

1BC and 1BC2

Nicolas Lachiche and Peter Flach

February 2004

1 Introduction and installation

1BC and 1BC2 are naive Bayesian classifiers in a first-order logic representation.

They are freely available for academic research and teaching. They can be obtained from the following web page:

<http://www.cs.bris.ac.uk/Research/MachineLearning/1BC/>

1BC and 1BC2 are provided in a single executable. Once you have created a directory containing all the source files and examples, you only have to type `make 1BC` in that directory to generate the executable. The `-r` option is used to choose between 1BC and 1BC2 :

`-r LANGUAGE` calls 1BC (actually the new efficient recursive implementation of 1BC)

`-r INDIVIDUAL` calls 1BC2

Without any `-r` option, the old implementation of 1BC , based on dynamic propositionalisation, is used.

2 Overview

1BC and 1BC2 works from the command line. They are called from the command line in a shell. Their required arguments are the maximum numbers of literals `l` and variables `v`, and a file stem `stem`. The simplest call to 1BC (resp. 1BC2) is of the form `1BC -r LANGUAGE l/v stem` (resp. `1BC -r INDIVIDUAL l/v stem`). For instance, `1BC -r LANGUAGE class 3/3 unfriendly` outputs the accuracy on the test set `unfriendly.test`:

Accuracy: 38/42 = 0.904762

Accuracy: 38/42 = 0.904762

3 Files

1BC and 1BC2 use mainly two input files and produce an output file. All file names are built on the stem provided on the command line.

3.1 Language definition

The `stem.prd` file defines the hypotheses language. It contains the definition of all the predicates. Here is for instance the `unfriendly.prd` file:

```
--INDIVIDUAL
mol 1 mol cwa
atom 1 atom cwa
bond 1 bond cwa
--STRUCTURAL
atm 2 1:mol *:atom 1 cwa li
bond2atom1 2 *:bond 1:atom 1 cwa li
bond2atom2 2 *:bond 1:atom 1 cwa li
--PROPERTIES
class 2 mol #class cwa
ind1 2 mol #num1 1 cwa
inda 2 mol #numa 1 cwa
lumo 2 mol #lumo 1 cwa
logp 2 mol #logp 1 cwa
atomel 2 atom #letter 1 cwa
atomty 2 atom #number 1 cwa
atomch 2 atom #charge 1 cwa
bondtype 2 bond #type 1 cwa
```

One line corresponds to one predicate. Each predicate is defined by its name, its arity, the type of its arguments, and finally whether the closed world assumption (`cwa`) is used for that predicate. If the closed world assumption isn't used, negative ground facts on that predicate must be given explicitly in the `stem.fct` file, and the `cwa` tag must be replaced by `explicit` at the end of the definition of the predicate.

Notice that predicate arguments are typed. All constants of a type used in the examples define its domain. Continuous domains can optionally be discretised.

`1BC` works only on domains containing a clear notion of individuals.

An *individual* can be, for instance, a molecule. The individual can be related to other kinds of objects. All their definitions are separated from the other predicate definitions by a label `--INDIVIDUAL`. Some predicates are only used to refer to relate objects (and introduce new variables) whereas some predicates are properties of the objects (and consume variables). The former predicates are called *structural predicates*, the latter *properties*, and are separated by the appropriate labels. A distribution (`bv,li,ss`) must be specified for each structural predicate.

The number preceding the `cwa` tag refers to the number of repetitions of the predicate in the hypothesis. `*` stands for unlimited (actually the maximum number of literals allowed at `1BC` call). It is an optional field in the definition of a predicate, its default value is 1. A value of zero means that this predicate won't be used in the hypothesis, moreover its ground facts will be skipped in

the `stem.fct` file. Please note that all predicates corresponding to ground facts in the `stem.fct` file must be defined in the `stem.prd` file, so setting the number of repetitions to zero is a way of changing the language bias without having to change the example file.

The `#` in front of a type means that this argument of the predicate is actually a parameter, i.e. it is always instantiated. The values of this argument in the `stem.fct` file define its domain. Please note that the values of parameters are shared between partitions, roughly among individuals.

3.2 Data

The `stem.fct` file contains the data. It can only contain ground facts, the use of other data is explained later in this document.

Each line corresponds to a ground fact. Here is, for instance, the beginning of the `unfriendly.fct` file:

```
!  
class(d190,mutagenic).  
ind1(d190,0.0).  
inda(d190,0.0).  
lumo(d190,-0.7980000000000004).  
logp(d190,2.1299999999999999).  
act(d190,0.5699999999999995).  
atm(d190,d190_1).  
atm(d190,d190_10).  
...  
atomel(d190_10,n).  
atomty(d190_10,38).  
atomch(d190_10,0.7930000000000004).  
...  
bondtype(d190_1d190_14,1).  
bond2atom1(d190_1d190_14,d190_1).  
bond2atom2(d190_1d190_14,d190_14).  
...
```

In this domain, all predicates make use of the closed world assumption. However, if the `explicit` tag is used for some predicate, negative ground facts must be preceded by `-` (and exactly one space before the predicate name).

The `%!` label is used to distinguish subsets of the data, called partitions. Indeed, it doesn't make sense in some domains to combine constants from several "examples". For instance, it doesn't make sense to combine atoms of molecule `d190` with those of train `d191`. Note that however constants corresponding to parameters are shared between partitions.

3.3 Other files

Results are stored into the file `stem.res`. More information is stored in the `stem.sta` file if the verbose option `-v` is used.

A test file `stem.test` must be provided, unless a cross-validation is performed.

4 Commands

A call to `1BC` (resp. `1BC2`) is of the form: `1BC -r LANGUAGE [options] Number of literals/Number of variables stem (resp., 1BC -r LANGUAGE [options] Number of literals/Number of variables stem)` where the `stem` of files describing a domain, and the maximum number of literals and the maximum number of variables contained in a result are given. Several options can be used on the command line.

A threshold can be used to select only the most discriminative features `-t threshold`. If the absolute difference between the observed and the expected “probabilities” of a given class and the feature is greater than the threshold for some class, then the feature is considered by `1BC`. The default threshold is 0.

The maximum of RAM that `1BC /1BC2` uses can be changed with the option `-m max_memory`. The default is `-m 10` corresponding roughly to 10 Mb.

The discretisation of continuous domain is done with the `-d type n sdm—eqb` option. With `sdm` (standard deviation centered on the mean) discretisation, the `type` is discretised into intervals of width equal to the standard deviation and centered on the mean. `n` intervals are defined on each side of the mean (i.e. there are $2n+1$ intervals), the last intervals on each side are extended to the infinity. With `eqb` (equal bins) discretisation, the `type` is discretised into `n` intervals containing as many instances each.

The features considered by `1BC` can be used as bodies of rules defining new predicates for the individuals that can be used by other learners (a kind of predicate invention). With `-w` option, those rules are written to a `.fof` file.

Coordinates of points can be saved in `.roc`, `.roc1`, `.roc2`, and `.roc3` files with `-roc` option in order to build a roc curve, a crosshair corresponding to the current threshold, and its iso-accuracy line.

The roc curve can be used to find the best threshold with `-o num_folds` option. This should be used with a cross-validation only. It performs an internal cross-validation for each training set in order to choose the best threshold. Finally, it finds the best threshold overall and generates the whole roc curve.