robert
SysTime:22-11-2008 22:31:56.259 0C0C 58 Robert
SysTime:22-11-2008 22:31:58.122 0C0C e8 Ariella
SysTime:22-11-2008 22:31:58.552 0C0C 58 Ariella
SysTime:22-11-2008 22:31:59.464 2F0C 58 Ariella
SysTime:22-11-2008 22:32:00.265 0C0C 58 Robert
SysTime:22-11-2008 22:32:08.176 0C0C 98 Ariella
SysTime:23-11-2008 21:20:59.956 02FD 78 Joan
SysTime:23-11-2008 21:21:00.416 02FD 78 Joan
SysTime:23-11-2008 21:39:25.706 0C0C 18 Robert
SysTime:23-11-2008 21:39:59.034 0C0C 90 Robert
SysTime:23-11-2008 21:39:59.144 0C0C 90 Robert
```

Figure A.4: ComMonitor log file

A.2 UPD- User Pattern Data Gathering Tool

In order to gather data from computer usage we wrote our UPD application. UPD collects data on mouse activity and keyboard actions. We also collect data on window activity. The UPD runs on windows in the background gathering data while a user works normally. We felt the need to build this application on our own for two main reasons: The first was that we wanted an application that monitors all three kinds of activity simultaneously. The second was the fear of trusting this type of application from an outside source. After initial development of the first version of the UPD we began deployment. We found many problems with the original structure, and made various changes based on our experience. We describe the current deployed version here.

The UPD gathers information about user activity by hooking the windows system. All data retrieved is logged and saved in files. The UPD system is composed of the following units:

- Hooking entity
- Buffer
- Encryption
- Log file
- Decryption entity

The Hooking entity gathers information about all keyboard, mouse and window activity. This data is saved in a buffer on the local computer. When enough data is collected it is encrypted and then passed to log files. The log files are saved on a central computer in the encrypted form. The encrypted files are downloaded at relatively frequent intervals to a safe location where they are decrypted and used.

Much effort was put into reducing the amount of overhead to the system. The Idea was to minimize the amount of intervention felt by the user in order to achieve natural usage patterns and as much cooperation from the users as possible. An example to this is buffering of the data. Rather than logging each action as it occurs, actions are saved in a buffer and logged in chunks. This reduces the number of times we perform writing to a file thus minimizing the load on the system. Another point taken into consideration was that we designed UPD in such a way that if for some reason it faults UPD will not cause other applications to get stuck.

The UPD generates various types of data. Some of this data may be sensitive. By sensitive we mean that a user might not want the data to be exposed. This obviously includes data such as passwords but also other types of data. We found that when collecting data for research many people felt uncomfortable exposing for the names of the windows that are opened (this shows for example the address of an internet site being viewed), or text being typed in an email. We feel that this issue is to be addressed in the context of where the final application is installed. It may be fitting to collect all data in places where high security is needed, and considered privacy invasion in others. We provide a variety of privacy level options. One option is to collect the data 'as is' saving all letters and window names. The second option is not saving names of letter keys, but only function keys (shift, ctrl etc.) or saving the letter keys as codes, so that the content is not clear to the naked eye just by looking at the log files, but patterns in the text can be derived. These options are easily controlled using a configuration file and can be adjusted depending on where UPD is deployed.

The precautions used with the configuration file are sufficient for data that is held in a secure environment and known not to be exploited. However while the data sits on a common server there is need to take stronger precautions. For This we developed an Encryption unit. The encryption unit Encrypts the data logged in

the buffer before it is printed to the log file. All data sitting on the main server is encrypted and therefore unreadable to someone trying to search the files for secure information (such as user passwords). The decryption is performed in a secure setting.

Each activity is logged along with the time and date it occurs. We chose to report the following activities to a log file:

- Keyboard activity
  - key-down, key-up
  - name of key - depending on security level.
    - actual key name
    - ascii value
    - only special keys such as 'return' and 'shift' are logged (no text)
- Mouse
  - mouse move
  - mouse click:
    - right/left button down
    - right/left button up
    - right/left double click
- Window activity (accompanied by text from window title)
  - window created
  - window closed
  - window minimized/maximized

```
Tue Jan 18 2011 11:01:35.921 WH_CBT HCBT_KEYSKIPPED 249 1966081
Tue Jan 18 2011 11:01:36.078 WH_CBT HCBT_KEYSKIPPED 249 -1071775743
Tue Jan 18 2011 11:01:38.828 WH_CBT HCBT_CLICKSKIPPED
WM_LBUTTONDOWN lParam is a pointer to MOUSEHOOKSTRUCT con-
taning pt.x, pt.y, hwnd, wHitTestCode, dwExtranInfo. lParam = 22084988
Tue Jan 18 2011 11:01:44.578 WH_CBT HCBT_KEYSKIPPED 18 540540929
Tue Jan 18 2011 11:01:44.671 WH_CBT HCBT_KEYSKIPPED 16 539623425
Tue Jan 18 2011 11:01:44.750 WH_CBT HCBT_KEYSKIPPED 18 -1070071807
Tue Jan 18 2011 11:01:45.218 WH_CBT HCBT_KEYSKIPPED 17 -2145583103
Tue Jan 18 2011 11:01:45.234 WH_CBT HCBT_KEYSKIPPED 16 -1070989311
Tue Jan 18 2011 11:01:46.671 WH_CBT HCBT_KEYSKIPPED 17 1900545
Tue Jan 18 2011 11:01:46.765 WH_CBT HCBT_KEYSKIPPED 16 2752513
Tue Jan 18 2011 11:01:46.906 WH_CBT HCBT_KEYSKIPPED 16 -1070989311
Tue Jan 18 2011 11:01:54.203 WH_CBT HCBT_KEYSKIPPED 115 2031617
Tue Jan 18 2011 11:01:54.265 WH_CBT HCBT_KEYSKIPPED 115 -1071710207
Tue Jan 18 2011 11:01:54.406 WH_CBT HCBT_KEYSKIPPED 32 3735553
Tue Jan 18 2011 11:01:54.484 WH_CBT HCBT_KEYSKIPPED 32 -1070006271
```

Figure A.5: UPD log file

An example of part of an output file appears in Fig. A.5.

The data collected by UPD must be converted into events. It can not be used in the raw form since it consists of primitive actions performed on the computer (key pressed down, key released, mouse is moved, mouse clicked etc). Therefor we developed an application that uses the data provided in the UPD and converts it into events. More details follow in the section describing the Events.

UPD was deployed in one of our labs where we collect information from students using the computers in the lab. Data was collected and saved in the encrypted form on a central computer. This data was downloaded at frequent intervals in order to avoid loss or contamination. The downloaded data was saved in a secure place and decrypted for algorithm testing. At login the user was asked whether he agrees to participate in our experimental data collection, and if consent is provided the logging of activity automatically begins. The logging continues until a certain amount of data is collected (defined in configuration file)

We have collected data from over 150 subjects. The amount and quality of the data vary. For approximately a third of these subjects we have repeated sets of data. For the others single sets. The amount of data collected varies from very short sessions to 1-2 hour long sessions. The amount of data collected is determined by volume rather than time. We have 54 subjects with at least 2 sets of data where there are over 10 file in each set. (30 - two sets, 17 - three sets, 6- four sets, 1 five). These subjects are good candidates for simulating legal users. One the other hand there are 90 subjects with one set of data, these subjects can be used to simulate the invaders.

As shown in the previous section the actions logged by UPD are primitive actions. In order to increase our ability to learn significant patterns we convert these actions into **events**. An event is a meaningful task derived from a set of actions. For example instead of using key-down and key-up as input to the classification a series of these two actions is translated into a key-press event.

We have defined several events so far and propose to define more in the future. All events are composed of a name, and time. The name has two parts: the type of the event for example *key* and the sub-type that defines the actual event for example *key-down*. The time that accompanies and event is an evaluation of the time interval related to the event. For example with keyboard events we describe whether the key-press event is long short or intermediate. We do not save the exact time that that event took place since it is not interesting. We do not expect a user to press a specific key at a specific time. However we do expect him to press a key for a similar length of time each time he presses the key. We define time intervals for each type of activity. All the time intervals are configurable and have an affect on the outcome of the algorithm.

The events that have been defined are:

- KEY

  - K-PRESS: pressing a key
  - K-WAIT: time between key presses
  - K-VIRTUAL: virtual key pressed

- MOUSE
  - M-MOVE: mouse movement
  - M-WAIT: time between mouse movements
  - M-CLICK: mouse click
  - M-DBL-CLICK: mouse double click

- WINDOW
  - W-FOCUS: window gets focus
  - W-CLOSE: window closed
  - W-OPEN: window opened

When logged data files are converted to events. The events are saved in sequences. The length of these sequences is configurable. These sets of sequences are the input used by our algorithm for modeling users and for finding invaders.

Appendix B

Implementation of REEF and CUBS

There are several executables that comprise the code used for implementation of the algorithms described in this thesis in Chapters 4 and 5. We describe all the executables and then how they are used to implement REEF and CUBS. An overview of the possible flow of the executable units is found in Fig. B.1. The full version of the code, data and scripts is available by contacting the authors.

Configuration File

The executables use a configuration file. The configuration file defines the following parameters:

- *DataType*- the data base used. Options are: Zapping 0, Upd 1, syn 2, handwriting 3, text 4.
- *NumUsers*- the number of classes (users) followed by their names.
- *DataDir*- the directory for reading input data.

- *OutDir*-the directory for placing output.
- *ModelFileName*- the name of model file, including directory.
- *RemoteData*- The type of remote control used for zapping data. Can be 0 or 1.
- *IsKeyUsed*- for UPD data only, defines if data from keyboard is used. 1 if used, 0 if not.
- *IsMouseUsed*- for UPD data only, defines if data from mouse is used. 1 if used, 0 if not.
- *IsWindUsed*- for UPD data only, defines if data from window activity is used. 1 if used, 0 if not.

GetEvents

This executable reads the raw data and converts it into sequences of events. These are the input sequences that are used by **Sample** and **GetBestSeq**. The parameters for **GetEvents** are

- *config*- The name of the configuration file to use.
- *len*- The size of input sequences, also names window size.
- *c name*- *c* is used to state that *name* must appear in all the data input filenames.
- *n idx0 idx1 ... idxn*- *n* indicates that one of the idx should appear in input file name.

  i.e "c avi n 1 2 3" means that input filenames are "avi-1-hot", "avi-2-hot", "avi-3-hot" but not "avi-4-hot" or "guy-1-hot".

Sample

This executable is the sampling unit that performs sampling of the data and gathers statistics. The statistics are used in **GetBestSeq**. The parameters for **Sample** are:

- *config*- The name of the configuration file to use.
- *sample*- The sample size.
- *thresh*- a threshold for mining, should always be set to 0.

GetBestSeq

This executable finds the frequent sequences. This is the main frequent sequence mining executable. It can use minimum support (m specified in command line) and then is an implementation of SPADE, or z-score normalization (b in command line) and then is an implementation of REEF. Uses outputs from **GetEvents** and must be run after running **GetEvents**. If run with the z-score option (REEF) it must be run after **Sample**. The output of **GetBestSeq** is used as frequent or norm-frequent sequences, and can be used by the **GetSignificant** executable. The parameters for **GetBestSeq** are:

- *config*- The name of the configuration file to use.
- *threshtype*-
  - **m**- minimum support (SPADE)
  - **b**-best z-score (REEF)
  - **t**-threshold z-score, this is an obsolete version of REEF with a threshold rather than using best.
- *minSupPer*percentage of seq to use for 'm' option.

- *BestNum* number of best z-scores to use for 'b' option.
- *Zthresh* threshold to use for 't' option (obselete).
- distortion correction parametrs- if(threshtype=='b')
  - *sampleName*- name of file with output from **Sample**.
  - *SampleFixTypeAvg*- distortion correction equation for average.
  - *SampleFixTypeStd*- distortion correction equation for standard deviation.
  - *samp*- sample percentage rate
  - *average parameters*- 5 parameters for average correction.
  - *std parameters*- 5 parameters for standard deviation correction.

examples:

1. GetBestSeq config.txt b 0 100 0 samp.txt 4 1 10 -1.664 -0.229 -0.067 -0.087 1.226 0.928 -0.799

   Run as REEF with configuration file config.txt, with *best*=100, sampling file samp.txt, using equations 4 and 1 (those are reported as selected in thesis), sampling rate of 10, average correction parameters:-1.664 -0.229 -0.067 -0.087 1.226 and std correction parameters: 0.928 -0.799.

2. GetBestSeq config m 10 0 0 0 0 0 0

   Run as SPADE with configuration file config.txt, with *minsup*=10, all other parameters are 0 but must appear.

GetSignificantSeq

This executable finds the significant sequences from a set of frequent (or norm frequent) sequences. Uses output from GetBestSeq, that must run first. Outputs a model used for classification.

The parameters for **GetSignificantSeq** are:

- *config*- The name of the configuration file to use.
- *sig*- the number of significant sequences to return.

ClassUnknownSeq

This executable performs classification of test data. uses input from **GetBestSeq** that has been run on the "unknown"(test) data using the SPADE 'm' option. Uses the model created by **GetSignificantSeq** run on the training data.

Running REEF or CUBS

In order to run REEF we run the following executables in the following order:
GetEvents
Sample
GetBestSeq

In order to run CUBS we run the following executables in the following order:
GetEvents (on *train* data)
Sample (on *train* data)
GetBestSeq (on *train* data)
GetSignificantSeq (on *train* data)
GetEvents (on *test* data)
GetBestSeq (on *test* data)
ClassUnknownSeq (on *test* data)

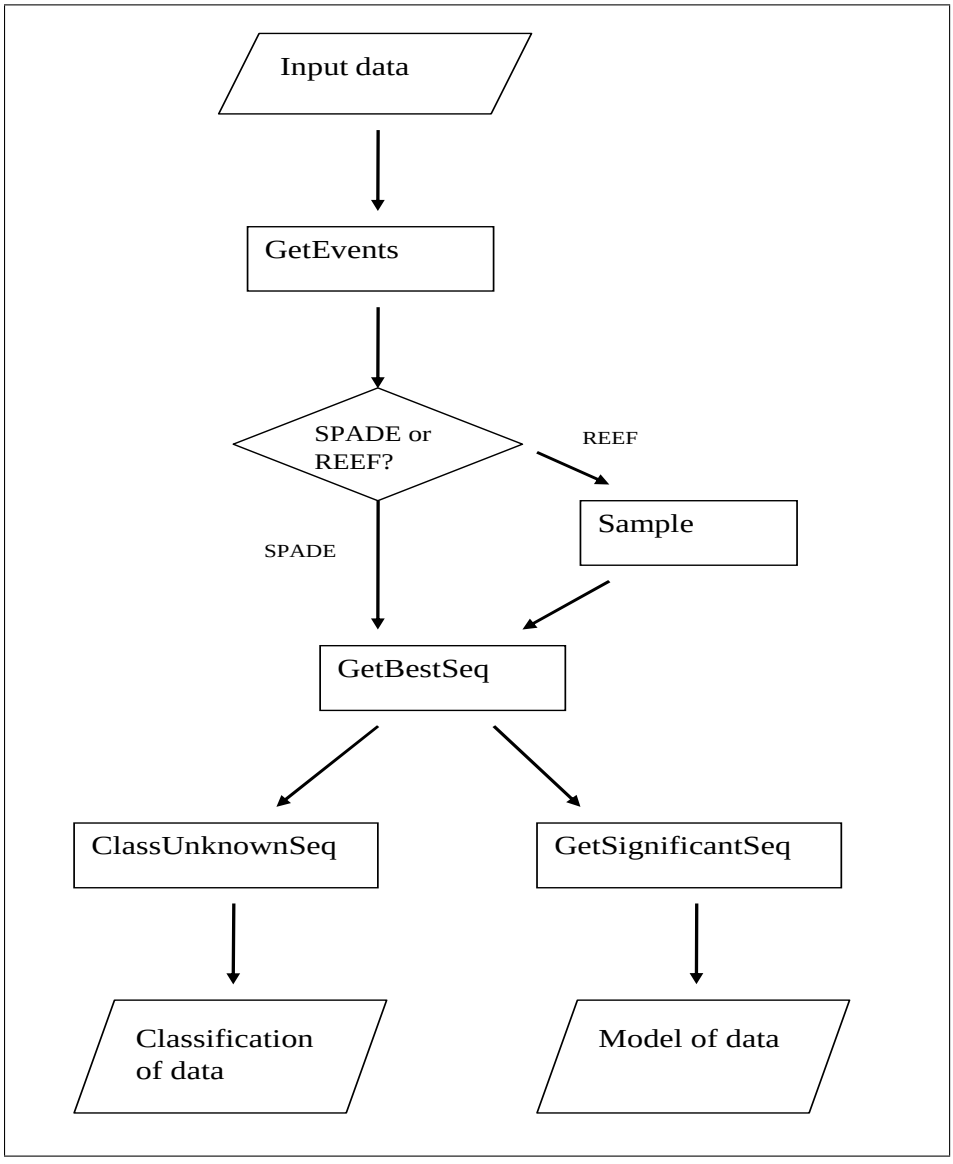


Figure B.1: Flow of executable units

Bibliography

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD '93 Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 22:207–216, June 1993.

[2] R. Agrawal, M. Mehta, J. Shafer, and R. Srikant. The QUEST data mining system. In *Proc. of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*, pages 244–249, 1996.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, 1995.

[4] A. E. Ahmed. A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4(3):165–179, 2007.

[5] D. Alberg and A. Ben-Yair. Online hoeffding bound algorithm for segmenting time series stream data. *Journal of Applied Quantitative Methods*, 5(3):446–453, 2010.

[6] D. Alberg, M. Last, and A. Ben-Yair. Induction of mean output prediction trees from continuous temporal meteorological data. *Journal of Applied Quantitative Methods*, 4(4):485–494, 2009.

[7] L. Ardissono, F. Portis, P. Torasso, F. Bellifemine, A. Chiarotto, and A. Difino. Architecture of a system for the generation of personalized electronic program guides. In *UM2001 Workshop on Personalization in Future TV (TV01)*, St-Louis, USA, July 2001.

[8] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick. Sequential pattern mining using a bitmap representation. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 429–435. ACM Press, 2002.

[9] C. Bahlmann. Directional features in online handwriting recognition. *Pattern Recogn.*, 39(1):115–125, 2006.

[10] P. Baudisch and L. Brueckner. Tv scout: Lowering the entry barrier to personalized tv program recommendation. In M. Hemmje, C. Niedersee, and T. Risse, editors, *From Integrated Publication and Information Systems to Information and Knowledge Environments*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005.

[11] R. A. Baxter, G. J. Williams, and H. He. Feature selection for temporal health records. *Lecture Notes in Computer Science*, 2035:198–209, 2001.

[12] A. Bharath, V. Deepu, and S. Madhvanath. An approach to identify unique styles in online handwriting recognition. In *ICDAR '05: Proceedings of the Eighth International Conference on Document Analysis and Recognition*, pages 775–779, Washington, DC, USA, 2005. IEEE Computer Society.

[13] Y. Blanco, J. J. Pazos, A. Gil, M. Ramos, A. Fernandez, R. P. Diaz, M. Lopez, and B. Barragans. Avatar: an approach based on semantic reasoning to recommend personalized tv programs. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1078–1079, New York, NY, USA, 2005. ACM Press.

[14] L. Carroll. Alice's Adventures in Wonderland. Project Gutenberg.

[15] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, September 1995.

[16] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(1-4):131–156, 1997.

[17] M. Deshpande and G. Karypis. Using conjunction of attribute values for classification. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 356–364, New York, NY, USA, 2002. ACM Press.

[18] C. Earl, S. Patrick, and P. I. A. Lloyd. Fuzzy logic based viewer identification. Patent No. WO/2007/131069, 2007.

[19] N. Erez and S. Parush. The hebrew handwriting evaluation (2nd ed.). Israel, Jerusalem: School of Occupational Therapy. Faculty of Medicine. Hebrew University of Jerusalem., 1999.

[20] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[21] J. Goerzen and O. Othman. Debian gnu/linux : Guide to installation and usage. Project Gutenberg.

[22] K. Gouda, M. Hassaan, and M. J. Zaki. Prism: A primal-encoding approach for frequent sequence mining. In *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 487–492, Washington, DC, USA, 2007. IEEE Computer Society.

[23] O. Guven, S. Akyokus, M. Uysal, and A. Guven. Enhanced password authentication through keystroke typing characteristics. In *AIAP'07: Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference*, pages 317–322, Anaheim, CA, USA, 2007. ACTA Press.

[24] R. Gwadera and F. Crestani. Discovering significant patterns in multi-stream sequences. In *ICDM '08: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 827–832, Washington, DC, USA, 2008. IEEE Computer Society.

[25] N. Haas, R. M. Bolle, N. Dimitrova, A. Janevski, and J. Zimmerman. Personalized news through content augmentation and profiling. *Proceedings of the 2002 International Conference on Image Processing (ICIP 2002)*, 2:9–12, Sept 2002.

[26] Y. Horman and G. A. Kaminka. Removing biases in unsupervised learning of sequential patterns. *Intelligent Data Analysis*, 11(5):457–480, 2007.

[27] R. Janakiraman and T. Sim. Keystroke dynamics in a general setting. In *Advances in Biometrics*, volume 4642/2007, pages 584–593, 2007.

[28] M. W. Kadous and C. Sammut. Constructive induction for classifying multivariate time series. In *15th European Conference on Machine Learning*, pages 192–204, 2004.

[29] M. K. Kalera, S. N. Srihari, and A. Xu. Offline signature verification and identification using distance statistics. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(7):1339–1360, 2004.

[30] K. Kee-Eung, C. Wook, C. Sung-Jung, S. Junghyun, L. Hyunjeong, J. Park, L. Youngbeom, and K. Sangryong. Hand grip pattern recognition for mobile

user interfaces. In *Proceedings of the 18th conference on Innovative Applications of Artificial Intelligence*, pages 1789–1794, 2006.

[31] C.-H. Lee and V. S. Tseng. PTCR-Miner: Progressive temporal class rule mining for multivariate temporal data classification. In *IEEE International Conference on Data Mining Workshops*, pages 25–32, 2010.

[32] J.-G. Lee, J. Han, X. Li, and H. Cheng. Mining discriminative patterns for classifying trajectories on road networks. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints), 2010.

[33] G. Lekakos, D. Papakyriakopoulos, and K. Chorianopoulos. An integrated approach to interactive and personalized tv advertising. In *Workshop on Personalization in Future TV*, 2001.

[34] Y. Liu, A. Niculescu-Mizil, A. Lozano, and Y. Lu. Learning temporal graphs for relational time-series analysis. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[35] E. Loekito, J. Bailey, and J. Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowledge and Information Systems*, 24:235–268, August 2010.

[36] C. Luo and S. M. Chung. A scalable algorithm for mining maximal frequent sequences using sampling. In *ICTAI '04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 156–165, Washington, DC, USA, 2004. IEEE Computer Society.

[37] C. Luo and S. M. Chung. A scalable algorithm for mining maximal frequent sequences using a sample. *Knowledge and Information Systems*, 15:149–179, May 2008.

[38] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences (extended abstract). In *1st Conference on Knowledge Discovery and Data Mining*, pages 210–215, 1995.

[39] F. C. Morabito and M. Versaci. Fuzzy neural identification and forecasting techniques to process experimental urban air pollution data. *Neural Networks - 2003 Special issue: Neural network analysis of complex scientific data: Astronomy and geosciences*, 16(3-4):493–506, 2003.

[40] T. Oates and P. R. Cohen. Searching for structure in multiple streams of data. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 346–354. Morgan Kaufmann, 1996.

[41] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440, 2004.

[42] M. Pusara and C. E. Brodley. User re-authentication via mouse movements. In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 1–8, 2004.

[43] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[44] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[45] C. Raissi and P. Poncelet. Sampling for sequential pattern mining: From static databases to data streams. *ICDM '07: Proceedings of the 2007 IEEE International Conference on Data Mining*, 0:631–636, 2007.

[46] A. Richardson, G. Kaminka, and S. Kraus. CUBS: Multivariate sequence classification using bounded z-score with sampling. In *IEEE International Conference on Data Mining Workshops*, pages 72–79, 2010.

[47] A. Richardson, S. Kraus, P. L. Weiss, and S. Rosenblum. COACH - cumulative online algorithm for classification of handwriting deficiencies. In *IAAI'08 Proceedings of the 20th national conference on Innovative applications of artificial intelligence*, pages 1725–1730, 2008.

[48] Rosenblum, S. Parush, S., and W. P.L. The in air phenomenon: temporal and spatial correlates of the handwriting process. *Perceptual Motor Skills*, 96(3 pt 1):933–954, Jun 2003.

[49] S. Rosenblum, P. L. Weiss, and S. Parush. Computerized temporal handwriting characteristics of proficient and non-proficient handwriters. *The American Journal of Occupational Therapy*, 57(2):129–138, Mar-Apr 2003.

[50] R. E. Schapire. The Boosting Approach to Machine Learning: An Overview. In MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA, USA, 2001.

[51] M. Seno and G. Karypis. LPMiner: An algorithm for finding frequent itemsets using length-decreasing support constraint. In *ICDM '01 Proceedings of the 2001 IEEE International Conference on Data Mining* , 2001.

[52] M. Seno and G. Karypis. SLPMiner: An algorithm for finding frequent sequential patterns using length-decreasing support constraint. In *ICDM '02: Proceedings*

of the 2002 IEEE International Conference on Data Mining, page 418, Washington, DC, USA, 2002. IEEE Computer Society.

[53] W. Shakespeare. A Midsummer Night's Dream. Project Gutenberg.

[54] A. Silvescu, C. Caragea, and V. Honavar. Combining super-structuring and abstraction on sequence classification. In *ICDM '9: Proceedings of the 2009 IEEE International Conference on Data Mining*, pages 986–991, 2009.

[55] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *EDBT '96 Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, pages 3–17, 1996.

[56] N. Tatti and B. Cule. Mining closed strict episodes. In *ICDM '10: Proceedings of the 2010 Tenth IEEE International Conference on Data Mining*, 2010.

[57] H. Toivonen. Sampling large databases for association rules. pages 134–145. Morgan Kaufmann, 1996.

[58] V. S. Tseng and C. Lee. Effective temporal data classification by integrating sequential pattern mining and probabilistic induction. *Expert Systems with Applications*, 36(5):9524–9532, 2009.

[59] P. Tzvetkov, X. Yan, and J. Han. TSP: Mining top-k closed sequential patterns. *Knowledge and Information Systems*, 7:438–457, May 2005.

[60] J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 79, Washington, DC, USA, 2004. IEEE Computer Society.

[61] J. Wang and G. Karypis. Bamboo: Accelerating closed itemset mining by deeply pushing the length-decreasing support constraint. In *Proceedings of the Fourth SIAM International Conference on Data Mining*, pages 432–436, 2004.

[62] A. Weiss, A. Ramapanicker, P. Shah, S. Noble, and L. Immohr. Mouse movements biometric identification: A feasibility study. Technical report, Ivan G Seidenberg School of CSIS, Pace University, 1 Martine Ave, White Plains, NY, 10606, USA, 2006.

[63] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

[64] Z. Xing, J. Pei, G. Dong, and P. S. Yu. Mining sequence classifiers for early prediction. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 644–655, 2008.

[65] U. Yun. An efficient mining of weighted frequent patterns with length decreasing support constraints. *Knowledge-Based Systems*, 21(8):741–752, 2008.

[66] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1/2):31–60, 2001.

[67] M. J. Zaki, N. Lesh, and M. Ogihara. PlanMine: Predicting plan failures using sequence mining. *Artificial Intelligence Review - Issues on the application of data mining*, 14(6):421–446, 2000.

[68] M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of sampling for data mining of association rules. In *RIDE '97 Proceedings of the 7th International Workshop on Research Issues in Data Engineering (RIDE '97) High Performance Database Management for Large-Scale Applications*, pages 42–50, 1997.

[69] Y. Zhao. Learning keystroke patterns for authentication. In *14th International Enformatica Conference*, volume 14, pages 65–70, 2006.