In Proc. of CIKM 2006, Arlington, VA, USA

3DString: A Feature String Kernel for 3D Object Classification on Voxelized Data

Johannes Aßfalg, Karsten M. Borgwardt, Hans-Peter Kriegel
Institute for Computer Science
Ludwig-Maximilians-University Munich, Germany
{assfalg|kb|kriegel}@dbs.ifi.lmu.de

ABSTRACT

Classification of 3D objects remains an important task in many areas of data management such as engineering, medicine or biology. As a common preprocessing step in current approaches to classification of voxelized 3D objects, voxel representations are transformed into a feature vector description.

In this article, we introduce an approach of transforming 3D objects into feature strings which represent the distribution of voxels over the voxel grid. Attractively, this feature string extraction can be performed in linear runtime with respect to the number of voxels. We define a similarity measure on these feature strings that counts common k-mers in two input strings, which is referred to as the spectrum kernel in the field of kernel methods. We prove that on our feature strings, this similarity measure can be computed in time linear to the number of different characters in these strings. This linear runtime behavior makes our kernel attractive even for large datasets that occur in many application domains. Furthermore, we explain that our similarity measure induces a metric which allows to combine it with an M-tree for handling of large volumes of data. Classification experiments on two published benchmark datasets show that our novel approach is competitive with the best state-of-the-art methods for 3D object classification.

1. INTRODUCTION

Over the last years an ever increasing number of 3D data has been generated in fields as different as Computer Aided Design (CAD) applications and biological structural databases like the Protein Data Bank (PDB) [3]. A huge number of 3D objects can also be found scattered all over the World Wide Web. In experimental datasets like the Princeton Shape Benchmark Dataset (PSB) [26] or the NTU 3D Model Benchmark Dataset [5] such objects from the WWW have been collected and manually been classified according to their function or the human perception of geometric similarity.

Due to the ongoing work in the field of structural biology and the increasing interest in industrial 3D CAD systems, the amount of available 3D data will keep on growing over the next years. Therefore, the automatic classification of newly created or newly found 3D data into known classes will continue to be a topic of interest in data engineering.

A very wide-spread technique in 3D object classification is to apply a voxelization preprocessing step to the original 3D data [13], e.g. provided by means of triangle meshes or in case of proteins by means of atomic coordinates. The advantage of a voxelized representation lies in the ability to level out small differences between similar models that may be due to the usage of different levels of detail. Furthermore, voxels are a more elegant way of modeling solid surfaces than a collection of connected points forming triangle meshes. Voxels can also be used to model filled interiors of 3D objects while triangle representations merely describe their surfaces. Finally, voxel representations usually have a lower complexity and thus allow for a more efficient computation of certain characteristics of 3D data.

Most work on the classification of voxelized data uses a feature-based approach [14, 8, 1, 15, 20], i.e. a specific number of numerical features is extracted for each voxelized object. These features form so-called feature vectors and the whole process can be regarded as a mapping of a voxelized object to a (potentially) high dimensional space called the feature space.

Mapping voxelized data to feature vectors is attractive, as it transfers a complex data type into a simpler one, on which a huge family of distances, similarity measures and efficient data mining algorithms are available. Unlike vectors, strings are complex data, providing information about the structure of an object. Thanks to the suffix tree, string mining can nevertheless be performed very efficiently, and distances, similarity measures and fast data mining algorithms have been developed for strings as well. For this reason, we decided to explore a novel approach to 3D voxelized object classification: Instead of a vector transformation, we examined if a structured description of 3D objects, namely feature strings, can be used for fast and accurate classification.

The remainder of the paper is organized as follows: In Section 1.1 we will review current approaches to 3D object classification that are based on feature vector transformations. After giving a short introduction to kernel methods in Section 2, we will review existing kernel methods on strings (Section 2.3). In Section 3, we will define our 3D feature strings on voxelized data and a similarity measure for these strings, a special subclass of the so-called spectrum kernel. Its performance is then evaluated and compared to other approaches on benchmark datasets in Section 4. In section 5 we discuss the outcome of the experiments. Section 6 concludes the paper.

1.1 Approaches to 3D Object Classification

Before we define our novel method, we will review the state-of-the-art in 3D object classification in the following. First, we will give a short introduction to voxelization in general and second, we will present the ideas of the most prominent approaches in 3D

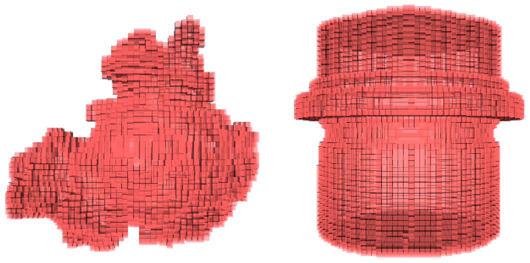

Figure 1: Voxelized Representations of a Protein and a Bolt

object classification.

Voxelization. Voxels are the three dimensional equivalent of two dimensional pixels. In order to voxelize a 3D object, at first a minimal bounding cube is constructed around the object. According to the desired voxel resolution r, each side of the surrounding cube is partitioned into r segments of the same size. This segmentation partitions the cube into r^3 small cubes, the voxels. A single voxel is considered filled or black if a triangle of the original triangle mesh intersects the voxel and is considered not filled or white otherwise. The collection of filled and not filled voxels inside the cube is called a voxel grid.

In [13] Kaufman introduced an algorithm that yields a conservative approximation of the surface of a 3D object by means of voxels. In Figure 1 the voxel grids of a protein and a bolt are depicted.

Shell Model. An intuitive technique to describe 3D shapes was presented by Ankerst et al. in [1]. At first the balance point M of an object is determined. Afterwards a number of spheres centered at M is constructed. The radius of the largest sphere is chosen to result in a minimum bounding sphere of the 3D object. The radii of the other spheres to be constructed are equally distributed between 0 and the radius of the largest sphere. The constructed spheres thus lead to a shell-like spatial partition of each object. Although the authors originally used surface points to represent the objects this technique can easily be applied to voxels. For each shell the number of filled voxels that lie inside this shell is determined. Being a cube actually, a filled voxel is considered lying inside a certain shell if the center of the voxel cube lies inside the shell. Thus each filled voxel is assigned to one shell only. The resulting histogram reflects the fraction of a 3D object that lies inside a certain shell. For each object its specific histogram is used as the feature vector.

Spherical Harmonics. In [14] and [8] Kazhdan, Funkhouser et al. defined a mapping for voxelized objects to feature space based on spherical harmonics.
According to the theory of spherical harmonics any spherical function $f(\theta, \phi)$ can be decomposed as

$$f(\theta,\phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{m=l} a_{lm} Y_l^m(\theta,\phi)$$

where Y_l^m is the so-called spherical harmonic function. The authors define a number of spherical functions by intersecting spheres of different radii with the voxel grid. The function associated with a certain sphere yields 1 if the point specified by the radius of the sphere and the two angles θ and ϕ lies inside a filled voxel and 0 otherwise.

Each of the spherical functions is subsequently decomposed as described above. The coefficients $a_{lm} \in \mathbb{C}$ are finally used to calculate the value $s_l \in \mathbb{N}$ for a certain choice of l where

$$s_l = \sqrt{\sum_{|m| \leq l} |a_{lm}|^2}$$

Thus every sphere intersecting the voxel grid yields several values s_l and the collection of all s_l of all spheres constitutes the feature vector for the current 3D object.

Eigenvalue Model. The authors of [15] use the Principal Component Analysis technique [12] to define meaningful numerical features for 3D objects. At first the minimum bounding box of an object is calculated. The largest extent of this box determines the size of the minimal bounding cube of the object into which the object is placed in such a way that the minimum bounding box is centered inside the minimum bounding cube. The cube is then partitioned into $n^3 (n \in \mathbb{N})$ cubical partitions. Obviously n should be smaller than the voxel resolution r so that more than one voxel is covered by each spatial partition.

After the object is voxelized, three numerical features are extracted for each of the n^3 cubical partitions. Let $V = \{v_1, \ldots, v_m\}$ be a set of filled voxels lying inside the same cubical partition. In the next step, each $v_i \in V$ is translated such that the balance point M of V coincides with the origin, and the covariance matrix $\mathbf{C}$ for V is computed as follows:

$$\mathbf{C} = \frac{1}{|V|-1} \sum_{j=1}^{m} (\vec{v_j} - M) \cdot (\vec{v_j} - M)^T$$

The covariance matrix can be decomposed as $\mathbf{C} = \mathbf{VEV^T}$, where $\mathbf{V}$ is an orthonormal matrix containing the eigenvectors of $\mathbf{C}$ and $\mathbf{E}$ is a diagonal matrix containing the eigenvalues of $\mathbf{C}$. The eigenvectors are called *principal axes* of V. They describe the three orthogonal axes where the scattering of the elements is greatest. The three eigenvalues describe the variance along the principal axes and thus can be used to characterize the shape of the elements of V.

After three eigenvalues have been calculated for each partition, a feature vector for the complete object can finally be derived.

Grid D2. In [20] Osada et al. presented a technique called D2. A number of points is distributed randomly on the surface of the object to be mapped to the feature space. Then all possible pairwise Euclidean distances are computed. These distances are finally used to create a histogram for each object reflecting the distribution of distances between the surface points of a specific object.

This technique was adapted by Shih et al. in [25] for the use with voxelized data. The method the authors call Grid D2 randomly chooses two filled voxels and calculates their Euclidean distance. This is repeated r^3 times (again, r is the voxel resolution) so that a histogram reflecting the voxel distribution can be created. The entries of the histogram are finally normalized by dividing them by r^3. Again, the histogram for each object can be used as the corresponding feature vector.

2. PRIMER ON KERNEL METHODS

In the following, we will shortly review the basic concepts of kernel methods, as we will employ kernel functions for 3D voxelized object comparison in this article. The interested reader is referred to excellent books [23] and tutorials [4] for a complete introduction to kernel methods, respectively.

2.1 Kernel Methods

Kernel methods represent a family of related machine learning and data mining algorithms. As their core component, they all rely on a kernel function, which can be thought of as a positive definite measure of similarity on input data. Based on this kernel function, tasks such as as classification via Support Vector Machines [29], regression [7], clustering [2] and Principal Component Analysis [24] or kernel Fisher Discriminant Analysis [19] can be handled.

2.2 Kernel Trick

All kernel methods utilize the so-called "kernel trick". Data points from input space are mapped into a usually higher-dimensional feature space. Linear hypotheses in this feature space may correspond to non-linear hypotheses in input space, thereby allowing to use algorithms for linear problems in feature space in order to solve non-linear ones in input space.

Evaluating a complex mapping from input space to feature space might be computationally or numerically problematic. Yet a second "kernel trick" is the fact that all computations in feature space can be done implicitly by evaluating the so-called "kernel function". This kernel function represents a scalar product in feature space and a measure of similarity in input space. Defining a kernel function therefore allows us to deal with hypotheses in feature space without explicitly mapping points from input into feature space.

2.3 Kernels on Structured Objects

As kernel methods were successful in many application areas, the interest in kernel methods on non-vectorial data grew. As an essential step, Schölkopf [22] realized that the kernel trick can also be applied to non-vectorial data, simply by expressing similarity between structured data via a kernel function. Haussler [9] and Watkins [31] were the first to define a principled way of designing kernels on structured objects. Based on this framework, kernels on structured objects such as strings and trees, transducers, dynamical systems, on nodes in graphs and on graphs have been defined over recent years.

2.4 Kernels on Strings

Following [27], kernels for strings can be divided into five categories: first, polynomial-like kernels (e.g. [32]); second, kernels derived from probabilistic models (e.g. [10]); third, kernels based on alignments (e.g. [18]); fourth, spectrum-like kernels that count common substrings in two input strings ([30, 17, 16]; fifth, kernels incorporating positional information on substrings (e.g. [21]).

Most interesting for our application is the class of spectrum-like kernels. These are all built on the fundamental idea to count identical subsequences, called k-mers, of input strings [16]. More formally, given two strings x and x' from an alphabet Σ. Then the exact matching spectrum kernel is defined as:

$$k(x, x') := \sum_{s \sqsubseteq x, s' \sqsubseteq x'} \omega_s \delta_{s,s'} = \sum_{s \in \Sigma^*} num_s(x) num_s(x') \omega_s .$$

where s and s' are substrings of x and x' respectively, ω_s is a weight assigned to string s and $\delta_{s,s'}$ is the delta function that equals 1 if s and s' are identical, 0 otherwise. $num_s(x)$ is the number of occurrences of substring s in string x.

This class of string kernels comprises the "bag of character approach" and the "bag of words" kernel which compare all pairs of characters and words in two strings, respectively [11]. One common approach to save runtime is to consider strings up to a certain length only. This method is sometimes referred to as the *limited range correlations* string kernel. The alternative approach is to consider substrings of fixed length k only, usually referred to as the k-spectrum kernel.

While dynamic programming implementations of the spectrum kernel computation require quadratic runtime, computation with runtime linear to the added total length of input strings has been made possible by [30].

Beside exact matching kernels, string kernels that count gappy or non-completely identical substrings in two strings have been developed, often motivated by the problem of aligning biological sequences in bioinformatics [17].

3. 3D FEATURE STRINGS

3.1 Derivation of 3D Strings

In this section we will describe the generation of the feature strings we use to characterize 3D structures. At first we state the general technique of mapping 3D objects to a voxel grid. Then we will outline an algorithm that yields the string representations of 3D objects in linear runtime with respect to the size of a given voxel grid.

3.1.1 From 3D Objects to Voxel Grids

As mentioned above, a three dimensional object is at first transformed into a voxel grid. Such a three dimensional grid consists of r^3 voxels where r is called the resolution of the voxel grid. A voxel is considered filled if and only if the original 3D object intersects the voxel. Each voxel can be addressed by its coordinates (x, y, z), where $1 \leq x, y, z \leq r$ and $x, y, z \in \mathbb{N}$. A voxel grid v can therefore be considered as a function

$$\begin{array}{lcl} v : \mathbb{N}^3 & \rightarrow & \{0, 1\} \\ v(x, y, z) & = & \begin{cases} 1 & \text{if} \quad (x, y, z) \quad \text{is a filled voxel} \\ 0 & \text{else} \end{cases} \end{array}$$

3.1.2 From Voxels to Feature Strings

The straightforward feature extraction step we next describe is a key step of our application. Our motivation was to generate strings that describe the distribution of filled voxels along each of the three axes, x, y and z. For the computation of the 3D feature strings we iterate through the voxel grid, once for each dimension x, y and z, and create one feature string for each dimension.

Without loss of generality, let us assume now that we want to determine the feature string for dimension x. First, we consider all voxels with x=1 only; these voxels form a y-z-plane. We then count the number $\#filled$ of filled voxels in this y-z-plane. Afterwards, we append $\#filled$ times the current value of x, which is "1", in the first iteration, to our feature string s_x (see Figure 2). We repeat this procedure for all values of x up to r, i.e. we consider all y-z-planes defined by x from 1 to r. The following pseudocode illustrates the feature string generation:

```
Given: empty strings s_x
       voxel grid v with resolution r
for (1<=x<=r)
  for (1<=y,z<=r) //iterate through plane
    if (v(x,y,z)==1)
      s_x=append_character(s_x,'x')
```

The strings s_y and s_z can be constructed analogously. After three iterations through a given voxel grid, all the required strings have been constructed in linear time w.r.t. the number of voxels. Thus the complexity of the string extraction step is equal to $O\left(r^3\right)$.

Note that we append character representations of numbers to the strings. This is different from adding the single digits a number consists of. Imagine an empty string s. Let s_1 be the result of the

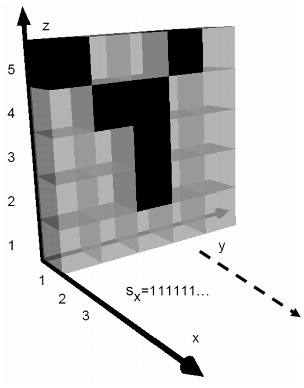

Figure 2: First Considered Plane for the Construction of s_x.

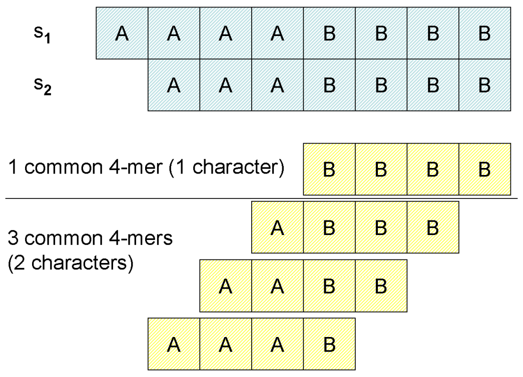
s1 A A A A B B B B
s2 A A A B B B B
1 common 4-mer (1 character) B B B B
3 common 4-mers (2 characters)
A B B B
A A B B
A A A B

Figure 3: Spectrum kernel computation for 4-mers on two input strings.

concatenation of s with the numbers 1 and 2, i.e. $s_1 = s$+'1'+'2'. Similarly the attachment of the number '12' to s results in s_2. Then we consider $s_1 \neq s_2$. This can be implemented by using a special character as separator between the numbers or by bijectively mapping the numbers to an alphabet that is made up of r single-character elements.

Obviously, the feature strings generated are sorted, as all occurrences of the same character appear in one consecutive block. We will exploit this characteristic of our feature strings for fast string kernel computation.

Note furthermore that our kernel can handle the strings in a compressed representation, e.g. the string 'AAAAABBBBCCC' can also be stored and be processed as '5A4B3C'.

To conclude this subsection, let us summarize the mapping from a three dimensional object to a set of strings describing this object: At first a given object is transformed into a voxel grid. Then this voxel grid is traversed plane-wise and whenever the voxel function v equals 1 for a triple of voxel coordinates in this plane, the string s_x is elongated by one character, the x coordinate of this filled voxel. These strings reflect the distribution of filled voxels along the three axes of the coordinate system. Finally, after repeating this procedure for dimension y and z, a three dimensional object is described by a set of three strings s_x, s_y, and s_z.

3.2 3D String Kernel

We choose to use a basic similarity measure on these feature strings, namely a so-called *spectrum kernel*. The *spectrum kernel* counts pairs of identical substrings in two input strings as a similarity measure for two strings (see Figure 3).

3.2.1 *All k-mers, Limited Range Correlation, and k-Spectrum*

We explore three variants of the spectrum kernel, namely one version that considers all k-mers in two input strings, and another version that looks at all k-mers up to a certain length K, and a third kernel that scans strings for k-mers for one fixed k-mer length K only. We will refer to the first one as the all k-mer kernel, to the second one as the limited range correlation (lrc) kernel and the third one as the k-spectrum kernel. We will present formal definitions of all three in the following.

Given two strings x_1 and x_2 from an alphabet Σ. s and s' are substrings of x_1 and x_2 respectively, which is denoted by $s \sqsubseteq x_1$ and $s' \sqsubseteq x_2$. $|x_1|$ is the length of string x_1. $\delta_{s,s'}$ is the delta function that equals 1 if s and s' are identical, 0 otherwise. $num_s(x_1)$ is the number of occurrences of substring s in string x_1, throughout the remainder of the paper.

DEFINITION 1 (ALL K-MER KERNEL). *Then the all k-mer kernel is defined as*

$$k(x_1, x_2) := \sum_{s \sqsubseteq x_1, s' \sqsubseteq x_2} \omega_s \delta_{s,s'} = \sum_{s \in \Sigma^*} \omega_s * num_s(x_1) * num_s(x_2),$$

where $\omega_s = 1$ *for all s.*

DEFINITION 2 (LRC KERNEL). *Under the same conditions, the lrc kernel with a maximum k-mer length K is defined as*

$$k(x_1, x_2) := \sum_{s \sqsubseteq x_1, s' \sqsubseteq x_2} \omega_s \delta_{s,s'} = \sum_{s \in \Sigma^*} \omega_s * num_s(x_1) * num_s(x_2),$$

where $\omega_s = 1$ *for* $|s| \leq K$ *and* $\omega_s = 0$ *for* $|s| > K$.

DEFINITION 3 (K-SPECTRUM KERNEL). *Under the same conditions, the k-spectrum kernel with a fixed k-mer length K is defined as*

$$k(x_1, x_2) := \sum_{s \sqsubseteq x_1, s' \sqsubseteq x_2} \omega_s \delta_{s,s'} = \sum_{s \in \Sigma^*} \omega_s * num_s(x_1) * num_s(x_2),$$

where $\omega_s = 1$ *for* $|s| = K$ *and* $\omega_s = 0$ *for* $|s| \neq K$.

Beside using the all k-mers kernel that considers all k-mers, it seems attractive to use the lrc kernel as well, because the all k-mers kernel weighs long k-mers much stronger than short k-mers, although $\omega_s = 1$ for all s. The reason is that every common k-mer contains two common (k-1)-mers, consequently the total weight of a k-mer is twice the weight of a (k-1)-mer plus 1. Hence, in our 3D object classification task, the all k-mer kernel would give high similarity scores to objects that have one long match in their feature vectors. It therefore seems plausible to explore a second approach, the lrc kernel, which neglects matches longer than a threshold K. Thus the lrc kernel is more likely to give high similarity scores to two objects with many common K-mers. The third approach we examine, the k-spectrum kernel measures similarity in terms of k-mers of one fixed length only. Longer and shorter matches are not considered for determining the match score.

3.2.2 *Joint 3D String Kernels*

Given two voxel objects v_1 and v_2, we compute the spectrum kernel pairwise for the x-strings, y-strings and z-strings of v_1 and v_2 and hence obtain three string kernel values for v_1 and v_2. As addition and pointwise multiplication of kernels preserve positive definiteness, a simple way of fusing these three similarity measures into one joint kernel is both addition and pointwise multiplication. We therefore defined two variants of a joint 3D string kernel on all three x- , y-, z-strings, namely the sum 3D string kernel

$$k_{sum}(v_1, v_2) = k_x(s_x(v_1), s_x(v_2))) + \\ +k_y(s_y(v_1), s_y(v_2)) + k_z(s_z(v_1), s_z(v_2)),$$

and the pointwise product (pp) 3D string kernel as

$$k_{pp}(v_1, v_2) = k_x(s_x(v_1), s_x(v_2))).* \\ .*k_y(s_y(v_1), s_y(v_2)).*k_z(s_z(v_1), s_z(v_2)),$$

where .* denotes pointwise multiplication, not matrix multiplication. To scale all kernel values to the same range, we normalized all kernel values beforehand by

$$K_{normalized}(x, y) = \frac{K(x, y)}{\sqrt{K(x, x) * K(y, y)}}.$$

Our choice of similarity measure has two great advantages: First, as a positive definite kernel function, the spectrum kernel allows us to employ kernel methods for classification, such as Support Vector Machines, regression or PCA on our feature strings.

Second, the special characteristic of our 3D strings, namely that characters appear ordered only, allows us to compute the spectrum kernel of two 3D strings extremely fast. Whereas string kernels in general can be computed in time linear with respect to the total common length of two input strings (as shown in [30]), the spectrum kernel on our feature strings can be computed in time linear to the number of different characters in both input strings. As the number of different characters in our strings is upper-bounded by the voxel grid resolution r, one could also describe the runtime complexity of the spectrum kernel on our feature strings as linear with respect to the voxel grid resolution.

We will show in the following that this result holds for all of our three 3D string kernels, the all k-mers, the lrc and the k-spectrum 3D string kernel.

3.3 Fast 3D String comparison

First, we will explain the linear time computation of the all k-mers string kernel. Besides showing that this is possible in time linear to the voxel grid resolution, we will provide the actual algorithm for doing so in the proof of the following theorem.

THEOREM 4. *The all k-mers 3D string kernel can be computed in linear time with respect to the number of distinct characters in both input strings.*

PROOF. Given two input strings x_1 and x_2. First, we assume that both strings contain the same character a only. Then the number of k-mers s in string x of length $|x|$ is obviously $num_s(x) = |x| - k + 1$. The number of common k-mers in x_1 and x_2 is then $num_s(x_1) * num_s(x_2)$. If length $|x_1| = n$ and $|x_2| = m$ and $z = min(n, m)$, then the number of all common k-mers with $1 \leq k \leq z$ can be computed as

$$compute\_k\_mers\_one\_char(x_1, x_2, a) =$$
$$= \sum_{i=1}^{z}(n - i + 1)(m - i + 1) =$$
$$= z(nm + n + m + 1) + (-n - m - 2)\sum_{i=1}^{z} i + \sum_{i=1}^{z} i^2 =$$
$$= \frac{1}{3}z^3 - \frac{1}{2}(n + m + 1)z^2 + \frac{1}{2}(2 * nm + n + m + \frac{1}{3})z$$

Second, we assume that both strings contain the same two different characters a and b only. Then the number of common k-mers can be divided into three classes: pure a k-mers (consisting of a's only), pure b k-mers (consisting of b's only), and ab k-mers (consisting of a's and b's). The former two can be easily computed as described above. The number of ab k-mers in x_1 and x_2 can be computed as

$$compute\_k\_mers\_two\_chars(x_1, x_2, [a, b]) =$$
$$\min(num_a(x_1), num_a(x_2)) * \min(num_b(x_1), num_b(x_2)).$$

Third, we assume that both strings consist of three (or more) characters a, b, c each. Common k-mers that consist of three different characters a, b, c can only occur if the number of b's in x_1 and x_2 is identical. After determining all common pure a, pure b and ab k-mers, we transform all a's into b's in both strings (resulting in x'_1 and x'_2) and then compute all bc k-mers as

$$\min(num_b(x'_1), num_b(x'_2)) * min(num_c(x'_1), num_c(x'_2)).$$

The number of common bc k-mers in x'_1 and x'_2 is obviously identical to the added number of common bc and abc k-mers in x_1 and x_2. This procedure recursively turns all k-mers based on three or more characters into k-mers of two different characters, which can be computed efficiently as described above.

This results in the following linear-time algorithm for computing the all k-mers kernel, where character(i) is the i-th character in the alphabet Σ (For ease of presentation, we assume that both strings contain every character in Σ at least once):

function $compute\_3D\_string\_kernel(x_1, x_2)$

```
Given: strings x_1 and x_2, alphabet Sigma

for i from 1 to size(Sigma)
   c = character(i);
      if (num_c(x_1)> 0 and num_c(x_2)> 0 )
         all_k_mers = all_k_mers +
         compute_k_mers_one_char(x_1,x_2,c);
      end
end

for i from  2 to size(Sigma)
c = character(i);
   if (num_c(x_1)> 0 and num_c(x_2)> 0 )
      all_k_mers = all_k_mers +
      compute_k_mers_two_chars(x_1,x_2,
        [character(i-1),character(i)]);
      if num_c(x_1) == num_c(x_2)
         turn all character(i-1) in x_1
           into character(i) -> x_1'
         turn all character(i-1) in x_2
           into character(i) -> x_2'
         x_1'->x_1
         x_2'->x_2
   end
end
return all_k_mers;
```

The fact that we have to iterate over all characters twice only shows that our algorithm requires linear runtime to calculate the 3D string kernel. □

Second, if we are not interested in all k-mers, but only in k-mers up to a certain length K only, computation of the string kernel has to exclude all longer k-mers. We will show that computational effort remains linear to the voxel grid resolution in the following theorem.

THEOREM 5. *The limited range correlation 3D string kernel with a maximum k-mer length of K can be computed in linear time with respect to the number of distinct characters in both input strings.*

PROOF. The proof for the lrc kernel is analogous to the proof for the all k-mer string kernel, except for two definitions: As we are considering k-mers up to a fixed length K only, the functions *compute_k_mers_one_char* and *compute_k_mers_two_chars* have to be redefined.

To compute k-mers from two strings both consisting of a sequence of the same character only:

$$compute\_k\_mers\_one\_char(x_1, x_2, a) =$$
$$= \sum_{i=1}^{z}(n-i+1)(m-i+1) =$$
$$= z(nm+n+m+1) + (-n-m-2)\sum_{i=1}^{z} i + \sum_{i=1}^{z} i^2 =$$
$$= \frac{1}{3}z^3 - \frac{1}{2}(n+m+1)z^2 + \frac{1}{2}(2*nm+n+m+\frac{1}{3})z$$

but z is now set to $min(|x_1|, |x_2|, K)$[1] because no matching k-mer may be longer than K, or longer than any of the two strings.

To compute k-mers from two strings both consisting of the same two characters only, we have to define some notation first: Given two strings x_1 and x_2. We define $z_a = \min(num_a(x_1), num_a(x_2))$ and $z_b = \min(num_b(x_1), num_b(x_2))$. We then have to distinguish two cases, namely

$$Case\,1: \; z_a + z_b \leq K$$
$$compute\_k\_mers\_two\_chars(x_1, x_2, [a, b]) = z_a * z_b, \quad (1)$$
$$Case\,2: \; z_a + z_b > K$$
$$compute\_k\_mers\_two\_chars(x_1, x_2, [a, b]) =$$
$$= \sum_{i=2}^{K} min(z_a, z_b, i-1) =$$
$$= \sum_{i=2}^{min(z_a,z_b)} (i-1) + \sum_{i=min(z_a,z_b)+1}^{K} min(z_a, z_b) = \quad (2)$$
$$= \frac{1}{2} * (min(z_a, z_b) - 1) * min(z_a, z_b) +$$
$$+ (K - min(z_a, z_b)) * min(z_a, z_b) \quad (3)$$

This first case can be dealt with as in the all k-mers setting, as no common k-mer can be longer than K in this setting. In the second case, we have to compute k-mers from length 2 to K, as the number of k-mers cannot be larger than z_a or z_b or $k-1$, as we are interested in k-mers that contain at least one a and at least one b.

Since both terms can be computed directly, i.e. in constant time, the overall runtime of our algorithm does not change, i.e. remains linear as for the all k-mer kernel. □

If as a third alternative, we want to consider k-mers of a fixed length K only, we have to set kernel values for pairs of longer and shorter matches to zero. Again, we explain how to achieve this in linear time in the following theorem and proof.

THEOREM 6. *The K-spectrum kernel 3D string kernel with a fixed k-mer length of K can be computed in linear time with respect to the number of distinct characters in both input strings.*

PROOF. The argumentation follows the two previous proofs. The two functions are now redefined as:

$$compute\_K\_mers\_one\_char(x_1, x_2, a) =$$
$$max(0, (num_a(x_1) - K + 1)) * max(0, (num_a(x_2) - K + 1))$$

The number of common K-mers in x_1 and x_2 is the product of the number of K-mers in x_1 and x_2. Obviously, there cannot be any K-mers in one string if this string contains less than K characters.

$$compute\_K\_mers\_two\_chars(x_1, x_2, [a, b]) =$$
$$= max(0, min(z_a, z_b, K-1, z_b + z_a - K + 1))$$

This means that the number of common K-mers of our two strings is $z_a + z_b - K + 1$ at maximum. There cannot be more than $K-1$ K-mers, as each K-mers has to contain a and b at least once. Furthermore, the number of matching K-mers is upper bounded by z_a and z_b, i.e. the minimum of the number of occurrences of a and b in x_1 and x_2. □

3.4 Scalability

Apart from a fast theoretical runtime, our feature string kernels allow handling of larger databases as well. As every kernel function induces a metric

$$d(p_1, p_2) = \sqrt{K(p_1, p_1) + K(p_2, p_2) - 2K(p_1, p_2)},$$

where p_1 and p_2 are objects in feature space, we can define a metric on voxel objects based on our 3D string kernels. We can then use this metric to create an M-tree [6] index structure for efficient storage and access of large datasets of 3D objects that do not fit into main memory.

4. EXPERIMENTS

In this section, we will present the results of our experimental evaluation. In particular we compared the feature-based techniques described in section 1.1 with our new string kernel based approach presented in this article. We will give a detailed overview of the settings used in our evaluations first. This includes a brief description of the considered datasets and the parameters used in combination with the different techniques. We will also describe how we measured the ability to correctly classify 3D objects, before giving the actual results in the following subsection.

4.1 Experimental Setting

We will first explain which objects of the NTU [5] and the PSB [26] dataset we used for the classification experiments and describe the parameters used for the description of the 3D objects.

4.1.1 Datasets

The NTU dataset consists of 1833 3D models that have been manually assigned to classes of similar functionality by the publishers of this dataset. This resulted in 47 classes with in total 549 models. The remaining models were labeled with 'miscellaneous'. We discarded those unlabeled objects and made the 549 remaining objects with corresponding class labels our first benchmark dataset. It will be referred to as NTUALL in the remainder of this paper. We chose the elements of the 3 largest classes out of the 47 classes as our test set based on the NTU dataset. In the following, we will refer to this reduced dataset as the NTU dataset. In particular this smaller set consists of 67 objects assigned to the class 'car', 52 objects assigned to the class 'chair', and 64 objects labeled with 'plane'. Hence the total number of objects in this NTU subset is 183; it will be denoted as NTU183 in the following.

[1]If we use min or max operators with more than 2 arguments, this simply means we are selecting the minimum or maximum element out of a set of given terms.

The second test set we created is based on the PSB set that contains 1814 3D objects collected from the World Wide Web. Along with the models, a hierarchical classification system is provided. We decided to use the leaves of the classification system as class labels and so the set is partitioned into 161 disjoint classes. For experiments, we employed two datasets derived from PSB. The first one comprises all objects from PSB, 1814 in total from 161 classes, and will be called PSBALL from now on.

Additionally, to create a dataset with few classes and many instances per class, we scanned for classes with approximately 50 or more members. This resulted in a smaller test set with 4 classes where 100 objects are members of the class 'fighter-jet-airplane', 100 models are assigned to the class 'human-biped', 51 objects are marked with 'potted-plant-plant', and 51 more objects can be found in the class 'rectangular-table'. In total, the second test set based on the PSB set consequently contains 302 elements. From now on, this dataset will be referred to as PSB302 set.

4.1.2 Methods and Parameters

In this section, we shortly describe our choice of parameters crucial to the analyzed methods. In order to determine the best possible parameter settings for each voxel resolution for each method we performed a number of classification experiments for each method on the NTUALL dataset. The so derived optimal parameters for a certain method and a certain voxel resolution were also used for the classification experiments on the other datasets. In the following we describe the exact parameter values for each of the methods we compared our 3D string kernel method with.

4.1.2.1 Voxelization.

The authors of the different techniques use varying voxel resolutions. At first we set the voxel resolution for all compared approaches to 15, i.e. 15^3 black and white voxels are used for the representation of each three dimensional object. To study the effect of different voxelizations on classification accuracy, we also examined voxel resolutions of 20 and 25.

4.1.2.2 Shell Model.

We varied the number of space partitioning shells and learned that 7 is the best choice for the number of shells for the voxel resolution of 15. Objects represented by 20^3 voxels are best described with 9 shells and for a voxel resolution of 25 we used 11 shells. In the following we use the abbreviations 'SM7', 'SM9', and 'SM11' to denote the Shell Model based on different numbers of shells.

4.1.2.3 Spherical Harmonics.

The most important parameter of this method is the number of spherical functions defined on the voxel grid. For 15^3 voxels, 7 spherical functions led to the best classification results on our benchmark dataset NTUALL. For both, 20^3 and 25^3 voxels the optimal setting was 9 spherical functions. So we use the notation 'SH7' and 'SH9' to refer to this method.

Following the original work, we computed the s_l sums for the first 16 frequencies.

4.1.2.4 Eigenvalue Model.

The classification results of this similarity model differ with the number of cubical partitions that divide the 3D objects. We found that 5^3 partitions is the best suited value for this parameter for all the considered voxel resolutions. In the following we consequently refer to this model as 'EM5'.

4.1.2.5 Grid D2.

An important parameter of this feature calculation method is the number of calculated distances. In the original work this number equals the number of filled and not-filled voxels, so we randomly selected 15^3, 20^3, and 25^3 pairs of voxels and calculated their Euclidean distances. In the following this method is denoted by 'GD2'.

4.1.3 Performance Comparison

We will now outline how we measured the classification ability of the different approaches.

4.1.3.1 Classification.

The idea of the feature-based methods is to find a suitable mapping of voxel grids to the feature space so that the distance between two points in the feature space reflects the human sense of geometrical or functional dissimilarity of the associated voxel grids.

While distances represent dissimilarity, kernel values are a measure of similarity. It is, however, unproblematic to turn kernel values of pairs of input data into distances. For example, this can be reached via

$$distance(p_1, p_2) = \sqrt{K(p_1, p_1) + K(p_2, p_2) - 2K(p_1, p_2)}$$

where K is a kernel function and p_1 and p_2 are input data.

After defining distances for both feature vector- and kernel-based approaches, we were able to apply a Nearest Neighbor classifier to our datasets. Nearest Neighbor classifiers predict the class label of a test data point t by finding the data point in the training set with minimum distance to t, i.e. the Nearest Neighbor of t. The class label of t is then predicted to be the label of its Nearest Neighbor in the training set.

4.1.3.2 Classification Accuracy.

We performed Leave-One-Out Classification on all datasets. One object was used a test set, while all other objects belong to the training set. This is repeated until each object has been part of the test set exactly once. The classification accuracy we report is the mean of these iterations.

4.2 Results

4.2.1 Comparison of Three Kernels on NTUALL

First, we tested our three versions of the 3D string kernel on the NTUALL dataset at a voxel resolution of 15. For the lrc kernel and the k-spectrum kernel we examined values of
k in $\{10, 20, 30, \ldots, 1200\}$, where 1200 is the maximum length of a feature string in NTUALL. We report results of these experiments in Figure 4 for addition and in Figure 5 for pointwise multiplication of x-,y-, and z-kernels.
While the lrc kernel and the k-spectrum kernel yield different classification accuracies for different k values, the all k-mer kernel obviously is independent of the choice for k. Note that we nonetheless included the classification results of the all k-mer kernel in the above mentioned figures so that the results can more easily be compared.

The lrc kernel outperforms both the all k-mer kernel and the k-spectrum kernel, both for pointwise multiplication and addition. Note that the lrc kernel converges to the all k-mer kernel as k increases. k-spectrum kernel and lrc kernel reach their best result for k in 10 to 50, indicating that longer matches are either seldom or mislead the classifier. In this experiment, the addition of kernels gave slightly better results than pointwise multiplication, although differences in accuracy are small, especially for small choices of k.

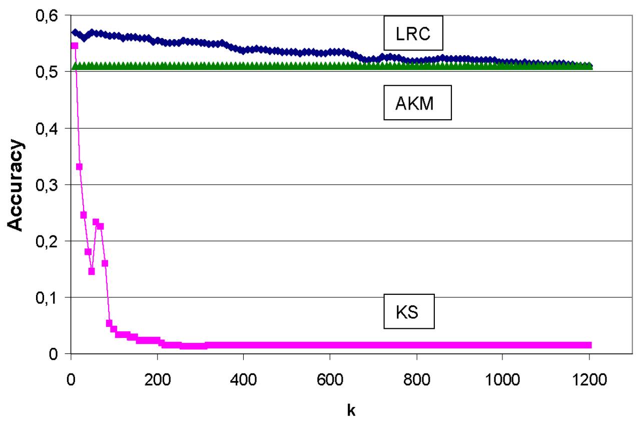

Figure 4: The Limited Range Correlation (LRC) Kernel, the k-Spectrum Kernel (KS), and the all k-mer Kernel (AKM) (Sum of x-,y-,z-Kernels) .

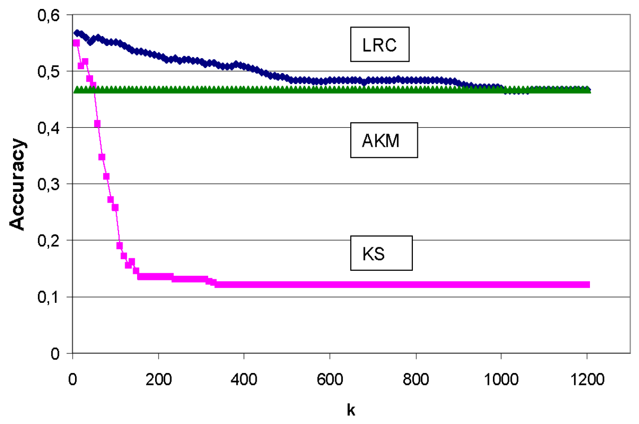

Figure 5: The Limited Range Correlation (LRC) Kernel, the k-Spectrum Kernel (KS), and the all k-mer Kernel (AKM) (Pointwise Product of x-,y-,z-Kernels).

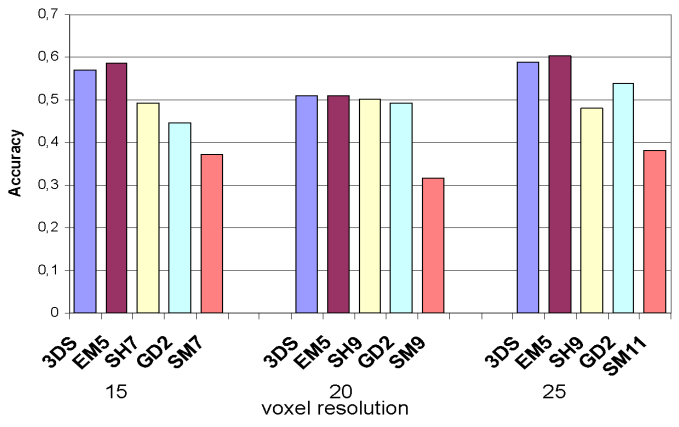

Figure 6: NTUALL Optimized-Kernel (3DS) in Comparison to other Techniques on the NTUALL Dataset.

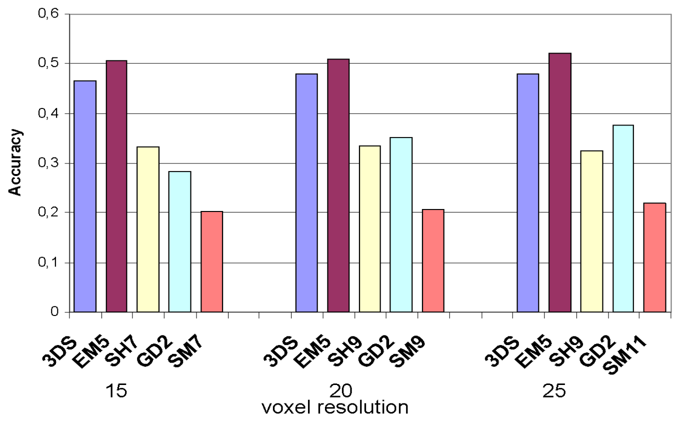

Figure 7: NTUALL Optimized-Kernel (3DS) in Comparison to other Techniques on the PSBALL Dataset.

4.2.2 Benchmark Test on NTUALL

Afterwards we compared the results of the best 3D string kernel so far, the lrc kernel that had outperformed all other string kernels on NTUALL, to four state-of-the-art techniques (see Section 1.1). This comparison was conducted on NTUALL for all three resolutions 15, 20, and 25. We optimized parameters for each of the resolutions for each method individually. We report results in Figure 6.

The lrc kernel outperforms 3 out of 4 competing state-of-the art methods. Only the EM method reaches slightly better accuracies for resolutions of 15 and 25, and the same accuracy for 20 voxel resolution.

4.2.3 Benchmark Test on PSBALL and PSB302

We then applied all methods to the datasets PSB302 and PSBALL in Leave-One-Out-Validation. We used the parameterization optimized for NTUALL unchanged on these datasets, to avoid artificially good results by overfitting of parameters. Results for PSBALL are given in Figure 7 and for PSB302 in Figure 8.

Again, the 3D string kernel reaches better results than all methods except for EM which is slightly better on the PSBALL dataset across different voxelizations. On PSB302, the 3D string kernel yields the best result. It outperforms all other method with a margin of 4% for 15 voxels, 3% for 20 voxels, and 2% for 25 voxels.

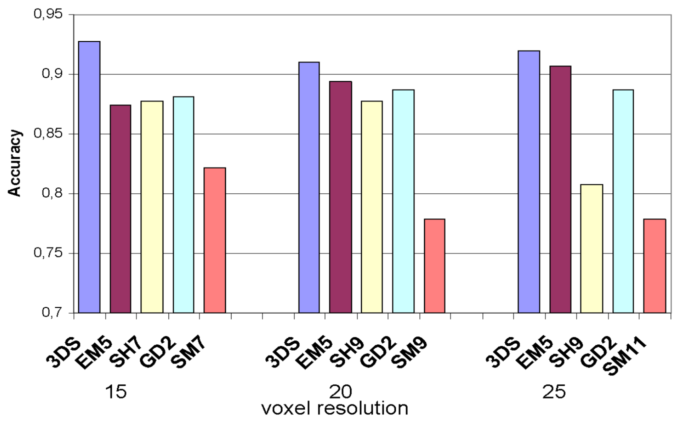

Figure 8: NTUALL Optimized-Kernel (3DS) in Comparison to other Techniques on the PSB302 Dataset.

On all three datasets, the results achieved by the 3D string kernel are robust with respect to different voxelizations and do not change significantly.

5. DISCUSSION

We have presented a novel approach to 3D object classification based on feature strings and 3D string kernels as measure of similarity. We have shown that the feature string extraction can be performed in linear runtime with respect to the number of voxels. For all of our string kernels, we have given proofs and algorithms that they can be computed in time linear to the voxel grid resolution on our feature strings. Our algorithm is therefore suited for 3D object classification on large datasets. If datasets do not fit into main memory, our kernel can be readily combined with an M-tree, as it induces a metric on 3D objects. In comparison experiments with four state-of-the-art techniques on two published benchmark datasets, our approach was always best or second best.

The good performance of our method might be caused by both the feature strings and the kernel we use. Our feature strings describe the distribution of filled voxels over the voxel grid in three dimensions separately. Our kernel gives high similarity scores to two 3D objects if the voxel distributions in corresponding dimensions are similar.

One could also interpret our feature strings and string kernels as a histogram-based approach. In fact, we create histograms of the distribution of voxels in each dimension and then represent these histograms by strings instead of vectors. Unlike a histogram vector approach that calculates some distance between corresponding bins in two histograms, our approach allows to measure similarity between neighboring bins as well. This is made possible by looking at k-mers made up of two or more different characters.

We have defined three different kernels on our 3D feature strings: one that examines all common k-mers (all k-mer kernel) in two strings, one that considers all common k-mers up to a fixed k (lrc kernel), and a third that looks at k-mers of one fixed k-mer length only (k-spectrum kernel). Among these, the lrc kernel outperforms all others in all of our experiments.

The lrc-kernel yields better results than the all k-mer kernel because it does not overweight long matches, whereas a match that is one character longer gets twice the weight by the all k-mer kernel. Therefore the all k-mer kernel considers one long match much stronger than several shorter matches. According to our experiments, this is not a good choice in 3D object feature strings classification and many shorter matches seem to indicate similarity.

The opposite of the all k-mer kernel, the k-spectrum kernel that considers k-mers of one fixed length only, is also not as successful as the lrc kernel in our experimental evaluation. The k-spectrum kernel considers object similar with many common fixed-length k-mers. However, in real-world datasets length of k-mers that are important for 3D object classification seem to differ in length.

As the lrc kernel considers all k-mers up to a fixed length only, it does not overweight long strings as the all k-mer kernel. As it is not limited to one k-mer length as the k-spectrum, it takes several potentially important k-mers lengths in account. These characteristics might explain its superior experimental performance.

In our benchmark experiments with state-of-the-art methods, the lrc kernel is successful and always comes in first or second. It is as accurate as the EM method on NTUALL at 15 voxel resolution. On the remaining resolutions for NTUALL and on PSBALL, the lrc kernel performs slightly worse than EM, but better than all other state-of-the-art approaches. On PSB302, our method outperforms all other techniques, at all resolutions.

A possible explanation for these results is the fact that the 3D string kernel finds objects that are very similar to each other as it examines the location of each voxel individually, whereas EM is better suited for remote similarity detection as it summarizes the object characteristics in terms of eigenvalues of the covariance matrix. PSBALL and NTUALL comprise many classes and many of those contain a few objects only, whereas PSB302 contains 302 objects from 4 classes. If we are dealing with many small classes, remote similarity might be important, whereas few classes with many objects increase the chance that a query object is very similar to another object in its class. This would explain why the lrc kernel outperforms all others on PSB302, whereas EM is best on PSBALL and NTUALL.

6. CONCLUSIONS AND OUTLOOK

The above experiments show that our 3D string kernel approach is comparable to the best competitor in classification accuracy.

As an additional advantage, our feature strings can be stored and processed in compressed format. The memory requirement for this compressed format is at maximum twice the resolution of the voxel grid, namely for each character and the number of its occurrences. This low memory requirement of our approach makes scaling to large datasets even easier.

Furthermore, our feature string description of 3D objects has attractive invariance properties: It is invariant with respect to scaling and translation. However, it is not rotational-invariant and requires - like many state-of-the-art approaches - a prior uniform orientation of the objects.

A further attractive feature of our feature string approach is the fact that all recent advances on string classification in machine learning can be transferred to the task of 3D object classification. For example, a technique using suffix trees that speeds up Support Vector Machine training on large sets of strings has recently been developed [27]. We could use this method on our feature strings as well and make them even more attractive for large 3D object dataset classification.

A possibility to enhance the accuracy of all k-mer kernel could be to assign smaller weights to longer strings, to avoid overweighting of longer matches. Furthermore, methods for learning weights of individual k-mers based on multiple kernel learning have recently been developed [28]. That could be exploited in our setting to detect k-mers that are especially important for correct classification of 3D objects. We plan to explore this weight learning on 3D object feature strings in future research.

7. REFERENCES

[1] M. Ankerst, G. Kastenmüller, H.-P. Kriegel, and T. Seidl. 3D Shape Histograms for Similarity Search and Classification in Spatial Databases. In *Proceedings of the 6th International Symposium on Spatial Databases*, pages 207–226, 1999.

[2] A. Ben-Hur, D. Horn, H. Siegelmann, and V. Vapnik. A support vector method for hierarchical clustering. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 367–373. MIT Press, 2001.

[3] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

[4] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[5] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On Visual Similarity Based 3D Model Retrieval. In *Computer Graphics Forum*, volume 22, pages 223–232, 2003.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, 1997.

[7] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik. Support vector regression machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 155–161, Cambridge, MA, 1997. MIT Press.

[8] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A search engine for 3d models. In *ACM Transactions on Graphics*, pages 83–105, 2003.

[9] D. Haussler. Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99 - 10, Computer Science Department, UC Santa Cruz, 1999.

[10] T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 487–493. MIT Press, 1999.

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

[12] I. Jolliffe. *Principal Component Analysis.* Springer, 1986.

[13] A. Kaufman. An Algorithm for 3D Scan-Conversion of Polygons. In *Proc. Eurographics*, pages 197–208, 1987.

[14] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *Symposium on Geometry Processing*, pages 167–175, 2003.

[15] H.-P. Kriegel, P. Kröger, Z. Mashael, M. Pfeifle, M. Pötke, and T. Seidl. Effective Similarity Search on Voxelized CAD Objects. In *Proc. 8th Int. Conf. on Database Systems for Advanced Applications, Kyoto, Japan*, pages 27–36, 2003.

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

[17] C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for SVM protein classification. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, volume 15, Cambridge, MA, 2002. MIT Press.

[18] L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *RECOMB*, pages 225–232, 2002.

[19] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller. Fisher discriminant analysis with kernels. In Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX*, pages 41–48. IEEE, 1999.

[20] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Matching 3D Models with Shape Distributions. In *International Conference on Shape Modeling and Applications*, pages 154–166, 2001.

[21] G. Ratsch, S. Sonnenburg, and B. Scholkopf. Rase: recognition of alternatively spliced exons in c.elegans. *Bioinformatics*, 21 Suppl 1:i369–i377, Jun 2005.

[22] B. Schölkopf. *Support Vector Learning*. R. Oldenbourg Verlag, Munich, 1997. Download: http://www.kernel-machines.org.

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

[24] B. Schölkopf, A. J. Smola, and K.-R. Müller. Kernel principal component analysis. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - -Support Vector Learning*, pages 327–352. MIT Press, Cambridge, MA, 1999.

[25] J.-L. Shih, C.-H. Lee, and J. Wang. 3D object retrieval system based on grid D2. *Electronics Letters*, 41(4), 2005.

[26] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. The Princeton Shape Benchmark. In *Shape Modeling International, Genova, Italy*, 2004.

[27] S. Sonnenburg, G. Rätsch, and B. Schölkopf. Large scale genomic sequence svm classifiers. In *International Conference on Machine Learning*, 2005.

[28] S. Sonnenburg, G. Rätsch, and B. Schölkopf. Large scale genomic sequence svm classifiers. In *International Conference on Machine Learning*, 2005.

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

[30] S. V. N. Vishwanathan and A. J. Smola. Fast kernels for string and tree matching. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 569–576. MIT Press, Cambridge, MA, 2003.

[31] C. Watkins. Dynamic alignment kernels. CSD-TR-98- 11, Royal Holloway, University of London, Egham, Surrey, UK, 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, Sept. 2000.