A Survey of (pseudo)-Distance Functions for Structured Data *

V. Estruch-Gregori C. Ferri-Ramírez J. Hernández-Orallo
M.J. Ramírez-Quintana

Departament Sistemes Informàtics i Computació

Universitat Politècnica de València

{vestruch, cferri, jorallo, mramirez}@dsic.upv.es

May 31, 2005

Abstract

Learning from structured data is becoming increasingly important. Besides the well-known approaches which deals directly with complex data representation (inductive logic programming and multi relational data mining), recently new techniques have been proposed by upgrading propositional learning algorithms. Focusing on distance-based methods, they are extended by incorporating similarity functions defined over structured domains, for instance a k-NN algorithm solving a graph classification problem. Since a measure between objects is the essential component for this kind of methods, this paper consists of a brief survey about some of the recent similarity functions defined over common structured data (lists, sets, terms, etc.).

keywords: ILP, distance-based methods, kernel and distance functions, structured data.

1 Introduction

Most real world data has no natural representation by means of tuples of pairs of attribute-value being the value a nominal or a numerical data. Some recent challenges in machine learning, such as web and text mining, mollecular classification, etc., demand a more powerful and expresive instance representation language. For example, imagine how much structures a web page is. Several categories and a variety of information can be distinguished: the title, the content, the multimedia information, the links, etc. Thus, a web site might not have a trivial description as a collection of nominal or numerical atributes. However, a more readable and intuitive description way can be achieved by allowing a data-typed representation language. Namely, chain of atoms and list of words can be mapped into lists of data, a bag of words or a multiple-instance problem [1] can be modelled by means of sets or multi-sets, and a mollecule or a web-site topology (internal connections) clearly corresponds to a graph.

Inductive logic programming (ILP) and multi-relational data mining (MRDM) have been considered as the classical approaches in order to handle structured instances directly. Nevertheless, during the last years, an increasing interest of upgrading some propositional data mining methods has appeared. Thus, we can find kernel and distance functions, and probalistic distributions defined over structured domains in order to adapt kernel-based (e.g. SVM), distance-based (e.g. k-Means) and probabilistic-based (e.g. Naive Bayes) methods to these complex scenarios.

In this paper we review some of the recent similarity functions (concretely distance and pseudo-distance functions) defined over structured data types. These (pseudo-)distances have been employed by well-known learning al-

*This work has been partially supported by the EU (FEDER), the Spanish MEC, under grant TIN-2004-7943-C04-02, and the Generalitat Valenciana (MEDIM).

Actas del III Taller Nacional de Minería de Datos y Aprendizaje, TAMIDA2005, pp.233-242
ISBN: 84-9732-449-8 **© 2005 Los autores, Thomson**

gorithms (SVM, k-Means, DBDT) and tested in different learning problems, as we remark. Additionally, several (pseudo-)distances are studied for the same data type because each one mesures the proximity notion in some particular way, and sometimes, this quantification becomes crucial for the perfomance of the learning method.

The outline of the paper is as follows. Section 2 introduces different structured data representation language. A collection of (pseudo-)distances are explained in Section 3. Finally, some conclusions are given.

2 Representing Structured Data

Unlike non-structured data, where instances are usually represented by attribute-value pairs, there does not exist a standard notation to represent structured information. However, several benefits can be derived from a unified notation, for instance, a cost reduction in the preprocessing task (parsing, reading and filtering). So far, the main steps in this line rely on logic formalisms. Next, we sketch some of these proposals: ISP (Individuals, Structural predicates and Properties), relational and term based representations [4].

- ISP representation: It is based on flattened Prolog and consists of three parts declaring the individuals (instances), the structural predicates (structure of the instances) and the properties (nominal or numerical of the structured components). This representation layout is feature-extraction oriented. Let us show it by means of an example:

  Example 1 *We need to represent a set of tuples like this, $\langle molecule, weight\rangle$, which describe a molecule and its correspondent weight. Consider the concrete case: $\langle H_2O\,, 32.99\rangle$:*

```
% INDIVIDUAL ; m1 is the molec. id.
molecule(m1,label_class)
% STRUCTURAL PREDICATES
% ai is the atom id.
mol_to_atom(m1,a1)
mol_to_atom(m1,a2)
mol_to_atom(m1,a3)
% PROPERTIES
weight(m1,32.99)
symbol(a1,H)
symbol(a2,H)
symbol(a3,O)
```

  The features could be defined from the structural and the property predicates.

- Relational representation: It is based on relational data bases. Although under certain restrictions it is equivalent to ISP, however, it leads to more compact descriptions.

  Example 2 *Relational representation of $\langle H_2O\,, 32.99\rangle$:*

```
molecule(m1,32.995)
symbol(a1,H,m1)
symbol(a2,H,m1)
symbol(a3,O,m1)
```

- Term-based representation: It is a declarative description based on typed logic languages. Some ILP researchers advocate the use of typed data in order to exploit the information provided by the type to prune the hypothesis space. This notation leads to self-contained objects.

  Example 3 *Term-based representation of $\langle H_2O\,, 32.99\rangle$:*

```
% DECLARING DATA TYPES
type Molecule = ({Atom},weight,class)
type Atom = (Symbol)
% DECLARING INSTANCES
Water = ({Oxygen,Oxygen,Hydrogen},
         32.99,class)
Atom Oxygen = O, Hydrongen = H
```

3 Some Recent (pseudo-)Distances over Structured Data

In this section we review some novel (pseudo-)distances introduced over structured domain which have been embedded in distance-based classification or clustering algorithms.clustering problems over complex data.

The pseudo-distance definitions exposed in the second subsection come from kernel definitions. They can easily be deduced, as we will see, thanks to the formal relationship existing between the concept of inner product (scalar product) and the concept of distance.

3.1 Distance Functions

Distances over lists and trees are omitted at this point because the widest-used definitions are still dated from the last sixties and seventies decades [20, 21, 19]. However some advances have been made as for other data types such as sets and first-order terms.

- *Sets:* Finite collections of items appear frequently in computer science problems. So, measuring the similarity between two sets of items have many useful applications not only in machine learning but also in other areas such as computational geometry.

  Maybe, the most intuitive distance between two finite sets is given by the cardinality of the symmetric difference between them. Although it is is easily computable, some more sophisticated definitions are required in order to improve the performance of the learning methods.

  Two interesting distances from a practical point of view, explained at this point are the Hausdorff distance [1][14] and a matching-based distance [16].

  At first sight, the *Hausdorff distance* possesses a weird and an artificial formulation, but as years went by, it turned out to play an important role not only in fractal geometry but also as similarity measure for sets in classification and clustering problems (multiple-instance problem). It is defined as follows,

  Definition 1 *Given X a set of points, and d, a metric between points, the Hausdorff metric $d_h : 2^X \times 2^X \rightarrow \mathbf{R}^+ \cup \{0\}$, then*

  $$d_h(A,B) = max \left\{ \begin{array}{l} max_{a \in A}(A(a)) \\ max_{b \in B}(B(b)) \end{array} \right.$$

  where $A(a) = min\{d(a,b) : b \in B\}$ and $B(b) = min\{d(a,b) : a \in A\}$.

  In a few words, the distance between the sets A and B is given by the maximum distance of a set to the nearest point in the another set. This makes it very sensitive to outlying points from A or B. For instance, let us consider $A = \{1,2,3\}$ and $B = \{4,5,20\}$, where 20 is some large distance away from every point of A. In this case, applying Definition 1, we obtain that the distance between A and B is 17 and, it is basically determined by the outlying value. In [17] the Hausdorff distance has been tested over known structured data sets (multi-instance data sets) achieving quite competitive results. In [22] an extension of this distance is proposed in order to reduce the noise sensibility.

  The *matched-based distance* is a formal instantiation of a novel and attractive schema introduced by [2]. This schema consists of two equations. First,

  Definition 2 *Given two finite sets A and B and a known distance d defined over the items belonging to A and B, then*

  $$d(r,A,B) = \left[\sum_{(x,y) \in r)} d(x,y) \right] + M \cdot \frac{|B - r(A)| + |A - r^{-1}(B)|}{2}$$

  where r is a mapping from A to B.

[1] Introduced by the German mathematician Felix Hausdorff.

It means that one sums the distances of the pair of elements in r and adds a penalty $M/2$ for each element not belonging to r. The constant M stands for the maximal possible distance between two elements from A and B. The second equation is just employed to define the distance (d^m) between A and B.

Definition 3 *Let A and B be two sets, the distance between A and B is defined*

$$d^m(A,B) = min_{r \in m(A,B)} d(r,A,B)$$

where $m(A,B)$ (simplifying the original notation) is a family of mappings (surjections, fair surjections or linkings are the family of mappings considered by the authors) between A and B.

This schema leads to a semi-distance function. The matched-distance definition follows the schema above but forcing the mappings to be a matching (a mapping between A and B is a matching if each element of A is associated to at most one element of B.). Adding this new condition, the authors show that the semi-distance d^m turns into a distance, where in this case m denotes all possible matchings between A and B [17].

Example 4 *Given the sets $A = \{1,2,3\}$ and $B = \{1,4\}$ and consider the distance between numbers as the absolute difference. Applying Definition 3, we obtain that the optimal mapping is $r = \{(1,1),(3,4)\}$, $M = 4-1 = 3$, and $d^m(A,B) = 0+1+3 \cdot \frac{1+1}{2} = 4$.*

The authors report some experimental results as well (multi-instance and biochemical data sets), showing that the matched-distance performs better than the semi-distance functions. Additionally, these results are comparable w.r.t those achieved by special-purpose methods.

- *Terms and atoms*: Before introducing the distances defined over terms and atoms, we briefly recall some logic terminology. The set of terms T is built from the set of variables V and the set of functors F. An additional set A denoting the the set of predicates symbols is needed. A variable is a term, and a f/n a functor symbol of arity n and $t_1,\ldots,t_n$ terms, then $f(t_1,\ldots,t_n)$ is a term. Similarly, if p/n is a predicate symbol of arity n then $p(t_1,\ldots,t_n)$ is an atom. Given two atoms a_1 and a_2 we will say that a_1 is more general if there exists a substitution σ (a function which instantiates the variables of an atom a by terms and it is denoted by $a\sigma$) such that $a_2 = a_1\sigma$. Given two atoms a_1 and a_2 the least general atom w.r.t. a_1 and a_2 is denoted by $lgg(a_1,a_2)$.

  Initially some ad-hoc similarity functions to handle first order terms were introduced [3] but they do not preserve some intuitive properties about the proximity between two atoms. For instance, given the atoms $p(a)$, $p(b)$ and $p(X)$, $p(a)$ should be closer to $p(X)$ than to $p(b)$ because $p(a)$ is a instance of $p(X)$. Consider also the case when $p(X)$ and $p(Y)$, then the distance should be zero.

  It sounds reasonably that before measuring how far two first order rules are, initially we need to quantifies the separation between two atoms. The works in this line concentrate in showing a feasible similarity function over atoms.

  In order to define an adequate distance, an incremental strategy is followed in [15]. First, a distance between to ground atoms is defined, and then, this distance is extended to atoms containing variables using the Hausdorff distance between sets and the notion of Herbrand space.

  Definition 4 *Let E be the set of ground terms and atoms, and let $a_1 = p(s_1,\ldots,s_n)$ and $a_2 = q(t_1,\ldots,t_n)$ be two*

items belonging to E, then

$$d(a_1, a_2) = \begin{cases} 0, \text{ if } a_1 = a_2 \\ 1, \text{ if } p \neq q \\ \frac{1}{2n} \sum_{1=1}^{n} d(s_i, t_i) \end{cases}$$

The following example illustrate how this distance works.

Example 5 *Given the ground atoms* $a_1 = p(f(a), g(a, b))$ *and* $a_2 = g(f(b), b)$ *then*

$$d(a_1, a_2) = \frac{1}{4}(d(f(a), f(b)) + d(g(a, b), b))$$

$$= \frac{1}{4}(\frac{1}{2} + 1) = \frac{3}{8}$$

Now, If we want to calculate the distance between no-ground atoms, it is necessary to compute the Herbrand base of each atom and calculate the Hausdorff distance between them. However, the last definition yields not desirable results when non-ground atoms are involved, in order to overcome that, a new distance is proposed in [17]. The authors of this work present an original distance schema based on a previous semi-distance definition in [9]. This distance is on agreement with the intuitive proximity relations between atoms informally mentioned at the beginning of this point. The mentioned distance between two atoms (not necessarily ground) is expressed as a pair of integer values (F, V) reflecting the differences of them w.r.t. their *lgg*. The distance definition is based on an auxiliary function $s(a) = (F, V)$, called *size*, which reflects the structure of the atom a. Roughly speaking, F is a function which counts the number of predicate and function symbols occurring in a, and the function V returns the sum of the squared frequency of appearance of each variable in a. More formally,

Definition 5 *Given* a_1 *and* a_2 *two atoms, then*

$$d(a_1, a_2) = [s(a_1) - s(lgg(a_1, a_2))] +$$

$$[s(a_2) - s(lgg(a_1, a_2))]$$

Example 6 *Consider the atoms* $a_1 = p(a, b)$ *and* $a_2 = p(b, b)$. *The distance* $d(a_1, a_2)$ *is calculated as follows. First, we compute the lgg of both atoms, that is,* $lgg(a_1, a_2) = p(X, b)$ *and then, we measure each atom structure by means of the function size:* $s(a_1) = s(a_2) = (3, 0)$ *and* $s(lgg(a_1, a_2)) = (2, 1^2)$. *Finally, the distance between* a_1 *and* a_2 *is*

$$d(a_1, a_2) = [(3, 0) - (2, 1)] +$$

$$[(3, 0) - (2, 1)] = (1, -1) + (1, -1)$$

$$= (2, -2)$$

Note that with this definition of distance the proximity relation (how far two atoms are) is not as intuitive as in a conventional metric space where its associated distance returns only a positive real number (and not a pair of values). For this reason, the authors introduce a total order relation over the pair of values which allows to specify a proximity notion. Given two ordered pairs $A = (F_1, V_1)$ and $B = (F_2, V_2)$, $A < B$ iff $F_1 < F_2$ or $F_1 = F_2$ and $V_1 < V_2$ (lexicographic order). Let us illustrate how this order relation can be used to determine the proximity among atoms.

Example 7 *Let* $a_1 = p(a, b)$, $a_2 = p(a, a)$ *and* $a_3 = p(b, b)$ *be three atoms. Since* $d(a_1, a_3) = (2, -2)$ *and* $d(a_2, a_3) = (4, -8)$ *we can conclude according to the order relation that* a_3 *is closer to* a_1 *than to* a_2.

3.2 Kernel-Based (pseudo-)Distances

Some learning techniques need the structured instances to be previously represented in a more adequate space. This space is called the *feature space* and the correspondent *feature transformation* is referred by σ. Commonly, the feature transformation leads to an attribute-value representation language in

which a complex object is downgraded to a vector of nominal and numerical values. As we said in the introduction, a structured object may not have a natural representation as a vector of values. Therefore, the efforts in this way lie on defining transformations which try to reflect the semantic of the original representation data. However, this task is not straightforward.

Feature transformations are implicitly used by kernel methods (SVM, Gaussian processes and kernel principal analysis). These introduce a special function, called kernel, embedding a feature transformation. Theoretically speaking, a kernel (denoted by $k(\cdot,\cdot)$) is just an application which computes the inner product between two elements previously mapped into their correspondent feature space. More formally, $k(x,y) = \langle \sigma(x), \sigma(y) \rangle$ where $\langle \cdot, \cdot \rangle$ stands for the inner product [2]. However, the attractiveness of a kernel function comes from the fact it can be directly applied without explicitly computing σ.

As for distance-based methods, a kernel function offers a new and rich possibility in order to define a (pseudo-)distance (induced (pseudo-)distance) by exploiting the existing formal relationship between a distance and a kernel. Let us see:

Definition 6 *Let $k : X \times X \to R$ be a positive definite kernel on X and let $x,y \in X$. Then, $d_k(x,y) = \sqrt{k(x,x) + k(y,y) - 2k(x,y)}$ is the distance induced by k.*

The expression above is obtained by considering that $d_k(x,y)^2 = \langle \sigma(x) - \sigma(y), \sigma(x) - \sigma(y) \rangle$ such as it is pointed out by [6]. The function d_k is a distance if σ is injective, otherwise, it is a pseudo-distance. Although we are interested in (pseudo-)distances, in what follows, only the kernel definition will be given since automatically the (pseudo-)distance function can be derived.

As we proceed in the subsection above, next, we will sketch some kernel functions for well-known structured data. Some of them have been tested in real-world applications.

[2] For the sake of consistency the feature space must be endowed of a Hilbert structure.

- *Sets and multi-sets*: A valid kernel for two finite sets A and B can intuitively be defined by considering the cardinality of the set $A \cap B$. Although this definition is rather simple, the underlying ideas, which lead to it (convolution kernels), are slightly more sophisticated [8]. The attractiveness of this initial kernel function comes from the fact it is a particular case of a more general kernel definition [7]. This one was specially introduced to address the multiple-instance problem, proving in the same work the appropriateness of the kernel for this concrete task. The definition is as follows

  Definition 7 *Let U be a set of items and let X and Y be two finite subsets of U, then*

  $$k_{MI} = \sum_{x \in X, y \in Y} k_I^p(x,y),$$

  where k_I is a kernel function defined over the elements of U and p is a positive real number.

  Note that this definition is valid for multi-sets as well. We only have to treat those repeated items as different ones. By doing that, if the item x is n times in X, then the term $k_I^p(x,\cdot)$ will appear n times in the sum.

  Example 8 *Trivially, if we let $k_I = k_\delta(x,y)$ where $k_\delta(x,y) = 1$ if $x = y$ or 0 otherwise (discrete kernel), then the equation above turns into the symmetric difference between sets: $k_{MI}(A,B) = |A \cap B|$.*

  The kernel function 7 was embedded in a SVM and tested in a drug prediction problem being k_I the Gaussian kernel. The performance achieved was great even w.r.t. specific-purpose algorithms.

- *Basic terms*: A high-order instance description language is presented in [12].

This formalism allows to express any structured data where an instance is represented by a closed term. Thus, a term collects all the information relative to an instance in a single object. Each term is built from "indivisible" basic items: function types (to represent data types such as sets and multi-sets), product types (to construct fixed-size tuples) and constructor types (to introduce constant symbols, numbers, lists, trees, etc.).

Example 9 *The list* $L = [A, B, B]$ *is represented by the term* $A : B : B : []$ *where the symbols* $:$ *and* $[]$ *are the list and the empty list constructor respectively. The multi-set* $S = \{A, B, B\}$ *corresponds to the function expression (in* λ *notation):* $S = \lambda x$ if $x = A$ then 1, else if $x = B$ then 2, otherwise 0.

The kernel function for closed terms is as follows.

Definition 8 *Let* s *and* t *be two closed terms.*

If $s = C\, s_1 \cdots s_n$ *and* $t = D\, t_1 \cdots t_n$ *are basic terms from instantiating type constructors (lists, trees, etc.), then*

$$k(s,t) \begin{cases} k_T(C,D) \text{ if } C \neq D \\ k_T(C,D) \sum_{i=1}^{n} k(s_i, t_i) \end{cases}$$

if s *and* t *correspond to* λ*-expressions (sets or multi-sets), then*

$$k(s,t) = \sum_{u \in s, v \in t} k(V(s\,u), V(t\,u)) k(u,v)$$

where $V(s\,u)$ *computes how many times the item* u *appears in* s *(*t*).*

Finally, if $s = s_1 \cdots s_n$ *and* $t = t_1 \cdots t_n$ *are terms built from a tuple constructor, then*

$$k(s,t) = \sum_{i=1}^{n} k(s_i, t_i)$$

Let us illustrate these ideas by means of next example.

Example 10 *Given the lists* $L_1 = A : B : C : []$ *and* $L_2 = A : D : []$*, then*

$$\begin{aligned} k(L_1, L_2) &= k(:,:) + k(A,A) + \\ &\quad k(B : C : [], D : []) \\ &= 1 + 1 + k(:,:) + \\ &\quad k(B,D) + k(C : [], []) \\ &= 3 \end{aligned}$$

Now, let us consider the multi-sets $M_1 = \{A, B, B\}$ *and* $M_2 = \{A, B\}$*,*

$$\begin{aligned} k(M_1, M_2) &= k(1,1)k(A,A) + \\ &\quad k(1,1)k(A,B) + \\ &\quad k(2,1)K(B,A) \\ &\quad + k(2,1)k(B,B) \\ &= 1 + 0 + 0 + 2 \\ &= 3 \end{aligned}$$

Although this syntax-driven kernel performs really well for known data sets (biochemical and spatial clustering data sets), it turns out to be excessively general. Next, some more specific kernels are slightly described.

- *Lists (sequences)*: In what follows, we consider that a sequence is built up from a finite set of items (Σ). Roughly speaking, the underlying idea of calculating the inner product between two strings is based on counting, in a some way, the number of common subsequences. According to [13], the problem is formalised in a infinite dimensional space, where each dimension corresponds to a word belonging to Σ^*. Then, a string s is mapped into an array of real-coefficient polynomial where each polynomial encodes what words from Σ^* are a subsequence of s (not necessarily contiguous) and how frequent and long these subsequences are. Let us see the example below:

Example 11 *Consider the alphabet* $\Sigma = \{a, b, c\}$ *and the words* $w_1 = aac$ *and* $w_2 =$

ac. First, we obtain the subsequences in w_1 (u_{1i}) and in w_2 (u_{2i}). Then, we associate them the correspondent polynomial (see Table 1).

	subsequence	polynomial
u_{11}	a	2λ
u_{12}	c	λ
u_{13}	aa	λ^2
u_{14}	ac	$\lambda^3 + \lambda^2$
u_{15}	aac	λ^3
u_{21}	a	λ
u_{22}	b	λ
u_{23}	ab	λ^2

Table 1: Subsequences of the words w_1 (u_{1i}) and w_2 (u_{2i}).

The parameter λ is powdered as times as the length of a subsequence is in w_i (including the gaps) whereas its coefficient represents how many times a subsequence appears in w_i. Then, the scalar product of w_1 and w_2 is computed using the polynomials associated to the common subsequences of w_1 and w_2. The common subsequences are $\{a, c, ac\}$. Organising all these information into a vector, we have that

$$\begin{array}{rcl} \sigma(w_1) & = & \left(\begin{array}{ccc} 2\lambda & \lambda & \lambda^3 + \lambda^2 \end{array} \right) \\ \sigma(w_2) & = & \left(\begin{array}{ccc} \lambda & \lambda & \lambda^2 \end{array} \right) \end{array}$$

and applying the inner product definition,

$$\begin{array}{rcl} k(w_1, w_2) & = & \langle \sigma(w_1), \sigma(w_2) \rangle \\ & = & \lambda^2(3 + \lambda^2 + \lambda^3) \end{array}$$

The parameter λ is the so-called decay factor. It quantifies how important a common subsequence is (generally $\lambda \leq 1$). So, setting $\lambda = 1/2$ and by Definition 6

$$\begin{array}{rcl} d(w_1, w_2) & = & \sqrt{1.39 + 0.56 - 1.57} \\ & \backsimeq & 0.61 \end{array}$$

The calculus of the kernel can be expressed by the following definition,

Definition 9 *Let Σ a finite set of symbols and let w_1 and w_2 be two words from Σ^*, then*

$$k(w_1, w_2) = \sum_{\forall u \in \Sigma^*} \phi_u(w_1)\phi_u(w_2)$$

$$= \sum_{u \in w_1 \cap w_2} (f_u(w_1) + f_u(w_2))\lambda^{l_u}$$

where $l_u = l_u(w_1) + l_u(w_2)$ ($l_u(w_i)$ is the length of u in w_i), $\phi_u(w_i)$ computes the polynomial associated to the subsequence u in w_i, $f_u(w_i)$ returns the appearance frequency of u in w_i, and $w_1 \cap w_2$ stands for the common subsequences.

This latter kernel has been used for text classification problems. Other kernel definitions consider that the subsequences u must be contiguous in w_i. In [5] more kernel functions for string data are explained, as well as plenty of references to related works.

- *Graphs*: As a graph is a really highly expressive and complex data structure, defining a kernel for this data type based on counting shared sub-graphs drives to a NP-hard problem [18]. Some attempts have been made in order to define competitive kernels (expressive and less computational expensive) by considering, as a similarity measure, common particular sub-graphs (paths, walks, random walks, trees) rather than all possible sub-graphs. Each of this approach leads to a different kernel definition. For brevity, we illustrate one based on common walks which employs the formula $k(G_1, G_2) = \sum_{\forall g} \lambda(s(g))$ to compute the kernel between the graphs G_1 and G_2 (g denotes a common walk, $s(g)$ corresponds to the size/length of g and $\lambda(\cdot)$ is the weight function).

Example 12 *Given the directed and labelled graphs $G_1 = \{(a, b), (b, c), (a, d)\}$ and $G_2 = \{(a, b), (a, d)\}$. The*

common walks of G_1 *and* G_2 *are* $\{\{(a,b)\},\{(a,c)\}\}$*, and letting* $s(g) = length(g)$ *and* $\lambda = 1/s(g)!$*, then*

$$\begin{aligned} k(G_1, G_2) &= \lambda((a,b)) + \lambda((a,c)) \\ &= \tfrac{1}{1!} + \tfrac{1}{1!} = 2 \end{aligned}$$

More elaborated and detailed kernel definitions can be found in [11, 10]. Kernels over graphs have been successfully employed in several tasks such as natural language processing, digital image interpretation, classification and clustering of chemical compounds, etc.

4 Conclusions

Along with ILP and MRDM, new approaches to handle structured data have emerged by upgrading well-known propositional techniques. For instance, distance-based method are extended by defining similarity functions over structured domains. In this paper we particularly described some useful (pseudo-)distances introduced for concrete data types and, in a succinct way, their application area.

References

[1] T. Dietterich, R. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1–2):31–71, 1997.

[2] T. Eiter and H. Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):109–133, 1997.

[3] W. Emde and D. Wettschereck. Relational instance-based learning. In *Fachgruppentreffen der Fachgruppe Machinelles Lernen der GI, FGML'95*. University of Dortmund, 1995.

[4] P.A. Flach and N. Lachiche. Naive bayesian classification of structured data. *Machine Learning*, 57:233–269, 2004.

[5] T. Gärtner. A survey of kernels for structured data. *SIGKDD Explor. Newsl.*, 5(1):49–58, 2003.

[6] T. Gartner, J. W. Lloyd, and P. A. Flach. Kernels and distances for structured data. *Machine Learning*, 57, 2004.

[7] Thomas Gartner, Peter A. Flach, Adam Kowalczyk, and Alex J. Smola. Multi-instance kernels. In Claude Sammut and Achim Hoffmann, editors, *Proceedings of the 19th International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, July 2002.

[8] David Haussler. Convolution kernels on discrete structure. Technical Report UCSC-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA, July 1999.

[9] A. Hutchinson. Metrics on terms and clauses. In Maarten van Someren and Gerhard Widmer, editors, *Proceedings of the 9th European Conference on Machine Learning*, volume 1224 of *LNAI*, pages 138–145, Berlin, April 24–24 1997. Springer.

[10] H. Kashima, K. Tsuda, and A. Inokuchi. Kernels for graphs, 2004.

[11] J. Lafferty and R. Imre Kondor. Diffusion kernels on graphs and other discrete input spaces, 2002.

[12] J. W. Lloyd. *Logic for learning: learning comprehensible theories from structured data*. Springer-Verlag, 2003.

[13] H. Lodhi, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 563–569. MIT Press, 2001.

[14] B. Mendelson. *Introduction to Topology*. Dover Pubn., 3rd edition, July 1990.

[15] S. Nienhuys-Cheng. Distance between Herbrand interpretations: A measure for approximations to a target concept. In Nada Lavrač and Sašo Džeroski, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *LNAI*, pages 213–226, Berlin, September 17–20 1997. Springer.

[16] J. Ramon and M. Bruynooghe. A polynomial time computable metric between point sets. *Acta Informatica*, 37(10):765–780, August 2001.

[17] J. Ramon, M. Bruynooghe, and W. Van Laer. Distance measures between atoms. In *CompulogNet Area Meeting on Computational Logic and Machine Learing*, pages 35–41. University of Manchester, UK, 1998.

[18] J. Ramon and T. Gärtner. Expressivity versus efficiency of grapgh kernels. In Luc De Raedt and Takashi Washio, editors, *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences (MGTS-2003)*, pages 65–74. ECML/PKDD'03 workshop proceedings, 2003.

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

[20] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26(3), 1979.

[21] R. Wagner and M. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[22] J. Wang and J.D. Zucker. Solving multiple-instance problem: A lazy learning approach. In *Proceedings of the 17th International Conference on Machine Learning*, pages 1119–1125, 2000.