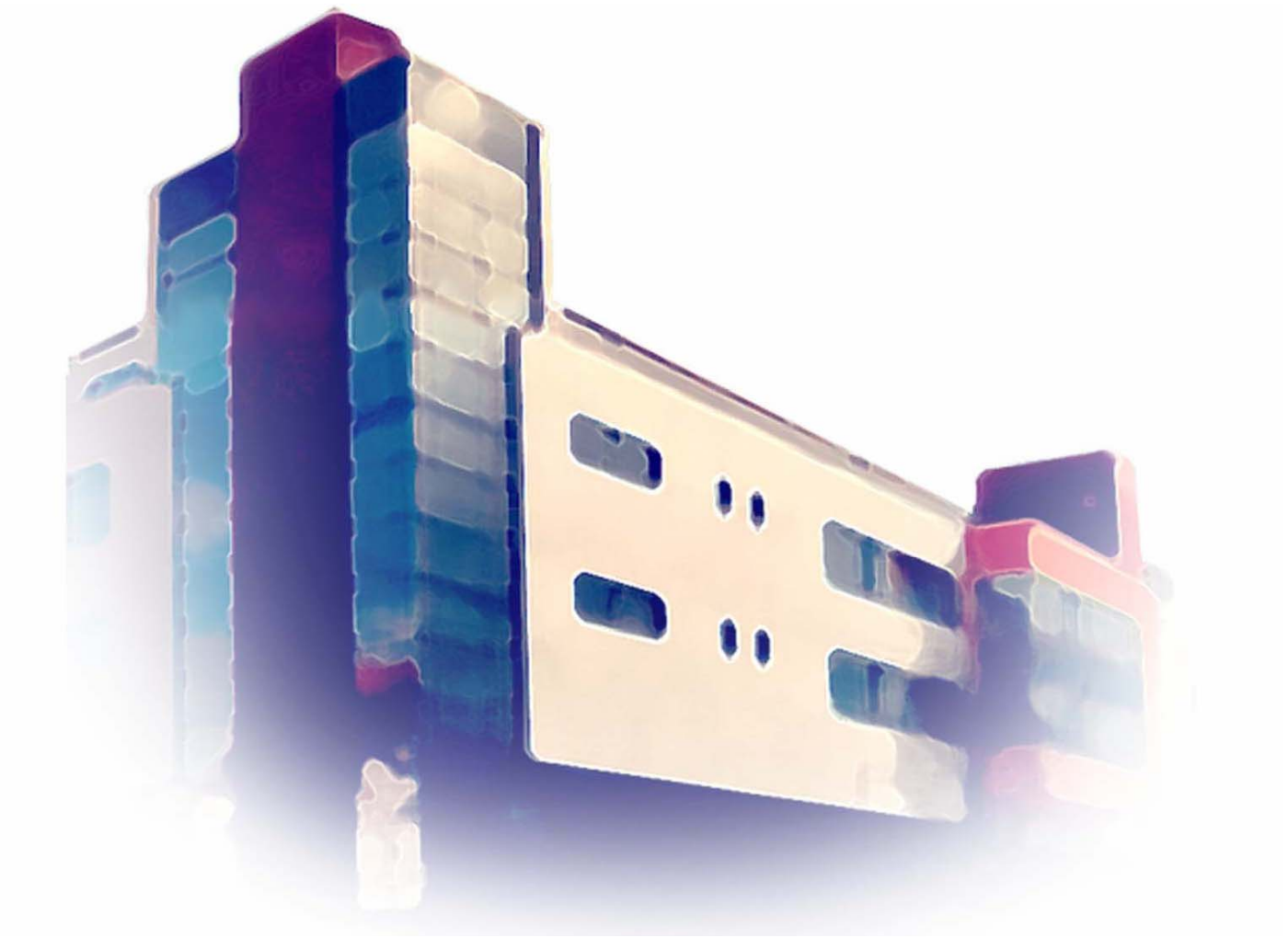
Christine Carl

Kernels for Structures



PICS

Publications of the Institute of Cognitive Science

Volume 9-2004

ISSN: 1610-5389

Series title: PICS
Publications of the Institute of Cognitive Science

Volume: 9-2004

Place of publication: Osnabrück, Germany

Date: October 2004

Editors: Kai-Uwe Kühnberger
Peter König
Petra Ludewig

Cover design: Thorsten Hinrichs

Bachelor's Thesis

Kernels for Structures

Christine Carl

Cognitive Science
University of Osnabrück
ccarl@uos.de
January 2004

Supervisors:
Dr. Barbara Hammer, University of Osnabrück
Prof. Dr. Volker Sperschneider, University of Osnabrück

Abstract

This work presents kernel functions for the classification problem using Support Vector Machines. Several approaches to the application of kernel techniques for a machine learning classification task exist. Most of them, however, require a vector space as data input space and are consequently inappropriate if dealing with discrete structures like strings. Therefore, kernels that are especially designed for handling strings as input data will be the focus of this thesis. Different string kernels proposed so far in the literature will be compared. A general subsuming notion of the building characteristics of string kernels based on substructures will be developed and, with the help of this general formulation, the effects of varying these kernels will be examined. These effects concern the computational efficiency and the notion of similarity the different kernels implement, as well as the experimental results in practical classification tasks in the field of text categorization and protein homology detection.

Contents

1 Introduction

During the past decade there has been an explosion in development of computation and information technology. With it have come vast amounts of data in a variety of fields such as medicine, biology, or business. This data can be highly complex, inconsistent, noisy and redundant, which makes it difficult to evaluate and understand the information collected. Data mining tries to find a response to the challenge of handling huge amounts of information by reducing data and retrieving the high level conceptual information of greatest interest. One important problem within this context is the automatic classification of data into given classes.

Let us consider a typical classification task to motivate the use of classifiers: the classification of given proteins as homologues. As a result of the Human Genome Project and related efforts, DNA, RNA and protein data accumulate at an accelerating rate. Mining these biological data to extract useful knowledge is therefore essential. One way to describe and understand the functionality of a protein is to detect similar (i.e. homologous) proteins with already known properties by comparing their primary structures, i.e. their sequences of amino acids. This problem can be formulated as the classification problem to map two given sequences of amino acids to one of the two classes : 1, if the proteins are homologous and -1 otherwise.
Another typical classification problem we will be referring to throughout this work is the area of text categorization, which is becoming increasingly important, e.g. due to the world wide web.

As a consequence, it is necessary to develop fast and reliable automated classification methods for these application areas. In this thesis, we will focus on discriminative approaches, which can be informally described as follows: Given a set of labelled examples, classification methods try to build a model of the data as a function of attributes of the training data set. The labels refer to target categories that should be learned by the classifier. With the help of the learned model, it should be possible to classify new, unseen data by defining whether the data belong to one of the target classes. There exists a variety of discriminative machine learning algorithms for classification problems like decision trees, neural networks, prototype based classifiers, linear classifiers or the Support Vector Machine. In this thesis, we will deal with the Support Vector Machine as one of the most successful machine learning tools. The Support Vector Machine basically consists of two parts: a simple linear classifier and a fixed nonlinear preprocessing for the input data using a so-called kernel function.
Linear classifiers, in their original form, can only classify data that is linearly separable. Vividly, this means that it is possible to classify all data input as target or non-target by laying a hyperplane through the data that separates

target from non-targets. Often the input data, that is to be classified, cannot be linearly separated, in this case the Support Vector Machine makes use of the "kernel trick": Data is made linearly separable by first mapping the data into a high-dimensional feature space. Since this mapping can be computationally very expensive it can be done implicitly via the help of kernel functions. A kernel function can be interpreted as a form of similarity measure. Thus, it forms a natural interface to integrate prior knowledge about the data and to extend the given learning method to complex data types. The kernel functions will be the topic of interest of this thesis.

Kernel functions are a well known method used widely in the machine learning sector, not only for classification tasks -the context in which we will get to know them- but also for regression, preprocessing, information extraction, etc. Most of these kernels are designed for data structures that can be represented in a vector space. As a consequence, kernel methods, like most pattern recognition tools, are mainly used in application areas with simple data structures like in the case of picture processing. For some data, e.g. sequences of amino acids of proteins or strings of characters in a text, these kernel functions cannot be applied directly. Even though some preprocessing techniques, like the bag-of-words approach for text categorization, exist that enable us to work in a vector space representation of the data, these techniques are computationally expensive and they are usually accompanied by loss of information. For example, the bag-of-words representation neglects the information regarding the word position in the text by representing documents only through word frequencies (with possibly additional weighting or normalization). Therefore, a new class of kernel functions has recently been developed: kernels which can directly deal with discrete structures of possibly arbitrary size. In this work, we will focus on string kernel approaches that can handle input data represented as a sequence of symbols. We will gain an insight into the possibilities such structure based kernels offer for classification tasks by examining the different string kernel approaches in detail.

The aim of this work is to provide a self-contained representation, unification and discussion of an important subclass of kernels for strings, namely kernels based on substructures. As an example, two of these approaches will be presented in detail and illustrated by examples. Several questions of further interest, such as the efficient computation of these kernels, will also be tackled. Starting from these two kernel approaches, we will propose a unified formulation for string kernels based on substructures which allows a better comparison of the different existing methods. This will finally lead us to define several criteria for a taxonomy of this class of string kernels. Thereby, this work does not only incorporate and summarize an extremely important and topical problem of the current literature in the machine learn-

ing sector, but it also outlines several criteria that provide guidelines for the application of the various approaches.

Before introducing different types of kernels used for discrete structures in section 4, we will give a short introduction to Support Vector Machines as a kernel based machine learning method and explain why kernel functions are extremely useful. Section 3 describes kernel functions in general, whereas section 4 explains the specialities of different classes of kernel functions for structures. Section 5 introduces some general definitions useful in the context of string kernels. Section 6 holds a short overview of the terminology and common methods used in text and protein classification and evaluation of classification experiments. We will shortly present a standard approach for text classification, the bag-of-words approach, in section 7, before examining in detail two string kernel approaches -the spectrum kernel and the subsequence kernel- in section 8 and 9. In section 10 a general subsuming notion of the building characteristics of all string kernels based on substructures is provided to describe the different effects of the variants. In conclusion, we will discuss our results and give suggestions for future research.

2 Support Vector Machines

To understand the function of kernel techniques it is helpful to get a general notion of machine learning methods that make use of such kernels. We will therefore give a short introduction to Support Vector Machines as an efficient kernel based classifier, before dealing with the characteristics of kernel functions themselves. Support Vector Machines (SVMs) are learning devices that expand the idea of perceptrons as linear classifiers to systems that are able to learn nonlinear functions in a kernel-induced, often high-dimensional feature space. That is possible because the generalization performance of a SVM does not depend, like for the perceptron, on the dimension of the input space. Originally developed by Vapnik, SVMs are trained with a learning algorithm from optimization theory which implements a learning bias founded on statistical learning theory [27],[28].
SVMs address mainly two problems: pattern recognition and linear regression. For purpose of introducing kernel functions in the following, this thesis focusses on SVMs as classifiers and more precisely on the maximal (hard) margin approach[1]. We will first describe the linearly separable case, i.e. the training sample is already linearly separable in the original input space.

[1]The soft margin approach is introduced at the end of section 1.

2.1 SVMs in the Linearly Separable Case

Suppose a training sample S consists of l labelled input vectors (x_i, y_i), $i = 1, ..., l$ with $x_i \in \mathbb{R}^n$ and $y_i \in \{\pm 1\}$. In this case, x_i denotes an instance of the training input data, while y_i indicate whether this instance is a target of the concept to be learned ($y_i = 1$) or not ($y_i = -1$). The linear classification rule f is defined via

$$f : \mathbb{R}^n \to \{-1; +1\}; \quad x \to \langle w, x \rangle + b$$

with $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Our aim is to find a linear decision boundary defined by the parameters w and b that classifies the training data x upon its position relative to the boundary, i.e. $f(x) > 0$ ($f(x) < 0$) if $y_i = 1$ ($y_i = -1$). The resulting hyperplane $\langle w, x \rangle + b = 0$ with normal vector w and bias b should now be chosen in such a way to be as reliable on new test data as possible. By scaling of the weight vector w we can resume the above considerations to

$$y_i \langle w, x_i \rangle + b - 1 \geq 0 \quad \forall i. \tag{1}$$

The hard margin approach tries to achieve good generalization by specifying the hyperplane that maximizes the geometrical margin, i.e. the minimal distance between training data and the decision boundary has to be maximized with respect to condition (1):

$$\max_{w,b} \min_{x_i \in S} \frac{y_i \langle w, x_i \rangle + b}{\|w\|}.$$

The vectors x_i for which the equality holds in condition (1) have a minimal distance to the hyperplane of $\frac{1}{\|w\|}$. They are called support vectors and define the margin of the hyperplane ($m_S(f)$) by $m_S(f) = \frac{2}{\|w\|}$. Therefore, the hyperplane that maximizes the geometrical margin can be calculated by minimizing $\frac{1}{2}\langle w, w \rangle$ subject to condition (1). The Lagrangian function of this optimization problem is

$$L(w, b, \alpha) = \frac{1}{2}\langle w, w \rangle - \sum_{i=1}^{l} \alpha_i [y_i(\langle w, x_i \rangle + b) - 1] \quad \forall \alpha_i \geq 0. \tag{2}$$

According to the Kuhn-Tucker-Conditions[2], a convex function f with the convex conditions $y_i(w_i x_i + b) - 1 \geq 0$ has one global minimum in w* and b*, if the complementary condition $\alpha_i[y_i(w^* x + b*) - 1] = 0$ holds. By differentiating with respect to w and b, imposing stationarity, we get:

$$\frac{\partial L(w, b, \alpha)}{\partial w} = w - \sum_{i=1}^{l} y_i \alpha_i x_i = 0, \quad \Longrightarrow \quad w = \sum_{i=1}^{l} y_i \alpha_i x_i,$$

[2]See N. Cristianini and J. Shawe-Taylor [5] (p.87) and R. Fletcher [6].

$$\frac{\partial L(w,b,\alpha)}{\partial b} = \sum_{i=1}^{l} y_i\alpha_i = 0 \quad \Longrightarrow \quad 0 = \sum_{i=1}^{l} y_i\alpha_i.$$

The re-substitution of this relations into the primal Lagrangian (equation 2) provides the following dual problem:

$$\text{Maximize} \quad \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle, \tag{3}$$

$$\text{subject to} \quad \sum_{i=1}^{l} y_i\alpha_i = 0, \quad \alpha_i \geq 0 \quad \forall i. \tag{4}$$

The optimal solutions a^* for this quadratic programming problem can be found with one of the common procedures of optimization theory. For a survey on convex optimization for SVMs see Burges [2] and Smola and Schölkopf [23]. If a solution of the learning problem exists, the complementary condition of the Kuhn-Tucker-Conditions causes the optimal weight vector w^* to be a linear combination of the support vectors:

$$w^* = \sum_{i=1}^{l} y_i\alpha_i x_i = \sum_{i \in sv} y_i\alpha_i x_i,$$

where sv is the set of indices of the support vectors. The optimal value for the bias b^* has to be calculated via the primal constraints:

$$b^* = -\frac{\max_{y_i=-1}(\langle w^*, x_i \rangle) + \min_{y_i=1}(\langle w^*, x_i \rangle)}{2}$$

The optimal hyperplane can now be expressed in the dual representation:

$$f(x, \alpha^*, b^*) = \sum_{i=1}^{l} y_i\alpha_i^* \langle x_i, x \rangle + b^* = \sum_{i \in sv} y_i\alpha_i^* \langle x_i, x \rangle + b^* \tag{5}$$

2.2 Mapping of the Input Data into a High-Dimensional Feature Space in the Non-Linearly Separable Case

If the training examples are not linearly separable because the underlying classifying function is not linear, a change of representation of the data by mapping the input space X to a high-dimensional feature space $F = \{\Phi(x) | x \in X\}$[3] can make originally linearly not separable data linearly separable. Changing the representation of the data is a common preprocessing strategy in machine learning. Thereby, the image of x lives in a high-dimensional feature space, but it is simply a contorted version of the original input whose intrinsic dimension is that of the low dimensional input space (for an example see figure 1).

[3] See section 3 *Kernel Functions* for an example.

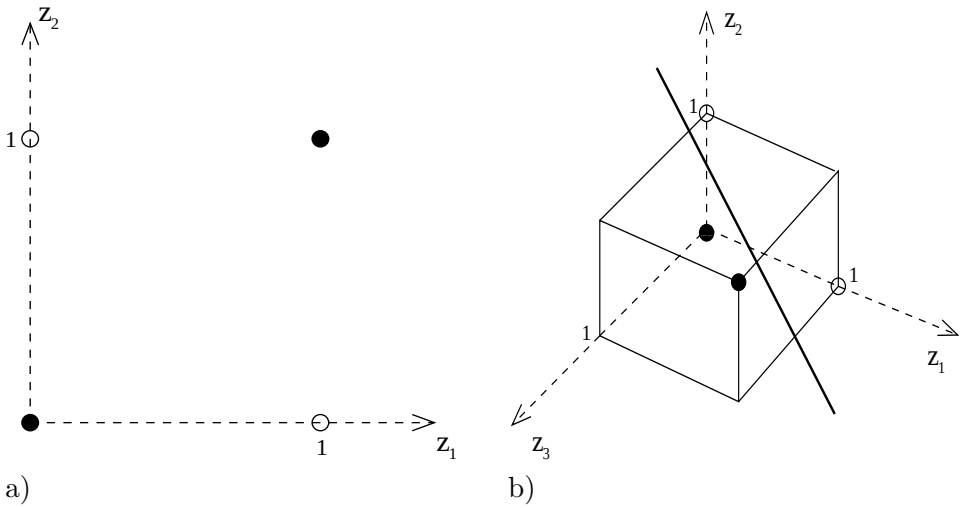


Figure 1: Let us consider the XOR classification problem to illustrate the benefits of a mapping of the input data into a high-dimensional feature space. The XOR-problem refers to the representation of the Exclusive-OR connection, that is usually represented by $\succ\prec$ in propositional logic. An Exclusive-OR connection between two formulas C and D holds, if exactly one of the two formulas is true (=1) and the other one is false (=0). If both sentences are true (1,1) or both are false (0,0) the Exclusive-OR connection does not hold. For instance, the sentence -*You will either love chocolate or you hate it, but you can't do both.*- represents an XOR-relation. Let the two-dimensional vectors $(0,0);(1,1);(0,1);(1,0)$ represent the four possible relations between two formulas C and D. C and D can be both true (1,1), both false (0,0) or the XOR-relation holds: (1,0); (0,1). The classifier has to differentiate between the positive instances $(1,0)$ and $(0,1)$ and the non-targets $(1,1)$ and $(0,0)$. As a) shows, it is not possible to define a hyperplane that could correctly separate the data in a two-dimensional vector space representation of the data. (Try to draw a straight line into the graphic that separates the black points from the transparent ones.) However, if we map each vector $z = (z_1, z_2)$, $z \in X$ from the input space $X = \mathbb{R}^2$ into the three-dimensional feature space $F = \mathbb{R}^3$ by the mapping
$\Phi : X \to F; \Phi(z) = (z_1, z_2, z_1 z_2)$, we can lay a hyperplane through the data points that classifies the instances correctly, as shown in figure b). Here, the observer is looking at the graphic as if he is in the hyperplane represented by the continuous, thick straight line. (Note that the indices of z refer to the components of the vector z in this case.)

The decision boundary has now to be reformulated:

$$f(\Phi(x), \alpha^*, b^*) = \sum_{i \in sv} y_i \alpha_i^* \langle \Phi(x_i), \Phi(x) \rangle + b^* \tag{6}$$

As equation 6 shows, a new data point can be classified by simply calculating the inner product of two feature vectors $\Phi(x)$. The inner product requires the feature space to define any inner product. Therefore, the feature space is a form of generalization of the Euclidian space, that is referred to as Hilbert space in the literature [5]. In this work the scalar dot product is usually used as example.
It should be noticed that the mapping of input data into a high-dimensional feature space causes two problems: Firstly, the computation of the inner product in the high-dimensional feature space becomes inefficient. This problem can be dealt with by using kernel functions as introduced in section 3. The second problem, the generalization ability of a machine learning tool in case of high-dimensional input data, will be tackled in the following.

2.3 Generalization Error of SVMs

The high dimension of the feature space the SVM works on poses the question why this does not impoverish the generalization ability of the SVM as it would normally do, e.g. for perceptrons. The number of free parameters of both, the perceptron and the SVM, equals the dimension of the feature space plus 1 $(n + 1)$. These parameters are determined during training. Both, the SSV and the perceptron generate similar solutions, namely hyperplanes that separate positive from negative examples. Because the SVM searches for the hyperplane with maximum margin, its generalization ability is better than the one of the perceptron. This becomes clearer if we compare the generalization errors of both classifiers:

Theorem 1 (generalization error of a perceptron) *Suppose H is the hypothesis space of a perceptron and n is the dimension of the input space. For any probability distribution P on the input space $X \times \{-1, 1\}$, any hypothesis $f \in H$ that is consistent with the l training examples S will have with probability $1 - \delta$ error no more than:*

$$\varepsilon \leqslant \varepsilon(l, H, \delta) = \frac{2}{l}\left((n+1)\log\frac{2el}{n+1} + \log\frac{2}{\delta}\right)$$

provided $(n+1) \leqslant l$ and $l > \frac{2}{\varepsilon}$ (see N.Cristianini and J. Shawe-Taylor, p.56, [5]).

Thus, the generalization error -given a confidence of $1 - \delta$ and l training examples- is roughly linearly increasing with the dimension of the input space n. The generalization error of a SVM on the other hand, depends

only on the maximal margin independently of the dimension of the input space:

Theorem 2 (generalization error of a SVM) *Let H be thresholding real-valued linear functions with unit weight vectors on the inner product space X and fix $\gamma \in \mathbb{R}_+^*$. For any probability distribution P on $X \times \{-1, 1\}$ with support in a ball of radius R around the origin, for a probability $1 - \delta$ over l random examples S, any hypothesis $f \in H$ that has a margin $m_S(f) \geqslant \gamma$ on S has error no more than :*

$$\varepsilon \leqslant \varepsilon(l, H, \delta, \gamma) = \frac{2}{l}\left(\frac{64R^2}{\gamma^2}\log\frac{el\gamma}{8R^2}\log\frac{32l}{\gamma^2} + log\frac{4}{\delta}\right),$$

provided $l > \frac{2}{\varepsilon}$ and $\frac{64R^2}{\gamma^2} < l$ (N.Cristianini and J. Shawe-Taylor, p.63, [5]).

The generalization error of the SVM depends above all on the number of the training examples, (it decreases approximately linearly with the increase of examples) and the proportion of $\frac{R}{\gamma}$, but it is dimension free and therefore allows high-dimensional vector spaces without deteriorating the generalization ability[4].

2.4 The Soft Margin Approach

The criteria of maximizing the geometrical margin while developing a consistent learning algorithm assumes that the training sets are linearly separable, at least after mapping into a high-dimensional feature space. This is, for instance, not the case if the training data is very noisy, i.e. if it contains a lot of incorrectly labelled training examples. Some generalizing extensions of the above presented SVM allow the consideration of noise in the training data. These soft margin approaches seek at a trade-off between maximal geometrical margin and minimal empirical error on the training set. For a more precise formulation on soft margin approaches see (Cristianini [5]).

Since an explicit mapping $\Phi(x)$ into a high-dimensional feature space and the calculation of its inner product is complicated and computationally expensive, one would like to introduce an alternative technique that circumvents these drawbacks. Kernel functions offer such an alternative and will be discussed in the following section.

[4]For a detailed examination of this problem see [5].

3 Kernel Functions

The introduction of kernel functions provides a tremendous computational advantage of SVMs for high-dimensional feature spaces so that a lot of effort is done to develop and examine these functions.

3.1 What are the Characteristics of a Kernel Function?

Definition 1

$$K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

is a kernel function if there exists a Hilbert space H and a mapping

$$\Phi : \mathbb{R}^n \to H; \quad x \to \Phi(x)$$

such that for all $x, z \in \mathbb{R}^n$ holds:

$$K(x, z) = \langle \Phi(x), \Phi(z) \rangle.$$

Hereby, $\langle \cdot, \cdot \rangle$ denotes the dot product in the Hilbert space H. Note that in this section we will refer to a vector x by $x = (x_1, ..., x_n)$ with $x \in \mathbb{R}^n$.

The dual formulation of the learning problem of SVMs depends only on the inner products $\langle x_i, x \rangle$ (see equation 3). Therefore, the input vectors x never appear isolated and the mapping $\Phi(x)$ can be done implicitly with the help of kernel techniques. If we can define a function $K(x, z)$ that replaces the inner product of the feature vectors, we can avoid the explicit calculation of the inner product in a high-dimensional feature space. But how do we know if the function $K(x, z)$ satisfies the criteria of a kernel function as introduced in the definition above? A sufficient condition of the function K to be a kernel function is defined by the *Mercer's theorem*:
If $K : [a, b]^n \times [a, b]^n \to \mathbb{R}$ is a symmetric and continuous function than K is a kernel function, if

$$\forall \Psi \quad \Psi : [a, b]^n \to \mathbb{R}; \quad \int_{[a,b]^n} \Psi(x)^2 dx < \infty$$

it holds

$$\int_{[a,b]^n} \int_{[a,b]^n} \Psi(x) K(x, z) \Psi(z) dx dz \geq 0.$$

In other words, the Mercer's condition ensures that the prospective kernel is actually a dot product in some space. In the following, some examples of standard kernels used for vector spaces as input spaces will give the reader an idea of how kernel functions may look like.

3.2 Examples of Widely Used Kernels

3.2.1 The Polynomial Kernels

Polynomial kernels are kernels of the form

$$K_p(x, z) = \langle x, z\rangle^p \qquad \text{for } x, z \in \mathbb{R}^n.$$

Suppose we choose the input data in $\mathbb{R}^2$ and $K(x, z) = \langle x, z\rangle^2$. A feature space F and a feature mapping Φ for this kernel could be, for example:

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3 \qquad \Phi(x) = \Phi(x_1, x_2) = (x_1^2, x_2x_1, x_2^2).$$

Hereby, the indices denote the components of the vector x. Note that neither the feature space F nor the mapping Φ is unique for a given kernel. The above one could equally well be based on the mapping:

$$\Phi : \mathbb{R}^2 \to \mathbb{R}^4; \qquad \Phi(x_1, x_2) = (x_1^2, x_1x_2, x_2x_1, x_2^2).$$

More generally, the feature mapping for polynomial kernels is the set of all products of p (in this example $p = 2$) factors that can be built out of the components in the input vector, whereby each product can be weighted. A component of a vector in the feature space can also be a sum of such products.
More formally, let us denote $w_j = x_jz_j$, and $K(x, z) = (w_1 + ... + w_n)^p$. We can explicitly define the mapping for any p and n:

$$\Phi(x) = (\phi_{r_1r_2...r_n}(x))_{r_1r_2...r_n}, \text{ whereby } \sum_{j=1}^{n} r_j = p; \qquad r_j \geq 0.$$

$$\phi_{r_1r_2...r_n}(x) = \sqrt{\frac{p!}{r_1!r_2!...r_n!}} x_1^{r_1}x_2^{r_2}...x_n^{r_n};$$

Since in this case the feature mapping is known explicitly, the proof that $K(x, z)$ is a kernel function can be done directly by calculating the inner product for the feature vectors. We will prove that $K_p(x, z) = \langle x, z\rangle^p$ is a kernel by showing that the Mercer's condition holds:

$$\int (\sum_{j=1}^{n} x_jz_j)^p \quad \Psi(x)\Psi(z)dxdz \geqslant 0, \tag{7}$$

since the multinomial expansion of $(\sum_{j=1}^{n} x_jz_j)^p$ contributes to the factorization of the left hand side of equation 7:

$$\int (\sum_{j=1}^{n} x_jz_j)^p \quad \Psi(x)\Psi(z)dxdz$$

$$= \frac{p!}{r_1! r_2! ... (p - r_1 - r_2 ...)!} (\int x_1^{r_1} x_2^{r_2} ... \Psi(x) dx)^2 \geqslant 0.$$

Every homogeneous polynomial kernel of degree p has an underlying minimal embedding feature space of $\binom{n+p-1}{p}$, if n is the dimension of the input space[5]. If we would like to modify the homogeneous polynomial kernel to get all polynomials up to degree p in one kernel, we would get the inhomogeneous kernel $K(x, z) = (1 + \langle x, z \rangle)^p$.

3.2.2 The Gaussian Kernels

Other often used kernel functions are the Gauss-kernels, also called Radial Basis Functions:

$$K_\sigma(x, z) = \exp^{-\frac{\|x-z\|^2}{2\sigma^2}} \qquad \text{with } \sigma \in \mathbb{R} \text{ as variance.}$$

The embedding space of these kernels is infinite dimensional [2].

4 Kernels for Structures

Kernels like the Gaussian kernels require the input space of the data to be a vector space. Since kernel based methods like the SVM are efficient and precise classification devices, it would be desirable to use such methods for classification tasks like text categorization or protein homology detection. However, these problems deal with discrete structures like sequences of characters and therefore, the underlying input space X is not a vector space, but a set of structures. The kernels presented in the following are able to deal with these discrete structures, which can be word sequences and other string-based structures, trees, graphs, as well as finite state machines etc. As long as any kernel based approach succeeds to extract real-valued features $\Phi(x)$, $\Phi(z)$ from the structures x and z in X and transform the input space into a vector space F, kernel based methods like SVMs can deal with the data. One can roughly distinguish four main directions of kernels for discrete structures:

- principle design criteria of kernels and mathematical closure properties
- problem specific kernels with fixed structures
- structure based kernels of variable size
- kernels based on probabilistic models.

While the third one will be further examined in detail throughout this work, all approaches will be outlined in this section.

[5] For a proof see C. J. C. Burges [2]

4.1 General Mathematical Design Criteria of Kernels

General mathematical design criteria of kernels for discrete structures and their closure properties were established by D. Haussler [8] and C. Watkins [31]. These characteristics form a basis for structure based kernels and some kernels with an underlying probabilistic model. One important issue Haussler and Watkins stated is, for example, that kernel functions can be defined over general sets by assigning an inner product to each pair of elements of the discrete structures in the feature space.

4.2 Kernels for Fixed Structures

These type of kernels allow only structures of a fixed size. Thus, structures can be represented as finite dimensional vectors and they can be compared in the standard way by referring to the elements at fixed positions. In addition, specific features according to the considered structures can be added. The approaches of S. Sonnenburg et al. [24] and A. Zien et al. [32] take local correlations into account by allowing neighboring elements to contribute to the elementwise match of the symbols of the two structures. Another kernel for fixed structures, the P-kernel based on rare common substrings [29], determines the closeness of two structures on the basis of common substrings. This kernel compares subsets at the respective positional entries and, in case of matching, the common substrings can be weighted according to the inverse proportional of their probability, so that rare common substrings favor a high similarity score between the sequences which are to be compared.

4.3 Discrete Kernels for Arbitrary Sized Structures

As for the P-kernel based on rare common substrings, the extraction of substructures for comparison of two sequences is the main idea of the kernels presented in detail in this work: discrete kernels for structures of arbitrary size. These methods define, in principle, similarity via common occurrences of substructures at arbitrary positions in the original sequences. The finite set of possible substructures considered in the specific kernel forms the indices of the feature vector, calculating the number of times the substructure occurred. The different kernels alter with an optional weighting scheme for different substructures, the efficiency of computation and the types of substructures considered. These substructures of interest can be either only consecutive substrings or substructures where gaps are allowed, substructures that require perfect or partial match or substructures weighted e.g. according to their probability density. In this thesis, we will explain two such kernels in detail and give an overview and comparison of several alternatives, whereby we will introduce a unified notation.

4.4 Kernels Based on Probabilistic Models

This approach starts from a probabilistic model of the data, where prior information about the data is usually required. The kernel combines the power of information extraction through statistical modelling methods with the efficiency of a classifier like the SVM. The data can be compared either by using parameters of the probabilistic model or by drawing up the classifier directly from the probabilistic model itself. Examples for such kernels include the TOP kernel as introduced in [24] and the Fisher kernel proposed in [12], [13]. The latter one constitutes a popular state-of-the-art approach used for various problems in computational biology and data mining.

5 Definitions and Notations

Before examining the characteristics of structure based kernels in detail, let us begin with introducing some notation we will use in the following, if not indicated otherwise:

Definition 2 (Strings) *Let Σ be a finite alphabet. Any finite sequence $x = x_1x_2...x_m$ of m characters from $\Sigma, (x_i \in \Sigma, 1 \leq i \leq m)$, for $m = 0, 1, 2, ...$ is a string, the empty string is represented by ε.*
Let m be the length of a string. Then, the set of all strings of length m is denoted by Σ^m, the set of all strings of arbitrary length by Σ^*:

$$\Sigma^* = \bigcup_{m=0}^{\infty} \Sigma^m.$$

In the following, we will use $a \in \Sigma$ to denote characters and $s, t, u, v, w, x \in \Sigma^*$ to denote strings. $|x|$ will be the length of the string x and xu or xa the concatenation of two strings, respectively a string and a character.

Definition 3 (Contiguous and noncontiguous substrings, suffixes and prefixes)
Given a string $x = uvw$, then u is a prefix of x, v is a substring and w is the suffix of x. If $s = s_1...s_{|s|}$, let $v = s[i : j]$, or $v \sqsubseteq s$ for short, denote the substring $s_i...s_j$ of s. A sequence u is a possibly noncontiguous subsequence of s, if there exist indices $i = (i_1, ..., i_{|u|})$, with $1 \leq i_1 < ... < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$, for $j = 1, ..., |u|$, or $u = s[i]$ for short. The length $l(i)$ of the window in s spanned by the subsequence u is $i_{|u|} - i_1 + 1$ (for an example see figure 2).

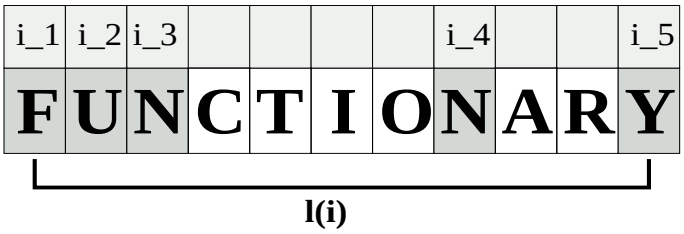


Figure 2: The string *functionary* contains the substring *funny*. While the contiguous string *funny* has the length $|u| = 5$, the length of the noncontiguous substring is $l(i) = 11$.

6 Some Issues about the Application of Kernels for Classification

As mentioned beforehand, most of the structure based kernel approaches have been experimentally evaluated within two main application areas: text categorization and protein classification. The following sections will refer to some experimental results of several structure based kernels. For convenience, we will shortly introduce various common methods and terminology used within this context.

6.1 Text Classification

Reuters Dataset : This data set was edited by David Lewis (1987) and contains stories from Reuters new agency. The database contains predefined splits for training and test sets, which are to be classified according to different topics the documents are about. The split used in the approaches presented was the 'ModeApte' split, which comprises 9603 training and 3299 test documents. Furthermore, it should be mentioned that a Reuters category can contain as few as 1 document for training respectively testing and up to as many as 2877 in the training and 1066 documents in the test set.

F1-test : To evaluate the performance of a classifier two different aspects have to be considered: Firstly, we would like a classifier to assign *only* positive examples as targets and secondly, a classifier should recognize *all* positive examples as targets. The F1-test is a performance measure that considers both aspects, called precision and recall. Before referring to the F1-test we would like to have a closer look at these two influences on performance:

A classification of one single instance can have four different outputs: **true positive (TP):** The instance was classified correctly as a target.

false positive (FP): The item was assigned as positive example without being one actually.
false negative (FN): A target that was not found by the classifier.
true negative (TN): A non-target that was correctly assigned as a negative example.

We can now define precision and recall ($n(y)$ being the number of instances that were classified with the output specified in y):
Precision measures the portion of the assigned categories that were correct:

$$p := \frac{n(TP)}{n(TP) + n(FP)}.$$

Recall measures the portion of the correct categories that were assigned:

$$r := \frac{n(TP)}{n(TP) + n(FN)}.$$

Naturally, if a classifier performed perfectly, it would have a precision and recall of the maximal value 1, i.e. it would assign the correct categories and only the correct categories. Since this is often not the case, we need performance measures for classifiers that take both, precision and recall, into account.

The F1-test is a performance measure that evaluates a classification by equally weighting precision and recall. It is defined as

$$F1 = \frac{2pr}{(p + r)}$$

whereby p is precision and r recall. The F1-score falls in a range from 0 to 1, with 1 being the best score.

The F1 measure constitutes a single measure that is worth trying to maximize on its own - considering the fact that one can get a perfect precision score by always assigning zero categories[6] or a perfect recall score by always assigning every category.

6.2 Protein Classification

Remote Homology Detection refers to the problem of detecting homology between proteins in cases of low sequence similarity in the primary structure, i.e. in the sequence of amino acids of the proteins. Two sequences are said to be homologous if they share a common evolutionary ancestor. Since we do not have access to ancestral protein sequences, the homology detection task is necessarily inferential. It

[6]Hereby, we define $\frac{0}{0} := 1$ for the precision score, if zero categories are assigned.

consists of classifying a given protein or protein family as a member of a superfamily, i.e. all distantly related protein sequences in a large unannotated database have to be found.

SCOP Database : The Structural Classification of Proteins (SCOP) [11] database provides a description of the relationships of known proteins structures in detail. The classification is based on hierarchical levels: the first two levels, family and superfamily, describe near and far evolutionary relationships on the basis of the percentage residues are identical between the proteins, as well as further structural and functional similarities. The third level describes geometrical relationships, i.e. similarities in the major secondary structures. For the remote homology detection task as they are formulated for the kernels presented in this thesis only the first two levels are relevant, since a classification of proteins by the kernels presented here is based on the primary structure of the proteins. The SCOP data base is used for evaluation of the homology detection performance of a kernel: Usually a target SCOP family is separated from the remaining families of a given superfamily. While these remaining families serve as positive training examples, negative training examples are chosen from outside the superfamily's fold. The proteins of the hold out family provide positive test examples. Each kernel based classification of a test example is considered as correct if it recognizes this target protein as a member of the given superfamily. The SCOP database is publicly available at http://scop.mrc-lmb.cam.ac.uk/scop.

$\boldsymbol{ROC_{50}}$: With the help of ROC_{50} scores the performance of different classification methods can be compared. The ROC_{50} score is the area under the receiver operating characteristic curve, a curve where the fraction of true positives in the classification result is plotted against the fraction of false positives. Because it is hard to compare two ROC-curves, the area under the ROC-curve is used as a summary of the ROC-curve in a single number. The index 50 indicates that the area under the ROC-curve up to the first 50 false positives is considered. This score yields higher values for a better separation of positives from negatives, the value of 1 indicates perfect separation while a value of 0 would result, if none of the first 50 sequences selected as targets by the algorithm would be a positive example.

7 Standard Bag-of-Words Kernel

Suppose we would like to determine the semantic similarity between the following two sentences:

1. *The girl could not decide which ketchup to buy for the barbecue -the yellow, green or pink-white striped one.*
2. *If she should buy the green, the pink-white striped or the yellow ketchup for the barbecue, the girl was unable to decide.*

It is easy to observe that both expressions have nearly the same semantics, even though their appearance is different because of variations in the choice of words or the order of the expressions. Often, the semantics of a text document can already be captured by some key words (like in this case *barbecue, ketchup,* etc.), while other repeatedly occurring words like articles or prepositions are ignored, a fact that is often exploited by search engines in the internet. These observations lead to a standard kernel approach for text classification: the bag-of-words kernel. This kernel, which is based on standard text representation techniques of G. Salton et al. [19], was introduced by T. Joachims [14]. The standard word kernel measures the similarity of structures by calculating the occurrences of common words. It is therefore especially designed for text categorization tasks.
The kernel underlies the simple assumption that two documents are similar if they share common words while the word order in the structures can be ignored for text classification. The feature vectors are sparse vectors that are indexed by all possible words of the alphabet of words W, vector entries define the number of times a word occurred in the document. Mostly, non-informative words, called stop words, and punctuation marks are removed because of their little significance for the content of the document. To support the different significance of words, several weighting schemes can be introduced, the most common being a $tfidf$ weighting scheme; let tf be the word frequency of a word in a document while n represents the total number of documents in the training database and df the number of documents in the database where the relevant word occurs, then the term frequency tf is substituted by $log(1 + tf) * log(n/df)$ for each vector entry. The feature mapping

$$\Phi : X \to F, s \to \Phi(s)$$

$$(X \text{ being the set of all documents}),$$

whereby the u-coordinate $\phi_u(s)$ for each $u \in W$ is given by

$$\phi_u(s) = log(1 + tf) * log(n/df),$$

takes into consideration that very common words are assigned little importance for similarity judgements, while rare words are given special attention.

8 Spectrum Kernel

In the case of text classification, declinations and conjugations or colloquial language let the same word appear in many different forms. Without any preprocessing methods, like stemming algorithms, the word kernel would probably perform poorly. Consider the famous sentence from the movie Casablanca: *Here's lookin' at you kid.* If we changed the colloquial expression into written English: *Here is looking at you kid.* a kernel that could only deal with exact word matches could not catch the semantic similarity of the sentences. As long as exact matches of words are required as document similarity measure, the change of word forms, e.g. because of declinations or conjugations, is a problem we would encounter frequently. To address that problem, the bag-of-words kernel applies stemming algorithms and other preprocessing algorithms. We can avoid the use of such preprocessing algorithms through an approach that considers subsequences of the texts. For instance, the subsequence *lookin* can still be matched with the word *looking*, even if the last letter is missing in one of the two strings compared by the kernel.
Such an approach based on substructures would furthermore widen the application area of these kernels: A standard word kernel is restricted to linguistic data sets. For biological databases, like in the case of protein homology detection, it is not applicable since word boundaries are missing in structures like DNA sequences. One form of string kernel introduced by C. Leslie et al. in [16], called the spectrum kernel, satisfies these considerations and can be used for any sequence based classification problem: The spectrum kernel is based on a k-spectrum of an input sequence, i.e. it considers the set of all contiguous subsequences of an arbitrary fixed length k contained in the input sequence. The spectrum kernel allows SVM classification in linear time and in a general way without incorporating any prior knowledge about the data.

8.1 Kernel Definition and Computation

Given an input space X of all finite length sequences of characters from an alphabet Σ, and a feature space $\mathbb{R}^{|\Sigma|^k}$ of all permutations u of length k from alphabet Σ, the feature map is defined by a vector indexed by all possible sequences u. Every index stores the number of occurrences of the respective subsequence in the sequence s of the input space:

Definition 4 (Spectrum Kernel)
The feature mapping is defined by:

$$\Phi : X \to \mathbb{R}^{|\Sigma|^k}, \Phi(s) = (\phi_u(s))_{u \in \Sigma^k},$$

where $\phi_u(s)$ is the number of occurrences of u in s. It is clear that this feature map provides a weighting of subsequences of length k according to the number of times they occur in the sequence mapped. The k-spectrum kernel is then defined by the inner product:

$$K_{spectrum}(s,t) = \langle \Phi(s), \Phi(t) \rangle.$$

Since the feature mapping is defined explicitly in definition (4), it does not have to be proved that the Mercer condition holds to verify that the above definition is a kernel.
On the other hand, for an efficient computation of the kernel it would be favorable to avoid the direct computation of feature vectors. Hence, the following observation helps: Despite the large feature space each feature vector is sparse, i.e. the number of non-zero coordinates of the vector for an input sequence s is bounded by $|s| - k + 1$. This sparsity offers the possibility to introduce efficient methods for computing the kernel matrix of the training input sequences without an explicit representation of the feature vectors.
A method of computation that is very easy to implement collects the set of all substructures u actually occurring in the string s respectively t into an array A_s respectively A_t. The arrays are then sorted and double entries are summarized into one entry of the arrays. With the completed arrays the inner product, and hence $K_{spectrum}(s,t)$, can be computed dependent on the length of the input sequences s and t. Since the effort of sorting elements is $O(n \cdot \log(n))$, with n being the length of the documents, the overall complexity of the kernel computation is $O(n \cdot \log(n))$.
A more efficient and advanced method that is widely used for sequence matching problems is the construction of suffix trees. Suffix trees allow to compute the kernel matrix in linear time. Leslie et al. do not provide a detailed description of the suffix tree that is built for an efficient computation in this case. They just mention that the suffix tree is built out of "the collection of k-length subsequences" of s and t, and proclaim that the suffix tree for one input sequence s has $O(kn)$ nodes (whereby n is the length of the input sequence s). The suffix tree stores at the k-depth leaf nodes the number of occurrences of the subsequences in the sequence s, each subsequence is specified by the branch labelling from the root to the leaf. At this point, we would like to go into further detail about the computation of a spectrum kernel via suffix trees, so that the reader will get a notion what a suffix tree looks like and how it is built.

A suffix tree is a data structure that allows us to solve many string processing problems efficiently. For example the question, whether a string u occurs in a string t can be solved in $O(|u|)$ time, even though the whole sequence t of length n has $n(n+1)/2$ substrings. The spectrum kernel faces exactly

this matching problem since it has to be determined how often a substring occurs in two sequences s and t. Having built the suffix tree for a sequence s, it is easy to determine whether the subsequences of another string t match with the ones in s[7].
A suffix tree for a sequence s is built in a way that all subsequences of the sequence s can be accessed most easily by following the appropriate path starting from the root in the search tree. Since many substrings of the string s may share common prefixes, they can share a common path from the root of the suffix tree. Any search (sub)string must therefore be a prefix of a suffix of s, if it occurs in s. (An illustration of two exemplary suffix trees is given in figure 3.)
For a formal definition of a suffix tree according to Giegerich and Kurtz [7] we have first to define an Σ^+-tree:

Definition 5 (Σ^+-tree) *Let Σ^+ denote the set of strings of a finite alphabet without the empty string ε. An Σ^+-tree T is a rooted tree with edge labels from Σ^+. For each character $a \in \Sigma$, every node k in T has at most one a-edge $k \xrightarrow{aw} k'$, with $w \in \Sigma^*$. A string x is occurring in T if there exists a path from the root to the leaf node marked by xu for some possibly empty u. The set of strings occurring in T is referred to as $words(T)$.*

Definition 6 (Suffix tree) *A suffix tree for a string t is a Σ^+-tree T where $words(T) = \{w | w \text{ is a substring of } t\}$.*

There exist different versions of suffix trees, two of them are important for further considerations of this work:

Definition 7 (atomic vs compact suffix trees)
A suffix tree is atomic, if every edge is labelled with a single character.
A suffix tree is compact, if every node is either the root, a leaf node or a branching node.

Thus, a compact suffix tree is a smallest suffix tree possible. For illustrations of an atomic and a compact suffix tree see figure 3.

The suffix tree needed to compute the above defined kernel should have special properties: Since the spectrum kernel only considers subsequences of the predefined length k, it would be most intuitive to build an atomic suffix tree that contains at depth k all subsequences of length k. Since subsequences of further length are of no interest the suffix tree could be cut at depth k, so that the leaves denote the substrings of length k labelled through the branches from root to leaf node. Furthermore, the leave nodes should store the number of times the respective subsequence occurs in the sequence

[7] For a reminder on the linguistic terminology of suffixes and prefixes see definition 3 in section 5.

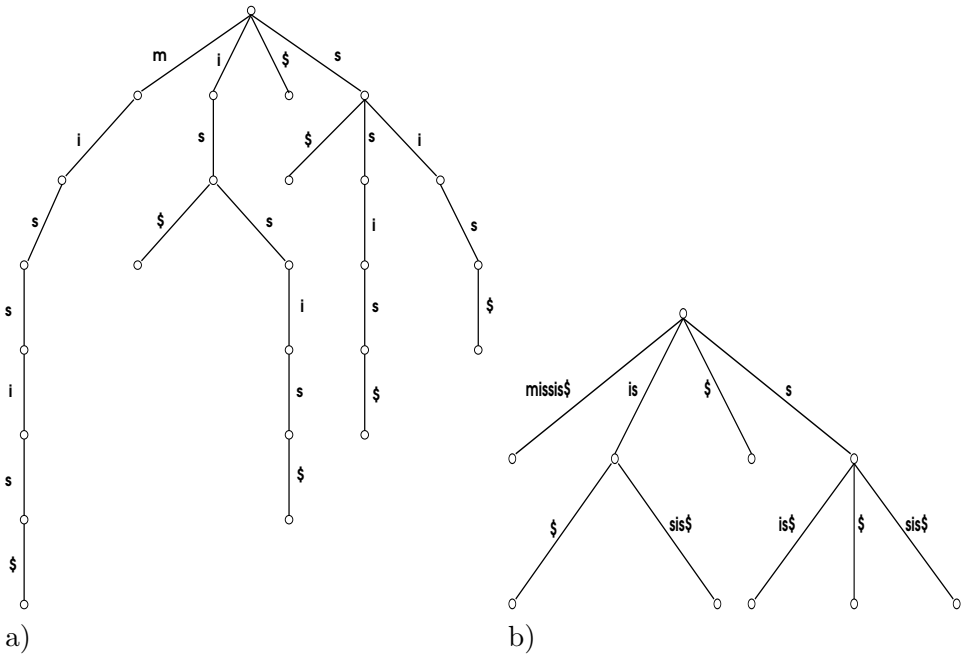


Figure 3: An atomic (a) and a compact (b) suffix tree of the string *missis$*, $ denotes the end of a string.

the suffix tree refers to, because that will be the relevant information for the kernel computation.

Until now, we have assumed that every suffix tree is built out of one input sequence. Now suppose we build one suffix tree out of all input sequences of the training data set. To do so, we have simply to store all k-length subsequences occurring in any one of the input sequences in the suffix tree[8]. Remember, the important information for the kernel computation is just the number of times the substrings of the input sequences occur. Therefore, at every leaf node the single numbers of occurrences of the relevant subsequence in each input string are stored together with a pointer to the appropriate input string.

The kernel can now be easily computed by parsing the tree and building the sum of the products of the counts stored in the leafs of depth k. That means, if we want to compute the kernel value for the strings s and t, we have to calculate the sum of the pairwise products of the two counts in the leaf nodes that point to the sequences they belong to. The great advantage of this procedure is that it is possible to compute the whole kernel matrix with one traversal of the suffix tree, provided the tree is created out of all

[8]How we denote the end of the single sequences remains to be defined, e.g. the suffix tree for all sequences is built by starting at the root for every new sequence or by introducing the sentinel character $ to denote the end of one sequence.

sequences to be compared.

A suffix tree cannot only be parsed in linear time depending on the number of nodes, but also the construction can be done in linear time as a function of the number of nodes of the suffix tree. Weiner, McCreight and Ukkonen presented linear algorithms of suffix tree construction, whereby Ukkonen's algorithm is the most comprehensible one, a left-to-right on-line algorithm that maintains a suffix tree for $t[1..i]$ at each step as i is increased from 1 to n. Since these algorithms are quite complex, we will not go into detail of the construction and refer to Giegerich and Kurtz [7], who compare the above mentioned different algorithms. It must be mentioned here that an atomic suffix tree has $O(n^2)$ nodes, n being the length of the input sequence. Therefore, an efficient computation of the spectrum kernel would have to use a compact suffix tree with $O(n)$ nodes to realize a linear kernel computation with respect to the length of the input sequences. For simplicity of notation, we have presented the atomic tree version, though the use of compact suffix trees in this case is similar. Most important for the reader, however, is to keep in mind that the computation of the spectrum kernel can be done in linear time (depending on the length of the input strings) by the use of a suffix tree.

8.2 Protein Classification with the Spectrum Kernel

The spectrum kernel, as introduced by C. Leslie et al., was first tested for protein classification. We will therefore briefly describe the further procedure of this classification task with SVMs and present the experimental results showing the performance of the spectrum kernel in comparison with other standard kernels.
The parameters of the classifier to be determined are again w and b as in section 2. Assumed the threshold $b = 0$, the classification of test examples can simply be computed by

$$f(x) = \langle \Phi(x) \cdot w \rangle = \sum_{i \in \mathrm{sv}} \alpha_i y_i K(x, x_i),$$

whereby the normal vector w is given by

$$w = \left(\sum_{i \in \mathrm{sv}} \alpha_i y_i \phi_u(x_i) \right)_{u \in \Sigma^k}$$

with sv as the set of indices of the support vectors. In the case of the spectrum kernel the test sequences don't need to be mapped to the high-dimensional feature space, but the product $\langle \Phi(x) \cdot w \rangle$ can be computed directly: The non-zero coefficients of the normal vector w are less than mn entries, m being the number of training examples and n the length of

the largest input sequence, because the number of support vectors is much smaller than the number of all training sequences. Therefore, the non-zero coefficients can be stored in a look-up table with their index u indicating the contributing k-length subsequences. For each k-length subsequence of the test sequence x the classifier value $f(x)$ is incremented by the amount of the associated coefficient in the look-up table. The complexity of the test phase is therefore $O(n)$.

8.3 Experiments

Eventually, we will briefly sketch the experiments that C. Leslie et al. performed and their outcome. The spectrum kernel was tested for a remote homology detection problem over the SCOP database[9] with a soft margin optimization algorithm. C. Leslie et al. used relatively small spectrum kernels with $k = 3$ and $k = 4$ and tested performance with the above introduced unnormalized kernel and a normalized kernel given by

$$K^{\text{Norm}}_{spectrum}(s,t) = \frac{K_{spectrum}(s,t)}{\sqrt{K_{spectrum}(s,s)}\sqrt{K_{spectrum}(t,t)}}.$$

Leslie et al. used the ROC_{50}[10] scores to compare the performance of the spectrum kernel with the Fisher-SVM method (introduced in section 4.4), and two other standard remote protein homology detection methods ; SAM-T98 iterative Hidden Markov Model [15] and PSI-BLAST [1]. Thereby, the normalized spectrum kernel with $k = 3$ showed slightly better results than the other variants of the spectrum kernel. Compared with the other remote homology detection methods using the two-tailed signed rank test [10], the SVM-Fisher kernel performed significantly better than the spectrum kernel with a p-value of 0.042, but the spectrum kernel shows comparable performance with the other methods. Hence, the spectrum kernel seems to represent a method qualitatively equal to state-of-the art protein classification methods, especially if one considers that no prior biological knowledge was incorporated into the kernel design. This could be an advantage over methods like the Fisher kernel that use a generative model.

[9] The SCOP database is introduced in section 6.

[10] For a description of the ROC_{50} score see section 6.

9 String Subsequence Kernel

Let us remember the example in section 8, this time with a small variation: This time the sentences

1. *The woman is lookin' at you kid.*
2. *The women are looking at you kid.*

should be compared. If we assume, for simplicity of illustration, that the spectrum kernel most probably will match only within single words, the spectrum kernel could still match small variations at the word boundaries (like *lookin'* and *looking*) but would have difficulties in comparing subsequences with a differing letter in the middle of the sequence (like *wom**a**n* and *wom**e**n*). In this case, an extension of the spectrum kernel that could match the subsequences *wom-n*, while ignoring differing characters in between (e instead of a) might improve performance. In the case of text classifications, slight changes in the middle of a word can occur for plural forms, misspellings or for conjugations of verbs, like in the German language if the present form of the verb *find* is replaced by the conjunctive: e.g. *Er **fand** es gut.* vs. *Er **fände** es gut.* For subsequence matches across word boundaries a kernel considering noncontiguous subsequences might, in general, enlarge the tolerance towards noise by accepting more matches between similar but not exact subsequences. Protein classification problems could benefit from such an approach as well, for example, in the case of proteins that contain similar amino acid sequences that differ only elementwise. Therefore a kernel was developed by Lodhi et al. that takes the above considerations into account: The string subsequence kernel (SSK). It extends the spectrum kernel by examining noncontiguous common substrings.

9.1 Kernel Definition

The subsequence kernel as introduced by Lodhi et al. in [18] is the inner product of feature vectors generated by not necessarily contiguous subsequences of length k. Hereby, a text is considered as one sequence including word spaces but ignoring punctuation marks. The string subsequence kernel uses only symbol sequences without incorporating any domain knowledge. The feature space consists of the set of all k-tuples out of a finite alphabet Σ. Thus, the set contains $|\Sigma|^k$ elements. That means all permutations of k symbols form the dimensions of the feature space. The feature vector of a document assigns to each of the subsequences the number of occurrences of these substrings in the text, whereby the subsequences in the document can be noncontiguous and hence, are weighted by a factor exponentially decaying with their full length in the text.
The kernel defines the similarity of two documents by the frequency and compactness of their common subsequences ignoring all information of word

order in the documents. The idea of this comparison is that two documents have a similar content, if they have many substrings in common. Thereby, the degree of continuity of each substring in the document sequence determines how much it contributes to the similarity. A decay factor λ of an arbitrary value of the interval $[0, 1]$ is used to determine how compactly each substring is embedded in the text, it grows exponentially with the length of the respective substring. The kernel is formally defined as follows:

Definition 8 (String Subsequence Kernel (SSK))
Remember that a noncontiguous substring u of a string s is denoted by u=s[i] (see section 4, Definition 3). The feature mapping Φ over the feature space $F = \mathbb{R}^{|\Sigma|^k}$ is defined by:

$$\Phi : X \to F, s \to \Phi(s)$$

whereby the u-coordinate $\phi_u(s)$ for each $u \in \Sigma^k$ is given by

$$\phi_u(s) = \sum_{i:u=s[i]} \lambda^{l(i)} \quad \textit{for some } \lambda \in [0, 1].$$

The kernel as the inner product of the feature vector of the strings s and t is consequently:

$$\begin{aligned} K_k(s,t) &= \textstyle\sum_{u\in\Sigma^k} \phi_u(s)\phi_u(t) = \sum_{u\in\Sigma^k} \sum_{i:u=s[i]} \lambda^{l(i)} \sum_{j:u=t[j]} \lambda^{l(j)} \\ &= \textstyle\sum_{u\in\Sigma^k} \sum_{i:u=s[i]} \sum_{j:u=t[j]} \lambda^{l(i)+l(j)}. \end{aligned} \tag{8}$$

The first sum of equation 8 refers to all possible subsequences of length k, the other ones consider the respective non-continuous subsequences of the strings s and t.
Like in the case of the spectrum kernel, because the feature mapping is defined explicitly, the Mercer's condition does not have to be verified.

Let us consider an example to get a concrete impression:
Assume we want to decide the grade of similarity between the three nephews of Donald Duck: Huey, Dewey and Louie, who are called *Tic*, *Tric* and *Trac* in German. For a kernel that considers subsequences of length 2, given an alphabet $\Sigma = \{t, i, r, c, a\}$ we obtain a $|\Sigma|^2 = 25$ dimensional feature space, where the sentences are mapped as follows (zero-entries in the same coordinates of all three feature vectors are omitted):

	$t-i$	$t-a$	$t-c$	$t-r$	$r-i$	$r-c$	$r-a$	$i-c$	$a-c$
$\phi(tic)$	λ^2	0	λ^3	0	0	0	0	λ^2	0
$\phi(tric)$	λ^3	0	λ^4	λ^2	λ^2	λ^3	0	λ^2	0
$\phi(trac)$	0	λ^3	λ^4	λ^2	0	λ^3	λ^2	0	λ^2

Thus, the kernel between *Tic* and *Tric* is $\lambda^5 + \lambda^7 + \lambda^4$. The similarity score between *Tic* and *Trac* is less, because the subsequence $t - c$ is the only one for which both sequences have non-zero entries in their feature vectors. This kernel value is only λ^7.
By the choice of the decaying factor we can model the kernel properties: A very small decaying factor will penalize longer noncontiguous subsequences considerably, so that only the continuous subsequences have a great influence on the kernel value. For example, for a decaying factor $\lambda = 0.5$, each additional gap symbol will contribute to dividing the respective continuous feature value by two. On the other hand, a decaying vector of the value $\lambda = 1$ would treat any noncontiguous subsequence with an arbitrary number of gaps in the same way, thus gaps would not be penalized at all, which, of course, would be not a very reasonable choice.

9.2 Efficient Kernel Computation via a Dynamic Programming Technique

A direct computation of the features by searching each possible subsequence of length k in the two input strings s and t would be of order of $|\Sigma|^k$ for time and space, whereby $|\Sigma|^k$ is the number of features involved. This is infeasible already for small values of k. Thus, a more efficient possibility for the kernel computation is needed. For a recursive formulation of the problem, that simplifies the computation of the kernel and therefore reduces the computing time, an additional function has to be introduced:

$$K'_l(s,t) = \sum_{u \in \Sigma^l} \sum_{i:u=s[i]} \sum_{j:u=t[j]} \lambda^{|s|+|t|-i_1-j_1+2}$$

$$\text{for } l = 1, ..., k-1,$$

whereby i_1 and j_1 are the indices of the input strings s and t where the first character of the substrings $u = s[i]$ and $u = t[j]$ occurs.
Furthermore, we set:

$$K'_0(s,t) = 1.$$

The difference between this auxiliary function and the original kernel is that, instead of counting the length of the particular subsequences of the feature space within the input sequences, it considers the length from the beginning of the subsequence to the end of the input strings.

Theorem 3 (Recursive computation of the subsequence kernel)
The following recursive computation scheme is valid:

1. Base cases*:*
   (a) $K'_0(s,t) = 1, \quad \text{for all} \quad s,t \in \Sigma^*$
   (b) $K'_l(s,t) = 0, \quad \text{if} \quad min(|s|,|t|) < l$
   (c) $K_l(s,t) = 0, \quad \text{if} \quad min(|s|,|t|) < l$
2. $K'_l(sa,t) = \lambda K'_l(s,t) + \sum_{j:t_j=a} K'_{l-1}(s,t[1:j-1])\lambda^{|t|-j+2}$
   for $\quad l = 1,...,k-1, \quad$ and $\quad a \in \Sigma$
3. $K_k(sa,t) = K_k(s,t) + \sum_{j:t_j=a} K'_{k-1}(s,t[1:j-1])\lambda^2.$

Note that this formula is based on the main idea that for each added character in the input string the kernel incurs a factor λ for each additional length unit. The kernel computation uses two kinds of recursions, one that shortens the subscript of the kernel respectively of the auxiliary function until it becomes smaller than the subsequences considered and one that consecutively reduces the length of the subsequences considered by the kernel and the auxiliary function.
The computation of the auxiliary function $K'_l(sa,t)$ is based on a recursive call without the last character a, whereas the second term as in (3) has to supplement all terms for the subsequences of $k-1$ characters where a is the kth character of both sequences. The first term in (2) is multiplied by λ because every index of the feature vector of s that builds the kernel $K'_l(s,t)$ will be multiplied by a single factor λ, if a is concatenated with the string s and therefore prolonged by one character. The kernel $K_{l-1}(s,t[1:j-1])$ in the second term of equation (2) will get the value 1 for every recursion descent as soon as $l = 0$. The factor $\lambda^{|t|-j+2}$ then provides the value for the length of the string of the previous recursive call and hereby yields $|t|-j+2$ factors λ for each fewer character when the string sa is reduced to s within the recursion.
The kernel $K_k(sa,t)$ in (3) is built by a kernel whose one argument misses the last character a and by a second term that considers all additional common subsequences that would occur in both feature vectors of s and t, if the missing character in the first term of (3) would also be considered: The second term of (3), therefore, counts all common subsequences of $k-1$ characters of s and t where the kth character is the missing a in the first term of (3). This last character a of the k-sequence is considered by a factor λ^2 because each one of those common subsequences will end with the character a and will therefore prolong each subsequence by the length of 1. Thus, $K'_{l-1}(s,t)$ adds all those subsequences to a kernel $K_l(s,t)$ that would be neglected, if the last character a of the string sa of the kernel $K_l(sa,t)$ did not exist. The second term of equation (3) allows to understand why we need

the auxiliary function $K'_l(s,t)$ that is counting the length from a particular subsequence to the end of the strings s and t: the auxiliary function is here called by the input strings s and $t[1:j-1]$ which ensures that for each additional $(k-1)$-subsequence the correct amount of factors λ is added. Let us reconsider the previous example, this time calculating the kernel of the strings *tic* and *tac* explicitly:

Example 1 ***(Kernel computation of the strings tic and tac)***

If we compute the kernel $K_2(tic, tac)$ by directly computing the feature vectors of both strings, we get the following feature vectors ignoring the indices of the vectors where both vectors have zero entries for simplicity:

$$\begin{array}{ll} \text{We denote:} & \Phi(s) = (\phi_{ti}(s), \phi_{ta}(s), \phi_{ic}(s), \phi_{ac}(s), \phi_{tc}(s)) \\ \text{then} & \Phi(tic) = (\lambda^2, 0, \lambda^2, 0, \lambda^3), \\ & \Phi(tac) = (0, \lambda^2, 0, \lambda^2, \lambda^3). \end{array}$$

The kernel is then defined by

$$K_2(tic, tac) = \langle \Phi(tic), \Phi(tac) \rangle = \lambda^6.$$

The recursive computation of the same kernel is as follows: (Remember that $tac_j = c$ denotes that the string *tac* contains the character c at index j.)

$$\begin{array}{ll} (1) & K_2(tic, tac) = K_2(ti, tac) + \sum_{j:tac_j=c} K'_1(ti, tac[1:j-1]\lambda^2 \\ (2) & K_2(ti, tac) = K_2(t, tac) + 0 \\ & \text{since the string } tac \text{ does not contain any character } i. \\ (3) & K_2(t, tac) = 0 \text{ (base case)} \\ (1.1) & \sum_{j:tac_j=c} K'_1(ti, tac[1:j-1])\lambda^2 = K'_1(ti, ta)\lambda^2 \\ & = \lambda^2 \cdot \lambda K'_1(t, ta) + 0 \\ & \text{since the string } ta \text{ does not contain any character } i. \\ (1.1.1) & K'_1(t, ta) = \lambda K'_1(\varepsilon, ta) + \sum_{j:ta_j=t} K'_0(\varepsilon, \varepsilon)\lambda^{|ta|-1+2} \\ & = 0 + 1 \cdot \lambda^3 \end{array}$$

Inserting in 1.1 and then 1 yields:

$$K_2(tic, tac) = 0 + \lambda^6 = \lambda^6$$

The complexity of computation of the SSK for the strings s and t according to the recursive formulation above is $O(k|s||t|^2)$, where k is the length of the subsequences (in their contiguous form). This is only valid, if a dynamic programming approach is used where all the computed terms of the facultative function $K'_l(s,t)$ are reused. This is evident because there is a sum over t with $k|s||t|$ different terms. Leslie et al. propose a more efficient extension of the recursive computation of the SSK that measures the similarity of two documents in time proportional to $O(k|s||t|)$, but we will not go into further detail here.

9.3 Experiments

The series of experiments conducted by Leslie et al. satisfy three main objectives:

1. Compare the performance of the SSK to state-of-the art text classification methods like the spectrum kernel, also called n-grams-kernel (NGK)[11], and the standard bag-of-words kernel (WK), introduced in section 7. The spectrum kernel and the WK are both linear kernels. For the WK the documents are indexed by words with a variant of the $tfidf$ weighting scheme, namely $log(1+tf) * log(n/df)$ (with n being the total number of documents, tf represents the term frequency and df the document frequency).

2. Examine the influence of varying the tunable parameters k (length) and λ (weight) on the kernel outcome.

3. Investigating advantages of combinations of different kernels.

For training and test phase a subset of documents from the Reuters-21578 data set was used. It comprised 470 documents of which 380 were training examples and 90 documents were texts of the test set. Preprocessing of the documents was in all cases limited to the removal of stop words and punctuation marks. The categories to be learned and tested were "earn", "acquisition", "crude" and "corn". For evaluation, the F1 performance measure (see section 6) given by $2pr/(p+r)$, where p is precision and r is recall, was used.
Pointing at aim 3, two combinations of the SSK with different parameters were examined, as well as a combination of the NGK and the SSK. Since out of the tested combinations only the combination of variations of the SSK with different length parameter k provided some improvement of performance over the individual kernels, we will not go into further detail here. The two other objectives can be pursued at the same time by comparing

[11]Note that we refer to n-grams as subsequences of a length k, i.e. in this work we should say k-grams to avoid confusion.

the performance of the SKK with different parameter values to the respective other kernels. Thus, firstly, the length of the considered subsequences (k) in a range of 3 to 14 for the SKK as well as the spectrum kernel was varied while fixing the value of the weight decay parameter to 0.5. The performance of all these variants of the three kernel methods was compared. Secondly, the results for varying weight decay parameter between the values 0.01 to 0.9 with a fixed length k set to 5 were examined. The results showed best generalization performance of the SVM classifier in conjunction with the SSK for a sequence-length between 4 to 7, whereas the λ-value influence on performance differed with each text category. However, the F1 numbers always reached a peak at a certain λ-value and decreased for higher ones. What seems to be most important, the generalization performance of the SSK was comparable to the spectrum kernel and even better in most cases than the classical text representation technique, the WK.

10 Generalization of String and Tree Kernels

Having described two approaches of kernel functions in detail, we would like to give an overview of some kernels presented in the literature. We hereby aim in providing a general formulation of the kernel's building principles that the different approaches offer. On the basis of this general definition, we will examine which kind of variations are possible with which type of kernel and evaluate the effects of these variations.

10.1 General Definition of String Kernels

One can identify the following design criteria behind the string kernels we have just introduced: A feature space expansion over all features u, v possibly occurring in the data structure results in a formulation of a kernel as a sum over all features u in a predefined set of features F, e.g. the set of strings of length k. To extract the relevance of the feature for the data structure examined, the features have to be weighted in some sort depending on the input data. For this purpose, a weighting function w, possibly simply counting the number of occurrences of the features, is introduced. Finally, the classification of the data structure requires the introduction of a similarity measure that compares two data structures. This is realized by a function $d(u, v)$ that compares the single features occurring in the input data which should be classified. These considerations lead to the following equation:

Definition 9 (General Definition of String Kernels) *A general similarity measure $K : X \times X \rightarrow \mathbb{R}$ from the set X of documents to the real*

numbers is defined by

$$K(s,t) := \sum_{u,v \in F} w_s(u) \quad w_t(v) \quad d(u,v)$$

where F is a fixed set of features, $w_.(\cdot) : X \times F \to \mathbb{R}$ is the weighting function and $d : X \times X \to \mathbb{R}$ is the similarity function.

Thereby, u and v are all possible features in the relevant feature space F and $w_s(u)$ respectively $w_t(v)$ are the weights these features are assigned according to the input strings s and t. The terms $w_s(u)$ respectively $w_t(v)$ weight the features a priori or after seeing the data, e.g. according to the frequency of the features in the input string or the length of the subsequences that contain the feature for noncontiguous subsequences. In the simplest case they determine whether the relevant feature is actually present in the input sequence. The function $d(u,v)$ depicts the similarity of u, v; as the reader will notice, this function will often be chosen to denote the identity function written as the Kronecker delta:

$$d(u,v) = \delta_{uv} = \begin{cases} 0 & \text{for } u \neq v \\ 1 & \text{for } u = v \end{cases}$$

The above equation does not always satisfy Mercer's condition for kernels and therefore it yields not necessarily a kernel. However, the definition (9) always constitutes a kernel, if the function $d(u,v)$ denotes a kernel.
We will get a better idea of the function of the different terms in the general definition, if we have a closer look at the choice of the weights and the similarity function $d(u,v)$ for each kernel approach. Two exemplary sentences might help to illustrate the different comparative methods of the various kernel approaches:[12]

Example 2 *He comes to look at the colorful jelly beans.*

Example 3 *He came and looked at the colourful jellies.*

10.2 Application of the General Definition for Various Kernel Approaches

10.2.1 Spectrum Kernel

As introduced in section 8 the spectrum kernel considers subsequences of an input sequence. Thus, in case of linguistic data input, it is able to catch semantic similarities without exact word matches in the two input sequences.

[12]For sake of comprehension and illustration the examples we present are restricted to the area of text classification. Please note that we hereby do not claim that these kernels should be used favorably for text classification, because they are, in most cases equally suitable for other application areas like remote homology detection of proteins.

Concerning the above example, a spectrum kernel that considers subsequences containing 4 symbols can match subsequences like *look* (example 2) and *looked* (example 3) by matching their common subsequence *look*. (The same holds for the word *jelly* (example 2) and *jellies* (example 3)). To consider only substrings of a predefined length k, the kernel is built out of a sum over all possible subsequences of the alphabet of length k. The weights $w_s(u) = n_s(u)$ and $w_t(v) = n_t(v)$ denote the frequency of the contiguous subsequences that actually occur in the input strings. The function $d(u, v)$ checks the identity of u and v: This yields the following kernel equation:

$$K_{spectrum}(s,t) = \sum_{u,v \in \Sigma^k} n_s(u) \quad n_t(v) \quad \delta_{u,v}.$$

10.2.2 String Subsequence Kernel (Standard SSK)

Let us again refer to the above examples 2 and 3: A spectrum kernel cannot match words that origin from the same stem but change characters within themselves like different verb forms of irregular verbs (*come* and *came*) or different spellings of a word (*colourful* and *colorful*). The SSK extends the spectrum kernel in this sense by considering noncontiguous subsequences. For instance, a SSK that considers subsequences of a length of three symbols could match the subsequence *c-me* simply considering the gap as an additional penalizing factor.
As described in section 9, the SSK counts occurrences of possibly noncontiguous subsequences of a fixed number of symbols weighted by a decaying factor according to the length of the subsequences as they occur in the document. Like for the spectrum kernel, a sum over all consecutive substrings u, v of the alphabet Σ restricted to length k is built. The information about non-contiguity of the substrings actually occurring in the input strings is handled via the weights $w_s(u)$ and $w_t(v)$; they denote the frequency of the feature actually present in the input sequence but also weight each present substring by an exponentially decaying factor λ according to its density in the input string, i.e. the length of the subsequence ($l(i)$) that contains u respectively v [13] in s respectively t. Again, the function $d(u, v)$ determines the identity function of u and v. The kernel is calculated as follows:

$$K_{SSK}(s,t) = \sum_{u,v \in \Sigma^k} \sum_{u=s[i]} \lambda^{l(i)} \sum_{v=t[j]} \lambda^{l(j)} \quad \delta_{u,v}$$

$$\begin{aligned} &\text{with:} \\ &d(s,t) = \delta_{u,v} \\ &w_s(u) = \sum_{u=s[i]} \lambda^{l(i)} \\ &w_t(v) = \sum_{v=t[j]} \lambda^{l(j)} \end{aligned} \tag{9}$$

[13] Remember: If $u = s[i]$ is a noncontiguous substring, $l(i)$ denotes the length that u spans in the sequence s.

One can also consider the kernel as a sum over non-consecutive subsequences of undefined length. The number of exact matches of those substrings can be counted via the weights. In this case, the kernel sum is not generated over the set of possible features but the distance measure $d(u, v)$ has to encode the occurrences of the noncontiguous feature subsequences of length k: Thereby, the first and the last element of v respectively u are matched and all common k-length feature sequences occurring in v and u are considered and penalized according to the length of the corresponding subsequence u respectively v. However, a distance measure over sequences of undefined length seems to be computationally too expensive, so that we will stick to the first representation of the SSK.

Taking these two kernels as a basis one can consider variations by varying the parameters of the kernels described by definition 9. Firstly, to strengthen the resistance of a kernel against noise in a classification task the requirement for exact matches of features in the input strings can be loosened by allowing mismatches. Secondly, one can vary the granularity of the elements of a string kernel, e.g. instead of comparing single characters for an elementwise match of a subsequence with a feature one can treat syllables or even whole words as elements a feature is built of. This has major effects on the computational efficiency, for example. In the following, we would like to explore and compare some of the existing approaches that realize these variations on the basis of the general definition 9, before introducing a taxonomy for string kernels based on substructures.

10.2.3 Extensions of the SSK

The Syllable Kernel A main drawback of the string subsequence kernel is its immense computational cost, which increases quadratically with the size of the data. On data sets where the string subsequence kernels have to be approximated to reduce computational complexity, a variation of the SSK might be beneficial; the syllable kernel, that examines the data on syllable level instead of on character level, shortens computing time by compressing the data: The syllable kernel has the same restrictions to the weight factors and the distance measure as the string subsequence kernel except that the subsequences considered are built out of syllables as atomic units and not characters. It has, therefore, the same representation as the SSK in equation 9, whereby the alphabet Σ consists here of syllables instead of single characters. Note that this kernel is designed preferably for text classification tasks, because syllables do not naturally occur in other data sets like sequences of amino acids. Like the SSK, this kernel replaces certain preprocessing techniques like lemmatization by still finding matches for similar words where only certain syllables differ, i.e. words like *look-ed* and *look* of the examples 2 and 3 would still contribute to a matching subsequence

since the syllable (*look*), that represents the stem, remains the same. On the other hand, misspellings or changes of one symbol in a word (like from *come* to *came*) cannot be taken into consideration by this kernel.

Word Sequence Kernel The word sequence kernel is a sequence kernel like the original SSK operating on the word (possibly word stem) level. Cancedda et al. showed that for categorization with the help of string kernels on character level the highly discriminant features, i.e. the features u with a high absolute weight in the linear decision :

$$|w_u| = \left| \sum_j \alpha_j y_j \phi_u(x_j) \right|$$

are often subsequences that match on more than one word [3] (for an example see figure 4). Thus, a kernel that works on word level instead on syllable or character level and that takes the order and locality of words into account seems to be beneficial for text categorization. Furthermore, the word sequence kernel can be calculated more efficiently than the SSK and even the syllable kernel. The word kernel is computed in a similar way as the SSK by replacing single characters by words. This causes a high increase in dimension of the feature space since the feature space should contain all existing words of the data sets the kernel has to deal with. This leads to extremely sparse implicit representations in the feature space and therefore small similarity scores for sequence matches. To prevent the similarity score of the kernel from being too small, a combination of kernels that considers up to k-length subsequences is considered, whereby every single kernel is weighted according to optimization methods like cross-validation:

$$K_{word}(s,t) = \sum_{p=1}^{k} \mu^{1-p} K'_p(s,t)$$

with

$$p = 1, .., k. \quad \text{and} \quad \mu \in \mathbb{R}_+$$

$$K'_p(s,t): \quad \text{word kernel that considers solely features of length } p$$

Thus, the word kernel requires the same formulation as the SSK, with the extension to sum up over a combination of kernels (This time, Σ denotes the finite alphabet of words.):

$$K_{word}(s,t) = \sum_{p=1}^{k} \mu^{1-p} \sum_{u,v \in \Sigma^p} \sum_{u=s[i]} \lambda^{l(i)} \sum_{v=t[j]} \lambda^{l(j)} \quad \delta_{u,v}$$

with $\mu \in \mathbb{R}_+$,

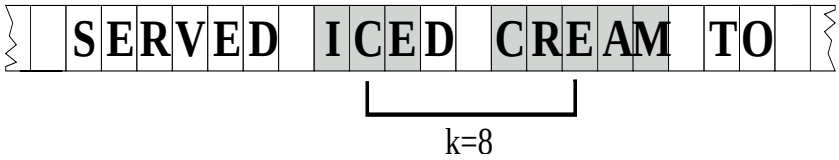


Figure 4: Single features of a string kernel may match parts of consecutive words. In this way, the locality of the words is preserved in the kernel representation (never *cream ice* instead of *ice cream*).

By the parameter μ the different subsequences lengths can be weighted relatively: the bigger μ, the smaller is the influence of multi-word matches. μ should be chosen in a range of $[0, 3]$, for larger values the effects of using a word kernel that tries to match multiple words would vanish almost completely[14]. Before combination the single kernels are sometimes normalized to prevent a weighting because of length of the subsequences considered. Finally, it should be mentioned that for a good performance of the word kernel a certain kind of preprocessing like lemmatization and part-of-speech tagging is necessary; while the standard SSK and the syllable kernel generate some sort of morphological normalization the word kernel obviously does not. Without any preprocessing techniques the two sentences of example 2 and 3, that are semantically very similar, could indeed hardly be matched. Other variations of the SSK do not refer to the change of scale, i.e. characters, syllables or words might be considered as atomic units, but these kernels extend the standard version of the SSK to allow soft matches or different weighting for different symbols.

The String Kernel Weighting Different Symbols with Different Decay Factors (SSK with DDF) This kernel defines a priori different values for λ for each atomic unit, e.g. characters or syllables the subsequence kernel is built of. With this kernel, some building units that one considers of greater importance for a meaning of a sequence can be assigned a higher weight. The kernel is computed in the same way as the original SSK, the components of $w_s(u)$ and $w_t(v)$ only do not decay exponentially with the length of the substring for a unique λ-value, but with the product of the different weights each single symbol is assigned. For example, let us consider the original SSK, that computes a kernel over subsequences of characters. A feature *nap* for the string *narcolepsy* that would receive a weight of λ^8 in

[14] $\mu = 0$ would recover the bag-of-words kernel with

$$\mu^{1-p} = \begin{cases} 0^0 = 1 & \text{for } p = 1 \\ 0 & \text{for } p > 1 \end{cases}$$

the original SSK would now be assigned the weight $\lambda_n\lambda_a\lambda_r\lambda_c\lambda_o\lambda_l\lambda_e\lambda_p$. The whole kernel is computed as follows:

$$K_{SSKw\lambda}(s,t) = \sum_{u,v\in\Sigma^k} \sum_{u=s[i]} \prod_{i_1\leq q\leq i_{|u|}} \lambda_{s_q} \sum_{v=t[j]} \prod_{j_1\leq h\leq j_{|v|}} \lambda_{t_h} \quad \delta_{u,v}$$

$$\text{with } \lambda_{s_q};\lambda_{t_h} \in [0,1]$$

It should be mentioned that in case of text classification a symbol dependent decay factor seems to be more reasonable for word or syllable kernels. On syllable level, for instance, syllables like common verb endings (e.g. the ending *ing* in a present participle (*running*, *laughing*, *playing*) could be assigned a lower weight than syllables constituting stems that contain the meaning of a verb (e.g. *run*, *laugh*, *play*). On character level, however, a reasonable assignment of weights for each letter of the alphabet seems to be more difficult.

The String Subsequence Kernel with Independent Decay Factors for Gaps and for Symbol Matches (SSK with DDFGS) The previously presented kernel does not differentiate, if an atomic unit associated with a certain decay factor is actually a matching symbol or an element of a gap. However, it would be reasonable to penalize highly relevant elements, if they occur in a gap by use of a small decay factor and, on the other hand, to reward important matching symbols with a high decay factor. For example, let us consider the sequence *ice cream with chocolate*. Weighting the single elements according to their part-of-speech we could assign *with* a high decay factor (close to one), if it appears in a gap, but a low matching score (close to zero), if it is part of the feature. As an extension of the previous kernel, this kernels's formulation differs from the above presented kernel only in the λ-values of the weights: For example the weight $w_s(u) = \sum_{u=s[i]} \prod_{i_1\leq q\leq i_{|u|}} \lambda_{s_q}$ of the above kernel is replaced by:

$$w_s(u) = \sum_{u=s[i]} \prod_{i_1\leq q\leq i_{|u|}, s_q\in u} \lambda_{m,s_q} \prod_{i_1<p<i_{|u|}, s_p\notin u} \lambda_{g,s_p}$$

$$\text{with } \lambda_{m,s_q};\lambda_{g,s_p} \in [0,1]$$

Here, λ_{m,s_q} denotes the weights for a specific element in the feature that matches with one of the input strings, λ_{g,s_p} determines the weight by which a symbol of the feature is penalized, if it does not occur in the data. Let us briefly examine the influence of these parameters on the kernel: If we set $\lambda_g = 0$ we recover the spectrum kernel, where only contiguous subsequences are considered, setting $\lambda_g = 1$ allows any number of gaps, and $\lambda_g = 1 - \lambda_m$ implements an IDF-weighting scheme.

The String Kernel with Soft Matches (Soft Kernel) In the original string subsequence kernels, only exact matches can contribute to the similarity of two strings. That might be disadvantageous, for example in the area of text classification: On the world level, synonyms should be considered as similar, and for the syllable or standard subsequences string kernel syllables that change characters because of irregular declination or pluralization forms like the change from f to v for some plural forms in English (for example in *half*, *halves*) should be possible to match. Finally, a kernel realizing soft matches becomes more resistent to noise caused by misspellings. Concerning bio-sequences data e.g. for protein classification, soft matches could be useful to detect common ancestors by neglecting certain amino acids that were replaced during evolution.
An extension of the string subsequence kernels that makes use of a similarity matrix in the implicit feature space could take such soft matches into account. The similarity function $d(u, v)$ is hereby given by the entry A_{uv} of a matrix A that defines the grade of similarity between two k-length features $u, v \in \Sigma^k$. Setting $A_{uv} = 1$ if $u = v$ and $A_{uv} = 0$ otherwise, would recover the original string kernels, on the other hand, the value A_{uv} can be interpreted as the probability that v can be replaced by the feature u. To avoid a feature space expansion to gain the matrix $A \in \mathbb{R}_+^{|\Sigma|^k \times |\Sigma|^k}$ -namely to define a similarity value for each feature combination uv- the similarity on subsequence level might be expressed as a product on the symbol level: $A_{u,v} = \prod_{n=1}^{k} a_{u_n v_n}$. Hereby, $a_{u_n v_n}$ could depict the probability that the element $v_n \in v$ is replaced by $u_n \in u$[15]. In any case, the matrix $A = [A_{uv}]$ has to be positive definite, so that equation 10 expresses a valid kernel[16]. To put the soft matching kernel in the above equation scheme, the function $d(u, v)$ that recovers the identity in the original SSK has to be replaced by the similarity score $A_{u,v}$:

$$K_{soft}(s,t) = \sum_{u,v \in \Sigma^k} \sum_{u=s[i]} \lambda^{l(i)} \sum_{v=t[j]} \lambda^{l(j)} \quad A_{uv}$$

$$\text{with:} \quad \begin{array}{l} d(u,v) = A_{u,v} \\ w_s(u) = \sum_{u=s[i]} \lambda^{l(i)} \\ w_t(v) = \sum_{v=t[j]} \lambda^{l(j)} \end{array} \tag{10}$$

[15]This definition for the matrix A seems to be reasonable especially for protein classification tasks, where substitution matrices, defining which amino acid has been substituted by another one, already exist (e.g. PAM [22]). For text classification the definition $A_{uv} = \prod_{n=1}^{k} a_{u_n v_n}$ should be preferred for kernels on word or syllable level since it is rather difficult to decide which character from our alphabet is often replaced by another one without changing the word meaning.

[16]Remember that according to the general definition 9 $K(s, t)$ defines a valid kernel, if $d(u, v)$ is a kernel.

10.2.4 Mismatch String Kernel

The mismatch kernel [17] is very similar to the above presented string kernel with soft matches. Like the soft matching kernel, it can detect the similarity of different word forms with a common stem and take misspellings into account. The mismatch kernel operates on character level. In contrast to the soft matching kernel, it considers contiguous subsequences of a fixed length k and allows only up to m mismatches per feature. Like the spectrum kernel, the mismatch kernel evaluates the similarity of two documents by counting common subsequences of a fixed length k, but it allows in a graded version also similar subsequences that contain up to m mismatches. There are various ways to weight the similarity of subsequences with up to m mismatches. Leslie et al. use a substitution matrix of probabilities to define $P(u'_n|u_n)$ denoting the probability that the symbol u_n is replaced by the symbol u'_n ($n \in \{1, .., k\}$). The probability that a whole feature u is replaced by u' is calculated by the product $P(u'|u) = P(u'_1|u_1)P(u'_2|u_2)...P(u'_k|u_k)$. Again, as for the soft matching kernel, a substitution matrix seems to be more reasonable for biological data, where substitution matrixes sometimes already exist (e.g. PAM [22]), than for linguistic data.
We can formulate this kernel according to definition 9 via two possibilities: The first one resembles the calculation of the SSK: Again, the sum over all k-length subsequences is built and $d(u, v)$ denotes the identity. All information about the occurrences of substrings in the document, the degree to which these subsequences differ from the feature that is looked for and the weighting of the mismatches according to their probability of being replaced by another element is contained in the weights $w_s(u)$ and $w_t(v)$. To put it in the above scheme, we will have to define m-similar subsequences: If a string u' differs from another string u by at most m characters it is m-similar to u denoted by $u' \overset{m}{\sim} u$. Let $n_s(u')$ and $n_t(v')$ denote the number of occurrences of the substrings u' and v' in s respectively t. We can then define the kernel according to definition 9:

$$K_{mis}(s,t) = \sum_{u,v \in \Sigma^k} \sum_{u' \overset{m}{\sim} u} n_s(u')P(u'|u) \quad \sum_{v' \overset{m}{\sim} v} n_t(v')P(v'|v) \quad \delta_{u,v}$$

$$\begin{array}{lll} \text{with} & P(u'|u) & = & P(u'_1|u_1)P(u'_2|u_2)...P(u'_k|u_k) \\ \text{setting:} & w_s(u) & = & \sum_{u' \overset{m}{\sim} u} n_s(u')P(u'|u) \\ & w_t(v) & = & \sum_{v' \overset{m}{\sim} v} n_t(v')P(v'|v) \\ & d(u,v) & = & \delta_{u,v} \end{array}$$

A second possibility, which reduces the online-calculation costs, can be achieved by transforming the above equation. This transformation provides a different semantics of the function $d(u, v)$ that no longer denotes the identity of two substrings but the probability that substrings are modifications of a common original string which they have replaced.

$$K_{mis}(s,t)$$

$$= \sum_{u,v\in\Sigma^k} \overbrace{\sum_{u'\overset{m}{\sim}u} n_s(u')P(u'|u)}^{w_s(u)} \overbrace{\sum_{v'\overset{m}{\sim}v} n_t(v')P(v'|v)}^{w_t(v)} \overbrace{\delta_{u,v}}^{d_{u,v}} =: (*)$$

With regrouping the sums we get a sum over all subsequences u', v' similar to the features u, v

$$(*) = \sum_{u',v'\in\Sigma^k} \sum_{u,v\in\Sigma^k} \chi_{\substack{u'\overset{m}{\sim}u \\ v'\overset{m}{\sim}v}} n_s(u')n_t(v')P(u'|u)P(v'|v) \quad \delta_{u,v}$$

whereby:

$$\chi_{\substack{u'\overset{m}{\sim}u \\ v'\overset{m}{\sim}v}} = \begin{cases} 1 & \text{if} \quad u' \overset{m}{\sim} u \text{ and } v' \overset{m}{\sim} v \\ 0 & \text{else} \end{cases}$$

Since $\delta_{u,v}$ is the identity function we can replace v with u and then eliminate $\delta_{u,v}$:

$$(*) = \sum_{u',v'\in\Sigma^k} \sum_{u\in\Sigma^k} \left(\chi_{\substack{u'\overset{m}{\sim}u \\ v'\overset{m}{\sim}u}} P(u'|u)n_s(u')n_t(v')P(v'|u) \right)$$

Since $n_s(u')$ and $n_t(v')$ are independent of u we can extract these terms out of the sum over the features u. Hereby, we get a new representation of the mismatch kernel where $d(u', v')$ holds all information about the similarity of the features u', and v' and the weights $w_s(u')$ and $w_t(v')$ just count the occurrences of u' and v' in s respectively t:

$$(*) = \sum_{u',v'\in\Sigma^k} \underbrace{n_s(u')}_{w_s(u')} \underbrace{n_t(v')}_{w_t(v')} \underbrace{\sum_{u\in\Sigma^k} \chi_{\substack{u'\overset{m}{\sim}u \\ v'\overset{m}{\sim}u}} \quad P(u'|u)P(v'|u)}_{d(u',v')}$$

Note that $P(u'|u)P(v'|u)$ is similar to the matrix element A_{uv} in the soft matching kernel, except that here the similarity is not computed directly but via the feature u. In this case $w_s(u')$ and $w_t(v')$ simply determine the occurrences of the string in the document. $d(u, v)$ is not the simple identity like before but all information about the specific aspects of the mismatch kernel is contained in the function $d(u', v')$, which is obviously still a kernel. This approach offers the possibility to calculate all values of $d(u', v')$ beforehand. The resulting matrix stores the probabilities that a certain feature is replaced by two similar sequences of at most m differing characters. In the case of protein classification, for example, it could denote the probability that two proteins have evolved from an original protein. Thus, $d(u', v')$ can be defined according to some pre-known semantic similarity or equivalent functionality of the data structure. To calculate a specific kernel, simply the exact matches of all k-length subsequences of the input string have to be counted, while the similarity of each particular sequence to the feature that is examined can be determined via a look-up table.

10.2.5 Tree Kernels

In some application areas the data is represented in form of trees instead of strings, e.g. as parse trees in natural language processing tasks. It is well known that all trees with d nodes can be rewritten as strings by getting a unifying tree representation e.g. a prefix representation. Thus, every tree kernel can be reformulated to a string kernel. S.V.N. Vishwanathan and A. Smola propose tree kernels based on this transformation in [30]. However, if inner subtree structures, apart from the leaf nodes, are of interest, it is often still necessary to maintain the tree structure, which prohibits the application of string kernels.

10.3 A Possible Taxonomy for Sequence Kernels

Having presented different approaches of string kernels, we would like to present a taxonomy for string kernels that specifies the context in which a certain kernel should be used as well as the advantages and disadvantages of each kernel approach. We propose four evaluation criteria that characterize a kernel:

Efficiency Depending on the fact whether a string kernel computation is based on dynamic programming techniques or the use of suffix tree representations the computational efficiency can vary. Suffix tree based approaches have linear costs in the length of the sequences which are to be classified while methods based on dynamic programming techniques can be a bit less efficient with quadratic time complexity. Among the kernels presented in this section, only the spectrum kernel is computed via suffix trees. The mismatch kernel makes use of a variation of a suffix tree but still requires $O(kn^2)$ time complexity. Thus, apart from the spectrum kernel, all kernels are computed with quadratic time complexity. Still, since the time complexity depends on the length of the input sequence n, the kind of sequences considered can influence efficiency: A data sequence that is examined character by character is much longer than the same sequence, if it is treated as a sequence of words. Therefore, we have specified in table 1 for the efficiency criterium what kind of elementary symbol the considered sequences contain. In table 1 the length of the considered substructures k is treated as a constant and therefore omitted in the efficiency measure.

Tolerance towards Noise The robustness of the presented kernel methods against noise varies with the degree soft matches or mismatches are allowed in the string kernels. Here, soft matches and mismatches denote the same, namely that two subsequences are considered as matching, even if single elements of the subsequences differ.

A second possibility to increase the tolerance towards noise offer those kernels that consider noncontiguous subsequences. The requirement of exact matches is here weakened by allowing the match of sequences that contain additional elements which are irrelevant for the match.
Beside the question of exact or soft matches and contiguity, the length and complexity of the features considered for kernel computation can influence the tolerance towards noise: For instance, a kernel that can match only three-word-features may be more intolerant towards noise than a string kernel matching all equal features with a length of three characters. On sentence level, the former one will get a high kernel score only for sentences that are nearly identical, while the latter can match a lot of subsequences, even if the sentences which have to be compared differ greatly. Since this aspect may be considered as a rather implicit influence on noise robustness we will restrict the measurement of noise tolerance to the two above mentioned aspects of different degrees of freedom the kernels allow for feature matches.

Adaptivity It can be favorable to adapt a kernel for a specific application area, so that prior knowledge or characteristics of the data can influence the kernel calculation. The ability to incorporate prior knowledge into a kernel can offer an important decision criterion on which kernel is to be preferred. Presently, there exist two ways to incorporate prior knowledge into the considered kernels for discrete structures:
Firstly, with the introduction of different decay factors into the string subsequence kernels each element of a subsequence can be weighted according to some prior knowledge. (Remember, for example, that words in the word kernel can be assigned different decay factors according to their part-of-speech. Due to our prior knowledge we assume that certain parts-of-speech like nouns are more important for text classification than others, e.g. prepositions, and therefore, they should have a higher influence on the kernel outcome.)
Secondly, the approaches allowing soft matches provide a possibility to influence the kernel properties by the use of prior knowledge. All these approaches assume that there exists some form of measurement how similar two differing elements of a soft match are. (Compare the similarity matrix A in the soft matching kernel or $P(u'|u)$, the probability that a feature u was replaced by u' as used in the mismatch kernel.) If we have some knowledge at our disposal when features or elements of features are to be considered similar despite looking different, we can incorporate this knowledge in the above mentioned similarity measurements. A prototypical situation where the consideration of prior knowledge could significantly improve the classification by kernel based methods is a protein classification task where evolutionary tendencies are already known and, if incorporated in the classification task, can help to identify similar proteins that evolved from a common ancestor. PAM [22] and BLOSUM [9] matrices are for example,

empirically-derived substitution matrices for proteins that can provide the necessary prior knowledge.

Applications The two main application areas for string kernels presented in the literature are text classification and classification of biological data, namely protein classification. Therefore, these are the application areas along which we would like to distinguish the strengths and weaknesses of each string kernel approach. Since the SSK with individual decay factors can be built from a standard SSK, as well as a syllable or word kernel, its possibilities for applications depend on these underlying kernels.

For convenience, we add a table which characterizes all kernels introduced above with respect to the four criteria presented here.

Table 1: Characteristics of the presented kernels. Since the length k of considered subsequences in the string kernels can be handled as a constant, it is neglected here for the efficiency measure. The computation efficiency depends therefore only on the length of the sequences n over the respective alphabet Σ.

	Efficiency	**Tolerance of Noise**	**Adaptivity**	**Application**
Standard SSK	$O(n^2)$ (n: length of character sequence)	moderate: noncontiguous subsequences, exact matches	no adaptivity	suitable for biological and linguistic data
Syllable Kernel	$O(n^2)$ (n: length of syllable sequence)	moderate: noncontiguous subsequences, exact matches	no adaptivity	only suitable for linguistic data
Word Kernel	$O(n^2)$ (n: length of word sequence)	moderate: noncontiguous subsequences, exact matches	no adaptivity	only suitable for text classification

	Efficiency	**Tolerance of Noise**	**Adaptivity**	**Application**
SSK with DDF	$O(n^2)$ (n: optionally character, syllable or word sequence)	moderate: noncontiguous subsequences, exact matches	adaptivity through symbol dependent decay factors	depends on the SSK it is build of
SSK with DDFGS	$O(n^2)$ (n: optionally character, syllable or word sequence)	moderate: noncontiguous subsequences, exact matches	adaptivity through symbol dependent decay factors	depends on the SSK it is build of
Soft Kernel	$O(n^2)$ (n: optionally character, syllable or word sequence)	high: noncontiguous subsequences, soft matches	adaptivity through similarity matrix A defining weights for soft matches	with standard SSK this kernel is suitable for linguistic as well as biological data
Spectrum Kernel	$O(n)$ (n: length of character sequence)	low: contiguous subsequences, exact matches	no adaptivity	suitable for biological and linguistic data
Mismatch Kernel	$O(n^2)$ (n: length of character sequence)	moderate: contiguous subsequences, soft matches	adaptivity through definition of weights for mismatches, e.g. by a substitution matrix	preferably suitable for biological data

11 Conclusions and Future Research

Discrete structures like strings cause a main problem for learning tasks like categorization, clustering, ranking etc., because common efficient learning algorithms require document representation in a vector space. Kernel methods solve this problem since they allow the use of efficient learning algorithms like the Support Vector Machine for structures that lack such a vector representation. Therefore, finding appropriate and efficient kernel methods that handle data structures, like strings and trees, have recently been of great interest in the machine learning sector. The need of categorization of biological sequences, e.g. for protein classification, emphasizes the practical importance of kernel based learning methods for structures in the field of Bioinformatics. Text classification problems yield another urgent example of an application area for sequence kernels as well. Due to this fact, we have focussed in this work on categorization of texts and biological data sequences with the help of sequence kernels.

There exist several other kernel approaches for sequences, the most common one being the bag-of-words kernel and the Fisher kernel. Sequence kernels represent one of the first competitive alternatives to those approaches, because they are widely suitable for linguistic as well as biological data, computationally efficient and they do not require any generative model, like the Fisher kernels introduced by Jaakkola et al. [12],[13].

Numerous different approaches for sequence kernels have been presented in the literature varying in several aspects like computational efficiency or performance issues. It is therefore important to gain a unifying overview of these kernels that emphasizes the differing possibilities the various approaches offer. With this work we have examined two basic approaches, the spectrum and the string subsequence kernel in detail. We have then provided an overview of the existing approaches, and, for better comparison, we have introduced a unifying representation for sequence kernels. We have developed a taxonomy that brings out the crucial evaluation criteria for sequence kernels. This should yield a good starting point for further investigations:

Most of the presented kernels still have relatively high computational costs. The use of suffix trees for kernel computation can reduce time complexity to linear costs in the input sequence length. At present, among the presented approaches, only the spectrum kernel can make use of this efficient algorithm. Hence, it seems promising to examine whether any possibilities exist to extend the use of suffix trees for kernels allowing soft matches.

Up to now, the sequence kernels were in principle developed for the use of documents in string representation. It should be further investigated if it is possible to extend the use of such kernels to other discrete structures like trees, finite state automata and other dynamical systems.

A main advantage of sequence kernels over e.g. Fisher kernels is that these

kernels are able to capture semantic information about the data without any prior knowledge. Nevertheless, it can be beneficial to incorporate prior knowledge if available. Several approaches implementing soft matches or different decay factors offer mechanisms to adapt kernels according to prior knowledge about the application area. However, except the mismatch kernel they lack experimental investigations. Concrete definitions of similarity matrices for soft matches and symbol dependent decay factors to adapt kernels with the help of prior knowledge should be tested. For text categorization, clever implementations of soft matches may even allow multilingual document processing.

Furthermore, it may improve performance to combine different kernel methods. The combination of kernels were examined by Lodhi et al. [18] in an initial attempt, but never systematically, across different approaches.

Having shown that classification with the help of sequence kernels yield an alternative for state-of-the-art classification methods, it remains to further investigate and improve these kernels, define optimal parameter adjustments and eventually find out under which circumstances the sequence kernels can outperform existing approaches.

12 Acknowledgements

I would like to express my thanks to my supervisor Barbara Hammer for helpful advice and motivation. She has always supported this work with great interest and enthusiasm. Furthermore, I thank Volker Sperschneider for assessing this work and I would like to thank all who have provided useful statements and discussion, notably Aida Senkpiel and Sebastian Blohm.

13 References

References

[1] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25:3389-3402, 1997.

[2] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2).121-157, 1998.

[3] N. Cancedda, E. Gaussier, C. Goutte, and J.-M. Renders. Word-Sequence Kernels. *Journal of Machine Learning Research* 3(Feb):1059-1082, 2003.

[4] M. Collins and N. Duffy. Convolution kernels for natural language. *Neural Information Processing Systems NIPS 14*, 2001.

[5] N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines*. Cambridge, 2000.

[6] R. Fletcher. *Practical methods of Optimization* John Wileyand Sons, Inc., 2nd edition, 1987.

[7] R. Giegerich and S. Kurtz. From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3): pp. 331-353, 1997.

[8] D. Haussler. Convolution kernels on discrete structures. *Technical Report UCSC-CRL-99-10*, University of California, Santa Cruz, July 1999.

[9] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *PNAS*, 89:10915-10919, 1992.

[10] S. Henikoff and J.G. Henikoff. Embedding strategies for effective use of information from multiple sequence alignments. *Protein Science*, 6(3):698-705, 1997.

[11] T. Hubbard, A. Murzin, S. Brenner, and C. Chothia. Scop: a structural classification of proteins database. *Nucleic Acids Research*, 25: 236-239, 1997.

[12] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 2000.

[13] T. Jaakkola, M. Diekhans, and D. Haussler. Using the fisher kernel method to detect remote protein homologies. *ISMB*, pp.149-158. AAAI Press, 1999.

[14] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In Claire Nédellec and Cline Rouveirol, editors, *Proceedings of the European Conference on Machine Learning*, pp. 137-142, Berlin, 1998, Springer.

[15] K. Karplus, C. Barrett, and R.Hughey. Hidden Markov Models for Detecting Remote Protein Homologies. *Bioinformatics*, 14(10):846-856, 1998.

[16] C. Leslie, E. Eskin, and W.S. Noble. The spectrum kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, pp. 564-575, 2002.

[17] C. Leslie, E. Eskin, J. Weston, and W.S. Noble. Mismatch string kernels for SVM protein classification. *Neural Information Processing Systems 15*, 2002.

[18] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2002.

[19] G. Salton, A. Wong, and C.Yang. A vector space model for automatic indexing. *Communications at the ACM*, 18(11):613-620, 1975.

[20] S.L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1:3, 317-327, 1997.

[21] C. Saunders, H. Tschach, and J.Shawe-Taylor. Syllables and other String Kernel Extensions. *Proceedings of the Nineteenth International Conference on Machine Learning (ICML '02)*, 2002.

[22] R.M. Schwartz and M.O. Dayhoff. *Atlas of Protein Sequence and Structure*, chapter: Matrices for detecting distant relationships, pp.353-358. National Biomedical Research Foundation, Silver Spring, MD, 1978.

[23] A. Smola and B. Schölkopf. A tutorial on support vector regression. *NeuroCOLT Technical Report NC-TR-98-030*, Royal Holloway College, University of London, UK, 1998.

[24] S. Sonnenburg, G. Rätsch, A. Jagota, and K.-R. Müller. New methods for splice site recognition. *Proceedings of the International Conference on Artificial Neural Networks*, 2002.

[25] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249-260, 1995.

[26] V. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian].* Nauka, Miscow, 1979. (English translation: Springer Verlag, New York, 1982).

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

[28] V. Vapnik. *Statistical Learning Theory*, John Wiley and Sons, Inc., New York, 1998.

[29] J.-P. Vert. Support vector machine prediction of signal peptide cleavage site using a new class of kernels for strings. *Proceedings of the Pacific Symposium on Biocomputing 2002*, pp. 649-660. World Scientific, 2002.

[30] S.V.N. Vishwanathan and A.Smola. Fast kernels for string and tree matching. *Neural Information Processing Systems 15*, 2002.

[31] C. Watkins. Dynamic alignment kernels. *Technical Report CSD-TR-98-11*, Royal Holloway, University of London, January 1999.

[32] 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 (2000).