Manual
Manual for CN2 version 6.1

Robin Boswell
The Turing Institute Limited

TI/P2154/RAB/4/1.5
January 1990

Task	:	T
Document type	:	Manual
Status	:	Draft
Classification	:	Public
Document class	:	ITM
Int. Doc. ID	:	TI/MLT/4.0T/RAB/1.2
Distribution	:	Universal

Abstract

This document is an introduction and user's manual for release 6.1 of the rule-induction program CN2.

Contents

1 Introduction

The program described in this manual is version 6.1 of the Turing Institute's implementation of CN2. For details of the CN2 algorithm itself, and plans for further work, see [Clark, 1989].

For brevity, I shall use the term "CN2" in the following account to refer both to the algorithm and to its current implementation.

2 Overview

For instructions on installing CN2, see Appendix §A.

CN2 is a rule-induction program which takes a set of *examples* (vectors of attribute-values), and generates a set of *rules* for classifying them. It also allows you to *evaluate* the accuracy of a set of rules (in terms of a set of pre-classified examples).

To use the program, you must provide an ASCII file of examples in a standard format (described in § 3). When the program has induced some rules from these examples, you may display or evaluate them, or store them in a file. Rules are stored in a human-readable form, but they may also be read back in by CN2, to save re-calculating them.

If you have data files in MLT CKRL format, then these can be converted into CN2's format by a CKRL translator (provided by the Turing Institute). In addition, CN2 can itself translate attributes, examples and decision-trees into CKRL format (see the **Ckrl** command in section 5.1.1).

See Appendix §B for an example of a terminal session with CN2.

The interface to CN2 is similar to that for the Turing Institute's other recently-developed program NewID described in [Boswell, 1990], so if you are using both programs, you may find the following comparisons useful:

Input The format of attribute and example files is identical for the two programs. (Declarations of attributes as 'binary' are meaningful only for NewID: such declarations will be silently ignored by CN2.)

Commands The interfaces are similar, but NewID has a smaller menu of commands than CN2, because it has fewer customisable parameters (such as 'significance

threshold').

Output The output formats differ (as do the algorithms): CN2 generates production rules and NewID generates decision trees.

3 Example and Attribute Files

The input to CN2 consists of:

1. A set of attribute declarations.
2. A set of examples.

Items 1. and 2. may be presented in the same file, or in two different files. Storing the attributes separately from the examples makes it possible to read and process several example files in succession without having to re-load the attribute declarations (provided all the examples use the same attributes).

3.1 Files

Input files are of three types:

1. Attribute files.
2. Examples files.
3. Attribute and Example files.

The most complex of these, the attribute and example file, consists of a header, followed by attribute declarations, then examples. The two types of data are separated by punctuation, as specified in § 3.2.3. (See § 3.4 for a specimen of such a file).

Attribute and example files consist of the appropriate header followed respectively by attributes and examples.

In all file types, characters between a '%' and the next end-of-line are regarded as comments and ignored.

Formally:

attribute-and-example-file ::=
    att-and-ex-header attribute-delarations separator examples separator

att-and-ex-header ::=
    "**ATTRIBUTE AND EXAMPLE FILE**"

separator ::=
    "@"

attribute-file ::=
    att-header attributes

att-header ::=
    "**ATTRIBUTE FILE**"

example-file ::=
    ex-header examples

ex-header ::=
    "**EXAMPLE FILE**"

The format for *attributes* and *examples* in the above grammar is defined below.

3.2 Attributes

3.2.1 Semantics

Attributes are of two types: discrete and ordered. A discrete attribute takes one of a finite set of values (and the set of values has no further structure). An ordered attribute takes numeric values: integer or floating point.

An attribute *declaration* consists of the name of the attribute, together with either:

- Its possible values (for a discrete attribute), or
- Its numeric type (for an ordered attribute).

For the purposes of the rule-induction algorithm, one of the attributes is distinguished as the *class attribute*. The aim of the algorithm is to find rules whereby the class

attribute of an example may be inferred from the non-class attributes. The class attribute must be discrete.

3.2.2 Attribute ordering

In some domains, it may be desirable to impose constraints on the order in which attributes are tested. For example, one should determine whether a patient is female before asking whether she is pregnant, rather than afterwards. Such constraints may be imposed on CN2 by means of *attribute ordering declarations*, which (optionally) follow the attribute declarations.

Note that it is not possible to make the *result* of one attribute test a precondition of some other test (for example, "if sex is male, then don't ask about pregnancy") but given sensible data, an inappropriate test will yield a low entropy gain, and so be excluded anyway by CN2's existing criteria.

An attribute ordering declaration takes the form:

$$\text{Attribute}_1 \text{ BEFORE } \text{Attribute}_2;$$

You may include as many such declarations as you wish. Note that CN2 does not currently check for "loops" (A before B before C before A ...): if you introduce such a loop, the result will be that none of the attributes involved will ever be used.

Finally, note that the order in which attribute tests are *displayed* in CN2's output has no significance (as it happens, it matches the order of attribute declaration), so in particular is not affected by attribute ordering constraints. The only effect of, for example, the constraint `sex BEFORE pregnancy` is to ensure that any rule containing a test of `pregnancy` also contains a test of `sex`.

3.2.3 Syntax

An attribute declaration consists of the attribute's name, followed either by its possible values (if it's discrete), or its numeric type (if it's ordered), separated by appropriate punctuation. The class attribute must be declared last. (i.e. CN2 will treat the last attribute as the class attribute).

Formally:

```
attribute-data ::=
        attribute-declarations    att-ordering-section
    |   attribute-declarations

attribute-declarations ::=
        attribute-declaration
    |   attribute-declarations    attribute-declaration

attribute-declaration ::=
        string ":"  attribute-values ";"          % For discrete attributes
    |   string ":"  "(" type ")"                  % For ordered attributes

attribute-values ::=
        string    attribute-values
    |   string

type ::=
        "FLOAT"
    |   "INT"

string ::=
        quoted-string
    |   unquoted-string
att-orderings-section ::=
        "ORDERINGS" attribute-orderings

attribute-orderings ::=
        attribute-ordering
    |   attribute-orderings attribute-ordering

attribute-ordering ::=
        string "BEFORE" string ";"
```

An *unquoted-string* is a sequence of one or more characters made up of upper or lower-case letters, digits, and the characters "–", "+" and "_", with the constraint that the first character must be a letter or "_".

A *quoted-string* is any sequence of printable characters (except newline and the double-quote character) surrounded by double-quotes.

Thus the following are all valid strings:

```
Fred   _42  "42"  "%^=- !"
```

and the following are not valid strings:

```
42   -green  +
```

For example:

```
sex: male female;
job: cleaner secretary manager director hacker;
salary: (FLOAT)
status: content discontented   % The class attribute

ORDERING

sex BEFORE salary;
```

3.3 Examples

3.3.1 Semantics

Each example consists of a set of values, one for each attribute, and may also include a weight.

3.3.1.1 Values

Each value must be of the appropriate type. However, in addition to the numeric and string values given in the attribute declarations, any attribute except the class attribute may take one of the two special values "Unknown" or "Don't Care".

The "Unknown" value is used to represent an unknown value (!), such as frequently occur in real-world data. The "Don't Care" value, on the other hand, is assigned to an attribute whose value is irrelevant to the classification of the example. In practice, "Don't Care" values are most likely to arise as a means of compressing synthetic data.

Thus, an "Unknown" attribute corresponds roughly to an existentially quantified variable, whereas a "Don't Care" attribute is universally quantified.

A brief account of how NewID handles "unknown" and "don't care" values appears in Appendix C

3.3.1.2 Weights

By default, each example is assigned a weight of 1, and this is sufficient for most applications. However, in some cases it may be useful to assign different weights to examples; the weight must be a positive real number. If you use this facility, you should note that references in this document to the "number of examples" in a set (e.g. 4.3.1) really mean "the sum of the weights of the examples".

3.3.2 Syntax

```
examples ::=
        example
    |   examples    example

example ::=
        values ";"
    |   values "w"  number

values ::=
        value
    |   values    value

value ::=
        string
    |   number
    |   "?"                 % Unknown
    |   "*"                 % Don't Care

number ::=
        integer
    |   float
```

In addition, the order of items in *values* must correspond to the order of the attributes as previously declared, so the length of *values* must be equal to the number of attributes, and each $value_n$ must match the type of $attribute_n$.

3.4 Specimen input file

This is a small "Attribute and Example" file:

```
**ATTRIBUTE AND EXAMPLE FILE**
% Vertebrate classification
skin_covering: none hair feathers scales;
milk:          yes no;
homeothermic:  yes no;
habitat:       land sea air;
reproduction:  oviparous viviparous;
breathing:     lungs gills;
class: mammal fish reptile bird amphibian;

@
% mammal
hair           yes     yes     land    viviparous      lungs mammal;
none           yes     yes     sea     viviparous      lungs mammal;
hair           yes     yes     sea     oviparous       lungs mammal;
hair           yes     yes     air     viviparous      lungs mammal;

% fish
scales         no      no       sea    oviparous       gills fish w 4;

% reptile
scales         no      no      land    oviparous       lungs reptile w 3.2;
scales         no      no      sea     oviparous       lungs reptile;

% bird
feathers       no      yes     air     oviparous       lungs bird;
feathers       no      yes     land    oviparous       lungs bird;

% amphibian
none           no      no      land    oviparous       lungs amphibian;
@
```

4 Rules: files and evaluation

4.1 File format

When CN2 writes a rule set to file, it does so in a human-readable ASCII form, but unlike example and attribute files, you are not expected to write or modify rule files yourself. (i.e. you do so at your own risk, and should note that CN2 does very little error-checking when reading rule files). In future releases, a graphical interface may be provided for manipulating rules.

A small rule-set appears as part of the trace on page 22.

4.2 Class distributions

The list of numbers at the end of each rule indicates the number of training examples covered by that rule, divided into classes.

The precise significance of the counts depends on whether the rules are ordered or unordered. Ordered rules are intended to be executed in order (!), so the counts associated with rule N refer to the examples covered[1] by rule N which were *not* covered by any of the rules 1 through $N-1$. This applies also to the default rule—since this rule has no conditions, its counts comprise all the examples which were not covered by the preceding rules.

The counts associated with each of a set of *unordered* rules, however, comprise all the examples covered by that rule including those which may be covered by other rules as well. Again, this applies equally to the default rule, whose count therefore comprises the whole example set.

4.2.1 Rare classes

Given the above definition of "class distribution", you might expect that the class predicted by a rule would always show the largest class-count in the corresponding distribution; so to avoid possible confusion when using the program, you should note that this won't always be the case.

In the case of ordered rules, the class predicted by each rule will be the majority class among the examples covered, *by definition*.

In the case of unordered rules, however, this property will not always hold, though it usually will. The exceptions are rules which give better than average predictions of rarely-occurring classes. For example, in a medical domain, where the program is trying to predict the rarely-occurring phenomenon of diabetes, the following rule might be induced:

```
IF      temperature = high
  AND   fluid-intake = high
THEN    diagnosis = diabetes [420 0 0 126]
```

[1] A rule is said to *cover* an example if the example satisfies the conditions of the rule

where there are 126 examples of diabetes, and 420 of something different (colds, as it happens). Since the occurrence of diabetes in the general population is only 0.1%, a rule that predicts it with an accuracy of 30% is clearly of some use, even if 70% of patients satisfying the conditions just have colds.

4.3 Evaluation

The evaluation module takes a set of rules and a pre-classified set of examples, and compares the classification given by the rules with the given class-values. There are currently two modes of evaluation: *all* and *individual*. (Individual rule evaluation is a recent addition in response to local demand).

4.3.1 "All" rule evaluation

In this mode, the evaluation is of a set of rules taken as a whole. The results are displayed in the form of a matrix:

	PREDICTED					
ACTUAL	mammal	fish	reptil	bird	amphib	Accuracy
mammal	3	0	0	0	0	100 %
fish	0	1	0	0	0	100 %
reptile	0	0	1	0	0	100 %
bird	1	0	1	2	0	50 %
amphibia	0	0	0	0	1	100 %

Overall accuracy: 80 %
Default accuracy: 40 %

The entry in row i, column j of the matrix is the number of examples classified by the rules as class j which were really of class i. (So, in the sample matrix above, there was 1 example which the rules classified as a mammal, but which was really a bird.) Fractional values may arise if the examples include "Unknown" or "Don't Care" values.

The "Default accuracy" is only applicable to unordered rule sets; it is the accuracy resulting from classifying all examples according to the majority class (which is what the default rule does).

4.3.2 "Individual" rule evaluation

This mode of evaluation is currently available only for unordered rule sets, though a similar facility for ordered rule sets may be provided in future.

For each rule, a matrix similar to the above is calculated, but in this case, the class values are reduced to "class predicted by rule" and "other classes". For example:

```
———————— Rule 2 ————————
IF      7.50 < number_of_legs < 25.00
THEN    class = spider [0.25 2 0 0]
```

	FIRED?		
ACTUAL CLASS	Yes	No	Accuracy
spider	2.00	0.00	100.0 %
Not spider	0.25	7.75	96.9 %

Overall accuracy: 97.5 %

4.4 Inducing and testing rules

To create and test a set of rules, you will need to carry out the following operations (the actual commands required are described in the next section).

1. Load some attributes and examples
2. Induce some rules
3. Load some more examples
4. Evaluate the rules

In future releases, I shall probably provide commands for (e.g.) partitioning example sets, and selecting sub-sets, within the program, thus simplifying the above sequence.

5 Using CN

5.1 Interactive use

Note: in specimens of terminal interaction user input appears underlined, and "↙" represents the use of the 'Return' key. For example:

```
READ> Both, Atts, or Examples?  Both
READ> Filename?  Data/animals ↙
```

In the first line, the user typed the single character 'B' (and did *not* hit the return key), and the program expanded the command to 'Both'. In the second line, the user typed the file name 'Data/animals', and then hit 'Return'.

When you start CN2, you will see this friendly greeting:

```
*******************************
*                             *
*         Welcome to CN2!     *
*                             *
*******************************
```

followed by the prompt:

```
CN>
```

This is the *top-level* prompt. If you now type 'h' or '?' (for help), the program will display a list of the commands that may be entered at this point, and then re-display the top-level prompt, ready for another command.

Some operations require several commands to perform. For example, if you want CN2 to read in a file, you have to tell it what sort of thing the file contains, and the name of the file. In this case, your initial 'read' command will cause CN2 to change from top-level mode to file-reading mode, and it will display a different prompt, thus:

```
READ>
```

When the file-reading has been completed, CN2 redisplays its top-level prompt, so the whole sequence might appear as follows:

```
CN> Read
READ> Both, Atts, or Examples?  Both
READ> Filename?  Data/animals ↙
Reading attributes and examples ...
10 examples!
Finished reading attribute and example file!
CN>
```

5.1.1 Top-level commands

Note that this section assumes an understanding of the CN2 algorithm as specified in [Clark, 1989], and you should refer to this document if you are unsure of the meaning of "star size", "significance threshold", "ordered rules", etc.

Each of these commands is invoked by typing its initial letter (upper or lower case), except for "Execute", which is invoked by "x".

Read: Read an attribute, example, attribute-and-example or rule file. When loading several files in succession, you should bear in mind that CN2 can only retain in memory one set of attributes, and one set of examples, at any one time. Consequently:

1. Before loading an example or rule file, you must load the appropriate attributes.
2. Loading a file of any type overwrites any data of that type which you may have loaded previously (even if the load fails—for example, because of a syntax error in the new file).
3. In addition, loading a set of attributes causes any previously-loaded examples to be lost.

The entry of file-names is facilitated by the following features:

1. As in EMACS, "<TAB>" invokes file-name completion. If the prefix you have typed is ambiguous, the program will show you a list of the possible completions.
2. As in EMACS, "<ESC> h" is bound to "delete-word-backwards".
3. Each time you type a '/' as part of a UNIX path-name, the program checks that the directory referred to exists and is readable. If not, it will prevent you from typing further (you must delete back and correct the invalid path).

Induce: Run CN2 on the most recently read set of examples. The behaviour of CN2 can be modified by changing various parameters (see the commands below). However, all these parameters have default values set at start-up, so there is no need to set them before processing data.

Write: Write a set of rules to file. (You must first have read in some examples and run CN2 on them). File-name entry works as for **Read**, above.

Ckrl: Write data in CKRL format. Either the current set of attributes and examples (together) or the current decision-tree may be written. Note that the former option allows CN2 to be used as a translator from CN2 format to CKRL.

eXecute: Enter "evaluation" mode, in which the performance of the current rules may be assessed with respect to the current examples. (See § 4.3). In evaluation mode, valid commands are:

All: Evaluate the set as a whole.

Individual: Evaluate rules individually.

Help/?: Display a menu of commands.

<RET>: Return to the **CN>** prompt.

Algorithm: Specify whether CN2 is to produce ordered or unordered sets of rules.

Error estimate: Specify whether CN2 is to use the Laplacian or the naïve estimate to assess the accuracy of a rule.

The formulae for the two estimates are as follows:

Laplacian:	#Correct+1 / #Examples + #Classes
Naïve:	#Correct / #Examples

Preciesly which example sets these formulae are applied to depends on whether the rule set being induced is ordered or unordered (see the discussion of "counts" in § 4.2).

Star size: Query or alter the star size.

Threshold: Query or alter the significance threshold.

Display: Provide further information about the search. By default, the program just displays each rule as it finds it, but if required it can, for example, indicate whenever a new "best node" is found, or which node is currently being specialised. (Type 'h' at the "SET TRACE FLAGS>" prompt for a list of options.)

This facility can be used to clarify what the algorithm is doing, particularly if it is generating unexpected answers.

Help/?: Display a menu of commands.

Quit: Quit

5.2 Non-interactive use

It is possible to run CN2 non-interactively by supplying it with a sequence of commands in a file. In this case, CN2 will write a trace of its activities to the standard output, so if you want to run it in the background, you should redirect its output either to a log file or to /dev/null. For example:

```
% cn < cn.commands > cn.log &
```

Commands should be listed in the command file more or less as they would be typed in answer to CN2's prompt. In the case of single-character commands, the complete command may be used, to make the file more readable (the program will ignore all but the first character of the word): each such command should be separated from the next item by some form of whitespace, but not necessarily a newline. Filenames must be followed by a newline.

Characters from '#' to the next end-of-line will be regarded as comments and ignored (c.f. shell scripts).

For example:

```
read atts Data/animals.atts
read exs Data/animals.exs
alg unordered
star 7
display none quit
induce
write
Data/animals.rules
quit
```

References

[Boswell, 1990] R.A. Boswell. Manual for NewID version 2.0. Technical Report TI/P2154/RAB/4/, Turing Institute, January 1990.

[Clark, 1989] P.E. Clark. Functional specification of CN and AQ. Technical Report TI/P2154/PC/4/1, Turing Institute, September 1989.

A Getting Started

To read the contents of your release tape, load the tape into your tape-drive, **cd** to the directory where you want the sources and data to be installed, and type:

tar xvf ***tape-device-name***

This should create a directory **Release**, with sub-directories **NewId**, **CN2**, **Simple_Data**, and **Docs**.

Next, **cd** to **Release/CN2**, where you should find the sources for CN2, and a **Makefile**.

If you are using SunOS 3, then you will need to change the definition of "OS" in "mdep.h". Replace the line:

```
#define OS (4)
```

with

```
#define OS (3)
```

Now type:

```
make cn
```

This will cause **cn** to be compiled.

You will find some example and attribute files in the directory **Release/Simple_Data**.

B Example run

This section records a terminal session in which CN2 is applied to a small example file called 'animals'. This file has been included on your release tape, so that you can experiment with it without having to type it in.

As in § 5.1, user input is underlined, and carriage-return denoted by ↙.

Points to Note

- You will notice that in response to the first 'Induce' command, CN2 prints out the resulting rule set twice. This is because, at the default level of tracing, the program prints out each rule as it finds it, and then prints out the complete set of rules when it has finished. In the case of unordered rules, the rules are not accompanied by class distributions when first calculated, since it is more convenient to delay calculation of these distributions until the rule-set is complete.

  § 4.2 discusses the precise meaning of the class distributions.

- Note how raising the significance threshold from 1 to 10 for the second "run" gives smaller but less accurate rules.

```
*********************************
*                               *
*         Welcome to CN2!       *
*                               *
*********************************

CN> Read
READ> Both, Atts, or Examples?  Both
READ> Filename?  Data/animals ↙
Reading attributes and examples ...
10 examples!
Finished reading attribute and example file!
CN> Algorithm
Algorithm is currently set to UN_ORDERED
ALGORITHM> Ordered
CN will produce an ordered rule set
CN> Induce
Running CN on current example set...
Best rule is:
IF    milk = yes
THEN  class = mammal [4 0 0 0 0]
Best rule is:
IF    skin_covering = feathers
THEN  class = bird [0 0 0 2 0]
Best rule is:
IF    skin_covering = scales
  AND breathing = lungs
THEN  class = reptile [0 0 2 0 0]
```

```
Best rule is:
IF    skin_covering = none
THEN  class = amphibian [0 0 0 0 1]
Best rule is:
ELSE (DEFAULT) class = fish [0 1 0 0 0 0]


          *--------------------------------*
          |                                |
          |      ORDERED RULE LIST         |
          |                                |
          *--------------------------------*


IF    milk = yes
THEN  class = mammal [4 0 0 0 0]
ELSE
IF    skin_covering = feathers
THEN  class = bird [0 0 0 2 0]
ELSE
IF    skin_covering = scales
  AND breathing = lungs
THEN  class = reptile [0 0 2 0 0]
ELSE
IF    skin_covering = none
THEN  class = amphibian [0 0 0 0 1]
ELSE
(DEFAULT) class = fish [0 1 0 0 0]
CN> Algorithm
Algorithm is currently set to ORDERED
ALGORITHM> Unordered
CN will produce an unordered rule set
CN> Threshold
Current threshold is 1.00
New threshold:  7 ↙
OK!
CN> Display
SET TRACE FLAGS> None
All tracing is now turned OFF
SET TRACE FLAGS> ↙
CN> Induce
Running CN on current example set...


          *----------------------------------*
          |                                  |
          |      UN-ORDERED RULE LIST        |
          |                                  |
```

```
          *----------------------------------*

IF    milk = yes
THEN  class = mammal [4 0 0 0 0]

IF    skin_covering = scales
THEN  class = fish [0 1 2 0 0]

IF    skin_covering = scales
THEN  class = reptile [0 1 2 0 0]

IF    homeothermic = no
  AND breathing = lungs
THEN  class = amphibian [0 0 2 0 1]

ELSE (DEFAULT) class = mammal [4 1 2 2 1]

CN> Quit
Have a nice day!
```

C Handling of Unknowns and Don't Cares

C.1 Training (Learning) Phase

During rule generation, a similar policy of handling unknowns and don't cares is followed: unknowns are split into fractional examples and dontcares are duplicated.

Strictly speaking, the counts attached to rules when writing the ruleset should be those encountered during rule generation. However, for unordered rules, the counts to attach are generated *after* rule generation in a second pass, following the execution policy of splitting an example with unknown attribute value into *equal* fractions for each value rather than the Laplace-estimated fractions used during rule generation.

C.2 Testing (Execution) Phase

When normally executing unordered CN rules without unknowns, for each rule which fires the class distribution (ie. distribution of training examples among classes) attached to the rule is collected. These are then summed. Thus a training example

satisfying two rules with attached class distributions [8,2] and [0,1] thus has an expected distribution [8,3] which results in C1 being predicted, or C1:C2 = 8/11:3/11 if probabilistic classification is desired. The built-in rule executer follows the first strategy (the example is classed simply C1).

With unordered CN rules, an attribute test whose value is unknown in the training example causes the example to be 'split'. If the attribute has three values, 1/3 of the example is deemed to have passed the test and thus the final class distribution is weighted by 1/3 when collected. A similar rule later will again cause 1/3 of the example to pass the test. A don't care value is always deemed to have passed the attribute test in full (ie. weight 1). The normalisation of the class counts means that an example with a dontcare can only count as a single example during testing, unlike NewID where it may count as representing several examples.

With ordered CN rules a similar policy is followed, except after a rule has fired absorbing, say, 1/3 of the testing example, only the remaining 2/3s are send down the remainder of the rule list. The first rule will cause 1/3 $\times$ class distribution to be collected, but a second similar rule will cause 2/3 $\times$ 1/3 $\times$ class distribution to be collected. Thus the 'fraction' of the example gets less and less as it progresses down the rule list. A don't care value always passes the attribute test in full, and thus no fractional example remains to propagatete further down the rule list.