

1BC2: A True First-Order Bayesian Classifier

Nicolas Lachiche¹ and Peter A. Flach²

¹ LSIT - Université Robert Schuman, France
lachiche@lsiit.u-strasbg.fr

² Department of Computer Science, University of Bristol, United Kingdom
Peter.Flach@bristol.ac.uk

Abstract. In previous work [3] we presented 1BC, a first-order Bayesian classifier. 1BC applies dynamic propositionalisation, in the sense that attributes representing first-order features are generated exhaustively within a given feature bias, but during learning rather than as a pre-processing step. In this paper we describe 1BC2, which learns from structured data by fitting various parametric distributions over sets and lists to the data. We evaluate the feasibility of the approach by various experiments.

1 Introduction

In its general form, a Bayesian classifier predicts the most likely class value c of an object given its description d : $\operatorname{argmax}_c P(c|d)$. Applying Bayes theorem, this formula can be turned into: $\operatorname{argmax}_c \frac{P(d|c)P(c)}{P(d)} = \operatorname{argmax}_c P(d|c)P(c)$. In typical machine learning problems, the number of possible descriptions of individuals is much greater than the number of available examples, therefore it is impossible, in general, to get a good estimate of $P(d|c)$. This is the usual bias-variance tradeoff [4]: a fundamental challenge of constructing classifiers is finding the right complexity of hypothesis (greater complexity generally decreases bias but increases variance) for a given amount of data (more data decreases variance). In this paper, we will consider a single change of complexity, relying on the naive Bayes assumption.

In an attribute-value representation, the object is described by its values a_1, \dots, a_n for a fixed set of attributes A_1, \dots, A_n . Since it is difficult to get a good estimate of $P(a_1, \dots, a_n|c)$, the naive Bayes assumption consists in decomposing this probability distribution into the product of the probability of each attribute value: $P(a_1|c) \times \dots \times P(a_n|c)$, assuming that the probability of attribute A_i taking on value a_i is independent of the probabilities of the values taken by the other attributes. Roughly, since the counting of the examples of class c having the description a_1, \dots, a_n is difficult, it is evaluated by counting the examples of class c having the description a_i and by combining those estimates.

While the decomposition of probability distributions on attribute-value tuples is well understood, we are not aware of any studies of the decomposition of probability distributions on structured individuals. There have been works on Bayesian classification on structured individuals. Pompe and Kononenko describe an application of naive Bayesian classifiers in a first-order context [9]. 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. The 1BC system we developed [3] does not learn first-order rules, it generates instead a set of first-order conditions, that are used then as attributes in a classical attribute-value naive Bayesian classifier. It can thus be classified as a propositionalisation approach (although the propositionalisation is done dynamically, not in a pre-processing step as in LINUS [7], and it considers only satisfied features, i.e. the one occurring in the individual). In this paper we propose a different approach that is less ‘propositional’ and directly considers probability distributions on structured individuals made of sets, tuples and multisets.

We start by considering probability distributions over first- and higher-order terms in Section 2. In particular, we obtain a distribution over sets which is different from the standard bitvector distribution in that it only depends on the probabilities of its elements. Section 3 describes the 1BC2 system including its main algorithm and the declarative bias it employs. Section 4 describes the results of our experiments. Section 5 concludes.

2 Probability Distributions over Lists and Sets

In this section we assume a finite *alphabet* $A = \{x_1, \dots, x_n\}$ of atomic objects (e.g. integers or characters – we are only dealing with categorical data analysis in this paper), and we consider the question: how to define probability distributions ranging over lists and sets of elements from A ?

2.1 Probability Distributions over Lists

We can define a uniform probability distribution over lists if we consider only finitely many of them, say, up to and including length L . There are $\frac{n^{L+1}-1}{n-1}$ of those for $n > 1$, so under a uniform distribution every list has probability $\frac{n-1}{n^{L+1}-1}$ for $n > 1$, and probability $\frac{1}{L+1}$ for $n = 1$. Clearly, such a distribution does not depend on the internal structure of the lists, treating each of them as equiprobable.

A slightly more interesting case includes a probability distribution over lengths of lists. This has the additional advantage that we can define distributions over all (infinitely many) lists over A . For instance, we can use the geometric distribution over list lengths: $P_\tau(l) = \tau(1-\tau)^l$, with parameter τ denoting the probability of the empty list. Of course, we can use other infinite distributions, or arbitrary finite distributions, as long as they sum up to 1. The geometric distribution corresponds to the head-tail representation of lists.

We then need, for each list length l , a probability distribution over lists of length l . We can again assume a uniform distribution: since there are n^l lists of length l , we would assign probability n^{-l} to each of them. Combining the two distributions over list lengths and over lists of fixed length, we assign probability $\tau(\frac{1-\tau}{n})^l$ to any list of length l . Such a distribution only depends on the length of the list, not on the elements it contains.

We can also assume a probability distribution P_A over the alphabet, and use this to define a non-uniform distribution over lists of length l . For instance, among the lists of length 3, list $[a, b, c]$ would have probability $P_A(a)P_A(b)P_A(c)$, and so would its 5 permutations. Combining P_A and P_τ thus gives us a distribution over lists which depends on the length and the elements of the list, but ignores their positions or ordering.

Definition 1 (Distribution over lists). *The following defines a probability distribution over lists:*

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

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

Introducing an extended alphabet $A' = \{\epsilon, x_1, \dots, x_n\}$ and a renormalised distribution $P_{A'}(\epsilon) = \tau$ and $P_{A'}(x_i) = (1 - \tau)P_A(x_i)$, we have $P_{\text{li}}([x_{j_1}, \dots, x_{j_l}]) = P_{A'}(\epsilon) \prod_{i=1}^l P_{A'}(x_{j_i})$. That is, under P_{li} we can view each list as an infinite tuple of finitely many independently chosen elements of the alphabet, followed by the stop symbol ϵ representing an infinite empty tail.

Example 1 (Order-independent distribution over lists). Consider an alphabet $A = \{a, b, c\}$, and suppose that the probability of each element occurring is estimated as $P_A(a) = .2$, $P_A(b) = .3$, and $P_A(c) = .5$. Taking $\tau = (1 - .2)(1 - .3)(1 - .5) = .28$, i.e. using the bitvector estimate of an empty list, but another estimate could be used (cf. 3.2), we have $P_{A'}(a) = (1 - .28) * .2 = .14$, $P_{A'}(b) = .22$, and $P_{A'}(c) = .36$, and $P_{\text{li}}([a]) = .28 * .14 = .04$, $P_{\text{li}}([b]) = .06$, $P_{\text{li}}([c]) = .10$, $P_{\text{li}}([a, b]) = .28 * .14 * .22 = .009$, $P_{\text{li}}([a, c]) = .014$, $P_{\text{li}}([b, c]) = .022$, and $P_{\text{li}}([a, b, c]) = .28 * .14 * .22 * .36 = .003$. We also have, e.g., $P_{\text{li}}([a, b, b, c]) = .28 * .14 * .22 * .22 * .36 = .0007$.

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 alphabet, so that $[a, b, c]$ would have probability $P_{A^2}(ab)P_{A^2}(bc)$ among the lists of length 3. To include lists of length 0 and 1, we can add special start and stop symbols to the alphabet. Such a distribution would take some aspects of the ordering into account, but note that $[a, a, b, a]$ 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.

A different approach is obtained by taking not ordering but position into account. For instance, we can have three distributions $P_{A,1}$, $P_{A,2}$ and $P_{A,3+}$ over the alphabet, for positions 1, 2, and 3 and higher, respectively. Among the lists of length 4, the list $[a, b, c, d]$ would get probability $P_{A,1}(a)P_{A,2}(b)P_{A,3+}(c)P_{A,3+}(d)$; so would the list $[a, b, d, c]$.

In summary, all except the most trivial probability distributions over lists involve (i) a distribution over lengths, and (ii) distributions over lists of fixed length. The latter take the list elements, ordering and/or position into account. We do not claim any originality with respect to the above distributions, as these and similar distributions are commonplace in e.g. bioinformatics. In the next section we use list distributions to define distributions over sets and multisets. Notice that, while lists are first-order terms, sets represent predicates and therefore are higher-order terms. In this respect our work extends related work on probability distributions over first-order terms.

2.2 Probability Distributions over Sets and Multisets

A multiset (also called a bag) differs from a list in that its elements are unordered, but multiple elements may occur. Assuming some arbitrary total ordering on the alphabet, each multiset has a unique representation such as $\{[a, b, b, c]\}$. Each multiset can be mapped to the equivalence class of lists consisting of all permutations of its elements. For instance, the previous multiset corresponds to 12 lists. Now, given any probability distribution over lists that assigns equal probability to all permutations of a given list, this provides us with a method to turn such a distribution into a distribution over multisets. In particular, we can employ P_{li} above which defines the probability of a list, among all lists with the same length, as the product of the probabilities of their elements.

Definition 2 (Distribution over multisets). *For any multiset s , let l stand for its cardinality, and let k_i stand for the number of occurrences of the i -th element of the alphabet. The following defines a probability distribution over multisets:*

$$P_{ms}(s) = \frac{l!}{k_1! \dots k_n!} P_{li}(s) = l! \tau \prod_i \frac{P_{A'}(x_i)^{k_i}}{k_i!}$$

where τ is a parameter giving the probability of the empty multiset.

Here, $\frac{l!}{k_1! \dots k_n!}$ stands for the number of permutations of a list with possible duplicates.

Example 2 (Distribution over multisets). Continuing Example 1, we have $P_{ms}(\{[a]\}) = .04$, $P_{ms}(\{[b]\}) = .06$, $P_{ms}(\{[c]\}) = .10$ as before. However, $P_{ms}(\{[a, b]\}) = .02$, $P_{ms}(\{[a, c]\}) = .03$, $P_{ms}(\{[b, c]\}) = .04$, $P_{ms}(\{[a, b, c]\}) = .02$, and $P_{ms}(\{[a, b, b, c]\}) = .008$.

This method, of defining a probability distribution over a type by virtue of that type being isomorphic to a partition of another type for which a probability distribution is already defined, is more generally applicable. Although in the above case we assumed that the distribution over each block in the partition is uniform, so that we only have to count its number of elements, this is not a necessary condition. Indeed, blocks in the partition can be infinite, as long as we can derive an expression for its cumulative probability. We will now proceed to derive a probability distribution over sets from distribution P_{li} over lists in this manner.

Consider the set $\{a, b\}$. It 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 alphabet besides a and b . The cumulative probability of lists of the second type is easily calculated.

Lemma 1. *Consider a subset S of l elements from the alphabet, with cumulative probability $P_{A'}(S) = \sum_{x_i \in S} P_{A'}(x_i)$. The cumulative probability of all lists of length at least l containing only elements from S is $f(S) = \tau \frac{(P_{A'}(S))^l}{1 - P_{A'}(S)}$.*

Proof. We can delete all elements in S from the alphabet and replace them by a single element x_S with probability $P_{A'}(S)$. The lists we want consist of l or more occurrences of x_S . Their cumulative probability is

$$\sum_{j \geq l} \tau (P_{A'}(S))^j = \tau \frac{(P_{A'}(S))^l}{1 - P_{A'}(S)}$$

$f(S)$ as defined in Lemma 1 is not a probability, because the construction in the proof includes lists that do not contain all elements of S . For instance, if $S = \{a, b\}$ they include lists containing only a 's or only b 's. More generally, for arbitrary S the construction includes lists over every possible subset of S , which have to be excluded in the calculation of the probability of S . In other words, the calculation of the probability of a set iterates over its subsets.

Definition 3 (Subset-distribution over sets). Let S be a non-empty subset of l elements from the alphabet, and define

$$P_{\text{ss}}(S) = \sum_{S' \subseteq S} (-P_{A'}(S'))^{l-l'} * f(S')$$

where l' is the cardinality of S' , and $f(S')$ is as defined in Lemma 1. Furthermore, define $P_{\text{ss}}(\emptyset) = \tau$. $P_{\text{ss}}(S)$ is a probability distribution over sets.

Example 3 (Subset-distribution over sets). Continuing Example 2, we have $P_{\text{ss}}(\emptyset) = \tau = .28$, $P_{\text{ss}}(\{a\}) = f(\{a\}) = \tau \frac{P_{A'}(\{a\})}{1 - P_{A'}(\{a\})} = .05$, $P_{\text{ss}}(\{b\}) = .08$, and $P_{\text{ss}}(\{c\}) = .16$. Furthermore, $P_{\text{ss}}(\{a, b\}) = f(\{a, b\}) - P_{A'}(\{a\}) * f(\{a\}) - P_{A'}(\{b\}) * f(\{b\}) = .03$, $P_{\text{ss}}(\{a, c\}) = .08$, and $P_{\text{ss}}(\{b, c\}) = .15$. Finally, $P_{\text{ss}}(\{a, b, c\}) = f(\{a, b, c\}) - P_{A'}(\{a, b\}) * f(\{a, b\}) - P_{A'}(\{a, c\}) * f(\{a, c\}) - P_{A'}(\{b, c\}) * f(\{b, c\}) + (P_{A'}(\{a\}))^2 * f(\{a\}) + (P_{A'}(\{b\}))^2 * f(\{b\}) + (P_{A'}(\{c\}))^2 * f(\{c\}) = .18$.

P_{ss} takes only the elements occurring in a set into account, and ignores the remaining elements of the alphabet. For instance, the set $\{a, b, c\}$ will have the same probability regardless whether there is one more element d in the alphabet with probability p , or 10 more elements with cumulative probability p . This situation is analogous to lists.

In this section, we defined probability distributions over lists, sets and multisets. Those probability distributions allowed us to build up a true first-order naive Bayesian classifier that can deal with structured data involving nested tuples, lists, sets and multisets: 1BC2.

3 The 1BC2 System

In this section we describe the main features of the 1BC2 system: first the flattened Prolog representation and declarations used to describe the input data, then the main algorithm to fit parametrised probability distributions to hierarchically structured training data. A typewriter font will be used to denote examples of data and algorithms.

3.1 Data Representation

1BC2 makes use of a data representation similar to the one of 1BC [3], i.e. it distinguishes structural predicates from properties.

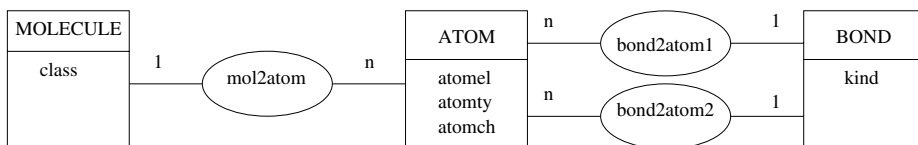


Fig. 1. ER representation of mutagenesis.

Definition 4 (Structural predicate). A structural predicate is a binary predicate representing the relation between one type and another. Given an individual of one type, a functional structural predicate, or structural function, refers to a unique individual of the other type, while a non-determinate structural predicate is non-functional.

Definition 5 (Property). A property is a predicate characterising an individual. A parameter is an argument of a property which is always instantiated. If a property has no parameter (or only one instantiation of its parameters), it is boolean, otherwise it is multivalued.

1BC2 makes use of an ISP (Individual-Structural-Properties) declaration where all structural predicates and properties are declared. Actually the declaration of the individuals is optional since they can be inferred from the other declarations: Indeed individuals are non parameter arguments of properties and arguments of structural predicates. Here is the ISP declaration for mutagenesis:

```
--INDIVIDUAL
mol 1 mol
--STRUCTURAL
mol2atom 2 1:mol *:atom 1 li
bond2atom1 2 *:bond 1:atom 1 li
bond2atom2 2 *:bond 1:atom 1 li
--PROPERTIES
class 2 mol #class 1
atome1 2 atom #element 1
atomey 2 atom #number 1
atomch 2 atom #charge 1
kind 2 bond #kind 1
```

Parameters are denoted by a hash: class, element, number, charge, and kind. Other domains (mol, atom, and bond) are considered as individuals. Each line defining one predicate contains: the predicate name, its arity, the domain of each argument, and the number of times this predicate can be used (useful either to disable some predicates, or to limit loops with structural predicates). Structural predicates require one more column: the kind of decomposition that should be used (bv, li, ms, ss). Non-determinate structural predicates are denoted with a star when several individuals (e.g. atoms) can be linked to a single individual (e.g. molecule).

This data representation is in between an Entity-Relationship model (cf. Figure 1) and a relational model [6]: each entity is represented by one individual, it can have some

properties, and it can have some relationships (structural predicates) with other entities. For instance, in mutagenesis domain, molecules involve atoms and bonds. So there are three individuals: `mol`, `atom` and `bond`. To keep a representation compatible with previous representations, in order to compare our results, we consider three structural predicates: `mol2atom`, `bond2atom1`, and `bond2atom2`. The first structural predicate links a molecule to its atoms, the second and third structural predicates respectively link a bond to its first atom and to its second atom. Indeed in previous representations, bonds were represented by an ordered pair of atoms. Finally, the molecule has a single property: its `class`. An atom has three separate properties: its atomic element `atome1`, its atomic type `atomy` and its charge. A bond is characterised by a single property: its `kind`.

3.2 The 1BC2 Algorithm

1BC2 is given a target predicate, e.g. `class`. Let us first detail the classification phase, then the learning phase.

The main function is `getClassDistribution(individual i)`. It returns a table with an estimate of the probabilities of individual `i` for each class. We will use uppercase `P` to denote probabilities (actually they are the Laplace estimates from counts).

```

getClassDistribution(i) {
  CD = prior class distribution
  For all properties prop of i
    Find parameter value v such that prop(i,v)
    For each class c
      CD[c] = CD[c] x P(v|c)
  For all structural predicates struc involving i
    If struc is functional given i then
      Find the related individual j
      CD' = getClassDistribution(j)
      For each class c, CD[c] = CD[c] x CD'[c]
    Else /* non-determinate struc. pred. li/ms/ss */
      Find all related individuals J
      For each individual j of J
        CD''[j] = getClassDistribution(j)
      For each class c
        CD[c] = CD[c] x estimateP(li/ms/ss,CD'',c)
  Return CD}

```

In order to deal with lists, multisets and sets, `estimateP(li/ms/ss,CD'',c)` applies the appropriate formula (Definitions 1,2,3) using `CD''[j][c]` as the probability of each element `j`.

Example 4. Here is a partial description of molecule `d1`:

```

class(d1,mutagenic).          atomch(d1_1,-0.117).
mol2atom(d1,d1_1).          ...

```

```

mol2atom(d1, d1_2) .          bond2atom1(d1_1d1_2, d1_1) .
...                          bond2atom2(d1_1d1_2, d1_2) .
atomel(d1_1, c) .           kind(d1_1d1_2, 7) .
...                          ...
atomty(d1_1, 22) .
...

```

In order to classify molecule `d1` in mutagenesis, `1BC2` estimates its probability given each class using the properties of that individual, and the probabilities of individuals it is linked to. Since a molecule has no properties but the target predicate class, it considers the structural predicate: `mol2atom`. A molecule can have several atoms (`*:atom` in the ISP declaration of `mol2atom`) and P_i estimate (`li` at the end of the line) will be used (cf. Section 2). In order to apply P_i , `1BC2` requires the probability of each atoms of molecule `d1`: `d1_1`, `d1_2`, ... given a class `c`. Then the same algorithm is applied recursively! The probability of atom `d1_1` is estimated using its properties: `atomel(d1_1, c)`, `atomty(d1_1, 22)`, and `atomch(d1_1, -0.117)`, and the probabilities of the individuals it is linked to: `bond2atom1(d1_1d1_2, d1_1)`, `bond2atom2(d1_6d1_1, d1_1)`, `bond2atom1(d1_1d1_7, d1_1)`. First for each class the a priori probability of that class is multiplied by the estimates of the probability of a carbon atom, the probability of an atomic number of 22, and the probability of a -0.117 charge for that class. Then it requires to estimate the probabilities of each bond, e.g. `d1_1d1_2`. Again the same algorithm is applied. In order to avoid to loop and come back to atoms from the bond, a limit on the number of times a structural predicate can be used is fixed by the user in the ISP declaration. Those limits are sensitive parameters. An accurate representation of individuals and consequently estimation of their probabilities depend on them. As illustrated by this example, a value of 1 prevents cycles when they are undesirable. When cycles are allowed, e.g. the tail predicate in lists, then the user could fix the limit to the maximal observed length. So here the probability of bond `d1_1d1_2` is estimated using its property: `kind(d1_1d1_2, 7)`. We will detail in next paragraph how the probability of a bond being of kind 7 given each class is estimated. Those probabilities are then returned as the probabilities of bond `d1_1d1_2`. The same is done for all bonds in which atom `d1` occurs. Then, given those probabilities, P_i formula is used to get the probability of atom `d1_1` for each class. We will detail in next paragraph how τ is estimated. The same is done for all atoms of molecule `d1`. Finally P_i formula is used again to get the probability of the molecule given a list of its atoms.

The learning phase has to estimate the probabilities of empty lists, e.g. of atoms and of bonds, and the probabilities of each property, for each class. The current implementation relies on counting how many individuals satisfy each value of the property, taking into account the target predicate. For instance, it counts how many carbon atoms belong to mutagenic molecules, and how many carbon atoms belong to non-mutagenic molecules. It does the same for all parameters (oxygen, hydrogen, ...) of the `atomel`, and for all parameters of all other properties (atomic type, charge, kind of bond). The laplace estimate is used given those countings. The probability of the empty list is actually the parameter τ of a geometric distribution $P(l) = \tau(1 - \tau)^l$ that has to be estimated given observed values: l_1, l_2, \dots, l_k . We use a maximum likelihood estimates, i.e. τ that maximises the probabilities $p = P(l_1) \times P(l_2) \times \dots \times P(l_k) = \tau(1 - \tau)^{l_1} \dots \tau(1 - \tau)^{l_k}$.

It is the same as maximising $\log(p) = \sum_i (\log(\tau) + l_i \times \log(1 - \tau))$ or $\frac{\log(p)}{k} = \log(\tau) + \bar{l} \times \log(1 - \tau)$, where \bar{l} is the average of l_i . The maximum necessarily corresponds to a zero of the derivative with respect to τ : $\frac{1}{t} - \bar{l} \times \frac{1}{1-\tau} = 0$. Therefore $\tau = \frac{1}{1+\bar{l}}$. So 1BC2 calculates the average number of atoms in molecules and the average number of bonds in atoms, for each class, in order to estimate respective probabilities of the empty lists. Of course, different estimates could be used.

4 Experiments

In this section we describe experimental results on several domains: Alzheimer’s disease, mutagenesis and diterpene structure elucidation.

Experiments have been carried out with the list distribution (Definitions 1). The multiset distribution (Definition 2) has not been used since it gives the same results as the list distribution. Indeed, $P_{\text{ms}}(s) = \frac{l!}{k_1! \dots k_n!} P_{\text{li}}(s)$ and $\frac{l!}{k_1! \dots k_n!}$ does not depend on the class, so $\text{argmax}_c P_{\text{ms}}(s|c) = \text{argmax}_c P_{\text{li}}(s)$. The definition of P_{ss} involves a sum over all subsets. The number of subsets is exponential in the number of elements, therefore it is only practical in domains involving small sets, for instance sets in Alzheimer’s disease involve a maximum of 9 elements, those in diterpenes contain 20 elements, but molecules in mutagenesis have up to 40 atoms. On Alzheimer’s disease, we got very close, slightly lower, results for the subset distribution than for the list distribution. We intend to investigate this more fully in future work.

Experiments are evaluated both with accuracy and with ROC curves. *ROC curves* (ROC: Receiver Operating Characteristic) evaluate classifier performance in terms of *false positive rate* $FPr = \frac{FP}{TN+FP}$ (plotted on the X -axis) that needs to be minimized, and *true positive rate* $TPr = \frac{TP}{TP+FN}$ (plotted on the Y -axis) that needs to be maximized. The performance of a categorical classifier on a test set gives rise to a single point in ROC space. However, the performance of a probabilistic classifier on a test set gives rise to a ROC curve connecting (0,0) and (1,1) as described below. The area under this ROC curve (*AUC*) can be used as an evaluation metric. This metric is an aggregate of the expected performance of the classifier under different class and cost distributions.

In order to turn a probabilistic classifier into a categorical one, the decision rule used is usually ‘if the predicted probability of being positive is larger than the predicted probability of being negative then predict positive else negative’. Equivalently, this corresponds to setting a fixed threshold 0.5 on the positive probability: if the positive probability is larger than this threshold we predict positive, else negative. A ROC curve can be constructed by varying this threshold from 1 (all predictions negative, corresponding to (0,0) in ROC space) to 0 (all predictions positive, corresponding to (1,1) in ROC space). This results in $n + 1$ points in the ROC space, where n is the total number of examples. Equivalently, we can order all examples by decreasing predicted probability of being positive, and trace the ROC curve by starting in (0,0), stepping up when the example is actually positive and stepping to the right when it is negative, until we reach (1,1)¹. The area under this ROC curve indicates the aggregated quality of all classifiers

¹ In the case of ties, we make the appropriate number of steps up and to the right at once, drawing a diagonal line segment.

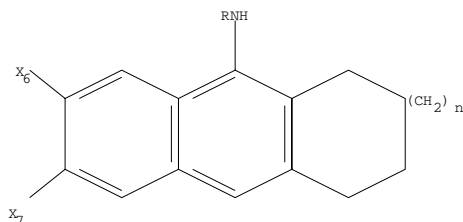


Fig. 2. Template of the tacrine molecule.

Table 1. Accuracy in the Alzheimer's disease domain.

Target	1BC	1BC2	Best rule inducer
Inhibit amine reuptake	67.7%	75.3%	86.1%
Low toxicity	74.4%	73.8%	81.9%
High acetyl cholinesterase inhibition	69.1%	68.7%	75.5%
Reversal of scopolamine-induced memory deficiency	62.2%	76.6%	61.0%

that can be constructed by setting different thresholds. Notice that the resulting curve is monotonic but not necessarily convex, especially not when it is constructed from a test set or in cross-validation.

4.1 Alzheimer's Disease

This dataset is about drugs against Alzheimer's disease [1]. It aims at comparing four desirable properties of such drugs:

- inhibit amine reuptake
- low toxicity
- high acetyl cholinesterase inhibition
- good reversal of scopolamine-induced memory deficiency

The aim is not to predict whether a molecule is good or bad, but rather whether the molecule is better or worse than another molecule for each of the four properties above. All molecules considered in this dataset have the structure of the tacrine molecule and they differ only by some substitutions on the ring structures (Figure 2).

New compounds are created by substituting chemicals for R , X_6 and X_7 . The X substitution is represented by a substituent and its position. The R substitution consists of one or more alkyl groups linked to one or more benzene rings. The linkage can either be direct, or through N or O atoms, or through a CH bond. The R substitution is represented by its number of alkyl groups, a set of pairs of position and substituent, a number of ring substitutions, and a set of pairs of ring position and substituent.

Table 1 compares the accuracies of 1BC and 1BC2 on each of the four desirable properties. All experiments in this domain are carried out using a 10-fold cross validation. The last column shows the best accuracies reported by Boström and Asker using up-to-date rule inducers [1]. It should be noted that Boström and Asker applied several rule

Table 2. AUC in the Alzheimer's disease domain.

Target	1BC	1BC2
Inhibit amine reuptake	0.785	0.797
Low toxicity	0.816	0.805
High acetyl cholinesterase inhibition	0.780	0.773
Reversal of scopolamine-induced memory deficiency	0.680	0.649

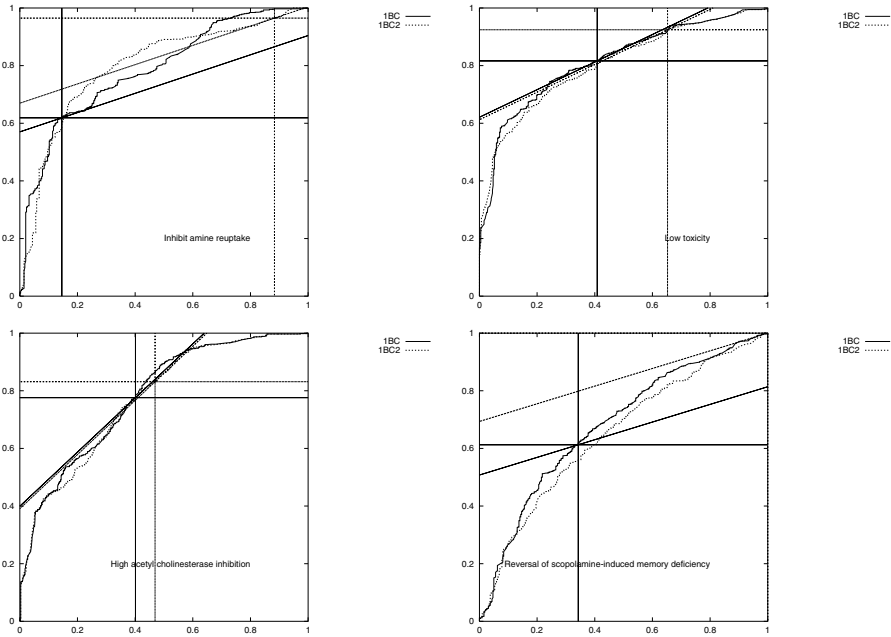


Fig. 3. ROC curves in the Alzheimer's disease domain for 1BC and 1BC2. The crosshairs denote the point chosen by the default probability threshold of 0.5, and the diagonal lines indicate iso-accuracy lines at those points (higher lines are better).

inducers, and that none of them got the best accuracy on all four target. Therefore those accuracies are surely an overestimate of the best overall rule inducer.

Table 2 compares the AUC values of 1BC and 1BC2 on the same four datasets. We see that the differences are very small. Figure 3 shows the corresponding ROC curves. Also indicated are: the points with probability threshold 0.5 corresponding to the accuracy results in Table 1 (the crosshairs), and the iso-accuracy lines through these points (the diagonal lines). From these ROC curves some interesting conclusions can be drawn. In the case of the second and third targets, the selected points on the curve by both classifiers are near-optimal – in fact, the curves have a whole flat segment which is near-optimal. For the first target, while 1BC2 doesn't select the best point, it does a much better job than 1BC, whose choice is clearly sub-optimal. Finally, for the fourth target 1BC2 again manages to choose the optimal point, which happens to be the majority class classifier – i.e., 1BC and all rule inducers performed worse than default!

Table 3. Accuracy in the mutagenesis domain.

Settings	1BC	1BC2	Progol	Regression
lumo and logp only	71.3%	71.3%		85%
lumo, logp, inda and ind 1 only	83.0%	83.0%		89%
Atoms and bonds only	80.3%	72.9%		
Plus lumo and logp	82.4%	72.9%	88%	
Plus inda and ind1	87.2%	72.9%	88%	

Table 4. AUC in the mutagenesis domain.

Target	1BC	1BC2
lumo and logp only	0.761	0.757
lumo, logp, inda and ind 1 only	0.895	0.895
Atoms and bonds only	0.857	0.861
Plus lumo and logp	0.862	0.862
Plus inda and ind1	0.904	0.863

Overall, the experiments on the Alzheimer’s disease dataset suggest that 1BC2 is faster than 1BC (30 seconds compared to 1 minute), that the two produce fairly similar curves, and that 1BC2 does a much better job at determining a near-optimal point than 1BC. 1BC2 consistently does better on the positives and worse on the negatives than 1BC.

4.2 Mutagenesis

This problem concerns identifying mutagenic compounds [10,8]. We considered the “regression friendly” dataset. 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. The latter four properties are propositional.

Table 3 reports the accuracy of different classifiers. The first two settings are propositional. Therefore, 1BC and 1BC2 get the same accuracy and almost identical ROC curves (Figure 4). In the relational settings, 1BC’s is able to take advantage of the added propositional properties, while 1BC2 accuracy remains constant and significantly lower. This suggests that the trade-off between propositional and relational features chosen by 1BC2 leaves room for improvement. 1BC2 again chooses points on the curve which classify more instances as positive than 1BC, but on this domain this leads to worse accuracy results. The curves and AUC values are again fairly similar. It should be noted that 1BC runs the 10-fold cross validation in 5 minutes for relational settings while 1BC2 requires only 5 seconds!

4.3 Diterpene Structure Elucidation

The last dataset is concerned with Diterpenes, which are one of a few fundamental classes of natural products with about 5000 members known [2]. Structure elucidation

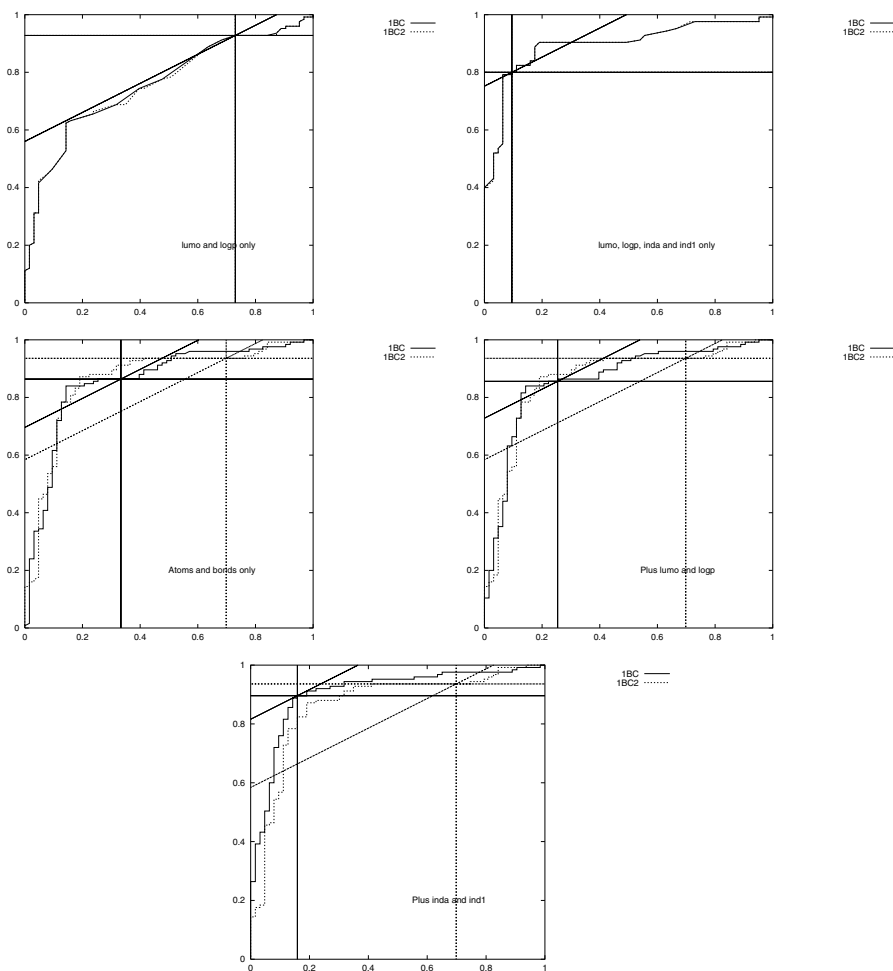


Fig. 4. ROC curves in the mutagenesis domain.

of diterpenes from 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 dataset is concerned with the second stage, the identification of the skeleton. A skeleton is a unique connection of 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.

Table 5. Results in the diterpenes domain.

Settings	1BC	1BC2	C4.5	FOIL	RIBL
Propositional	77.6%	77.6%	78.5%	70.1%	79.0%
Relational	63.7%	55.1%		46.5%	86.5%
Propositional and relational	71.3%	66.5%		78.3%	91.2%

The data contain 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 [2], the accuracy is evaluated with a 3-fold cross-validation, and three settings were considered: each representation separately, and then together. We did not perform a ROC analysis on this domain because it has more than two classes (23 classes).

In this domain, 1BC and 1BC2 get a better accuracy than FOIL in the first two settings, but a worse accuracy than RIBL. In particular, they do not benefit from the combination of propositional and relational data as well as RIBL. At least they perform comparably to C4.5 on the propositional representation. This indicates a minimal property of a first-order learner, namely to perform as well as a propositional learner on propositional data. Comparing 1BC and 1BC2, while 1BC2's accuracy is slightly lower than 1BC's, 1BC2's run time is 20 minutes and 1BC's is 6 hours.

5 Concluding Remarks

Our previous system 1BC is a first-order naive Bayesian classifier that generates a set of first-order features that are then used dynamically as attributes of a propositional naive Bayesian classifier [3]. 1BC2 is a first-order naive Bayesian classifier too. However, it is much more difficult to conceive an equivalent algorithm consisting of a propositionalisation, followed by some propositional naive Bayesian classifier. For this reason, it is a true first-order Bayesian classifier.

In order to properly upgrade the attribute-value naive Bayes assumption to first-order objects, we first defined probability distributions over lists, multisets and sets that allow us to estimate the probability of a list (resp. a multiset and a set) of elements given the probability of those elements. The principle of 1BC2 to classify an structured individual is then to consider what its top-level structure is (tuple, list, set or multiset of elements) and then to estimate the probability of this structure from the probability of its elements. If an element is a structured object itself, its probability is estimated using the same principle. The probabilities of non-structured objects are estimated using Laplace estimate from countings done during the learning phase.

We evaluated the performance of 1BC2 on three datasets. On the first two datasets, a ROC analysis showed that 1BC and 1BC2 obtain comparable ROC curves, but that the default 0.5 probability thresholds lead 1BC2 to classify more instances as positive than 1BC. While on Alzheimer's disease this led to comparable or significantly better accuracy, on mutagenesis this situation was reversed. On the third dataset 1BC2 obtained considerably worse accuracy than 1BC when relational features were involved. This

suggests that the trade-off between propositional and relational features chosen by 1BC2 leaves room for improvement, something we intend to investigate in future work.

The main strength of 1BC2 lies in its simplicity: since 1BC2 does not consider complex hypotheses, its learning time is considerably reduced. In this respect, 1BC2 is clearly quicker than 1BC: while 1BC has to generate first-order features at learning time, and then to match them at classification time, 1BC2 just go through the structure of each individual, to count at learning time, and to estimate probabilities according to appropriate distributions at classification time. Their run times on structured data differ roughly of one order of magnitude: minutes vs. seconds in mutagenesis, hours vs. minutes for diterpenes.

We have also found in our experiments that 1BC2 behaved identically, regardless of whether the list or the subset distribution was chosen. At present we do not know whether this was a coincidence on the domains considered, or whether they are genuinely equivalent when used in a naive Bayesian classifier. This will be investigated in future work. Further perspectives for future work include a non-exponential formula for the subset distribution, further investigations on estimates of non-structured and structured objects, in particular to get the best accuracy of 1BC and 1BC2 at the speed of 1BC2, and finally the use of those new estimators in other applications, for instance Support Vector Machines [5].

Acknowledgements

Thanks are due to Henrik Boström and Sašo Džeroski for providing us with the Alzheimer and Diterpene datasets, respectively. Part of this work is supported by the IST project IST-1999-11495 *Data Mining and Decision Support for Business Competitiveness: Solomon Virtual Enterprise*.

References

1. H. Boström and L. Asker. Combining divide-and-conquer and separate-and-conquer for efficient and effective rule induction. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 33–43. Springer-Verlag, 1999.
2. Sasō Džeroski, Steffen Schulze-Kremer, Karsten R. Heidtke, Karsten Siems, Dietrich Wettschereck, and Hendrik Blockeel. Diterpene structure elucidation from ^{13}C nmr spectra with inductive logic programming. *Applied Artificial Intelligence*, 12(5):363–383, July–August 1998. Special Issue on First-Order Knowledge Discovery in Databases.
3. P. Flach and N. Lachiche. 1BC: A first-order Bayesian classifier. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 92–103. Springer-Verlag, 1999.
4. Jerome H. Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, March 1997.
5. Thomas Gärtner and Peter Flach. A linear kernel on strongly typed terms. In *Multi-Relational Data Mining workshop, Joint workshop of ECML'01 and PKDD'01*, 2001.
6. Nicolas Lachiche and Peter A. Flach. A first-order representation for knowledge discovery and bayesian classification on relational data. In Pavel Brazdil and Alipio Jorge, editors, *Data*

Mining, decision Support, Meta-learning and ILP : Forum for Practical Problem Presentation and Prospective Solutions (DDMI-2000), Workshop of 4th International Conference on Principles of Data Mining and Knowledge Discovery (PKDD-2000), pages 49–60, Lyon, September 2000.

7. N. Lavrač, S. Džeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 265–281. Springer-Verlag, 1991.
8. S. Muggleton, A. Srinivasan, R. King, and M. Sternberg. Biochemical knowledge discovery using Inductive Logic Programming. In H. Motoda, editor, *Proceedings of the first Conference on Discovery Science*, Berlin, 1998. Springer-Verlag.
9. U. Pompe and I. Kononenko. Naive Bayesian classifier within ILP-R. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 417–436. Department of Computer Science, Katholieke Universiteit Leuven, 1995.
10. A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237 of *GMD-Studien*, pages 217–232. Gesellschaft für Mathematik und Datenverarbeitung MBH, 1994.