Machine Learning, 57, 233–269, 2004

Naive Bayesian Classification of Structured Data

PETER A. FLACH peter.flach@bristol.ac.uk
Department of Computer Science, University of Bristol, United Kingdom

NICOLAS LACHICHE lachiche@lsiit.u-strasbg.fr
LSIIT, Université Louis Pasteur, Strasbourg, France

Editor: Stan Matwin

Abstract. In this paper we present 1BC and 1BC2, two systems that perform naive Bayesian classification of structured individuals. The approach of 1BC is to project the individuals along first-order features. These features are built from the individual using structural predicates referring to related objects (e.g., atoms within molecules), and properties applying to the individual or one or several of its related objects (e.g., a bond between two atoms). We describe an individual in terms of elementary features consisting of zero or more structural predicates and one property; these features are treated as conditionally independent in the spirit of the naive Bayes assumption. 1BC2 represents an alternative first-order upgrade to the naive Bayesian classifier by considering probability distributions over structured objects (e.g., a molecule as a set of atoms), and estimating those distributions from the probabilities of its elements (which are assumed to be independent). We present a unifying view on both systems in which 1BC works in language space, and 1BC2 works in individual space. We also present a new, efficient recursive algorithm improving upon the original propositionalisation approach of 1BC. Both systems have been implemented in the context of the first-order descriptive learner Tertius, and we investigate the differences between the two systems both in computational terms and on artificially generated data. Finally, we describe a range of experiments on ILP benchmark data sets demonstrating the viability of our approach.

Keywords: bayesian classifier, structured data, inductive logic programming, knowledge representation, first-order features

1. Introduction

Probabilistic approaches to classification typically involve modelling the conditional probability distribution $P(C \mid D)$, where C ranges over classes and D over descriptions, in some language, of objects to be classified. Given a description d of a particular object, we assign the class $\text{argmax}_c P(C = c \mid D = d)$. A Bayesian approach splits this posterior distribution into a prior distribution $P(C)$ and a likelihood $P(D \mid C)$:

$$\underset{c}{\text{argmax}}\ P(C = c \mid D = d) = \underset{c}{\text{argmax}}\ \frac{P(D = d \mid C = c)P(C = c)}{P(D = d)} \tag{1}$$

The denominator $P(D = d)$ is a normalising factor that can be ignored when determining the maximum *a posteriori* class, as it does not depend on the class.

The key term in Eq. (1) is $P(D = d \mid C = c)$, the likelihood of the given description given the class (often abbreviated to $P(d \mid c)$). A Bayesian classifier estimates these likelihoods

from training data, but this typically requires some additional simplifying assumptions. For instance, in an attribute-value representation (also called *propositional* or single-table representation), the individual is described by a vector of values $a_1, \ldots, a_n$ for a fixed set of attributes $A_1, \ldots, A_n$. Determining $P(D = d \mid C = c)$ here requires an estimate of the joint probability $P(A_1 = a_1, \ldots, A_n = a_n \mid C = c)$, abbreviated to $P(a_1, \ldots, a_n \mid c)$. This joint probability distribution is problematic for two reasons: (1) its size is exponential in the number of attributes n, and (2) it requires a complete training set, with several examples for each possible description. These problems vanish if we can assume that all attributes are independent given the class:

$$P(A_1 = a_1, \ldots, A_n = a_n \mid C = c) = \prod_{i=1}^{n} P(A_i = a_i \mid C = c) \tag{2}$$

This assumption is usually called the *naive Bayes assumption*, and a Bayesian classifier using this assumption is called the *naive Bayesian classifier*, often abbreviated to 'naive Bayes'. Effectively, it means that we are ignoring interactions between attributes within individuals of the same class.

Even in cases where the assumption is clearly false, the naive Bayesian classifier can give good results (Domingos and Pazzani, 1997). This can be explained by considering that we are not interested in $P(d \mid c)$ *per se*, but merely in calculating the most likely class. Thus, if $\text{argmax}_c P(a_1, a_2 \mid c) = \text{argmax}_c P(a_1 \mid c) P(a_2 \mid c)$ for all values a_1 and a_2 of attributes A_1 and A_2, the naive Bayes assumption will not result in a loss of predictive accuracy as a result of the interaction between A_1 and A_2, even if the actual probabilities are different. Roughly speaking, this requires that the combined attribute $A_1 \wedge A_2$ correlates with the class in a similar way as the individual attributes A_1 and A_2. For instance, a case where the naive Bayesian classifier clearly fails is where a_1 and a_2 separately correlate positively with class c, but their conjunction correlates negatively with it (e.g., XOR or parity problems). Another way of saying this is that naive Bayes can be a good ranker, even if it is a poor probability estimator.

Inductive logic programming (ILP), also called multi-relational data mining, studies how to upgrade machine learning methods to the richer representation formalism of first-order logic in general and logic programming languages such as Prolog in particular (Muggleton, 1992; Muggleton and De Raedt, 1994; Džeroski and Lavrač, 2001). Originally perceived as a form of logic program synthesis from examples—which is not a pure classification task—ILP appeared to be very different from attribute-value learning. However, in recent years a view of ILP has emerged which stresses the similarities rather than the differences with propositional learning. Central to this view is the notion of an *individual-centred* representation (Flach, 1999; Lavrač and Flach, 2001): the data describes a collection of objects (e.g., molecules, customers, or sales transactions), we distinguish the *individuals* (e.g., molecules) that are classified according to a target predicate (e.g., class) from the other *related objects* (e.g., atoms, bonds), and the induced rules generalise over the individuals, mapping them to a class. Clearly, the notion of a structured individual arises in many domains, including molecular biology and bio-informatics, information retrieval (structured documents, web pages), image processing, and so on.

From the probabilistic viewpoint pursued in this paper, the main question is how to model the influence of the related objects on the likelihood $P(d \mid c)$ that is used to classify

the individual. The chief complication here is that each individual may have an arbitrary number of related objects. As a result, the logical representation space is not a simple Cartesian product of attributes as assumed by the attribute-value probability space employed by naive Bayes. In order to resolve this mismatch we can either downgrade the representation space, or upgrade the probability space. The first approach employs aggregated properties of related objects as attributes of the individual. As it closely follows the language of possible descriptions of an object, we call it the *language space* approach. The second approach involves defining and estimating probability distributions over structured objects. Since this approach follows the structure of actual individuals to be classified we call it the *individual space* approach. In this paper we study both approaches to upgrading the propositional naive Bayesian classifier to structured data. In the first approach, leading to the `1BC` system, we select a restricted set of first-order conditions for d, that are used as attributes in a classical propositional naive Bayesian classifier. The second system, `1BC2`, generalises propositional naive Bayes by introducing ways to estimate probability distributions over structured objects described by what is essentially an (annotated) entity-relationship model. We use a geometric distribution for modelling probabilities of collections in order to see how far such a simple approach takes us. We present the two systems from a unifying perspective, focusing on the representation.

The outline of the paper is as follows. We start by defining our representation of structured objects in Section 2. Clearly, the representation of data and hypotheses is of crucial importance in machine learning, and we will therefore spend some time to discuss the links between our ISP representation, relational representations, and term-based representations. We also discuss the two ways of upgrading the attribute-value classifier to such structured data: either by considering the language space, or by considering the individual space. Sections 3 and 4 are devoted respectively to the language-based approach and to the individual-based approach. In Section 5 we compare the two approaches regarding their algorithmic properties, and investigate the differences in their behaviour on several artificial data sets. Section 6 describes further experimental results on several ILP data sets. In Section 7 we discuss a range of related work. Finally, Section 8 concludes with a summary of the main contributions and an outlook on future work.

2. Representing structured data

In this section, different representations of structured data are compared. First we present our representation, based on ISP declarations (individuals, structural predicates, and properties). Sections 2.2 and 2.3 compare ISP representations to flattened relational and term-based representations, respectively. The term-based representation is relevant because it forms the conceptual basis for `1BC2`'s probability distributions. Finally, in Section 2.4 we take a closer look at the individual and language spaces.

2.1. ISP representation

We mostly follow a flattened Prolog (or Datalog) notation. A *literal* consists of a predicate with a number of arguments between brackets. Arguments of predicates can be either

variables (starting with an upper-case character) or constants (starting with a lower-case character). For instance, `element(A,carbon)` is a literal consisting of the binary predicate `element`, the variable `A` and the constant `carbon`. Rules or clauses are written as backwards implications, with `:-` replacing the implication connective ←. So, the following rule states that any molecule `M` is `mutagenic` if any one of its atoms `A` is a `carbon` atom:

```
class(M,mutagenic):-molecule2atom(M,A),element(A,carbon)
```

Variables are implicitly universally quantified. There is, however, a distinction between variables occurring in both antecedent and consequent of a rule (here `M`) and variables occurring only in the antecedent (here `A`). The latter are sometimes called *existential variables* because the universal quantifier for such variables can be pushed into the antecedent, thus becoming an existential quantifier. Roughly speaking, variables that occur in both antecedent and consequent refer to the individual that is being classified, while existential variables refer to parts of these individuals. The first-order features we consider in this paper can be interpreted as antecedents of rules; we must therefore rely on the data model to inform us which of the variables in the feature denotes the individual.

Our data modelling approach has been inspired by the strongly-typed term-based representation proposed in Flach, Giraud-Carrier, and Lloyd (1998) which we translated to a flattened Prolog setting in Flach and Lachiche (1999) and Flach and Lachiche (2001). It has also been used in other propositionalisation approaches to rule learning (Lavrač and Flach, 2001) and relational subgroup discovery (Lavrač, Železný, and Flach, 2002). The data model we use consists of ISP declarations, which have three parts: a part declaring the individuals, a part declaring structural predicates, and a part declaring properties. Each part starts with the appropriate keyword. We illustrate ISP declarations using the mutagenesis data set (Srinivasan et al., 1994). This is concerned with predicting whether molecules are likely to cause mutations in cells. A molecule is described by its structure (atoms and bonds between atoms) as well as measurements of certain chemical properties.

```
--INDIVIDUAL
molecule 1 mol
--STRUCTURAL
mol2atom 2 1:mol *:atom
fr_atom2bond 2 1:atom *:bond
to_atom2bond 2 1:atom *:bond
--PROPERTIES
class 2 mol #class
lumo 2 mol #lumo
logp 2 mol #logp
element 2 atom #element
atomType 2 atom #atomType
charge 2 atom #charge
bondType 2 bond #bondType
```

As in mode declarations used in e.g., `Progol` (Muggleton, 1995), a *domain* is assigned to each argument of the declared predicates. (We avoid using the word 'type' here, as that is reserved in this paper for term-based representations (Section 2.3).) For instance, `class 2 mol #class` is the declaration of a predicate `class` of arity 2, with the first argument ranging over the domain `mol` and the second argument ranging over domain `class`. The hash # denotes a *parameter*, i.e., an argument of the predicate which is always instantiated in hypotheses with constants found in the data; parameters only occur in properties. Non-parameter domains indicate object identifiers.

The above ISP declaration identifies the domain of the individual as `mol`. (The unary predicate `molecule` is a dummy predicate whose sole purpose is to give the individual declarations the same syntax as the other declarations.) Structural predicates express *n*-to-*m* relationships between objects. For instance, the declaration `mol2atom 2 1:mol *:atom` states that there is a one-to-many relationship between a molecule and its atoms (an atom belongs to exactly one molecule, whereas a molecule can be related to several atoms). To keep the representation compatible with previous representations for mutagenesis in the literature, bonds have been oriented and the first atom of the bond is distinguished from the second one: `fr_atom2bond`, and `to_atom2bond`.

The remaining declarations denote properties of the individual and its related objects. Properties always assign a value to an object. For instance, `element 2 atom #element` declares a predicate specifying the element of an atom, e.g., `element(d11,c)` states that the atom `d11` is a carbon atom. The parameter is always instantiated by a constant, both in the examples and in the hypotheses or features. The predicate stands for an attribute of an object, the parameter stands for its value, and the remaining argument identifies the object. A property is similar to a propositional attribute, except that it does not necessarily refer to the individual. The parameter represents the value of the attribute.

Definition 1 (Structural predicate and property). A *structural predicate* is a binary predicate representing the relation between one domain and another. Given an object of one domain, a *functional* structural predicate, or *structural function*, refers to a unique object of the other domain, while a *non-determinate* structural predicate is non-functional. A *property* is a binary predicate characterising an object, one of whose domains is a parameter. A *parameter* is an argument of a property which is always instantiated.

Non-determinate structural predicates are used to form collections of objects. We use the term 'collection' as a generic term for flexible-size aggregates such as lists, sets and multi-sets, and the term 'non-determinacy' to refer to the existence of such flexible-sized collections . It should be noted that properties are assumed to be discrete, but `1BC` and `1BC2` can be instructed to discretise continuous properties before training (they only model discrete random variables).

Here is part of the description of a molecule given the above ISP declarations.

```
class(d1,mutagenic).
lumo(d1,-1.246).
logp(d1,4.23).
```

```
% first atom                          % second atom
mol2atom(d1,d1_1).                   mol2atom(d1,d1_2).
element(d1_1,c).                     element(d1_2,c).
atomType(d1_1,22).                   atomType(d1_2,22).
charge(d1_1,-0.117).                 charge(d1_2,-0.117).
fr_atom2bond(d1_1,d1_1d1_2).         to_atom2bond(d1_2,d1_1d1_2).
fr_atom2bond(d1_1,d1_1d1_7).         ...
to_atom2bond(d1_1,d1_6d1_1).
% bonds
bondType(d1_1d1_2,7).
bondType(d1_1d1_7,1).
bondType(d1_6d1_1,7).
```

As is usual in a flattened representation, the structure of an individual is encoded in the predicates. An individual such as molecule `d1` is described by all ground facts referring to `d1`, and by ground facts referring to related objects linked to `d1` by structural predicates.

This concludes the description of our ISP data model. In Section 3.1 we show how the data model can be used to define first-order features, as well as the subset of elementary features used in `1BC`. In the remainder of this section, we discuss the relation between ISP declarations and two closely related data models: entity-relationship models, and type signatures.

2.2. Relational representation

Relational representations are well-known from databases, and include the relational database model and the entity-relationship (E-R) model. The E-R model is essentially a formalism for representing meta-data, while the relational model is more closely associated with operations on the data. In practice, an E-R model is often used first; the process of transforming it into a relational model is well-understood (Date, 1995). We will show the close link between our ISP declarations and a restricted E-R model. The E-R-model consists of *entities* (objects) and *relationships* between entities. Figure 1 gives an entity-relationship model of the mutagenesis data set.

The restrictions imposed by ISP declarations are: (1) identifiers (keys) are not composite; (2) only binary relationships are allowed; and (3) relationships carry no property. The first restriction is merely technical, since identifiers can easily be restricted to a single variable

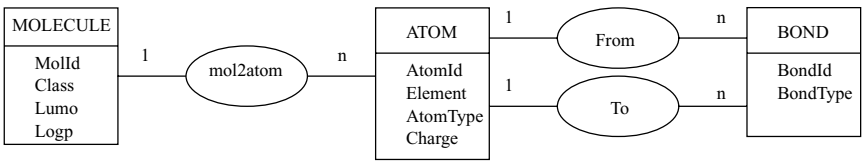


Figure 1. Entity-relationship model of the mutagenesis data set.

by introducing a new identifier if necessary. The second restriction roughly means that any n-ary relationship becomes a new entity. The last restriction implies that properties that would normally be carried by relationships will become properties of entities, either of existing entities for relationships corresponding to functional dependencies, or of a new entity for many-to-many relationships.

The link between the ISP declarations and the restricted E-R model is straightforward. The individual section of the ISP declarations corresponds to the entities. All properties in the ISP declarations correspond to attributes of entities in the E-R model. Finally, the structural predicates correspond to the relationships.

2.3. Term-based representation

Full (i.e., non-flattened) Prolog has the ability to represent structured data by means of functors, which are non-evaluated function symbols whose arguments are terms. Non-recursive functors (also called data constructors) can be used to represent tuples, while recursive functors represent flexible-size ordered datatypes such as lists and trees. In a non-typed language such as Prolog, functors are introduced by using them, while in a typed language they need to be declared first. The use of a typed language in ILP was advocated in Flach, Giraud-Carrier, and Lloyd (1998), because the type declarations establish a strong declarative bias similar to the entity-relationship data model.

As an example, here is a type definition for mutagenesis written in the Escher language, which is a higher-order logic and functional programming language based on the Haskell functional programming language (Lloyd, 1999, 2003):

```
type Molecule = (Atoms,LUMO,LogP);
type Atoms = {Atom};
type LUMO = Float;
type LogP = Float;
type Atom = (Element,AtomType,Charge,FromBonds,ToBonds);
data Element = Br | C | Cl | F | H | I | N | O | S;
type AtomType = Int;
type Charge = Float;
type FromBonds = {Bond};
type ToBonds = {Bond};
type Bond = (BondType);
type BondType = Int;
```

A molecule is a tuple consisting of a set of atoms, and the two attributes `LUMO` and `LogP`. An atom is a tuple consisting of attributes `Element` (an enumerated datatype), `AtomType` (an integer), and `Charge` (a floating point number). The remainder of the type structure deals with bonds between atoms.

The presence of bonds means that molecules are graphs rather than trees. In pure term representations (i.e., without object identifiers), graph structures such as molecules always pose representational problems. In this simplified example we obtain a tree representation

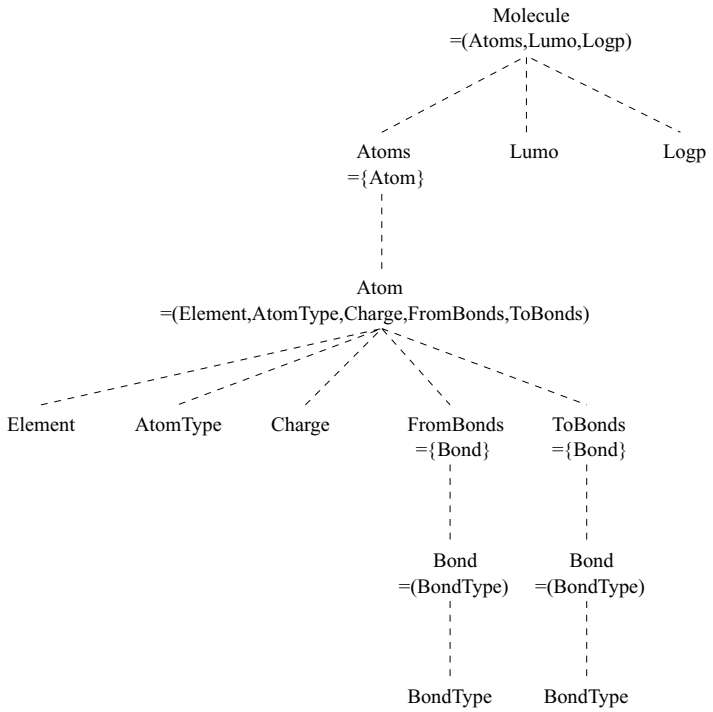


Figure 2. Type structure of the mutagenesis data set.

from a molecule by cutting each bond in the middle. In order to maintain consistency with the benchmark data set, bonds are directed. Consequently, each atom has two associated sets of bonds: the ones which depart from that atom, and the ones which arrive at that atom. Bonds can have several attributes; for simplicity we have represented it as a 1-tuple with a single attribute `BondType`. Figure 2 gives a graphical representation of this type structure.

The relation between the term representation and the ISP representation is well-defined but slightly less immediate than in the entity-relationship case. Generally speaking, translating a term representation to ISP (i.e., flattening (Rouveirol, 1994)) involves *naming* non-atomic subterms,[1] i.e., introducing object identifiers. Structural predicates correspond to going from a composite term to a subterm. This can often be simplified if tuples are involved: e.g., we can go from a molecule to an atom in one step if sets of atoms are not interesting in their own right (i.e., we collapse the one-to-one relation from molecules to sets of atoms and the one-to-many relation from sets of atoms to atoms into a single one-to-many relation from molecules to atoms). Each ISP property corresponds to an atomic component of a tuple.

Comparing the three representations, we see that the bias imposed by ISP and E-R representations is predominantly concerned with distinguishing between one-to-one and one-to-many relationships. The bias imposed by term representations is finer-grained, as it is able to represent different kinds of non-determinacy arising from different datatypes such as list, sets, and multisets. Another advantage of the term representation is that each

term is self-contained. On the other hand, terms cannot easily handle graphs that are not trees—either this leads to loss of information, as in the above example, or we need to resort to naming related objects in order to deal with co-occurrences (e.g., atoms occurring in bonds). While ISP representations are closer to E-R representations, it is useful to keep the term representation in mind as the conceptual basis for the probability decompositions used by `1BC2`. In order to facilitate this, the ISP declarations we use in practice do inherit some of the characteristics of the term representation, particularly the ability to annotate structural predicates with a datatype (list, set, multiset).

2.4. Individual space and language space

The starting point for any upgrade to the propositional naive Bayesian classifier is given by the right-hand side of Eq. (1): we need to obtain the likelihood $P(d \mid c)$ of observing an individual with description d given it is of class c. Following the naive Bayesian approach, we need to decide how to decompose this likelihood into independent components that are more easily and more reliably estimated.

We propose two ways of upgrading the propositional naive Bayesian classifier to structured data. The first approach consists in upgrading propositional attributes to first-order features (see Section 3). This involves generating something equivalent to propositional attributes, the so-called first-order features, that take into account the relevant properties of the individual and its related objects. In other words, the structured representation is projected onto a tuple of feature values. This approach is usually referred to as propositionalisation (Kramer, Lavrač, and Flach, 2001). A straightforward propositionalisation approach would first generate the propositionalised data set before applying the propositional learner. Section 5 shows how that approach can be more efficiently implemented using a recursive algorithm. That is, the propositionalised feature space is not generated explicitly, but handled implicitly in the classification phase. This is similar to the use of kernels in support vector machines and other kernel-based methods (see Gärtner, Lloyd, and Flach, 2004 for a kernel defined on structured objects).

Our second approach to upgrading propositional naive Bayes involves modifying its probabilistic engine. No representation transformation is involved, as we work directly on the structured representation. Suppose the individual to be classified is a collection of objects, then its description d is an enumeration of the objects included in the collection. Our modified naive Bayes assumption states that if we know the probabilities of those objects (which are assumed to be independent), we can determine the probability of the collection as a whole. This will be covered in Section 4.

As already mentioned, the two approaches have a common core that can be captured by the same abstract algorithm, which follows the structure of the data. From this common perspective, the difference between the two approaches is that `1BC` works in the *language* space while `1BC2` works in the *individual* space. In order to build the first-order features, the first approach considers the language only. Features can be built recursively from the domain `O` of the object: it is either a property of `O` or a first-order feature of a domain `D` related to `O` through a structural predicate. The second approach works in the individual space. The estimated probability of a given individual `i` is the product of the estimated probabilities of

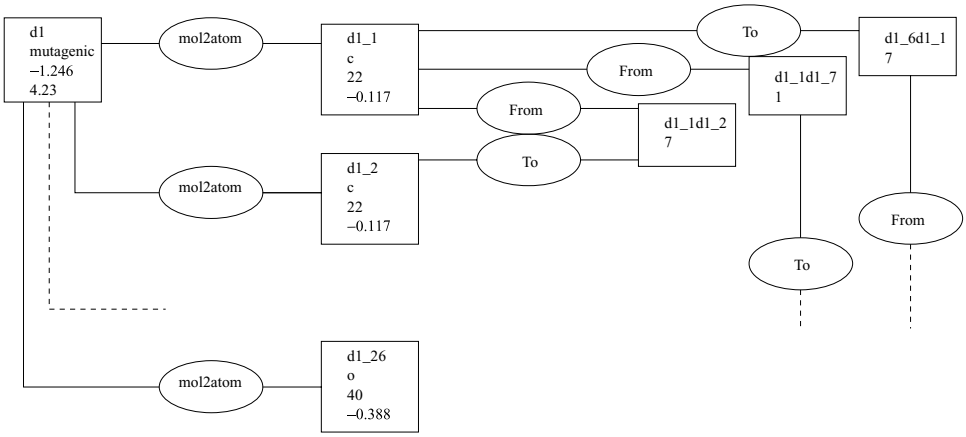


Figure 3. An individual in the relational representation of the mutagenesis data set.

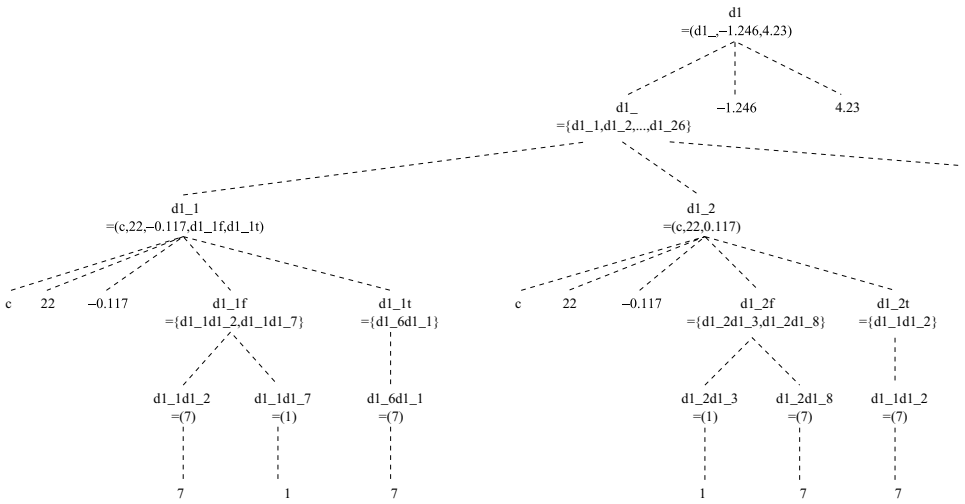


Figure 4. An individual in the term-based representation of the mutagenesis data set.

the properties of `i` and the estimated probability of the collection of the objects `e` related to `i` through a structural predicate, which in turn is obtained from the (recursively) estimated probabilities of each object `e`. The common principle of both approaches is to consider the properties of the current object, and recursively consider the related objects.

The practical difference between both approaches is that the actual structures in the language space and in the individual space differ. Figures 3 and 4 show the entity-relationship and term-based representations of an individual of a molecule in the mutagenesis data set. These individual structures should be contrasted with the language structure as expressed by the data model (cf. figures 1 and 2).

3. Upgrading naive Bayes in language space

In this section, our language space approach for upgrading the propositional naive Bayesian classifier is presented. First, we introduce first-order features as a generalisation of propositional attributes (Section 3.1). Section 3.2 discusses the implementation in the `1BC` system.

3.1. First-order features

First-order features as defined here first appeared in Flach and Lachiche (1999), and have later been used in other propositionalisation approaches including (Lavrač and Flach, 2001; Lavrač, Železný, and Flach, 2002). They are built from the ISP declarations introduced in Section 2, attributing to the individual a property of its related objects, linked to the individual by one or more structural predicates. A first-order feature can be defined according to the following syntax.

Definition 2 (First-order feature). Given a *global* variable denoting the individual, a *first-order feature* (often abbreviated to 'feature') of that individual is a conjunction of literals involving structural predicates and properties where:

1. each structural predicate uses either the global variable or one of the local variables introduced by other structural predicates, and introduces a new local variable;
2. properties do not introduce new variables;
3. all variables are used at least once by a structural predicate or a property.

There are two kinds of first-order features, depending on whether some of the structural predicates involved are non-determinate. A fairly common case is that all structural predicates are non-determinate, in which case the feature is *existential* (also called non-determinate). For instance, `mol2atom(M,A),fr_atom2bond(A,B),bondType(B,7)` is a first-order feature that relates the property of a bond `B` to its molecule `M` through appropriate structural predicates as well as intermediate objects, here an atom `A`. All variables are existentially quantified except the variable corresponding to the individual, which is free (i.e., not quantified). (In a proper first-order predicate language this feature would be written as $\exists A, B : mol2atom(M, A) \wedge fr\_atom2bond(A, B) \wedge bondType(B, 7)$.) The existential quantifier acts as an aggregation operator, taking the disjunction over all possible instantiations of the variable. Additionally, different aggregation operators (e.g., minimum, maximum, average, cardinality) could be used to increase expressiveness (Knobbe, de Haas, and Siebes, 2001; Krogel and Wrobel, 2001). While the existential operator seems appropriate for many data sets including multi-instance problems (Dietterich, Lathrop, and Lozano-Perez, 1997), other aggregation operators may make sense for other data sets, e.g., the PKDD'99 challenge involving analysis of financial data (Knobbe, de Haas, and Siebes, 2001). Only existential aggregation is currently implemented in `1BC`.

If some of the structural predicates involved are functional, the new variables they introduce do not require quantification as they are uniquely determined by other variables. However, if the feature contains at least one non-determinate structural predicate the feature

as a whole is still existential and the domain of the feature is Boolean (i.e., the feature is either true or false). However, features involving only structural functions inherit their domains from the properties involved. A feature is *functional* if all structural predicates involved in the feature are functional from the individual to its related objects. The domain of a functional feature is the Cartesian product of the domains of the parameters of its properties. Functional features are treated separately in the algorithms below. A special case of functional features are those without structural predicates, that refer to properties of the individual (also called propositional attributes). For instance, `lumo(M,L)` is a functional feature of a molecule `M`, since `lumo` is declared as a property.

The reader will have noticed that there is no minimality requirement in Definition 2. For instance, the following conjunction satisfies the definition:

```
mol2atom(M,A1),element(A1,c),mol2atom(M,A2),atomType(A2,22)
```

even though it is logically equivalent to a conjunction of two simpler first-order features, which only share the global variable (the first two literals and the last two literals, respectively). It is easy to define first-order features to be minimal in this sense, as is done in e.g., Lavrač and Flach (2001). However, notice that there will still be a subsumption relation between many of the features, e.g., `mol2atom(M,A),element(A,c)` subsumes `mol2atom(M,A), element(A,c),atomType(A,22)`. The naive Bayes assumption of features being conditionally independent is clearly invalid if there is a subsumption relation between features. For that reason, we use a much stronger constraint on first-order features that guarantees the absence of subsumption relations.

Definition 3 (Elementary features). A first-order feature is *elementary* if it contains a single property. The *context* of an elementary feature is the conjunction of structural predicates occurring in it. Given an individual, the *value* of an elementary feature is the parameter value of the property if the feature is functional, and true or false depending on whether the feature is satisfied for the individual if the feature is non-determinate.

It can be easily seen that, if we exclude the use of any structural predicates, all propositional features (i.e., properties of the individual) are elementary. The restriction to elementary features, and assuming their independence, thus can be said to generalise the propositional naive Bayes assumption. However, it should be noted that, in the first-order case, this restriction leads to loss of not only statistical but also logical knowledge, because not all features (i.e., those involving more than one property referring to the same non-determinate local variable) can be written as conjunctions of elementary features.

3.2. *The `1BC` first-order naive Bayesian classifier*

`1BC` is an upgrade of the propositional naive Bayesian classifier relying on elementary first-order features. `1BC` has been implemented in C in the context of the first-order rule discovery system `Tertius` (Flach and Lachiche, 2001).

`Tertius` uses ISP declarations as explained in Section 2.1; that is, each argument of a predicate belongs to a named type and the set of constants belonging to a type defines

its domain. Moreover, if a domain is continuous, `Tertius` can discretise it into either intervals of one standard deviation each and centered around the mean, or into a specified number of equal-size bins. `Tertius` constructs first-order features in a top-down search, starting with the empty feature and iteratively refining it by adding structural predicates and properties. In order to prevent redundancy, the refinement steps (i.e, adding a literal, unifying two variables, and instantiating a variable) are applied in a fixed order. Once a particular refinement step is applied, none of its predecessors are applicable anymore.[2] Since there might be an infinite number of refinements, the search is restricted to a maximum number of literals and variables, specified on the command line. In addition, a maximum number of occurrences of a particular predicate can be specified as part of the ISP declarations. Elementary first-order features are generated by constraining `Tertius` to generate only hypotheses containing exactly one property and no unnecessary structural predicates. The features can optionally be read from a file.

The `1BC` system presented in Flach and Lachiche (1999) proceeds as follows. In the training phase it generates the first-order features and counts how many individuals of each class satisfy each value of each feature. Specifically, for each feature F_i, each feature value f_{ij} and each class c we obtain counts $n(f_{ij} \wedge c)$ and $n(c)$. The function `estP` calculates probability estimates from these counts using Laplace correction, i.e., $\texttt{estP}(f_{ij} \mid c) = \frac{n(f_{ij} \wedge c)+1}{n(c)+\phi_i}$ where ϕ_i is the number of values of F_i (the number of parameter values for a functional feature, 2 for a non-determinate feature).

The classification phase involves calculating the class likelihoods for each test instance, multiplying them with the class prior, and determining the class that maximises the posterior. Algorithm 1 details the calculation of the class likelihoods.

Algorithm 1 (`1BC` classification algorithm, non-recursive version). *To obtain the likelihoods $P(i \mid c)$ of an individual i given the class c, for all classes*:

```
getLikelihood(i) {
/* initialise likelihoods */
FOR each class c
    CL[c] = 1;
/* iterate through features */
FOR each feature F
    let f be the value of F for i
    FOR each class c
        CL[c] = CL[c] x estP(f|c);
Return CL}
```

We re-iterate the point that the value of a feature, as referred to in Algorithm 1, is Boolean in case of a non-determinate feature. This means that both the presence and the absence of property values (e.g., whether the molecule contains a carbon atom) is taken into account. In contrast, for a functional feature only the property value actually occurring in the individual influences its likelihood. This distinction is also relevant in the case of missing values: while for functional features we do not make any assumption about the missing property value, for

non-determinate features we apply the closed-world assumption in the sense that we only assume a non-determinate feature to be true if we have evidence for it, and false otherwise.

Section 5.1 will show that a recursive implementation, similar to the one used in the individual space, can significantly decrease the runtime compared to the non-recursive implementation.

4. Upgrading naive Bayes in individual space

In the previous section we derived a first-order upgrade of the naive Bayesian classifier in language space. Essentially, our approach was to map the first-order representation to a propositional feature space, on which a probability distribution can be defined in the usual way. The propositional probability space is defined in terms of summary statistics of the original data (in particular whether the example has a related object with a certain property). The advantage of the approach is its simplicity: in particular, we did not have to modify the naive Bayes engine. On the other hand, the loss of logical information can be a disadvantage.

In this section we consider an alternative way of upgrading naive Bayes, by considering the individual space of structured objects directly (rather than their data model). This requires the definition of probability distributions over structured objects, as considered in the next section. Then, the `1BC2` system that implements them is presented (Section 4.2).

4.1. Probability distributions over structured objects

`1BC2` directly models a probability space ranging over structured individuals. This requires the definition of probability distributions over structured datatypes, or in our ISP representation, over structural predicates. The probability of the individual is defined as the joint probability of its properties on the one hand and of its related objects (to which it is linked by a structural predicate) on the other.

By extending the propositional naive Bayes assumption (the properties of the individual are conditionally independent given the class) to related objects—which are likewise assumed to be conditionally independent—we naturally arrive at a recursive approach: the probability of an object is composed from the marginal probabilities of each of its properties and the probabilities of each of its related objects. This recursive composition considerably simplifies the analysis, as we only have to consider the case of an object consisting of a collection of related objects, where the probability distribution over possible related objects is known. The related objects can therefore be considered to be atomic; in the following, $D = \{x_1, \ldots, x_n\}$ denotes a domain of related objects (e.g., atoms), and P_D is a probability distribution over this domain. The question then is: how to define probability distributions over collections of elements of D.

In Flach, Gyftodimos, and Lachiche (to appear) we analysed this question in more detail; we summarise the results here. We considered three possibilities, loosely referred to by their aggregation mechanisms: lists, multisets, and sets. The list distribution corresponds to a simple stochastic logic program (Muggleton, 1996; Cussens, 2001) which extends a type-checking logic program for lists with probability distributions over clauses:

```
0.2: element(a).
0.3: element(b).
0.5: element(c).

0.28: list([]).
0.72: list([H|T]):-element(H),list(T).
```

The probability of any given list L is then obtained by multiplying the probabilities of the clauses involved in a refutation of the query `?-list(L)`.[3] For instance, the probability of the list `[a]` is $0.72 \times 0.2 \times 0.28 = 0.04$, and the probability of the list `[a,b,c]` is $0.72^3 \times 0.2 \times 0.3 \times 0.5 \times 0.28 = 0.003$.

Put differently, we have a geometrically decaying distribution $P(l) = \tau(1-\tau)^l$ over list lengths, where τ is the probability of the empty list, which is combined with P_D as follows.

Definition 4 (Geometric distribution over lists). P_{li} is defined as follows:

$$P_{\text{li}}\left(\left[x_{j_1}, \ldots, x_{j_l}\right]\right) = \tau(1-\tau)^l \prod_{i=1}^{l} P_D\left(x_{j_i}\right)$$

where $0 < \tau \leq 1$ is a parameter determining the probability of the empty list.

It is readily seen that this defines a proper probability distribution. Clearly, all terms in the product are between 0 and 1, so the result will be between 0 and 1. If we fix the list length l, then $\prod_{i=1}^{l} P_D(x_{j_i})$ adds up to 1 for all possible lists of length l (because the distribution is essentially a tuple distribution with independent components) and thus the cumulative probability for lists of length l is $\tau(1-\tau)^l$. Finally, $\sum_{l \geq 0} \tau(1-\tau)^l = 1$. An equivalent definition is obtained by using an extended domain $D' = \{\epsilon, x_1, \ldots, x_n\}$, where ϵ is a stop symbol, and a renormalised distribution $P_{D'}(\epsilon) = \tau$ and $P_{D'}(x_i) = (1-\tau)P_D(x_i)$. It follows that $P_{\text{li}}([x_{j_1}, \ldots, x_{j_l}]) = P_{D'}(\epsilon) \prod_{i=1}^{l} P_{D'}(x_{j_i})$. $P_{D'}$ represents an $(n+1)$-sided die, which generates list elements until the stop symbol appears.

Notice that the ordering of the list elements is disregarded in the sense that all permutations of a particular list obtain the same probability. If we want to include the ordering of the list elements in the distribution, we can assume a distribution P_{A^2} over pairs of elements of the domain, so that $[a, b, c]$ would have probability $P_{A^2}(ab)P_{A^2}(bc)$ among the lists of length 3. Such a distribution would take some aspects of the ordering into account, but note that, e.g., $[a, a, b, a]$ and $[a, b, a, a]$ would still obtain the same probability, because they consist of the same pairs (aa, ab, and ba), and they start and end with a. Obviously we can continue this process with triples, quadruples etc., but note that this is both increasingly computationally expensive and unreliable if the probabilities must be estimated from data.

Based on this geometric distribution, we defined the probability of a multiset in Flach, Gyftodimos, and Lachiche (to appear) to be the total probability mass of all its permutations. For any multiset x, let k_i stand for the number of occurrences of the i-th element of the domain, and let $l = \sum_i k_i$ denote the cardinality of the multiset. The probability of x is then obtained as $\tau l! \prod_i \frac{P_{D'}(x_i)^{k_i}}{k_i!}$. However, since the k_i (and therefore l) are independent of the class, treating lists as multisets would not affect the classification performance of `1BC2`.

In a similar vein, a set can be seen as the equivalence class of all lists (or multisets) containing each element of the set at least once, and no other elements. For instance, the set $\{a, b\}$ can be interpreted to stand for all lists of length at least 2 which contain (i) at least a and b, and (ii) no other element of the domain besides a and b. The cumulative probability of lists satisfying condition (ii) is easily calculated, but the first condition is computationally expensive as it requires iterating over all subsets of the set. Let S be a non-empty subset of l elements from the domain, then we defined the probability of set S as

$$P_{\text{ss}}(S) = \sum_{S' \subseteq S} \tau(-1)^{l-l'} \frac{P_{D'}(S')^l}{1 - P_{D'}(S')}$$

where

$$l' = |S'| \quad \text{and} \quad P_{D'}(S') = \sum_{x \in S'} P_{D'}(x).$$

This subset distribution takes only the elements occurring in a set into account, and ignores the remaining elements of the domain. For instance, the set $\{a, b, c\}$ will have the same probability regardless whether there is one more element d in the domain with probability p, or 10 more elements with cumulative probability p. In contrast, the usual approach of propositionalising sets by representing them as bitvectors over which a tuple distribution can be defined has the disadvantage that the probability of a set depends on the elements in its extension as well as its complement. Another advantage of the subset distribution is that it is applicable to (finite subsets of) infinite domains.

An obvious disadvantage of the subset distribution is that it is exponential in the cardinality of the set and therefore expensive to compute for large sets. As it turns out, the list distribution can be used to approximate the subset distribution for any set S with $P_{D'}(S) \ll 1$, which is likely in the case of large domains (e.g., sets ranging over structured objects). In this case we have $P_{\text{ss}}(S) \approx l! P_{\text{li}}(L_S)$, where L_S is a list constructed from S by choosing an arbitrary order for its elements (the proof can be found in Flach, Gyftodimos, and Lachiche (to appear)). In our experiments with the `1BC2` system, we found indeed that—whenever it was computationally feasible to use the subset distribution—it yielded virtually identical classification results to the list distribution. In the experiments reported in Section 6 we exclusively used the list distribution. Our goal is to see how much improvement we can get out of such a simple probabilistic model of collections. As such, our work on `1BC2` should be seen as a starting point, which we aim to refine in future work by considering more sophisticated probability distributions over collections.

The general format of these different distributions is the same: given a domain distribution P_D and a parameter τ, a probability distribution over collections of objects from D is defined by $P_{Col(D)} = \mathit{ComposeP}(P_D, \tau)$, where *ComposeP* is a higher-order function taking a probability distribution in its first argument and returning a probability distribution. For instance, in the case of the list distribution we have

$$\mathit{ComposeP}(P_D, \tau) = \lambda X : \tau(1 - \tau)^{|X|} \prod_{x \in X} P_D(x)$$

We assume that *ComposeP* is well-behaved in the sense that if P_D is a probability distribution and $0 \leq \tau \leq 1$, *ComposeP* is also a probability distribution, which is the case for the list, multiset and subset distributions described above. We can therefore nest these distributions recursively, passing on the result of *ComposeP* on one level to *ComposeP* again, in order to define a distribution over collections on the next level of aggregation. This explains the recursive nature of Algorithm 3 in the next section.

To illustrate, given a type structure as the one depicted in figure 2, we can compute a probability distribution over molecules in a bottom-up fashion. Once we know the distribution $P_{\texttt{BondType}}$ over the domain of bond types, we can calculate $P_{\texttt{FromBonds}}$ and $P_{\texttt{ToBonds}}$ using the subset distribution or its approximation by the list distribution. We can then combine this with the distributions $P_{\texttt{Element}}$, $P_{\texttt{AtomType}}$ and $P_{\texttt{Charge}}$ using the tuple distribution, yielding $P_{\texttt{Atom}}$. Using the subset distribution once again, and combining this with $P_{\texttt{LUMO}}$ and $P_{\texttt{LogP}}$ using the tuple distribution, we finally obtain $P_{\texttt{Molecule}}$.

Finally, we notice that a class-conditional version of $P_{\text{Col}(D)}$, as required by the naive Bayesian classifier, is obtained by plugging a class-conditional domain distribution P_D, as well as a class-conditional value of τ, into *ComposeP*. How these are estimated from the data is described in the next section.

4.2. *The 1BC2 first-order naive Bayesian classifier*

In this section we present our implementation of the ideas above in the 1BC2 probabilistic classifier. As explained above, and unlike its predecessor, 1BC2 is fundamentally recursive: distributions over collections of elements take the probability of their elements into account. If an element has a composite type, then its probability is estimated from the probabilities of its components, and so on. The recursion stops on properties. For instance, the probability $P_{\texttt{Element}}(\texttt{element(d1\_1,c)} \mid \texttt{mutagenic})$ of an atom d1_1 to be a carbon atom in a mutagenic molecule is estimated from the proportions of atoms in mutagenic molecules that are carbon atoms. The difference with the 1BC case is that we are not counting the number of mutagenic molecules that have carbon atoms, but rather the number of carbon atoms occurring in mutagenic molecules. That is, we are counting property values rather than feature values.

The probability $P_{\texttt{BondType}}(\texttt{bond(d1\_1d1\_2,7)} \mid \texttt{mutagenic})$ of the bond between atoms d1_1 and d1_2 to be of type 7 is estimated in almost the same way. However, we need to take into account that, since bonds are ordered in our representation, there are two paths between a molecule and a bond: either through the first atom of the bond, or through the second atom of the bond. Therefore, the *context* in which the property occurs, i.e., the path from the individual to the property, is taken into account, and if the same object occurs in different contexts these occurrences are treated as distinct objects. Similarly, structural predicates and properties are indexed by the context in which they occur (essentially, the preceding structural predicates in the first-order feature).

In the mutagenesis data set the notion of context is slightly artificial, since it only arises because of an arbitrary orientation of the bonds. However, in other data sets—such as the finite element mesh design data set (Dolšak and Muggleton, 1992)—it is crucial. This data set concerns predicting the number of finite elements needed to model an edge given its

properties, and the properties of the topologically related edges. Here it makes sense to take into account that a property is related differently to the class depending on whether it is a property of the individual, i.e., the mesh being classified, of the edge on the left-hand side of the individual, and so on.

We will now detail the classification and training phases. The training phase has to estimate the probabilities of each property value, for each class and each context. This involves counting how many objects satisfy each value of the property, taking into account the target predicate and the context in which the property is connected to the individual in the example. That is, we maintain a four-dimensional matrix `propValueCounts` which is indexed by the property, its value, the context, and the class, and fill this matrix by iterating through all training individuals. As before, the classification algorithm uses the Laplace estimate to convert these counts into class-conditional probability estimates. This is done by the function `estP2`, which differs from the function `estP` used in `1BC` because it estimates the probability of property values rather than feature values.

Algorithm 2 (`1BC2` training algorithm). *To take into account the contribution of an object o to the counts for a class c and a context $ctxt$*:

```
addContribution(o,ctxt,c) {
/* handle properties of the object */
FOR all properties prop of o
    let v be the corresponding parameter value;
    propValueCounts[prop,v,ctxt,c] += 1;
/* handle related objects */
FOR all structural predicates struc involving o
    ctxt' = add struc to ctxt;
    IF struc is functional given o THEN
        let o' be the related object;
        addContribution(o',ctxt',c);
    ELSE /* non-determinate structural predicate */
        let O' be the set of related objects;
        parentCounts[struc,ctxt,c] += 1;
        childrenCounts[struc,ctxt,c] += |O'|;
        FOR each object o' of O'
            addContribution(o',ctxt',c);
}
```

At the top level, we initialise all counts to 0, and iterate `addContribution(i,Null,c)` over all training individuals `i`, given their class `c`, where `Null` stands for the empty context.

`parentCounts` and `childrenCounts` collect counts needed for the class-conditional τ_c estimates. After the whole training set has been processed, we divide `parentCounts` by `childrenCounts` to obtain, for each structural predicate, context, and class, the average number $\overline{l_c}$ of objects in the associated collections. The probability τ_c of

empty collections is then estimated as $\frac{1}{1+\bar{l}_c}$ (this is the maximum likelihood estimate of τ_c if cardinalities are distributed according to $P(l_c) = \tau_c(1-\tau_c)^{l_c}$ (Flach, Gyftodimos, and Lachiche, to appear)). This is implemented by the function `estTau`.

For classification, the main function is `getLikelihood(object o, context ctxt)`. It returns a table with an estimate, for each class, of the class-conditional likelihoods of object `o` (either the individual or one of its related objects). Notice that the context is passed on as an argument for all relevant calculations. If a new context or a new parameter value occurs in the test set but not in the training set, then it is ignored in the estimation of the likelihood.

Algorithm 3 (1BC2 classification algorithm). *To obtain the likelihoods $P(o|c)$ of an object o given the class c, for all classes, in a context $ctxt$*:

```
getLikelihood(o,ctxt) {
/* initialise likelihoods */
FOR each class c
    CL[c] = 1;
/* handle properties of the object */
FOR all properties prop of o
    let v be the corresponding parameter value;
    FOR each class c
        CL[c] = CL[c] x estP2(prop,v,ctxt|c);
/* handle related objects */
FOR all structural predicates struc involving i
    ctxt' = add struc to ctxt;
    IF struc is functional given o THEN
        let o' be the related object;
        CL' = getLikelihood(o',ctxt');
        FOR each class c
            CL[c] = CL[c] x CL'[c];
    ELSE /* non-determinate structural predicate */
        let O' be the set of related objects;
        FOR each object o' of O'
            CL''[o'] = getLikelihood(o',ctxt');
        FOR each class c
            tau = estTau(struc,ctxt|c);
            CL[c] = CL[c] x composeP(CL'',tau,c);
Return CL}
```

`composeP(CL'',tau,c)` calculates the class-conditional probability of a collection of objects from the class-conditional probabilities `CL''[o'][c]` of each element `o'` using the list distribution defined in Definition 4 (or possibly the multiset or subset distributions also discussed in Section 4.1). At the top level, we iterate `getLikelihood(i,Null)` over all test individuals `i`.

We illustrate the classification algorithm by means of a worked example.

Example 1 (*Classification of a molecule*). In order to classify molecule `d1` (see figure 3), `1BC2` estimates its likelihood given each class using the properties of that object, and the probabilities of objects it is linked to. For molecules, we retrieve the probabilities of a lumo value of -1.246 and a logp value of 4.23 as estimated during the training phase. Next, we consider the structural predicate `mol2atom`. A molecule can have several atoms; in order to apply P_{li}, `1BC2` requires the probability of each atom of molecule `d1` conditional on the class. This is obtained by applying the algorithm recursively.

The probability of atom `d1_1` is estimated using its properties: `element(d1_1,c)`, `atomType(d1_1,22)`, and `charge(d1_1,-0.117)`, and the probabilities of the objects it is linked to: `fr_atom2bond(d1_1,d1_1d1_2)`, `to_atom2bond(d1_1, d1_6d1_1)`, `fr_atom2bond(d1_1,d1_1d1_7)`, etc. First, for each class we look up the estimates of the probability of a carbon atom, the probability of an atom number of 22, and the probability of a -0.117 charge for that class. Then we need to estimate the probabilities of each bond by applying the same algorithm again. The probability of bond `d1_1d1_2` is estimated using its property: `bondType(d1_1d1_2,7)` in the context of the first atom of the bond. The same is done with appropriate contexts for all bonds in which atom `d1` occurs. Then, given those probabilities, the P_{li} formula is used to get the probability of both sets of bonds. We now have enough information to calculate the probability of atom `d1_1` for each class, by multiplying the bond-set probabilities with those for `element`, `atomType` and `charge`. The same is done for all atoms of molecule `d1`. Finally, the P_{li} formula is used again to calculate the probability of the set of atoms from the probabilities of each atom, which is combined with the probabilities of lumo and logp to yield the probability of the molecule.

This example assumed that lumo, logp and charge were not discretised. If they are discretised, their actual value is replaced with an identifier of the appropriate bin.

The flattened language used by `1BC2` allows more representational flexibility than a pure term language because subterms are named. This means that graph structures such as molecules can be modelled, and we need to take care to avoid cyclic computations. ISP declarations can limit the number of times a structural predicate is used (set to 1 in the above example). In addition, a global limit on the number of structural predicates can be set. In practice, this means that the individual is represented by the set of paths to its related objects satisfying these length constraints.

5. Comparison between `1BC` and `1BC2`

In this section we compare the two systems in order to understand their differences in operation and performance. In Section 5.1 we present an alternative, recursive implementation of `1BC` that has runtime performance similar to that of `1BC2`. Section 5.2 explains the main differences between the computations performed by the two systems on a small example. Section 5.3 investigates the behaviour of and differences between `1BC` and `1BC2` on three artificially generated data sets.

Table 1. Runtimes of the non-recursive implementation of 1BC compared to the recursive implementation of 1BC2 performing a 10-fold cross-validation on several data sets.

Data set	1BC (seconds)	1BC2 (seconds)
Alzheimers (any target)	50	29
Mutagenesis (propositional)	3	1
Mutagenesis (relational)	274	3
Diterpenes (propositional)	21	14
Diterpenes (relational)	21,600	1,100

5.1. Algorithmic comparison

Table 1 presents the runtimes of 1BC and 1BC2 on several data sets described in the next section. All experiments were run on a normal workstation (PC 800MHz, 384MB RAM). From this table, it is obvious that the non-recursive implementation of 1BC (Algorithm 1) originally proposed in Flach and Lachiche (1999) runs considerably slower than the recursive implementation of 1BC2, in particular on the relational data sets. As it turns out, the calculations performed by 1BC and 1BC2 are sufficiently similar to allow for a more efficient recursive implementation of 1BC. This will be detailed in the remainder of this section.

The recursive 1BC training algorithm initialises all counts to 0, and iterates addContribution(i,Null,c,i) as defined in Algorithm 4 over all training individuals i, given their class c, where Null is the initially empty context. The main issue here is that we use existential aggregation and are counting over individuals, so each property is counted only once for an individual and ignored afterwards (hence the current individual is passed on as an additional argument).

Algorithm 4 (1BC training algorithm, recursive version). *To add the contribution of an individual i of class c to the counts of all properties of an object o in a context $ctxt$:*

```
addContribution(o,ctxt,c,i) {
/* handle properties */
FOR all properties prop of o
  not already encountered for i in ctxt
  let v be the corresponding parameter value;
  propValueCounts[prop,v,ctxt,c] += 1;
/* handle related objects */
FOR all structural predicates struc involving o
    ctxt' = add struc to ctxt;
    let O' be the set of related objects;
    FOR each object o' of O'
        addContribution(o',ctxt',c,i);
}
```

As in 1BC, we use estP to convert counts to probability estimates. Below, ctxt & prop(o,v) stands for the elementary feature built from the structural predicates in ctxt and the property prop; estP deals with the counts in the appropriate way depending on whether ctxt is non-determinate or functional.

The classification phase proceeds in two stages. To calculate the likelihood $P(i \mid c)$ of an individual i we first collect the contribution of all properties and related objects occurring in i, making sure that each feature ctxt & prop(o,v) is processed only once. Then, we take all remaining non-determinate features that are not satisfied into account; that is, we apply the closed-world assumption (but only to non-determinate features).

Algorithm 5 (1BC classification algorithm, recursive version). *To obtain the likelihoods $P(i \mid c)$ of an individual i given the class c, for all classes*:

```
getLikelihood(i) {
CL = getContribution(i,Null,i);
FOR all non-determinate contexts ctxt
    FOR all properties prop and parameter values v such that
      prop(o,v) has not been encountered yet for i in ctxt
        FOR each class c
            CL[c] = CL[c] x (1-estP(ctxt & prop(o,v)|c));
Return CL}
```

To obtain the contribution of object o in context $ctxt$ to the class-conditional likelihood of the individual i:

```
getContribution(o,ctxt,i) {
/* initialise likelihoods */
FOR each class c
    CL[c] = 1;
/* handle properties */
FOR all properties prop of o
    let v be the corresponding parameter value;
    IF prop(o,v) has not been encountered yet for i in ctxt
    THEN FOR each class c
             CL[c] = CL[c] x estP(ctxt & prop(o,v)|c);
/* handle related objects */
FOR all structural predicates struc involving o
    ctxt' = add struc to ctxt;
    let O' be the set of related objects;
    FOR each object o' of O'
        CL' = getContribution(o',ctxt',i);
        FOR each class c
            CL[c] = CL[c] x CL'[c];
Return CL}
```

This recursive implementation of 1BC produces exactly the same probabilities as the non-recursive implementation, and its runtime is similar to the runtime of the recursive implementation of 1BC2, i.e., the second column of Table 1.

5.2. Comparison on a worked example

Assuming that 1BC2 uses the geometric list distribution to estimate probabilities of non-determinate structural predicates, there are four main differences between 1BC and 1BC2. A small training set and a test individual will be used to illustrate those differences. For notational brevity we use a term-based representation (cf. Section 2.3). Each individual is a pseudo-molecule consisting of a collection of atoms where an atom is a pair of an element and a type (arbitrarily chosen). The training set contains three positive examples:

```
{(o,22),(c,14),(o,21),(o,22),(n,38),(h,9)}
{(h,9),(o,22),(o,22),(n,38),(h,9)}
{(o,21),(o,22),(c,14),(h,9)}
```

and three negative examples:

```
{(c,14),(c,17)}
{(h,9),(o,22),(c,14)}
{(o,22)}
```

The test individual is:

```
{(o,22),(c,17),(h,9),(h,9),(h,9)}
```

Table 2 lists the conditional likelihood estimates obtained by 1BC and 1BC2 in the training phase.

The first difference is that the counts are based on the individuals in 1BC and on all the related objects appearing in a given context in 1BC2. For instance, consider the feature `mol2atom(M,A),element(A,o)`, stating that the molecule contains an oxygen atom. 1BC counts that 3 out of 3 positive examples contain an oxygen atom, hence the Laplace estimate is $\frac{3+1}{3+2}$ (the feature is Boolean). 1BC2 counts that 7 atoms out of the 15 (belonging to a molecule, as required by the context) are oxygen atoms, and the element property can take on 4 different values, hence the Laplace estimate is $\frac{7+1}{15+4}$.

The second difference is that the number of atoms of each molecule is taken into account in 1BC2. While 1BC makes use of the likelihood estimates of Table 2 directly, 1BC2 makes use of the P_{li} and $P_{D'}$ distributions. So the probability associated with each atom is multiplied by $(1 - \tau)$ to obtain $P_{D'}$. τ is estimated as $\frac{1}{1+\bar{l}}$, where the average cardinality $\bar{l}$ is 5 for the positives and 2 for the negatives.

The third difference is that a property is only taken into account once for each individual in 1BC, while each occurrence is taken into account in 1BC2. So, when classifying the test individual, 1BC uses the probabilities associated with the hydrogen element only once, while 1BC2 uses them thrice.

Table 2. Conditional probabilities learned by 1BC and 1BC2 on a small example.

	1BC		1BC2	
	Positive	Negative	Positive	Negative
mol2atom(M,A),element(A,c)	3/5	3/5	3/19	4/10
mol2atom(M,A),element(A,h)	4/5	2/5	5/19	2/10
mol2atom(M,A),element(A,n)	3/5	1/5	3/19	1/10
mol2atom(M,A),element(A,o)	4/5	3/5	8/19	3/10
mol2atom(M,A),type(A,9)	4/5	2/5	5/21	2/12
mol2atom(M,A),type(A,14)	3/5	3/5	3/21	3/12
mol2atom(M,A),type(A,17)	1/5	2/5	1/21	2/12
mol2atom(M,A),type(A,21)	3/5	1/5	3/21	1/12
mol2atom(M,A),type(A,22)	4/5	3/5	6/21	3/12
mol2atom(M,A),type(A,38)	3/5	1/5	3/21	1/12
τ	N/A	N/A	1/6	1/3

The fourth difference is that 1BC2 does not use the probabilities associated with properties that do not occur in the test individual, while 1BC takes into account the non-determinate features that are not satisfied. So when classifying the test individual, 1BC uses the probabilities associated with the nitrogen element, while 1BC2 does not.

Putting all this together, 1BC calculates the likelihood of the test individual given the positive class as $3/5 \times 4/5 \times (1-3/5) \times 4/5 \times 4/5 \times (1-3/5) \times 1/5 \times (1-3/5) \times 4/5 \times (1-3/5) = 1.3\ 10^{-3}$. Similarly, the likelihood given the negative class is $3/5 \times 2/5 \times (1-1/5) \times 3/5 \times 2/5 \times (1-3/5) \times 2/5 \times (1-1/5) \times 3/5 \times (1-1/5) = 2.8\ 10^{-3}$.

1BC2 classifies the test individual following its structure. It is a set of atoms, so it first estimates the likelihood of each atom, i.e., for the positive class: $P_D((o,22)|+) = 8/19 \times 6/21 = 48/399$; $P_D((c,17)|+) = 3/19 \times 1/21 = 3/399$; $P_D((h,9)|+) = 5/19 \times 5/21 = 25/399$. Taking $\tau = 1/6$ into account, we obtain the likelihood given the positive class as $1/6 \times 5/6 \times 48/399 \times 5/6 \times 3/399 \times (5/6 \times 25/399)^3 = 1.5\ 10^{-8}$. Similarly, the likelihood given the negative class is calculated as $1/3 \times 2/3 \times 3/10 \times 3/12 \times 2/3 \times 4/10 \times 2/12 \times (2/3 \times 2/10 \times 2/12)^3 = 8.1\ 10^{-9}$.

Since the prior distribution is uniform, 1BC predicts the test individual to be negative while 1BC2 predicts it to be positive. This can be explained by the fact that both the number of atoms in the test individual as well as the three occurrences of (h,9) suggest the positive class, but both these factors are ignored by 1BC.

5.3. Experimental comparison on artificial data sets

In this section we describe some of our experiments with artificially generated data. Here and in the next section, performance is assessed by accuracy estimates obtained by cross-validation. In addition, we analysed the performance of the underlying probabilistic model by means of ROC curves. ROC curves provide more information than accuracy estimates

because poor accuracy can result from a poorly chosen decision threshold from an otherwise good ROC curve (see Lachiche and Flach, 2003 for a method to determine good thresholds for two-class and multi-class naive Bayesian classifiers). For probabilistic classifiers, ROC curves evaluate how well the predicted probabilities do in separating positives from negatives, without committing to a particular decision threshold—that is, they concern ranking performance rather than classification performance. The area under the ROC curve (AUC) indicates the aggregated quality of all classifiers that can be constructed by setting different thresholds, and can be interpreted as the probability that a randomly drawn positive example will be ranked higher by the model than a randomly drawn negative example (Hand and Till, 2001). In our experiments, ROC curves were constructed by cross-validation, by merging the rankings obtained on each test fold. This is a simple way of averaging ROC curves, but doesn't give error bars (Fawcett, 2003).

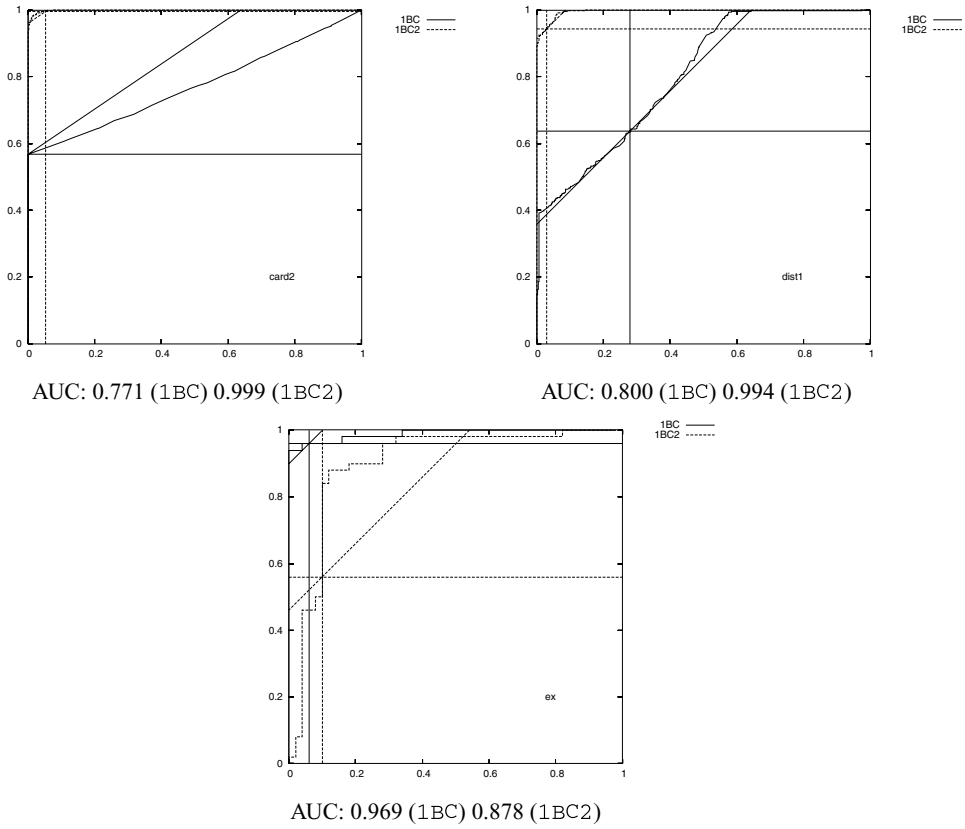
We generated artificial data sets where examples are collections of (pairs of) constants. Collections were generated by sampling from the geometric list distribution with parameter τ. The components of pairs were generated independently by sampling from a fixed distribution over 5 values. The first data set, `card2`, was generated to test the hypothesis that `1BC2` is better able to distinguish between collections of different cardinality. To this end we generated 10,000 examples with average cardinality approximately 32 ($\tau = 0.03$); the collections contained pairs of uniformly distributed constants. Examples with at least 30 elements were designated positive, the rest negative; this resulted in roughly 40% positives and 60% negatives. The accuracy and AUC results in Table 3 and the ROC curves in figure 5 clearly show that `1BC2` achieves near-perfect performance on this task. `1BC` performs well on nearly 60% of the positives, but achieves random performance on the remainder of the data.

The second data set, `dist1`, was generated to test the hypothesis that `1BC2` can model distributions over domains, e.g., elements of atoms, whereas `1BC` only models existential aggregation. We generated collections of average length 19 consisting of pairs, where for each component one value had probability 0.6 and the other 4 values had probability 0.1; we chose different majority values for the positive and negative class (500 examples each). The results show again superior performance of `1BC2` over `1BC`; the `1BC` ROC curve shows that it performs well on roughly 40% of the positives and 40% of the negatives, with random performance on the remainder.

The third data set, `ex`, was designed to demonstrate that `1BC2` needs more data to learn the distributions, and hence its performance may degrade on smaller data sets. We generated small collections of uniformly distributed constants with 50 possible values. A collection was positive if it contained a particular constant and negative otherwise, i.e., an existential concept. The data set contains 50 positive and 50 negative examples; because of the way

Table 3. Accuracy results on artificial data sets, obtained by 10-fold cross-validation (3-fold in the case of `ex`).

Data set	`1BC` (%)	`1BC2` (%)
`card2`	74.2	97.6
`dist1`	67.9	95.7
`ex`	95.0	73.0



AUC: 0.771 (1BC) 0.999 (1BC2)

AUC: 0.800 (1BC) 0.994 (1BC2)

AUC: 0.969 (1BC) 0.878 (1BC2)

Figure 5. ROC curves on artificial data sets. The crosshairs denote the point chosen by the default probability threshold of 0.5, and the diagonal lines indicate iso-accuracy lines at those points (higher lines are better).

the examples were generated there was a small difference in average cardinality (7 for the positives, 4 for the negatives) which 1BC2 should be able to take advantage of. Nevertheless, 1BC2 performs clearly worse than 1BC on this simple task (1BC2's accuracy suffers from the fact that the default probability threshold is clearly sub-optimal in this case).

We conclude from these and other experiments that 1BC2 is generally preferable over 1BC except for small data sets. Also, with 1BC2 the structural probabilities tend to dominate the probabilities of propositional attributes, as will be demonstrated in the next section on the mutagenesis data set.

6. Experimental evaluation

We evaluated the classification performance of 1BC and 1BC2 on three ILP data sets: identifying mutagenic compounds, drugs against Alzheimer's disease, and diterpene structure elucidation.

Table 4. Accuracy on the mutagenesis regression-friendly data set. The majority class is 66.5% in all settings. 1BC and 1BC2 accuracies are reported as $n\%/m\%$, where $n\%$ indicates the accuracy achieved with the default threshold, and $m\%$ indicates the accuracy achieved with an optimised threshold as described in Lachiche and Flach (2003).

Settings	1BC (%)	1BC2 (%)	Progol (%)	Regression (%)
lumo and logp only	85.6/85.6	85.6/85.6		85
lumo, logp, inda and ind1 only	89.9/91.5	89.9/91.5		89
Atoms and bonds only	77.1/79.8	79.3/82.4		
Plus lumo and logp	79.3/82.4	81.4/80.3	88	
Plus inda and ind1	85.1/85.1	83.0/84.6	88	

6.1. Mutagenesis

This problem concerns distinguishing between mutagenic and non-mutagenic compounds (Srinivasan et al., 1994; Muggleton et al., 1998). We considered both "regression-friendly" and "regression-unfriendly" data sets. In these experiments, we used the atom and bond structure of the molecule as one setting, adding the lumo and logp properties to get a second setting, and finally adding Boolean indicators I_a and I_1 as well. We also did experiments involving only lumo and logp, and all four propositional attributes.

Regression-friendly data set. Continuous properties lumo, logp, and charge were discretised into respectively 10, 10 and 100 intervals of equal size (i.e., each interval contains the same number of individuals). Those numbers were chosen in order to get as many intervals as possible but still containing a representative number of individuals (around 30). A 10-fold cross-validation was performed.

Accuracy results are listed in Table 4. The Progol and regression results are quoted from Muggleton et al. (1998). The first thing to note is that on the propositional data, where 1BC and 1BC2 achieve identical performance by construction, naive Bayes outperforms regression. The accuracy drops considerably when only atoms and bonds are considered. The addition of the propositional attributes improves those accuracies, but their weight is generally not enough to compensate for the negative influence of atoms and bonds. We can notice that 1BC benefits more than 1BC2 of propositional features—because the probabilities of first-order features in 1BC are of the same order as the propositional features—whereas in 1BC2 probabilities of collections are smaller, and the differences between likelihoods of collections tend to dominate.

ROC curves and AUC results are given in figure 6. The default thresholds are also indicated.[4] The first two settings are propositional, and it can clearly be seen that 1BC and 1BC2 produce identical probabilistic models. In the relational settings, the ROC curves produced are worse than the propositional one with indicator attributes.

Regression-unfriendly data set. This data set contains 42 molecules. Accuracy estimates and ROC curves were obtained by leave-one-out, as is customary on this data set. We did

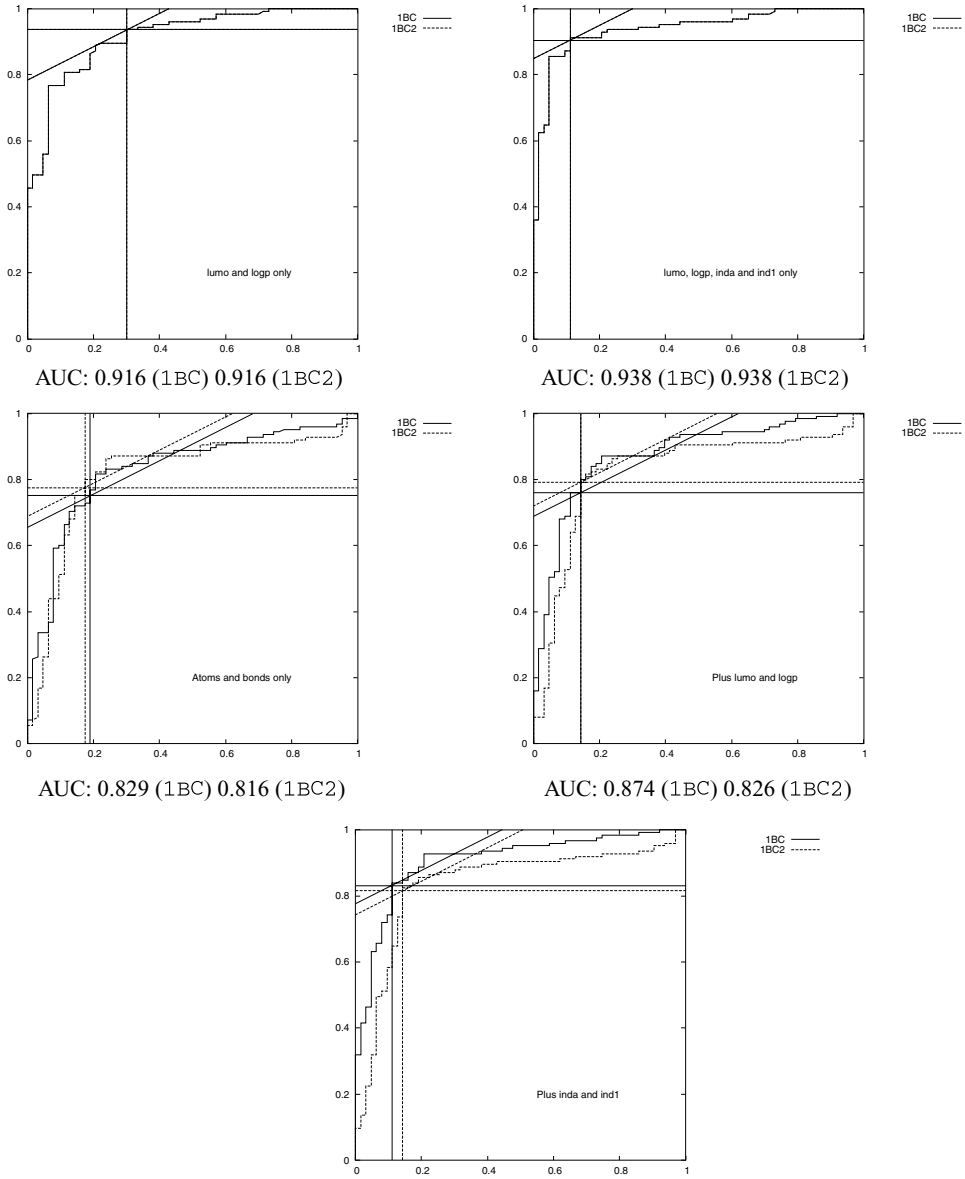


Figure 6. ROC curves on the mutagenesis regression-friendly data set. Performance is best on the propositional representation with all four attributes. On the relational data sets 1BC is slightly better than 1BC2, although the default thresholds (which are optimal or near-optimal) work in 1BC2's advantage on two of them.

Table 5. Accuracy on the mutagenesis regression-unfriendly data set. The majority class is 69% in all settings.

Settings	1BC (%)	1BC2 (%)	Progol (%)	Regression (%)
lumo and logp only	71.4/71.4	71.4/71.4		67
lumo, logp, inda and ind1 only	66.7/66.7	66.7/66.7		67
Atoms and bonds only	71.4/78.6	73.8/76.2		
Plus lumo and logp	71.4/81.0	73.8/83.3	83	
Plus inda and ind1	71.4/78.6	76.2/83.3	83	

not discretise continuous properties because the data set is too small to get a reasonable number of examples in each interval. Accuracy results are given in Table 5, and ROC curves can be found in figure 7.

From the ROC curves it can be seen that performance in the propositional settings is poor. On this data set the addition of the indicator attributes to lumo and logp decreases performance to worse than majority class, whereas with only lumo and logp performance is slightly better than majority class (notice that regression achieves worse than majority class performance in both cases). In the relational settings performance increases considerably, although the choice of the default threshold is poor. In the last two settings 1BC2 with optimised decision threshold achieves the same performance as Progol. Again we notice that the addition of propositional attributes to the atoms and bonds representation has little effect on 1BC2 in terms of the ROC curve, although it does influence the location of the default and optimal thresholds.

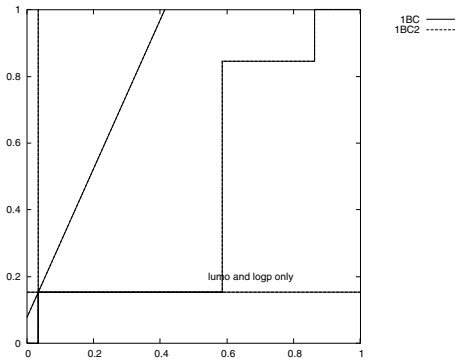
6.2. Alzheimer's disease.

This data set is about drugs against Alzheimer's disease (Boström and Asker, 1999). It aims at comparing four desirable properties of such drugs: (1) inhibit amine reuptake, (2) low toxicity, (3) high acetyl cholinesterase inhibition, and (4) good reversal of scopolamine-induced memory deficiency. Each of these properties leads to a distinct classification task.

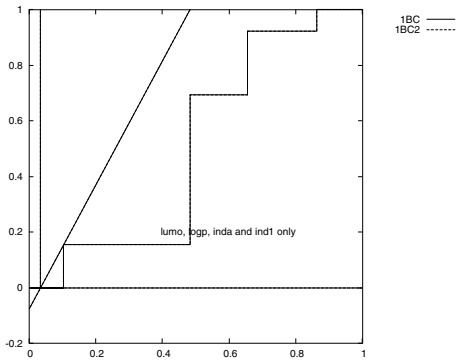
Table 6 compares the accuracies of 1BC and 1BC2 on each of the four properties. RAC is the Reconsider-And-Conquer rule learning algorithm proposed in Boström and Asker (1999). Figure 8 shows the ROC curves obtained from these data sets. The 1BC and 1BC2

Table 6. Accuracy on the Alzheimer's disease data set.

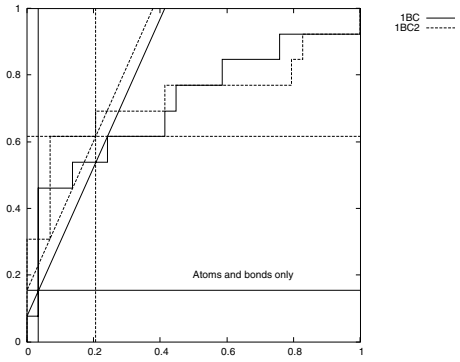
Target	1BC (%)	1BC2 (%)	RAC (%)	Majority class (%)
Inhibit amine reuptake	67.9/79.4	69.1/78.2	85.6	74.9
Low toxicity	74.2/73.8	74.4/74.8	81.8	67.6
High acetyl cholinesterase inhibition	68.1/68.1	69.2/68.3	75.0	51.6
Reversal of memory deficiency	62.5/76.1	67.3/76.4	60.2	76.6



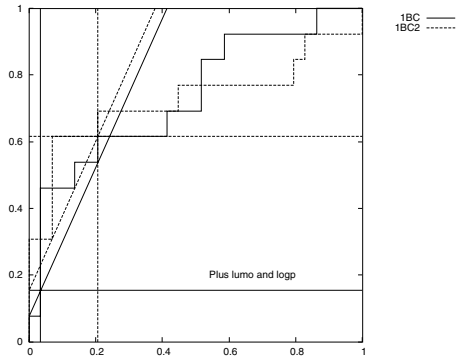
AUC: 0.456 (1BC) 0.456 (1BC2)



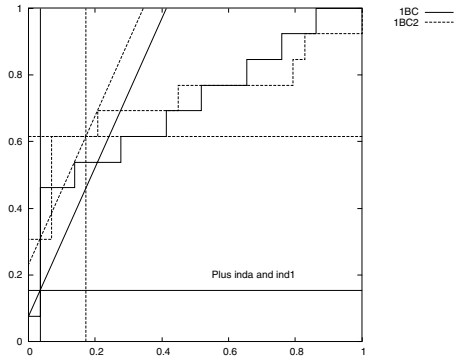
AUC: 0.507 (1BC) 0.507 (1BC2)



AUC: 0.711 (1BC) 0.729 (1BC2)



AUC: 0.737 (1BC) 0.727 (1BC2)



AUC: 0.708 (1BC) 0.727 (1BC2)

Figure 7. ROC curves on the mutagenesis regression-unfriendly data set. The top two curves (propositional data) are poor, and the default threshold on the second data set results in worse than majority class performance. The curves on the relational data sets are better, but the default threshold is sub-optimal in all cases. Notice that the 1BC2 curves are virtually identical on all three relational representations, i.e., the atoms and bonds dominate.

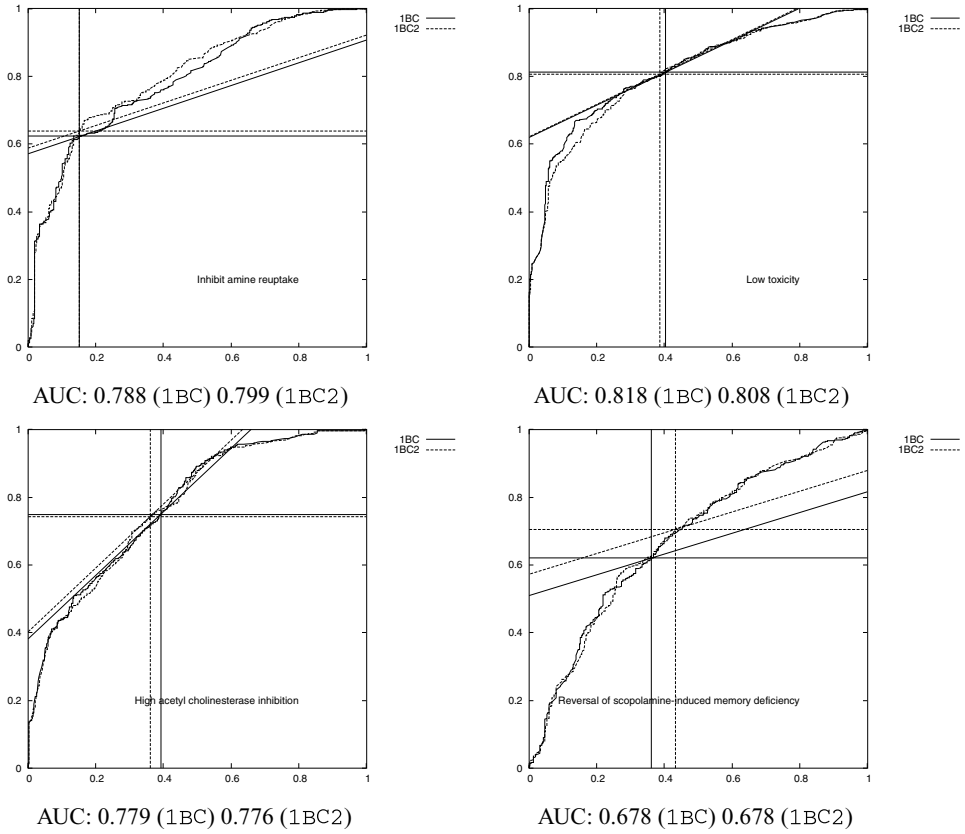


Figure 8. ROC curves on the Alzheimer's disease data set. On the first and fourth curve the default threshold results in worse than majority class performance. Notice with the fourth curve and the given class distribution performance can never improve on majority class classification, whereas on the first data set we can improve on the majority class. This is confirmed by the optimised accuracy results in Table 6.

curves are very similar. For the first and fourth targets, the default decision threshold of 0.5 is clearly sub-optimal. In fact, it can be seen that the optimal point for the fourth target is (1,1)—i.e., the majority class classifier. Notice that on this target RAC performs considerably worse than majority class; on the other targets it outperforms naive Bayes.

6.3. Diterpene structure elucidation

Diterpenes are one of a few fundamental classes of natural products with about 5000 members known (Džeroski et al., 1998). Structure elucidation of diterpenes from ^{13}C NMR spectra (Nuclear magnetic Resonance) can be separated into three main stages: (1) identification of residues (ester and/or glycosides), (2) identification of the diterpene skeleton, and (3) arrangement of the residues on the skeleton. This data set is concerned with the second stage, the identification of the skeleton. A skeleton is a unique connection of

Table 7. Accuracy results on the diterpenes data set. The majority class in all three settings is 29.8%.

Settings	1BC (%)	1BC2 (%)	C4.5 (%)	Foil (%)	RIBL (%)
Propositional	78.2/78.8	78.2/78.8	78.5	70.1	79.0
Relational	63.3/66.9	68.8/66.5		46.5	86.5
Propositional and relational	70.9/79.8	81.9/81.9		78.3	91.2

20 carbon atoms, each with a specific atom number and, normalized to a pure skeleton molecule without residues, a certain multiplicity (s, d, t or q) measuring the number of hydrogens directly connected to a particular carbon atom: s stands for singulet, which means there is no proton (i.e., hydrogen) connected to the carbon; d stands for a doublet with one proton connected to the carbon; t stands for a triplet with two protons and q for a quartet with three protons bound to the carbon atom.

The data contains information on 1503 diterpenes with known structure. Two representations are available: a propositional one where the atom numbers are known, and a relational one without this information. In order to compare our results with Džeroski et al. (1998), the accuracy is evaluated with a 3-fold cross-validation, and three settings were considered: each representation separately, and then together. As this is a multi-class classification problem, we do not show ROC graphs. We did, however, apply our method for finding improved decision thresholds.

Table 7 lists our experimental results along selected results from Džeroski et al. (1998). On the propositional data set, 1BC and 1BC2 perform comparable to C4.5 and RIBL (a relational instance-based learner), and better than Foil. On the relational representation they perform considerably better than Foil but considerably worse than RIBL. The combination of propositional and relational features improves 1BC2's performance relative to the other two settings, so in this case a reasonable trade-off is achieved. However, naive Bayes cannot compete with instance-based methods on this data set in the relational settings.

7. Related work

In this section we discuss some of the most closely related work from the literature. Basically, these fall in two groups: approaches in inductive logic programming to incorporate or upgrade the naive Bayesian classifier, and probabilistic models for structured data and first-order logic. We address these two groups in turn.

The earliest paper that considers naive Bayesian classification in an ILP context is by Pompe and Kononenko (1995). Their approach consists in learning a set of first-order rules in a first step, then using them as new attributes in a classical attribute-value naive Bayesian classifier. This is most closely related to the 1BC approach, replacing 1BC's exhaustive feature generation with selective feature construction using a rule learner. On the other hand, their work does not use any data modelling facilities, hence limiting the possibility to adapt the representation to the domain.

The work by Slattery and Craven (1998) goes in the other direction: it uses naive Bayes to invent new predicates to be used by Foil. The invented predicates are Boolean features

of the individual. They demonstrate increased performance over the constituent learners on various text classification data sets. In principle, 1BC or 1BC2 could be used in a similar way to invent more complex predicates for use by Foil.

More recently, Ceci, Appice, and Malerba (2003) presented Mr-SBC, a multi-relational simple Bayesian classifier, which is fairly similar to 1BC. They use the same knowledge representation involving binary structural predicates and properties, and consider features consisting of a path of structural predicates followed by a single property. The main difference is that Mr-SBC estimates the probabilities of the structural path and the property separately. This allows storing the path probabilities, as well as computing the probability of a longer path from the probability of a prefix. Finally, Mr-SBC interfaces directly with a relational database.

There is an extensive body of work on probabilistic models for structured data; here we discuss only a selection. Among the best-known are Koller's probabilistic relational models (PRMs) (Getoor et al., 2001). These are an upgrade of Bayesian networks that define probabilistic dependencies on top of an entity-relationship model. If there is a one-to-many relationship between entities E_1 and E_2, a probabilistic dependency of E_1 on E_2 is modelled by means of some aggregation of the linked E_2 objects. Thus, PRMs are closely related to 1BC, both with regards to knowledge representation and the use of aggregation to deal with non-determinacy. The difference is that PRMs are not individual-centred and thus the same model is learned from data regardless of whether, e.g., molecules or atoms are the individuals. Later developments that also aggregate over links include (Taskar, Segal, and Koller, 2001; Lu and Getoor, 2003).

Approaches that combine logic programs with probabilities include stochastic logic programs (SLPs) (Muggleton, 1996; Cussens, 2001), Bayesian logic programs (Kersting and De Raedt, 2000; Kersting and De Raedt, 2001), and PRISM (programming in statistical modelling) (Sato and Kameya, 1997, 2001). The last two treat all ground atoms in the Herbrand base of the program as random variables over which probability distributions are defined. In contrast, SLPs define probability distributions over terms. The latter approach is closely related to 1BC2. As we showed in Section 4.1, the geometric list distribution can be generated by an SLP. If we only use this distribution for non-determinate links, the ISP data model can be transformed into an equivalent SLP, and the training phase of 1BC2 would correspond to estimating the probabilistic parameters of that SLP.

While close relationships exist between 1BC and probabilistic relational models on the one hand, and 1BC2 and stochastic logic programs on the other, there are clear advantages in employing a simple probabilistic model like naive Bayes. For instance, while general parameter estimation procedures exist for SLPs (Cussens, 2001), they are bound to be computationally expensive and slow to converge. In contrast, the maximum likelihood estimate of the τ parameter of the geometric distribution is simply derived from the average cardinality.

Finally, we mention our own ongoing work on hierarchical and higher-order Bayesian networks (Gyftodimos and Flach, 2003; Flach, Gyftodimos, and Lachiche, to appear). The idea here is to lift the naive Bayesian independence assumptions by defining probabilistic models on a higher-order type structure. The probabilistic decomposition employed by 1BC2 would be a special case of a higher-order Bayesian network, in a similar way as

the propositional naive Bayesian classifier is a special case of a propositional Bayesian network.

8. Conclusions and future work

We presented two approaches to upgrade the propositional naive Bayesian classifier to deal with structured data. One approach is based on upgrading attributes to first-order features, i.e., it works in the language space. This approach has been implemented in the 1BC system. The second approach generalises the naive Bayes estimate of the probability of a tuple of elements from probabilities of its components, to collections of elements. It works in the individual space, and has led to the 1BC2 system. Both systems are available on-line.[5]

This paper provides an unified view of the 1BC and 1BC2 approaches. In particular, each approach has been presented as one way of upgrading the propositional naive Bayesian classifier. Compared to Lachiche and Flach (2002), a context dependency has been added that takes into account explicitly the structure of the object, and enables 1BC2 to deal with objects involving similar components at different locations in their structure. That context is the key to a recursive implementation of 1BC, compared to the previous non-recursive implementation (Flach and Lachiche, 1999). Now 1BC inherits the efficiency of 1BC2, which is much better than the typical propositionalisation approach. Propositionalisation (i.e., mapping to a different feature space) is now performed only implicitly, in the spirit of kernel methods.

We have compared the two systems both with respect to the probabilities they compute, and regarding their performance on artificially constructed data sets. These experiments confirm that 1BC2 outperforms 1BC on problems where modelling the cardinality of collections or the distributions of elements of collections is important. On the other hand, they also demonstrate that 1BC2 requires more data to estimate these probabilities reliably. Experimental results have been obtained on a range of benchmark ILP data sets. They confirm that the two systems perform comparably to other ILP systems.

Probabilistic classifiers are often criticised because they do not induce declarative, interpretable models. However, an often overlooked advantage is that the probabilistic model is a ranker rather than a classifier, and thus can be calibrated to improve classification performance. Using ROC analysis we have shown that the default decision threshold used to turn the ranker into a classifier is often sub-optimal. In Lachiche and Flach (2003), we suggest an algorithm to improve the accuracy and cost of a probabilistic classifier on two-class and multi-class problems. This includes both the 1BC and 1BC2 systems and works on structured data as well as on attribute-value data.

In the propositional case, the naive Bayesian decomposition leads to loss of probabilistic information but not of logical information. In the relational case, logical information is lost as well. This means that relational naive Bayesian classifiers such as the ones proposed in this paper can be expected to be sensitive to the representation. In our experiments so far we have not paid particular attention to this, and have worked with the simplest or most obvious representation. In future work we plan to analyse the effect of the independence assumptions made by 1BC and 1BC2, as well as the effect of the representation, in detail.

Another issue concerns continuous properties (e.g., electric charge of an atom). Currently they are dealt with either as nominal values or by discretisation in pre-processing. One option would be to directly estimate the probability of a continuous property (John and Langley, 1995). Another option would be a propositionalisation based on various aggregation operators (Knobbe, de Haas, and Siebes, 2001). Our aim is to make the naive Bayesian classifier effective on structured data, whatever its complexity.

An important feature of many ILP systems is the incorporation of logical background knowledge. Currently, our systems do not allow this, and we plan to alleviate this restriction in future work. A good starting point might be the work of Dehaspe (1997) who combines a maximum entropy approach with logical constraints.

Finally, we plan to look at the effect of using different probability distributions over collections. Apart from a bitvector distribution (which corresponds to a propositionalisation), the geometric list distribution is one of the simplest ways of modelling probabilities of collections. The main aim of our work so far on `1BC2` was to investigate how much improvement we could get with such simple means. The next step is to see whether further improvements can be achieved with more sophisticated distributions over collections.

Acknowledgments

Part of this work was supported by the EU Framework V project *Data Mining and Decision Support for Business Competitiveness: Solomon Virtual Enterprise* (IST-1999-11495). Thanks are due to Henrik Boström and Sašo Džeroski for providing us with the Alzheimer and Diterpene data sets, respectively. We also thank the three anonymous reviewers for their extensive and insightful comments, which helped us to considerably improve the paper.

Notes

1. Integers, reals, enumerated types etc. are atomic; terms of other types (tuples, lists, trees, sets, etc.) are non-atomic or compound, that is, they are composed of subterms of types lower in the type structure.
2. Badea and Stanciu (1999) independently conceived a similar refinement operator and showed it is weakly perfect.
3. This only holds if there is exactly one refutation for each instantiation of L, which is the case here.
4. The optimised threshold obtained with the method from Lachiche and Flach (2003) cannot be indicated, because in general a different threshold is obtained in each fold of the cross-validation.
5. http://www.cs.bris.ac.uk/Research/MachineLearning/1BC/

References

Badea, L., & Stanciu, M. (1999). Refinement operators can be (weakly) perfect. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming* (pp. 21–32). Springer-Verlag.

Boström, H., & Asker, L. (1999). Combining divide-and-conquer and separate-and-conquer for efficient and effective rule induction. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming* (pp. 33–43). Springer-Verlag.

Ceci, M., Appice, A., & Malerba, D. (2003). Mr-SBC: A multi-relational naïve Bayes classifier. In *Proceedings of the Seventh European Conference on Principles and Practice of Knowledge Discovery in Databases* (pp. 95–106). Springer-Verlag.

Cussens, J. (2001). Parameter estimation in stochastic logic programs. *Machine Learning,* 43, 245–271.

Date, C. (1995). *An introduction to database systems*. Addison Wesley.

Dehaspe, L. (1997). Maximum entropy modeling with clausal constraints. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming* (pp. 109–124). Springer-Verlag.

Dietterich, T., Lathrop, R., & Lozano-Perez, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89, 31–71.

Dolšak, B., & Muggleton, S. (1992). The application of inductive logic programming to finite-element mesh design. In S. Muggleton (Ed.), *Inductive logic programming*. Academic Press.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning,* 29, 103–130.

Džeroski, S., Schulze-Kremer, S., Heidtke, K., Siems, K., Wettschereck, D., & Blockeel, H. (1998). Diterpene structure elucidation from ^{13}C NMR spectra with inductive logic programming. *Applied Artificial Intelligence,* 12, 363–383.

Džeroski, S., & Lavrač, N. (Eds.). (2001). *Relational data mining*. Springer-Verlag.

Fawcett, T. (2003). *ROC graphs: Notes and practical considerations for data mining researchers*. Technical report HPL-2003-4. HP Laboratories, Palo Alto, CA, USA. Available at http://www.hpl.hp.com/techreports/2003/HPL-2003-4.pdf.

Flach, P., Giraud-Carrier, C., & Lloyd, J. (1998). Strongly typed inductive concept learning. *Proceedings of the Eighth International Conference on Inductive Logic Programming* (pp. 185–194). Springer-Verlag.

Flach, P., & Lachiche, N. (1999). 1BC: A first-order Bayesian classifier. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming* (pp. 92–103). Springer-Verlag.

Flach, P. (1999). Knowledge representation for inductive learning. *Symbolic and Quantitative Approaches to Reasoning and Uncertainty* (pp. 160–167). Springer-Verlag.

Flach, P., Gyftodimos, E., & Lachiche, N. (to appear). Probabilistic reasoning with terms. *Linkoping Electronic Articles in Computer and Information Science*. Available at http://www.ida.liu.se/ext/epa/cis/2002/011/tcover.html.

Flach, P., & Lachiche, N. (2001). Confirmation-guided discovery of first-order rules with tertius. *Machine Learning*, 42, 61–95.

Gärtner, T., Lloyd, J., & Flach, P. (2004). Kernels and distances for structured data. *Machine Learning*, 57:3, 205–232.

Getoor, L., Friedman, N., Koller, D., & Pfeffer, A. (2001). Learning probabilistic relational models. In S. Džeroski and N. Lavrač (Eds.), *Relational data mining*. Springer-Verlag.

Gyftodimos, E., & Flach, P. (2003). Hierarchical Bayesian networks: an approach to classification and learning for structured data. In *Proceedings of the Work-in-Progress Track at the Thirteenth International Conference on Inductive Logic Programming* (pp. 12–21). Department of Informatics, University of Szeged.

Hand, D., & Till, R. (2001). A simple generalisation of the Area Under the ROC Curve for multiple class classification problems. *Machine Learning*, 45, 171–186.

John, G., & Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence* (pp. 338–345). Morgan Kaufmann.

Kersting, K., & De Raedt, L. (2000). Bayesian logic programs. In *Proceedings of the Work-in-Progress Track at the Tenth International Conference on Inductive Logic Programming* (pp. 138–155). CEUR Workshop Proceedings Series, Vol. 35.

Kersting, K., & De Raedt, L. (2001). Towards combining inductive logic programming with Bayesian networks. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming* (pp. 118–131). Springer-Verlag.

Knobbe, A., de Haas, M., & Siebes, A. (2001). Propositionalisation and aggregates. In *Proceedings of the Fifth European Conference on Principles of Data Mining and Knowledge Discovery* (pp. 277–288). Springer-Verlag.

Kramer, S., Lavrač, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. In S. Džeroski and N. Lavrač (Eds.), *Relational data mining*. Springer-Verlag.

Krogel, M.-A., & Wrobel, S. (2001). Transformation-based learning using multirelational aggregation. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming* (pp. 142–155). Springer-Verlag.

Lachiche, N., & Flach, P. (2002). 1BC2: a true first-order Bayesian classifier. In *Proceedings of the Twelfth International Conference on Inductive Logic Programming* (pp. 133–148). Springer-Verlag.

Lachiche, N., & Flach, P. (2003). Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC curves. In *Proceedings of the Twentieth International Conference on Machine Learning* (pp. 416–423). AAAI Press.

Lavrač, N., & Flach, P. (2001). An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, 2, 458–494.

Lavrač, N., Železný, F., & Flach, P. (2002). RSD: Relational subgroup discovery through first-order feature construction. In *Proceedings of the Twelfth International Conference on Inductive Logic Programming* (pp. 149–165). Springer-Verlag.

Lloyd, J. (1999). Programming in an integrated functional and logic language. *Journal of Functional and Logic Programming*, *1999*.

Lloyd, J. (2003). *Logic for learning: learning comprehensible theories from structured data*. Springer-Verlag.

Lu, Q., & Getoor, L. (2003). Link-based classification. In *Proceedings of the Twentieth International Conference on Machine Learning* (pp. 496–503). AAAI Press.

Muggleton, S. (Ed.). (1992). *Inductive logic programming*. Academic Press.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing,* 13, 245–286.

Muggleton, S. (1996). Stochastic logic programs. In L. De Raedt (Ed.), *Advances in inductive logic programming*. IOS Press.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20, 629–679.

Muggleton, S., Srinivasan, A., King, R., & Sternberg, M. (1998). Biochemical knowledge discovery using Inductive Logic Programming. In *Proceedings of the first International Conference on Discovery Science* (pp. 326–341). Springer-Verlag.

Pompe, U., & Kononenko, I. (1995). Naive Bayesian classifier within ILP-R. In *Proceedings of the Fifth International Workshop on Inductive Logic Programming* (pp. 417–436). Department of Computer Science, Katholieke Universiteit Leuven.

Rouveirol, C. (1994). Flattening and saturation: Two representation changes for generalization. *Machine Learning,* 14, 219–232.

Sato, T., & Kameya, Y. (1997). Prism: A symbolic-statistical modeling language. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (pp. 1330–1335). Morgan Kaufmann.

Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15, 391–454.

Slattery, S., & Craven, M. (1998). Combining statistical and relational methods for learning in hypertext domains. In *Proceedings of the Eighth International Conference on Inductive Logic Programming* (pp. 38–52). Springer-Verlag.

Srinivasan, A., Muggleton, S., King, R., & Sternberg, M. (1994). Mutagenesis: ILP experiments in a non-determinate biological domain. In *Proceedings of the Fourth International Workshop on Inductive Logic Programming* (pp. 217–232). Gesellschaft für Mathematik und Datenverarbeitung MBH.

Taskar, B., Segal, E., & Koller, D. (2001). Probabilistic clustering in relational data. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence* (pp. 870–87). Morgan Kaufmann.

Received March 21, 2003
Revised June 13, 2004
Accepted June 14, 2004
Final manuscript June 17, 2004