Journal of Machine Learning Research X (X) X Submitted 5/07; Published X/X

Linear-Time Computation of Similarity Measures for Sequential Data

Konrad Rieck KONRAD.RIECK@FIRST.FRAUNHOFER.DE
Pavel Laskov PAVEL.LASKOV@FIRST.FRAUNHOFER.DE
Fraunhofer FIRST.IDA
Kekuléstraße 7
12489 Berlin, Germany

Editor: Nello Cristianini

Abstract

Efficient and expressive comparison of sequences is an essential procedure for learning with sequential data. In this article we propose a generic framework for computation of similarity measures for sequences, covering various kernel, distance and non-metric similarity functions. The basis for comparison is embedding of sequences using a formal language, such as a set of natural words, k-grams or all contiguous subsequences. As realizations of the framework we provide linear-time algorithms of different complexity and capabilities using sorted arrays, tries and suffix trees as underlying data structures.

Experiments on data sets from bioinformatics, text processing and computer security illustrate the efficiency of the proposed algorithms – enabling peak performances of up to 10^6 pairwise comparisons per second. The utility of distances and non-metric similarity measures for sequences as alternatives to string kernels is demonstrated in applications of text categorization, network intrusion detection and transcription site recognition in DNA.

Keywords: String Kernels, String Distances, Learning with Sequential Data

1. Introduction

Sequences of discrete symbols are one of the fundamental data representations in computer science. A great deal of applications – from search engines to document ranking, from gene finding to prediction of protein functions, from network surveillance tools to anti-virus programs – critically depend on analysis of sequential data. Providing an interface to sequential data is therefore an essential prerequisite for applications of machine learning in these domains.

Machine learning algorithms have been traditionally designed for vectorial data – probably due to the availability of well-defined calculus and mathematical analysis tools. A large body of such learning algorithms, however, can be formulated in terms of pairwise relationships between objects, which imposes a much looser constraint on the type of data that can be handled. Thus, a powerful abstraction between algorithms and data representations can be established.

The most prominent example of such abstraction is *kernel-based learning* (e.g. Müller et al., 2001; Schölkopf and Smola, 2002) in which pairwise relationships between objects are expressed by a Mercer kernel, an inner product in a reproducing kernel Hilbert space. Following the seminal work of Boser et al. (1992), various learning methods have been re-formulated in terms of kernels, such as PCA (Schölkopf et al., 1998b), ridge regression (Cherkassky et al., 1999), ICA (Harmeling

et al., 2003) and many others. Although the initial motivation for the "kernel trick" was to allow efficient computation of an inner product in high-dimensional feature spaces, the importance of an abstraction from data representation has been quickly realized (e.g. Vapnik, 1995). Consequently kernel-based methods have been proposed for non-vectorial domains, such as analysis of images (e.g. Schölkopf et al., 1998a; Chapelle et al., 1999), sequences (e.g. Jaakkola et al., 2000; Watkins, 2000; Zien et al., 2000) and structured data (e.g. Collins and Duffy, 2002; Gärtner et al., 2004).

Although kernel-based learning has gained significant attention in recent years, a Mercer kernel is only one of many possibilities for defining pairwise relationships between objects. Numerous applications exist for which relationships are defined as metric or non-metric distances (e.g. Anderberg, 1973; Jacobs et al., 2000; von Luxburg and Bousquet, 2004), similarity or dissimilarity measures (e.g. Graepel et al., 1999; Roth et al., 2003; Laub and Müller, 2004; Laub et al., 2006) or non-positive kernel functions (e.g. Ong et al., 2004; Haasdonk, 2005). It is therefore imperative to address pairwise comparison of objects in a most general setup.

The aim of this contribution is to develop a *general framework* for pairwise comparison of sequences. Its generality is manifested by the ability to handle a large number of kernel functions, distances and non-metric similarity measures. From considerations of efficiency, we focus on algorithms with linear-time asymptotic complexity in the sequence lengths – at the expense of narrowing the scope of similarity measures that can be handled. For example, we do not consider super-linear comparison algorithms such as the Levenshtein distance (Levenshtein, 1966) and the all-subsequences kernel (Lodhi et al., 2002).

The basis of our framework is embedding of sequences in a high-dimensional feature space using a *formal language*, a classical tool of computer science for modeling semantics of sequences. Some examples of such languages have been previously used for string kernels, such as the bag-of-words, k-gram or contiguous-subsequence kernel. Our formalization allows one to use a much larger set of possible languages in a unified fashion, for example subsequences defined by a finite set of delimiters or position-dependent languages. A further advantage of embedding using formal languages is separation of embedding models from algorithms, which allows one to investigate different data structures to obtain optimal efficiency in practice.

Several data structures have been previously considered for specific similarity measures, such as hash tables (Damashek, 1995), sorted arrays (Sonnenburg et al., 2007), tries (Leslie et al., 2002; Shawe-Taylor and Cristianini, 2004; Rieck et al., 2006), suffix trees using matching statistics (Vishwanathan and Smola, 2004), suffix trees using recursive matching (Rieck et al., 2007) and suffix arrays (Teo and Vishwanathan, 2006). All of these data structures allow one to develop linear-time algorithms for computation of certain similarity measures. Most of them are also suitable for the general framework developed in this paper; however, certain trade-offs exist between their asymptotic run-time complexity, implementation difficulty and restrictions on embedding languages they can handle. To provide an insight into these issues, we propose and analyze three data structures suitable for our framework: *sorted arrays*, *tries* and *suffix trees* with an extension to suffix arrays. The message of our analysis, supported by experimental evaluation, is that the choice of an optimal data structure depends on the embedding language to be used.

This article is organized as followed: In Section 2 we review related work on sequence comparison. In Section 3 we introduce a general framework for computation of similarity measures for sequences. Algorithms and data structures for linear-time computation are presented in Section 4. We evaluate the run-time performance and demonstrate the utility of distinct similarity measures in Section 5. The article is concluded in Section 6

2. Related Work

Assessing the similarity of two sequences is a classical problem of computer science. First approaches, the string distances of Hamming (1950) and Levenshtein (1966), originated in the domain of telecommunication for detection of erroneous data transmissions. The degree of dissimilarity between two sequences is determined by computing the shortest trace of operations – insertions, deletions and substitutions – that transform one sequence into the other (Sankoff and Kruskal, 1983). Applications in bioinformatics motivated extensions and adaptions of this concept, for example defining sequence similarity in terms of local and global alignments (Needleman and Wunsch, 1970; Smith and Waterman, 1981). However, similarity measures based on the Hamming distance are restricted to sequences of equal length and measures derived from the Levenshtein distance (e.g. Liao and Noble, 2003; Vert et al., 2004), come at the price of computational complexity: No linear-time algorithm for determining the shortest trace of operations is currently known. One of the fastest exact algorithms runs in $O(n^2/\log n)$ for sequences of length n (Masek and Patterson, 1980).

A different approach to sequence comparison originated in the field of information retrieval with the vector space or bag-of-words model (Salton et al., 1975; Salton, 1979). Textual documents are embedded into a vector space spanned by weighted frequencies of contained words. The similarity of two documents is assessed by an inner-product between the corresponding vectors. This concept was extended to k-grams – k consecutive characters or words – in the domain of natural language processing and computer linguistic (e.g. Suen, 1979; Cavnar and Trenkle, 1994; Damashek, 1995). The idea of determining similarity of sequences by an inner-product was revived in kernel-based learning in the form of bag-of-words kernels (e.g. Joachims, 1998; Drucker et al., 1999; Joachims, 2002) and various string kernels (e.g. Zien et al., 2000; Leslie et al., 2002; Vishwanathan and Smola, 2004). Moreover, research in bioinformatics and text processing advanced the capabilities of string kernels, e.g. by considering gaps, mismatches and positions in sequences (e.g. Lodhi et al., 2002; Leslie et al., 2003; Leslie and Kuang, 2004; Rousu and Shawe-Taylor, 2005; Rätsch et al., 2007). The comparison framework proposed in this article shares the concept of embedding sequences with all of the above kernels, in fact most of the linear-time string kernels (e.g. Joachims, 1998; Leslie et al., 2002; Vishwanathan and Smola, 2004) are enclosed in the framework.

A further alternative for comparison of sequences are kernels derived from generative probability models, such as the Fisher kernel (Jaakkola et al., 2000) and the TOP kernel (Tsuda et al., 2002). Provided a generative model, for example a HMM trained over a corpus of sequences or modeled from prior knowledge, these kernel functions essentially correspond to inner-products of partial derivatives over model parameters. The approach enables the design of highly specific similarity measures which exploit the rich structure of generative models, e.g. for prediction of DNA splice sites (Rätsch and Sonnenburg, 2004). The run-time complexity of the kernel computation, however, is determined by the number of model parameters, so that only simple models yield run-time linear in the sequence lengths. Moreover, obtaining a suitable parameter estimate for a probabilistic model can be difficult or even infeasible in practical applications.

3. Similarity Measures for Sequential Data

Before introducing the framework for computation of similarity measures, we need to establish some basic notation. A *sequence* $\mathbf{x}$ is a concatenation of symbols from an *alphabet* $\mathcal{A}$. The set of all possible concatenations of symbols from $\mathcal{A}$ is denoted by $\mathcal{A}^*$ and the set of all concatenations

of length k by $\mathcal{A}^k$. A *formal language* $L \subseteq \mathcal{A}^*$ is any set of finite-length sequences drawn from $\mathcal{A}$ (Hopcroft and Motwani, 2001). The length of a sequence $\mathbf{x}$ is denoted by $|\mathbf{x}|$ and the size of the alphabet by $|\mathcal{A}|$. A contiguous subsequence w of $\mathbf{x}$ is denoted by $w \sqsubseteq \mathbf{x}$, a prefix of $\mathbf{x}$ by $w \sqsubseteq_p \mathbf{x}$ and a suffix by $w \sqsubseteq_s \mathbf{x}$. Alternatively, a subsequence w of $\mathbf{x}$ ranging from position i to j is referred to as $\mathbf{x}[i..j]$.

3.1 Embedding Sequences using a Formal Language

The basis for embedding of a sequence $\mathbf{x}$ is a formal language L, whose elements are sequences spanning an $|L|$-dimensional feature space. We refer to L as the *embedding language* and to a sequence $w \in L$ as a *word* of L. There exist numerous ways to define L reflecting particular aspects of application domains, yet we focus on three definitions that have been widely used in previous research:

1. **Bag-of-words**. In this model, L corresponds to a set of words from a natural language. L can be either defined explicitly by providing a dictionary or implicitly by partitioning sequences according to a set of delimiter symbols $D \subset \mathcal{A}$ (e.g. Salton, 1979; Joachims, 2002).

$$L = \text{Dictionary (explicit)}, \quad L = (\mathcal{A} \setminus D)^* \text{ (implicit)}$$

2. **K-grams** and **blended k-grams**. For the case of k-grams (in bioinformatics often referred to as k-mers), L is the set of all sequences of length k (e.g. Damashek, 1995; Leslie et al., 2002). The model of k-grams can further be "blended" by considering all sequences from length j up to k (e.g. Shawe-Taylor and Cristianini, 2004).

$$L = \mathcal{A}^k \text{ (k-grams)}, \quad L = \bigcup_{i=j}^{k} \mathcal{A}^i \text{ (blended k-grams)}$$

3. **Contiguous sequences**. In the most general case, L corresponds to the set of all contiguous sequences or alternatively to blended k-grams with infinite k (e.g. Vishwanathan and Smola, 2004; Rieck et al., 2007).

$$L = \mathcal{A}^* \quad \text{or} \quad L = \bigcup_{i=1}^{\infty} \mathcal{A}^i$$

Note that the alphabet $\mathcal{A}$ in the embedding languages may also be composed of higher semantic constructs, such as natural words or syntactic tokens. In these cases a k-gram corresponds to k consecutive words or tokens, and a bag-of-words models could represent textual clauses or phrases.

Given an embedding language L, a sequence $\mathbf{x}$ can be mapped into the $|L|$-dimensional feature space by calculating a function $\phi_w(\mathbf{x})$ for every $w \in L$ appearing in $\mathbf{x}$. The embedding function Φ for a sequence $\mathbf{x}$ is given by

$$\Phi : \mathbf{x} \mapsto (\phi_w(\mathbf{x}))_{w \in L} \text{ with } \phi_w(\mathbf{x}) := \mathrm{occ}(w, \mathbf{x}) \cdot \mathcal{W}_w \tag{1}$$

where $\mathrm{occ}(w, \mathbf{x})$ is the number of occurrences of w in the sequence $\mathbf{x}$ and $\mathcal{W}_w$ a weighting assigned to individual words. Alternatively $\mathrm{occ}(w, \mathbf{x})$ may be defined as frequency, probability or binary flag for the occurrences of w in $\mathbf{x}$.

While the choice and design of an embedding language L offer a large degree of flexibility, it is often necessary to refine the amount of contribution for each word $w \in L$, for example it is a common practice in text processing to ignore stop words and terms that do not carry semantic content. In the embedding function (1) such refinement is realized by the weighting term $\mathcal{W}_w$. The following three weighting schemes for defining $\mathcal{W}_w$ have been proposed in previous research:

1. **Corpus dependent weighting.** The weight $\mathcal{W}_w$ is based on the occurrences of w in the corpus of sequences (see Salton et al., 1975). Most notable is the inverse document frequency (IDF) weighting, in which $\mathcal{W}_w$ is defined over the number of documents N and the frequency $d(w)$ of w in the corpus.

   $$\mathcal{W}_w = \log_2 N - \log_2 d(w) + 1$$

   If $\mathrm{occ}(w, \mathbf{x})$ is the frequency of w in $\mathbf{x}$, the embedding function (1) corresponds to the well-known term frequency and inverse document frequency (TF-IDF) weighting scheme.

2. **Length dependent weighting.** The weight $\mathcal{W}_w$ is based on the length $|w|$ (see Shawe-Taylor and Cristianini, 2004; Vishwanathan and Smola, 2004), e.g. so that longer words contribute more than shorter words to a similarity measure. A common approach is defining $\mathcal{W}_w$ using a decay factor $0 \leq \lambda \leq 1$.

   $$\mathcal{W}_w = \lambda^{-|w|}$$

3. **Position dependent weighting.** The weight $\mathcal{W}_w$ is based on the position of w in $\mathbf{x}$. Vishwanathan and Smola (2004) propose a direct weighting scheme, in which $\mathcal{W}_w$ is defined over positional weights $\mathcal{W}(k, \mathbf{x})$ for each position k in $\mathbf{x}$ as

   $$\mathcal{W}_w = \mathcal{W}_{\mathbf{x}[i..j]} = \sum_{k=i}^{j} \mathcal{W}(k, \mathbf{x}).$$

   An indirect approach to position dependent weighting can be implemented by extending the alphabet $\mathcal{A}$ with positional information to $\tilde{\mathcal{A}} = \mathcal{A} \times \mathbb{N}$, so that every element $(a, k) \in \tilde{A}$ of the extended alphabet is a pair of a symbol a and a position k.

The introduced weighting schemes can be coupled to further refine the embedding based on L, e.g. in text processing the impact of a particular term might be influenced by the term frequency, inverse document frequency and its length.

3.2 Vectorial Similarity Measures for Sequences

With an embedding language L at hand, we can now express common vectorial similarity measures in the domain of sequences. Table 1 and 2 list well-known kernel and distance functions (see Vapnik, 1995; Schölkopf and Smola, 2002; Webb, 2002) in terms of L. The histogram intersection kernel in Table 1 derives from computer vision (see Swain and Ballard, 1991; Odone et al., 2005) and the Jensen-Shannon divergence in Table 2 is defined using $H(x, y) = x \log \frac{2x}{x+y} + y \log \frac{2y}{x+y}$.

Kernel	$k(\mathbf{x}, \mathbf{y})$
Linear	$\sum_{w \in L} \phi_w(\mathbf{x}) \phi_w(\mathbf{y})$
Polynomial	$\left(\sum_{w \in L} \phi_w(\mathbf{x}) \phi_w(\mathbf{y}) + \theta\right)^p$
Sigmoidal	$\tanh\left(\sum_{w \in L} \phi_w(\mathbf{x}) \phi_w(\mathbf{y}) + \theta\right)$
Gaussian	$\exp\left(\frac{-d(\mathbf{x},\mathbf{y})^2}{2\sigma^2}\right)$
Histogram intersection	$\sum_{w \in L} \min(\phi_w(\mathbf{x}), \phi_w(\mathbf{y}))$

Table 1: Kernel functions for sequential data.

Distance	$d(\mathbf{x}, \mathbf{y})$	**Distance**	$d(\mathbf{x}, \mathbf{y})$
Manhattan	$\sum_{w \in L} \lvert\phi_w(\mathbf{x}) - \phi_w(\mathbf{y})\rvert$	Chebyshev	$\max_{w \in L} \lvert\phi_w(\mathbf{x}) - \phi_w(\mathbf{y})\rvert$
χ^2 distance	$\sum_{w \in L} \frac{(\phi_w(\mathbf{x}) - \phi_w(\mathbf{y}))^2}{\phi_w(\mathbf{x}) + \phi_w(\mathbf{y})}$	Geodesic	$\arccos \sum_{w \in L} \phi_w(\mathbf{x}) \phi_w(\mathbf{y})$
Canberra	$\sum_{w \in L} \frac{\lvert\phi_w(\mathbf{x}) - \phi_w(\mathbf{y})\rvert}{\phi_w(\mathbf{x}) + \phi_w(\mathbf{y})}$	Hellinger^2	$\sum_{w \in_L} (\sqrt{\phi_w(\mathbf{x})} - \sqrt{\phi_w(\mathbf{y})})^2$
Minkowski^p	$\sum_{w \in L} \lvert\phi_w(\mathbf{x}) - \phi_w(\mathbf{y})\rvert^p$	Jensen-Shannon	$\sum_{w \in L} H(\phi_w(\mathbf{x}), \phi_w(\mathbf{y}))$

Table 2: Distance functions for sequential data.

A further and rather exotic class of vectorial similarity measures are *similarity coefficients* (see Sokal and Sneath, 1963; Anderberg, 1973). These coefficients have been designed for comparison of binary vectors and often express non-metric properties. They are constructed using three summation variables a, b and c, which reflect the number of matching components $(1/1)$, left mismatching components $(0/1)$ and right mismatching components $(1/0)$ in two binary vectors. Common similarity coefficients are given in Table 3.

Sim. Coeff.	$s(\mathbf{x}, \mathbf{y})$	**Sim. Coeff.**	$s(\mathbf{x}, \mathbf{y})$
Simpson	$a / \min(a + b, a + c)$	Kulczynski (1)	$a/(b + c)$
Jaccard	$a/(a + b + c)$	Kulczynski (2)	$\frac{1}{2}(a/(a + b) + a/(a + c))$
Braun-Blanquet	$a / \max(a + b, a + c)$	Otsuka, Ochiai	$a/\sqrt{(a + b)(a + c)}$
Czekanowski, Sørensen-Dice	$2a/(2a + b + c)$	Sokal-Sneath, Anderberg	$a/(a + 2(b + c))$

Table 3: Similarity coefficients for sequential data

For application to non-binary vectors the summation variables a, b, c can be extended in terms of an embedding language L (Rieck et al., 2006):

$$\begin{aligned} a &= \sum_{w \in L} \min(\phi_w(\mathbf{x}), \phi_w(\mathbf{y})) \\ b &= \sum_{w \in L} [\phi_w(\mathbf{x}) - \min(\phi_w(\mathbf{x}), \phi_w(\mathbf{y}))] \\ c &= \sum_{w \in L} [\phi_w(\mathbf{y}) - \min(\phi_w(\mathbf{x}), \phi_w(\mathbf{y}))] \end{aligned}$$

The above definition of a matches the histogram intersection kernel k provided in Table 1, so that alternatively all summation variables can be expressed by

$$a = k(\mathbf{x}, \mathbf{y}),\ b = k(\mathbf{x}, \mathbf{x}) - k(\mathbf{x}, \mathbf{y}),\ c = k(\mathbf{y}, \mathbf{y}) - k(\mathbf{x}, \mathbf{y}). \tag{2}$$

Hence, one can consider the similarity coefficients given in Table 3 as variations of the histogram intersection kernel, e.g. the Jaccard coefficient can be formulated solely in terms of k:

$$s(\mathbf{x}, \mathbf{y}) = \frac{a}{a + b + c} = \frac{k(\mathbf{x}, \mathbf{y})}{k(\mathbf{x}, \mathbf{x}) + k(\mathbf{y}, \mathbf{y}) - k(\mathbf{x}, \mathbf{y})}$$

3.3 A Generic Framework for Similarity Measures

All of the similarity measures introduced in the previous section share a similar mathematical construction: an inner component-wise function is aggregated over each dimension using an outer operator, e.g. the linear kernel is defined as the sum of component-wise products and the Chebyshev distance as the maximum of component-wise absolute differences.

One can exploit this shared structure to derive a unified formulation for similarity measures (Rieck et al., 2006, 2007), consisting of an inner function m and an outer operator $\oplus$ as follows

$$s(\mathbf{x}, \mathbf{y}) = \bigoplus_{w \in L} m(\phi_w(\mathbf{x}), \phi_w(\mathbf{y})). \tag{3}$$

For convenience in later design of algorithms, we introduce a "multiplication" operator $\otimes$ which corresponds to executing the $\oplus$ operation k times. Thus, for any $n \in \mathbb{N}$ and $x \in \mathbb{R}$, we define $\otimes$ as

$$x \otimes n := \underbrace{x \oplus \ldots \oplus x}_{n}.$$

Given the unified form (3), kernel and distance functions presented in Table 1 and 2 can be re-formulated in terms of $\oplus$ and m. Adaptation of similarity coefficients to the unified form (3) involves a re-formulation of the summation variables a, b and c. The particular definitions of outer and inner functions for the presented similarity measures are given in Table 4. The polynomial and sigmoidal kernels as well as the Geodesic distance are not shown since they can be expressed using a linear kernel. For the Chebyshev distance the operator $\otimes$ represents the identity function, while for all other similarity measures it represents a multiplication.

As a last step towards the development of comparison algorithms, we need to address the high dimensionality of the feature space induced by the embedding language L. The unified form (3) theoretically involves computation of m over all $w \in L$, which is practically infeasible for most L.

Kernel	$\oplus$	$m(x,y)$
Linear	$+$	$x \cdot y$
Histogram inters.	$+$	$\min(x,y)$

Sim. Coef.	$\oplus$	$m(x,y)$
Variable a	$+$	$\min(x,y)$
Variable b	$+$	$x - \min(x,y)$
Variable c	$+$	$y - \min(x,y)$

Distance	$\oplus$	$m(x,y)$
Manhattan	$+$	$\lvert x-y \rvert$
χ^2 distance	$+$	$(x-y)^2/(x+y)$
Canberra	$+$	$\lvert x-y \rvert/(x+y)$
Minkowski^p	$+$	$\lvert x-y \rvert^p$
Chebyshev	$\max$	$\lvert x-y \rvert$
Hellinger^2	$+$	$(\sqrt{x}-\sqrt{y})^2$
Jensen-Shannon	$+$	$H(x,y)$

Table 4: Unified formulation of similarity measures.

Yet the feature space induced by L is sparse, since a sequence $\mathbf{x}$ comprises only a limited number of contiguous subsequences – at most $(|\mathbf{x}|^2+|\mathbf{x}|)/2$ subsequences. As a consequence of the sparseness only very few terms $\phi_w(\mathbf{x})$ and $\phi_w(\mathbf{y})$ in the unified form (3) have non-zero values. We exploit this fact by defining $m(0,0) = \mathbf{e}$, where $\mathbf{e}$ is the neutral element for the operator $\oplus$, so that for any $x \in \mathbb{R}$ holds

$$x \oplus \mathbf{e} = x, \quad \mathbf{e} \oplus x = x$$

By assigning $m(0,0)$ to $\mathbf{e}$, the computation of a similarity measure can be reduced to cases where either $\phi_w(\mathbf{x}) > 0$ or $\phi_w(\mathbf{y}) > 0$, as the term $m(0,0)$ does not affect the result of expression (3). We can now refine the unified form (3) by partitioning the similarity measures into *conjunctive* and *disjunctive* measures using an auxiliary function $\tilde{m}$:

$$s(\mathbf{x},\mathbf{y}) = \bigoplus_{w \in L} \tilde{m}(\phi_w(\mathbf{x}), \phi_w(\mathbf{y}))$$

1. **Conjunctive similarity measures.** The inner function m only accounts pairwise non-zero components, so that for any $x \in \mathbb{R}$ holds $m(x,0) = \mathbf{e}$ and $m(0,x) = \mathbf{e}$.

$$\tilde{m}(x,y) = \begin{cases} m(x,y) & \text{if } x > 0 \textbf{ and } y > 0 \\ \mathbf{e} & \text{otherwise.} \end{cases}$$

   Kernel functions fall into this class, except for the distance-based RBF kernel. By using a kernel to express similarity coefficients as shown in expression (2), similarity coefficients also exhibit the conjunctive property.

2. **Disjunctive similarity measures.** The inner function m requires at least one component to be non-zero, otherwise $m(0,0) = \mathbf{e}$ holds.

$$\tilde{m}(x,y) = \begin{cases} m(x,y) & \text{if } x > 0 \textbf{ or } y > 0 \\ \mathbf{e} & \text{otherwise.} \end{cases}$$

Except for the Geodesic distance, all of the presented distances fall into this class. Depending on the embedding language, this class is computational more expensive than conjunctive measures.

As a result, any similarity measure, including those in Table 1, 2 and 3, composed of an inner and outer function can be applied for efficient comparison of embedded sequences, if (a) a neutral element $\mathbf{e}$ for the outer function $\oplus$ exists and (b) the inner function m is either conjunctive or disjunctive, that is at least $m(0,0) = \mathbf{e}$ holds.

4. Algorithms and Data Structures

We now introduce data structures and algorithms for efficient computation of the proposed similarity measures. In particular, we present three approaches differing in capabilities and implementation complexity covering simple sorted arrays, tries and generalized suffix trees. For each approach, we briefly present the involved data structure, provide a discussion of the comparison algorithm and give run-time bounds for extraction and comparison of embedded sequences. Additionally, we provide implementation details that improve run-time performance in practice.

As an example running through this section we consider the two sequences $\mathbf{x} = abbaa$ and $\mathbf{y} = baaaab$ from the binary alphabet $\mathcal{A} = \{a, b\}$ and the embedding language of 3-grams, $L = \mathcal{A}^3$. For a data structure storing multiple words $w \in L$ of possibly different lengths, we denote by k the length of longest words.

4.1 Sorted Arrays

Data structure. A simple and intuitive representation for storage of embedded sequences are *sorted arrays* or alternatively sorted lists (Joachims, 2002; Rieck et al., 2006; Sonnenburg et al., 2007). Given an embedding language L and a sequence $\mathbf{x}$, all words $w \in L$ satisfying $w \sqsubseteq \mathbf{x}$ are maintained in an array X along with their embedding values $\phi_w(\mathbf{x})$. Each field x of X consists of two attributes: the stored word $\mathit{word}[x]$ and its embedding value $\mathit{phi}[x]$. The length of an array X is denoted by $|X|$. In order to support efficient comparison, the fields of X are sorted by contained words, e.g. using the lexicographical order of the alphabet $\mathcal{A}$. Figure 1 illustrates the sorted arrays of 3-grams extracted from the two example sequences $\mathbf{x}$ and $\mathbf{y}$.

Algorithm. Comparison of two sorted arrays X and Y is carried out by looping over the fields of both arrays in the manner of merging sorted arrays (Knuth, 1973). During each iteration the inner function m is computed over contained words and aggregated using the operator $\oplus$. The

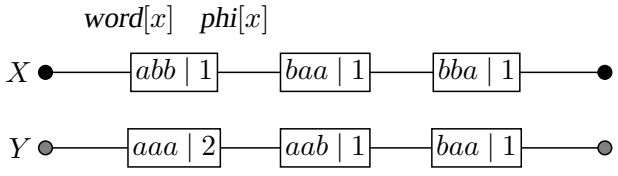


Figure 1: Sorted arrays of 3-grams for $\mathbf{x} = abbaa$ and $\mathbf{y} = baaaab$. The number in each field indicates the number of occurrences.

corresponding comparison procedure in pseudo-code is given in Algorithm 1. Herein, we denote the case where a word w is present in $\mathbf{x}$ and $\mathbf{y}$ as *match* and the case of w being contained in either $\mathbf{x}$ or $\mathbf{y}$ as *mismatch*. For run-time improvement, these mismatches can be ignored in Algorithm 1 if a conjunctive similarity measure is computed (cf. Section 3.3).

Algorithm 1 Array-based sequence comparison

```
1:  function COMPARE(X, Y : Array) : ℝ
2:      s ← e, i ← 1, j ← 1
3:      while i ≤ |X| or j ≤ |Y| do
4:          x ← X[i], y ← Y[j]
5:          if y = NIL or word[x] < word[y] then                ▷ Mismatch at x
6:              s ← s ⊕ m(phi[x], 0)
7:              i ← i + 1
8:          else if x = NIL or word[x] > word[y] then           ▷ Mismatch at y
9:              s ← s ⊕ m(0, phi[y])
10:             j ← j + 1
11:         else                                                ▷ Match at x and y
12:             s ← s ⊕ m(phi[x], phi[y])
13:             i ← i + 1, j ← j + 1
14:     return s
```

Run-time. The comparison algorithm based on sorted arrays is simple to implement, yet it does not enable linear-time comparison for all embedding languages, e.g. if $L = \mathcal{A}^*$. However, sorted arrays enable linear-time similarity measures, if there exist $O(|\mathbf{x}|)$ words with $w \sqsubseteq \mathbf{x}$, which implies all $w \in L$ have no or constant overlap in $\mathbf{x}$. Examples are the common bag-of-words and k-gram embedding languages.

Under these constraints a sorted array is extracted from a sequence $\mathbf{x}$ in $O(k|\mathbf{x}|)$ time using linear-time sorting, e.g. radix sort (Knuth, 1973), where k is the maximum length of all words $w \in L$ in $\mathbf{x}$. The comparison algorithm requires at most $|\mathbf{x}| + |\mathbf{y}|$ iterations, so that the worst-case run-time is $O(k(|\mathbf{x}| + |\mathbf{y}|))$. For extraction and comparison the run-time complexity is linear in the sequence lengths due to the constraint on constant overlap of words, which implies $k|\mathbf{x}| \in O(|\mathbf{x}|)$ for any k and $\mathbf{x}$.

Implementation notes. The simple design of the sorted array approach enables a very efficient extension. If we consider a CPU with registers of b bits, we restrict the maximum word length k, so that every word fits into a CPU register. This restriction enables storage and comparison operations to be performed directly on the CPU, that is operations on words w with $|w| \leq k$ are executed in $O(1)$ time. Depending on the size of the alphabet $|\mathcal{A}|$ and the CPU bits b, the maximum word length is $\lfloor b / \log_2 |\mathcal{A}| \rfloor$. In many practical applications one can strongly benefit from this extensions, as k is either bounded anyway, e.g. for k-grams, or longer words are particular rare and do not increase accuracy significantly. For example on current 64 bit CPU architectures one can restrict k to 32 for DNA sequences with $|\mathcal{A}| = 4$ and to $k = 10$ for textual documents with $|\mathcal{A}| \leq 64$. Alternatively, embedded words may also be represented using hash values of b bits, which enables considering words of arbitrary length, but introduces a probability for hash collisions (see Knuth, 1973).

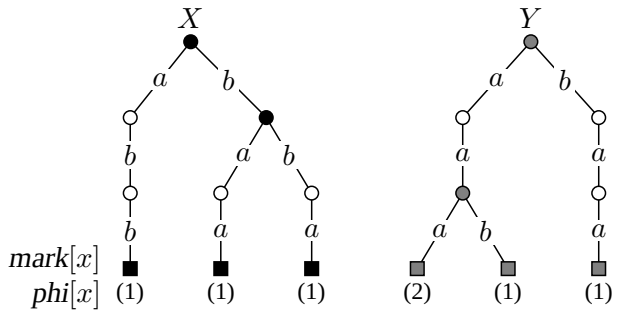


Figure 2: Tries of 3-grams for $\mathbf{x} = abbaa$ and $\mathbf{y} = baaaab$. The number in brackets at leaves indicate the number of occurrences. Marked nodes are squared. White nodes are implicit and not maintained in a compact trie representation.

Another extension for computation of conjunctive measures using sorted arrays has been proposed by Sonnenburg et al. (2007). If two sequences $\mathbf{x}$ and $\mathbf{y}$ have unbalanced sizes $|\mathbf{x}| \ll |\mathbf{y}|$, one loops over the shorter sorted array X and performs a binary search procedure on Y, in favor of processing both sorted arrays in parallel. The worst-case run-time for this comparison is $O(k(|\mathbf{x}| \log_2 |\mathbf{y}|))$, so that one may automatically apply this extension if for two sequences $\mathbf{x}$ and $\mathbf{y}$ holds $|\mathbf{x}| \log_2 |\mathbf{y}| < |\mathbf{x}| + |\mathbf{y}|$.

4.2 Tries

Data structure. A *trie* is a tree structure for storage and retrieval of sequences. The edges of a trie are labeled with symbols of $\mathcal{A}$ (Fredkin, 1960; Knuth, 1973). A path from the root to a marked node x represents a stored sequence, hereafter denoted by $\bar{x}$. A trie node x contains a vector of size $|\mathcal{A}|$ linking to child nodes, a binary flag to indicate the end of a sequence $\mathit{mark}[x]$ and an embedding value $\mathit{phi}[x]$.[1] The i-th child node representing the i-th symbol in $\mathcal{A}$ is accessed via $\mathit{child}[x, i]$. If the i-th child is not present $\mathit{child}[x, i] = \text{NIL}$.

A sequences $\mathbf{x}$ is embedded using a trie X by storing all $w \in L$ with $w \sqsubseteq \mathbf{x}$ and corresponding $\phi_w(\mathbf{x})$ in X (Shawe-Taylor and Cristianini, 2004). Figure 2 shows tries of 3-grams for the two example sequences $\mathbf{x}$ and $\mathbf{y}$. Note, that for the embedding language of k-grams considered in Figure 2 all marked nodes are leaves, while for other embedding languages they may correspond to inner nodes, e.g. for the case of blended k-grams, where every node in a trie marks the end of a sequence.

Algorithm. Comparison of two tries is performed as in Algorithm 2: Starting at the root nodes, one recursively traverses both tries in parallel. If the traversal passes at least one marked node the inner function m is computed as either a matching or mismatching word occurred (Rieck et al., 2006). To simplify presentation of the algorithm, we assume that $\mathit{mark}[\text{NIL}]$ returns false and $\mathit{child}[\text{NIL}, i]$ returns NIL.

1. For convenience, we assume that child nodes are maintained in a vector, while in practice sorted lists, balanced trees or hash tables may be preferred.

Algorithm 2 Trie-based sequence comparison

```
1:  function COMPARE(X, Y : Trie) : ℝ
2:      s ← e
3:      if X = NIL and Y = NIL then                          ▷ Base case
4:          return s
5:      for i ← 1, |A| do
6:          x ← child[X, i],  y ← child[Y, i]
7:          if mark[x] and not mark[y] then                  ▷ Mismatch at x
8:              s ← s ⊕ m(phi[x], 0)
9:          if not mark[x] and mark[y]  then                 ▷ Mismatch at y
10:             s ← s ⊕ m(0, phi[y])
11:         if mark[x] and mark[y] then                      ▷ Match at x and y
12:             s ← s ⊕ m(phi[x], phi[y])
13:         s ← s ⊕ COMPARE(x, y)
14:     return s
```

Run-time. The trie-based approach enables linear-time similarity measures over a larger set of formal languages than sorted arrays. For tries we require all $w \in L$ with $w \sqsubseteq \mathbf{x}$ to have either constant overlap in $\mathbf{x}$ or to be prefix of another word, e.g. as for the blended k-gram embedding languages.

To determine the run-time complexity on tries, we need to consider the following property: A trie storing n words of maximum length k has depth k and at most kn nodes. Thus, a sequence $\mathbf{x}$ containing $O(|\mathbf{x}|)$ words of maximum length k is embedded using a trie in $O(k|\mathbf{x}|)$ run-time. As an invariant for the comparison procedure, the nodes x and y in Algorithm 2 stay at the same depth in each recursion. Hence, the comparison algorithm visits at most $k|\mathbf{x}| + k|\mathbf{y}|$ nodes, which results in a worst-case run-time of $O(k(|\mathbf{x}| + |\mathbf{y}|))$. The extraction and comparison run-time is linear in the sequence lengths, as we require words to either have constant overlap, which implies $k|\mathbf{x}| \in O(|\mathbf{x}|)$, or to be prefix of another word, which implies that both words share an identical path in the trie.

Implementation notes. The first extension for the trie data structure is aggregation of embedding values in nodes. If in Algorithm 2 a mismatch occurs at node x, the algorithm recursively descends to all child nodes of x. At this point, however, it is clear that all nodes below x will also be mismatches, as all words w with $\bar{x} \sqsubseteq_p w$ are not present in the compared trie. This fact can be exploited by maintaining an aggregated value φ_x at each node x given by

$$\varphi_x := \bigoplus_{w \in I} \phi_w(\mathbf{x}) \text{ with } I = \{w \in L \mid \bar{x} \sqsubseteq_p w\}.$$

Instead of recursively descending at a mismatching node x, one utilizes φ_x to retrieve the aggregation of all lower embedding values. The extension allows disjunctive similarity measures to be computed as efficient as conjunctive measures at a worst-case run-time of $O(k \min(|\mathbf{x}|, |\mathbf{y}|))$.

The second extension originates from the close relation of tries and suffix trees. The nodes of a trie can be classified as *implicit* if they link to only one child node and as *explicit* otherwise. By iteratively removing implicit nodes and appending their labels to edges of explicit parent nodes one obtains a *compact trie* (cf. Knuth, 1973; Gusfield, 1997). Edges are labeled by subsequences

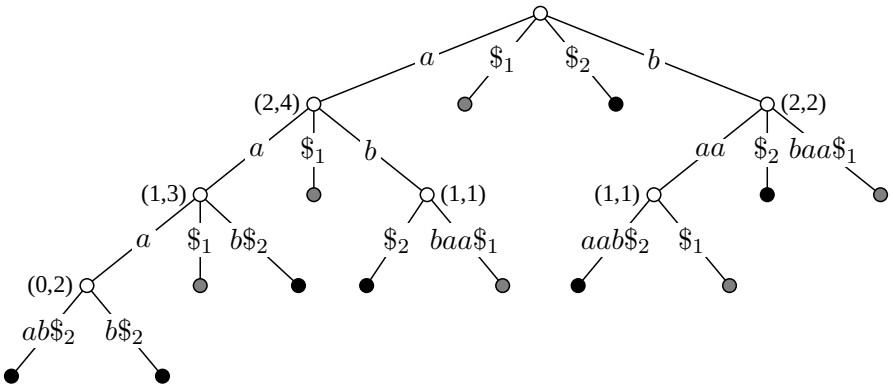


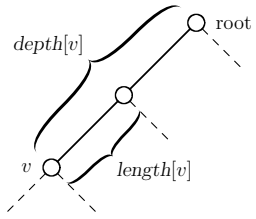
Figure 3: Generalized suffix tree for $\mathbf{x} = abbaa\$_1$ and $\mathbf{y} = baaaab\$_2$. The numbers in brackets at each inner node v correspond to *phi*$[v, 1]$ and *phi*$[v, 2]$. Edges are shown with associated subsequences instead of indices.

encoded using indices i and j pointing to $\mathbf{x}[i..j]$. The major benefit of compact tries is reduced space complexity, which decreases from $O(k|\mathbf{x}|)$ to $O(|\mathbf{x}|)$ independent of the maximum length k of stored words.

4.3 Generalized Suffix Trees

Data structure. A *generalized suffix tree* (GST) is a compact trie containing all suffixes of a set of sequences $\mathbf{x}_1, \ldots, \mathbf{x}_l$ (Gusfield, 1997). Every path in a GST from the root to a leaf corresponds to one suffix. A GST is obtained by extending each sequence $\mathbf{x}_i$ with a delimiter $\$_i \notin \mathcal{A}$ and constructing a suffix tree from the concatenation $\mathbf{z} = \mathbf{x}_1\$_1 \ldots \mathbf{x}_l\$_l$.

For each GST node v we denote by *children*$[v]$ the set of child nodes, by *length*$[v]$ the number of symbols on the incoming edge, by *depth*$[v]$ the total number of symbols on the path from the root node to v and by *phi*$[v, i]$ the number of suffixes of $\mathbf{x}_i$ passing through node v. As every subsequence of $\mathbf{x}_i$ is a prefix of some suffix, *phi*$[v, i]$ reflects the occurrences (alternatively frequency or binary flag) for all subsequences terminating on the edge to v. An example of a GST is given in Figure 3.



In the remaining part we focus on the case of two sequences $\mathbf{x}$ and $\mathbf{y}$ delimited by $\$_1$ and $\$_2$, computation of similarity measures over a set of sequences being a straightforward extension.

Algorithm. Computation of similarity measures is carried out by traversing a GST in depth-first order (Rieck et al., 2007). An implementation in pseudo-code is given in Algorithm 3. At each node v the inner function m is computed using *phi*$[v, 1]$ and *phi*$[v, 2]$. To account for different words along an edge and to support various embedding languages the function FILTER is employed, which selects appropriate contributions similar to the weighting introduced by Vishwanathan and Smola (2004). At a node v the function takes *length*$[v]$ and *depth*$[v]$ of v as arguments to determine how much the node and its incoming edge contribute to the similarity measure, e.g. for the embedding language of k-grams only nodes up to a path depth of k need to be considered.

Algorithm 3 GST-based sequence comparison

```
1:  function COMPARE(X, Y : A*) : R
2:      T ← CONCAT(X, Y)
3:      S ← SUFFIXTREE(T)
4:      return TRAVERSE(root[S])
5:  function TRAVERSE(v : Node) : R
6:      s ← e
7:      for c ← children[v] do
8:          s ← s ⊕ TRAVERSE(c)                    ▷ Depth-first traversal
9:      n ← FILTER(length[v], depth[v])            ▷ Filter words on edge to v
10:     s ← s ⊕ m(phi[v, 1], phi[v, 2]) ⊗ n
11:     return s
```

Algorithm 4 shows a filter function for k-grams. The filter returns 0 for all edges that do not correspond to a k-gram, either because they are too shallow or too deep in the GST, and returns 1 if a k-gram terminates on the edge to a node v.

Algorithm 4 Filter function for k-grams, $L = \mathcal{A}^k$

```
function FILTER(v : Node) : N
    if depth[v] ≥ k and depth[v] − length[v] < k then
        return 1
    return 0
```

Another example of a filter is given in Algorithm 5. The filter implements the embedding language $L = \mathcal{A}^*$. The incoming edge to a node v contributes to a similarity measure by $\mathit{length}[v]$, because exactly $\mathit{length}[v]$ contiguous subsequences terminate on the edge to v.

Algorithm 5 Filter function for all contiguous subsequences, $L = \mathcal{A}^*$

```
function FILTER(v : Node) : N
    return length[v]
```

The bag-of-words model can be implemented either by encoding each word as a symbol of $\mathcal{A}$ or by augmenting nodes to indicate the presence of delimiter symbols on edges. Further definitions of weighting schemes for string kernels, which are suitable for Algorithm 3, are given by Vishwanathan and Smola (2004).

Run-time. Suffix trees are well-known for their ability to enhance run-time performance of string algorithms (Gusfield, 1997). The advantage exploited herein is that a suffix tree comprises a quadratic amount of information, namely all suffixes, in a linear representation. Thus, a GST enables linear-time computation of similarity measures, even if a sequence $\mathbf{x}$ contains $O(|\mathbf{x}|^2)$ words and the embedding language corresponds to $L = \mathcal{A}^*$.

There are well-known algorithms for linear-time construction of suffix trees (e.g. Weiner, 1973; McCreight, 1976; Ukkonen, 1995), so that a GST for two sequences $\mathbf{x}$ and $\mathbf{y}$ can be constructed in $O(|\mathbf{x}| + |\mathbf{y}|)$ using the concatenation $\mathbf{z} = \mathbf{x}\$_1\mathbf{y}\$_2$. As a GST contains at most $2|\mathbf{z}|$ nodes,

the worst-case run-time of any traversal is $O(|\mathbf{z}|) = O(|\mathbf{x}| + |\mathbf{y}|)$. Consequently, computation of similarity measures between sequences using a GST can be realized in time linear in the sequence lengths independent of the complexity of L.

Implementation notes. In practice the GST algorithm may suffer from high memory consumption, due to storage of child nodes and suffix links. To alleviate this problem an alternative data structure with similar properties – *suffix arrays* – was proposed by Manber and Myers (1993). A suffix array is an integer array enumerating the suffixes of a sequence $\mathbf{z}$ in lexicographical order. It can be constructed in linear run-time, however, algorithms with super-linear run-time are surprisingly faster on real-world data (see Manzini and Ferragina, 2004; Maniscalco and Puglisi, 2006).

Abouelhoda et al. (2004) propose a generic procedure for replacing suffix trees with enhanced suffix array, e.g. as performed for the string kernel computation of Teo and Vishwanathan (2006). This procedure involves several auxiliary data structures for maintenance of child nodes and suffix links. In our implementation we follow a different approach based on the work of Kasai et al. (2001a,b). Using a suffix array and an array of longest-common prefixes (LCP) for suffixes, we replace the traversal of the GST by looping over a generalized suffix array in linear time.

Application of suffix arrays reduces memory requirements by a factor of 4. About $11|\mathbf{z}|$ bytes are required for the modified GST algorithm: 8 bytes for a suffix and inverse suffix array, 2 bytes for sequence indices and on average 1 byte for an LCP array. In comparison, a suffix tree usually requires over $40|\mathbf{z}|$ bytes (Abouelhoda et al., 2004) and the enhanced suffix array of Teo and Vishwanathan (2006) about $19|\mathbf{z}|$ bytes.

5. Experiments and Applications

In order to evaluate the run-time performance of the proposed comparison algorithms in practice and to investigate the effect of different similarity measures on sequential data, we conducted experiments on real world sequences. We chose nine data sets from the domains of bioinformatics, text processing and computer security. Details about each data set, contained sequences and references are given in Table 5.

5.1 Run-time Experiments

The linear-time algorithms presented in Section 4 build on data structures of increasing complexity and capability – sorted arrays are simple but limited in capabilities, tries are more involved, yet they do not cover all embedding languages and generalized suffix trees are relatively complex and support the full range of embedding languages. In practice, however, it is the absolute and not asymptotic run-time that matters. Since the absolute run-time is affected by hidden constant factors, depending on design and implementation of an algorithm, it can only be evaluated experimentally.

Therefore each algorithm was implemented using enhancements covered in implementation notes. In particular, for Algorithm 1 we incorporated 64-bit variables to realize a sorted 64-bit array, for Algorithm 2 we implemented a compact trie representation and for Algorithm 3 we used generalized suffix arrays in favor of suffix trees. For each of these algorithms we conducted experiments using different embedding languages to assess the run-time on the data sets given in Table 5.

We applied the following experimental procedure and averaged run-time over 10 individual runs: 500 sequences are randomly drawn from a data set and a 500×500 matrix is computed for the Manhattan distance using a chosen embedding language. The run-time of the matrix computation is

Name	Sequence type	#	$\lvert\mathcal{A}\rvert$	$\lvert\mathbf{x}\rvert_\mu$	Reference
Bioinformatics					
ARTS	DNA sequences	46794	4	2400	Sonnenburg et al. (2006)
C.Elegans	DNA sequences	10025	4	10000	Wormbase WS120
SPROT	Protein sequences	150807	23	467	O'Donovan et al. (2002)
Text processing					
Reuters	News articles	19042	92	839	Lewis (1997)
Heise	News articles	30146	96	1800	`www.heise.de`
RFC	Text documents	4589	106	49954	`www.rfc-editor.org`
Computer security					
HIDS	System call traces	25979	83	156	Lippmann et al. (2000)
NIDS	Connection payloads	21330	116	1274	Lippmann et al. (2000)
Spam	Emails bodies	33702	176	1539	Metsis et al. (2006)

Table 5: Data sets of sequences. The number of sequences in each set is denoted by #, the alphabet size by $\lvert\mathcal{A}\rvert$ and the average sequence length by $\lvert\mathbf{x}\rvert_\mu$.

measured and reported in pairwise comparisons per second. Note, that due to the symmetry of the Manhattan distance only $(n^2 + n)/2$ comparisons need to be performed for an $n \times n$ matrix.

Embedding language: bag-of-words. As a first embedding language, we consider the bag-of-words model. Since natural words can be defined only for textual data, we limit the focus of this experiment to text data sets in Table 5. In particular, we use the embedding language of *word k-grams* – covering the classic "bag of words" as word 1-grams – by using an alphabet of words instead of characters. Each symbol of the alphabet is stored in 32 bits, so that up to 2^{32} different words can be represented. Experiments are conducted for values of k ranging from 1 to 8.

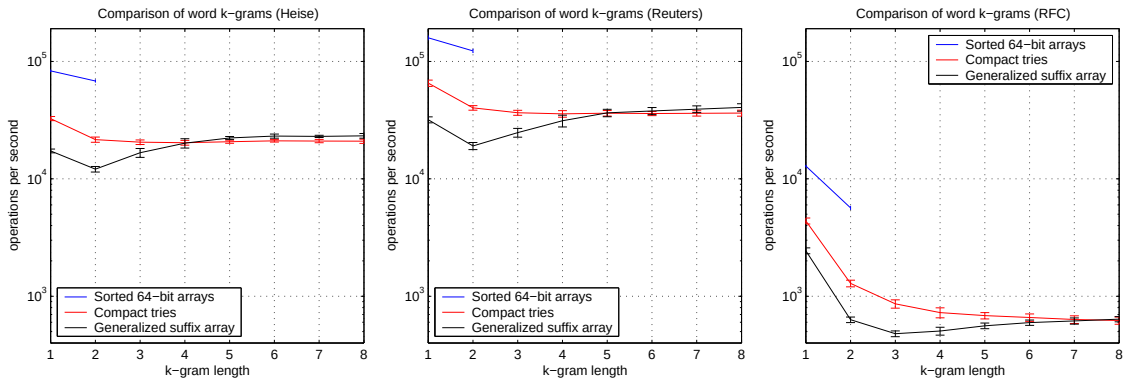


Figure 4: Run-time of sequences comparison over word k-grams for different algorithms. The x-axis gives the word k-gram lengths. The y-axis shows the number of comparison operations per second in log-scale.

Figure 4 shows the run-time performance of the implemented algorithms as a function of k on the Reuters, Heise and RFC data sets. The sorted array approach significantly outperforms the other algorithms on all data sets, yet it can only be applied for $k \leq 2$, as it is limited to 64 bits. For small values of k suffix arrays require more time for each comparison compared to compact tries, while for $k > 5$ their performance is similar to compact tries. This difference is explained by the number of unique k-grams $\nu_{\mathbf{x}}$ in each sequence $\mathbf{x}$. For small values of k often holds $\nu_{\mathbf{x}} < |\mathbf{x}|$, so that a trie comparison requires $O(k(\nu_{\mathbf{x}} + \nu_{\mathbf{y}}))$ time in contrast to $O(|\mathbf{x}| + |\mathbf{y}|)$ for a suffix array. The worse run-time performance on the RFC data set is due to longer sequences.

Embedding language: k-grams. For this experiment we focus on the embedding language of k-grams, which is not limited to a particular type of sequences, so that experiments were conducted for all data sets in Table 5. In contrast to the previous setup, k-grams are associated with the original alphabet of each data set: DNA bases and proteins for bioinformatics, characters for texts, and system calls and bytes for computer security. For each data set the value of k is varied from 1 to 19.

The run-time as a function of k for each data set and algorithm is given in Figure 5. The sorted array approach again yields the best performance on all data sets. Moreover, the limitation of sorted arrays to 64 bits does not effect all data sets, so that for DNA all k-gram lengths can be computed. The suffix array slightly outperforms the trie comparison for larger value of k, as its worst-case run-time is independent of the length of k-grams. Absolute performance in terms of the number of comparisons per second differs among data sets due to different average sequence lengths. For data sets with short sequences (e.g. HIDS, ARTS) performance rates up to 10^6 comparisons per second are attainable, while for data sets with longer sequences (e.g. Spam, RFC) generally up to $10^3 - 10^4$ comparisons per second are achievable at best.

5.2 Applications

We now demonstrate that the ability of our approach to compute diverse similarity measures is beneficial in real applications, especially in unsupervised learning scenarios. Our experiments are performed for: (a) categorization of news articles, (b) intrusion detection in network traffic (c) transcription start site recognition in DNA sequences.

Unsupervised text categorization. For this experiment news articles from the topics "Coffee", "Interest", "Sugar" and "Trade" in the Reuters data set are selected. The learning task is to categorize these articles using clustering, without any prior knowledge of labels. As preprocessing we remove all stop words and words that occur in single articles only. We then compute dissimilarity matrices for the Euclidean, Manhattan and Jensen-Shannon distances using the bag-of-words embedding language as discussed in Section 3. The embedded articles are finally assigned to four clusters using complete linkage clustering (see Duda et al., 2001).

Figure 6(a) shows projections of the embedded articles obtained from the dissimilarity matrices using multidimensional scaling (see Duda et al., 2001). Although projections are limited in describing high-dimensional data, they visualize structure and, thus, help to interpret clustering results. For example, the projection of the Euclidean distances in Figure 6(a) noticeably differs in shape compared to the Manhattan and Jensen-Shannon distances.

The cluster assignments are presented in Figure 6(b) and the distribution of topic labels among clusters is given in Figure 6(c). For the Euclidean distance the clustering fails to unveil features discriminative for article topics, as the majority of articles is assigned to a single cluster. In compar-

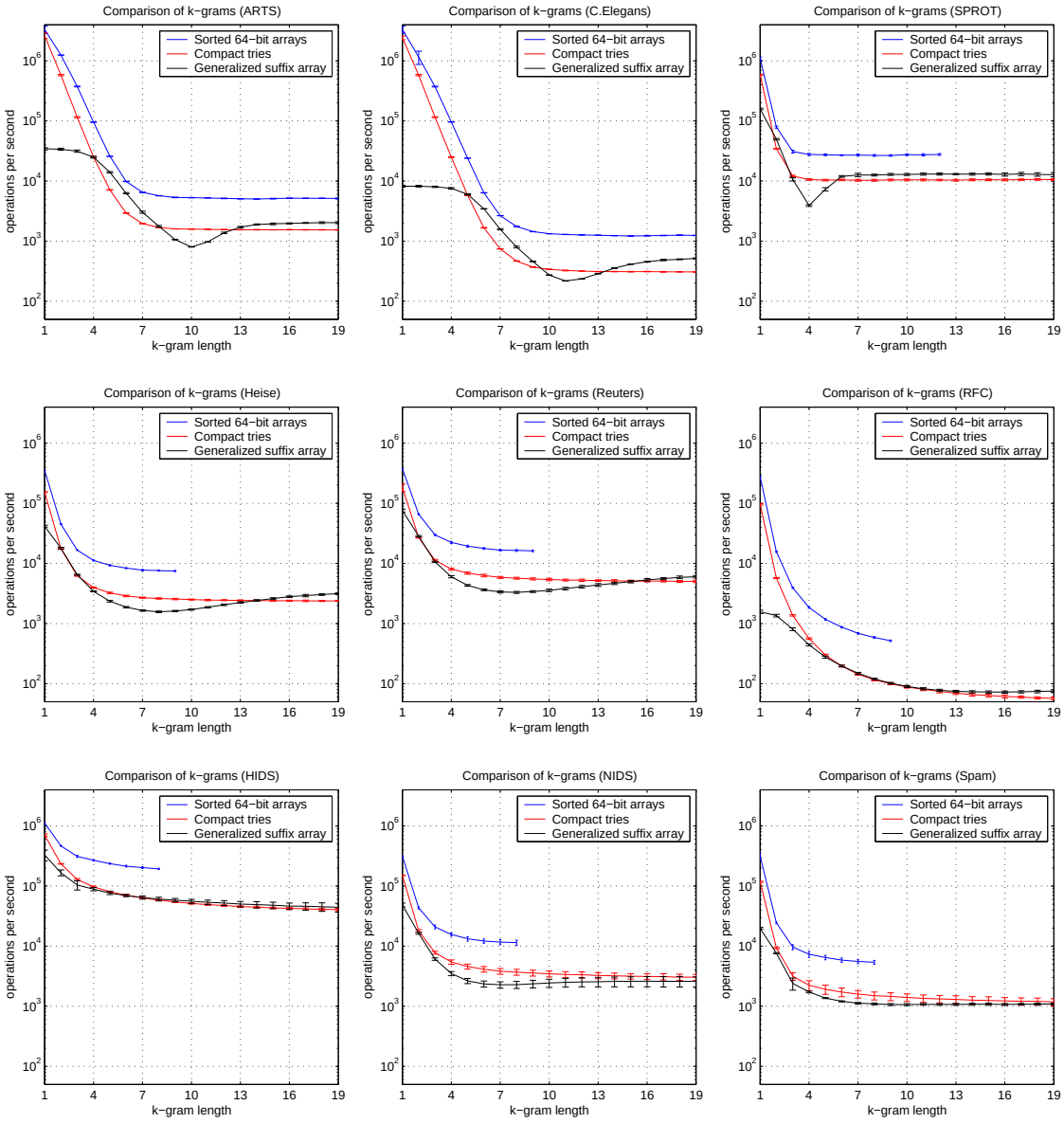
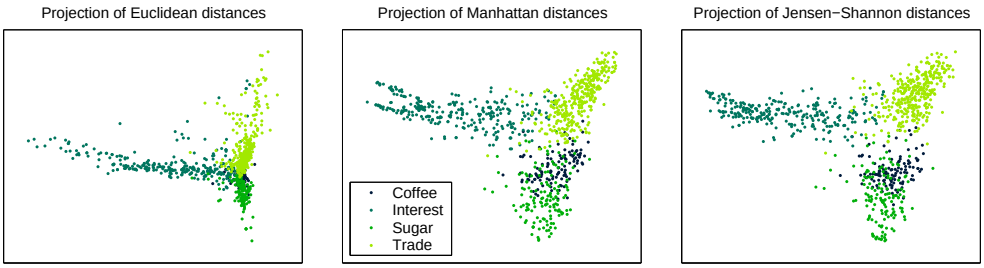
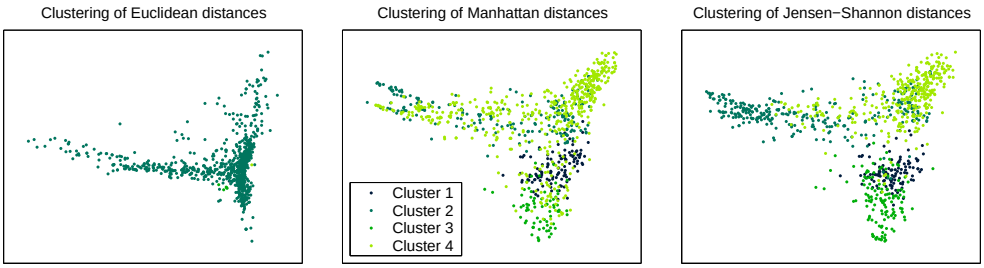


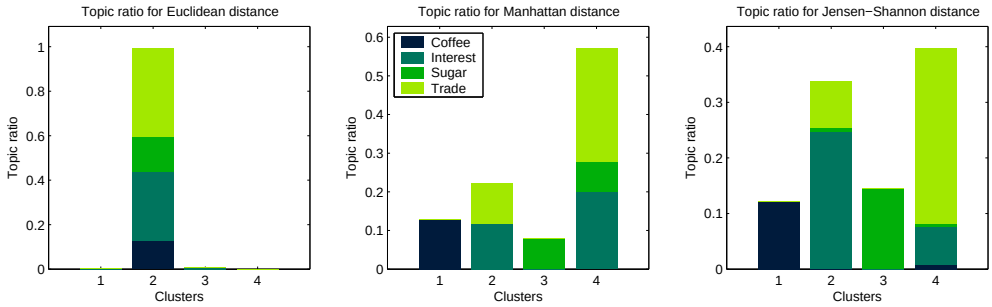
Figure 5: Run-time of sequences comparison over k-grams for different algorithms. The x-axis gives the k-gram lengths. The y-axis shows the number of comparison operations per second in log-scale.



(a) Projection of embedded articles with true topic label assignments



(b) Projection of embedded articles with cluster assignments



(c) Ratio of topic labels in clusters of embedded articles

Figure 6: Clustering of Reuters articles using different similarity measures (bag-of-words).

ison, the Manhattan and Jensen-Shannon distance allow categorization of the topics “Coffee” and “Sugar”, due to observed high frequencies of respective words in articles. However, the Manhattan distance does no allow discrimination of the other two topics, as both are mixed among two clusters. The Jensen-Shannon distance enables better separation of all four topics. The topics “Coffee” and “Sugar” are almost perfectly assigned to clusters and the topics “Interest” and “Trade” each constitute the majority in a respective cluster.

Network intrusion detection. Network intrusion detection aims to automatically identify hacker attacks in network traffic. As labels for such data are hard to obtain in practice, unsupervised learning has gained attention in intrusion detection research. The NIDS data set used for the runtime experiments (cf. Table 5) is known to contain major artifacts (see McHugh, 2000). In order to provide a fair investigation of the impact of various similarity measures on detection of attacks, we generated a smaller private data set. Normal traffic was recorded from the members of our laboratory by providing a virtual network. Additionally attacks were injected by a security expert.

For this experiment we pursue an unsupervised learning approach to network intrusion detection (Rieck and Laskov, 2007). The incoming byte sequences of network connections are embedded using the language of 5-grams, and Zeta, an unsupervised anomaly detection method based on k-nearest neighbors, is applied over the following similarity measures: the Euclidean, Manhattan and Jensen-Shannon distance and the second Kulczynski coefficient (see Section 3.2).

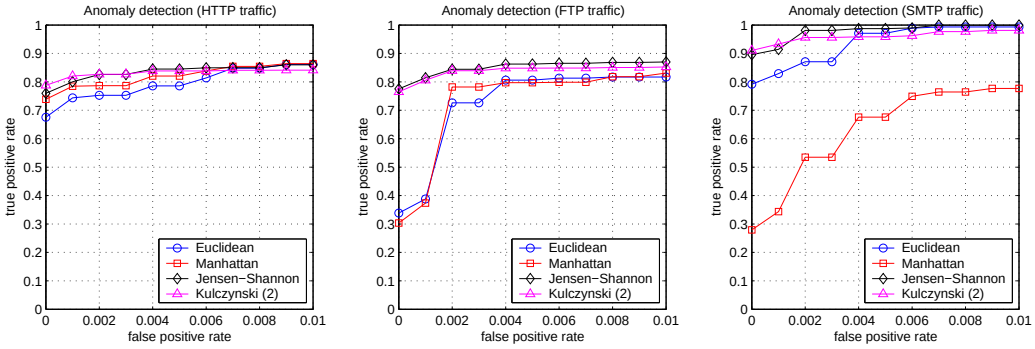


Figure 7: ROC curves for unsupervised anomaly detection on 5-grams of network connections and attacks recorded at our laboratory.

ROC curves for the detection of attacks in the network protocols HTTP, FTP and SMTP are shown in Figure 7. Application of the Jensen-Shannon distance and second Kulczynski coefficient yield the highest detection accuracy. Over 78% of all attacks are identified with no false-positives in an unsupervised setup. In comparison, the Euclidean and Manhattan distance give significantly lower detection rates on the FTP and SMTP protocols. The poor detection performance of the latter two similarity measures emphasizes that choosing a discriminative similarity measure is crucial for achieving high accuracy in a particular application.

Transcription start site recognition. The last application focuses on recognition of transcription start sites (TSS), which mark the beginning of genes in DNA. We consider the ARTS data set, which comprises human DNA sequences that either cover the TSS of protein coding genes or have been extracted randomly from the interior of genes. Following the approach of Sonnenburg et al. (2006) these sequences are embedded using the language of 6-grams and a support vector machine (SVM) and a bagged k-nearest neighbor classifier are trained and evaluated on the different partitions of the data set. Each method is evaluated for the Euclidean distance, the Manhattan distance and the second Kulczynski coefficient. As only 10% of the sequences in the data set correspond to transcription start sites, we additionally apply the unsupervised outlier detection method Gamma (Harmeling et al., 2006), which is similar to the method employed in the previous experiment.

Figure 8 shows the performance achieved by the bagged k-nearest neighbor classifier and the unsupervised learning method.[2] The accuracy in both setups strongly depends on the chosen similarity measures. The metric distances yield better accuracy in the supervised setup. The second Kulczynski coefficient and also the Manhattan distance perform significantly better than the Euclidean distance in unsupervised application. In the absence of labels these measures express better

2. Results for the SVM are similar to the bagged k-nearest neighbor classifier and have been omitted.

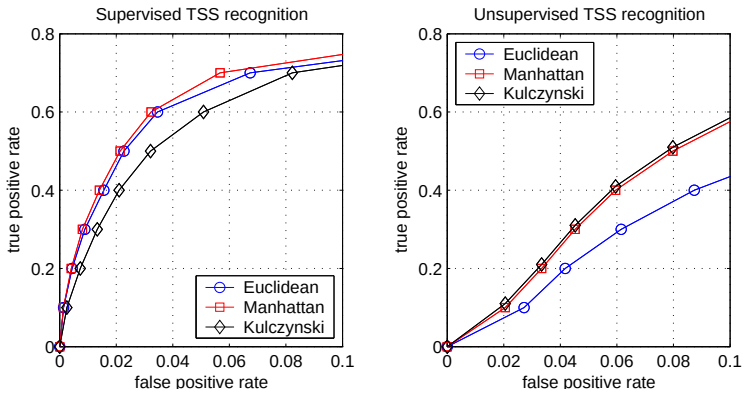


Figure 8: ROC curves for supervised and unsupervised recognition of transcription start sites (TSS) on 6-grams of DNA sequences (ARTS data set).

discriminative properties for TSS recognition, that are difficult to access through Euclidean distances. For the supervised application, the classification performance is limited for all similarity measures, as only some discriminative features for TSS recognition are encapsulated in k-gram models (cf. Sonnenburg et al., 2006).

6. Conclusions

The framework for comparison of sequences proposed in this article provides means for efficient computation of a large variety of similarity measures, including kernels, distances and non-metric similarity coefficients. The framework is based on embedding of sequences in a high-dimensional feature space using formal languages, such as k-grams, contiguous subsequences, etc. Three implementations of the proposed framework using different data structures have been discussed and experimentally evaluated.

Although all three data structures that were considered – sorted arrays, tries and generalized suffix trees – have asymptotically linear run-time, significant differences in the absolute run-time have been observed in our experiments. The constant factors are affected by various design issues illustrated by our remarks on implementation of the proposed algorithms. In general, we have observed a consistent trade-off between run-time efficiency and complexity of embedding languages a particular data structure can handle. Sorted arrays are the most efficient data structure; however, their applicability is limited to k-grams and bag-of-words models. On the other end of the spectrum are generalized suffix trees (and their more space-efficient implementation using suffix arrays) that can handle unrestricted embedding languages – at a cost of more complicated algorithms and lower efficiency. The optimal data structure for computation of similarity measures thus depends on the embedding language to be used in a particular application.

The proposed framework offers machine learning researchers an opportunity to use a large variety of similarity measures for applications that involve sequential data. Although an optimal similarity measure – as it is well known and has been also observed in our experiments – depends on a particular application, the technical means for seamless incorporation of various similarity measures can be of great utility in practical applications of machine learning. Especially appealing is the

possibility for efficient computation of non-Euclidean distances over embedded sequences, which extend applicable similarity measures for sequences beyond inner-products and kernel functions.

In general, the proposed framework demonstrates an important advantage of abstracting data representation – in the form of pairwise relationships – from learning algorithms, which will hopefully motivate further investigation of learning algorithms using a general form of such abstraction.

Acknowledgements

The authors gratefully acknowledge the funding from *Bundesministerium für Bildung und Forschung* under the project MIND (FKZ 01-SC40A) and REMIND (FKZ 01-IS07007A). The authors would like to thank Klaus-Robert Müller, Sören Sonnenburg, Mikio Braun and Vojtech Franc for fruitful discussions and support.

References

M. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

M. Anderberg. *Cluster Analysis for Applications*. Academic Press, Inc., New York, NY, USA, 1973.

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

W. B. Cavnar and J. M. Trenkle. N-gram-based text categorization. In *Proc. SDAIR*, pages 161–175, Las Vegas, NV, USA., Apr. 1994.

O. Chapelle, P. Haffner, and V. Vapnik. SVMs for histogram-based image classification. *IEEE Transaction on Neural Networks*, 9, 1999.

V. Cherkassky, S. Xuhui, F. Mulier, and V. Vapnik. Model complexity control for regression using vc generalization bounds. *IEEE transactions on neural networks*, 10(5):1075–1089, 1999.

M. Collins and N. Duffy. Convolution kernel for natural language. In *Proc. NIPS '2001*, pages 625–632, 2002.

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

H. Drucker, D. Wu, and V. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054, 1999.

R. Duda, P.E.Hart, and D.G.Stork. *Pattern classification*. John Wiley & Sons, second edition, 2001.

E. Fredkin. Trie memory. *Communications of ACM*, 3(9):490–499, 1960.

T. Gärtner, J. Lloyd, and P. Flach. Kernels and distances for structured data. *Machine Learning*, 57 (3):205–232, 2004.

T. Graepel, R. Herbrich, P. Bollmann-Sdorra, and K. Obermayer. Classification on pairwise proximity data. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 438–444. MIT Press, 1999.

D. Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, 1997.

B. Haasdonk. Feature space interpretation of svms with indefinite kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):482–492, 2005.

R. W. Hamming. Error-detecting and error-correcting codes. *Bell System Technical Journal*, 29(2): 147–160, 1950.

S. Harmeling, A. Ziehe, M. Kawanabe, and K.-R. Müller. Kernel-based nonlinear blind source separation. *Neural Computation*, 15:1089–1124, 2003.

S. Harmeling, G. Dornhege, D. Tax, F. C. Meinecke, and K.-R. Müller. From outliers to prototypes: ordering data. *Neurocomputing*, 69(13–15):1608–1618, 2006.

J. Hopcroft and J. Motwani, R. Ullmann. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2 edition, 2001.

T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *J. Comp. Biol.*, 7:95–114, 2000.

D. Jacobs, D. Weinshall, and Y. Gdalyahu. Classification with nonmetric distances: Image retrieval and class representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22 (6):583–600, 2000.

T. Joachims. *Learning to classify text using support vector machines*. Kluwer, 2002.

T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In C. Nédellec and C. Rouveirol, editors, *Proceedings of the European Conference on Machine Learning*, pages 137 – 142, Berlin, 1998. Springer.

T. Kasai, H. Ariumar, and A. Setsuo. Efficient substring traversal with suffix arrays. Technical report, Technical Report 185, Department of Informatics, Kyushu University, 2001a.

T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Combinatorial Pattern Matching (CPM), 12th Annual Symposium*, pages 181–192, 2001b.

D. Knuth. *The art of computer programming*, volume 3. Addison-Wesley, 1973.

J. Laub and K.-R. Müller. Feature discovery in non-metric pairwise data. *Journal of Machine Learning*, 5(Jul):801–818, July 2004.

J. Laub, V. Roth, J. Buhmann, and K.-R. Müller. On the information and representation of non-euclidean pairwise data. *Pattern Recognition*, 39(10):1815–1826, Oct. 2006.

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

C. Leslie, E. Eskin, and W. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proc. Pacific Symp. Biocomputing*, pages 564–575, 2002.

C. Leslie, E. Eskin, A. Cohen, J. Weston, and W. Noble. Mismatch string kernel for discriminative protein classification. *Bioinformatics*, 1(1):1–10, 2003.

V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1966.

D. Lewis. Reuters-21578 text categorization test collection. AT&T Labs Research, 1997.

L. Liao and W. Noble. Combining pairwise sequence similiarity and support vector machines for detecting remote protein evolutionary and structural relationships. *Journal of Computational Biology*, 10:857–868, 2003.

R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, 2000.

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

U. Manber and G. Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

M. Maniscalco and S. Puglisi. An efficient, versatile approach to suffix sorting. *ACM Journal of Experimental Algorithmics*, 2006. to appear.

G. Manzini and P. Ferragina. Engineering a lightweight suffix array construction algorithm. *Algorithmica*, 40:33–50, 2004.

W. Masek and M. Patterson. A faster algorithm for computing string edit distance. *Journal of Computer and System sciences*, 20(1):18–31, 1980.

E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23 (2):262–272, 1976.

J. McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information Systems Security*, 3(4):262–294, 2000.

V. Metsis, G. Androutsopoulos, and G. Paliouras. Spam filtering with naive bayes - which naive bayes? In *Proc. of the 3rd Conference on Email and Anti-Spam (CEAS)*, 2006.

K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2):181–201, May 2001.

S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarties in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

F. Odone, A. Barla, and A. Verri. Building kernels from binary strings for image matching. *IEEE Transactions on Image Processing*, 14(2):169–180, 2005.

C. O'Donovan, M. Martin, A. Gattiker, E. Gasteiger, A. Bairoch, and R. Apweiler. High-quality protein knowledge resource: SWISS-PROT and TrEMBL. *Briefings in Bioinformatics*, 3(3): 275–284, 2002.

C. Ong, X. Mary, S. Canu, and A. Smola. Learning with non-positive kernels. In R. Greiner and D. Schuurmans, editors, *Proceedings of ICML*, pages 639–646. ACM Press, 2004.

G. Rätsch and S. Sonnenburg. *Accurate Splice Site Prediction for Caenorhabditis Elegans*, pages 277–298. MIT Press series on Computational Molecular Biology. MIT Press, 2004.

G. Rätsch, S. Sonnenburg, J. Srinivasan, H. Witte, R. Sommer, K.-R. Müller, and B. Schölkopf. Improving the c. elegans genome annotation using machine learning. *PLoS Computational Biology*, 3:e20, 2007.

K. Rieck and P. Laskov. Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology*, 2(4):243–256, 2007.

K. Rieck, P. Laskov, and K.-R. Müller. Efficient algorithms for similarity measures over sequential data: A look beyond kernels. In *Pattern Recognition, Proc. of 28th DAGM Symposium*, LNCS, pages 374–383, Sept. 2006.

K. Rieck, P. Laskov, and S. Sonnenburg. Computation of similarity measures for sequential data using generalized suffix trees. In *Advances in Neural Information Processing Systems 19*, pages 1177–1184, Cambridge, MA, 2007. MIT Press.

V. Roth, J. Laub, M. Kawanabe, and J. Buhmann. Optimal cluster preserving embedding of nonmetric proximity data. *IEEE Trans. PAMI*, 25:1540–1551, Dec. 2003.

J. Rousu and J. Shawe-Taylor. Efficient computation of gapped substring kernels for large alphabets. *Journal of Machine Leaning Research*, 6:1323–1344, 2005.

G. Salton. Mathematics and information retrieval. *Journal of Documentation*, 35(1):1–29, 1979.

G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

D. Sankoff and J. Kruskal. *Time wraps, String edits, and Macromulecules: The Theory and Practice of Sequence Comparison*. Addision-Wesley Publishing Co., 1983.

B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

B. Schölkopf, P. Simard, A. Smola, and V. Vapnik. Prior knowledge in support vector kernels. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10, pages 640–646, Cambridge, MA, 1998a. MIT Press.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998b.

J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 2004.

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

R. Sokal and P. Sneath. *Principles of Numerical Taxonomy*. W.H. Freeman and Company, San Francisco, CA, USA, 1963.

S. Sonnenburg, A. Zien, and G. Rätsch. ARTS: Accurate Recognition of Transcription Starts in Human. *Bioinformatics*, 22(14):e472–e480, 2006.

S. Sonnenburg, G. Rätsch, and K. Rieck. Large scale learning with string kernels. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, pages 73–103. MIT Press, Cambridge, MA., 2007.

C. Y. Suen. N-gram statistics for natural language understanding and text processing. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1(2):164–172, Apr. 1979.

M. Swain and D. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1), 1991.

C. Teo and S. Vishwanathan. Fast and space efficient string kernels using suffix arrays. In *Proceedings, 23rd ICMP*, pages 939–936. ACM Press, 2006.

K. Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K. Müller. A new discriminative kernel from probabilistic models. *Neural Computation*, 14:2397–2414, 2002.

E. Ukkonen. Online construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

J.-P. Vert, H. Saigo, and T. Akutsu. *Kernel methods in Computational Biology*, chapter Local alignment kernels for biological sequences, pages 131–154. MIT Press, 2004.

S. Vishwanathan and A. Smola. Fast kernels for string and tree matching. In K. Tsuda, B. Schölkopf, and J. Vert, editors, *Kernels and Bioinformatics*, pages 113–130. MIT Press, 2004.

U. von Luxburg and O. Bousquet. Distance-based classification with Lipschitz functions. *Journal of Machine Learning Research*, 5:669–695, 2004.

C. Watkins. Dynamic alignment kernels. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39–50, Cambridge, MA, 2000. MIT Press.

A. Webb. *Statistical Pattern Recognition*. John Wiley and Sons Ltd., 2002.

P. Weiner. Linear pattern matching algorithms. In *Proc. 14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.

A. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Müller. Engineering Support Vector Machine Kernels That Recognize Translation Initiation Sites. *BioInformatics*, 16(9):799–807, Sept. 2000.