APRENDIZAGEM EFICIENTE DE CLASSIFICADORES SEQUENCIAIS EM PADRÕES LONGOS

GESSÉ SILVA FERREIRA DE DAFÉ

APRENDIZAGEM EFICIENTE DE CLASSIFICADORES SEQUENCIAIS EM PADRÕES LONGOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Adriano Alonso Veloso

Belo Horizonte

Janeiro de 2013

GESSÉ SILVA FERREIRA DE DAFÉ

LEARNING SEQUENTIAL CLASSIFIERS IN LONG-RANGE PATTERNS EFFICIENTLY

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

ADVISOR: ADRIANO ALONSO VELOSO

Belo Horizonte

January 2013

Dafé, Gessé Silva Ferreira de.

D124a Aprendizagem eficiente de classificadores sequenciais em padrões longos / Gessé Silva Ferreira de Dafé. — Belo Horizonte, 2013
xxiv, 43 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais. Departamento de Ciência da Computação.

Orientador: Adriano Alonso Veloso

1. Computação - Teses. 2. Mineração de Dados (Computação) – Teses. I. Orientador. II. Título.

CDU 519.6*72 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Aprendizagem eficiente de classificadores sequenciais em padrões longos

GESSÉ SILVA FERREIRA DE DAFÉ

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. ADRIANO ALONSO VELOSO - Orientador
Departamento de Ciência da Computação - UFMG

PROF. MOHAMMED JAVEED ZAKI
Rensselaer Polytechnic Institute

PROF. NIVIO ZIVIANI
Departamento de Ciência da Computação - UFMG

PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 07 de janeiro de 2013.

I dedicate this work to my parents, as they have dedicated their whole lives to love and educate me, over any other priority.

Acknowledgments

First, my utmost gratitude to God, my refuge and comfort in difficult moments of my life, source of all love, peace and hope that the human heart can feel.

In second place, I am thankful to those who I love most in this world: my parents, Magda and Darcy, who always did their best to lead me to the path of happiness and to whom I owe everything that I am. I also thank my beloved sister, Sara, for the encouragement words, and for the unconditional love and care on which I can always count.

Undoubtedly, I owe my deepest gratitude to my advisor, Adriano Veloso, from who I have never heard a discouraging word or a reprimand. By contrast, he has always been more than a mentor in this journey: he was a partner and a friend. I also thank Wagner Meira Jr. and Mohammed Zaki, for the immeasurable collaboration to this work.

Last but not least, I thank my great friends/advisers/squires Gisnálbert and Paulo Afonso, always present, always available to support me in whatever I needed... and I thank all my friends, whose hands are always ready to help, in hard times, or to propose a toast, in moments of success!

Thank you all!

"O correr da vida embrulha tudo. A vida é assim: esquenta e esfria, aperta e daí afrouxa, sossega e depois desinquieta. O que ela quer da gente é coragem."
(João Guimarães Rosa)

Resumo

Muitas aplicações, tais como extração de informação, detecção de intrusão e reconhecimento de enovelamento de proteínas, podem ser expressas como uma sequência de eventos (ao invés de um conjunto não-ordenado de atributos), ou seja, existe uma relação de ordem entre os elementos que compõem cada instância presente nos dados. Essas aplicações podem ser modeladas como problemas de classificação e, nesse caso, o classificador precisa ser construído de forma a ser capaz de capturar tais relações e usá-las como fonte de informação. As principais abordagens para esse problema incluem: (i) a aprendizagem de Modelos Ocultos de Markov, (ii) a exploracão de sequências frequentes extraídas dos dados e (iii) o cálculo de string kernels para Máquinas de Vetores de Suporte (SVMs). Essas abordagens, entretanto, são computacionalmente difíceis e a alta dimensionalidade, típica dos dados sequenciais, representa sérios desafios à viabilidade de tais métodos, em especial se os dados possuem dependências longas (i.e., padrões longos são necessários para modelar os dados). Neste trabalho apresentamos algorítmos que geram classificadores sequenciais de alta eficiência através da exploração dos conceitos de adjacência ou proximidade entre os elementos de uma sequência, a fim de aprimorar a acurácia ou garantir, juntamente com a limitação dinâmica dos tamanhos das sequências enumeradas, um custo de aprendizagem de $O(n\sqrt{n})$, onde n é a dimensão (número de atributos) da instância a ser classificada. Nossos algorítmos baseiam-se na enumeração sob demanda de padrões (aproximadamente) contíguos presentes nos dados de treino, usando um método flexível e leve de casamento de padrões e uma estratégia inovadora de enumeração que chamamos de*silhuetas de padrões*, que fazem com que nossos classificadores sejam rápidos porém robustos mesmo em dados ruidosos. Nossos resultados empíricos, obtidos sobre conjuntos de dados reais, mostram que, na maioria dos casos, nossos calssificadores são mais rápidos que as soluções existentes (em alguns casos, ordens de grandeza mais rápidos) e proporcionam ganhos significativos de acurácia.

Abstract

Many applications, such as information extraction, intrusion detection and protein fold recognition, can be expressed as sequences of events or elements (rather than unordered sets of features), that is, there is an order dependence among the elements composing each data instance. These applications may be modeled as classification problems, and in this case the classifier must be built using sequential interactions among the elements, so that the ordering relationship among them is properly captured. Dominant approaches to this problem include: (i) learning Hidden Markov Models, (ii) exploiting frequent sequences extracted from the data and (iii) computing string kernels for Support Vector Machines. Such approaches, however, are computationally hard, and the typically high-dimensional nature of sequential data poses serious challenges to their feasibility, especially if the data shows long range dependencies (i.e., long patterns are necessary in order to model the data). In this paper we introduce algorithms that build highly effective sequential classifiers by exploiting adjacency or proximity information, either to improve classification accuracy or to ensure $O(n\sqrt{n})$ learning cost, where n is the dimension (number of features) comprising a given test instance. Our algorithms are based on enumerating (approximately) contiguous sequences from the training data on a demand-driven basis, exploiting a lightweight and flexible sequence matching function and an innovative sequence enumeration strategy called *pattern silhouettes*, which make our classifiers fast but also robust even in noisy data. Our empirical results on actual datasets show that, in most of the cases, our classifiers are faster than existing solutions (sometimes orders of magnitude faster), also providing significant accuracy improvements in most of the evaluated cases.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Machine Learning is the term used to define a wide research area whose interest is to develop techniques that allow machines to discover, with some degree of autonomy, solutions to computational problems which cannot be solved (or are very difficult to solve) by direct implementation. A machine learning process is said to be supervised when it is based on programming the machine to solve a problem by learning with examples of similar problems solutions. If these solutions assume predetermined values, we call that process Classification. In a classification problem, each sample provided to the machine comprises an input, representing an instance of the problem, and an output, representing the solution related to that instance. An input may be composed of one or more attributes and each possible output value is said to be a Class.

Classification algorithms usually interpret each training example as an unordered set of features. In many applications, however, a training example is represented as a sequence of events or elements, and, therefore, there is an explicit ordering relationship among these elements. This is commonly observed in application scenarios such as information extraction, trajectory prediction and protein fold recognition [Han et al., 2004, 2005; Lodhi et al., 2009; Silva et al., 2011]. In such cases, it may be advantageous to interpret examples as sequences of elements, since the order relation may contain relevant (or even necessary) information about the meaning of data. Just to illustrate this intuition, the following fictitious sentences have opposite meanings, despite being composed of the same words. There is no difference between them except the order in which words happen:

> *"Netanyahu does not want the war, he wants the peace agreement."*
> *"Netanyahu wants the war, he does not want the peace agreement."*

Algorithms that produce classifiers from sequential data (i.e., sequential classifiers) may rely on (i) building generative classifiers by learning Hidden Markov Models [Rabiner, 1989], (ii) transforming the original feature space into another one in which each transformed feature corresponds to a frequent sub-sequence of elements in the original space [Tseng and Lee, 2005], (iii) combining frequent pattern mining and Hidden Markov Model approaches [Zaki et al., 2010] and (iv) computing string kernels for Support Vector Machines [Lodhi et al., 2002]. These algorithms, however, may suffer from non-negligible shortcomings, possibly preventing them to capture long range sequential dependencies in the data, provide fast learning times or deal with noisy sequential data. Furthermore, such algorithms are often devised for a specific learning task [Agrawal and Srikant, 1995; Eddy, 1998], or depend on a non-trivial set of parameters [Zaki et al., 2010] (e.g., minimum support, maximum sequence length and maximum gap).

In this thesis we propose general purpose sequential classification algorithms that exploit the proximity among elements within the training examples in order to learn sequential classifiers that are efficient both in terms of classification accuracy and learning time. Proximity information is used as a pruning strategy to determine only related co-occurrence patterns, and in addition being time efficient. The proposed algorithms also yield improvements in terms of classification effectiveness, showing that sequences composed of contiguous (or adjacent) elements may be discriminative enough to produce highly effective sequential classifiers. By dynamically bounding the length of these sequences so that the corresponding classifier captures not only short, but also long range dependencies in the data, we show that *contiguous sequential classifiers* can be learned in $O(n\sqrt{n})$ time, where n is the number of elements within a test instance. A drawback, however, is that contiguous sequential classifiers may become excessively restrictive, since no violation in the ordering relationship among elements is allowed while extracting the sequences. As a result, the classifier is usually composed of very few sequences, being not sufficiently expressive or very sensitive to noise, compromising classification effectiveness. To overcome this limitation we relax the sequence enumeration process in order to extract *approximately* contiguous sequences. We formulate a lightweight matching function which compares a test instance with the relevant examples in dataset and measures the similarity betwen them. The similarity is based on how much contiguous matching features they have in common. Violations in the contiguity of the sequences (i.e., mismatches) are allowed and properly penalized. Relevant examples are extracted from the dataset in a demand-driven basis, by indexing what we call pattern silhouettes. As a result, sequential classifiers become

more expressive, but are still efficiently learned in $O(n\sqrt{n})$.

To validate our claims and to evaluate the effectiveness of the proposed algorithms, we performed a systematic set of experiments using real sequential data obtained from a variety of application scenarios such as information extraction, protein fold recognition, intrusion detection, spelling correction, among others. The results show that our proposed algorithms provide, in most of the cases, significant improvements in terms of execution time, without putting classification accuracy at risk when compared against state-of-the-art solutions. In fact, in most of the cases, our proposed algorithms are able to improve classification accuracy as well.

The rest of this thesis is organized as follows. In Chapter 2 we cover some basic concepts and then discuss existing approaches that are closely related to our work. In Chapter 3 we provide an in-depth description of our proposed method. In Chapter 4 we present and discuss our experiments and results. Finally, in Chapter 5 we offer conclusions and possible directions for future work.

Chapter 2

Preliminaries

In a single-label [1] classification problem, each example provided to the machine comprises an input (a set of features) and an output (a class). To make the machine learn and therefore be able to infer the class to which a given unforeseen input belongs, it needs to find, among the existing examples, a suitable function that can map inputs to outputs. A classification algorithm aims to find such a function (or a set of functions) that represents, as general and accurately as possible, the relationship between inputs and outputs contained in a set of examples. However, the search space for this mapping function is potentially unlimited. Randomly searching mapping functions is not practicable: it is essential to narrow it down.

According to the strategy used to narrow down the search space of mapping functions, classification algorithms can be grouped into different categories. The most popular categories include: algorithms based on distance metrics - known as KNNs (K-Nearest Neighbors) [Cover and Hart, 1967; Weinberger and Saul, 2009], algorithms based on decision trees [Quinlan, 1979, 1993; Breiman et al., 1984], techniques based on hyperplanes for discrimination of examples (Support Vector Machines) [Cortes and Vapnik, 1995; Chang and Lin, 2011a], models based on perceptrons [Rosenblatt, 1958; Hopfield, 1982; Kohonen, 1982], statistical models [Domingos and Pazzani, 1997; Friedman et al., 1997] and algorithms that exploit association rules between attributes and classes [Li et al., 2001a; Han et al., 2000; Veloso et al., 2006].

The algorithms we propose in this work has their origins in associative classification, so they rely on several concepts related to pattern mining. In this chapter we present definitions and concepts that will help the reader to better understand the

[1] There is also the multi-label classification problem, where more than one class is predicted for a same test instance. Our work focuses in single-lable prediction, although it can also be applied in a multi-label scenario.

context of this work. First, we present definitions related to pattern mining, next we discuss algorithms that use those strategies and finally we present the works that are most closely related to our proposal.

2.1 Basic Concepts

In data mining and associative classification tasks, different strategies of pattern enumeration can be used to extract information from data. In this section, we present those strategies which comprise the background to better understand our work.

2.1.1 Simple Patterns

In data mining, the extraction of association rules is a popular method to discover interesting relationships between the attributes that compose a database. In simple terms, association rules represent a relationship having the form $A \rightarrow B$, whose semantics can be interpreted as "when A occurs, B also occurs". Among the possible ways to model A (antecedent) and B (consequent), the simplest way is the concept of *itemset* or *simple patterns*. Given a set of attributes $I = \{i_1, i_2, ..., i_m\}$, where $m > 0$, a itemset is any non-empty subset of I. Being D a database comprising a collection of instances, where each instance has the form $I = \{i_1, i_2, ..., i_m\}$, a given itemset is said to be frequent if it is a subset of, at least, $k\%$ of the instances in D. In this case, k is a predefined value, called *minimal support*.

Because of its exponential computational complexity, the enumeration of simple patterns has gained notoriety since the publication of the Apriori algorithm [Agrawal et al., 1994], which proposed the concept of minimal support and the anti-monotonic property of patterns frequency to achieve a viable computational cost in mining association rules.

2.1.2 Sequential Patterns

Originally used to detect purchase patterns of consumers in large retail companies [Agrawal and Srikant, 1995], *sequential patterns* are subject of great scientific interest since the mid-90s, being applied in various fields since then [Guralnik and Haigh, 2002; Ezeife and Lu, 2005; Kim et al., 2008; Shie et al., 2011]. When used to solve problems whose data modeling enables the concept of order, sequential patterns are able to improve the quality of association rules and reduce the computational cost of

this process [Agrawal and Srikant, 1995; Srikant and Agrawal, 1996; Li et al., 2001b; Zaki, 2001; Pei et al., 2004].

To understand the concept of sequential patterns, we first need to address the concept of sequence. If $I = \{i_1, i_2, ..., i_m\}$ is a set of items, we call *event* a non-empty and unordered subset of I, represented by the notation $e = \{i_1, i_2, ...i_k\}$. Finally, a sequence s is an ordered list of events, which can be represented by the notation $s = e_1 \rightarrow e_2 \rightarrow ... \rightarrow e_n$, where the symbol $\rightarrow$ represents a relationship whose semantics can be defined as "occurs after". A sequence containing k items is called k-sequence, where $k = \sum_i |e_i|$.

Another important concept is the notion of sub-sequence. A sequence s is said to be a sub-sequence of another sequence r when two assumptions are met: (1) all the events in s are contained in r or are subsets of events in r; (2) the order in which the events occur in r is unchanged in s. When s is a sub-sequence of r, we can say that r contains s.

A database D, from which sequences can be mined, is a collection of input sequences, each one having an unique identifier sId. The events of these sequences are also uniquely identified by singular identifiers called eId. If s is a sequence in D, we say that the support of s, denoted by $\sigma(s, D)$, is the total number of sequences in D that contain s. If this number is higher than a certain threshold (minimum support), we say that s is a frequent sequence. Finally, given a database and a predefined minimum support, the problem known as mining frequent sequential patterns is to find all frequent sequences contained within such database.

2.1.3 N-Grams

Patterns known as n-grams have been used for decades and with notorious success in various applications in Computer Science. In particular, n-grams are widely used in domains related to language and information retrieval. Tasks such as spelling correction [Riseman and Hanson, 1974], document categorization [Damashek et al., 1995] and extraction of opinions [Pak and Paroubek, 2010] are examples of problems where n-grams can be successfully applied. The formal definition of a n-gram is quite simple: given a list of items $I = \{i_1, i_2, ..., i_m\}$, whose order is well known, we call n-gram any sub-list $S \subseteq I$ such that $|S| = n$. In applications involving information retrieval, n-grams of size 2 (bigrams) and 3 (trigrams) are widely reported as providing better results than longer n-grams [Pedersen, 2001; Dave et al., 2003; Halácsy et al., 2007].

2.2 Related Work

In this section we mention works that are related with our proposal. We start by talking about associative classifiers, which are the origins of our algorithms. Next we go deeper on the question of dealing with sequential data, which is the main target of our work. Next, w present an overview of the most relevant and closely related methods to address this question.

2.2.1 Associative Classification

The associative classification method is based on finding patterns of attribute co-occurrences among the examples and discover how strongly each pattern is related to each class. A big challenge in designing associative classifiers resides in its computational cost: the number of possible patterns in a training set grows linearly regarding to the number of examples but exponentially with their number of features. More precisely, a hypothetical full-range classifier C, which investigates all possible patterns in training, would have to enumerate the following number of patterns:

$$|C| = |X| \times \sum_{k=1}^{n} \binom{n}{k} = |X| * 2^n \tag{2.1}$$

where X is a given training set, containing $|X|$ examples and where each example contains n features. Due to this impeditive cost, an associative classifier which investigates all possible patterns probably will not be suitable in most real-world domains. Fortunately, there are strategies which can be used to reduce the number of patterns to be investigated. Among these strategies is the Demand-Driven Associative Classification.

2.2.2 Demand-Driven Associative Classification

The LAC algorithm (Lazy Associative Classifier) [Veloso et al., 2006] proposes a set of strategies aimed at making the process of associative classification efficient. The main strategy, which gives the name to the algorithm, consists in the fact that the rule extraction is made on demand: only when an instance is submitted for classification, the rules applicable to that entry are extracted from the training. Thus, the search space is filtered, considering only those instances whose attribute set has intersection with the set of attributes comprising such test. In addition to this strategy, the LAC algorithm uses pruning mechanisms to limit the number of rules extracted from the training and thus significantly reduce the computational cost of the task.

Table 2.1. Total Number of Patterns in a Single Test Instance

	$\|\mathbf{t}\| = \mathbf{5}$	$\|\mathbf{t}\| = \mathbf{10}$	$\|\mathbf{t}\| = \mathbf{25}$	$\|\mathbf{t}\| = \mathbf{50}$
$\mathbf{m = 1}$	5	10	25	50
$\mathbf{m = 2}$	15	55	325	1,275
$\mathbf{m = 3}$	25	175	2,625	20,875
$\mathbf{m = 4}$	30	385	15,275	251,175
$\mathbf{m = 5}$	31	637	68,405	2,369,935

A pruning mechanism of fundamental importance used by LAC algorithm is the imposition of a maximum size (number of features) for the rules being extracted. According to this strategy, the number of features comprising a rule must be parametrically set. Rules having more features than this threshold are not investigated. So, for m being the maximum number of features comprising a rule, the total number P of patterns that must be investigated, in order to classify a single test instance t, is given by:

$$P = \sum_{k=1}^{m} \binom{|t|}{k} \tag{2.2}$$

We can easy notice that the maximum rule size strategy is not enough to avoid an impracticable computational cost when dealing with high-dimensional data. In such cases, the number of feature combinations in a single test instance become excessively large, unless we constrain m to extremely modest values. Looking at the Table 2.1, we can have a clearer understanding of the impact of high-dimensionality on computational cost. The Table shows the number of patterns investigated when classifying a single test instance t, regarding to its number of features $|t|$ and arbitrary (despite very low) m values.

Clearly, when we use the algorithm LAC in areas of high dimensionality, we are forced to apply very low values of m. In other words, we work only with very short patterns, eventually discarding potentially interesting patterns on behalf of computational feasibility.

2.2.3 Approaches based on Sequential Patterns

In the mid 90s, there was a strong demand for solutions to problems involving detection of patterns and extraction of association rules in corporate databases [Agrawal et al., 1993]. Experts in Data Mining and Machine Learning endeavored to create algorithms that could make these tasks computationally feasible. An important milestone in this research area was the Apriori algorithm [Agrawal et al., 1994], which consolidated the

strategy of incremental pattern mining. This strategy is based on the anti-monotonic property of patterns frequency: if a set is infrequent, then all its super-sets will also be.

The problem of enumerating sequences that occur frequently in the data was first studied by Agrawal and Srikant [1995]. Improved algorithms for finding frequent sequential patterns were proposed in [Srikant and Agrawal, 1996; Zaki, 2000; Antunes and Oliveira, 2003]. These algorithms employ constraints such as minimum and maximum gaps between consecutive elements, allowing for a more flexible enumeration of sequences (i.e., sequences that occur after some given time interval). The use of sequential patterns as features for the sake of learning sequential classifiers was initially proposed in [Lesh et al., 1999], and more recently in [Lin et al., 2009]. As a major deficiency of these algorithms, it is worth mentioning that they are unable to extract high-order, long sequential patterns efficiently, therefore capturing only short-range dependencies in the data.

Algorithms based on the recursive data mining approach, proposed in [Szymanski, 2004], are able to bridge large gaps between consecutive elements in the sequence, but these algorithms were devised for solving specific tasks, such as intrusion detection and author identification.

In [Syed et al., 2009] the authors proposed an algorithm which efficiently finds patterns in labeled sequences, that is then used for classification of sequential data. In [Bannister, 2007] an algorithm is proposed to learn classifiers from sequential data by exploiting the algorithmic relationship between association rule mining and sequential pattern mining. Frequent sequential patterns are mined and then the constrained adaptive methodology is applied to select patterns to be used for classifying the outcome. This algorithm, which we call ASC (Associative Sequential Classifier), is a representative of state-of-the-art solutions for several applications based on sequential data. We implemented a similar version of this algorithm and used it as one of the baselines for comparison.

Efficiently capturing long-range dependencies among items and dealing with sequence violations pose a very challenging problem. Not only methods based on mining sequential patterns suffer to overcome these issues. Next we discuss solutions based on learning Hidden Markov Models, which try different strategies to handle this problem.

2.2.4 Approaches based on Hidden Markov Models

Hidden Markov models (HMMs) are probabilistic models of sequential data [Rabiner, 1989]. HMMs can be viewed as stochastic generalizations of finite-state automata, when both the transitions between states and the generation of output symbols are governed by probabilistic mechanisms. Specific HMM based approaches have been proposed for different applications, such as DNA and protein modeling [Durbin et al., 1998; Hughey and Krogh, 1996], speech recognition [Rabiner, 1989; Sha and Saul, 2006], handwritten character recognition [Hu et al., 1996], gesture recognition [Müller et al., 2000], among others.

In order to capture long range sequential dependencies, which is not feasible with simple (first-order) HMMs, many approaches based on fixed high-order HMMs have been proposed. In [Kriouile et al., 1990], a first-order HMM, based on Viterbi and Baum-Welch algorithms [Durbin et al., 1998], is used for state prediction and to directly train a second-order HMM. The method proposed by [Du Preez, 1998] converts a fixed high-order HMM to an equivalent first-order model which is used to incrementally train a high-order HMM. Another fixed-order approach is found in [Law and Chan, 1996], a n-gram-based HMM for language modeling. Although providing an elegant and sound sequential data modeling methodology, a major drawback in fixed high-order HMMs is that such models suffer from high state-space complexity, since a k-th order HMM, with alphabet Σ, can potentially have $|\Sigma|^k$ states. Therefore, estimating the joint probabilities of each k-th order state is extremely difficult. Furthermore, none of those techniques are able to capture eventual mismatches in sequences, maybe being excessively rigid to deal with noisy data.

Other approaches try to efficiently build variable-order HMMs. Mixed order Markov models were proposed by [Schwardt and Preez, 2000], but, since they rely on expectation-maximization methods, they are susceptible to local optima. In [Srivatsan et al., 2005], specific Episode Generating HMMs (EGMs) are built for each frequent episode (sequence) mined from the data. Besides the huge number of generated models, only non-overlapping sequences are found and violations in sequences are not explicitly handled. In [Bicego et al., 2003a], a pruning strategy was proposed to avoid excessively large number of states in HMMs. In [Wang et al., 2006], a variable-length HMM (VLHMM) is built upon Markov chains of variable memory length [Bühlmann and Wyner, 1999], by storing context in a prediction prefix tree. This method also employs an expectation-maximization strategy for training, so is prone to local optima. Furthermore it requires the number of states to be given as user parameter. The similarity-based recognition paradigm was also used in the context

of HMMs, resulting in significant improvements with respect to standard HMM-based approaches when applied to clustering, pattern recognition and sequential classification related tasks [Bicego et al., 2006, 2003b, 2004].

In [Zaki et al., 2010], the authors proposed the VOGUE algorithm, which addresses the main limitations of HMMs. This algorithm proposes a mixture between the two approaches: mining sequential patters and learning HMMs. It relies on a variable gap sequence mining approach, extracting frequent sequential patterns with different lengths and gaps between elements. The extracted patterns are then used to build a variable order HMM. The VOGUE algorithm is a representative of state-of-the-art solutions for several applications based on sequential data, and is used as one of the baselines for our proposed algorithms.

2.2.5 Approaches based on String Kernels for Support Vector Machines

The first approaches that used SVMs in sequential data were devised to text categorization task [Joachims, 1998]. Those approaches were based on transforming each training sample in a vector of features, where each coordinate usually depicted the presence or absense of a given word. Stop words removal and stemming were commonly combined to reduce the data dimensionality. Such strategy was not able to capture the ordering relationship among the words and thus lost relevant information from the text.

Later works introduced the concept of stirng kernels that have been used for text classification [Lodhi et al., 2002] and also for general purposes [Watkins, 1999; Haussler, 1999]. However, the cost of computing each kernel scales quadratically $(n * m)$ in relation to the size of the input sequences (n and m, respectively), being unfeasibly slow for applications were sequences are long. Furthermore, they do not handle mismacthes between sequences, being excessively sensitive to noise.

Other approaches try to overcome the strict matching disadivantages by employing mismatch-tolerant comparison techniques for building more flexible kernels. However, as these methods aim at computing similarity for all pairs of sequences in a particular feature space, they face a serious computational challenge. To narrow down the cost of this task, these techniques typically restrict the length and expressive power of the subsequences used as features, by indexing subsequences that they call k-mers, where k is a subsequence length and usually a reasonable small number. An example of these approches can be found in Spectrum-k Kernel [Leslie et al., 2002], which implicitly compares k-mers, where k is a parameter of the model. Starting from

the same essence, the Mismatch Kernel [Leslie et al., 2004] generalizes the Spectrum-k Kernel by allowing mismatches to accomodate mutations. The maximum number of consecutive mismatches is limited parametrically by the user. The Sparse Spatial Sample Kernel (SSSK) [Kuksa et al., 2008] generalizes the Mismatch Kernel by sampling the sequences at different resolutions and comparing the resulting spectra. Mismatches are handled by probes, whose number, lengths and distances are given as model parameters.

2.3 Motivation

All aforementioned strategies suffer to extract information from long-range and possibly noisy sequences. The hypothesis that motivates this work is the intuition that, in many real-world applications, the information contained in the ordering relationship between the features comprising samples is enough to allow a drastic pruning in the pattern search space and thus leading to faster execution times without putting accuracy at risk. We also claim that it is possible build light-weight mechanisms for partial matching patterns, which can be used to deal with noisy data, improving accuracy without making the computational cost impracticable, in contrast with other approaches.

2.4 Objectives

The main objective of this work is to build algorithms that confirm our claims. The second but no less important, is to evaluate the performance of the proposed algorithms in different domains, identifying the situations in which they can lead to gains in accuracy and time.

Chapter 3

Learning Demand-Driven Sequential Classifiers

The problem of learning sequential classifiers can be formally stated as follows:

> Given a training set D composed of d groups of sequences $D = \{D_1, D_2, \ldots, D_d\}$ such that all sequences that belong to the same group D_i have the same class label y_i, learn a classifier C which maps an arbitrary sequence to the most likely class label.

This formulation is sufficiently general to be applied to a wide variety of application scenarios where sequential data exists [Syed et al., 2009]. We modify this formulation in order to make explicit the notion of learning classifiers on a demand-driven basis, that is, a specific classifier is built for each test instance.

> Given a training set D composed of d groups of sequences $D = \{D_1, D_2, \ldots, D_d\}$ and a test set T, learn a classifier C_t, which is a function mapping sequence t to the most likely class label, for all sequences $t \in T$.

Next we discuss classification algorithms that learn sequential classifiers under this formulation. We start with a very simple algorithm, called SC-SC (Strictly-Contiguous Sequential Classifier), which learns sequential classifiers based on contiguous sequences, that is, sequences composed only of adjacent elements. Although simple, the SC-SC algorithm is very effective, mainly because it is able to enumerate sequences without relying on minimum support thresholds, being thus able to exploit longer and highly discriminative sequences. Then, we introduce a

more sophisticated algorithm, called AC-SC (Approximately-Contiguous Sequential Classifier), which learns classifiers by allowing the inclusion of approximate sequences, that is, approximate matches with the test instance are allowed. A similarity function is used to assess the amount of contiguity that exists between the test instance and the enumerated sequences, making AC-SC specially suited for dealing with noisy data.

3.1 Learning Strictly-Contiguous Sequential Classifiers

In this section we describe the SC-SC algorithm which benefits from the explicit ordering among elements to drastically reduce the number of sequences that need to be enumerated, improving learning time while increasing learning effectiveness. A very simple way to preserve the ordering relationship among elements and to reduce learning time is to perform contiguous matching, as defined next.

Definition 1. [Contiguous Matching] A sequence X is said to contiguously match instance $t \in T$ (which is given as a sequence of n elements $\{a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n\}$), if $X = \{a_i \rightarrow a_{i+1} \rightarrow \ldots \rightarrow a_{i+k}\}$, provided that $k \geq 0$ and $\forall$ pair $(a_i, a_{i+1}) \in X$, a_{i+1} immediately follows a_i in instance t.

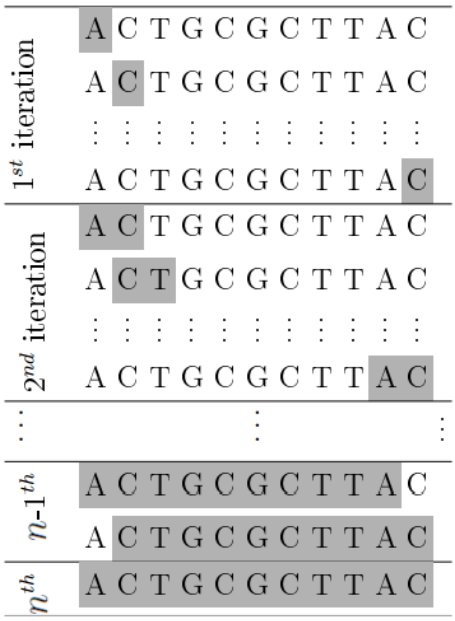
The strategy for enumerating strictly contiguous sequences may be seen as an iterative sliding-window process in which the window size is increased after each iteration. More specifically, given a test instance $t \in T$ such that $t = \{a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n\}$, the sequence enumeration process starts by enumerating singleton elements that are in t. In the second iteration, the window size increases and sequences composed of pairs of consecutive elements are enumerated. The process iterates enumerating ever increasing contiguous sub-sequences of instance t. Figure 3.1 shows the enumeration process given an arbitrary test instance.

Enumerating Contiguous Patterns

As easily noticed, the number of sequences that contiguously match an arbitrary instance $t \in T$ is given by an arithmetic progression which clearly grows quadratically with the number of elements within t (i.e., n). Therefore, the cardinality of C_t (i.e., the classifier built for instance t) is given by Equation 3.1:

$$|C_t| = \sum_{k=1}^{n} n - k + 1 = \sum_{k=1}^{n} k = \frac{n^2 + n}{2} = O(n^2) \tag{3.1}$$

Figure 3.1. Sliding Window Sequence Enumeration



In practice however, given a test instance $t \in T$, there is no need for an exhaustive enumeration of all contiguous sequences in t: classification accuracy and sequence length are not linearly related [Malik and Kender, 2008]. In fact, classification accuracy typically increases slowly (and eventually stabilizes) as the sequences composing the classifier become longer [Tseng and Lee, 2005]. Therefore, we may bound the length of the sequences to be enumerated, so that no sequence with more than m features is enumerated from D. We employ a variable limitation strategy, skipping subsequences with more than $\sqrt{n}$. This ensures $O(n\sqrt{n})$ learning cost with respect to the number of elements in t (i.e., n), as can be seen in equation Equation 3.2.

$$\sqrt{n} \implies |C_t| = (n) + (n-1) + (n-2) + ... + (n-\sqrt{n}) = O(n\sqrt{n}) \tag{3.2}$$

Other variable limits could also be used. Figure 3.2 shows the trade-off between accuracy, time and sequence length, for the SC-SC algorithm, in each dataset used in our evaluations. Limitation strategies evaluated include linear functions ($\frac{n}{10}$, $\frac{n}{2}$ and n), logarithmic functions ($log_2 n$ and $log_{10} n$) and the square root function. As noticed, better balanced results are reached for $\sqrt{n}$ and $log_2(n)$. We adopt the first strategy since it is less restrictive and is expected to be more expressive in datasets predominantly comprised of small sequences. More details about the employed datasets are given in

Section 4.1.

Calculating Class Membership

For each sequence $X \in C_t$, we calculate $\theta(X \rightarrow y_i)$, which is the conditional probability of X being associated with class label y_i (i.e., the probability of X being in D_i). Such sequences are viewed as weighted votes, where the weight depends on the θ value associated with the corresponding sequence. Weighted votes for label y_i are averaged, giving a score for label y_i with regard to instance t, as shown in Equation 3.3:

$$w(t, y_i) = \sum \frac{\theta(X \rightarrow y_i)}{|C_t^{y_i}|} \tag{3.3}$$

where $C_t^{y_i}$ are those sequences in C_t that are associated with class label y_i. Finally, the scores are normalized, as expressed by the scoring function $\hat{p}(y_i|t)$, shown in Equation 3.4. The scoring function estimates the likelihood of label y_i being the correct label for instance $t \in T$.

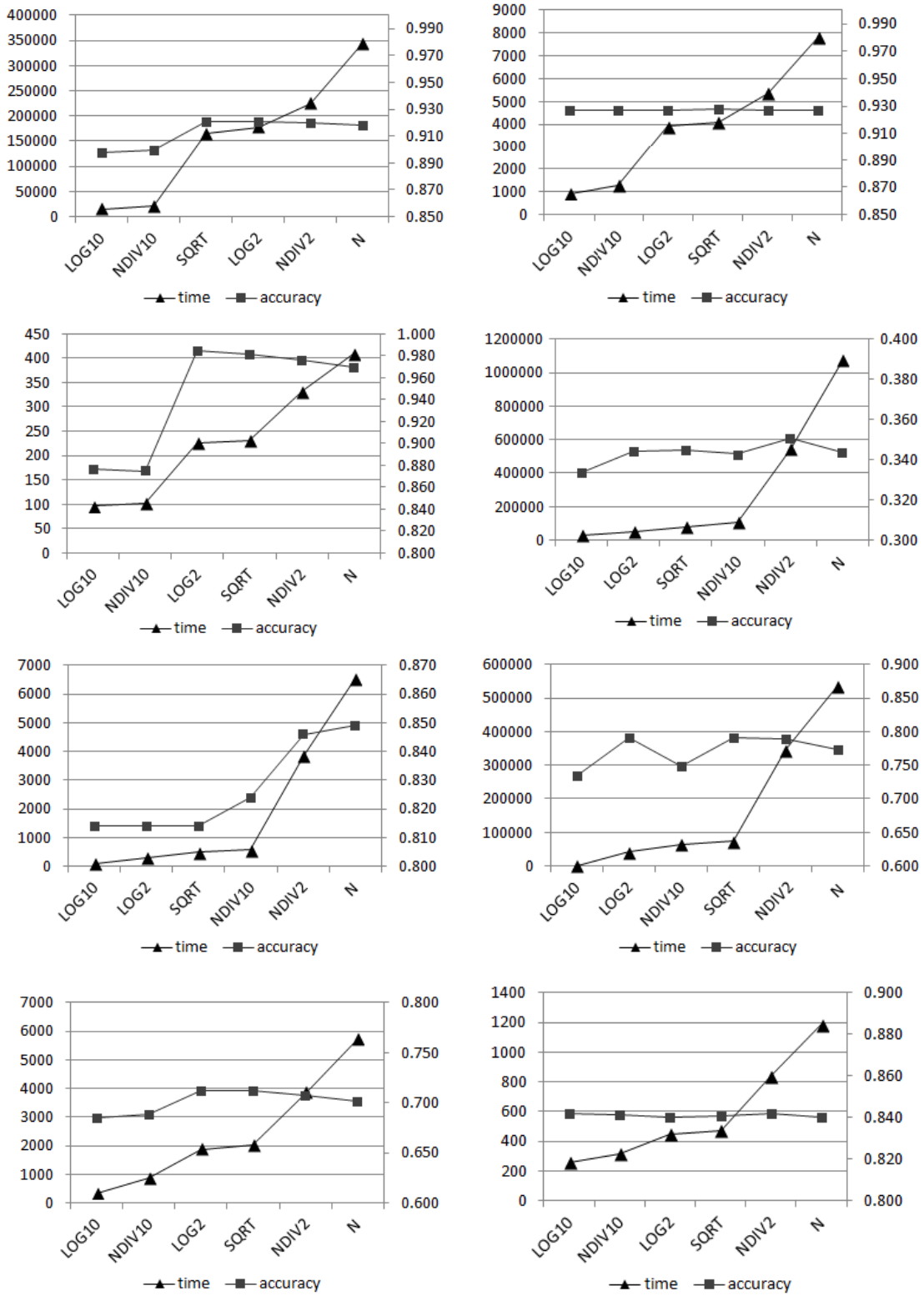
$$\hat{p}(y_i|t) = \frac{w(t, y_i)}{\sum_{j=0}^{n} w(t, y_j)} \tag{3.4}$$

Despite being computationally efficient, the SC-SC algorithm may produce sequential classifiers that are excessively restrictive, since no violation in the ordering relationship among elements is allowed while enumerating sequences. A single mismatch between a test instance and a training example causes the rejection of several (potentially relevant) sequences. As a result, the classifier will be probably very sensitive to noise and be composed of very few sequences, compromising classification accuracy. Fortunately, these disadvantages can be avoided by relaxing the sequence enumeration process without sacrificing learning time, as discussed in the following.

3.1.1 Learning Sequential Classifiers by Accessing Similarity Between Instances

In contrast to strictly-contiguous sequential classifiers, approximately-contiguous sequential classifiers are lenient with mismatches when enumerating sequences. We propose a classification algorithm for learning approximately-contiguous sequential classifiers, which relies on assessing the similarity between sequences. This similarity is calculated for each pair (x, t) such that $x \in D$ and $t \in T$. The proposed AC-SC algorithm works in three main steps as described next.

Figure 3.2. Trade-off between accuracy, time (ms) and sequence length limitation strategy. Datasets employed (in order): Dilma Rousseff, Felipe Melo, Author Name, Protein Fold, Protein Family, Intrusion, Spelling and Web Log.



Pre-Indexing Pattern Silhouettes

Scanning the entire training set D searching for approximate sequences every time an instance $t \in T$ is given to classification is obviously unfeasible. Since we are interested in dealing with approximately-contiguous matching, which we define next, optimizing sequence enumeration by pre-indexing sequence occurrences in D is not an option. The challenge here is to narrow down the search space for sequences, by investigating only training examples $x \in D$ that have the same shape of the test instance being classified. The way we propose to solve this problem is pre-indexing what we call *Pattern Silhouettes*, which can be seen as a kind of hash function designed to tell which training examples *may be* similar to a given test instance.

Definition 2 – [Pattern Silhouettes and Approximate Matching] Let $t = \{a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_n\}$ denote an arbitrary test instance in T. We call Pattern Silhouette any triple of form $s = (a_l, a_r, k)$, with $1 \leq l \leq r$, $k > 0$, where a_l and a_r are elements in t and k is the length of the sub-sequence ranging from a_l to a_r. A sequence X is said to approximately match instance $t \in T$, if X and t shares at least one pattern silhouette.

The AC-SC algorithm first enumerates all *eligible* pattern silhouettes for each training example $x \in D$. Then, it constructs an inverted index, which links each pattern silhouette to all examples in D containing it. This inverted index is used in further steps. The process of enumerating silhouettes follows the same sliding-window approach described for enumerating strictly-contiguous sequences.

Projecting the Training Set Using Pattern Silhouettes

When a test instance $t \in T$ is given for classification, the AC-SC algorithm filters training examples $x \in D$ approximately matching instance t, in order to learn a specialized classifier C_t. In this case, an approximate matching example must share at least one pattern silhouette with instance t. In order to find approximate matching examples in D, when a test instance t is given, the AC-SC algorithm enumerates all eligible pattern silhouettes present in t. This processes is illustrated in Figure 3.3. Therefore, the same sequence length limitation ($m = \sqrt{n}$) is imposed in this process, which ensures $O(n\sqrt{n})$ learning cost for a given test t containing n features.

For each enumerated silhouette in t, the algorithm filters the training set D in order to get only examples having that shape. This filtering process is nothing more than a simple lookup at the inverted index previously constructed. At this point, the

Figure 3.3. Pattern Silhouette Enumeration

A C T G G C T T	**Silhouette**
A C T G C G T T	$(A, A, 1)$
A C T G C G T T	$(C, C, 1)$
⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮	⋮
A C T G C G T T	$(T, T, 1)$
A C T G C G T T	$(A, C, 2)$
A C T G C G T T	$(C, T, 2)$
⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮	⋮
A C T G C G T T	$(T, T, 2)$
A T G C G T T	$(A, T, 3)$
A C G C G T T	$(C, G, 3)$
⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮	⋮
A C T G C G T	$(G, T, 3)$
⋮ ⋮	⋮
A T T	$(A, T, 7)$
A C T	$(C, T, 7)$
A T	$(A, T, 8)$

AC-SC algorithm works using the following information: a test instance t, its pattern silhouettes (and the position where they occur in t), and all approximately-matching training examples according to those silhouettes. The algorithm now advances to the next step.

Calculating Class Membership

Calculating class membership involves assessing how similar are a test instance t and its approximately-matching examples in D (relatively to a given silhouette). This process iterates calculating the similarity between each instance $t \in T$ and approximately-matching examples in D, as detailed next.

Silhouette Alignment

Given an arbitrary instance $t \in T$, the goal of silhouette alignment is to find, for each approximately-matching training example $x \in D$, the position in which the target silhouette s occurs. This is done by traversing the elements in example x, looking for the first element matching the leftmost element in silhouette s. If element $a_j \in x$ matches the leftmost element in s, then we simply check if element $a_{j+k} \in x$ also matches the rightmost element in s, where k is the length of s. If both matches succeed, we have an alignment between instances t and x, and, since we already know where the silhouette occurs in t, we can now measure how similar are these instances. Specifically, we measure the similarity between a pair of instances (t, x) by exploiting the intuition that there exists valuable information in the proximity among elements in t and x.

Acessing Similarity Between Instances

We propose a novel similarity metric, which we call Contiguity-Based Similarity Function, which expresses the similarity between a test instance t and approximately-matching examples $x \in D$, given a specific pattern silhouette. Such metric consists in pairwise comparing the elements within t and x, emphasizing consecutive matches. This metric is formulated as a function with the following properties: (i) it is monotonically increasing, (ii) it memorizes previous matches, (iii) consecutive matches make it increase fast, (iv) consecutive mismatches make it constant, and (v) isolated mismatches only delay its increase. This similarity function is better defined next.

Definition 3 − [Contiguity-Based Similarity Function] Consider two instances $t \in T$ and $x \in D$, both containing a pattern silhouette $s = (l, r, k)$. Let i denote the starting position where s occurs in t, and j the starting position where s occurs in x. The similarity between t and x, regarding to s, is given by Equation 3.5:

$$\Psi(t, x|s) = \sum_{c=0}^{k-1} \psi(t_{i+c}, x_{j+c}) \tag{3.5}$$

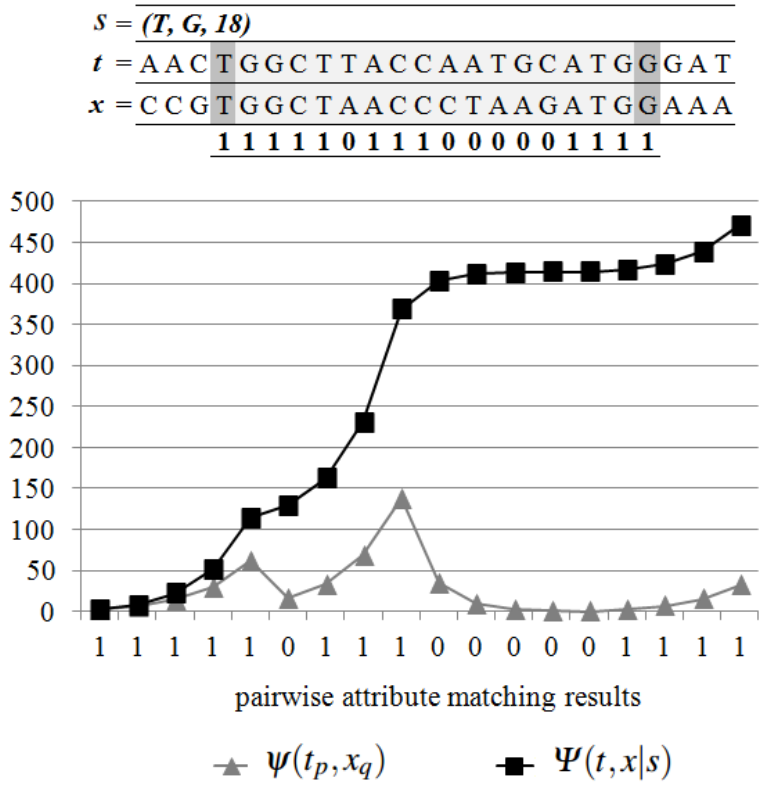
where ψ function is defined as:

$$\psi(t_p, x_q) = \begin{cases} 2 & \text{if } t_p = x_q \wedge p = i \\ 2 \times \lceil \psi(t_{p-1}, x_{q-1}) \rceil & \text{if } t_p = x_q \wedge p \neq i \\ \frac{1}{4} \times \psi(t_{p-1}, x_{q-1}) & \text{otherwise} \end{cases} \tag{3.6}$$

The ψ function is a pairwise comparing function, which is applied for each aligned pair of features from a test and a training instance. When a matching is well succeeded, it returns a value which is the double of the value returned by the previous matching. In other words, the longer the chain of previous successful matches, the bigger the value of the current ψ execution. On the other hand, when a mismatch happens, it returns just a quarter of the previous value. Expressly, consecutive mismatches make the ψ function tend fast to zero, but isolated mismatches do not cause such hard impact. The sum of each pairwise comparisons is accumulated by Ψ function, which will result in a metrics of similarity between the two instances being compared. As the ψ function is the core of the algorithm, being executed repeated times for each test instance, it must be as simple and fast as possible. We decided to build it using power-of-two (bitwise) operations, for computational efficience. The other steps of the algorithm are unaware of these operations, so the ψ function, which can be seen as a loss function, could be given as a parameter to the model, if necessary.

Figure 3.4 illustrates the behavior of ψ and Ψ functions when applied to a hypothetical pair of instances and a given silhouette of length 18. Zeroes represent mismatches. We can see the difference in impact between an isolated mismatch (at 6^{th} position) and consecutive mismatches (from 10^{th} to 14^{th} positions).

Figure 3.4. Behavior of ψ and Ψ functions.



Prediction: Multiple pattern silhouettes are investigated while processing an arbitrary instance $t \in T$. The similarity values between t and each silhouette are grouped together according to the class label associated with the corresponding training example. We calculate a score for each class label y_i as follows:

$$w(t, y_i) = \sum_{k \in S_t} \sum_{x \in D_s^{y_i}} \Psi(x, t|s) \tag{3.7}$$

where S_t is the set of all pattern silhouettes matching test instance t, and $D_s^{y_i}$ is the set of all training examples labeled as y_i containing a given silhouette s. Finally, these scores are normalized, as expressed by the prediction function $\hat{p}(y_i|t)$, previously shown in Equation 3.4.

Chapter 4

Experimental Evaluation

In this section we empirically analyze the effectiveness of our proposed algorithms for the sake of learning sequential classifiers. Similarly to other works [Zaki et al., 2010], we employ the standard precision as our basic evaluation measure. Our algorithms were implemented in Java language and executed over OpenJDK 7 platform. Learning times for all algorithms include the time spent during any pre-processing, training and testing, and are given in milliseconds, and all experiments were performed on a Linux-based PC with a Intel core i5 2.4 GHz processor and 4.0 GBytes RAM.

4.1 Application Scenarios

We employ diverse application scenarios in order to evaluate our algorithms under different aspects of sequential data, such as short- and long-range dependence, and different levels of noise. Application scenarios used in our experiments, include:

1. Sentiment Analysis: this task aims at determining the attitude that is implicit in a textual sentence, with respect to some topic or content. The attitude is usually represented by judgment or evaluation concerning the topic. We used two datasets in the experiments. Both of them comprise messages posted on Twitter. The first one is composed of messages concerning the Brazilian Defeat in the 2010 Soccer World Cup. We collected 3,214 messages in Portuguese referencing a particular player, Felipe Melo. We annotated the messages in order to track the sentiment of appreciation for the participation of Felipe Melo. The dataset contains 8,101 distinct terms, and each message was manually labeled by three to five human annotators. The second dataset contains messages concerning the Brazilian presidential election, occurred in 2010. We collected over

62,000 messages regarding candidate Dilma Rousseff during the campaign. These messages were manually annotated according to their approval or disapproval contents.

2. Name Disambiguation: given a citation record with ambiguous author names, determine the correct entity corresponding to that name. The dataset we used in the experiments is composed of authorship records extracted from the DBLP digital library. Each record in the dataset comprises co-author names, title and citations, and is associated with at least one ambiguous author name. There are 2,193 distinct terms in the dataset.

3. Protein Fold Recognition: this task aims at predicting the structure (or fold) of a protein from its amino acid sequence. The dataset we used in the experiments is composed of amino-acid sequences collected from the Protein Data Bank archive (`www.pdb.org`), which contains experimentally determined structures of proteins. The dataset contains 694 sequences, each one having up to 967 elements.

4. Protein Family Recognition: given a collection of amino-acid sequences belonging to different protein families, determine whether a query protein belongs to a given family or not. The dataset we used in the experiments is composed of long string of characters, where each character represents an amino-acid from a set of 20 possible ones. The dataset includes a curated classification of known protein structures with the secondary structure knowledge embedded in the dataset [Murzin et al., 1995]. This task has being largely employed in many applications of biological sequence analysis for finding homologous proteins [R. Durbin and Mitchison, 1998].

5. Intrusion Detection: given a sequence of UNIX commands performed by an arbitrary user, determine if the user is a masquerader or authentic one. The dataset we used in the experiments was collected from Purdue University [Lane and Brodley, 1999], over varying periods of time, using the (t)csh mechanism. Each command in the history data together with its arguments is treated as a single token.

6. Spelling Correction: given a sentence containing commonly confused words [Golding and Roth, 1996], determine if the target word is correctly or wrongly spelled. The total number of sentences in our dataset is 2,917 and there are 12,280 distinct terms.

7. Web Log Analysis: given a sequence of clicks performed by an arbitrary user, categorize the user based on his/her navigation behavior. The dataset we used in the experiments is composed of log files collected at the Department of Computer Science at the Rensselaer Polytechnic Institute during a period of 3 weeks. Those files where transformed into sequences of clicks (web navigation history) made by different users. Each sequence represents a web session of a specific user and is labeled according to the origin domain of that user. Users coming from "edu" or "ac" domains are taken as academic and users coming from other domains are taken as visitors. In all, the dataset contains 16,206 unique Web pages, which make up the alphabet.

More detailed descriptions of these tasks and datasets are available in [Zaki et al., 2010; Davis et al., 2012; Silva et al., 2011].

4.2 Baselines

We employ diverse baselines in our comparison analysis: (i) a k-order HMM [Galassi et al., 2007; Pitkow and Pirolli, 1999; Saul and Jordan, 1999; Deshpande and Karypis, 2004] as the representative of traditional solutions to model sequential data, (ii) the ASC algorithm [Bannister, 2007], as the representative of the state-of-the-art algorithms devised to tasks related to information extraction and protein fold recognition, (iii) the VOGUE algorithm [Zaki et al., 2010], as the representative of the state-of-the-art algorithms devised to tasks such as intrusion detection, spelling correction, and Web log analysis, (iv) the HMMER algorithm [Eddy, 1998], as a representative of the state-of-the-art algorithms devised to protein family recognition tasks, (v) the String Kernel for LIBSVM [Chang and Lin, 2011b], as the representative of popular mismatch-tolerant string kernel solutions available and (vi) the LAC algorithm, as the representative state-of-the-art demand-driven associative classifiers.

4.3 Results

The learning task in all application scenarios is the same, and it consists in correctly predicting the class label associated with test instances. We conducted ten-fold cross validation, and the results reported for each evaluated algorithm correspond to the average of the ten trials. Statistical significance tests were performed using a 2-sided paired t-test with $p-value < 0.05$. Best results, including statistical ties, are shown in

bold. SVM models were built with hyperparameters found with a grid search approach. The computation time of the grid search task was not considered in the results.

Tables 4.1 and 4.2 show accuracy numbers and the corresponding execution times for datasets concerning sentiment analysis. Table 4.3 shows accuracy and time results obtained in author name disambiguation task. Finally, Table 4.4 shows accuracy and time in protein fold recognition task. For these datasets, we used LAC, String Kernel and ASC algorithms as baselines. We performed an extensive evaluation by analyzing different parameter configurations, namely maximum sequence length, minimum support, and maximum gap size. In most cases, lower minimum support values yield best accuracy figures, but this usually increases learning time, as expected. In many cases, accuracy numbers tend to increase when larger gaps are allowed, but again, this also increases learning time.

Algorithm	*Max. Sequence*	*Min. Support*	*Max. Gap*	*Time*	*Accuracy*
LAC	1	-	-	**18,487**	0.781
	2	-	-	343,155	0.912
	3	-	-	2,531,995	0.930
	4	-	-	9,989,354	**0.935**
String Kernel	-	-	-	**10,598,900**	**0.854**
SC-SC	$\sqrt{n}$	-	-	**144,838**	**0.906**
AC-SC	$\sqrt{n}$	-	-	**185,785**	**0.921**

Table 4.1. Results in Sentiment Analysis Task - Dilma Rousseff Dataset

The length of the enumerated sequences is important for the sake of improving classification accuracy, but the cost of exploring longer sequences may become impracticable, in especial when combined with the gap strategy. In summary, higher accuracy figures are usually achieved by exploiting parameter configurations that lead to higher learning times. This is clearly observed, even in domains where short patterns (having 2 to 3 features) are expected to provide good results, as in sentiment analysis and author name disambiguation, which are related to language and text information retrieval. Our proposed algorithms, in most cases, offer either the fastest learning times or the highest accuracy numbers. More specifically, the SC-SC algorithm was the second fastest one in all evaluated cases, behind only of the LAC algorithm when limited to single-feature patterns, which is always less precise. Generally speaking, we can state that AC-SC algorithm offers the same (or a very similar) accuracy numbers when compared with LAC algorithm. However, such LAC results are obtained, in most cases, with higher pattern sizes, which make its execution much slower than all

Algorithm	*Max. Seq. Length*	*Min. Support*	*Max. Gap*	*Time*	*Accuracy*
ASC	1	0.00100	1	8,047	0.924
	1	0.00500	1	7,247	0.925
	1	0.01000	1	7,280	0.925
	2	0.00100	1	8,892	0.926
	2	0.00100	2	9,645	0.926
	2	0.00100	3	10,238	0.926
	2	0.00500	1	7,934	0.925
	2	0.00500	2	8,524	0.926
	2	0.00500	3	8,526	0.926
	2	0.01000	1	7,684	0.925
	2	0.01000	2	7,672	0.925
	2	0.01000	3	7,853	0.926
	3	0.00100	1	9,520	0.926
	3	0.00100	2	12,631	**0.928**
	3	0.00100	3	15,606	**0.929**
	3	0.00500	1	8,077	0.925
	3	0.00500	2	8,860	0.926
	3	0.00500	3	9,670	0.925
	3	0.01000	1	7,809	0.925
	3	0.01000	2	8,419	0.926
	3	0.01000	3	9,657	0.926
	4	0.00100	1	13,940	0.926
	4	0.00100	2	17,293	**0.929**
	4	0.00100	3	32,712	**0.929**
	4	0.00500	1	8,666	0.924
	4	0.00500	2	10,506	0.926
	4	0.00500	3	17,939	0.925
	4	0.01000	1	13,489	0.925
	4	0.01000	2	9,687	0.926
	4	0.01000	3	21,188	0.926
LAC	1	-	-	**750**	0.927
	2	-	-	2,637	0.927
	3	-	-	14,400	**0.930**
	4	-	-	57,815	0.928
String Kernel	-	-	-	**4,709**	**0.938**
SC-SC	$\sqrt{n}$	-	-	**1,794**	**0.927**
AC-SC	$\sqrt{n}$	-	-	4,881	**0.927**

Table 4.2. Results in Sentiment Analysis Task - Felipe Melo Dataset

Algorithm	*Max. Seq. Length*	*Min. Support*	*Max. Gap*	*Time*	*Accuracy*
ASC	1	0.00100	1	565	0.815
	1	0.00500	1	558	0.798
	1	0.01000	1	**484**	0.823
	2	0.00100	1	992	0.931
	2	0.00100	2	1,207	0.978
	2	0.00100	3	1,583	0.969
	2	0.00500	1	1,053	0.939
	2	0.00500	2	1,367	0.972
	2	0.00500	3	1,490	0.968
	2	0.01000	1	564	0.972
	2	0.01000	2	742	0.966
	2	0.01000	3	755	0.977
	3	0.00100	1	1,329	0.971
	3	0.00100	2	2,652	0.967
	3	0.00100	3	4,623	0.959
	3	0.00500	1	1,330	0.970
	3	0.00500	2	2,723	0.967
	3	0.00500	3	4,676	0.963
	3	0.01000	1	742	**0.986**
	3	0.01000	2	1,020	0.975
	3	0.01000	3	1,441	0.962
	4	0.00100	1	1,823	0.973
	4	0.00100	2	6,295	0.967
	4	0.00100	3	15,917	0.955
	4	0.00500	1	1,903	0.977
	4	0.00500	2	6,268	0.969
	4	0.00500	3	15,278	0.937
	4	0.01000	1	768	0.982
	4	0.01000	2	1,627	0.976
	4	0.01000	3	2,914	0.962
LAC	1	-	-	**47**	0.777
	2	-	-	62	0.923
	3	-	-	187	**0.955**
	4	-	-	702	0.942
String Kernel	-	-	-	**5,605**	**0.903**
SC-SC	$\sqrt{n}$	-	-	**92**	**0.953**
AC-SC	$\sqrt{n}$	-	-	**248**	**0.983**

Table 4.3. Results in Author Name Disambiguation Task

Algorithm	*Max. Seq. Length*	*Min. Support*	*Max. Gap*	*Time*	*Accuracy*
ASC	1	0.00100	1	28,745	0.046
	1	0.00500	1	21,294	0.043
	1	0.01000	1	19,489	0.046
	2	0.00000	1	69,864	0.328
	2	0.00000	2	364,959	0.334
	2	0.00100	1	54,896	0.301
	2	0.00100	2	64,629	0.308
	2	0.00100	3	74,983	0.311
	2	0.00500	1	33,515	0.299
	2	0.00500	2	38,892	0.298
	2	0.00500	3	47,003	0.319
	2	0.01000	1	22,894	0.049
	2	0.01000	2	23,345	0.051
	2	0.01000	3	23,501	0.072
	3	0.00000	1	250,840	0.335
	3	0.00000	2	696,949	**0.345**
	3	0.00100	1	72,328	0.301
	3	0.00100	2	143,993	0.309
	3	0.00100	3	269,732	0.292
	3	0.00500	1	33,104	0.297
	3	0.00500	2	38,731	0.325
	3	0.00500	3	44,226	0.329
	3	0.01000	1	22,947	0.050
	3	0.01000	2	23,444	0.055
	3	0.01000	3	24,668	0.074
	4	0.00100	1	100,952	0.318
	4	0.00100	2	388,748	0.294
	4	0.00100	3	1,146,505	0.261
	4	0.00500	1	33,263	0.313
	4	0.00500	2	40,466	0.334
	4	0.00500	3	44,458	0.325
	4	0.01000	1	23,362	0.049
	4	0.01000	2	23,254	0.050
	4	0.01000	3	23,624	0.071
LAC	1	-	-	**8,816**	0.046
	2	-	-	37,220	0.300
	3	-	-	3,511,546	**0.343**
String Kernel	-	-	-	**15,100**	**0.373**
SC-SC	$\sqrt{n}$	–	–	**8,758**	0.311
AC-SC	$\sqrt{n}$	–	–	78,409	**0.347**

Table 4.4. Results in Protein Fold Recognition Task

other approaches. A similar analysis can be done when comparing AC-SC with ASC algorithm: the similar accuracy results come with larger patterns and gaps, which produces slower executions. We can conclude that AC-SC algorithm compensates its smaller number of investigated patterns by exploring longer sequences and, specially, by executing a less rigid pattern matching, due to its ability to exploit approximate sequences. This conclusion becomes clearer when we compare those results with SC-SC results, which, in most of cases, can not achieve the same accuracy, despite exploring the same sequence lengths. The ASC algorithm, unfortunately, could not be applied in the first sentiment analysis dataset (presidential election). Its high-dimensionality, combined with the large alphabet size, makes ASC consume unpracticalble amounts of memory. When comparing AC-SC with the string kernel approach, our algorithm achieve better or similar accuracy results in sentiment analysis and author name disambiguation tasks, being musch faster in two datasets and slightly slower in one of them (Felipe Melo sentiment analysis). Protein fold recognition poses a very hard task for all algorithms. No one could reach even 40% in accuracy. String kernel technique had the highest accuracy of all and the best execution time, followed by AC-SC algorithm in two perspectives.

Table 4.5 shows accuracy numbers and the corresponding execution times for the dataset concerning protein family recognition. Next, table 4.6 presents accuracy and time results for intrusion detection task. The following table, 4.7, shows results related to the spelling correction application and, finally, table 4.8 contains accuracy and time numbers for Web log analysis. For these datasets, we used HMM, HMMER, and VOGUE algorithms as baselines. Finding the best parameters for these algorithms involved the evaluation of an enormous number of possible configurations, as discussed in [Zaki et al., 2010].

Table 4.5. Results in Protein Family Recognition Task

Algorithm	*Max. Sequence*	*Time*	*Accuracy*
HMM	-	610	0.725
HMMER	-	190	0.750
VOGUE	-	3,650	0.775
Str. Krn	-	224	**0.844**
SC-SC	$\sqrt{n}$	**153**	0.814
AC-SC	$\sqrt{n}$	445	0.840

Again, in almost all cases, our proposed algorithms, SC-SC and AC-SC, offer either the fastest learning times or the highest accuracy numbers. More specifically, the SC-SC algorithm was the fastest one in all evaluated cases. Further, for the

Table 4.6. Results in Intrusion Detection Task

Algorithm	*Max. Sequence*	*Time*	*Accuracy*
HMM	-	64,930	0.715
VOGUE	-	14,420	0.732
Str.Krn.	-	161,660	0.487
SC-SC	$\sqrt{n}$	**3,870**	0.767
AC-SC	$\sqrt{n}$	73,040	**0.791**

Table 4.7. Results in Spelling Correction Task

Algorithm	*Max. Sequence*	*Time*	*Accuracy*
HMM	-	26,450	0.624
VOGUE	-	44,630	0.685
Str. Krn.	-	1,602	0.581
SC-SC	$\sqrt{n}$	**421**	0.654
AC-SC	$\sqrt{n}$	2,079	**0.715**

Table 4.8. Results in Web Log Analysis Task

Algorithm	*Max. Sequence*	*Time*	*Accuracy*
HMM	-	1,200,720	0.798
VOGUE	-	77,590	0.810
Str. Krn.	-	269,860	0.835
SC-SC	$\sqrt{n}$	**2,250**	**0.842**
AC-SC	$\sqrt{n}$	3,510	**0.841**

application concerning Web Log analysis, SC-SC was also the best performer in terms of classification accuracy, and for the remaining applications, it also provides highly competitive accuracy figures, being, in most of the cases, higher than the accuracy numbers achieved by HMMER, HMM, and VOGUE. AC-SC achieves the highest accuracy in all cases, except in protein family recognition task, were string kernels had the higher result, despite similar. AC-SC also provides fast execution times in all cases, when compared with HMMs and VOGUE approaches. When compared with string kernels, we can see that SVMs approaches execution time floats. In Web log analysis and intrusion detection tasks, having larger datasets, its computation time is orders of magnitude higher than SC-SC and AC-SC algorithms. In the other two smaller datasets the time is in the same order of magnitude, being faster in only one case (spelling correction task). Yet in intrusion detection, AC-SC is slower than HMM approach, but its accuracy showed more than 10% of gain.

Table 4.9 compares the cases of classification success and failure for SC-SC and AC-SC algorithms. As expected, in almost all cases, the AC-SC algorithm enhances the accuracy in relation to SC-SC, having success in regions were SC-SC fails. Exceptions

happen only in datasets where both algorithms have the same or very near accuracy numbers.

Dataset	*Both Right*	*Both Wrong*	*AC-SC Right*	*SC-SC Right*
Dilma Roussef	91.8%	5.9%	1.6%	0.7%
Felipe Melo	91.1%	4.6%	2.1%	2.2%
Author Name	91.3%	0.4%	8.3%	0.0%
Protein Fold	33.3%	61.6%	3.0%	2.0%
Protein Family	78.9%	15.8%	5.3%	0.0%
Intrusion	77.8%	15.6%	4.5%	2.1%
Spelling	62.6%	20.1%	12.3%	5.0%
Web Log	84.3%	14.9%	0.4%	0.4%

Table 4.9. Accuracy analysis: SC-SC vs. AC-SC

4.3.1 Reproducibility

All experiments shown in this section can be directly reproduced. All algorithms used in our evaluations and all datasets employed are available to the scientific community. The official LAC algorithm version, is available in the Weka[1] central package repository or can be download from DCC's Machine Learning software repository[2]. All other algorithms, yet in beta version, and all employed datasets, can be found at Gessé Dafé's software page[3].

[1]http://www.cs.waikato.ac.nz/ml/weka/
[2]http://code.google.com/p/machine-learning-dcc-ufmg/
[3]http://code.google.com/p/gesse-dafe-publications/

Chapter 5

Conclusions and Future Work

This thesis focuses on the important problem of learning classifiers from sequential data. Such classifiers are used to distinguish between sequences belonging to different labeled groups. We have introduced new algorithms for learning these classifiers. Our proposed algorithms produce classifiers composed of long and discriminative sequences, being able to capture long-range dependencies in the data. The first, simpler algorithm, exploits proximity information to drastically narrow down the search space for sequences. More specifically, elements within a sequence must be contiguous in the sense that no violation in the ordering among elements is allowed. Sequence enumeration follows a simple sliding window approach, which warrants that the cardinality of the classifier increases by $O(n\sqrt{n})$ with the length of the test instances.

The second, more sophisticate algorithm, also exploits feature proximity in order to narrow down the search space for rules, but in this case violations or disruptions in the ordering between features within a rule are allowed to happen. We introduce a novel structure, called pattern silhouettes, in order to warrant effective rule enumeration with partial contiguous matchings.

To evaluate the effectiveness of our algorithms, we use real data obtained from applications scenarios such as sentiment analysis, author name disambiguation, intrusion detection, spelling correction, protein fold and family recognition and Web log analysis. Our results reveal that the proposed algorithms are highly effective and efficient, being able to produce accurate results with low computational cost, when compared against state-of-the-art associative classifiers, general purpose sequential classifiers and even domain-specific sequential classifiers. In some cases our algorithms are orders of magnitude faster to achieve the same or even better accuracy.

Future work include investigating other application scenarios where it is

important to learn sequential classifiers and investigating other approaches of similarity metrics for patterns partial matching.

Bibliography

Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *SIGMOD Conference*, pages 207–216.

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *ICDE*, pages 3–14.

Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487--499.

Antunes, C. and Oliveira, A. (2003). Generalization of pattern-growth methods for sequential pattern mining with gap constraints. In *MLDM*, pages 239–251.

Bannister, W. (2007). *Associative and sequential classification with adaptive constrained regression methods*. PhD thesis, Tempe, AZ, USA.

Bicego, M., Murino, V., and Figueiredo, M. (2003a). A sequential pruning strategy for the selection of the number of states in hidden markov models. *Pattern Recognition Letters*, 24(9-10):1395–1407.

Bicego, M., Murino, V., and Figueiredo, M. (2003b). Similarity-based clustering of sequences using hidden markov models. In *MLDM*, pages 86–95.

Bicego, M., Murino, V., and Figueiredo, M. (2004). Similarity-based classification of sequences using hidden markov models. *Pattern Recognition*, 37(12):2281–2291.

Bicego, M., Murino, V., Pelillo, M., and Torsello, A. (2006). Similarity-based pattern recognition. *Pattern Recognition*, 39(10):1813–1814.

Breiman, L., Friedman, J., Stone, C., and Olshen, R. (1984). *Classification and regression trees*. Chapman & Hall/CRC.

Bühlmann, P. and Wyner, A. (1999). Variable length markov chains. *The Annals of Statistics*, 27(2):480--513.

Chang, C. and Lin, C. (2011a). Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27.

Chang, C.-C. and Lin, C.-J. (2011b). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1--27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273--297.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21--27.

Damashek, M. et al. (1995). Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843--848.

Dave, K., Lawrence, S., and Pennock, D. M. (2003). Mining the peanut gallery: opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 519--528, New York, NY, USA. ACM.

Davis, A., Veloso, A., da Silva, A., Laender, A., and Meira Jr., W. (2012). Named entity disambiguation in streaming data. In *ACL*, pages 815–824.

Deshpande, M. and Karypis, G. (2004). Selective markov models for predicting web page accesses. *ACM Trans. Internet Techn.*, 4(2):163–184.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2):103--130.

Du Preez, J. (1998). Efficient training of high-order hidden markov models using first-order representations. *Computer speech & language*, 12(1):23--39.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

Eddy, S. (1998). Profile hidden markov models. *Bioinformatics*, 14(9):755–763.

Ezeife, C. and Lu, Y. (2005). Mining web log sequential patterns with position coded pre-order linked wap-tree. *Data Mining and Knowledge Discovery*, 10(1):5--38.

Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian network classifiers. *Machine learning*, 29(2):131--163.

Galassi, U., Giordana, A., and Saitta, L. (2007). Incremental construction of structured hidden markov models. In *IJCAI*, pages 798–803.

Golding, A. and Roth, D. (1996). Applying winnow to context-sensitive spelling correction. *CoRR*.

Guralnik, V. and Haigh, K. (2002). Learning models of human behaviour with sequential patterns. In *Proceedings of the AAAI-02 workshop Automation as Caregiver*, pages 24--30.

Halácsy, P., Kornai, A., and Oravecz, C. (2007). Hunpos: an open source trigram tagger. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 209--212. Association for Computational Linguistics.

Han, H., Giles, C., Zha, H., Li, C., and Tsioutsiouliklis, K. (2004). Two supervised learning approaches for name disambiguation in author citations. In *JCDL*, pages 296--305.

Han, H., Zha, H., and Giles, C. (2005). Name disambiguation in author citations using a k-way spectral clustering method. In *JCDL*, pages 334--343.

Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1--12. ACM.

Haussler, D. (1999). Convolution kernels on discrete structures. Technical report, Technical report, UC Santa Cruz.

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554--2558.

Hu, J., Brown, M., and Turin, W. (1996). Hmm based on-line handwriting recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(10):1039–1045.

Hughey, R. and Krogh, A. (1996). Hidden markov models for sequence analysis: extension and analysis of the basic method. *Computer Applications in the Biosciences*, 12(2):95–107.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, pages 137--142.

Kim, S., Park, S., Won, J., and Kim, S. (2008). Privacy preserving data mining of sequential patterns for network traffic data. *Information Sciences*, 178(3):694--713.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59--69.

Kriouile, A., Mari, J., and Haon, J. (1990). Some improvements in speech recognition algorithms based on hmm. In *ICASSP*, pages 545--548.

Kuksa, P., Huang, P.-H., and Pavlovic, V. (2008). A fast, large-scale learning method for protein sequence classification. In *8th Int. Workshop on Data Mining in Bioinformatics*, pages 29--37.

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

Law, H. and Chan, C. (1996). N-th order ergodic multigram hmm for modeling of languages without marked word boundaries. In *COLING*, pages 204--209.

Lesh, N., Zaki, M., and Ogihara, M. (1999). Mining features for sequence classification. In *KDD*, pages 342–346.

Leslie, C., Eskin, E., and Noble, W. S. (2002). The spectrum kernel: A string kernel for svm protein classification. In *Proceedings of the pacific symposium on biocomputing*, volume 7, pages 566--575. Hawaii, USA.

Leslie, C. S., Eskin, E., Cohen, A., Weston, J., and Noble, W. S. (2004). Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467--476.

Li, W., Han, J., and Pei, J. (2001a). Cmar: Accurate and efficient classification based on multiple class-association rules. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 369--376. IEEE.

Li, W., Han, J., and Pei, J. (2001b). CMAR: Accurate and efficient classification based on multiple class-association rules. In *ICDM*, pages 369–376.

Lin, M., Hsueh, S., Chen, M., and Hsu, H. (2009). Mining sequential patterns for image classification in ubiquitous multimedia systems. In *IIH-MSP*, pages 303–306.

Lodhi, H., Muggleton, S., and Sternberg, M. (2009). Multi-class protein fold recognition using large margin logic based divide and conquer learning. *SIGKDD Explorations*, 11(2):117–122.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *JMLR*, 2:419--444.

Malik, H. and Kender, J. (2008). Classifying high-dimensional text and web data using very short patterns. In *ICDM*, pages 923–928.

Müller, S., Eickeler, S., and Rigoll, G. (2000). Crane gesture recognition using pseudo 3-d hidden markov models. In *FG (Conf. on Automatic Face and Gesture Recognition)*, pages 398–402.

Murzin, A., Brenner, S., Hubbard, T., and Chothia, C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, 247(4):536–540.

Pak, A. and Paroubek, P. (2010). Twitter as a corpus for sentiment analysis and opinion mining. In *Proceedings of LREC*, volume 2010.

Pedersen, T. (2001). A decision tree of bigrams is an accurate predictor of word sense. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1--8. Association for Computational Linguistics.

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

Pitkow, J. and Pirolli, P. (1999). Mining longest repeating subsequences to predict world wide web surfing. In *USENIX Symposium on Internet Technologies and Systems*.

Quinlan, J. (1979). Discovering rules form large collections of examples: a case study.

Quinlan, J. (1993). *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann.

R. Durbin, S. Eddy, A. K. and Mitchison, G. (1998). *Biological Sequence Analysis*. Cambridge University Press.

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, 77(2):257–286.

Riseman, E. and Hanson, A. (1974). A contextual postprocessing system for error correction using binary n-grams. *Computers, IEEE Transactions on*, 100(5):480--493.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Saul, L. and Jordan, M. (1999). Mixed memory markov models: Decomposing complex stochastic processes as mixtures of simpler ones. *Machine Learning*, 37(1):75–87.

Schwardt, L. and Preez, J. D. (2000). Efficient mixed-order hidden markov model inference. In *ICSLP*, pages 238--241.

Sha, F. and Saul, L. (2006). Large margin hidden markov models for automatic speech recognition. In *NIPS*, pages 1249–1256.

Shie, B., Hsiao, H., Tseng, V., and Yu, P. (2011). Mining high utility mobile sequential patterns in mobile commerce environments. In *Database Systems for Advanced Applications*, pages 224--238. Springer.

Silva, I., Gomide, J., Veloso, A., Meira Jr., W., and Ferreira, R. (2011). Effective sentiment stream analysis with self-augmenting training and demand-driven projection. In *SIGIR*, pages 475–484.

Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *EDBT*, pages 3–17.

Srivatsan, L., Sastry, P., and Unnikrishnan, K. (2005). Discovering frequent episodes and learning hidden markov models: A formal connection. *IEEE Transactions on Knowledge and Data Engineering*, 17:1505–1517.

Syed, Z., Indyk, P., and Guttag, J. (2009). Learning approximate sequential patterns for classification. *Journal of Machine Learning Research*, 10:1913–1936.

Szymanski, B. (2004). Recursive data mining for masquerade detection and author identification. *Workshop on Information Assurance*, pages 424–431.

Tseng, V. and Lee, C. (2005). Cbs: A new classification method by using sequential patterns. In *SDM*.

Veloso, A., Meira, W., and Zaki, M. (2006). Lazy associative classification. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 645--654. IEEE.

Wang, Y., Zhou, L., Feng, J., Wang, J., and Liu, Z. (2006). Mining complex time-series data by learning markovian models. In *ICDM*, pages 1136--1140.

Watkins, C. (1999). Dynamic alignment kernels. *Advances in Neural Information Processing Systems*, pages 39--50.

Weinberger, K. and Saul, L. (2009). Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research*, 10:207--244.

Zaki, M. (2000). Sequence mining in categorical domains: Incorporating constraints. In *CIKM*, pages 422–429.

Zaki, M. (2001). Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60.

Zaki, M., Carothers, C., and Szymanski, B. (2010). Vogue: A variable order hidden markov model with duration based on frequent sequence mining. *TKDD*, 4(1).