Text Mining in Knime

P. Brazdil[1], P.Kosina[2], J.Gama[1]

[1] LIAAD INESC TEC, FEP, Univ. of Porto

[1] LIAAD INESC TEC, FI-Masaryk Univ., Brno

Jan. 2013

Overview

1. Introduction
2. Loading files and creating corpus
3. Preprocessing
4. Creating Document-Term matrix
5. Creating Train / Test set
6. Training classifiers and classifying
7. Feature elimination and retraining

1. Introduction

KNIME

- Professional open-source data mining software developed at Univ. of Konstanz
- User-friendly environment for creating workflows
- Workflows can be saved / retrieved
- www.knime.org

Text mining tools

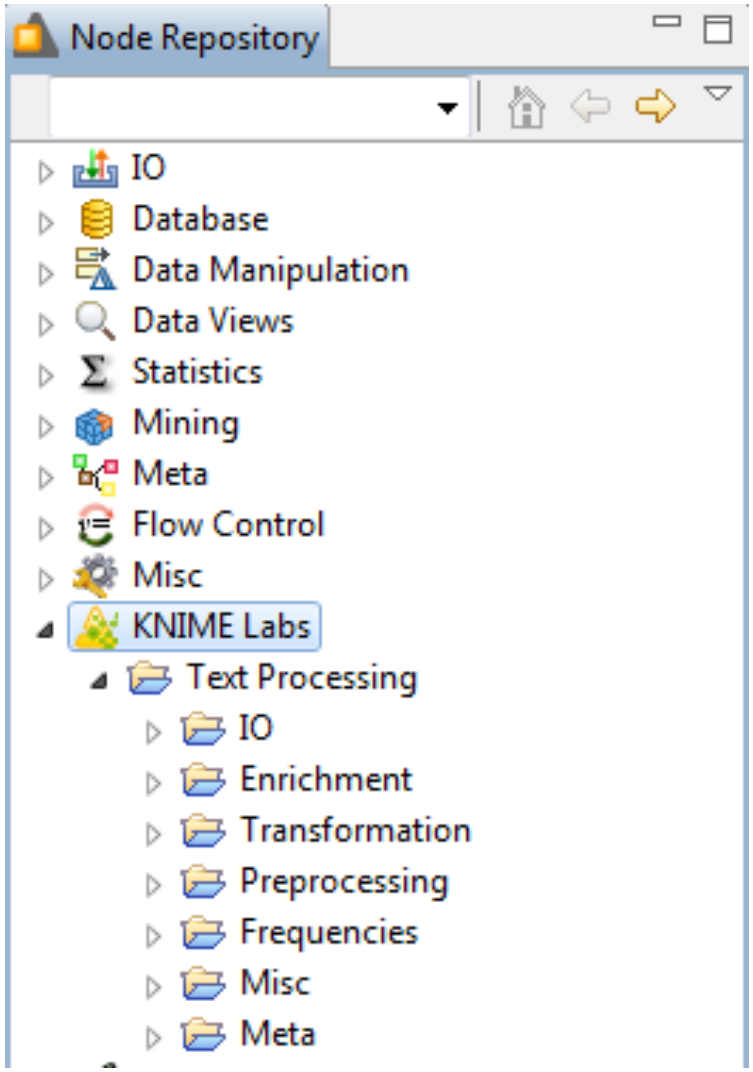
- Not part of common distribution
- Available under KNIME Labs
- It can be downloaded by evoking Install KNIME extensions

References:

- K. Thiel: The KNIME text processing plug-in

1. Introduction

Node Repository under KNIME Labs contains many tools for text processing. Explore, try out, play!



2. Loading Files and Creating Corpus

Knime Labs -> Text processing -> IO

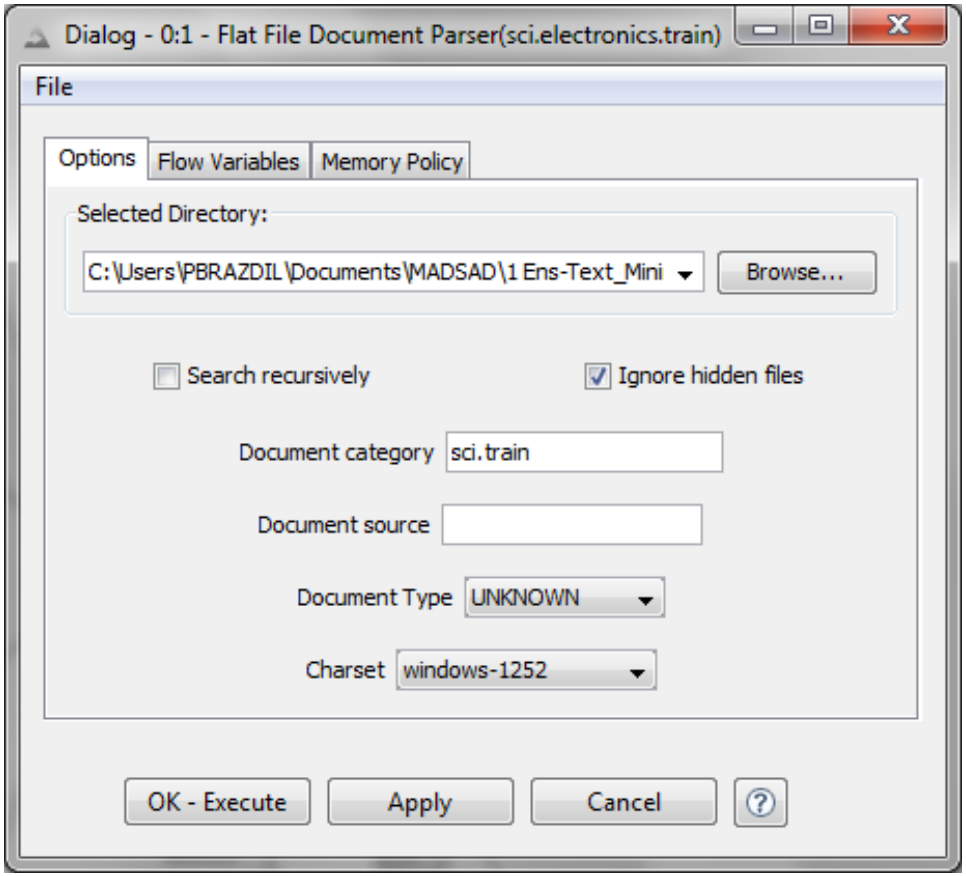
Flat File Document Reader

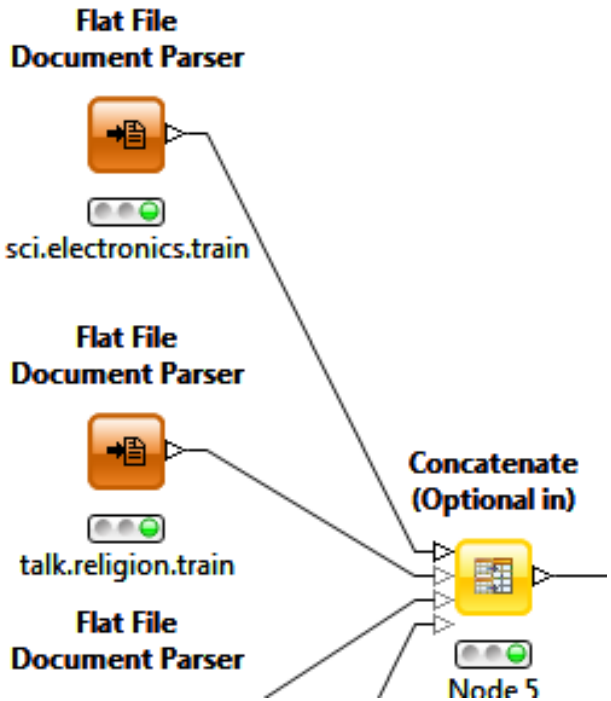
- The specified directory will be searched for all files.
- Reads flat text files and creates a document for each file.
- The documents title will be the first sentence of the file and the full text the remaining text contained in the file.
- The “document category” (corresponding to a class) can be set in a configure window

Concatenate (in Data Manipulation -> Row)

- Documents from different directories can be read-in separately and concatenated

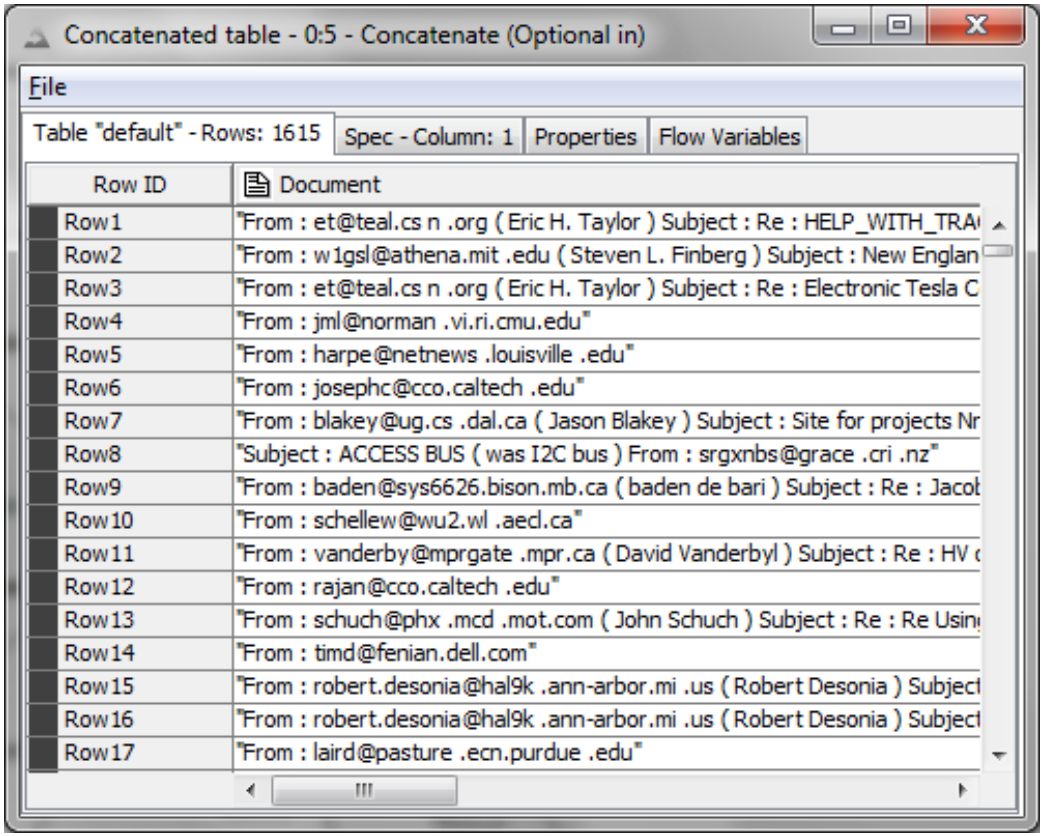
Loading Files and Creating Corpus





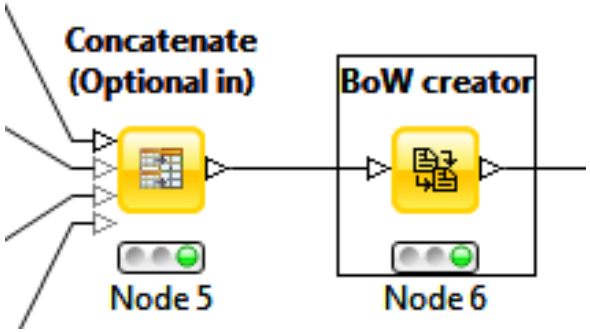
Result of Concatenation

Result of concatenation



Creating BoW Representation

The process of creating a corpus is finished by creating the bag of words:



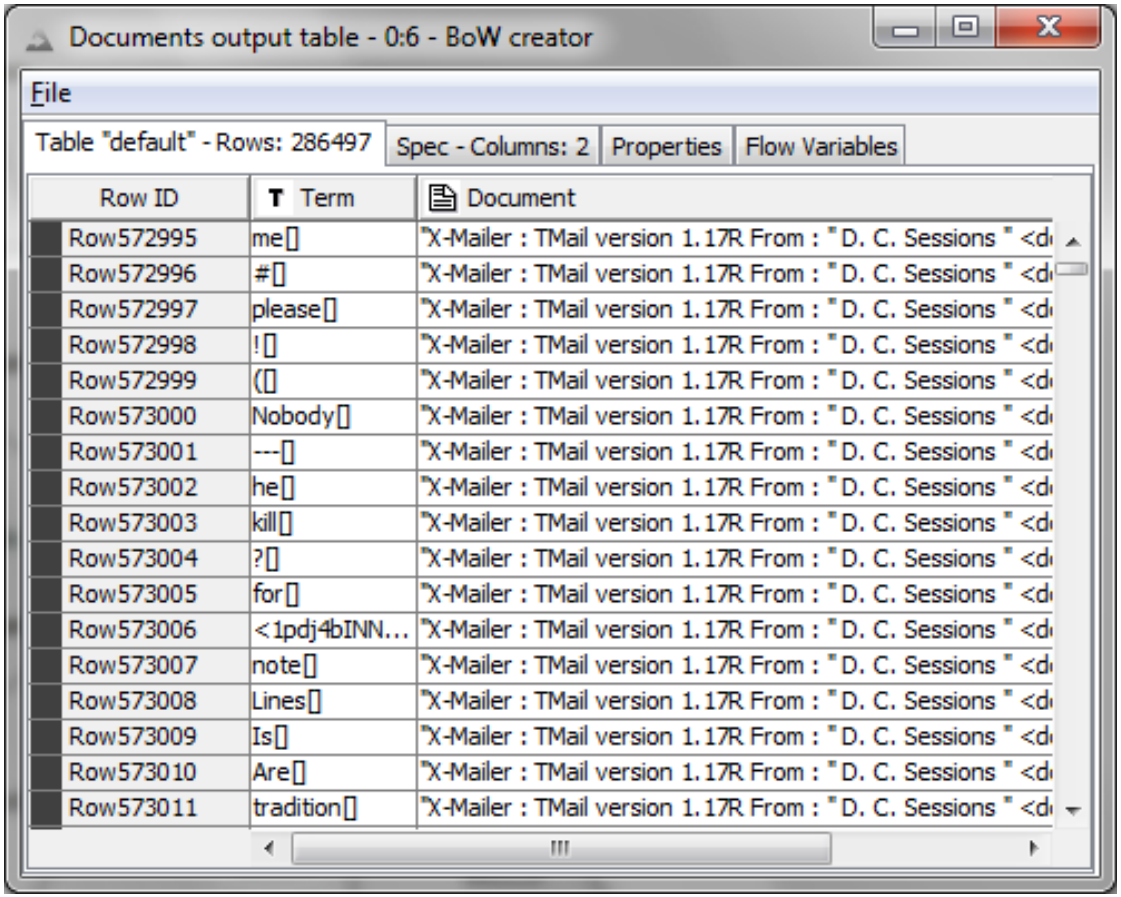
A BoW consists of at least two columns:

- one column containing the terms occurring in the corresponding document,
- the other column containing the documents.

A row represents a term contained in the related document.

BoW Representation

The bag of words representation:



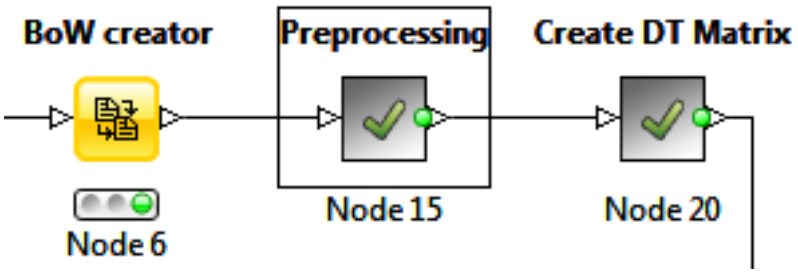
Row ID	Term	Document
Row572995	me[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row572996	#[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row572997	please[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row572998	![]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row572999	([]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573000	Nobody[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573001	---[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573002	he[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573003	kill[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573004	?[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573005	for[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573006	<1pdj4bINN...	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573007	note[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573008	Lines[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573009	Is[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573010	Are[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d
Row573011	tradition[]	"X-Mailer : TMail version 1.17R From : "D. C. Sessions " <d

3. Preprocessing

Involves a series of steps:

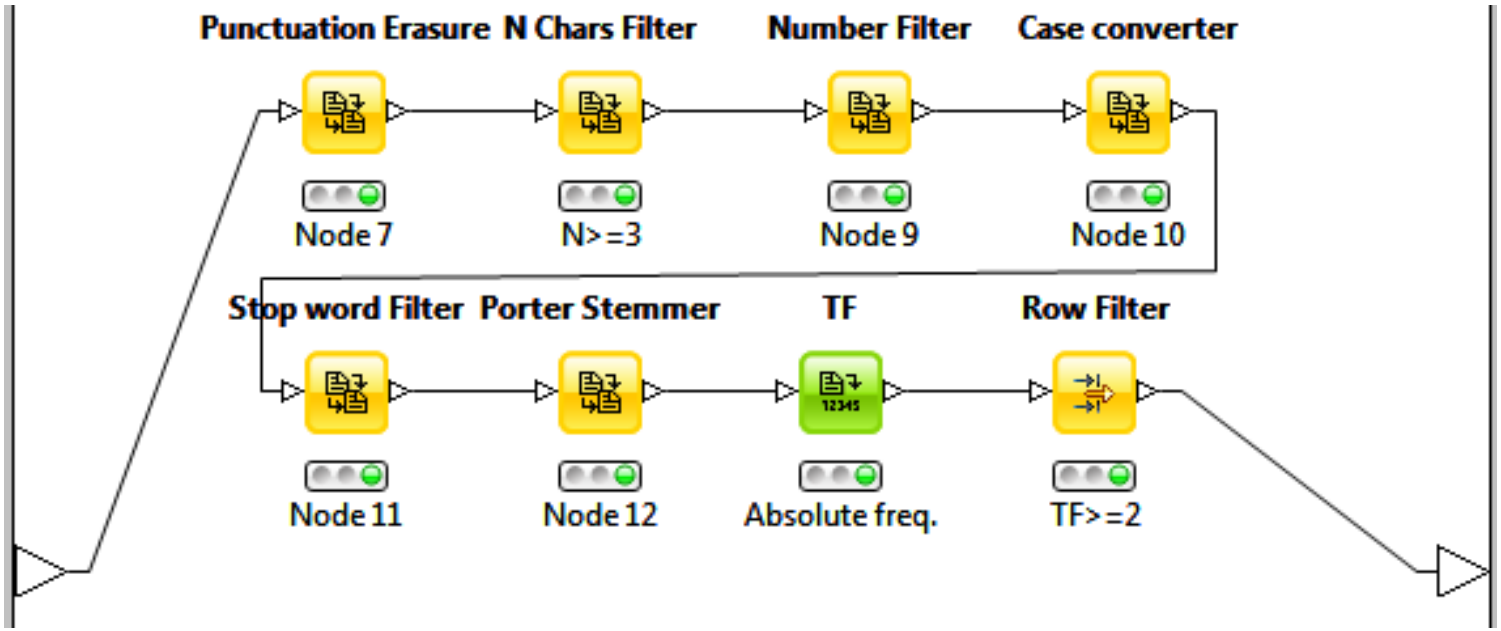
- Removing punctuation
- Removing short words (length 1; could be 2 or more)
- Removing numbers
- Conversion to lower case
- Eliminating stop words (in a given language)
- Stemming
- Adding frequency counts (TF) and using these to eliminate low-frequency words (1 occurrence)

Metanode “Preprocessing” contains these steps.



3. Preprocessing

Contents of Metanode “Preprocessing”



4. Creation of DT Matrix

Creation of Document-Term matrix involves the following steps:

- Creation of document vector

  One vector is created for each document.
  The values of the vectors are term frequencies (TFs) or Tf-Idf values
  The dimension of the vector is the number of distinct terms in BoW

- Category to class conversion

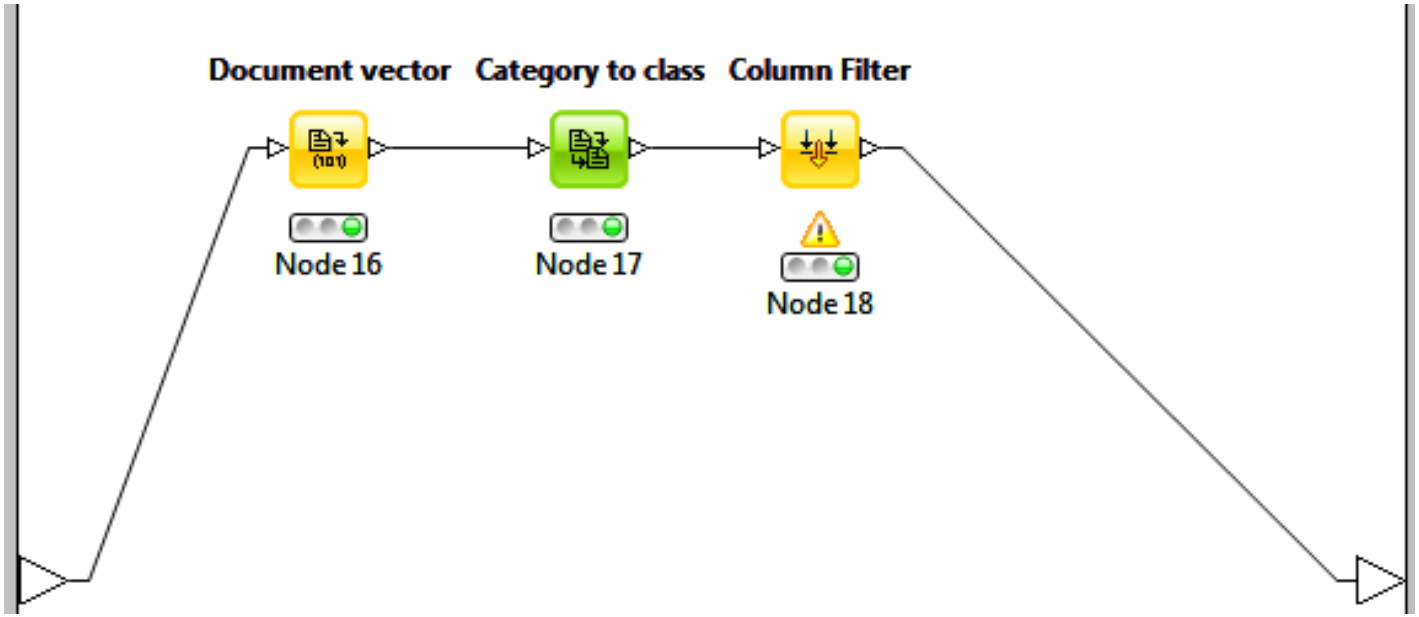
  The category “sci.train” is transformed to class value.

- Column filter

  Maintains all terms, but drops the document.

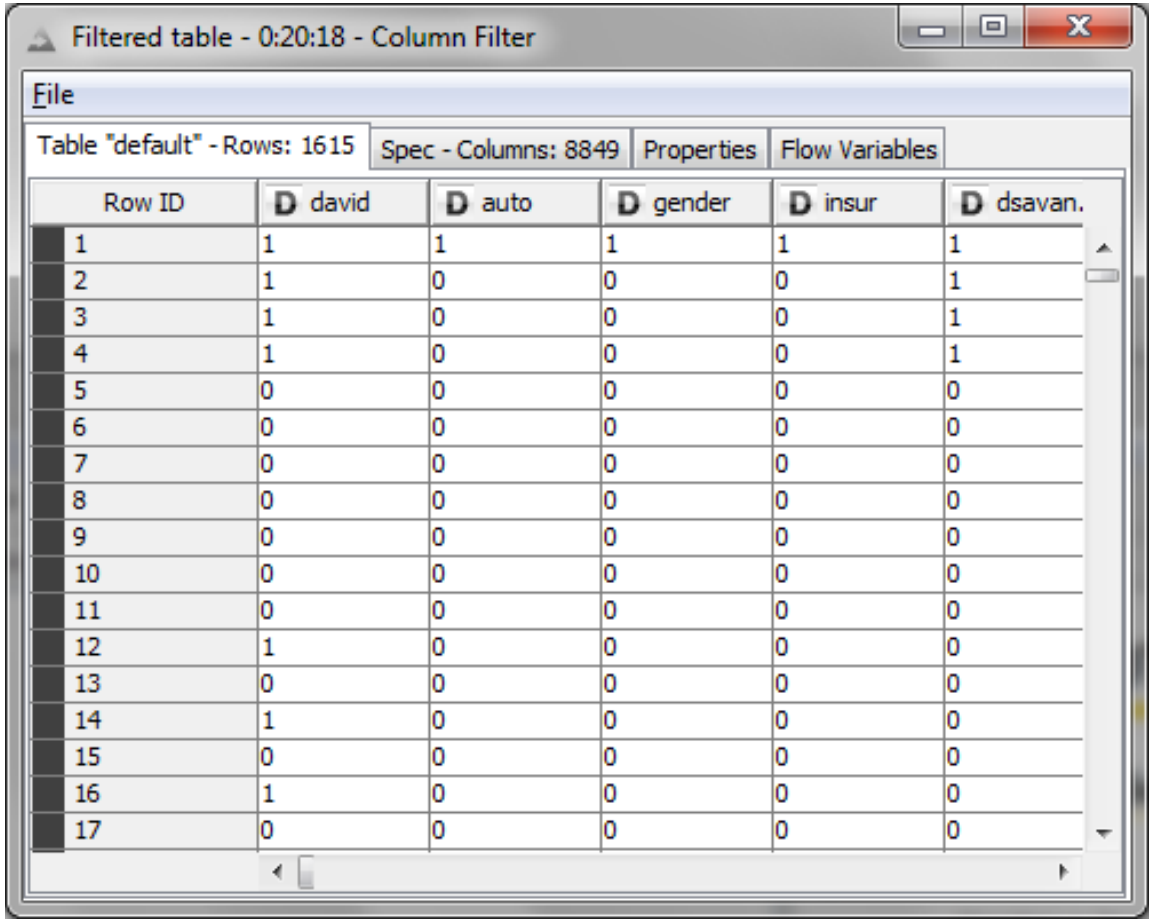
4. Creation of DT Matrix

Create DT matrix:



4. Creation of DT Matrix

Resulting DT matrix (1615 docs, 8849 terms):


Filtered table - 0:20:18 - Column Filter

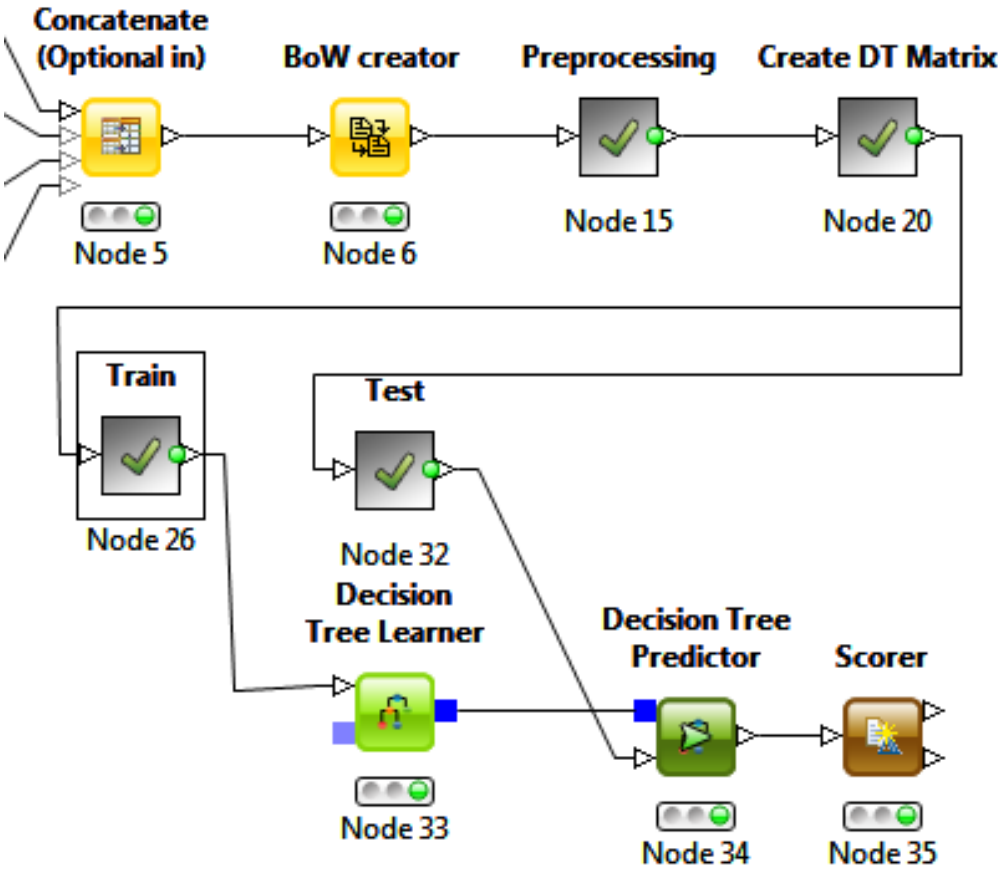
File

Table "default" - Rows: 1615 | Spec - Columns: 8849 | Properties | Flow Variables

Row ID	david	auto	gender	insur	dsavan.
1	1	1	1	1	1
2	1	0	0	0	1
3	1	0	0	0	1
4	1	0	0	0	1
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0
10	0	0	0	0	0
11	0	0	0	0	0
12	1	0	0	0	0
13	0	0	0	0	0
14	1	0	0	0	0
15	0	0	0	0	0
16	1	0	0	0	0
17	0	0	0	0	0

5. Creation of Train / Test Set

Use the output of DT matrix to create train / test set:



Creation of Train Set

Create the train set (metanode Train):

- Use Row Filter to select documents of the relevant class (sci.train, rel.train).

  Select the column to test: Document class
  Pattern: sci.train

- Use String Replacer to rename the class value sci.train -> sci and rel.train -> rel

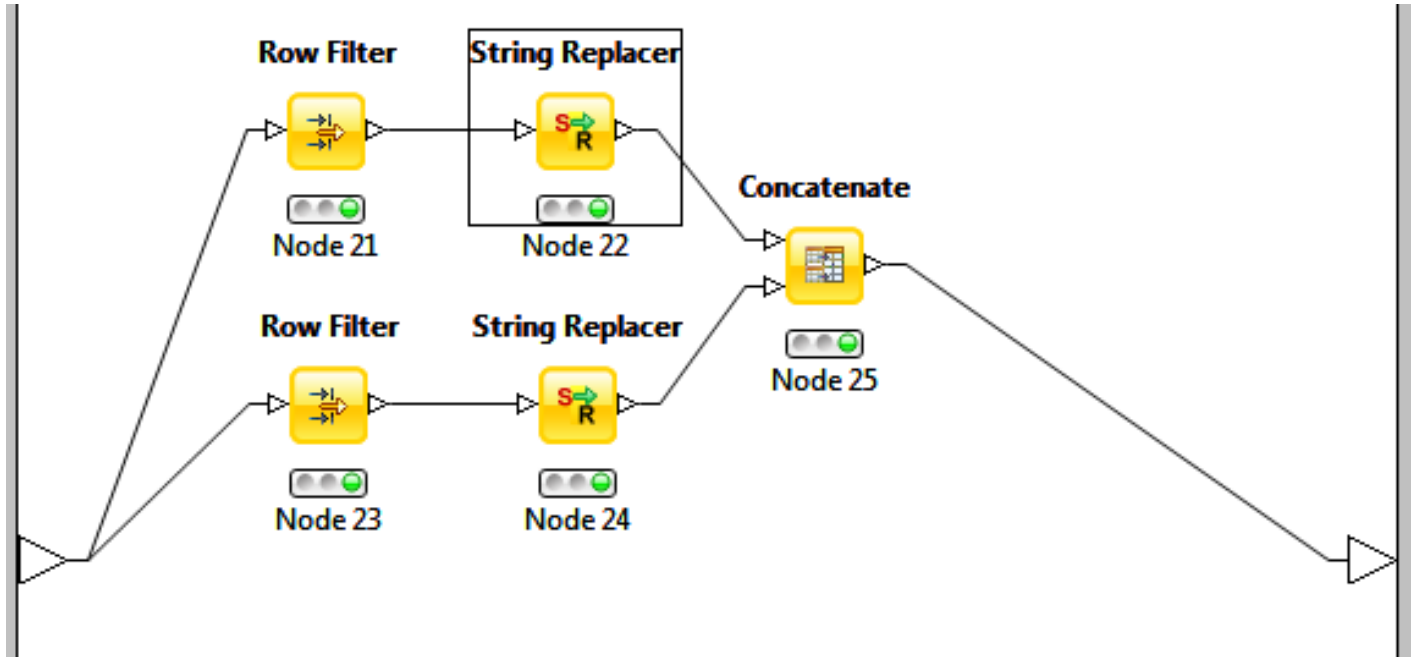
  Target column: Document class
  Wildcard pattern: sci.train
  Replacement text: sci

- Concatenate documents of sci and rel.

Use a similar process to create the test set.

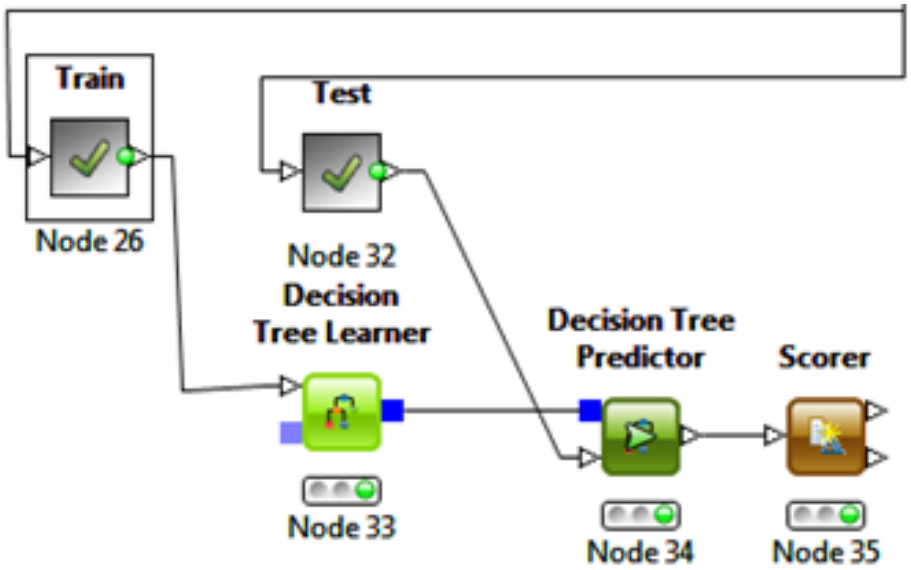
Creation of Train Set

Metanode Train:



6. Training a Classifier & Classifying Test Set

Here we show how to use a Decision Tree:



Confusion matrix:

	sci^	rel^
sci	362	34
rel	61	190

The success rate (calculated by Scorer) was 85.3%.

Training a Classifier & Classifying Test Set

We note that training a decision tree requires several minutes.
This is due to the fact that the number of terms is rather high (>8000).

The training can be speeded up by applying feature selection.

Other types of classifiers could be used (SVM, kNN, NB etc.).

As the number of features is high,
it is not feasible to use some of these classifiers (e.g. kNN),
unless we reduce the number of features first.

7. Feature Selection and Re-training

The feature elimination method of KNIME (in Meta)
is not appropriate for problems
where the number of features is rather high (>100).

We require a relatively fast filter method,
such as the one based in Information Gain.

We will create a R-snippet (local) for this task:

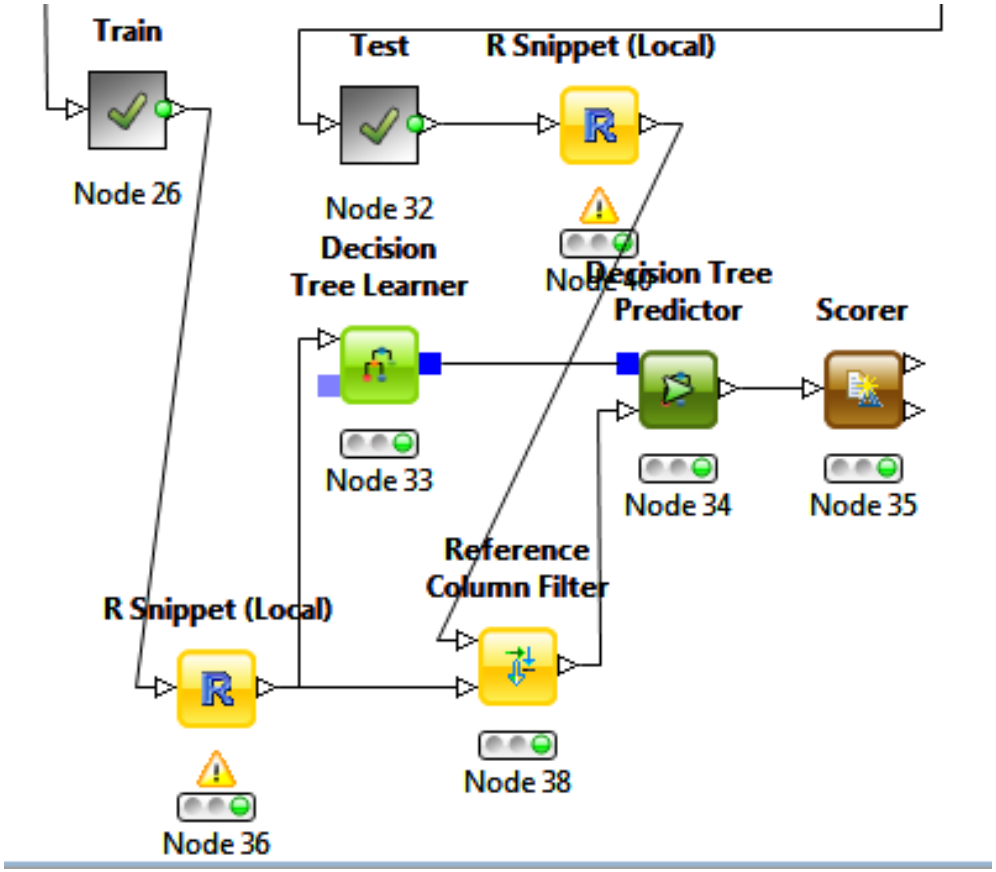
Include R-snippet (Local) as node.
Use train DT matrix as input to this node.
The R snippet returns a DT matrix with fewer columns.

Configure R snippet:

R binary: Indicate path to R.exe (check where the code is on your PC)
R Command: Copy R code (see next 2 slides) into R snippet area

Feature Selection and Re-training

KNIME workflow with R-snippets including R code



R-Snippet

```
info.terms <- vector()
find.info.terms <- function(dtm.tr, min.info){
    ix.class <- ncol(dtm.tr)
    default.info <- info(table(dtm.tr[, ix.class]))
    n.atr <- ncol(dtm.tr) - 1
    n.info.terms <- 0
    info.term.ixs <- vector()
    col.names <- names(dtm.tr)
    for (atri in 1:n.atr) {
        if (sum(dtm.tr[,atri])>0)
        {
            no.dif.atr.val <- length(table(dtm.tr[, atri]))
            atr.class.table <- table(dtm.tr[, atri], dtm.tr[, ix.class])
            n.rows<-nrow(dtm.tr)
            atr.info <- 0
            for (atr.val in 1 : no.dif.atr.val)
             { # begin for
                atr.weight <- sum( atr.class.table[atr.val,]) / n.rows
                atr.info1 <- atr.weight * info(atr.class.table[atr.val,])
                atr.info <- atr.info + atr.info1
          }
             info.gain <- default.info - atr.info
             if (info.gain > min.info)
               { info.term.ixs[n.info.terms] <- atri
                 n.info.terms <- n.info.terms + 1 }
          }
    }
    return(col.names[info.term.ixs])
} # end function
```

R-Snippet

```
info <- function(x){
inf <- 0
sumx <- sum(x)
for (i in x) {
  pi <- i/sumx
  infi <- (pi)*log2(pi)
  if (is.na(infi)) infi <- 0
  inf <- inf - infi }
  return(inf)
}
names <- find.info.terms(R,0.005)  # choose threshold
names <- append(colnames(R)[1], names)
names <- append(names, colnames(R)[length(colnames(R))])


R <- R[, names]
```

Note:
The parameter of minimum information gain of 0.005 cannot be passed to R, as KNIME can pass only one input (train DT matrix).
The value has to be set inside the program / snippet (see above).

Results of Feature Elimination

The number of terms was reduced from >8000 to 587 (i.e. <10%)

However, some column names were altered as a result.
The effect of invoking R snippet has the effect that
some columns in the Train data got automatically renamed
(e.g. 3-02 -> X3.02, health-care -> health.care etc.).

To guarantee that the same transformation is performed
also for the Test data, an R snippet needs to be included to process it.
The R snippet code includes just R <- R.

Processing the Test Set

We need to guarantee that the Test data
does not contain any terms (columns) that did not appear in the Train data.
(recall the Train data has been reduced by feature elimination).

We can use Reference Column Filter for this purpose:
Port 1 (top) : Test data from which we want to eliminate columns
Port 2 (bottom): Train data (reduced) used as reference for elimination

Only those columns in Test data that appear also in Train data
are passed through.

Results on Test Data

Train DT classifier on reduced Train set & apply it to the test data.

Success rate: 83.8%
(a bit lower compared to a variant that used all terms)

Confusion matrix:

	sci^	rel^
sci	353	43
rel	62	189

The performance of the classifier could be optimized
by searching for the best combination of preprocessing steps,
parameter setting(s) and the most appropriate type of classifier.