RIKTEXT: Rule Induction Kit for Text

Nitin Indurkhya

Contents

1

Preliminaries

1.1 Overview

The Rule Induction Kit for Text (RIKTEXT) is a complete software package for inducing highly compact decision rules for categorizing documents. Unlike complex numerical models, these rules are simple, logic rules that are often highly predictive. For example, a document with the word *dividend* suggests that the document can be categorized as one about company earnings. The objective is to determine the best set of rules for prediction and classification, where best is the smallest number of rules with a near-minimum error.

The technology and algorithms of RIKTEXT are discussed in Chapter 3 of the book *Text Mining: Predictive Methods for Analyzing Unstructured Information* (see http://www.data-miner.com for details on how to buy this book). They shall not be discussed in this guide, which focuses instead on how to use the program. This implementation of RIKTEXT is related to the more general-purpose Rule Induction Kit (RIK) (also available from www.data-miner.com). The main difference between the two is that RIKTEXT is customized for text categorization (2 classes only – documents belong to a category of interest, or not) and thus efficiently handles very large number of features. RIK, on the other hand, can handle a larger variety of classification problems, such as those with missing values. As a result, the data formats of the two programs are a bit different although it is not too difficult to convert between the two formats. There other functional differences between the two. However, this guide focuses on RIKTEXT only. For information on RIK, the reader may visit the data-miner.com website.

RIKTEXT is a *standalone* program. It does not require any other program or any other special installation (beyond what is provided by the operating system). A command line interface is used: The program name (*riktext*, in lowercase) is typed followed by a list of arguments. When using RIKTEXT, just type the program name with no arguments, and the program will display a description of the correct syntax.

1.2 Format of Rules

The rule-based classifier generated by RIKTEXT is a decision list in which the rules are ordered. All the rules are for a single (the positive) class except for the last rule

which is also the default (always satisfied) and is for the negative class. Here is an example of a ruleset generated by RIKTEXT:

```
shr --> EARN
div --> EARN
payout --> EARN
qtr --> EARN
dividend --> EARN
profit --> EARN
earnings & sees --> EARN
split & stock --> EARN
[TRUE] --> ~EARN
```

Thus, although the rules are ordered, the ordering is not very important (except for the rule).

1.3 Typographic Conventions

In this guide, for all examples demonstrating computer interaction the fontsize is smaller than the regular text font. The commandline prompt is shown as:

```
%
```

but naturally the actual prompt will differ from system to system (and even user to user). Commands to be typed by the user on a computer terminal immediately follow the prompt and take up the rest of the line. They will be in monospaced type. For example, if the user were to type the command *riktext*, this will be shown as:

```
% riktext
```

Output of programs is shown in monospaced type immediately after the user input. For example, the following shows the output of the program when the user types *riktext* without any arguments. As mentioned earlier, the program displays a description of the correct syntax and usage options.

```
% riktext
Usage:
riktext options dictionary categoryname vectorfile >ruleset
options:
  -q            No test cases, no pruning
  -p            No test cases, print table of pruned rulesets
  -h pct        Randomly selected pct training cases
  -t tfile      Use test cases in tfile
  -r k          Resampling with k-fold cross-validation
  -a rulefile   Apply rules in rulefile
  -s n          Output ruleset n (from table of pruned rulesets)

Default options: -p, "best" (1-SE) ruleset is output.
Other parameters provided in the file riktext.properties.
If riktext.properties not found, it is created with default values.
```

1.4 Installation

RIKTEXT is available for PC-Linux, and Windows. The installation process differs for each.

1.4.1 Linux Installation

The linux installation consists of the following steps:

- Create a directory named *riktext* ("mkdir riktext")
- Download the file from the data-miner.com website to directory riktext and save as riktext.tgz
- Type "tar -xvzf riktext.tgz"
- Add the directory *riktext* to your path

1.4.2 Windows Installation

The windows installation consists of the following steps (most of them require the user to type commands in either a command-prompt or MSDOS-prompt window):

- Create a directory named riktext ("mkdir c:\riktext")
- Download the file from the data-miner.com website to \riktext and save as installriktext.exe
- Type "cd \riktext" and run installriktext.exe by typing "installriktext"
- Add directory \riktext to your path and set environmental variable LFN to y.
  - Windows 9x: edit autoexec.bat using "notepad c:\autoexec.bat"; add "c:\riktext;" to the PATH line. (Alternative: add new line "PATH c:\riktext;%PATH%" at the end of the file.) Also add new line "set LFN=y" (enables long file names). A single space follows the words PATH and set. These lines should contain no additional spaces.
  - Windows NT/2000/XP click on control panel (system folder/environment). Add "c:\riktext;" (no spaces) to the environmental variable PATH. Create the environmental variable LFN and set it to a value of y.

Notes: The RIKTEXT program is run from an MS-DOS (or command-prompt) window. The path and environment changes may not take effect until the next login or restart. A disk other than c: may be used.

1.4.3 Verifying the Installation

If the installation was successful, the user should be able to type the command *riktext* without arguments in any directory and get the output shown in Section 1.3. If, instead, a *command not found* message is obtained, make sure that the riktext directory has the program (check for the file *riktext* or *riktext.exe*), that the program's executable permission is switched on, and that the path variable is changed and has taken effect.

1.4.4 Cleaning Up

Following successful installation, the following files may be deleted or moved to backup storage:

- For Linux: riktext.tgz
- For Windows: installriktext.exe

2

Data Format

2.1 Sparse Vectors

RIKTEXT processes data in a format we call sparse vector form. This format is shared with the TMSK (Text Miner Software Kit) that is available in conjunction with RIKTEXT. TMSK has tools for creating data in this format from XML text documents. The user is referred to the TMSK user guide for details on how to create vectors. In this chapter we shall describe the format of the data files required by RIKTEXT.

The documents are converted into a spreadsheet format where each row corresponds to a document, and each column corresponds to a word from a dictionary. Individual cells in the spreadsheet are filled with boolean values (indicating presence/absence of the word in the document) or frequency counts (number of times the word appears in the document). For more details, the user may refer to Chapters 2 and 3 of the book. Typically, the number of words (columns) is very large and for a given document (row), most of the words do not apply and the corresponding cells are zero. Hence it is more efficient to store only the information of the non-zero cells (the cell number and its value). This is the sparse vector form. Figure 2.1 gives a simple example of the sparse vector form that corresponds to a spreadsheet.

RIKTEXT expects a vector file in this format with the following constraints:

- All the non-zero pairs for a document appear on the same line.
- The non-zero pairs for each document should be in increasing order of column numbers.
- The non-zero pairs are separated by white space.
- Each non-zero pair is formatted as first the column number, then an character, and finally the value.
- The name of the vector file itself can be upto 64 characters.

If TMSK is used for generating the vectors, the vectorfile will be in the proper format.

For example, the vector file for RIKTEXT corresponding to the spreadsheet of Figure 2.1 would be as follows:

Spreadsheet				Sparse Vectors
0	15	0	3	(2,15) (4,3)
12	0	0	0	(1,12)
8	0	5	2	(1,8) (3,5) (4,2)

Figure 2.1: Spreadsheet to Sparse Vectors

```
2@15 4@3
1@12
1@8 3@5 4@2
```

2.2 Labeled and Unlabeled Vectors

For training a classifier, we need labeled documents. Having obtained a classifier, we would like to use it to classify unlabeled (new) documents. RIKTEXT processes both labeled and unlabeled vectors in the following format:

- If documents are labeled, then the labels appears at the front of the corresponding vectors (separated by whitespace from the non-zero pairs).
- Only binary labels are permitted – documents either belong to a class (label=1) or not (label=0). A categorizer is built for the positive class (cases with label=1).
- For training a classifier, labeled vectors must be provided.
- For applying an existing rule-based classifier, the vectors can be either labeled or unlabeled (if they are labeled, performance results are computed).
- In a vector file, all vectors must be of the same type (labeled or unlabeled).

If the vectors are generated by TMSK, the labels (if available) are attached properly.

In the previous section, the vector file shown consisted of unlabeled vectors. A labeled vector file for RIKTEXT corresponding to the spreadsheet of Figure 2.1 (with hypothetical labels) might be as follows (the first and last case belong to the class, the middle case does not belong to the class):

```
1 2@15 4@3
0 1@12
1 1@8 3@5 4@2
```

2.3 Dictionaries

The vector file contains only numbers, whitespace and characters. With a large number of columns, it is necessary to map the column numbers to the corresponding words in the dictionary. This is done by means of a separate dictionary file. TMSK has programs that generate both the dictionary and corresponding sparse vectors. Here we shall only describe the format of the dictionary file expected by RIKTEXT:

- There should be atleast as many words in the dictionary file as non-zero columns in the vector file (every non-zero column in the vector file should have a corresponding name in the dictionary file).
- Each word should be on a separate line.
- Words can be upto 64 characters long (longer words are truncated).
- Words are case-sensitive.
- If duplicate words are detected, RIKTEXT will terminate with an error message.
- The name of the dictionary file itself can be upto 64 characters.

As with the vectorfile, if TMSK is used for the generation process, the dictionary will be in the proper format.

For example, for the sparse vectors of Figure 2.1, there are four columns, and so a hypothetical dictionary for the corresponding vector file might be:

```
company
dividend
profits
share
```

Note that only the column names are provided in the dictionary file. Thus, the dictionary file remains the same whether the vectors are labeled or unlabeled. The name of the category (for labeled data) is provided to RIKTEXT on the commandline itself. This shall be discussed in more detail in the next chapter.

3

Running RIKTEXT

3.1 Command-Line Arguments

As mentioned earlier, running *riktext* without any arguments will give the syntax and options for running RIKTEXT:

```
% riktext
Usage:
riktext options dictionary categoryname vectorfile >ruleset
options:
  -q            No test cases, no pruning
  -p            No test cases, print table of pruned rulesets
  -h pct        Randomly selected pct training cases
  -t tfile      Use test cases in tfile
  -r k          Resampling with k-fold cross-validation
  -a rulefile   Apply rules in rulefile
  -s n          Output ruleset n (from table of pruned rulesets)

Default options: -p, "best" (1-SE) ruleset is output.
Other parameters provided in the file riktext.properties.
If riktext.properties not found, it is created with default values.
```

Input data is provided via the files *dictionary* and *vectorfile*. The format of these files was described in Chapter 2. The reader will notice that the program makes use of another file *riktext.properties* that determines some secondary characteristics of rule learning, If this file doesn't exist, it is created with default parameter values. It is described in more detail later in the documentation. The *categoryname* is a command-line argument that specifies the name of the positive class (note that RIKTEXT is always dealing directly with binary classification problems). The negative class is everything else. For example, the positive class may be referred to as *earnings*; the negative class would then be called *earnings*. The options represent various modes of operation for RIKTEXT and will be discussed in more detail later on.

The output is usually an induced ruleset and is written to *standard output* which can be redirected to a file; summary information about error-estimates and pruned rulesets is written to *standard error*, the terminal (which can therefore be saved in a separate file).

3.2 A Simple Example

In this section we show how RIKTEXT can be used on sample data. It is not important that the reader understand the data used for this purpose. It is sufficient to know that there are 3 files:

1. *etrn.vec*: containing labeled vectors for learning a classifier.
2. *enam.dx*: containing the corresponding dictionary.
3. *etst.vec*: containing labeled hidden test data (for evaluation).

The positive cases correspond to a category designated as *EARN*.

The following shows how riktext generates rules for etrn.vec, estimates the performance of different rulesets using 10-fold cross-validation, and prints out the "best" ruleset (which is saved in the file *etrn.rul*. Since riktext.properties does not exist, the program also creates this file with default values for future manipulation by the user.

```
% riktext -r 10 enam.dx EARN etrn.vec >etrn.rul
riktext: automatically generating default riktext.properties file

Table of pruned rule sets
(* = minimum error; ** = within 1-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      27      48     0.0309    0.0436   0.0021    143.4      1.00
    2      26      46     0.0311    0.0409   0.0020     45.5      1.00
    3*     19      32     0.0338    0.0385   0.0020     31.4      1.86
    4      16      25     0.0351    0.0387   0.0020     24.5      3.17
    5**    13      20     0.0376    0.0400   0.0020     19.8      4.80
    6       9      11     0.0437    0.0464   0.0021     10.7      6.56
    7       9      10     0.0451    0.0472   0.0022     10.0     13.00
    8       8       9     0.0468    0.0500   0.0022      8.9     16.00
    9       7       7     0.0523    0.0547   0.0023      6.8     26.50
   10       6       6     0.0561    0.0574   0.0024      5.9     37.00
   11       5       5     0.0596    0.0596   0.0024      5.0     57.00
   12       4       4     0.0667    0.0667   0.0025      4.0     69.00
   13       3       3     0.0788    0.0788   0.0027      3.0    116.00
   14       2       2     0.1015    0.1015   0.0031      2.0    218.00
   15       1       1     0.2996    0.2996   0.0047      1.0   1902.00
K-fold resampling, k=10

**********************************************************************
Selected rule set

    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. qtr --> EARN
    5. dividend --> EARN
    6. profit & expects --> EARN
    7. net & profit --> EARN
    8. profit & stg --> EARN
    9. earnings & sees --> EARN
   10. quarter & cts --> EARN
   11. split & stock --> EARN
   12. loss & results --> EARN
```

```
   13. [TRUE] --> ~EARN

Additional Statistics (Training Cases):
precision: 95.8121      recall: 91.4494         f-measure: 93.5799

Additional Statistics (Test Cases):
precision: 95.5759      recall: 90.8585         f-measure: 93.1575
```

The statistics for test cases are, of course, cross-validated estimates.

The ruleset is saved in a text file and can be readily viewed and edited (the first line contains complexity and training data performance and should not be edited):

```
% cat etrn.rul
Ruleset made using resampling mode. [4.8,0.037592,20]
EARN
~EARN
shr --> EARN
div --> EARN
payout --> EARN
qtr --> EARN
dividend --> EARN
profit & expects --> EARN
net & profit --> EARN
profit & stg --> EARN
earnings & sees --> EARN
quarter & cts --> EARN
split & stock --> EARN
loss & results --> EARN
[TRUE] --> ~EARN
```

We can apply the rules to new data. For example, the following applies the rules in *etrn.rul* to the evaluation test dataset *etst.vec*:

```
% riktext -a etrn.rul enam.dx EARN etst.vec >etst.res

Ruleset etrn.rul applied:
        Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
           13      20     0.0376    0.0273   0.0028      0.0      4.80

Additional Statistics (Test Cases):
precision: 96.1147      recall: 95.5842         f-measure: 95.8487

actual and predicted categories for 3299 test cases written on standard-output.
```

The output is a case-by-case listing of actual and predicted categories for the test data:

```
% cat etst.res
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
```

```
~EARN    EARN
~EARN    ~EARN
~EARN    ~EARN
~EARN    ~EARN
~EARN    ~EARN
~EARN    ~EARN
~EARN    ~EARN
~EARN    ~EARN
EARN     EARN
EARN     EARN
~EARN    ~EARN
~EARN    ~EARN
     .... (all 3299 lines are not shown here)
```

The riktext.properties can be edited and the program re-run with new parameters.

4

RIKTEXT Options

In Chapter 3 we saw a very simple example of using RIKTEXT. In this chapter we explain in more detail the various optional ways in which RIKTEXT can be used. We shall use the same set of data files throughout for illustration purposes:

- *etrn.vec*, a vectorfile of labeled training data
- *etst.vec*, a vectorfile of separate labeled test data
- *unlab.vec*, a vectorfile of unlabeled data
- *enam.dx*, a dictionary file that corresponds to the vectorfiles.

The positive category in the labeled data is referred to by the name *EARN*.

4.1 The -q Option: Covering Ruleset

This is used to obtain a covering set of rules that makes no errors on the training cases. For example, the following would generate a covering rule set for *etrn.vec*:

```
% riktext -q enam.dx EARN etrn.vec >etrn.cov

Table of pruned rule sets
(* = minimum error; ** = within 1-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1**    59     131     0.4231    0.0000   0.0000      0.0      0.00
Training cases only

*********************************************************************
Selected rule set

    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. profit & qtr --> EARN
    5. revs --> EARN
    6. oper --> EARN
    7. earnings & qtr --> EARN
```

```
    8. loss & qtr --> EARN
    9. results & qtr --> EARN
   10. results & st --> EARN
   11. payable & quarter --> EARN
   12. note & quarter --> EARN
   13. p & quarter --> EARN
   14. dividend --> EARN
   15. profit & ended --> EARN
   16. quarterly & loss --> EARN
   17. split & loss --> EARN
   18. losses & extraordinary --> EARN
   19. net & profit & operations --> EARN
   20. split & nine --> EARN
   21. prior & record & oil --> EARN
   22. profits & sees & profit --> EARN
   23. marks & profit --> EARN
   24. results & stg & rose --> EARN
   25. profit & stg --> EARN
   26. net & sees --> EARN
   27. cts & earnings --> EARN
   28. qtr & cash --> EARN
   29. expects & results --> EARN
   30. calif & net --> EARN
   31. earlier & net & costs --> EARN
   32. results & losses --> EARN
   33. qtr & losses --> EARN
   34. profits & seven & rose --> EARN
   35. operating & prior --> EARN
   36. profit & full & mln & ltd --> EARN
   37. note & net --> EARN
   38. quarterly & cts --> EARN
   39. sees & operations & quarter --> EARN
   40. marks & results --> EARN
   41. sees & results & quarter --> EARN
   42. split & common --> EARN
   43. split & payable --> EARN
   44. fiscal & results & earlier --> EARN
   45. company & sees & tax --> EARN
   46. split & sets --> EARN
   47. tax & results & sale --> EARN
   48. costs & results & sees --> EARN
   49. extraordinary & billion --> EARN
   50. profit & gain & billion --> EARN
   51. prior & reported & board --> EARN
   52. loss & prior & capital --> EARN
   53. unit & board & earnings --> EARN
   54. gain & sale --> EARN
   55. profits & income --> EARN
   56. results & unit --> EARN
   57. sees & operating & costs --> EARN
   58. sets & pay & cash --> EARN
   59. [TRUE] --> ~EARN

Additional Statistics (Training Cases):
precision: 95.8616      recall: 93.3959         f-measure: 94.6127
```

As is clear from the above example, covering sets can be quite large. As in this example, it may not be possible to obtain an error-free covering set. The main advantage of using this option is that one can quickly get an idea of the maximal complexity of solutions that can be obtained from the training data.

4.2 The -p Option: Pruned Rulesets

Instead of producing only a single ruleset, one may want to compare a number of potential solutions. This option prints a summary table with such information. The summary table contains valuable information about potential solutions. It is useful to see and compare the differing complexities of solutions and their performance on training data alone. Sometimes, this can highlight problems in the data or the particular parameters being used. For example, a summary table is produced by the following:

```
% riktext -p enam.dx EARN etrn.vec >etrn.cov

*********************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
     1     27    48     0.0309    0.0000   0.0000      0.0       1.00
    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. revs --> EARN
    5. oper --> EARN
    6. earnings & qtr --> EARN
    7. loss & qtr --> EARN
    8. net & qtr --> EARN
    9. note & quarter --> EARN
   10. dividend --> EARN
   11. profit & expects --> EARN
   12. net & profit --> EARN
   13. profits & expects --> EARN
   14. marks & profit --> EARN
   15. profit & stg --> EARN
   16. earnings & sees --> EARN
   17. cts & earnings --> EARN
   18. results & losses --> EARN
   19. quarter & cts --> EARN
   20. earnings & business & quarter --> EARN
   21. sees & results --> EARN
   22. split & stock --> EARN
   23. record & cts --> EARN
   24. split & shareholders --> EARN
   25. costs & results --> EARN
   26. profits & income --> EARN
   27. [TRUE] --> ~EARN

*********************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
     2     26    46     0.0311    0.0000   0.0000      0.0       1.00
    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. revs --> EARN
    5. oper --> EARN
```

```
    6. earnings & qtr --> EARN
    7. loss & qtr --> EARN
    8. net & qtr --> EARN
    9. note & quarter --> EARN
   10. dividend --> EARN
   11. profit & expects --> EARN
   12. net & profit --> EARN
   13. profits & expects --> EARN
   14. marks & profit --> EARN
   15. profit & stg --> EARN
   16. earnings & sees --> EARN
   17. cts & earnings --> EARN
   18. quarter & cts --> EARN
   19. earnings & business & quarter --> EARN
   20. sees & results --> EARN
   21. split & stock --> EARN
   22. record & cts --> EARN
   23. split & shareholders --> EARN
   24. costs & results --> EARN
   25. profits & income --> EARN
   26. [TRUE] --> ~EARN

*********************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
     3     19    32     0.0338    0.0000   0.0000      0.0       1.86
    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. revs --> EARN
    5. loss & qtr --> EARN
    6. net & qtr --> EARN
    7. dividend --> EARN
    8. profit & expects --> EARN
    9. net & profit --> EARN
   10. marks & profit --> EARN
   11. profit & stg --> EARN
   12. earnings & sees --> EARN
   13. cts & earnings --> EARN
   14. quarter & cts --> EARN
   15. sees & results --> EARN
   16. split & stock --> EARN
   17. record & cts --> EARN
   18. costs & results --> EARN
   19. [TRUE] --> ~EARN

*********************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
     4     16    25     0.0351    0.0000   0.0000      0.0       3.17
    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. revs --> EARN
    5. qtr --> EARN
    6. dividend --> EARN
    7. profit & expects --> EARN
    8. net & profit --> EARN
    9. profit & stg --> EARN
```

```
   10. earnings & sees --> EARN
   11. cts & earnings --> EARN
   12. quarter & cts --> EARN
   13. split & stock --> EARN
   14. record & cts --> EARN
   15. loss & results --> EARN
   16. [TRUE] --> ~EARN

*********************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
     5     13    20     0.0376    0.0000   0.0000      0.0       4.80
    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. qtr --> EARN
    5. dividend --> EARN
    6. profit & expects --> EARN
    7. net & profit --> EARN
    8. profit & stg --> EARN
    9. earnings & sees --> EARN
   10. quarter & cts --> EARN
   11. split & stock --> EARN
   12. loss & results --> EARN
   13. [TRUE] --> ~EARN

*********************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
     6      9    11     0.0437    0.0000   0.0000      0.0       6.56
    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. qtr --> EARN
    5. dividend --> EARN
    6. profit --> EARN
    7. earnings & sees --> EARN
    8. split & stock --> EARN
    9. [TRUE] --> ~EARN

*********************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
     7      9    10     0.0451    0.0000   0.0000      0.0      13.00
    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. qtr --> EARN
    5. dividend --> EARN
    6. profit --> EARN
    7. earnings & sees --> EARN
    8. split --> EARN
    9. [TRUE] --> ~EARN

*********************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
     8      8     9     0.0468    0.0000   0.0000      0.0      16.00
    1. shr --> EARN
    2. div --> EARN
    3. qtr --> EARN
```

```
    4. dividend --> EARN
    5. profit --> EARN
    6. earnings & sees --> EARN
    7. split --> EARN
    8. [TRUE] --> ~EARN

************************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
     9      7     7     0.0523    0.0000   0.0000      0.0      26.50
    1. shr --> EARN
    2. div --> EARN
    3. qtr --> EARN
    4. dividend --> EARN
    5. profit --> EARN
    6. split --> EARN
    7. [TRUE] --> ~EARN

************************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
    10      6     6     0.0561    0.0000   0.0000      0.0      37.00
    1. shr --> EARN
    2. div --> EARN
    3. qtr --> EARN
    4. dividend --> EARN
    5. profit --> EARN
    6. [TRUE] --> ~EARN

************************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
    11      5     5     0.0596    0.0000   0.0000      0.0      57.00
    1. shr --> EARN
    2. cts --> EARN
    3. dividend --> EARN
    4. profit --> EARN
    5. [TRUE] --> ~EARN

************************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
    12      4     4     0.0667    0.0000   0.0000      0.0      69.00
    1. shr --> EARN
    2. cts --> EARN
    3. profit --> EARN
    4. [TRUE] --> ~EARN

************************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
    13      3     3     0.0788    0.0000   0.0000      0.0     116.00
    1. cts --> EARN
    2. profit --> EARN
    3. [TRUE] --> ~EARN

************************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
    14      2     2     0.1015    0.0000   0.0000      0.0     218.00
    1. cts --> EARN
    2. [TRUE] --> ~EARN
```

```
*********************************************************************
 Prune  Rules  Vars  Train Err  Test Err  Test SD  MeanVar    Err/Var
    15      1     1     0.2996    0.0000   0.0000      0.0    1902.00
    1. [TRUE] --> ~EARN

Table of pruned rule sets
(* = minimum error; ** = within 1-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1**    27      48     0.0309    0.0000   0.0000      0.0      1.00
    2      26      46     0.0311    0.0000   0.0000      0.0      1.00
    3      19      32     0.0338    0.0000   0.0000      0.0      1.86
    4      16      25     0.0351    0.0000   0.0000      0.0      3.17
    5      13      20     0.0376    0.0000   0.0000      0.0      4.80
    6       9      11     0.0437    0.0000   0.0000      0.0      6.56
    7       9      10     0.0451    0.0000   0.0000      0.0     13.00
    8       8       9     0.0468    0.0000   0.0000      0.0     16.00
    9       7       7     0.0523    0.0000   0.0000      0.0     26.50
   10       6       6     0.0561    0.0000   0.0000      0.0     37.00
   11       5       5     0.0596    0.0000   0.0000      0.0     57.00
   12       4       4     0.0667    0.0000   0.0000      0.0     69.00
   13       3       3     0.0788    0.0000   0.0000      0.0    116.00
   14       2       2     0.1015    0.0000   0.0000      0.0    218.00
   15       1       1     0.2996    0.0000   0.0000      0.0   1902.00
Training cases only

*********************************************************************
Selected rule set

    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. revs --> EARN
    5. oper --> EARN
    6. earnings & qtr --> EARN
    7. loss & qtr --> EARN
    8. net & qtr --> EARN
    9. note & quarter --> EARN
   10. dividend --> EARN
   11. profit & expects --> EARN
   12. net & profit --> EARN
   13. profits & expects --> EARN
   14. marks & profit --> EARN
   15. profit & stg --> EARN
   16. earnings & sees --> EARN
   17. cts & earnings --> EARN
   18. results & losses --> EARN
   19. quarter & cts --> EARN
   20. earnings & business & quarter --> EARN
   21. sees & results --> EARN
   22. split & stock --> EARN
   23. record & cts --> EARN
   24. split & shareholders --> EARN
   25. costs & results --> EARN
   26. profits & income --> EARN
   27. [TRUE] --> ~EARN
```

```
Additional Statistics (Training Cases):
precision: 95.9402      recall: 93.6392         f-measure: 94.7757
```

As with the -q option, there are no test cases. The user is still exploring the possibilities from the training data. The summary table is written to standard error and can be saved in a file separately. Besides the summary table, the various pruned rulesets are also listed. The output ruleset is still the covering ruleset. In this example, notice that it is not the same as that with the -q option. It is cleaned up substantially – not only is it more compact, but performance is improved vastly.

4.3 The -h Option: Randomly Selected Test Cases

When no additional cases are available, but test cases are desired for evaluation of potential solutions, the classic approach is to hide some of the available cases from the training program and use them only for testing. Typically, test cases are selected randomly, with the user specifying how many cases should be used for testing, with the rest being used for training. Saving a third of the cases for testing (i.e. using the remaining two-thirds for training) is quite common when there are sufficient number of cases. For example, in the following, two-thirds of the available cases are randomly selected for training:

```
% riktext -h 66.7 enam.dx EARN etrn.vec >etrn.rul

Table of pruned rule sets
(* = minimum error; ** = within 1-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      51     111     0.0289    0.0463   0.0037      0.0      0.00
    2      41      81     0.0304    0.0441   0.0036      0.0      0.33
    3      25      45     0.0312    0.0375   0.0034      0.0      1.00
    4      23      40     0.0320    0.0385   0.0034      0.0      1.20
    5*     22      38     0.0325    0.0375   0.0034      0.0      1.50
    6      17      27     0.0347    0.0378   0.0034      0.0      1.45
    7      16      25     0.0351    0.0385   0.0034      0.0      3.00
    8      11      16     0.0409    0.0407   0.0035      0.0      4.44
    9**    10      12     0.0437    0.0407   0.0035      0.0      4.50
   10       9      10     0.0456    0.0450   0.0037      0.0      9.00
   11       8       9     0.0473    0.0466   0.0037      0.0     11.00
   12       6       7     0.0539    0.0532   0.0040      0.0     21.00
   13       5       5     0.0598    0.0591   0.0042      0.0     32.00
   14       4       4     0.0662    0.0679   0.0044      0.0     41.00
   15       3       3     0.0785    0.0794   0.0048      0.0     79.00
   16       2       2     0.0987    0.1073   0.0055      0.0    129.00
   17       1       1     0.2996    0.2996   0.0081      0.0   1287.00
Random test cases, 3198 (33.3%) test cases

*********************************************************************
Selected rule set

    1. shr --> EARN
    2. div --> EARN
    3. dividend --> EARN
    4. payout --> EARN
    5. qtr --> EARN
    6. earnings & sees --> EARN
```

```
    7. quarter & cts --> EARN
    8. split --> EARN
    9. profit --> EARN
   10. [TRUE] --> ~EARN

Additional Statistics (Training Cases):
precision: 93.4748      recall: 91.8187         f-measure: 92.6393

Additional Statistics (Test Cases):
precision: 94.1365      recall: 92.1712         f-measure: 93.1435
```

The covering set (and the summary table) is different from that with the -p option because it is generated from only two-thirds of the total cases. The pruned rulesets are evaluated using the remaining one-third test cases and the "best" ruleset is saved.

4.4 The -t Option: Separate Test Cases

If a separate set of validation cases is available, they can be used to evaluate the pruned rulesets using this option. For example, in the following, we use the *etst.vec* cases to evaluate the pruned rulesets:

```
% riktext -t etst.vec enam.dx EARN etrn.vec >etrn.rul

Table of pruned rule sets
(* = minimum error; ** = within 1-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      27      48     0.0309    0.0282   0.0029      0.0      1.00
    2      26      46     0.0311    0.0282   0.0029      0.0      1.00
    3      19      32     0.0338    0.0282   0.0029      0.0      1.86
    4      16      25     0.0351    0.0282   0.0029      0.0      3.17
    5**    13      20     0.0376    0.0273   0.0028      0.0      4.80
    6       9      11     0.0437    0.0336   0.0031      0.0      6.56
    7       9      10     0.0451    0.0361   0.0032      0.0     13.00
    8       8       9     0.0468    0.0367   0.0033      0.0     16.00
    9       7       7     0.0523    0.0391   0.0034      0.0     26.50
   10       6       6     0.0561    0.0388   0.0034      0.0     37.00
   11       5       5     0.0596    0.0576   0.0041      0.0     57.00
   12       4       4     0.0667    0.0564   0.0040      0.0     69.00
   13       3       3     0.0788    0.0752   0.0046      0.0    116.00
   14       2       2     0.1015    0.0867   0.0049      0.0    218.00
   15       1       1     0.2996    0.3295   0.0082      0.0   1902.00
Alternate database test cases, 3299 test cases

**********************************************************************
Selected rule set

    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. qtr --> EARN
    5. dividend --> EARN
    6. profit & expects --> EARN
    7. net & profit --> EARN
    8. profit & stg --> EARN
```

```
    9. earnings & sees --> EARN
   10. quarter & cts --> EARN
   11. split & stock --> EARN
   12. loss & results --> EARN
   13. [TRUE] --> ~EARN

Additional Statistics (Training Cases):
precision: 95.8121      recall: 91.4494         f-measure: 93.5799

Additional Statistics (Test Cases):
precision: 96.1147      recall: 95.5842         f-measure: 95.8487
```

All the training cases are used for obtaining the covering ruleset and the pruned rulesets. As a result, the summary table is identical to the one with the -p option except that it has test estimates filled in. The validation cases are used only for the final test and selection phase.

4.5 The -r Option: Cross-Validation Resampling

When limited number of cases are available and it is desired to maximise their use for learning, resampling techniques such as cross-validation are useful for obtaining test-set estimates. This option allows the use of k-fold cross-validation. The value of k is specified by the user (usually 10 is a good choice). For example, 10-fold cross-validation is used in the following:

```
% riktext -r 10 enam.dx EARN etrn.vec >etrn.rul

Table of pruned rule sets
(* = minimum error; ** = within 1-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      27      48     0.0309    0.0436   0.0021    143.4      1.00
    2      26      46     0.0311    0.0409   0.0020     45.5      1.00
    3*     19      32     0.0338    0.0385   0.0020     31.4      1.86
    4      16      25     0.0351    0.0387   0.0020     24.5      3.17
    5**    13      20     0.0376    0.0400   0.0020     19.8      4.80
    6       9      11     0.0437    0.0464   0.0021     10.7      6.56
    7       9      10     0.0451    0.0472   0.0022     10.0     13.00
    8       8       9     0.0468    0.0500   0.0022      8.9     16.00
    9       7       7     0.0523    0.0547   0.0023      6.8     26.50
   10       6       6     0.0561    0.0574   0.0024      5.9     37.00
   11       5       5     0.0596    0.0596   0.0024      5.0     57.00
   12       4       4     0.0667    0.0667   0.0025      4.0     69.00
   13       3       3     0.0788    0.0788   0.0027      3.0    116.00
   14       2       2     0.1015    0.1015   0.0031      2.0    218.00
   15       1       1     0.2996    0.2996   0.0047      1.0   1902.00
K-fold resampling, k=10

*********************************************************************
Selected rule set

    1. shr --> EARN
    2. div --> EARN
    3. payout --> EARN
    4. qtr --> EARN
```

```
    5. dividend --> EARN
    6. profit & expects --> EARN
    7. net & profit --> EARN
    8. profit & stg --> EARN
    9. earnings & sees --> EARN
   10. quarter & cts --> EARN
   11. split & stock --> EARN
   12. loss & results --> EARN
   13. [TRUE] --> ~EARN

Additional Statistics (Training Cases):
precision: 95.8121      recall: 91.4494         f-measure: 93.5799

Additional Statistics (Test Cases):
precision: 95.5759      recall: 90.8585         f-measure: 93.1575
```

Since the rules and summary table are obtained from all the training cases, the summary table is identical to the ones with the -t and -p options except for the test-set estimates (which, of course, are obtained by resampling in this option).

4.6 The -a Option: Apply Existing Rules

Once a ruleset is obtained, one may want to apply the ruleset to new cases. The new cases may or may not be labeled. This option allows the use of RIKTEXT for applying existing rulesets. For example, the ruleset obtained by resampling can be applied to the labeled cases in *etst.vec* as follows:

```
% riktext -a etrn.rul enam.dx EARN etst.vec >etst.res

Ruleset etrn.rul applied:
        Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
           13      20     0.0376    0.0273   0.0028      0.0      4.80

Additional Statistics (Test Cases):
precision: 96.1147      recall: 95.5842         f-measure: 95.8487

actual and predicted categories for 3299 test cases written on standard-output.
```

The output is a case-by-case listing of actual and predicted categories for the test data:

```
% cat etst.res
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   EARN
~EARN   ~EARN
~EARN   ~EARN
```

```
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
~EARN   ~EARN
EARN    EARN
EARN    EARN
~EARN   ~EARN
~EARN   ~EARN
     .... (all 3299 lines are not shown here)
```

If the cases are not labeled, no test set evaluation is possible. Instead, the predicted categories for the cases are written, one per line, to standard-output where they can be saved in a file. For example, if *unlab.vec* contains unlabeled cases, the ruleset can be applied to these cases as before. RIKTEXT automatically detects that these are unlabeled cases:

```
% riktext -a etrn.rul enam.dx EARN unlab.vec >unlab.res

Ruleset etrn.rul applied:
        Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
           13      20     0.0376    0.0000   0.0000      0.0      4.80

predicted categories for 1000 uncategorized cases written on standard-output.
```

The file *unlab.res* will contain the predicted categories:

```
% cat unlab.res
EARN
~EARN
EARN
~EARN
~EARN
EARN
~EARN
EARN
~EARN
EARN
~EARN
~EARN
EARN
~EARN
EARN
EARN
~EARN
     .... (all 1000 lines are not shown here)
```

4.7 The -s Option: Select a Specific Ruleset

In all the options so far, the output ruleset is determined by RIKTEXT. There are scenarios where the user may want to select and save a different ruleset from the summary table (for example, if rulesets need to be compared on independent cases). This option allows the user the specify the ruleset to be saved. The ruleset is specified by its number in the summary table. Typically, this option is used in a second run over the same data. The first run gives a summary table for examination. Note that this option works *in conjunction* with any of the other options. For example, to output and save the thirteenth ruleset (counting from the top of the summary table obtained

using the -h option with one-third of the cases kept as test cases), the following might be done:

```
 riktext -s 13 -h 66.7 enam.dx EARN etrn.vec >simp.rul

Table of pruned rule sets
(* = minimum error; ** = within 1-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      51     111     0.0289    0.0463   0.0037      0.0      0.00
    2      41      81     0.0304    0.0441   0.0036      0.0      0.33
    3      25      45     0.0312    0.0375   0.0034      0.0      1.00
    4      23      40     0.0320    0.0385   0.0034      0.0      1.20
    5*     22      38     0.0325    0.0375   0.0034      0.0      1.50
    6      17      27     0.0347    0.0378   0.0034      0.0      1.45
    7      16      25     0.0351    0.0385   0.0034      0.0      3.00
    8      11      16     0.0409    0.0407   0.0035      0.0      4.44
    9**    10      12     0.0437    0.0407   0.0035      0.0      4.50
   10       9      10     0.0456    0.0450   0.0037      0.0      9.00
   11       8       9     0.0473    0.0466   0.0037      0.0     11.00
   12       6       7     0.0539    0.0532   0.0040      0.0     21.00
   13       5       5     0.0598    0.0591   0.0042      0.0     32.00
   14       4       4     0.0662    0.0679   0.0044      0.0     41.00
   15       3       3     0.0785    0.0794   0.0048      0.0     79.00
   16       2       2     0.0987    0.1073   0.0055      0.0    129.00
   17       1       1     0.2996    0.2996   0.0081      0.0   1287.00
Random test cases, 3198 (33.3%) test cases

*********************************************************************
Selected rule set

    1. shr --> EARN
    2. cts --> EARN
    3. dividend --> EARN
    4. profit --> EARN
    5. [TRUE] --> ~EARN

Additional Statistics (Training Cases):
precision: 92.0131      recall: 87.6498         f-measure: 89.7785

Additional Statistics (Test Cases):
precision: 91.8390      recall: 88.1002         f-measure: 89.9307
```

5

Interpreting the Summary Table

In this chapter we shall explain the various parts of the summary table. Here is an example of a summary table:

RSet	Rules	Vars	Train Err	Test Err	Test SD	MeanVar	Err/Var
1	51	111	0.0289	0.0463	0.0037	0.0	0.00
2	41	81	0.0304	0.0441	0.0036	0.0	0.33
3	25	45	0.0312	0.0375	0.0034	0.0	1.00
4	23	40	0.0320	0.0385	0.0034	0.0	1.20
5*	22	38	0.0325	0.0375	0.0034	0.0	1.50
6	17	27	0.0347	0.0378	0.0034	0.0	1.45
7	16	25	0.0351	0.0385	0.0034	0.0	3.00
8	11	16	0.0409	0.0407	0.0035	0.0	4.44
9**	10	12	0.0437	0.0407	0.0035	0.0	4.50
10	9	10	0.0456	0.0450	0.0037	0.0	9.00
11	8	9	0.0473	0.0466	0.0037	0.0	11.00
12	6	7	0.0539	0.0532	0.0040	0.0	21.00
13	5	5	0.0598	0.0591	0.0042	0.0	32.00
14	4	4	0.0662	0.0679	0.0044	0.0	41.00
15	3	3	0.0785	0.0794	0.0048	0.0	79.00
16	2	2	0.0987	0.1073	0.0055	0.0	129.00
17	1	1	0.2996	0.2996	0.0081	0.0	1287.00

This table is printed to standard error (usually the screen) and can be redirected to a file if it needs to be saved. As can be seen, it displays a number of rule sets. Each rule set is numbered under the column *RSet*. A single "*" delineates the rule set with the minimum error rate. A "**" indicates the rule set that is the minimum or is very close to the minimum but may be simpler than the minimum.

The complexity of the rule set is the concern of the next two columns, *Rules*, the number of rules in the rule set and *Vars*, the total number of conjuncts in the left-hand-side of the rules in the rule set. Thus, for example, in the above table, rule set No. 12 has 6 rules and 7 conjuncts. Since each rule has atleast one conjunct, if the number of rules and conjuncts is close, it suggests simple rules. Are fewer, but longer rules better than many simple rules? This would, of course, depend on the nature of the application. Generally though, one needs both measures (rules and vars) to get a clear understanding of the complexity of the rule set.

The next column *Train Err* gives the error-rate of the rule sets on the training data. Error is measured in a straightforward manner – number of misclassified cases

divided by the total number of cases. The training error is saved in the first line of saved rule sets, so it is always available – even when applying an existing ruleset to new cases. When comparing rule sets, the training error provides an upper-bound on future performance (it is highly unlikely that performance on new unseen cases will be better than that on the training cases used for deriving the rules in the first place).

The next two columns, *Test Err* and *Test SD*, relate to future performance. The first is an error-rate estimate; the second is the standard deviation of the estimate. The test error estimate is obtained in a variety of ways depending on how RIKTEXT is invoked:

- From a randomly selected subset of the training cases
- By resampling (cross-validation)
- From a separate (hopefully independent) test set.

The standard deviation of the estimates is helpful in comparing the estimates.

MeanVar is the average number or variables of the resampled rule set that approximates in size the rule set for the full data. This helps determine the reliability of the resampled estimates.

Err/*var* indicates the number of new errors per variable that were introduced when the previous rule set was pruned to the smaller size. This gives an indication of the quality of the solutions.

5.1 Model Selection via the Summary Table

RIKTEXT automatically selects the "best" rule set based on test set error. It does this by examining the one with the lowest error, then selects the smallest one in the summary table within one standard error of this minimum. Thus, in the above example, rule set 5 is the minimum error one, but rule set 9 is significantly simpler but still within one standard error of the minimum. Hence rule set 9 is the one selected. The user can change the criterion by specifying a different threshold (different number of standar errors) in the properties file. If the system should always pick the one with the lowest error, then this threshold should be set to zero.

For real applications, the user may prefer another rule set to the one selected by RIKTEXT. Perhaps a simpler rule set appears more interesting. Perhaps the rules of the minimum rule set don't make much sense. Or perhaps solutions too far away from the covering rule set are not very useful in a scenario that involves voting multiple solutions. The summary table contains information about complexity and performance and enable the user to exercise appropriate judgement and select the rule set most appropriate for the objectives.

6

The Properties File

The file *riktext.properties* is used to further control the program. If this file is not present, the program will create one with default values. The user can edit this file to change the defaults. The file contains helpful comments to assist the user in changing the defaults. The defaults can always be regenerated by deleting the file and letting the program recreate it. The default riktext.properties is shown below:

```
# default options. this file created by riktext.
# comment lines have a '#' as the first character.
# other lines are of the type: option=value
#
# ftype (integer >=0) specifies frequency thresholding (value reduction).
# ftype=1 gives binary features (default),
# ftype=2 gives ternary features (values 0, 1 or 2),
# ftype=k gives features with values upto k (values>k reduced to k),
# ftype=0 means all frequencies used as given (no reduction).
#
ftype=1
#
# ttype (integer 1 or 2) specifies types of tests in rules.
# ttype=1 for only positive tests f>=n, where n>=1 (default),
# ttype=2 for allowing all kinds of tests.
#
ttype=1
#
# boost-recall (integer>=0) allows precision/recall tradeoff.
# boost-recall=0 gives equal weight to precision and recall (default)
# boost-recall>=1 increasingly favors recall over precision.
#
boost-recall=0
#
# se (real >=0) specifies how to define the "best" ruleset.
# se=f, "best" is the smallest within f std errors of min test-error ruleset.
# se=0, the "best" ruleset is the min test-error ruleset.
#
se=1
#
# short-rules=1 if quick short rules should be obtained.
# short-rules=0 if normal rules should be obtained (default).
#
short-rules=0
#
```

```
# maxrul (integer >1) specifies the maximum number of rules generated.
#
maxrul=5000
#
# optimization-threshold (real >=0) is for optimizing pruned rulesets.
# higher values imply less frequent optimization and
# the program will run faster, but may produce weaker results.
# specifying -1 computes the default value (ncases/200).
#
optimization-threshold=-1
#
```

6.1 Simplifying Feature Values

Interpretability of rules may be greatly enhanced by simplifying the range of values that features can take. For instance, is a value of 4 really different from a value of 3? What if we treat all values above 5 as 5? In the extreme, what if we simply treat all values greater than 1 as 1 (thereby making feature values binary – 0 or 1)? This aspect is controlled within the properties file by the property *ftype* which specifies a threshold. Values greater than the threshold are reduced to the threshold itself. For example if ftype is 3, any feature value of 4 or higher gets simplified to 3. The default is to simplify to binary features. But ternary features (that take on 3 possible values – 0, 1 or 2 – have been sometimes found to be useful. To switch off the simplification and use values as given, set ftype to 0.

Reducing feature values always tends to give more readily interpretable rules. A rule that involves a check to see if the word *property* occurs in the document, is far more intuitive than one that involves a check to see if the word occurs 6 times.

6.2 Positive and Negative Conjuncts

It is often more intuitive to have rules that check for presence of words, rather than their absence (its non-intuitive to have to check the entire document to make sure a word is absent, and then infer something from its absence). RIKTEXT allows users to specify what kind of conjuncts should appear in the rules by means of the property *ttype*. The default value is 1 which allows only positive conjuncts in the rules. But this can be changed to 2 (all kinds of conjuncts).

Allowing all kinds of conjuncts may result in smaller, more compact rule sets but these may not be necessarily more interpretable than those with only positive conjuncts.

6.3 Precision/Recall Tradeoff

RIKTEXT typically generates solutions that have a higher precision than recall. For document classification, sometimes we may want to improve the recall (at the expense of precision). The *boost-recall* property allows the user to favor solutions with higher

recall. The default value is 0 (both precision and recall treated equally), but higher values will increasingly favor recall over precision.

A consequence of boosting recall is that the rule sets will tend to become more complex. Boosting recall can also be very expensive computationally. Typically, it should be used only when the recall value is unacceptably lower than the precision value.

6.4 “Best” Rule Set

The best rule set is selected based a combination of complexity and error-rate considerations. We find the rule set with the minimum error-rate and then find a less complex rule set whose error-rate is reasonably close to this minimum error-rate.

The concept of “reasonably close” is governed by the property *se* which specifies the number of standard errors. By default, this is set to 1, so that “reasonably close” means “within one standard error”. The user can change this to any non-negative real number. If it is set to 0, the minimum error-rate rule-set is taken as the “best” rule set.

6.5 Short Rules

Rule length is controlled somewhat by the property *short-rules* property. RIKTEXT has an option to generate short rules very rapidly. These rules may not be the best in performance but give an indication of the sorts of solutions one may expect. To use this option, set this property to 1. The default value is 0, to generate normal rules.

6.6 Maximum Number of Rules

The property *maxrul* controls the maximum number of rules that RIKTEXT can generate. The default value is 5000 and it is not advisable to change this (especially not to a lower value).

6.7 Optimization Threshold

The rule optimization module of RIKTEXT can be controlled by the *optimization-threshold* property which can be set to any non-negative real number. Higher values result in less frequent optimization and RIKTEXT will run faster but may give weaker results. The default value is based on the number of cases. It is not advisable to change this property under normal circumstances.

7

An Extended Example

In this chapter we present an extended example of RIKTEXT in action. The purpose is to show how this software can be used iteratively to develop a rule-based classifier.

7.1 The Data

The data we use is the OHSUMED collection of abstracts gathered from MEDLINE. At least at the moment, the data can be downloaded from: ftp://medir.ohsu.edu/pub/ohsumed and probably other places (search the web).

The corpus that we use is actually an arbitrary selection from the total OHSUMED corpus. Since the OHSUMED collection is not in XML fromat, a special processing program was necessary to transform the data.

The program to convert the OHSUMED file to XML is shown below. Obviously it is not good for other data, but still, it illustrates how to go about the conversion.

```
/* ohsumed2XML                                  */
/** formats the OHSUMED files into XML          */
/*  java ohsumed2XML inputfile outputfile       */
/*                                              */
/* This program is specific to the OHSUMED      */
/* files but is an example of how to convert    */
/* other non-xml files                          */
/*                                              */
/* The OHSUMED files have an indicator tag      */
/* the data on the following line(s):           */
/*
/* .I 274314                                                    */
/* .U                                                           */
/* 91002386                                                     */
/* .S                                                           */
/* Br J Dermatol 9101; 123(3):365-73                            */
/* .M                                                           */
/* Adult; Female; Human; Male; Nails/AH/US; Ultrasonics; Water. */
/* .T                                                           */
/* Ultrasound velocity in human fingernail and effects of hydration:  */
/* validation of in vivo nail thickness measurement techniques. */
/* .P                                                           */
```

```
/* JOURNAL ARTICLE.                                                    */
/* .W                                                                  */
/* Distal nail thickness was measured using an electronic micrometer  */
/* and both distal and proximal nail ultrasound times were recorded    */
/*                        ......                                       */
/* .A                                                                  */
/* Finlay AY; Western B; Edwards C.                                    */
/*                                                                     */
/*                                          */
/* The XML version:                         */
/*                                          */
/* <DOC>                                                               */
/* <TITLE>                                                             */
/* Ultrasound velocity in human fingernail and effects of hydration:   */
/* validation of in vivo nail thickness measurement techniques.        */
/* </TITLE>                                                            */
/* <AUTHOR>                                                            */
/* Finlay AY; Western B; Edwards C.                                    */
/* </AUTHOR>                                                           */
/* <SUBJECTS>                                                          */
/* <SUBJECT>                                                           */
/* Adult                                                               */
/* </SUBJECT>                                                          */
/* <SUBJECT>                                                           */
/* Female                                                              */
/* </SUBJECT>                                                          */
/* <SUBJECT>                                                           */
/* Human                                                               */
/* </SUBJECT>                                                          */
/* <SUBJECT>                                                           */
/* Male                                                                */
/* </SUBJECT>                                                          */
/* <SUBJECT>                                                           */
/* Nails                                                               */
/* </SUBJECT>                                                          */
/* <SUBJECT>                                                           */
/* Ultrasonics                                                         */
/* </SUBJECT>                                                          */
/* <SUBJECT>                                                           */
/* Water                                                               */
/* </SUBJECT>                                                          */
/* </SUBJECTS>                                                         */
/* <TEXT>                                                              */
/* Distal nail thickness was measured using an electronic              */
/* micrometer and both distal and proximal nail ultrasound             */
/* times were recorded in 20 volunteers (10 male, 10 female),          */
/*          ...                                                        */
/* </TEXT>                                                             */
/* </DOC>                                                              */

import java.text.*;
import java.util.*;
import java.io.*;

class ohsu2XML {
public static void main(String[] args) {
```

```
   String line;
    int outlen = 72; // set output line length
    int linect;
    int posi;
    int marker;
    int ixent;
    boolean htmlflg;
    boolean textflg;
    boolean longflg = false;
   String str;

   String title;
   String body;
   String byline;
   String subject;
   String subjec;
   String source;
   StringBuffer text = new StringBuffer();

    try {
     if (args.length<2 || args.length>2)
  throw new tmskException("usage:\njava ohsumed2XML infile outfile");
     }
catch (tmskException e1) {System.out.println("mkdict: "+e1.getMessage());}
      BufferedReader in = null;
      PrintWriter out = null;
      PrintWriter pw = null;
    try {
      FileReader inpf = new FileReader(args[0]);
      // Create buffered reader
       in = new BufferedReader(inpf);
       out = new PrintWriter(System.out);

   } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
    try {
 pw = new PrintWriter(new FileWriter(args[1]));
 pw.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\" standalone=\"yes\"?>");
  pw.println("<CORPUS>");
  System.out.println("ready to process");
  str = "x"; // set for initial test below
  while (str != null){
       linect = 0;
        htmlflg = false;
      body = "";
      byline = "";
      subject = "";
      source = "";
      title = "";
      text.setLength(0);
       textflg = false;
      try {
      str = in.readLine(); // read first .I
      } catch (IOException e) {
 e.printStackTrace();
      }
```

```
     try {
 while ((str = in.readLine()) != null) {
       if (str.startsWith(".I")){
          break; //a new document starts here
         }
       if (str.startsWith(".U")) {
           continue;    // skip
          }
        else if (str.startsWith(".S")){
            source = in.readLine();  // source of doc
            continue;
            }
        else if (str.startsWith(".A")){
            byline = in.readLine();  // authors
            continue;
            }
        else if (str.startsWith(".T")){
            title = in.readLine(); // title
            continue;
            }
        else if (str.startsWith(".M")){
          // need to strip final . if there
            subject = in.readLine();  // the topics of the doc
           if (subject.endsWith(".")) {
             subject = subject.substring(0,subject.length() - 1);
            }
          subject = subject + ";";
            continue;
           }
        else if (str.startsWith(".W")){ // text
             while (!body.endsWith(".")) {
            body = body + in.readLine();
             }
            continue;
            }
        else  { // a skippable line
            continue;
          }
      }
     } catch (IOException e) {
e.printStackTrace();
     }
       if (body.length() == 0) continue;
  // now put saved data into XML file
        text.append(body);  // set stringbuffer
       int bodlen = body.length();
      // write out sgml
      pw.println("<DOC>");
      pw.println("<TITLE>");
      pw.println(title);
      pw.println("</TITLE>");
      pw.println("<AUTHOR>");
      pw.println(byline);
      pw.println("</AUTHOR>");
      pw.println("<SUBJECTS>");
 // iterate over the subjects, separated by semicolon
```

```
     StringTokenizer tksu = new StringTokenizer(subject,";");
      while (tksu.hasMoreTokens()){
        subjec = tksu.nextToken();
      // Increase frequency of subject by dropping suffix
       if (subjec.indexOf("/") > 0) {
         subjec = subjec.substring(0,subjec.indexOf("/"));
        }
         subjec = subjec.trim();
         subjec = subjec.replace(' ','_');
       pw.println("<SUBJECT>");
       pw.println(subjec);
       pw.println("</SUBJECT>");
      }
       pw.println("</SUBJECTS>");
       pw.println("<TEXT>");
   // instead of one long line, print text in short lines
      int nech = 0;
      int sch = 0; // starting point
      int ech = 60; // arbitrary ending point
    while(ech < bodlen) {
        nech = ech;
       for (int itx = ech; itx > sch; itx--){
        if (!(text.charAt(itx) == ' ')) {
           nech = itx;
        }
       else break;
       }
       ech = nech - 1;
       line = text.substring(sch,ech);
       pw.println(line);
       sch = nech;
       ech = sch + 60;
     }
       pw.println("</TEXT>");
       pw.println("</DOC>");
   }
   } catch (IOException e) {
      e.printStackTrace();
    }


    try {
       pw.println("</CORPUS>");
      out.close();
      in.close();
      pw.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
 }
}
```

Once the data is in XML format, it must be processed by TMSK to generate a dictionary and a set of labeled vectors. A dictionary of 1000 words was generated. After removing stopwords, we were left with 866 words. These were then used to generate vectors. We restrict ourselves to the single category *Diet* and all examples

were classified as either belonging to this category or not. The relevant attributes in the tmsk.properties file were:

```
doctag=DOC
bodytags=TITLE TEXT
labeltag=SUBJECTS
stopwords=stopwords.list
sentence-delimiters="\"<"
```

The vectors were randomly divided into two data sets, one used for training and one for validation. The training set had 11,560 cases and the test set had 3,157 cases. The training cases were put in the file *ohsumedxr.vec* and the validation cases in the file *ohsumedxt.vec*. The dictonary was in the file *ohsumed.dic*.

7.2 Initial Experiments

The first pass on the data is on the training set, using the validation data for model selection and the default riktext.properties:

```
% riktext -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul
riktext: automatically generating default riktext.properties file

Table of pruned rule sets
(* = minimum error; ** = within 1-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      81     220     0.0000    0.0260   0.0028      0.0      0.00
    2      61     159     0.0017    0.0225   0.0026      0.0      0.33
    3      59     153     0.0019    0.0225   0.0026      0.0      0.33
    4      41     115     0.0035    0.0184   0.0024      0.0      0.50
    5      34      94     0.0047    0.0174   0.0023      0.0      0.62
    6      28      75     0.0057    0.0177   0.0023      0.0      0.63
    7      23      60     0.0066    0.0177   0.0023      0.0      0.67
    8      22      56     0.0068    0.0174   0.0023      0.0      0.75
    9      12      27     0.0087    0.0181   0.0024      0.0      1.00
   10      11      24     0.0090    0.0187   0.0024      0.0      1.00
   11      10      21     0.0093    0.0190   0.0024      0.0      1.33
   12       8      17     0.0099    0.0187   0.0024      0.0      1.50
   13       7      15     0.0102    0.0190   0.0024      0.0      2.00
   14       5       9     0.0113    0.0165   0.0023      0.0      2.17
   15       3       5     0.0125    0.0155   0.0022      0.0      3.50
   16**     2       2     0.0149    0.0152   0.0022      0.0      9.00
   17       1       1     0.0174    0.0200   0.0025      0.0     29.00
Alternate database test cases, 3157 test cases

*********************************************************************
Selected rule set

    1. dietary --> Diet
    2. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 57.8378      recall: 53.2338         f-measure: 55.4404
```

```
Additional Statistics (Test Cases):
precision: 62.2951      recall: 60.3175         f-measure: 61.2903
```

The result with the default of binary features is not as good as one would like. The precision/recall are matched but the f-measure is quite low. Let us try again using ternary features. To do this, first we must edit riktext.properties and set *ftype=2*. Let us also set *se=0* so that the program always selects the best solution. Then, running riktext again gives:

```
% riktext -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 0-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      72     168     0.0000    0.0234   0.0027      0.0      0.00
    2      69     159     0.0003    0.0234   0.0027      0.0      0.33
    3      38      97     0.0029    0.0200   0.0025      0.0      0.50
    4      29      70     0.0045    0.0200   0.0025      0.0      0.67
    5      28      66     0.0048    0.0200   0.0025      0.0      0.75
    6      22      51     0.0060    0.0168   0.0023      0.0      0.93
    7      16      34     0.0074    0.0162   0.0022      0.0      1.00
    8      15      31     0.0078    0.0162   0.0022      0.0      1.33
    9      13      26     0.0084    0.0165   0.0023      0.0      1.40
   10      10      20     0.0087    0.0177   0.0023      0.0      1.50
   11       8      16     0.0092    0.0168   0.0023      0.0      1.75
   12       3       4     0.0119    0.0130   0.0020      0.0      3.17
   13**     2       2     0.0137    0.0130   0.0020      0.0     11.50
   14       1       1     0.0174    0.0200   0.0025      0.0     43.00
Alternate database test cases, 3157 test cases

**********************************************************************
Selected rule set

    1. dietary>=2 --> Diet
    2. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 77.9221      recall: 29.8507         f-measure: 43.1655

Additional Statistics (Test Cases):
precision: 86.6667      recall: 41.2698         f-measure: 55.9140
```

The precision is up quite a bit, but recall is down. Still, the difference between the two gives room to play. But first, lets try with no value reduction and using the full document frequencies for feature values:

```
% riktext -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 0-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      77     167     0.0000    0.0200   0.0025      0.0      0.00
    2      75     161     0.0003    0.0200   0.0025      0.0      0.67
```

```
    3      40      88     0.0063    0.0184   0.0024      0.0      0.97
    4      39      83     0.0065    0.0190   0.0024      0.0      1.00
    5      34      70     0.0081    0.0193   0.0024      0.0      1.38
    6      32      66     0.0088    0.0177   0.0023      0.0      2.00
    7      11      20     0.0144    0.0152   0.0022      0.0      1.87
    8       8      14     0.0158    0.0146   0.0021      0.0      2.83
    9       5       6     0.0174    0.0133   0.0020      0.0      2.62
   10**     4       4     0.0185    0.0133   0.0020      0.0      6.50
   11       3       3     0.0197    0.0139   0.0021      0.0     22.00
   12       2       2     0.0226    0.0152   0.0022      0.0     34.00
   13       1       1     0.0342    0.0200   0.0025      0.0    136.00
Alternate database test cases, 3157 test cases

*********************************************************************
Selected rule set

    1. diet>=2 --> Diet
    2. intake>=4 --> Diet
    3. dietary>=2 --> Diet
    4. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 68.7861      recall: 59.2040         f-measure: 63.6364

Additional Statistics (Test Cases):
precision: 71.4286      recall: 55.5556         f-measure: 62.5000
```

This is the best result so far. Still, there is some scope for tuning the program further.

7.3 Further Tuning

Lets fix on *ftype=2* (ternary feature values) and tweak the other parameters. We first try to use negative clauses as well as positive ones. To do this, we set *ttype=2* and run riktext:

```
% riktext -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 0-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      47     146     0.0000    0.0222   0.0026      0.0      0.00
    2      44     137     0.0003    0.0212   0.0026      0.0      0.33
    3      18      85     0.0025    0.0171   0.0023      0.0      0.50
    4      12      67     0.0035    0.0171   0.0023      0.0      0.67
    5      11      63     0.0038    0.0174   0.0023      0.0      0.75
    6       7      49     0.0048    0.0168   0.0023      0.0      0.93
    7       6      33     0.0061    0.0165   0.0023      0.0      1.00
    8       5      26     0.0071    0.0155   0.0022      0.0      1.71
    9       5      25     0.0073    0.0155   0.0022      0.0      2.00
   10       4      12     0.0095    0.0146   0.0021      0.0      2.08
   11       4      11     0.0098    0.0146   0.0021      0.0      3.00
   12       3       5     0.0113    0.0133   0.0020      0.0      3.00
   13       3       4     0.0118    0.0120   0.0019      0.0      5.00
```

```
   14**     3       3     0.0124    0.0120   0.0019      0.0      7.00
   15       2       2     0.0137    0.0130   0.0020      0.0     15.00
   16       1       1     0.0174    0.0200   0.0025      0.0     43.00
Alternate database test cases, 3157 test cases

**********************************************************************
Selected rule set

    1. dietary>=2 --> Diet
    2. diet>=2 --> Diet
    3. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 69.3333      recall: 51.7413         f-measure: 59.2593

Additional Statistics (Test Cases):
precision: 79.0698      recall: 53.9683         f-measure: 64.1509
```

Results are slightly better. Since the final model is short, it is worthwhile to try the short-rule option (*short-rules=1*):

```
% riktext -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 0-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      80     172     0.0000    0.0212   0.0026      0.0      0.00
    2      77     163     0.0003    0.0206   0.0025      0.0      0.33
    3      26      61     0.0047    0.0155   0.0022      0.0      0.50
    4      17      39     0.0065    0.0158   0.0022      0.0      0.95
    5      17      38     0.0066    0.0158   0.0022      0.0      1.00
    6      15      34     0.0069    0.0165   0.0023      0.0      1.50
    7       6      11     0.0106    0.0152   0.0022      0.0      2.04
    8       5       8     0.0110    0.0136   0.0021      0.0      2.00
    9**     3       3     0.0124    0.0120   0.0019      0.0      3.20
   10       2       2     0.0137    0.0130   0.0020      0.0     15.00
   11       1       1     0.0174    0.0200   0.0025      0.0     43.00
Alternate database test cases, 3157 test cases

**********************************************************************
Selected rule set

    1. diet>=2 --> Diet
    2. dietary>=2 --> Diet
    3. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 69.3333      recall: 51.7413         f-measure: 59.2593

Additional Statistics (Test Cases):
precision: 79.0698      recall: 53.9683         f-measure: 64.1509
```

While the rule set selected is identical to the one with normal rules, notice that the summary table consists different rulesets. Let us examine the next largest rule set instead (at number 8):

```
% riktext -s 8 -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 0-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      80     172     0.0000    0.0212   0.0026      0.0      0.00
    2      77     163     0.0003    0.0206   0.0025      0.0      0.33
    3      26      61     0.0047    0.0155   0.0022      0.0      0.50
    4      17      39     0.0065    0.0158   0.0022      0.0      0.95
    5      17      38     0.0066    0.0158   0.0022      0.0      1.00
    6      15      34     0.0069    0.0165   0.0023      0.0      1.50
    7       6      11     0.0106    0.0152   0.0022      0.0      2.04
    8       5       8     0.0110    0.0136   0.0021      0.0      2.00
    9**     3       3     0.0124    0.0120   0.0019      0.0      3.20
   10       2       2     0.0137    0.0130   0.0020      0.0     15.00
   11       1       1     0.0174    0.0200   0.0025      0.0     43.00
Alternate database test cases, 3157 test cases

**********************************************************************
Selected rule set

    1. diet>=2 --> Diet
    2. dietary>=2 & patient<=1 --> Diet
    3. intake>=2 & datum>=1 --> Diet
    4. dietary>=1 & similar>=1 --> Diet
    5. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 72.5610      recall: 59.2040         f-measure: 65.2055

Additional Statistics (Test Cases):
precision: 69.2308      recall: 57.1429         f-measure: 62.6087
```

The second rule has a negative clause, but the performance of the rule set is weaker. So we switch back to normal rules (*short-rules=0*). So far, the best result seems to be with *ftype=2* and *ttype=2* and the main problem with this is that the recall is much lower than the precision.

7.4 Boosting Recall

The lower recall suggests that we might be able to improve performance by tuning the boost-recall attribute. So, we set *boost-recall=1* and run riktext:

```
% riktext -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 0-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      42     171     0.0000    0.0244   0.0027      0.0      0.00
    2      40     165     0.0003    0.0238   0.0027      0.0      0.67
    3      20     113     0.0040    0.0209   0.0025      0.0      0.96
    4      20      91     0.0056    0.0209   0.0025      0.0      1.00
```

```
    5      18      78     0.0064    0.0222   0.0026      0.0      1.00
    6      11      47     0.0103    0.0206   0.0025      0.0      1.52
    7**     6      13     0.0161    0.0136   0.0021      0.0      2.12
    8       5       8     0.0176    0.0165   0.0023      0.0      3.80
    9       3       3     0.0197    0.0139   0.0021      0.0      6.00
   10       2       2     0.0226    0.0152   0.0022      0.0     34.00
   11       1       1     0.0342    0.0200   0.0025      0.0    136.00
Alternate database test cases, 3157 test cases

*********************************************************************
Selected rule set

    1. dietary>=1 & intake>=2 --> Diet
    2. dietary>=1 & diet>=1 --> Diet
    3. diet>=2 --> Diet
    4. dietary>=1 & animal<=0 & report<=0 & effect>=1 --> Diet
    5. intake>=2 & .>=1 & compare<=0 --> Diet
    6. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 70.3125      recall: 67.1642         f-measure: 68.7023

Additional Statistics (Test Cases):
precision: 67.2414      recall: 61.9048         f-measure: 64.4628
```

The recall is certainly higher now (at the cost of precision), and the f-measure is slightly improved. But note that the rule set is more complex. Some of the rules have negative clauses. Rule 5 involves checking for number of periods in each document.

Let us try switching back to positive clauses only, while keeping all the other settings. So we set *ttype=1* and get:

```
% riktext  -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 0-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      71     167     0.0000    0.0203   0.0025      0.0      0.00
    2      68     158     0.0005    0.0203   0.0025      0.0      0.67
    3      41      99     0.0054    0.0181   0.0024      0.0      0.97
    4      37      86     0.0064    0.0184   0.0024      0.0      1.00
    5      31      67     0.0083    0.0177   0.0023      0.0      1.26
    6      27      59     0.0094    0.0174   0.0023      0.0      1.62
    7      19      39     0.0120    0.0158   0.0022      0.0      1.70
    8       6       8     0.0179    0.0149   0.0022      0.0      2.42
    9       5       6     0.0181    0.0149   0.0022      0.0      1.50
   10       4       4     0.0190    0.0139   0.0021      0.0      5.50
   11**     3       3     0.0197    0.0139   0.0021      0.0     13.00
   12       2       2     0.0226    0.0152   0.0022      0.0     34.00
   13       1       1     0.0342    0.0200   0.0025      0.0    136.00
Alternate database test cases, 3157 test cases

*********************************************************************
Selected rule set

    1. diet>=2 --> Diet
```

```
    2. dietary>=1 --> Diet
    3. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 58.0786      recall: 66.1692         f-measure: 61.8605

Additional Statistics (Test Cases):
precision: 63.0137      recall: 73.0159         f-measure: 67.6471
```

Results have certainly improved and the rule set is also much simpler. Let us try with *ftype=0* which uses the full document frequencies:

```
% riktext  -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 0-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      77     167     0.0000    0.0200   0.0025      0.0      0.00
    2      75     161     0.0003    0.0200   0.0025      0.0      0.67
    3      40      88     0.0063    0.0184   0.0024      0.0      0.97
    4      39      83     0.0065    0.0190   0.0024      0.0      1.00
    5      34      70     0.0081    0.0193   0.0024      0.0      1.38
    6      32      66     0.0088    0.0177   0.0023      0.0      2.00
    7      11      20     0.0144    0.0152   0.0022      0.0      1.87
    8       8      14     0.0158    0.0146   0.0021      0.0      2.83
    9       5       6     0.0174    0.0133   0.0020      0.0      2.62
   10**     4       4     0.0185    0.0133   0.0020      0.0      6.50
   11       3       3     0.0197    0.0139   0.0021      0.0     22.00
   12       2       2     0.0226    0.0152   0.0022      0.0     34.00
   13       1       1     0.0342    0.0200   0.0025      0.0    136.00
Alternate database test cases, 3157 test cases

**********************************************************************
Selected rule set

    1. diet>=2 --> Diet
    2. intake>=4 --> Diet
    3. dietary>=2 --> Diet
    4. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 68.7861      recall: 59.2040         f-measure: 63.6364

Additional Statistics (Test Cases):
precision: 71.4286      recall: 55.5556         f-measure: 62.5000
```

Results are weaker, but recall is lower than precision, permitting a further tweak of boost-recall. We set *boost-recall=2* and try again:

```
% riktext  -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 0-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
```

```
    1      73     160     0.0000    0.0193   0.0024      0.0      0.00
    2      72     151     0.0007    0.0193   0.0024      0.0      1.00
    3      42      87     0.0080    0.0158   0.0022      0.0      1.44
    4      30      58     0.0118    0.0174   0.0023      0.0      1.97
    5      29      54     0.0123    0.0168   0.0023      0.0      1.75
    6       9      15     0.0201    0.0133   0.0020      0.0      2.54
    7       8      13     0.0206    0.0143   0.0021      0.0      4.00
    8       7      11     0.0211    0.0139   0.0021      0.0      6.00
    9**     6       8     0.0218    0.0120   0.0019      0.0      6.33
   10       5       6     0.0227    0.0149   0.0022      0.0      7.50
   11       4       4     0.0236    0.0149   0.0022      0.0     11.00
   12       3       3     0.0251    0.0139   0.0021      0.0     18.00
   13       2       2     0.0301    0.0152   0.0022      0.0     60.00
   14       1       1     0.0504    0.0200   0.0025      0.0    243.00
Alternate database test cases, 3157 test cases

*********************************************************************
Selected rule set

    1. diet>=2 --> Diet
    2. dietary>=2 --> Diet
    3. intake>=4 --> Diet
    4. dietary>=1 & increase>=1 --> Diet
    5. intake>=1 & dietary>=1 --> Diet
    6. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 64.9533      recall: 69.1542         f-measure: 66.9880

Additional Statistics (Test Cases):
precision: 68.1159      recall: 74.6032         f-measure: 71.2121
```

This is the best result we have seen so far. Recall is now better than precision, so nothing is to be gained by boosting it further. However, the rule set is a bit more complex and one might wonder how a simpler rule set within one standard error of the minimum might do. So we set *se=1* and rerun:

```
% riktext -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 1-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      73     160     0.0000    0.0193   0.0024      0.0      0.00
    2      72     151     0.0007    0.0193   0.0024      0.0      1.00
    3      42      87     0.0080    0.0158   0.0022      0.0      1.44
    4      30      58     0.0118    0.0174   0.0023      0.0      1.97
    5      29      54     0.0123    0.0168   0.0023      0.0      1.75
    6       9      15     0.0201    0.0133   0.0020      0.0      2.54
    7       8      13     0.0206    0.0143   0.0021      0.0      4.00
    8       7      11     0.0211    0.0139   0.0021      0.0      6.00
    9*      6       8     0.0218    0.0120   0.0019      0.0      6.33
   10       5       6     0.0227    0.0149   0.0022      0.0      7.50
   11       4       4     0.0236    0.0149   0.0022      0.0     11.00
   12**     3       3     0.0251    0.0139   0.0021      0.0     18.00
   13       2       2     0.0301    0.0152   0.0022      0.0     60.00
```

```
   14       1       1     0.0504    0.0200   0.0025      0.0    243.00
Alternate database test cases, 3157 test cases

**********************************************************************
Selected rule set

    1. diet>=2 --> Diet
    2. dietary>=1 --> Diet
    3. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 58.0786      recall: 66.1692         f-measure: 61.8605

Additional Statistics (Test Cases):
precision: 63.0137      recall: 73.0159         f-measure: 67.6471
```

This is identical to what we saw earlier. We could see what happens if only short-rules are generated *short-rules=1* (we also switch to selecting the minimum error solution with *se=0*):

```
% riktext -t ohsumedxt.vec ohsumed.dic Diet ohsumedxr.vec > diet.rul

Table of pruned rule sets
(* = minimum error; ** = within 0-SE of minimum error)

 RSet   Rules    Vars  Train Err  Test Err  Test SD  MeanVar   Err/Var
    1      85     172     0.0000    0.0196   0.0025      0.0      0.00
    2      84     165     0.0005    0.0200   0.0025      0.0      1.00
    3      59     115     0.0065    0.0181   0.0024      0.0      1.46
    4      27      46     0.0134    0.0171   0.0023      0.0      1.48
    5      26      43     0.0138    0.0184   0.0024      0.0      1.67
    6      25      41     0.0142    0.0171   0.0023      0.0      3.00
    7       8      12     0.0202    0.0168   0.0023      0.0      2.97
    8       5       6     0.0227    0.0149   0.0022      0.0      5.50
    9       4       4     0.0236    0.0149   0.0022      0.0     11.00
   10**     3       3     0.0251    0.0139   0.0021      0.0     18.00
   11       2       2     0.0301    0.0152   0.0022      0.0     60.00
   12       1       1     0.0504    0.0200   0.0025      0.0    243.00
Alternate database test cases, 3157 test cases

**********************************************************************
Selected rule set

    1. diet>=2 --> Diet
    2. dietary>=1 --> Diet
    3. [TRUE] --> ~Diet

Additional Statistics (Training Cases):
precision: 58.0786      recall: 66.1692         f-measure: 61.8605

Additional Statistics (Test Cases):
precision: 63.0137      recall: 73.0159         f-measure: 67.6471
```

While the summary table differs, the selected rule set is still the same – slightly weaker than the best rule set we have seen so far.

7.5 Conclusion

Our best result (as measured by f-measure) comes from using full frequency feature values, boosting the recall, and selecting the minimum-error rule set. A slightly weaker but substantially simpler solution can be obtained by selecting the simplest solution within one standard error of this minimum-error rule set. While this is by no means the best solution for this data, it illustrates the process involved in iteratively trying different parameter values and examining the rule sets obtained and their error-estimates.