Similarity Measures for Sequential Data

Konrad Rieck
Technische Universität Berlin
Germany

This is a preprint of an article published in Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery – http://wires.wiley.com/dmkd

Abstract

Expressive comparison of strings is a prerequisite for analysis of sequential data in many areas of computer science. However, comparing strings and assessing their similarity is not a trivial task and there exists several contrasting approaches for defining similarity measures over sequential data. In this article, we review three major classes of such similarity measures: edit distances, bag-of-word models and string kernels. Each of these classes originates from a particular application domain and models similarity of strings differently. We present these classes and underlying comparison concepts in detail, highlight advantages and differences as well as provide basic algorithms for practical application.

Introduction

Strings and sequences are a natural representation of data in many areas of computer science. For example, several applications in bioinformatics are concerned with studying sequences of DNA, many tasks of information retrieval center on analysis of text documents, and even computer security deals with finding attacks in strings of network packets and file contents. Application of data mining and machine learning in these areas critically depends on the availability of an interface to sequential data, which allows for comparing, analyzing and learning in the domain of strings.

Algorithms for data mining and machine learning have been traditionally designed for vectorial data, whereas strings resemble variable-length objects that do not fit into the rigid representation of vectors. However, a large body of learning algorithms rests on analysis of pairwise relationships between objects, which imposes a looser constraint on the type of data that can be handled. For instance, several learning methods, such as nearest neighbor classification, linkage clustering and multi-dimensional scaling, solely depend on computing distances between objects. This implicit abstraction from vectorial representations can be exploited for applying machine learning to sequential data. Instead of operating in the domain of vectors, the learning methods are applied to pairwise relationships of sequential data, such as distances, angles or kernels defined over strings. As a consequence, a powerful yet intuitive interface for learning with sequential data can be established, which is rooted in assessing the similarity or dissimilarity of strings.

Comparing strings and assessing their similarity is far from trivial, as sequential data is usually characterized by complex and rich content. As an example, this article itself is a string of data and comparison with related texts clearly resembles an involved task. As a consequence, several contrasting approaches have been proposed for comparison of strings, reflecting particular application domains and emphasizing different aspects of strings.

In this article, we review three major classes of similarity measures for sequential data, which have been widely studied in data mining and machine learning. First, we introduce the *edit distance* and related measures, which derive from early telecommunication and model similarity by transforming strings into each other. We then proceed to *bag-of-words models*, which originate from information retrieval and implement comparison of strings by embedding sequential data in a vector space. Finally, we present *string kernels*, a recent class of similarity measures based on the paradigm of kernel-based learning, which allows for designing feature spaces for comparison with almost arbitrary complexity. For each of the three classes of similarity measures, we discuss underlying concepts, highlight advantages and differences to related techniques, and introduce basic algorithms for application in practice.

Notation

Before presenting similarity measures in detail, we introduce some basic notation. Let $\mathcal{A}$ be a set of symbols denoted as *alphabet*, such as the characters in text or the bases in DNA. Then, a *string* x is a concatenation of symbols from the alphabet $\mathcal{A}$. Moreover, we denote the set of all strings by $\mathcal{A}^*$ and refer to the set of strings with length n

as $\mathscr{A}^n$. A function f for comparison of two strings takes the form $f : \mathscr{A}^* \times \mathscr{A}^* \to \mathbb{R}$ where for (dis)similarity measures the returned quantity increases with pairwise (dis)similarity of strings. For simplicity, we use the term similarity measure synonymously with dissimilarity measure, as both types of functions mainly differ in the direction of comparison. To navigate within a string x, we address the i-th symbol of x by $x_{[i]}$ and refer to the symbols from position i to j as a *substring* $x_{[i:j]}$ of x. Finally, we denote the empty string by ϵ and define $|x|$ to be the length of the string x.

Let us, as an example, consider the string $z = \mathsf{cute\ kitty}$. Depending on how we define the alphabet $\mathscr{A}$, the string z either corresponds to a concatenation of characters ($\mathsf{c} \circ \mathsf{u} \circ \mathsf{t} \ldots$) with $|z| = 10$ or a concatenation of words ($\mathsf{cute} \circ \mathsf{kitty}$) with $|z| = 2$. Thus, by selecting the alphabet of strings, we are able to refine the scope of the analysis depending on the application at hand. In the following, we restrict ourselves to the simple alphabet $\mathscr{A} = \{\mathsf{a}, \mathsf{b}\}$ and use the strings $x = \mathsf{ababba}$ and $y = \mathsf{bababb}$ as examples throughout the article. However, all of the presented similarity measures are applicable to arbitrary alphabets, which may range from binary symbols in telecommunication to huge dictionaries for natural language processing.

Edit Distance

As the first class of similarity measures for sequential data, we introduce the *edit distance* and variants thereof. The idea underlying the edit distance is to assess the similarity of two strings by transforming one string into the other and determining the minimal costs for this transformation. Strictly speaking, the edit distance is a metric distance and thus a dissimilarity measure which returns zero if no transformation is required and positive values depending on the amount of changes otherwise. A generic definition of distances for strings is provided in Box 1.

Box 1 *A distance for strings is a function comparing two strings and returning a numeric measure of their dissimilarity. Formally, a function* $d : \mathscr{A}^* \times \mathscr{A}^* \to \mathbb{R}$ *is a* distance *if and only if for all* $x, y, z \in \mathscr{A}^*$ *holds*

$$d(x,y) \geq 0 \qquad \textit{(non-negativity)}$$
$$d(x,y) = 0 \iff x = y \qquad \textit{(isolation)}$$
$$d(x,y) = d(y,x) \qquad \textit{(symmetry).}$$

A function d *is a* metric distance *if additionally*

$$d(x,z) \leq d(x,y) + d(y,z) \qquad \textit{(triangle inequality).}$$

To transform a string x into another string y, we require a set of possible *edit operations* that provide the basis for transformation of strings. In the original formulation of the edit distance these operations are defined as

(a) the *insertion* of a symbol into x,
(b) the *deletion* of a symbol from x and
(c) the *substitution* of a symbol of x with a symbol of y.

A transformation of strings can now be defined as a sequence of edit operations where the costs correspond to the length of this sequence. Assessing the similarity of two strings x and y thus amounts to determining the shortest sequence of edit operations that transforms x to y. In this view, the edit distance $d(x,y)$ is simply the length of this shortest sequence and corresponds to the minimal number of edit operations required to change x into y.

Formally, the edit distance $d(x,y)$ for two strings x and y can be defined recursively, starting with small substrings and continuing until the shortest transformation from x to y has been discovered. As base cases of this recursive definition, we have

$$d(\epsilon,\epsilon) = 0, \quad d(x,\epsilon) = |x|, \quad d(\epsilon,y) = |y|$$

where for the first base case the distance between empty strings is zero and in the other two cases the shortest sequence of operations corresponds to the length of the non-empty string. In all other cases, the edit distance $d(x,y)$ is defined recursively by

$$d(x_{[1:i]}, y_{[1:j]}) = \min \begin{cases} d(x_{[1:i-1]}, y_{[1:j]}) + 1 \\ d(x_{[1:i]}, y_{[1:j-1]}) + 1 \\ d(x_{[1:i-1]}, y_{[1:j-1]}) + m(i,j) \end{cases}$$

in which the indicator function $m(i,j)$ is 1 for a mismatch of symbols $x_{[i]} \neq y_{[j]}$, and 0 for a match of symbols $x_{[i]} = y_{[j]}$.

This recursive definition builds on the technique of dynamic programming: the distance computation for $x_{[1:i]}$ and $y_{[1:j]}$ is solved by considering distances of smaller substrings for each edit operation. In the first case, the symbol $x_{[i]}$ is discarded, which corresponds to deleting a symbol from x. The second case matches an insertion, as the last symbol of $y_{[j]}$ is omitted. If $x_{[i]} \neq y_{[j]}$, the third case reflects a substitution, whereas otherwise $x_{[i]} = y_{[j]}$ and no operation is necessary. The final distance value $d(x,y)$ of x and y is obtained by starting the recursion with $i = |x|$ and $j = |y|$.

In practice, this dynamic programming is realized by keeping intermediate distance values in a table, such that the edit distance of the substrings $x_{[1:i]}$ and $y_{[1:j]}$ can be efficiently determined using results of previous iterations. To illustrate this concept, we consider the strings $x = \mathsf{ababba}$ and $y = \mathsf{bababb}$ and construct a corresponding table.

		x					
		a	b	a	b	b	a
	0	1	2	3	4	5	6
b	**1**	1	1	2	3	4	5
a	2	**1**	2	1	2	3	4
b	3	2	**1**	2	1	2	3
a	4	3	2	**1**	2	2	2
b	5	4	3	2	**1**	2	3
b	6	5	4	3	2	**1**	**2**

y

During computation of the edit distance the table is filled from the upper left to the lower right corner. The cell (i,j) contains the distance value $d(x_{[1:i]}, y_{[1:j]})$ which is computed using the elements $[i,j-1]$, $[i-1,j]$ and $[i-1,j-1]$ determined in previous iterations. In our example, we get $d(x,y)=2$, as the string x can be transformed to y by inserting "b" to its beginning and removing its last symbol "a". The corresponding shortest sequence of edit operations can be obtained by traversing back from the lower right to the upper left corner of the table using an additional table of back pointers. The respective path in the above table is indicated by bold face font.

The run-time complexity for computing the edit distance is quadratic in the length of the strings. In particular, $(|x|+1)\cdot(|y|+1)$ distance values need to be computed, resulting in a complexity of $\Theta(|x|\cdot|y|)$. While there exist techniques to accelerate computation below quadratic complexity (Masek and Patterson, 1980), no linear-time algorithm is known for calculating the edit distance. The computational effort, however, often pays off, as not only a distance but a transformation is returned, which allows for insights into the nature of compared strings.

The edit distance has been first studied by Levenshtein (1966) as a generalization of the Hamming distance—a preceding distance restricted to strings of equal length (Hamming, 1950). The seminal concept of describing similarity in terms of edit operations has been extended in a variety of ways, for example, by incorporation of different cost functions and weightings for operations, symbols and strings. A discussion of several variants is provided by Sankoff and Kruskal (1983). Furthermore, the technique of *sequence alignment* is closely related to the edit distance. Although alignments focus on similarities of strings, the underlying framework of dynamic programming is identical in both settings (Needleman and Wunsch, 1970, Smith and Waterman, 1981, Doolittle, 1986). A comprehensive discussion of edit distances and sequence alignments along with implementations is provided in the book of Gusfield (1997).

Bag-of-Words Models

From the edit distance, we turn to another concept for comparison of strings widely used in data mining applications: *bag-of-words models*. In these models, sequential data is characterized using a predefined set of strings, such as a dictionary of words or fixed terms. We herein refer to this set as an embedding language $L \subseteq \mathcal{A}^*$ and to a string $w \in L$ as a word of L. The set L enables embedding strings in a vector space, such that vectorial similarity measures can be applied for analysis of sequential data. Before studying this embedding and respective similarity measures in detail, we present two definitions of embedding languages frequently used in previous work: *words* and *n-grams*.

The main representation of data in information retrieval and natural language processing are strings of regular text. Thus, L is usually defined as *words* of a natural language, such as English or Esperanto. In this setting, L is either given explicitly by providing a dictionary of terms or implicitly by partitioning strings according to a set of delimiter symbols $D \subset \mathcal{A}$, such that

$$L = (\mathcal{A} \setminus D)^*$$

where L corresponds to all possible concatenations of non-delimiter symbols. Based on this definition, the contents of a string can be described in terms of contained words of L, hence the term *"bag of words"*.

Models based on words are intuitive if strings derive from natural language text or similarly structured sequences. In several applications, however, the structure underlying strings is unknown and hence no delimiters can be defined a priori, for example, as in analysis of DNA and protein sequences. An alternative technique for defining an embedding language L is to move a sliding window of length n over each string and extract *n-grams* (substrings of length n). Formally, this embedding language can be defined as

$$L = \mathcal{A}^n,$$

where $\mathcal{A}^n$ is the set of all possible concatenations with length n. The contents of a string is characterized in terms of contained n-grams, in reference to original bag-of-words models, referred to as a *"bag of n-grams"*.

Equipped with an appropriate embedding language L, we are ready to embed strings in a vector space. Specifically, a string x is mapped to an $|L|$-dimensional vector space spanned by the words of L using a function ϕ defined as

$$\phi : \mathcal{A}^* \to \mathbb{R}^{|L|}, \quad \phi : x \mapsto (\phi_w(x))_{w \in L}$$

where $\phi_w(x)$ returns the number of occurrences of the word w in the string x. In practice, the mapping ϕ is often refined to particular application contexts. For example, $\phi_w(x)$ may be alternatively defined as frequency, probability or binary flag for the occurrences of w in x. Additionaly, a weighting of individual words can be introduced to embedding to account for irrelevant terms in strings (Salton and McGill, 1986).

As an example, let us consider the strings $x = \mathsf{ababba}$ and $y = \mathsf{bababb}$. If we choose the embedding language as $L = \mathcal{A}^3$ and define $\phi_w(x)$ to be the number of occurrences of w in x, we obtain the following vectors for the strings x and y:

$$\phi(x) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \begin{matrix} \mathsf{aaa} \\ \mathsf{aab} \\ \mathsf{aba} \\ \mathsf{abb} \\ \mathsf{baa} \\ \mathsf{bab} \\ \mathsf{bba} \\ \mathsf{bbb} \end{matrix} \quad \text{and} \quad \phi(y) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 2 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \mathsf{aaa} \\ \mathsf{aab} \\ \mathsf{aba} \\ \mathsf{abb} \\ \mathsf{baa} \\ \mathsf{bab} \\ \mathsf{bba} \\ \mathsf{bbb}. \end{matrix}$$

The strings are mapped to 8-dimensional vectors associated with all 3-grams of the alphabet $\mathcal{A} = \{\mathsf{a}, \mathsf{b}\}$. While the edit distance compares the global structure of strings, the embedding characterizes sequential data with a local scope. The substrings aba and abb are shared by both strings and reflected in particular dimensions—irrespective of their position and surrounding in the two strings. Similarly, the absence of the substring bba in y and the repeated occurrences of the substring bab in y are modeled in a local manner. This characterization of local similarity provides the basis for embedding of strings and enables effective comparison by means of vectorial similarity measures.

A large variety of similarity and dissimilarity measure is defined for vectors of real numbers, such as various distance functions, kernel functions and similarity coefficients. Most of these measures can be extended to handle sequential data using the mapping ϕ. For example, we can express the Euclidean distance (also denoted as ℓ^2 distance) as follows

$$d_{\ell_2}(x,y) = \sqrt{\sum_{w \in L} |\phi_w(x) - \phi_w(y)|^2}.$$

Similarly, we can adapt the Manhattan distance (also denoted as ℓ^1 distance) to operate over embedded strings,

$$d_{\ell_1}(x,y) = \sum_{w \in L} |\phi_w(x) - \phi_w(y)|,$$

and define the cosine similarity measure used in information retrieval for comparison of strings by

$$s_{\cos}(x,y) = \frac{\sum_{w \in L} \phi_w(x) \cdot \phi_w(y)}{\sqrt{\sum_{w \in L} \phi_w(x)^2 \sum_{w \in L} \phi_w(y)^2}}.$$

An extensive list of vectorial similarity and dissimilarity measures suitable for application to sequential data is provided by Rieck and Laskov (2008).

Besides the capability to adapt the embedding languages to a particular application domain, the proposed embedding brings about another advantage over other similarity measures, such as the edit distance. Several learning methods can be explicitly formulated in the induced vector space, such that not only the input but also the learned models are expressed using the map ϕ. For example, the centers of k-means clustering and the weights of linear classification can be directly phrased as linear combination of embedded strings, thereby providing a transparent and efficient representation of learned concepts (Sonnenburg et al., 2007).

The vector space induced by words and n-grams is high-dimensional, that is, for a typical alphabet with around 50 symbols an embedding using 4-grams induces $50^4 = 6,250,000$ distinct dimensions. Computing and comparing vectors in such high-dimensional spaces seems intractable at a first glance. However, for both types of languages, words and n-grams, the number of words contained in a string x is linear in its length. Consequently, the vector $\phi(x)$ contains at most $|x|$ non-zero dimensions—regardless of the actual dimensionality of the vector space. This sparsity can be exploited to design very efficient techniques for embedding and comparison of strings, for example, using data structures such as sorted arrays, tries and suffix trees (Rieck and Laskov, 2008). The run-time complexity for comparison is $O(|x| + |y|)$, such that embeddings using words and n-grams are a technique of choice in many real-time applications.

The idea of mapping text to vectors using a "bag of words" has been first introduced by Salton et al. (1975) under the term *vector space model*. Several extensions of this influential work have been studied for information retrieval and natural language processing using different alphabets, similarity measures and learning methods (Salton and McGill, 1986, Joachims, 2002). The concept of using n-grams for modeling strings evolved concurrently to this work, for example for analysis of textual documents and language structure (Suen, 1979, Damashek, 1995, Robertson and Willett, 1998). The ease of mapping strings to vectors and subsequent application of vectorial similarity measures also influenced other fields of computer science, such that there are several related applications in the fields of bioinformatics (Leslie et al., 2002, Sonnenburg et al., 2007) and computer security (Liao and Vemuri, 2002, Hofmeyr et al., 1998, Rieck and Laskov, 2007).

String Kernels

As the last and most recent class of similarity measures for sequential data, we consider *string kernels*. At the first sight, a string kernel is a regular similarity measure for sequential data—though, as we will see shortly, string kernels reach far beyond regular measures of similarity. In contrast

to similarity measures specifically designed for comparison of sequential data, string kernels rest on a mathematical motivation—the concept of kernel functions and the respective field of kernel-based learning. Hence, we first need to explore some underlying theory before moving on to practical kernels for strings.

Box 2 *A kernel for strings is a function comparing two strings and returning a numeric measure of their similarity. Formally, a function* $k : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{R}$ *is a* kernel *if and only if for any finite subset of strings* $\{x_1, \ldots, x_n\} \subset \mathcal{A}^*$*, the function* k *is symmetric and positive semi-definite, that is,*

$$\sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) \geq 0 \text{ for all } c_1, \ldots, c_n \in \mathbb{R}.$$

Note that a kernel is always associated with a (possibly implicit) feature space and corresponds to an inner product

$$k(x, y) = \langle \hat{\varphi}(x), \hat{\varphi}(y) \rangle$$

where $\hat{\varphi}$ *is a map from strings to the feature space.*

Formally, a *kernel function* or short *kernel* is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that assess the similarity of objects from a domain $\mathcal{X}$ and is linked with a *feature space* $\mathcal{F}$—a special form of a vector space (Schölkopf and Smola, 2002). A brief definition of string kernels is provided in Box 2. Simply put, any kernel function k induces a mapping $\hat{\varphi} : \mathcal{X} \to \mathcal{F}$ from the domain $\mathcal{X}$ to the feature space $\mathcal{F}$, where the kernel k equals an inner product in $\mathcal{F}$, that is,

$$k(x, y) = \langle \hat{\varphi}(x), \hat{\varphi}(y) \rangle.$$

This association between a kernel and an inner product allows for establishing a generic interface to the geometry in the feature space $\mathcal{F}$, independent of the type and structure of the input domain $\mathcal{X}$. To illustrate this interface, let us consider some examples, where we assume that the mapping $\hat{\varphi}$ simply embeds strings in $\mathcal{F}$ without giving any details on how this mapping is carried out. We will come back to this point shortly.

As first example, the length (ℓ^2-norm) of a vector corresponding to a string x can be expressed in terms of kernels as follows

$$\|\hat{\varphi}(x)\| = \sqrt{k(x, x)}.$$

Similar to the length, the Euclidean distance between two strings x and y in the feature space $\mathcal{F}$ can be formulated using kernels by

$$\|\hat{\varphi}(x) - \hat{\varphi}(y)\| = \sqrt{k(x, x) - 2k(x, y) + k(y, y)}.$$

Moreover, also the angle $\theta_{x,y}$ between two strings x and y embedded in the feature space $\mathcal{F}$ can be solely expressed in terms of kernel functions using the inverse cosince function as follows,

$$\theta_{x,y} = \arccos \frac{k(x, y)}{\sqrt{k(x, x) \cdot k(y, y)}}.$$

These examples demonstrate how geometric quantities corresponding to distances, angles and norms in feature space can be defined solely in terms of kernel functions. Several learning methods infer dependencies of data using geometry, such as separating hyperplanes, enclosing hyperspheres or sets of descriptive directions. By virtue of kernels all these geometric models can be formulated independent of particular input data $\mathcal{X}$, which builds the basis for kernel-based learning and respective methods, such as support vector machines and kernel principle component analysis (Müller et al., 2001, Schölkopf and Smola, 2002).

It is evident that the mapping $\hat{\varphi}$ is related to the function ϕ introduced for embedding strings in vector spaces and, clearly, by defining $\hat{\varphi} := \phi$, we obtain kernel functions for bag-of-words models. However, the mapping $\hat{\varphi}$ underlying a kernel is more general: any similarity measure with the properties of being *symmetric* and *positive semi-definite* is a kernel and corresponds to an inner product in a feature space—even if the mapping $\hat{\varphi}$ and the space $\mathcal{F}$ are totally unknown. As a result, we gain the ability to design similarity measures for strings which operate in implicit feature spaces and thereby combine the advantages of access to geometry and expressive characterization of strings.

On the basis on this generic definition of kernel functions, a large variety of string kernels has been devised, ranging from kernels for words (Joachims, 1999) and n-grams (Leslie et al., 2002) to involved string comparison with alignments (Watkins, 2000), gaps (Lodhi et al., 2002), mismatches (Leslie et al., 2003) and wild cards (Leslie and Kuang, 2004). From this wide range of kernels, we select the *all-substring kernel* (Vishwanathan and Smola, 2004) for presentation, as it generalizes the idea of characterizing strings using n-grams to all possible substrings, while preserving linear-time comparison. Formally, the all-substring kernel can be defined as follows

$$k(x, y) = \langle \hat{\varphi}(x), \hat{\varphi}(y) \rangle = \sum_{z \in \mathcal{A}^*} \hat{\varphi}_z(x) \cdot \hat{\varphi}_z(y).$$

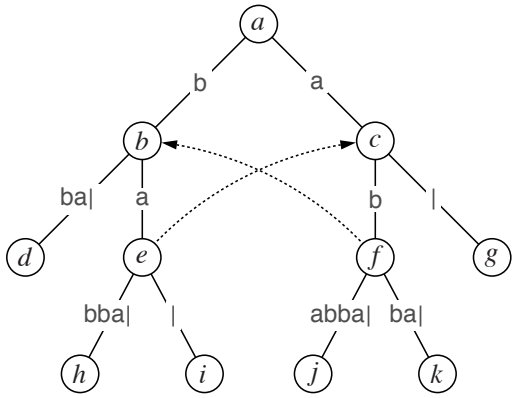
where $\hat{\varphi}_z(x)$ returns the number of occurrences of the substring z in the string x and the function $\hat{\varphi}$ maps strings to a feature space $\mathcal{F} = \mathbb{R}^\infty$ spanned by all possible strings $\mathcal{A}^*$.

In this generic setting, a string x may contain $(|x|^2 + |x|)/2$ different substrings z and explicitly accessing all values $\hat{\varphi}_z(x)$ is impossible in linear time. Thus, techniques for linear-time comparison of sparse vectors, as presented for words and n-grams, are not applicable. The key to efficiently computing the all-substring kernel is implicitly accessing the feature space $\mathcal{F}$ using two data structures: *a suffix tree* and *a matching statistic*.

A suffix tree S_x is a tree containing all suffixes of the string x. Every path from the root to a leaf corresponds to one suffix where edges are labeled with substrings. Suffixes that share the same prefix initially pass along the same edges and nodes. If one path from the root is a suffix of another path, the respective end nodes are connected by a so-called suffix link—a shortcut for traversing the tree. A suffix tree comprises a quadratic amount of information, namely all suffixes of x, in a linear representation and can be constructed in $\Theta(|x|)$ time and space (Gusfield, 1997).

A matching statistic M_y is a compact representation of all matches between two strings x and y. The statistic is computed in $\Theta(|y|)$ time by traversing the suffix tree S_x with the symbols of y, making use of suffix links on mismatches (Chang and Lawler, 1994). The statistic M_y contains only two vectors n and v of length $|y|$, where n_i reflects the length of the longest substring of x matching at position i of y and v_i the node in S_x directly below this match.

Additionally, we augment the nodes of the suffix tree S_x with three fields. For each node v, we store the number of symbols on the path to v in $d(v)$ and the number of leaves below v in $l(v)$. Moreover, we recursively pre-compute $s(v)$ for each node: the number of occurrences of all strings starting at the root and terminating on the path to v.



(a) Suffix tree S_x for string $x =$ ababba

y	b	a	b	a	b	b
n	3	5	4	3	2	1
v	h	j	h	k	d	b

(b) Matching statistic M_y for string $y =$ bababb

Figure 1: Suffix tree and matching statistic for exemplary strings x and y. The additional symbol | indicates the different suffices of x. For simplicity only two suffix links (dotted lines) are shown.

Figure 1 shows a suffix tree and a matching statistic for the exemplary strings x and y. The suffix tree S_x contains six leaves corresponding to the suffices of x. A suffix link connects node f with node b, as the substring terminating at b is a suffix of the path ending at f. Similarly, node e is linked to node c. For simplicity only these two suffix links are shown in Figure 1 where further suffix links have been omitted. The matching statistic M_y contains six columns corresponding to the positions of matches in the string y. For example, at position 3 of M_y, we have $v_3 = 4$ and $n_3 = h$ which indicates a match of length 4 between x and y starting at position 3 of string y and ending on the edge to node h in S_x. Obviously, this match is babb.

Based on the suffix tree S_x and the matching statistic M_y, the all-substring kernel can be computed by passing over M_y as follows

$$\begin{aligned} k(x,y) &= \sum_{z \in \mathcal{A}^*} \hat{\varphi}_z(x) \cdot \hat{\varphi}_z(y) \\ &= \sum_{i=1}^{|y|} \Big[s(v_i) - l(v_i) \cdot (n_i - d(v_i)) \Big]. \end{aligned}$$

To understand how this computation works, we first note that substrings z present in either x or y do not contribute to the kernel, as either $\hat{\varphi}_z(x)$ or $\hat{\varphi}_z(y)$ is zero. Hence, the computation can be restricted to the shared substrings reflected in the matching statistics M_y. Let us now consider a match at position i of M_y: If the match terminates exactly at node the v_i, we have $n_i - d(v_i) = 0$ and $s(v_i)$ corresponds to the occurrences of all shared substrings in x along this path. If the match terminates on the edge, however, $s(v_i)$ is slightly too large and we need to discount all occurrences of substrings below the match by substracting $l(v_i) \cdot (n_i - d(v_i))$. By finally adding up all occurrences of shared substrings in x, we arrive at the all-substring kernel $k(x,y)$.

The run-time for the kernel computation is linear in the lengths of the strings with $\Theta(|x| + |y|)$, as first a suffix tree is constructed and then a matching statistic is computed. Although the induced feature space has infinite dimension and there exist further involved weights of substrings (Vishwanathan and Smola, 2004), the all-substring kernel can be generally computed in linear-time—a capability shared with many string kernels and rooted in implicit access to feature space based on advanced data structures for string comparison.

Due to their versatility and ease of incorporation with kernel-based learning, string kernels have gained considerable attention in research. A large variety of kernels has been developed, starting from first realizations of Haussler (1999) and Watkins (2000), and extending to domain-specific variants, such as string kernels designed for natural language processing (Joachims, 1999, Lodhi et al., 2002) and bioinformatics (Zien et al., 2000, Leslie et al., 2002). The presented all-substring kernel has been proposed by Vishwanathan and Smola (2004). The challenge

	Edit distance	Bag of words	All-substring kernel
Run-time complexity	$O(\|x\| \cdot \|y\|)$	$O(\|x\| + \|y\|)$	$O(\|x\| + \|y\|)$
Implementation	dynamic progamming	sparse vectors	suffix trees
Granularity	characters	words	substrings
Comparison	constructive	descriptive	descriptive
Typical applications	spell checking	text classification	DNA analysis

Table 1: Comparison of similarity measures for sequential data.

of uncovering structure in DNA has influenced further advancement of string kernels by incorporating mismatches, gaps and wild cards (Leslie et al., 2003, Leslie and Kuang, 2004). Additionally, string kernels based on generative models (Jaakkola et al., 2000, Tsuda et al., 2002), position-dependent matching (Rätsch et al., 2005, Sonnenburg et al., 2006) and sequence alignments (Vert et al., 2004, Cuturi et al., 2007) have been devised in bioinformatics. An extensive discussion of several string kernels and applications is provided by Shawe-Taylor and Cristianini (2004) and Sonnenburg et al. (2007).

Comparison of Similarity Measures

As we have seen in the previous sections, there exist different approaches for assessing the similarity of strings. Each of the presented similarity measures compares strings differently and emphasizes other aspects of the sequential data. Consequently, the choice of a similarity measure in practice depends on the contents of the strings as well as the underlying application. As a first guide for the practitioner, Table 1 compares the presented similarity measures and summarizes relevant properties.

The *edit distance* is the similarity measure of choice, if the sequential data exhibits minor perturbations, such as spelling mistakes or typographical errors. As the comparison is constructive, the edit distance does not only assess similarity, but also provides a trace of operations to transform one string into the other. Hence, the edit distance is often applied for comparing and correcting text strings, for example, in applications of spell checking and optical character recognition (Sankoff and Kruskal, 1983). The constructive comparison, however, comes at a price. The run-time complexity of the edit distance is not linear in the length of the strings, such that the comparison of longer strings is often intractable.

The *bag-of-words model* comes particularly handy when analysing natural language text and structured data. The comparison of strings can be controlled using the set of considered words, which allows for adapting the model to the particular context of an application. As a result, bag-of-words models are prevalent in information retrieval and respective applications, such as searching of web pages, classification of text documents or analysis of log files (Salton and McGill, 1986). Moreover, the bag-of-words model enables a linear-time comparison of strings where its implementation using sparse vectors is straightforward (Rieck and Laskov, 2008).

The ability to efficiently operate in complex feature spaces finally renders *string kernels* preferable when learning with sequential data of involved structure. String kernels can be specifically designed for a learning task and equipped with various extensions, such as position-dependent comparison, inexact matching and wild-card symbols (Shawe-Taylor and Cristianini, 2004, Sonnenburg et al., 2007). Due to this flexibility the majority of string kernels has been designed for analysis of DNA and protein sequences, which possess highly complex and largely unknown structure. String kernels are the method of choice, if simpler similarity measures fail to capture the structure of strings sufficiently.

Conclusions

In this article we have studied the comparison of strings and corresponding similarity measures for sequential data. We have reviewed three major classes of such measures, with each modeling similarity differently and emphasizing particular aspects of strings. Starting with the edit distance and reaching over to bag-of-words models and string kernels, we have derived different concepts for assessing the similarity of strings along with their implementations, peculiarities and advantages in practice.

The abstraction realized by analysing pairwise relationships provides a generic interface to structured data beyond strings and sequences. While we have explored different ways for integrating strings into data mining and machine learning, there exists several related approaches modeling similarity and dissimilarity of other non-vectorial data. For example, kernels and distances for trees (Collins and Duffy, 2002, Rieck et al., 2010) and graphs (Gärtner et al., 2004, Vishwanathan et al., 2009) are one strain of development in this field. This divergence of research ultimately provides the basis for tackling challenging problems and designing novel methods that benefit from both worlds: machine learning and comparison of structured data.

Acknowledgments

The author would like to thank Tammo Krueger, Sören Sonnenburg and Klaus-Robert Müller for fruitful discussions and support. Moreover, the author acknowledges funding from the *Bundesministerium für Bildung und Forschung* under the project REMIND (FKZ 01-IS07007A).

References

W. I. Chang and E. L. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12 (4/5):327–344, 1994.

M. Collins and N. Duffy. Convolution kernel for natural language. In *Advances in Neural Information Proccessing Systems (NIPS)*, volume 16, pages 625–632, 2002.

M. Cuturi, J.-P. Vert, and T. Matsui. A kernel for time series based on global alignments. In *Proc. of the International Conferenc on Acoustics, Speech and Signal Processing (ICASSP)*, 2007.

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

R. Doolittle. *Of Urfs and Orfs: A Primer on How to Analyse Derived Amino Acid Sequences*. University of Science Books, 1986.

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

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

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

D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, UC Santa Cruz, July 1999.

S. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

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

T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

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

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.

Y. Liao and V. R. Vemuri. Using text categorization techniques for intrusion detection. In *Proc. of USENIX Security Symposium*, pages 51 – 59, 2002.

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.

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

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. Needleman and C. 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.

G. Rätsch, S. Sonnenburg, and B. Schölkopf. RASE: recognition of alternatively spliced exons in c. elegans. *Bioinformatics*, 21:i369–i377, June 2005.

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 and P. Laskov. Linear-time computation of similarity measures for sequential data. *Journal of Machine Learning Research*, 9(Jan):23–48, 2008.

K. Rieck, T. Krueger, U. Brefeld, and K.-R. Müller. Approximate tree kernels. *Journal of Machine Learning Research*, 11(Feb):555–580, 2010.

A. Robertson and P. Willett. Applications of n-grams in textual information systems. *Journal of Documentation*, 58(1):48–69, 1998.

G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1986.

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.

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

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

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

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

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.

S. Vishwanathan, N. Schraudoplh, R. Kondor, and K. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 2009. to appear.

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. Zien, G. Rätsch, S. Mika, B. Schölkopf, T. Lengauer, and K.-R. Müller. Engineering support vector machine kernels that recognize translation initiation sites in DNA. *BioInformatics*, 16(9):799–807, Sep 2000.