Efficient Algorithms for Similarity Measures over Sequential Data: A Look Beyond Kernels

Konrad Rieck[1], Pavel Laskov[1], and Klaus-Robert Müller[1,2]

[1] Fraunhofer FIRST.IDA, Kekuléstraße 7, 12489 Berlin, Germany
[2] University of Potsdam, Am Neuen Palais 10, 14469 Potsdam, Germany
{rieck, laskov, klaus}@first.fhg.de

Abstract. Kernel functions as similarity measures for sequential data have been extensively studied in previous research. This contribution addresses the efficient computation of distance functions and similarity coefficients for sequential data. Two proposed algorithms utilize different data structures for efficient computation and yield a runtime linear in the sequence length. Experiments on network data for intrusion detection suggest the importance of distances and even non-metric similarity measures for sequential data.

1 Introduction

Sequences are a common non-vectorial data representation used in various machine learning and pattern recognition applications, e.g. textual documents in information retrieval, DNA sequences in bioinformatics or packet payloads in intrusion detection. An essential procedure for analysis of such data is the efficient computation of pairwise similarity between sequences.

Beside specialized string distances [e.g. 1, 2] a large class of similarity measures for sequential data can be defined over contained subsequences by embedding them in a high-dimensional feature space. Previous research focused on computation of kernel functions in such feature spaces. For example, the inner-product over n-gram or word frequencies has been widely used for analysis of textual documents [e.g. 3, 4, 5] or host-based intrusion detection [e.g. 6]. The challenge of uncovering information in DNA has influenced further advancement of kernel functions, e.g. by exploring different sets of subsequences [e.g. 7, 8, 9, 10] or incorporating mismatches, gaps and wildcards [e.g. 11, 12, 13].

There exist, however, a large amount of learning algorithms which are not directly suitable for kernel functions. In principle, any inner-product induces a Euclidean distance in feature space [14], yet the richness of content in sequential data and the variability of its characteristics in feature spaces motivate application of other distance functions.

A general technique for computation of similarity measures suitable for kernels, distances and similarity coefficients is proposed in this contribution. It is based on incremental accumulation of matches and mismatches between subsequences comprising a feature space. Two algorithms are presented that utilize

K. Franke et al. (Eds.): DAGM 2006, LNCS 4174, pp. 374–383, 2006.
© Springer-Verlag Berlin Heidelberg 2006

different data structures for efficient computation: hash tables and tries. Both algorithms have linear runtime complexity in terms of sequence lengths.

The rest of the paper is organized as follows: Section 2 defines several similarity measures for sequential data including kernels, distances and similarity coefficients. Comparison algorithms and corresponding data structures are introduced in Section 3. Finally, experiments in Section 4 compare the efficiency of the introduced algorithms and illustrate their application in network intrusion detection.

2 Similarity Measures for Sequential Data

Given an alphabet Σ of size N, a sequence x is defined as a concatenation of symbols from Σ. The content of a sequence can be modeled as a set of possibly overlapping subsequences w taken from a finite language $L \subset \Sigma^*$. We refer to these extracted subsequences as *words*. The language L constitutes the basis for calculating similarity of sequences and typically corresponds to a bag of characters, words or n-grams. Given a sequence x and a language L, an embedding into feature space is performed by calculating $\phi_w(x)$ for every $w \in L$ appearing in x. Usually the function $\phi_w(x)$ returns the frequency of w in x, however, other definitions returning a count or a binary flag for w are possible. Furthermore we define l to be the length of x.

We assume that the total length of words in every sequence x is proportional to l. This assumption is valid, for example, for n-grams of fixed length n and non-overlapping words, and ensures linear runtime of the proposed algorithms. In context of kernels several approaches have been investigated that do not make such an assumption [e.g. 9, 10, 11, 12, 13], however, some of them come at a cost of super-linear complexity.

By utilizing the feature space induced through ϕ, one can adapt classical kernel and distance functions to operate on sequences. Table 1 lists kernel functions and Table 2 distance functions that are implemented using the algorithms presented in Section 3.

Yet another way of measuring similarity are so called similarity coefficients [e.g. 15, 16]. They are non-metric and have been primarily used on binary data.

Table 1. Kernel functions for sequential data

Kernel function	$k(x, y)$
Linear	$\sum_{w \in L} \phi_w(x)\phi_w(y)$
Polynomial	$\left(\sum_{w \in L} \phi_w(x)\phi_w(y) + \theta\right)^d$
RBF	$\exp\left(\frac{-d(x,y)^2}{\sigma}\right)$

Table 2. Distance functions for sequential data

Distance function	$d(x,y)$
Manhattan	$\sum_{w \in L} \lvert\phi_w(x) - \phi_w(y)\rvert$
Canberra	$\sum_{w \in L} \frac{\lvert\phi_w(x) - \phi_w(y)\rvert}{\phi_w(x) + \phi_w(y)}$
Minkowski	$\sqrt[k]{\sum_{w \in L} \lvert\phi_w(x) - \phi_w(y)\rvert^k}$
Chebyshev	$\max_{w \in L} \lvert\phi_w(x) - \phi_w(y)\rvert$

Table 3. Similarity coefficients for sequential objects

Similarity coefficients	$s(x,y)$
Jaccard	$\frac{a}{a+b+c}$
Czekanowski	$\frac{2a}{2a+b+c}$
Sokal-Sneath	$\frac{a}{a+2(b+c)}$
Kulszynski	$\frac{1}{2}\left(\frac{a}{a+b} + \frac{a}{a+c}\right)$

Similarity coefficients are constructed using three summation variables a, b and c. The variable a contains the number of positive matches (1-1), b the number of left mismatches (0-1) and c the number of right mismatches (1-0). The most common similarity coefficients are given in Table 3.

Similarity coefficients can be extended to non-binary data by modification of the summation variables. The degree of match for a word $w \in L$ can be defined as $\min(\phi_w(x), \phi_w(y))$ and the respective mismatches are defined as deviations thereof:

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

3 Algorithms and Data Structures

In order to calculate the presented kernels, distances and similarity coefficients, one needs to establish a general model of similarity measures for sequential data.

Table 4. Generalized formulations of distances

Distance	$\oplus$	$m^+(p,q)$	$m_x^-(p)$	$m_y^-(q)$
Manhattan	$+$	$\lvert p-q\rvert$	p	q
Canberra	$+$	$\lvert p-q\rvert/(p+q)$	1	1
Minkowski^k	$+$	$\lvert p-q\rvert^k$	p^k	q^k
Chebyshev	max	$\lvert p-q\rvert$	p	q

Table 5. Generalized formulations of summation variables

Variable	$\oplus$	$m^+(p,q)$	$m_x^-(p)$	$m_y^-(q)$
a	$+$	$\min(p,q)$	0	0
b	$+$	$p-\min(p,q)$	p	0
c	$+$	$q-\min(p,q)$	0	q

A key instrument for computation of kernel functions is finding words $w \in L$ present in two sequences x and y – we refer to these words as *matches*. For distances and similarity coefficients, we also need to consider words $w \in L$ present in x but not in y (and vice versa) – we refer to these words as *mismatches*[1].

Furthermore we introduce an outer function $\oplus$ which corresponds to the global aggregation performed in many similarity measures, e.g. the summation in various kernel and distance functions. Given these definitions, we can express a generic similarity measure s as

$$s(x,y) = \bigoplus_{w \in L} m(x,y,w) \tag{1}$$

$$m(x,y,w) = \begin{cases} m^+(\phi_w(x), \phi_w(y)) & \text{if } w \text{ is a match} \\ m_x^-(\phi_w(x)) & \text{if } w \text{ is a mismatch in } x \\ m_y^-(\phi_w(y)) & \text{if } w \text{ is a mismatch in } y \end{cases} \tag{2}$$

We can now reformulate the set of distances given in Table 2 using the functions $\oplus$, m^+, m_x^- and m_y^-. The generalized formulations of some distances are presented in Table 4.

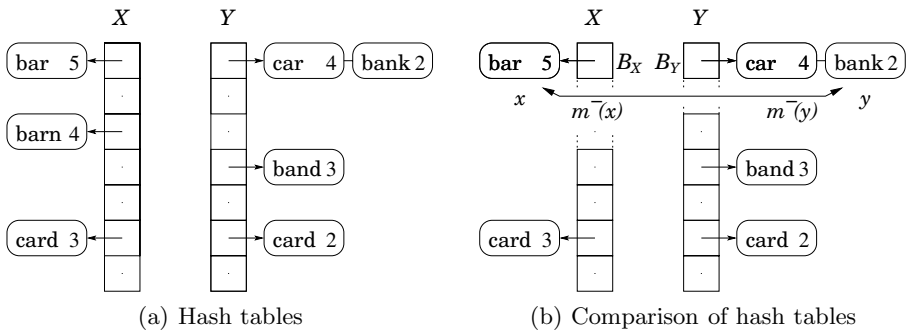
Adapting similarity coefficients to such a generic representation is even simpler, since only the three summation variables a, b and c need to be reformulated, as shown in Table 5.

3.1 Hash-Based Sequence Comparison

The classical scheme for computation of similarity measures over sequences utilizes indexed tables, or in the more general case hash tables [e.g. 4]. The words extracted from a sequence and corresponding frequencies or counts are stored in

[1] The term "mismatch" herein corresponds to two sequences being unequal and not, as often used in bioinformatics, to inexact matching of sequences.

the bins of a hash table. Figure 1(a) shows two hash tables carrying the words {"bar", "barn", "card"} and {"car", "bank", "band", "card"} with corresponding counts.



(a) Hash tables (b) Comparison of hash tables

Fig. 1. Hash table data structures (a) and their comparison (Case 2) (b)

Algorithm 1 defines the comparison of two hash tables X and Y with fixed size M. The algorithm proceeds by looping over all M bins, checking for matching (cf. Algorithm 1: Case 1) and mismatching words (cf. Algorithm 1: Case 2 & 3). Figure 1(b) illustrates this process at the mismatches "bar" and "bank" which are stored in corresponding bins.

Algorithm 1. Hash-based Sequence Comparison

```
function COMPARE(X, Y)
    s ← 0
    for i ← 1, M do
        B_X ← bins[X, i]
        B_Y ← bins[Y, i]
        if B_X ≠ NIL and B_Y ≠ NIL then
            for all x ∈ B_X and y ∈ B_Y do
                if x = y then
                    s ← s ⊕ m^+(x, y)                ▷ Case 1
                else
                    s ← s ⊕ m^-(x) ⊕ m^-(y)          ▷ Case 2
        else if B_X ≠ NIL then
            for all x ∈ B_X do
                s ← s ⊕ m^-(x)                       ▷ Case 3
        else if B_Y ≠ NIL then
            for all y ∈ B_Y do
                s ← s ⊕ m^-(y)                       ▷ Case 3
    return s
```

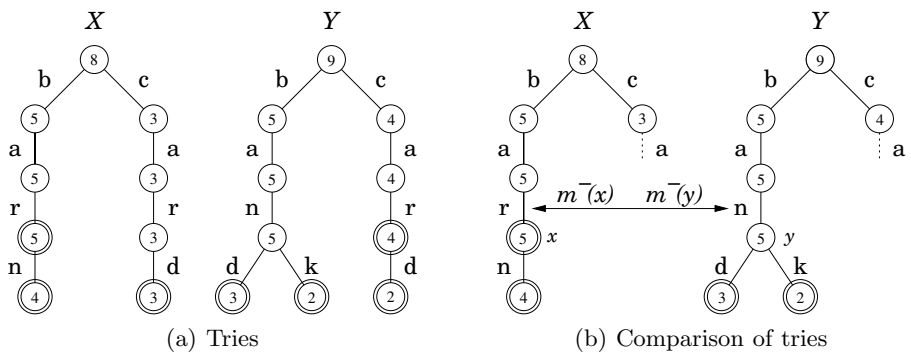
Since the size of the hash tables is fixed at M, the average runtime for a comparison is $\Theta(M)$. To avoid possible hash collisions, a high value of $M \gg l$ must be chosen in advance, otherwise the chaining of bins (Case 2) results in $O(l^2)$ worst-case runtime for $O(l)$ extracted words per sequence.

3.2 Trie-based Sequence Comparison

A trie is an N-ary tree, whose nodes are N-place vectors with components corresponding to the elements of Σ [17]. Figure 2(a) shows two tries X and Y containing the same words as the hash tables in Figure 1(a). The nodes of the tries are augmented to carry a variable reflecting the count of the passing sequence. The end of each extracted word is indicated by a marked circle. Application of tries to computation of kernel functions has been considered by [18].

Depending on the applied similarity measure the trie nodes can be extended to store other aggregated values which speed up calculations involving subtrees, e.g. for the Minkowski distance $\sum_w \phi_w(x)^k$ for all lower words w,

Comparison of two tries can be carried out as in Algorithm 2: Starting at the root nodes, one traverses both tries in parallel, processing matching and mismatching nodes. If the traversal passes two equal and marked nodes, a matching word is discovered (Case 1), if only one node is marked a mismatch occurred (Case 2). The recursive traversal is stopped if two nodes do not match, and thus two sets of underlying mismatching words are discovered (Case 3). Figure 2(b) shows a snapshot of such a traversal. The nodes x and y are not equal, and the words {"bar", "barn"} and {"band", "bank"} constitute mismatches.



(a) Tries (b) Comparison of tries

Fig. 2. Trie data structures (a) and their comparison (Case 3) (b)

As an invariant, the nodes under consideration in both tries remain at the same depth and thus the worst-case runtime is $O(l)$. An advantage of the trie data structure comes into play especially if the provided alphabet is large and a lot of mismatches occur. The traversal discovers mismatching words after passing the first few symbols and omits further unnecessary comparisons.

Algorithm 2. Trie-based Sequence Comparison

```
function COMPARE(X, Y)
    s ← 0
    for i ← 1, N do
        x ← child[X, i]
        y ← child[Y, i]
        if x ≠ NIL and y ≠ NIL then
            if end[x] and end[y] then
                s ← s ⊕ m⁺(x, y)                    ▷ Case 1
            else if end[x] then
                s ← s ⊕ m⁻(x)                       ▷ Case 2
            else if end[y] then
                s ← s ⊕ m⁻(y)                       ▷ Case 2
            s ← s ⊕ COMPARE(x, y)
        else
            if x ≠ NIL then
                s ← s ⊕ m⁻(x)                       ▷ Case 3
            if y ≠ NIL then
                s ← s ⊕ m⁻(y)                       ▷ Case 3
    return s
```

4 Experimental Results

4.1 Efficiency of Data Structures

Efficiency of the two proposed algorithms has been evaluated on four benchmark data sets for sequential data: DNA sequences of the human genome [19], system call traces and connection payloads from the DARPA 1999 data set [20] and news articles from the Reuters-21578 data set [21]. Table 6 gives an overview of the data sets and their specific properties.

For each data set 100 sequences were randomly drawn and n-grams of lengths 3, 5 and 7 extracted. The n-grams of each sequence were stored in tries and hash tables with varying size from 10^2 to 10^5. Subsequently the Canberra distance was calculated pairwise over the tries and hash tables using the proposed algorithms, resulting in 5000 comparison operations per setup. The procedure was repeated 10 times and the runtime was averaged over all runs. The experimental results are given in Table 7.

Table 6. Datasets of sequential objects

Name	Type	Alphabet	Min. length	Max. length
DNA	Human genome sequences	4	2400	2400
HIDS	BSM system call traces	88	5	129340
NIDS	TCP connection payloads	108	53	132753
TEXT	Reuters Newswire articles	93	43	10002

Table 7. Runtime experiments for Canberra distance

Dataset	n	Average runtime for 5000 comparisons (s)				
		Trie	Hash (10^2)	Hash (10^3)	Hash (10^4)	Hash (10^5)
DNA	3	**0.19**	0.22	0.28	0.66	6.04
	5	**2.21**	4.46	2.94	3.56	9.57
	7	10.72	37.63	13.02	**5.67**	9.43
HIDS	3	**0.06**	0.10	0.13	0.62	3.05
	5	0.15	0.15	0.16	0.66	5.23
	7	0.25	**0.19**	0.22	0.70	4.15
NIDS	3	**0.48**	1.70	1.07	1.43	5.12
	5	**0.86**	3.70	1.72	1.81	5.90
	7	**1.20**	4.83	2.10	2.42	6.08
TEXT	3	**1.12**	1.75	1.22	1.63	7.03
	5	1.65	3.85	1.64	1.89	7.58
	7	**2.13**	5.92	2.19	2.24	7.74

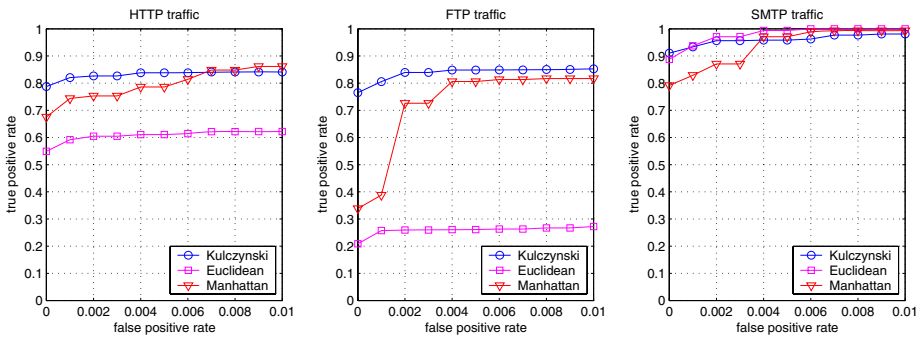


Fig. 3. Detection performance for network intrusion detection

The average runtime of the hash-based algorithm strongly depends on the size of the hash table. The optimal value varies for different data sets and values of n. However, in 10 of 12 cases the trie-based algorithm performs equally well or better than the best hash table setup, being independent of a parameter.

4.2 Application: Network Intrusion Detection

To demonstrate the proposed algorithms on realistic data, we conducted an experiment for unsupervised learning in network intrusion detection. The underlying network data was generated by the members of our laboratory using virtual network servers. Recent network attacks were injected by a penetration-testing expert.

A distance-based anomaly detection method [22] was applied on 5-grams extracted from byte sequences of TCP connections using different similarity measures: a linear kernel (Euclidean distance), the Manhattan distance and the

Kulczynski coefficient. Results for the common network protocols HTTP, FTP and SMTP are given in Figure 3.

Application of the Kulczynski coefficient yields the highest detection accuracy. Over 78% of attacks for each protocol are identified with no false-positives. In comparison the Euclidean distances fails to uncover good geometric properties for discrimination of attacks in this particular setup.

5 Conclusions

We have shown that, similarly to kernels, a large number of distances and similarity coefficients can be efficiently computed for sequential data. The use of such similarity measures allows one to investigate unusual metrics for application of machine learning in specialized problem domains. As an example, the best results in our experiments on unsupervised learning for network intrusion detection have been obtained with the Kulczynski coefficient over n-grams of connection payloads. Thus direct application of distances over sequential data may be favorable over implicit use of the Euclidean distance induced by kernels. Especially promising are further applications of the proposed algorithms in computer security and bioinformatics.

Acknowledgments

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

References

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

[2] Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk SSSR **163**(4) (1964) 845–848

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

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

[5] Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. Technical Report 23, LS VIII, University of Dortmund (1997)

[6] Eskin, E., Arnold, A., Prerau, M., Portnoy, L., Stolfo, S.: A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data. In: Applications of Data Mining in Computer Security. Kluwer (2002)

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

[8] Leslie, C., Eskin, E., Noble, W.: The spectrum kernel: A string kernel for SVM protein classification. In: Proc. Pacific Symp. Biocomputing. (2002) 564–575
[9] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. Journal of Machine Learning Research **2** (2002) 419–444
[10] Vishwanathan, S., Smola, A.: Fast Kernels for String and Tree Matching. In: Kernels and Bioinformatics. MIT Press (2004) 113–130
[11] Leslie, C., Eskin, E., Cohen, A., Weston, J., Noble, W.: Mismatch string kernel for discriminative protein classification. Bioinformatics **1**(1) (2003) 1–10
[12] Leslie, C., Kuang, R.: Fast string kernels using inexact matching for protein sequences. Journal of Machine Learning Research **5** (2004) 1435–1455
[13] Rousu, J., Shawe-Taylor, J.: Efficient computation of gapped substring kernels for large alphabets. Journal of Machine Leaning Research **6** (2005) 1323–1344
[14] Schölkopf, B.: The kernel trick for distances. In Leen, T., Diettrich, T., Tresp, V., eds.: Advances in Neural Information Processing Systems 13, MIT Press (2001)
[15] Jaccard, P.: Contribution au problème de l'immigration post-glaciaire de la flore alpine. Bulletin de la Société Vaudoise des Sciences Naturelles **36** (1900) 87–130
[16] Anderberg, M.: Cluster Analysis for Applications. Academic Press, Inc., New York, NY, USA (1973)
[17] Knuth, D.: The art of computer programming. Volume 3. Addison-Wesley (1973)
[18] Shawe-Taylor, J., Cristianini, N.: Kernel methods for pattern analysis. Cambridge University Press (2004)
[19] Sonnenburg, S., Zien, A., Rätsch, G.: ARTS: Accurate recognition of transcription starts in human. Bioinformatics (2006) submitted.
[20] Lippmann, R., Haines, J., Fried, D., Korba, J., Das, K.: The 1999 DARPA off-line intrusion detection evaluation. Computer Networks **34**(4) (2000) 579–595
[21] Lewis, D.D.: Reuters-21578 text categorization test collection. AT&T Labs Research (1997)
[22] Rieck, K., Laskov, P.: Detecting unknown network attacks using language models. In: Proc. DIMVA. (2006) to appear.