Decomposing probability distributions on structured individuals

Peter A. Flach[1] and Nicolas Lachiche[2]

[1] Department of Computer Science, University of Bristol, United Kingdom
Peter.Flach@bristol.ac.uk
[2] LSIIT - IUT de Strasbourg Sud, France
lachiche@lsiit.u-strasbg.fr

Abstract. Naive Bayesian classifiers have been very successful in attribute-value representations. However, it is not clear how the decomposition of the probability distributions on attribute-value tuples underlying those classifiers can be applied in the case of structured individuals, for instance sets of tuples as in the multiple instance problem. This paper presents a decomposition of probability distributions on structured individuals. Several probability distributions over lists, sets, and multisets are considered. In particular, an alternative to the bitvector distribution over sets is introduced. Decomposition as proposed in this paper introduces features (properties of the individual) that can be used, as in `1BC`, to estimate the most likely class of an individual given its description.

1 Introduction

In its general form, a Bayesian classifier predicts the most likely class value c of an object given its description d: $argmax_c P(c|d)$. Applying Bayes theorem, this formula can be turned into: $argmax_c \frac{P(d|c)P(c)}{P(d)} = argmax_c P(d|c)P(c)$. 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 where the naive Bayes assumption comes in.

In an attribute-value representation, the object is described by its values $a_1, \ldots, a_n$ for a fixed set of attributes $A_1, \ldots, A_n$. Since it is difficult to get a good estimate of $P(a_1, \ldots, a_n|c)$, the naive Bayes assumption consists in decomposing this probability distribution into the product of the probability of each attribute value: $P(a_1|c) \times \ldots \times P(a_n|c)$, 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 $a_1, \ldots, a_n$ 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 [6]. 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 [5]). In this paper we study an extension of the `1BC`-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 details the decomposition process. It also shows how it can be seen as a propositionalisation of the data guided by the decomposition of the structure of the individual and how existing systems such as `1BC` can be used. Section 4 concludes.

2 Probability distributions over lists and sets

In this section we assume a finite *alphabet* $A = \{x_1, \ldots, x_n\}$ 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:*

$$P_l([x_{j_1},\ldots,x_{j_l}]) = \tau(1-\tau)^l \prod_{i=1}^{l} P_A(x_{j_i})$$

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

Introducing an extended alphabet $A' = \{\varepsilon, x_1, \ldots, x_n\}$ and a renormalised distribution $P_{A'}(\varepsilon) = \tau$ and $P_{A'}(x_i) = (1-\tau)P_A(x_i)$, we have $P_l([x_{j_1},\ldots,x_{j_l}]) = P_{A'}(\varepsilon)\prod_{i=1}^{l} P_{A'}(x_{j_i})$. That is, under P_l 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 $\tau = (1-.2)(1-.3)(1-.5) = .28$ (see Example 4), we have $P_{A'}(a) = (1-.28)*.2 = .14$, $P_{A'}(b) = .22$, and $P_{A'}(c) = .36$, and $P_l([a]) = .28*.14 = .04$, $P_l([b]) = .06$, $P_l([c]) = .10$, $P_l([a,b]) = .28*.14*.22 = .009$, $P_l([a,c]) = .014$, $P_l([b,c]) = .022$, and $P_l([a,b,c]) = .28*.14*.22*.36 = .003$. We also have, e.g., $P_l([a,b,b,c]) = .28*.14*.22*.22*.36 = .0007$.

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 [2]. 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_{l} 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:*

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

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 $P_{\text{ms}}(\{[a]\}) = .04$, $P_{\text{ms}}(\{[b]\}) = .06$, $P_{\text{ms}}(\{[c]\}) = .10$ as before. However, $P_{\text{ms}}(\{[a,b]\}) = .02$, $P_{\text{ms}}(\{[a,c]\}) = .03$, $P_{\text{ms}}(\{[b,c]\}) = .04$, $P_{\text{ms}}(\{[a,b,c]\}) = .02$, and $P_{\text{ms}}(\{[a,b,b,c]\}) = .008$.

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_{l} 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.

Proposition 1 (Subset-distribution over sets). *Let S be a non-empty subset of l elements from the alphabet, and define*

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

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

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

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. The following distribution, on the other hand, defines the probability of a set in terms of both its members and its non-members.

Definition 3 (Bitvector-distribution over sets).

$$P_{\mathrm{bv}}(S) = \prod_{x \in S} P_A(x) \prod_{y \notin S} (1 - P_A(x))$$

That is, a set is viewed as a bitvector over the alphabet, each bit being independent of the others.

Example 4 (Bitvector-distribution over sets). Continuing Example 3, we have $P_{\mathrm{bv}}(\emptyset) = P(\{c\}) = .8 * .7 * .5 = .28$, $P_{\mathrm{bv}}(\{a\}) = P(\{a,c\}) = .07$, $P_{\mathrm{bv}}(\{b\}) = P(\{b,c\}) = .12$, and $P_{\mathrm{bv}}(\{a,b\}) = P(\{a,b,c\}) = .03$. Notice how, at the expense of $\{a,b,c\}$, the singleton subsets obtain higher probabilities, in particular $\{c\}$ (c being the most frequent element of the alphabet).

P_{bv} doesn't take the probability of the empty list or set as a parameter. We have therefore used $\tau = P_{\mathrm{bv}}(\emptyset)$ in the preceding examples. However, in general the bitvector-distribution requires a distribution over each bit, rather than a distribution over the alphabet. One can think of the former as being obtained from the latter through a parameter σ, so that the distribution associated with element x_i is $(\sigma P_A(x_i), 1 - \sigma P_A(x_i))$. The last example was constructed so that $\sigma = 1$. To differentiate between the distribution P_A and the distribution over a particular bit, we will sometimes use the notation $P_{x_i}(+)$ and $P_{x_i}(-)$ for the latter.

The two distributions over sets, P_{ss} and P_{bv}, differ in the independence assumptions they make. Which one is better thus depends on the domain. Notice that P_{ss} is expensive to compute for large sets, while P_{bv} is expensive for large universes.

3 Probability distributions over structured objects

In this section, we will discuss more generally the problem of probability distributions over structured objects, and how these may be broken down into simpler distributions by making certain independence assumptions. We will then show how the `1BC` approach fits into the picture.

Two perspectives will be helpful in our analysis: the term perspective and the database perspective. In the term perspective, our structured objects are viewed as terms in a typed programming language [4]; thus, we can say that an individual is described by a set of tuples (more precisely: by a term which is an instance of such a type). In the database perspective, a dataset is described by a database of relations with foreign keys, and an individual is described by the sub-database of tuples referring to that individual or parts thereof. The process of converting a term representation into a database representation is called *flattening*, and is described in more detail elsewhere. Here, it suffices to point out that a tuple term (i.e. with a type constructed by cartesian product) will correspond to a single row in a table, while set and list terms will correspond to 0 or more rows.

3.1 Decomposition of tuples: propositional naive Bayes

Approaching the propositional naive Bayes classifier from the database perspective, we see that it amounts to breaking up a table with $n+1$ columns into n binary tables (Figure 1). This is of course, in general, a transformation which results in loss of information: we lose the associations between attributes that exist in the dataset. The naive Bayes assumption is made to reduce computational cost and to make the probability estimates more reliable, but we cannot achieve this without loss of logical and statistical knowledge.

We thus see that in propositional naive Bayes, each n-tuple describing a single instance gets decomposed into n 1-tuples (we ignore the class value here, since it will obviously be included in any decomposed table). We can generalise this by allowing nested tuples: e.g., in the King-Rook-King problem a board can be described as a 3-tuple of positions of each of the pieces, where a position is a pair (file,rank). We now have the choice between a level-1 decomposition, in which we estimate probabilities of

Colour	Shape	Number	Class
blue	circle	2	$\oplus$
red	triangle	2	$\oplus$
blue	triangle	3	$\ominus$

Colour	Class
blue	$\oplus$
red	$\oplus$
blue	$\ominus$

Shape	Class
circle	$\oplus$
triangle	$\oplus$
triangle	$\ominus$

Number	Class
2	$\oplus$
2	$\oplus$
3	$\ominus$

Fig. 1. Propositional naive Bayes from the database perspective. In this simple example, each instance is described by a 3-tuple (Colour,Shape,Number). Following the naive Bayes assumption, the dataset in the left-hand table gets decomposed into the three tables on the right. Conditional probabilities of a particular combination of attribute values occurring given the class are estimated from the smaller relations. For instance, we have $P(\text{blue}|\oplus) = .5$, $P(\text{circle}|\oplus) = .5$, and $P(2|\oplus) = 1$, and hence $P((\text{blue}, \text{circle}, 2)|\oplus)$ is estimated at .25, instead of .5 which would be estimated from the left table.

positions (i.e. a 3-tuple of pairs gets decomposed into 3 pairs), and a level-2 decomposition in which we estimate probabilities of files and ranks (i.e. a board is now described by 6 1-tuples).

3.2 Decomposition of sets: the multiple instance problem

Such nesting of terms also occurs in the multiple instance problem [1], where each example is described by a variable number of attribute-value vectors. From the type perspective, this corresponds to a *set of tuples* (Figure 2).

Example	Colour	Shape	Number
e1	blue	circle	2
e1	red	triangle	2
e1	blue	triangle	3
e2	green	rectangle	3
e3	blue	circle	2
e3	green	rectangle	3

Example	Class
e1	$\oplus$
e2	$\ominus$
e3	$\oplus$

Fig. 2. A multiple instance problem. Here, the type of an individual is $\{$(Colour,Shape,Number)$\}$, i.e. a set of 3-tuples. Classes are assigned to sets rather than tuples, requiring a separate table.

How do we estimate $P(\text{e1}|\oplus)$? A simple approach would be to just use the table on the right. However, this does not generalise, as we would have no means to evaluate the probability of a set we haven't seen before. Also, the estimates would be extremely poor unless we see the same sets many times. We thus have to make use of the internal structure of the set, i.e., which tuples it contains. For instance, we can represent the set as a bitvector, treating each bit as statistically independent of the others (Figure 3).

Example	HasBlueCircle2	HasRedTriangle2	HasBlueTriangle3	HasGreenRectangle3	Class
e1	+	+	+	-	$\oplus$
e2	-	-	-	+	$\ominus$
e3	+	-	-	+	$\oplus$

Fig. 3. Representation of a set as a bitvector. Each attribute or *feature* corresponds to a particular tuple occurring in the set – e.g., HasBlueCircle2 corresponds to the tuple (blue, circle, 2). This table can now be decomposed into 4 smaller tables in the standard way. $P(\text{e1}|\oplus)$ would then be estimated as $P(\text{HasBlueCircle2} = +|\oplus)$ $P(\text{HasRedTriangle2} = +|\oplus)$ $P(\text{HasBlueTriangle3} = +|\oplus)$ $P(\text{HasGreenRectangle3} = -|\oplus) = .125$.

Alternatively, we can decompose the tuples first, turning a set of tuples into a tuple of sets (Figure 4). Subsequently, we can estimate probabilities of sets by treating them as bitvectors (Figure 5).

Example	Colour
e1	blue
e1	red
e1	blue
e2	green
e3	blue
e3	green

Example	Shape
e1	circle
e1	triangle
e1	triangle
e2	rectangle
e3	circle
e3	rectangle

Example	Number
e1	2
e1	2
e1	3
e2	3
e3	2
e3	3

Fig. 4. Decomposition of a set of tuples into a tuple of (multi)sets. The class assignment to sets is the same as in Figure 2.

In summary, in order to apply Bayesian classification to a multiple instance problem we can perform the following decompositions:

level-0: not to decompose at all, i.e. only count number of occurrences of a particular set (right table of Figure 2);

level-1: to treat the set as a bitvector and decompose the bitvector but not the tuples (Figure 3);

level-2: to decompose the set of tuples into a tuple of sets (Figure 4), and to further decompose the sets thus obtained into bitvectors with independent components (Figure 5).

Example	HasBlue	HasRed	HasGreen	HasCircle	HasTriangle	HasRectangle	Has2	Has3	Class
e1	+	+	-	+	+	-	+	+	$\oplus$
e2	-	-	+	-	-	+	-	+	$\ominus$
e3	+	-	+	+	-	+	+	+	$\oplus$

Fig. 5. Decomposition of the sets in Figure 4 into bitvectors. Effectively, each set is now described by an 8-tuple. Assuming the components of this bitvector are independent, $P(\mathrm{e1}|\oplus)$ is now estimated as $1 * .5 * .5 * 1 * .5 * .5 * 1 * 1 = .063$.

It should be noted that, while we use the bitvector representation of sets here, there is no reason why we couldn't use another probability distribution from Section 2, in particular the multiset distribution P_{ms} and the subset distribution P_{ss}. In this case, we only use the features that are true to estimate the probability of a particular (multi)set. The main point about the foregoing analysis is that it demonstrates the kind of features that enter the picture as soon as one starts decomposing naive-Bayes style. It also demonstrates that, in general, one has a choice with regard to the depth of the decomposition. Although the preceding analysis only considered the multiple instance problem, i.e. a set of tuples, a similar analysis can be applied to hierarchies of arbitrary depth.

3.3 A practical perspective

Each of the above decompositions represents some kind of transformation, where the dataset is completely described in terms of properties of individuals. We call those properties *features*.

In level-0 decomposition, the features are simply of the form `isSet(S,e1)`; in order to estimate $P(\mathrm{e1}|\oplus)$ we determine the proportion of positive examples satisfying this feature.

In level-1 decomposition features such as the following are used:

```
hasBlueCircle2(S):-set2tuple(S,T),isTuple(T,blue,circle,2).
hasRedTriangle2(S):-set2tuple(S,T),isTuple(T,red,triangle,2).
```

Here, `set2tuple(S,T)` non-deterministically extracts a tuple `T` from the set `S`, and `isTuple(T,blue,circle,2)` checks whether the tuple is the required one.

In level-2 decomposition features of the following kind are used:

```
hasBlue(S):-set2tuple(S,T),hasColour(T,blue).
hasCircle(S):-set2tuple(S,T),hasShape(T,circle).
has2(S):-set2tuple(S,T),hasNumber(T,2).
```

A first-order Bayesian classifier like 1BC [3] can generate and use such features. Therefore it can perform a decomposition of probability distribution over structured individuals.

4 Discussion

Bayesian classifiers require estimating the probability distribution of the description of an individual. It is usually impossible to get a good estimate of such a probability due to the large number of possible descriptions and the comparatively small number of examples available. A naive Bayesian classifier relies then on decomposing the probability distribution of a structured individual in terms of the probability distributions of its components. While the decomposition of a single tuple was well known, we have proposed a decomposition of lists, of sets and of multisets based on a bitvector representation or not, in particular we introduced a new probability distribution over sets. Furthermore, if the resulting components are structured terms themselves, the decomposition can then be iterated until reliable estimates are available. From a practical perspective, existing first-order Bayesian classifiers can be used to perform an iterative decomposition of probability distributions over structured individuals.

We believe the decomposition process we proposed defines a proper naive – i.e., the decomposition assumes that the components of a term are independent given the class – Bayesian classifier on structured individuals. The process is similar to propositionalisation, as at each decomposition level the data is described in terms of properties of the individual (features). At the level-0 decomposition, individuals are considered as a whole, i.e. they are represented by a single attribute taking on an individual value. For instance, example e347 is a `e1` set, i.e. a set containing exactly the tuples BlueCircle2, RedTriangle2 and BlueTriangle3. At the next level of decomposition, individuals are represented by the values their components take on. For instance, the attributes HasBlueCircle2, HasRedTriangle2 and HasBlueTriangle3 take on the positive value for set `e1`, and the attribute HasGreenRectangle3 takes on the negative value. While at each level individuals are described by their features only, the process is not the same as propositionalisation. For instance, if we use P_{ss} to estimate the probabilities of sets, we are only using the pluses in the bitvectors, not the minuses. Also, a propositional dataset is never actually constructed. Hypotheses and individuals are represented in a proper first-order language. The first-order part of the algorithm consists mainly in the decomposition process and being able to query the data in a first-order representation.

In its current implementation, `1BC` considers that the individual is represented by all first-order features that can be generated according to the language declaration provided by the user. So the user has to decide which features have to be considered. It could be automated by doing an iterative generation of features at level-0, level-1, and so on. Levels for which there are no properties would not be considered. For instance if property isSet is not provided at level-0, `1BC` will automatically consider level-1. The decomposition would be stopped as soon as reliable estimates exist. It can be checked by checking that features are satisfied by more than one example, or more generally by a cross-validation.

Acknowledgements

The comments of an anonymous reviewer have been helpful. A previous version of this paper appeared in the proceedings of the ECML2000 workshop on *Dealing with structured data in Machine Learning and Statistics*. Part of this work is supported by the Esprit V project IST-1999-11495 *Data Mining and Decision Support for Business Competitiveness: Solomon Virtual Enterprise*.

References

1. T. G Dietterich, R. H. Lathrop, and T. Lozano-Perez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89:31–71, 1997.
2. R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis – Probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.
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. P.A. Flach, C. Giraud-Carrier, and J.W. Lloyd. Strongly typed inductive concept learning. In D. Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of *Lecture Notes in Artificial Intelligence*, pages 185–194. Springer-Verlag, 1998.
5. 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.
6. 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.