Similarity Functions for Structured Data. An Application to Decision Trees

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

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València, Spain
{vestruch, cferri, jorallo, mramirez}@dsic.upv.es

Resumen

Learning from structured data is becoming increasingly important. Besides the well-known approaches which deal directly with complex data representations (inductive logic programming and multi-relational data mining), new techniques have been recently proposed by upgrading propositional learning algorithms. Focusing on distance-based methods, these techniques 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 starts with a description of some of the recent similarity functions defined over common structured data (lists, sets, terms, etc.). However, many of the most common classification techniques, such as decision tree learning, are not distance-based methods or cannot be directly adapted to be so (as kernel methods and neural networks have been adapted). In this work, we extend decision trees to use any kind of similarity function. The method is inspired by "centre splitting", which constructs decision trees by defining splits based on the distance to two or more centroids. We include an experimental analysis with both propositional data and complex data. Apart from the advantages of the new proposed method, it can be used as an example of how other partition-based methods can be adapted to deal with distances and, hence, with structured data.

Keywords: Distance-based Methods, Decision Trees, ILP, 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 with only nominal or numerical values. Some recent challenges in machine learning, such as web and text mining, molecular classification, etc., demand a more powerful and expressive instance representation language. For example, imagine how much structured 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 attributes. However, a more readable and intuitive description way can be achieved by considering a data-typed representation language. Namely, a chain of atoms and a list of words can be mapped into lists of data, a bag of words or a multiple-instance problem [5] can be modelled by means of sets or multi-sets, and a molecule 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 probabilistic 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 algorithms (SVM, k-Means) and tested in different learning problems, as we remark. Additionally, several (pseudo-)distances are studied for the same data type because each of them measures the proximity notion in some particular way, and sometimes, this quantification becomes crucial for the performance of the learning method.

Since the application of these measures is direct for many learning algorithms (kNN, SVM, k-means or even neural networks), as we have said, in this paper we address the problem of using all these measures in decision trees, so extending decision tree learning to cope with complex data.

We present a new algorithm for decision tree learning which performs the splits based on the distance measures defined for the different data types which are used for the problem definition. The main features of our algorithm are: First, we compute distances attribute by attribute instead of using distances between whole examples. Secondly, we also use attribute centroids (prototypes) from the set of examples in the training dataset, instead of creating new centroids (which might not correspond to valid objects). As a result, we get a generalisation of decision trees, capable of dealing with any (simple and complex) data type.

We include the definition of the algorithm and experiments on both non-structured and structured data.

The outline of the paper is as follows. A collection of (pseudo-)distances is explained in Section 2. Section 3 discusses how decision trees can be adapted to use distances. Section 4 presents the algorithm and Section 5 includes several experiments. Finally, some conclusions are given.

2. 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 for handling 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.

2.1. Distance functions

Distances over lists and trees are omitted at this point because the most widely used definitions are still dated from the last sixties and seventies [28, 30, 32]. However some advances have been made as for other data types such as sets and first-order terms.

2.1.1. 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 easily computable, some more sophisticated definitions are required in order to improve the performance of the learning methods.

At this point we will explain two interesting distances from a practical point of view: the Hausdorff distance [1][20] and a matching-based distance [24].

At first sight, the *Hausdorff distance* was introduced with a weird and an artificial formulation,

[1]Introduced by the German mathematician Felix Hausdorff.

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

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

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 2.1, we obtain that the distance between A and B is 17 and, it is basically determined by the outlying value. In [25] the Hausdorff distance has been tested over known structured data sets (multi-instance data sets) achieving quite competitive results. In [33] 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 [6]. This schema consists of two equations. First,

Definition 2.2 *Given two finite sets A and B and a known distance d defined over the items belonging to A and B, then*

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

where r is a mapping from A to B.

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 2.3 *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[2] 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.). By adding this new condition, the authors show that the semi-distance d^m becomes a distance, where in this case m denotes all possible matchings between A and B [25].

Example 2.4 *Given the sets $A = \{1,2,3\}$ and $B = \{1,4\}$ and consider the distance between numbers as the absolute difference. Applying Definition 2.3, 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.

2.1.2. Terms and atoms

Before introducing the distances defined over terms and atoms, we briefly recall some logic programming terminology. For any concept which is not explicitely defined the reader may refer [2, 18]. The set of terms T is built from the set of variables V and the set of functors Σ. An additional set Π denoting the set of predicates symbols is needed. Variables are represented by capital letters whereas functor and predicate symbols are

[2]Surjections, fair surjections or linkings are the family of mappings considered by the authors.

represented by small letters. A variable is a term, and if f/n is a functor symbol f of arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term. Similarly, if p/n is a predicate symbol p of arity n and $t_1, \ldots, t_n$ are terms 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 than a_2 if there exists a substitution θ such that $a_2 = a_1\theta$. Given two atoms a_1 and a_2 the least general generalisation of a_1 and a_2 (denoted by $lgg(a_1, a_2)$) [22] is the minimal atom which is more general than a_1 and a_2.

Initially some ad-hoc similarity functions to handle first order terms were introduced [7] 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 an instance of $p(X)$. Consider also the case with $p(X)$ and $p(Y)$, then the distance should be zero.

In order to define an adequate distance, an incremental strategy is followed in [21]. First, a distance between two 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 Universe.

Definition 2.5 *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 the distance d between a_1 and a_2 is defined as*

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

The following example illustrates how this distance works.

Example 2.6 *Given the ground atoms $a_1 = p(f(a), g(a, b))$ and $a_2 = p(f(b), b)$ 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 non-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 non desirable results when non-ground atoms are involved; in order to overcome that, a new distance is proposed in [25]. The authors of this work present an original distance schema based on a previous semi-distance definition in [14]. 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 2.7 *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 2.8 *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 2.9 *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.*

2.2. 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 data representation. 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 [3]. 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 2.10 *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 [11]. 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 proceeded 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.

2.2.1. 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 [13]. The attractiveness of this initial kernel function comes from the fact it is a particular case of a more general kernel definition [12]. 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 2.11 *Let U be a set of items and let A and B be two finite subsets of U, then*

$$k_{MI}(A, B) = \sum_{a\in A, b\in B} k_I^p(a, b),$$

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 a is n times in A, then the term $k_I^p(a, \cdot)$ will appear n times in the sum.

Example 2.12 *Trivially, if we let $k_I = k_\delta(a, b)$ where $k_\delta(a, b) = 1$ if $a = b$ 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 2.11 was embedded in a SVM and tested in a drug prediction problem being k_I the Gaussian kernel. The performance achieved was better than specific-purpose algorithms.

2.2.2. 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

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

counting, in a some way, the number of common subsequences. According to [19], the problem is formalised in an 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 2.13 *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 2.13).*

	subsequence	*ocurrences*	*polynomial*
u_{11}	a	$\underline{a}ac,\ a\underline{a}c$	2λ
u_{12}	c	$aa\underline{c}$	λ
u_{13}	aa	$\underline{aa}c$	λ^2
u_{14}	ac	$\underline{a\mathbf{a}c},\ a\underline{ac}$	$\lambda^3 + \lambda^2$
u_{15}	aac	$\underline{aac}$	λ^3
u_{21}	a	$\underline{a}c$	λ
u_{22}	c	$a\underline{c}$	λ
u_{23}	ac	$\underline{ac}$	λ^2

Table 2.13: **Subsequences of the words w_1 (u_{1i}) and w_2 (u_{2i}).**

The parameter λ is powered 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 informations 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 2.10

$$d(w_1, w_2) = \sqrt{1{,}39 + 0{,}56 - 1{,}57} \simeq 0{,}61$$

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

Definition 2.14 *Let Σ a finite set of symbols and let w_1 and w_2 be two words from Σ^*, then*

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

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

where $\phi_u(w_i)$ computes the polynomial associated to the subsequence u in w_i, $f_u(w_i)$ returns the appearance frequency of u in w_i, $l_u(w_i)$ is the length 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 [10] more kernel functions for string data are explained, as well as plenty of references to related works.

2.2.3. 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 [26]. Some attempts have been made in order to define "competitive kernels" (expressive and less computationally expensive) by considering, as a similarity measure, common particular sub-graphs (paths, walks, random walks, trees) rather than all possible sub-graphs. Each of these approaches 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 2.15 *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 $\{\{(a, b)\}, \{(a, d)\}\}$, and letting $s(g) = length(g)$ and $\lambda = 1/s(g)!$, then*

$$k(G_1, G_2) = \lambda((a, b)) + \lambda((a, c)) = \frac{1}{1!} + \frac{1}{1!} = 2$$

More elaborated and detailed kernel definitions can be found in [15, 17]. 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.

3. Distances and Decision Trees

After the previous account on several distance and pseudo-distance measures, we could apply any of them to any distance-based method. A more challenging problem would be to apply these measures to other non distance-based methods. In this section we show how one of the most popular non distance-based methods, decision tree learning, can be adapted to use any of these similarity metrics.

Our proposal is based on the use of distances in a quite similar way as centre splitting does. Centre splitting [31] is a distance-based technique used for the classification of numerical attribute-valued examples, and can be seen as a divide-and-conquer extension of a linear discriminant. Basically, this method consists of dividing the metric evidence space in different regions, where each region is represented by a special point called centre. In every iteration of the process, a centre is calculated for every different class presented in an area. The coordinates of the centre may match to an existing example or not. Then, every example is associated to its nearest centre. Since the first split usually generates impure regions, the process is iterated for each impure region. The process stops when the obtained regions are totally pure, i.e., the examples placed in one region belong to the same class.

While the method is efficient and flexible, models extracted by this method are only intelligible for problems with two dimensions, where regions can be represented graphically.

This happens because the centre splitting method manages every example as a whole. This appreciation leads us to propose a decision tree inference strategy where partitions are made only taking into account one attribute at a time. In this way centroids are computed considering only the values of one attribute. This fact allows the centre splitting and decision tree learning techniques to be joined in an elegant way. We call this approach distance-based decision tree learning.

For this purpose, a similarity space is associated to every attribute. The split is performed in a similar way, but only considering the distances and class distribution of the selected attribute. As usual, an attribute is selected depending on a heuristic function (gain ratio, GINI index, etc.).

The result of this adaptation of centre splitting is not very different from classical decision trees, when attributes are either nominal and numeric. The interesting point comes when we have structured data types for some attributes. In order to handle these types appropriately, we have to determine how the centres are computed. If we want one centre per class, the first idea would be to compute the centre which minimises the distance to the rest of examples for each class. It is important, however, to note that for many data types, the "centre" might not necessarily be inside the original data type. For instance, given a type of lists of constants, the centre of the set $\{[a,a],[a,a,a],[a,a,a,a,a]\}$ is a list containing 3.33 a's, which does not belong to the original data type. Consequently, it is much more reasonable to find a representative example, prototype or median. In the previous case, the most appropriate value would be the list $[a,a,a]$.

A similar idea of combining a distance-based approach with a method for tree induction was presented in [4] for clustering. In this case each node (and leaf) of the tree corresponds to a cluster. A clustering decision tree is a first-order logical decision tree in the sense that the test in each node is a conjunction of literals as in the TILDE system [3]. Consequently, this kind of trees can deal with multi-relational data. As in the centre splitting technique, a distance measure is used for computing the distance between examples. Finally, the splitting criterion of the tree is adapted to select in each node the test that maximizes the distance between the resulting cluster prototypes and its children. Therefore, this approach differs from our proposal in many ways: the test in the nodes of the tree are not defined in terms of metric conditions, the distance function is defined by comparing whole examples and not attribute by attribute, and the prototypes do not need to match with an original example.

4. Algorithm

With this idea of attribute prototypes, we can extend decision tree learning to handle complex structures and data types. Before introducing the algorithm, we require some notation. First, $Attr_{x_j}(e)$ returns the jth attribute of example e. Similarly, $Class(e)$ returns the class of example e. Additionally, two important functions to reduce the complexity of computing the distances are needed: $Values(S, x_j)$ returns the set of different values for attribute x_j in the sample S, and $Card(v, S, x_j)$ returns the number of occurrences of value v in the attribute x_j in the sample S, namely, $|\{e \in S : Attr_{x_j}(e) = v\}|$. The function $dist(x, y)$ computes the distances of values x and y, attributes which are of the same type. It is assumed that these distances are pre-calculated.

The next algorithm is the main procedure of our method. The inputs are a training dataset and a constant m, which is a parameter that limits the number of children per split (if we set $m = 2$ we only have binary partitions):

```
PROCEDURE DBDT($S$, $m$);
INPUT: $S$ is a set of examples of the form:
$(x_1, \ldots, x_n), n \geq 1$, $m$ is the maximum # of children
per node.
BEGIN
  $C \leftarrow \{Class(e) : e \in S\}$
  If $|C| \geq 2$ Then
   For each attribute $x_j$:
     If $|Values(x_j, S)| \geq 2$ Then
      $ProtList \leftarrow$ Compute_Prototypes$(x_j, S, m, C)$.
      If $Size(ProtList) > 1$ Then
       $Split_j \leftarrow \emptyset$
       For i ← 1 to $length(ProtList)$
         $\hat{S}_i \leftarrow \{e \in S : i =$ Attracts$(e, ProtList, x_j)\}$
         // $\hat{S}_i$ has the examples attracted by $i$
         $Split_j = Split_j \cup \hat{S}_i$
         // Add a new child
       End For
      End If
     End If
   End For
   $BestSplit = Argmax_{Split_j}(Optimality(Split_j))$
   // GainRatio, GINI, ...
   For each set $S_k$ in $BestSplit$
     DBDT$(S_k, m)$
   End For
  End If
END
```

The previous algorithm is just a typical decision tree learning algorithm which, in this case, determines, for each attribute, a ranked list of prototypes, which will lead to a set of children for each node. The main difference with classical decision tree learners lies in two functions: `Compute_Prototypes` and `Attracts`, which we describe next.

The function `Attracts` just determines which prototype is assigned with a new example. Consequently, this function is not only applied when learning the decision tree but also when using the decision tree to predict new examples, i.e. as the decision rule. Several options are possible here (e.g. considering the density or not), but the `Attracts` function below implements a very simple one. It just returns the closest prototype. In case of a tie, it returns the rightmost prototype.

```
FUNCTION Attracts$(e, ProtList, x_j)$;
INPUT: $e$ an example, $Protlist$ a ranked list of
prototypes, $x_j$ a chosen attribute.
RETURNS: Index of prototype that attracts $e$.
BEGIN
 For $i \leftarrow 1$ to $length(ProtList)$
   $v \leftarrow attr_{x_j}(e)$
   If $\forall k > i,\ dist(attr_{x_j}(ProtList[i]), v) <$
     $dist(attr_{x_j}(ProtList[k]), v)$ Then Return $i$
   End If
 End For
END
```

Finally, the function `Compute_Prototypes` is the most important one. This function can be performed in many different ways. Below, we show one of these possibilities, which just selects the best prototype (in the way that the distances to the prototype for the elements of the same class are minimised), removes its class and the value of the attribute and looks for the next best prototype of a different class and value for the attribute, and so on, until the limit m or the number of classes or attribute values is reached.

```
FUNCTION Compute_Prototypes$(x_j, S, m, C)$;
// Computes up to $m$ prototypes for attribute $x_j$
over dataset $S$.
INPUT: $x_j$ is the attribute, $S$ is the dataset,
$m$ is the maximum # of prototypes, $C$ is the set
of classes.
RETURNS: A ranked list of prototypes.
BEGIN
 For each class $c \in C$:
   $S_c \leftarrow \{e \in S : class(e) = c\}$. // Ex. of class $c$
   If $S_c \neq \emptyset$ Then
     $V_c \leftarrow Values(x_j, S_c)$
     For each element $v \in V_c$
       $MeanDistance_c[v] \leftarrow \frac{\sum_{i \in V_c} dist(v,i) \times Card(i, S_c, x_j)}{|S_c|}$
```

```
      End For
    End If
  End For
  $UV \leftarrow \emptyset$ // Values for the attribute used
  $ProtList \leftarrow \emptyset$ // List of prototypes
  $RC \leftarrow C$ // Remaining classes
  $Values \leftarrow |Values(x_j, S)|$ // Different Values in $S$
  $\#Prots \leftarrow min(|C|, m, Values)$
  // #Prototypes of the split
  For $k \leftarrow 1$ to $\#Prots$
  // best prototype not already chosen.
    $BestProt = argmin_e\{$
    $MeanDistance_c[Attr_{x_j}(e)]\}_{c \in RC, e \in S, Attr_{x_j}(e) \notin UV}$
    // If tie, select the one from the class with
more examples.
    $RC \leftarrow RC - Class(BestProt)$
    $ProtList \leftarrow append(ProtList, BestProt)$
    $UV \leftarrow UV \cup Attr_{x_j}(BestProt)$
  End For
  Return $ProtList$
END
```

The previous function is the core of the algorithm and it is (jointly with the splitting criterion) the point where efficiency is crucial. The complexity of the previous function is mostly determined by the first part (the first for), which is in $\mathcal{O}(|C| \cdot |V_c| \cdot |V_c|)$ where V_c is the number of different values for the attribute. Although this order is cubic, the cardinality of V_c is, in general, small, especially for nominal and for some structured attributes. In the case of numeric data types and types with many different possible values (e.g. sets), the distances can be pre-calculated before calling the procedure DBDT. The second part of the function Compute_Prototypes (the second for) has an $argmin$ which, overall, can be done in $\mathcal{O}(\#Prots \cdot |C| \cdot |V_c|)$, which, in general, is lower than the first part.

It is important to note that distances are computed between attribute values and not between examples. This issue is not only important for obtaining attribute-based decision trees but also, as we have seen, is crucial for efficiency.

5. Experimental evaluation

The aim of this section is to illustrate that the algorithm presented here can be used as a general purpose machine learning algorithm for both non-structured and structured problems. All the results have been obtained using 10x10 fold cross validation.

5.1. Implementation remarks

The algorithm presented in section 4 has been implemented using the WEKA libraries. Several distance and pseudo-distance functions have been implemented. For nominal and numerical data, the employed distances are the well-known discrete and absolute value difference respectively. Regarding structured data types, we have used several distances defined in Section 3.

Data Set	$J48$	DBDT
Balance Scale	78,4	76,0
B. cancer wdbc	93,4	92,1
B. cancer wisc	94,3	94,3
B. cancer wpbc	72,1	69,5
Cmc	49,5	46,6
Dermatology	93,4	92,6
Diabetes	73,6	68,6
Ecoli	81,5	76,2
Haber. Breast	70,0	66,2
Heart dis.	51,2	48,1
House votes	94,5	93,9
Ionosphere	89,8	88,7
Iris	94,7	93,8
Letter	87,96	86,3
Liver	65,0	61,1
Monks 1	95,1	96,1
Monks 2	62,0	96,5
Monks 3	98,5	97,6
Tic Tac Toe	78,9	74,8
Pima	73,7	68,6
Post operative	58,8	61,7
Wave form	76,39	74,3
Sonar	73,16	75,9
Mean	75,3	75,1

Table 1: **Classification accuracy (%) on UCI data sets.**

5.2. Non-structured problems

The first evaluation has been performed over twenty three attribute-value data sets extracted from the UCI repository, all the attributes being nominal or numerical data types. The results have been compared to $J48$[4] with its default parameters in WEKA. Pruning is disabled for $J48$ and DBDT but a minimum of two examples per node are forced in each node (see Table 1).

As we can observe, the average results for DBDT are not far from the results of J48. However, if we leave aside the data sets with only nominal attributes (monsk1, monks2, monsk3 and tic tac toe), the achieved average decreases about 1,8 points. The explanation could be that $J4{,}8$ implements

[4] $J48$ is the implementation in the Weka classifier package of the well-known decision tree induction algorithm $C4{,}5$ [23].

sophisticated and well-studied techniques for numerical data in front of the general treatment carried out by DBDT (the boundary point is just the median between the two prototypes). More elaborated (density or distribution based) ways of computing the boundary point could be developed once the prototypes are determined. Nevertheless the results are reasonably good for a decision tree learner that treats nominal and numerical attributes in the same way.

5.3. Structured problems

In order to evaluate our approach with our main aim, structured data, we will compare the results of DBDT with other techniques working on structured data such as the mutagenesis and the musk data sets. Both are two biochemical real-world data and have been widely used as a test for several structured data learning proposals.

5.3.1. The Mutagenesis data set

This problem aims at distinguishing between mutagenic and non-mutagenic compounds [29]. We used the "regression friendly" version. Several settings of the experiment have been studied in order to investigate how the way in which the information is handled acts on the final result. The set of experiments can be divided into two groups. The first experiments (see Table 2)are focused on studying the performance achieved by the algorithm with a non-structured representation (propositionalisation). The second series of experiments deals with a more structured approach (see Tables 3 and 4). In this last setting, we begin with an initial structured representation of compounds, and then, non-structured attributes are incorporated.

The first setting treats only with the numerical attributes *lumo* and *logp* which represent some chemical properties of the molecule (R1). The Boolean indicators *ind1* and *inda*, which codes some key structural information of a molecule, are added in a second representation (R2). Finally, the atomic composition of the molecule is captured, in a naive way, by incorporating as numerical attributes as the number of different possible atoms that exist in the problem (R3). Each of these new attributes contains the frequency of this atom in the molecule.

	lumo and logp (R1)	Indicators (R2)	Atoms (R3)
J48	79,9	89,2	88,5
DBDT	77,1	84,2	84,9

Table 2: **Accuracy (%) performed on non-structured version of Mutagenesis.**

The structural approach deals with a structured attribute which reflects the structural character of the data. This attribute is a set of 3−tuple where each tuple stores information relative to the atom (R4). Then, the non-structured information is used. First the attributes *lumo* and *logp* (R5), and finally, the Boolean indicators (R6). Two distance functions have been employeed. On the one hand, a Hausdorff distance for sets and on the other hand a pseudo-distance derived from the typical kernel defintion for sets. In both cases, a normalized distance and a normalized kernel for items have been used.

Set of tuples (R4)		+ l. and l. (R5)		+ Indic. (R6)	
sd-K	Hausd.	sd-k	Hausd.	sd-K	Hausd.
82,6	80,5	80,9	79,5	83,9	84,3

Table 3: **Accuracy performed by DBDT algorithm. *sd-K* means pseudo-distance derived from a kernel.**

We repeated the process but using more complex structured attributes. In this experiment (see Table 5.3.1) we capture not only the physical properties of each atom but also the information relative to its bonds. The structured attribute turns to be a 4−tuple, being the last component a multiset which models the bonds.

Set of 4-tuples	+ lumo and logp	+ Indicators
83,1	80,6	84,1

Table 4: **Accuracy performed by DBDT algorithm.**

In the last experiment (see Table 5), along with the set of tuples, a new structured attribute is added. This attribute is a set of 2−depth trees where each tree represents the atoms directly linked to one given atom.

Set of tuples and trees	+ lumo and logp	+ Indicators
84,6	81,0	84,1

Table 5: **Accuracy performed by DBDT algorithm.**

Some interesting observations can be extracted from the last experiments. When the numerical attributes are added, in general, the algorithm performs worse. But the accuracy ratio is again improved when the indicators are used. This is so because the indicators code information about the number of rings in a molecule. Thus, the indicators complement the structural information about the molecule contained in the structured attribute. However, we can notice that this improvement is not attained when more complex structured attributes are used. In fact, $J48$ seems to be a really good option for mutagenesis. DBDT performs similarly when a pure non-structured version and a pure structured version of the data set is given. Consequently, DBDT can compute feasible classifiers without using relevant information, such as the indicators attributes, provided by an expert.

In order to extract more reliable conclusions about the performance achieved by DBDT, some other known results obtained by ILP algorithms using different knowledge theories from a simple theory ($B0$) to another one more complex ($B4$) are summarised in the Table 6.

Alg.	$B0$	$B1$	$B2$	$B3$	$B4$
Progol	79,0	86,0	86,0	88,0	89,0
Progol	76,0	81,0	83,0	88,0	??
FOIL	61,0	61,0	83,0	82,0	??
Tilde	75,0	79,0	85,0	86,0	??
MRDTL	67,0	87,0	88,0	??	??

Table 6: **Accuracy (%) performed by ILP systems on friendly mutagenesis.**[5]

The accuracy achieved by the different ILP algorithms is clearly improved as the background knowledge is populated with more useful predicates. Let us remark that our method does not employ background knowledge, then our system should be compared to the B0 column results.

5.3.2. The Musk data set

This dataset might be one of the most representative examples of the so-called multiple-instance problem (MI). This problem can be seen as a complex extension of the attribute-value classification problem, in the sense that the instances are described by a set of vectors (tuples) and any of these vectors could be responsible for the classification of the instance. Thus, the difficulty relies on identifying the characteristic vector of the set. Although this concept was coined by [5], other similar problems were considered earlier in pattern recognition [16].

Id.	Algorithm	Accu.(%) 10CV
1	Iter. APR	92,4
2	GFS positive APR	83,7
3	Citat.-kNN	92,4
4	MI-SVM	89,0
5	mi-SVM	84,0
6	MI-nn	88
7	RELIC	84,3
8	IBL-Hausdorff	82,0
9	KeS	81,0
10	nn	75,0
11	C4.5	68,5
12	DBDT-Hausdorff	81,6
13	DBDT-psedo-distance	75,7

Table 7: **Accuracy (%) on musk1 data.**

Several techniques have been proposed to cope with the multi-instance problem. On the one hand, we find algorithms specially designed or adapted for this purpose like APR [5], and some upgraded techniques such as nearest neighbors (Bayesian-Knn and Citation-KNN [33]), neural nets (MI-NN [27]), decision tree learners (RELIC), support vector machine (MI-SVM and mi-svm [1]). On the other hand, we distinguish classical approaches, which ignore the MI setting during training (NeuralNets, C4.5) but not during testing, or general-purpose approaches able to face the MI problem without changing its learning scheme (IBL[24], KeS[9]). The DBDT algorithm is situated in this latter group, as a general-purpose system.

In our setting, a molecule will be represented by only one structured attribute, a set of tuples. In this case, we derive the pseudo-distance function from a MI-specific kernel proposed in [8]. The MI-kernel is a specifi-purpose kernel which fits the kernel for sets schema defined in Definition 2.11 , where k_i is in this special case a Gaussian kernel. The Hausdorff distance for sets was tested as well but it performed worse than the MI pseudo-distance did.

Table 7 is organised as follows. The first seven rows contains the accuracy of some of the specific-MI tools reported in the literature. The next four values are performed by general-purpose algorithms. We must stand out that the kernel method (KeS) achieves an optimal result (86,4 %) when a parameter is fixed to an optimal value. The last two results correspond to algorithms ignoring the MI setting in training. The DBDT algorithm performs 81,6 % and 75,7 % for the Hausdorff distance and the pseudo-distance functions respectively. Thus, the experiments show that our approach performs competitively to general pur-

[5]The *??* stands for an unknown result. The table has two entries for the PROGOL system because different results can be found in the literature.

pose methods.

6. Conclusions

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

Then we have introduced a new algorithm that combines decision tree learning and distance based learning. The key of our learning method is that it creates the split of the decision tree by using centroids of only one attribute, instead of other methods (centre splitting) that employ the whole examples to obtain the centroids. In this way we are able to induce models, i.e. decision trees which use the distance associated to the data type of the attribute selected by the splitting criterion. Consequently, the algorithm can deal with whichever data type that has an associate distance metric. Additionally, there is no need of nested distances for very complex types such as lists of trees, tuples of trees and graphs, etc. For common types, the user can choose the default distance metric or can introduce more elaborated distance metrics for each specific problem in order to produce better results. On the other hand, since the distance is computed between attributes instead of examples the efficiency of decision trees is partially preserved. Moreover, we pre-compute the distance matrices speeding up the learning process.

We have included an experimental evaluation where we study the performance of our method over propositional problems extracted from the UCI repository, and with datasets that contain structured attributes. Although in both cases there are other approaches that obtain better results, our system shows that can be employed in all these scenarios with a satisfactory performance. Probably, the inclusion of well-known refinements that improve decision trees (pruning, smoothing, etc.) could significantly increase the performance of our system. As a more general result of our work, the ideas can be extended to other partition or condition based machine learning or data mining methods, such as rule learning, association rules, ...

As current work we are studying how to convert the distance-based splits for structured data types into pattern-based splits, in order to make them as comprehensible as classical decision trees. We are also working on other distances for numerical attributes (e.g. quadratic) and for sets [24], and improvements on the function `attracts`, in order to take densities into account.

Acknowledgments

This work has been partially supported by the EU (FEDER), the Spanish MEC, under grant TIN-2004-7943-C04-02, the Generalitat Valenciana, under grant GV06-301, the UPV under grant TAMAT and the ICT for EU-India Cross Cultural Dissemination project ALA/95/23/2003/077-054.

Referencias

[1] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. *Advances in Neural Information Processing Systems*, 15, 2003.

[2] K. R. Apt. *From Logic Programming to Prolog.* Prentice-Hall, 1997.

[3] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artif. Intell.*, 101(1-2):285–297, 1998.

[4] Hendrik Blockeel, Luc De Raedt, and Jan Ramon. Top-down induction of clustering trees. *CoRR*, cs.LG/0011032, 2000.

[5] 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.

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

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

[8] T. Gärtner, P. Flach, A. Kowalczyk, and A. Smola. Multi-instance kernels. In *Proc. of the 19th Int. Conf. on Machine Learning*, pages 179–186, 2002.

[9] T. Gärtner, P. Flach, and J. Lloyd. Kernels and distances for structured data. *Journal of Machine Learning*, Vol. 57:205–232, 2004.

[10] T. Gärtner, J. W. Lloyd, and P. A. Flach. Kernels for structured data. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *LNAI*, pages 66–83, 2003.

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

[12] Thomas Gartner, Peter A. Flach, Adam Kowalczyk, and Alex J. Smola. Multi-instance kernels. In *Proceedings of the 19th Int. Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, 2002.

[13] 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.

[14] A. Hutchinson. Metrics on terms and clauses. In *Proc. of the 9th European Conference on Machine Learning*, volume 1224 of *LNAI*, pages 138–145, 1997.

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

[16] James D. Keeler, David E. Rumelhart, and Wee-Kheng Leow. Integrated segmentation and recognition of hand-printed numerals. In *Advances in Neural Information Processing Systems 3*. Morgan Kaufmann, 1991.

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

[18] J. W. Lloyd. *Foundations of Logic Programming*. Springer, 1987. Second edition.

[19] H. Lodhi, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. In *Advances in Neural Information Processing Systems 13*, pages 563–569. MIT Press, 2001.

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

[21] S.Ñienhuys-Cheng. Distance between Herbrand interpretations: A measure for approximations to a target concept. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *LNAI*, pages 213–226, 1997.

[22] G. Plotkin. A note on inductive generalization. *Machine Intelligence*, 6, 1970.

[23] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

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

[25] 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.

[26] J. Ramon and T. Gärtner. Expressivity versus efficiency of grapgh kernels. In *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences (MGTS-2003)*, pages 65–74, 2003.

[27] J. Ramon and L. De Raedt. Multi instance neural networks. In *In Attribute-Value and Relational Learning: Crossing the Boundaries.*, 2000.

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

[29] A. Srinivasan, S. Muggleton, R. King, and M. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In *Proc. of 4th Int. Workshop on ILP*, pages 217–232, 1994.

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

[31] Chris Thornton. *Truth from Trash: How Learning Makes Sense*. MIT Press, 2000.

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

[33] 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.