The KNIME Text Processing Plugin

Kilian Thiel

Nycomed Chair for Bioinformatics and Information Mining, University of Konstanz, 78457 Konstanz, Deutschland,
Kilian.Thiel@uni-konstanz.de

Abstract. This document describes the function, usage and underlying structure of the KNIME [1] text processing plugin. The plugin provides new data types and nodes, which allow to parse textual documents, represent these documents by KNIME data types and operate with them in multiple manners. Documents can be enriched in a way such that named entities are recognized, structural representation can be changed, like transforming a bag of words into a vector space and several preprocessing mechanisms can be applied. Altogether the KNIME text processing plugin is a powerful tool to integrate, process, and handle textual data the KNIME way.

1 Introduction

The KNIME Text Processing plugin was developed to process textual and natural language data. It is a mixture of natural language processing, text mining and information retrieval. With this plugin it is possible to read textual data, represent this data internally as documents and terms and apply several tagging, filtering, frequency computation, data structure transformation and other preprocessing and analysis tasks on it. Finally, the processed text data can be transformed into vector spaces and mined by usual data mining methods provided by KNIME. This means that the text processing plugin basically is a kind of intermediary plugin, reading, preprocessing and transforming textual data into a numerical representation.

In the next section the philosophy and usage of the plugin is described: which procedures to apply in which order and how to set them up. Section 3 depicts the available nodes and their functionality. In Section 4 the underlying data structures and new KNIME data types are explained, as well as the different data table structures, like bag of words, lists of documents, document vectors, etc.

2 Philosophy

In this section the philosophy of the text processing plugin is explained, meaning which categories of nodes have to or can be applied in which order. The order is important since the nodes of the plugin require certain structures or specifications of input data tables, i.e. the "BoW" node, creating a bag of words requires

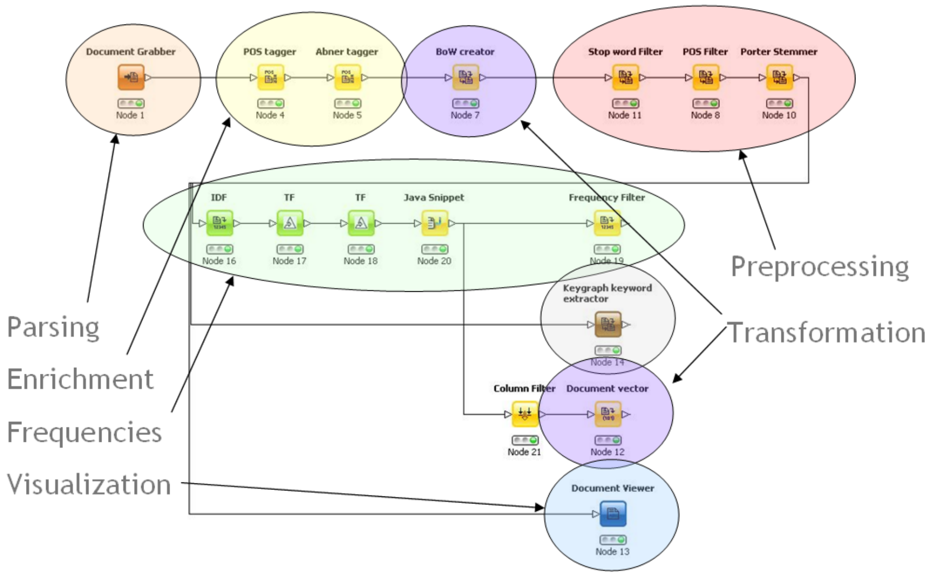


Fig. 1. An example workflow illustrating the basic philosophy and order of the text processing nodes.

a list of documents as input data and returns a bag of words as output data. This node will only work (configure) properly with a list of documents as input data. Like the "BoW" node also other nodes require specific input data table structures and return specific output data tables. Consequently certain types of nodes can only be used in a certain order. This order or configuration is the basic policy of the KNIME text processing plugin.

The example workflow in figure 1 illustrates the basic order of text processing nodes.

In the following the order of procedures is specified and explained in detail:

1. Parsing
2. Enrichment
3. Transformation
4. Preprocessing
5. Frequencies and keyword extraction
6. Transformation

2.1 Parsing

First of all the text documents have to be parsed. The text and the structure, if available, have to be extracted and represented in a data structure, KNIME is able to access and use in an appropriate manner. A document is represented internally by an instance of the Java class *Document*, this class is wrapped by the KNIME data type *DocumentCell* and *DocumentValue*. More details about these data types and the underlying data structure can be found in section 4. For now the only important thing is that the parser nodes read text stored in a certain format, analyze the structure and convert both into a *DocumentCell* for each document to parse.

All nodes of the KNIME text processing plugin can be found in the category "Textprocessing" of the node repository, if the plugin is installed properly. The parser nodes are in the subcategory "IO". Two nodes, the "DML Document Parser" and the "SDML Document Parser" parse XML document formats which are used to store the data of *DocumentCell*s internally or to serialize them respectively. Additionally a PubMed parser node is available enabling to read PubMed XML results. Furthermore the "Flat File Document Parser" node reads unstructured text from flat files and represents it in document form.

The question now is, how to get your text data into KNIME ? If you have PubMed text data you can simply use the PubMed parser, for flat file, the flat file parser will do, but what for a particular XML format ?

1. One possibility is to transform this particular XML into the internally used DML or SDML, which is basically the same but simpler. Once the particular XML is transformed, i.e. by XSLT, the DML or SDML parser can be used to parse the data.
2. Another way would be, to write your text into flat text files and use the flat file parser, the main drawback of this option is that the structural information of the text will be lost.
3. The third possibility is to implement an additional parser node, which is able to parse the particular format.

The last option is definitively the most complex and time consuming one but once the node is implemented no more transformation has to be done. As a start it is recommendable to use one of the existing parsers and transform the texts into the DML or SDML format.

Once the texts have been parsed, the output data table of the corresponding parser node contains the documents, in each row one, as illustrated in figure 2. The icon, shown in the header of the second column, points out that the type of the data contained by this column is the document data.

2.2 Enrichment and tagging

After parsing the text and converting it into document cells, the documents and the included information can be enriched, i.e. by recognizing named entities. All enrichment nodes require a data table containing exactly one column of document cells and returns an output data table with exactly one column of document cells. Each node analyses the documents in a way such that it combines multiple words to terms, if reasonable, and applies certain tags to these terms. These tagged terms can be recovered later on, and be treated in a certain way, for instance, be filtered. A document after the tagging process contains more information than a document before, it has been enriched.

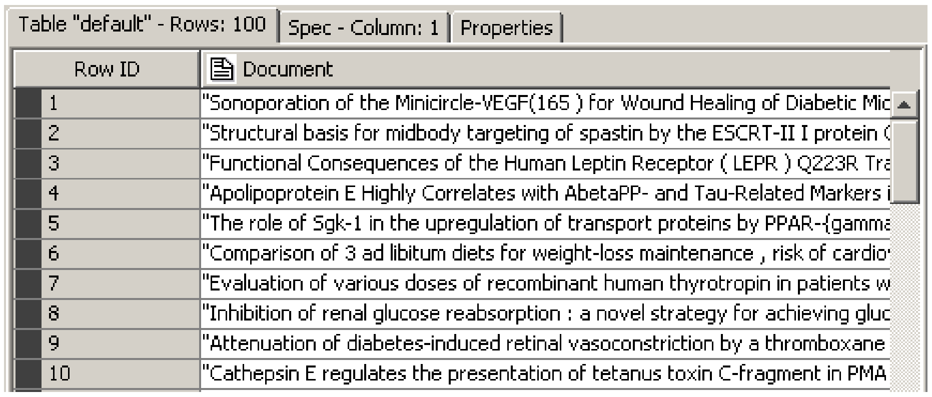


Table "default" - Rows: 100 | Spec - Column: 1 | Properties

Row ID	Document
1	"Sonoporation of the Minicircle-VEGF(165) for Wound Healing of Diabetic Mic
2	"Structural basis for midbody targeting of spastin by the ESCRT-II I protein (
3	"Functional Consequences of the Human Leptin Receptor (LEPR) Q223R Tra
4	"Apolipoprotein E Highly Correlates with AbetaPP- and Tau-Related Markers i
5	"The role of Sgk-1 in the upregulation of transport proteins by PPAR-{gamma
6	"Comparison of 3 ad libitum diets for weight-loss maintenance , risk of cardio
7	"Evaluation of various doses of recombinant human thyrotropin in patients w
8	"Inhibition of renal glucose reabsorption : a novel strategy for achieving gluc
9	"Attenuation of diabetes-induced retinal vasoconstriction by a thromboxane
10	"Cathepsin E regulates the presentation of tetanus toxin C-fragment in PMA

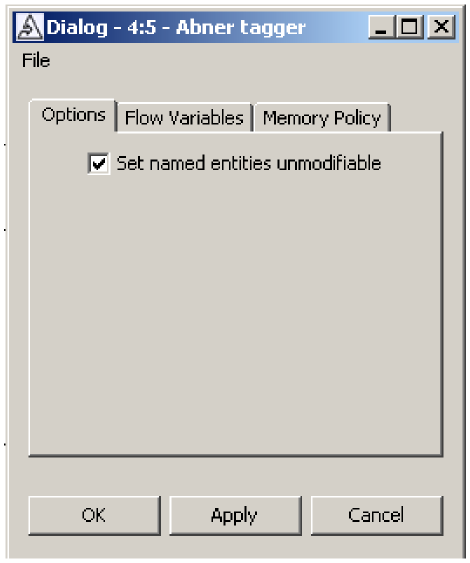
Fig. 2. A data table containing a list of PubMed documents, in each row one.

All available enrichment nodes can be found in the “Enrichment” node category. The“POS tagger” node adds part of speech tags, according to the Penn Treebank tagset[1], to each word, which is afterwards considered as a term. Later on a filtering can be applied based on the POS tags, i.e. is it possible to filter all nouns or verbs. The underlying model used to assign the POS tags was taken from the OpenNLP project[2]. OpenNLP models that recognize named entities, such as locations, organizations, or persons have been wrapped by the “OpenNLP NE tagger” node. Another enrichment node is the “Abner tagger” which recognizes biomedical named entities, such as cell lines, genes, or proteins. The model used here is the “ABNER” [2] model. Since there are many different types of named entities a “Dictionary tagger” has been implemented, able to read a certain dictionary of named entities, search the documents for them and tag the corresponding terms. Once a dictionary of interesting named entities, i.e. company names is created, this node can be used to find them.

Once words have been recognized as named entities, bundled together as terms and been tagged, these terms can be set *unmodifiable*, meaning that preprocessing nodes will not filter or change these terms anymore. To set the terms *unmodifiable* simply check the “Set named entities unmodifiable” check box of the enrichment nodes, as shown in Figure 3.

A very important behavior of these tagger nodes is that the node applied in the end of the enrichment workflow dominates in case of conflicts. Assume a certain named entity recognizer node finds and tags company names, now assume another enrichment node is applied exactly afterwards in the workflow and a conflict appears related to the tagging of a certain term. The last applied node breaks the conflicting terms into words, recombines them in a way of the enrichment strategy and finally applies tags to them. This means that the word combination of the antecessor node does not exist anymore.

[1] http://www.cis.upenn.edu/ treebank/
[2] http://opennlp.sourceforge.net/



[✓] Set named entities unmodifiable

Fig. 3. The dialog of the Abner tagger node with a check box to set the recognized named entities unmodifiable.

2.3 Transformation

Before the enriched documents can be preprocessed, i.e. filtered, stemmed etc., the data table structure has to be transformed into a bag of words. Therefore the "BoW creator" node has to be used. All nodes, transforming the data table structure can be found in the "Transformation" category. The BoW node requires a list of documents as input data table, consisting of exactly one column of document cells. The input documents and its terms are converted into a bag of words structure. The output data table consists of two columns, the first is the term column and the second the document column. A row of this data table means that the term in this row occurs in the document. Figure 4 shows a bag of words data table with two rows, one containing the terms and the other the documents. Be aware that the terms tags are listed in brackets after the term.

2.4 Preprocessing

Once a bag of words is created preprocessing nodes, such as filtering, stemming etc. can be applied, which can be found in the "Preprocessing" category. These nodes require a bag of words input data table and return a preprocessed bag of words output data table. Filters simply filter rows according to the terms of these rows and the filter criteria. For instance:

– the "Stop word Filter" node filters terms which are *stop words*
– the "N Char Filter" filters terms consisting of less then a specified number of characters

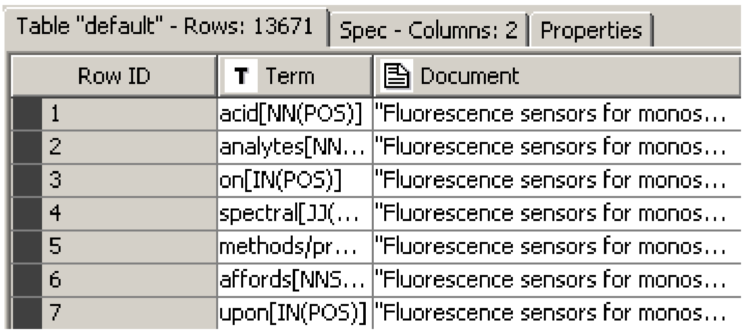


Table "default" - Rows: 13671 | Spec - Columns: 2 | Properties

Row ID	Term	Document
1	acid[NN(POS)]	"Fluorescence sensors for monos...
2	analytes[NN...	"Fluorescence sensors for monos...
3	on[IN(POS)]	"Fluorescence sensors for monos...
4	spectral[JJ(...	"Fluorescence sensors for monos...
5	methods/pr...	"Fluorescence sensors for monos...
6	affords[NNS...	"Fluorescence sensors for monos...
7	upon[IN(POS)]	"Fluorescence sensors for monos...

Fig. 4. A bag of words data table containing two columns, one consisting of terms and one of documents.

– the "POS Filter" filters terms which are tagged as nouns or verbs, etc.

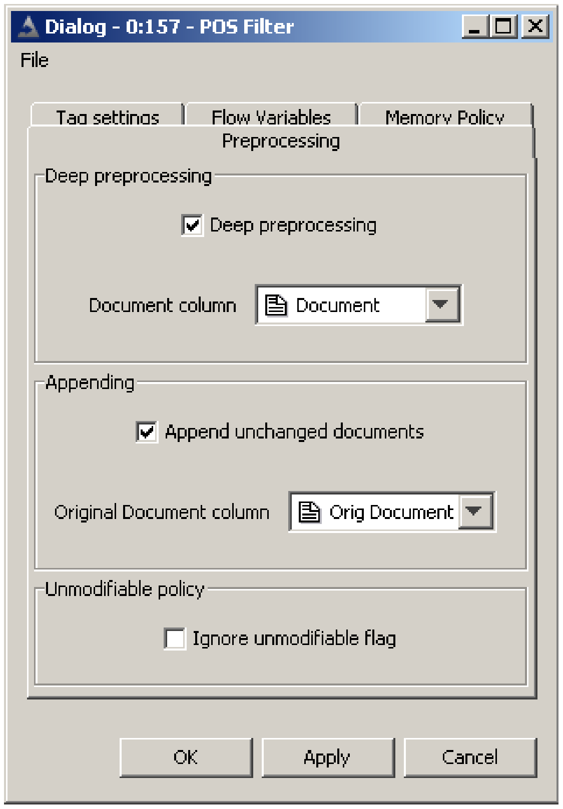
The available set of filters allows to filter irrelevant terms and reduce the size of the bag of words. Additionally preprocessing nodes are available which modify terms, such as

– the "Porter stemmer", reducing the words to their stems by the porter [3] algorithm
– the "Case converter"
– the "Punctuation erasure", removing all punctuation marks
– the "Replacer", replacing specified sequences of characters with other characters.

To stem English words the Porter or the Kuhlen stemmer can be used. For other languages the Snowball stemmer node has to be applied, using the Snowball[3] stemmer library. A very important issue about preprocessing a bag of words data table is that on the one hand only the terms contained in the term column can be preprocessed. On the other hand the terms contained in the document itself can be affected by the preprocessing procedure as well. This is called deep preprocessing. As one can imagine deep preprocessing is more time consuming than the shallow one, since it processes the terms inside the document too and builds a new document afterwards.

The deep preprocessing is essential if one wants to compute frequencies afterwards, based on the preprocessed terms. For example, if a term has been stemmed it is impossible to the term frequency node, to find and count the stemmed term in the original document. The terms contained in the document have to be stemmed too. Therefore the deep preprocessing option is available for all preprocessing nodes and can simply be activated by checking the "Deep preprocessing" check box in the node dialog, shown in Figure 5. The original document can be appended unchanged, so that both versions are available.

[3] http://snowball.tartarus.org/


Dialog - 0:157 - POS Filter

File

Tag settings | Flow Variables | Memory Policy | Preprocessing

Deep preprocessing

[✓] Deep preprocessing

Document column: Document

Appending

[✓] Append unchanged documents

Original Document column: Orig Document

Unmodifiable policy

[ ] Ignore unmodifiable flag

OK | Apply | Cancel

Fig. 5. The dialog of the POS filter. The topmost check box is the “Deep preprocessing” check box. If checked deep preprocessing will be applied.

Additionally it can be specified whether the preprocessing step of the node is applied to terms, which have been set unmodifiable as well, see bottom of Figure 5. By default this option is unchecked, meaning that terms which have been set unmodifiable by a enrichment node will not be affected by the preprocessing. In some cases ignoring the unmodifiable flag can be useful, i.e. when named entity recognition has been applied and persons and locations have been identified and tagged. To separate persons from locations the “Standard Named Entity Filter” node has to be used ignoring the unmodifiable flag of persons or locations respectively. Otherwise the node would not filter either persons or locations.

2.5 Frequencies and Keyword extraction

By applying preprocessing nodes, the bag of words can be reduced and irrelevant terms can be filtered out. To specify the relevancy of a term, several term frequencies can be computed and the terms can be filtered according to these values. To compute these frequencies two nodes are available, one to compute the well known term frequency tf, relative or absolute, and one to compute the inverse document frequency idf. These nodes require a bag of words as input data table, and return a bag of words as output data table, with additional columns containing the computed frequencies.

After the frequencies have been computed the terms can be filtered, by the "Frequency Filter" node, related to the frequency values. It is possible to specify minimum and maximum values or a particular number k, so that only the k terms with the highest frequency are kept. Beside extracting keywords based on their $tfidf$ value other nodes can be applied as well, like the "Chi-square keyword extractor" using a chi-square measure to score the relevance [4] or the "Keygraph keyword extractor" [5] using a graph representation of documents to find keywords.

2.6 Transformation

Now only the relevant terms should remain in the bag of words and one can finally begin with the mining task. Therefore one last transformation has to be done. The bag of words has to be transformed into vectors. Again the necessary transformation nodes can be found in the "Transformation" category. The "Document vector" node creates a document vector for each document, representing it in the term space. This means, every different term is considered as a feature and the document vector is a vector in this feature space. If a document contains a certain term, a specific value (usually $tfidf$ or 1) occurs in the document vector at the corresponding dimension. Analogous the "Term vector" node creates a term vector for each term, representing it in the document space. Once these vectors have been created it is possible to apply data mining methods to them. Unlabeled document vectors for instance can be clustered, labeled ones can be classified, or term association rules can be mined.

3 Nodes

In the following table all the available nodes are listed together with a short description of their functionality.

IO	
DML Document Parser	Parses DML formatted files.
Document Grabber	Sends specified query to PubMed, downloads and parses the resulting documents.
Flat File Document Parser	Reads flat files and creates one document for each.
PubMed Document Parser	Parses PubMed result files.
SDML Document Parser	Parses SDML formatted files.

Enrichment	
Abner tagger	Recognizes and tags biomedical named entities based on ABNER.
Dictionary tagger	Searches for named entities specified in a dictionary and tags them.
OpenNLP NE tagger	Recognizes persons, locations, organization, money, date, or time, based on different OpenNLP models.
POS tagger	Tags words with part of speech tags.
Transformation	
BoW creator	Transforms a list of documents into a bag of words.
Document Data Extractor	Extracts specified fields, like authors, publication date, title etc. of documents and adds them as string columns.
Document vector	Transforms documents in a bag of words into document vectors.
Sentence Extractor	Extracts all sentences of documents and adds them as string column.
String to Term	Converts strings into terms.
Strings to Document	Builds documents out of specified strings.
Tags to String	Converts tag values to strings.
Term to String	Converts a term into a string.
Term vector	Transforms terms in a bag of words into term vectors.
Preprocessing	
Abner Filter	Filters terms tagged with ABNER tags.
Case converter	Converts terms to lower or upper case.
Dict Replacer	Replaces certain terms or substrings of terms with a particular replacement all specified in a dictionary.
Kuhlen stemmer	Stems words with the Kuhlen algorithm.
Modifiable Term Filter	Filters terms which have been set un(modifiable).
N Chars Filter	Filters terms with less than N characters.
Number Filter	Filters term which are numbers.
POS Filter	Filters terms with specific part of speech tags.
Porter stemmer	Stems terms with the Porter algorithm.
Punctuation Erasure	Removes all punctuation marks from terms.
Replacer	Replaces in terms a certain character sequence by another one.

Snowball Stemmer	Stems terms using the Snowball stemmer library.
Standard named entity filter	Filters terms tagged with standard named entities.
Stop word Filter	Filters terms which are stop words.
Term Grouper	Groups terms w.r.t. their strings ignoring their tags.
Frequencies	
Frequency Filter	Filters terms with a certain frequency.
IDF	Computes the inverse document frequency of terms.
TF	Computes the term frequency of terms.
Misc	
Category to class	Adds the category of a document as string column.
Chi-square keyword extractor	Extracts keywords, estimated as relevant by the Chi-square measure.
Document Viewer	Displays the content of a document.
Keygraph keyword extractor	Extracts keywords, estimated as relevant by the keygraph algorithm.
String Matcher	Computes the Levenshtein distance between a list of strings and a string dictionary.
Tagcloud	Displays a bag of words as tag cloud.

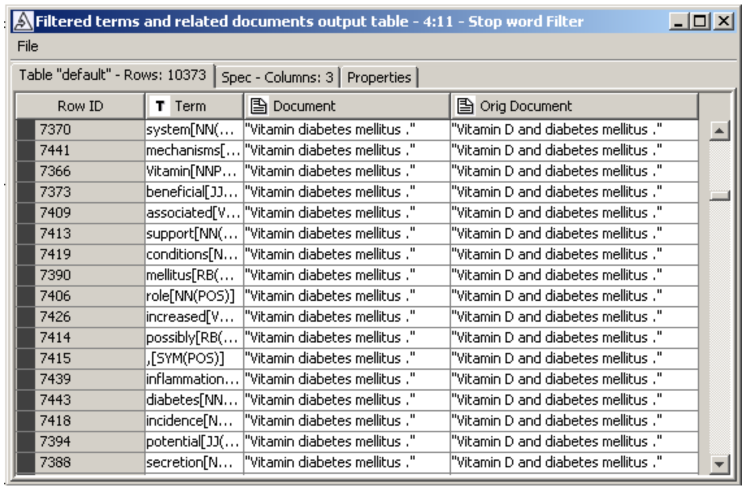
Table 1: A listing of all available nodes on the left and a short description on the right.

4 Structures

In this section the different data table structures required by the text processing nodes, the three new data types, *DocumentCell*, *DocumentBlobCell* and *TermCell* as well as the internal document and term structures and its serialization is described.

4.1 Data table structures

As already mentioned in section 2 the nodes, provided by this plugin, require a certain structure of input data tables in order to configure and execute properly. What kind of data table structure is specified by the category of the nodes. Basically there are three different kind of structures. A table with only one column containing a document cell in each row, shown in figure 2, a bag of words consisting of two columns, one containing a term cell, the other containing a document


Filtered terms and related documents output table - 4:11 - Stop word Filter

File

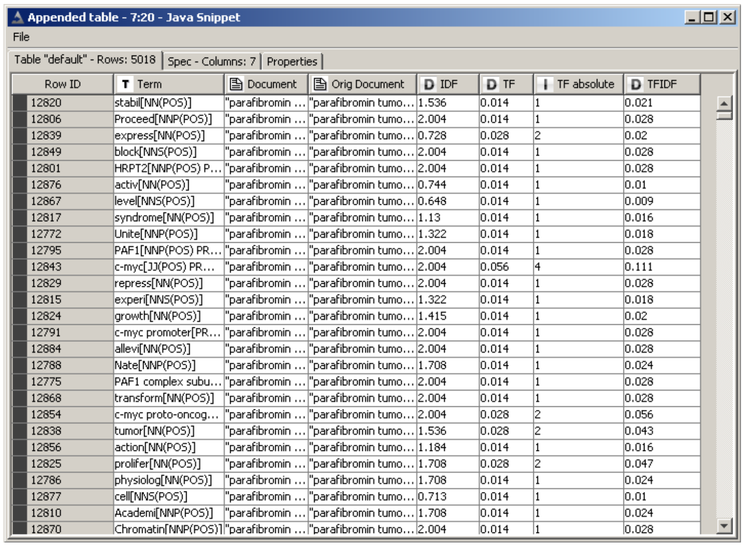
Table "default" - Rows: 10373 | Spec - Columns: 3 | Properties

Row ID	Term	Document	Orig Document
7370	system[NN(…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7441	mechanisms[…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7366	Vitamin[NNP…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7373	beneficial[JJ…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7409	associated[V…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7413	support[NN(…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7419	conditions[N…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7390	mellitus[RB(…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7406	role[NN(POS)]	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7426	increased[V…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7414	possibly[RB(…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7415	,[SYM(POS)]	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7439	inflammation…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7443	diabetes[NN…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7418	incidence[N…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7394	potential[JJ(…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."
7388	secretion[N…	"Vitamin diabetes mellitus ."	"Vitamin D and diabetes mellitus ."

Fig. 6. A bag of words data table with an additional column containing the original, unchanged document.

cell in each row, illustrated in figure 4 and a vector structure representing term or document vectors.

This basic bag of words structure can be expanded in a way such that additional columns can be added, like the original, unchanged document, when

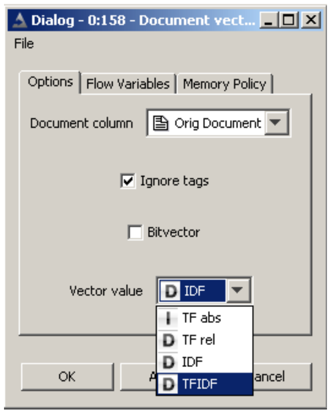

Appended table - 7:20 - Java Snippet

File

Table "default" - Rows: 5018 | Spec - Columns: 7 | Properties

Row ID	Term	Document	Orig Document	IDF	TF	TF absolute	TFIDF
12820	stabil[NN(POS)]	"parafibromin …	"parafibromin tumo…	1.536	0.014	1	0.021
12806	Proceed[NNP(POS)]	"parafibromin …	"parafibromin tumo…	2.004	0.014	1	0.028
12839	express[NN(POS)]	"parafibromin …	"parafibromin tumo…	0.728	0.028	2	0.02
12849	block[NNS(POS)]	"parafibromin …	"parafibromin tumo…	2.004	0.014	1	0.028
12801	HRPT2[NNP(POS) P…	"parafibromin …	"parafibromin tumo…	2.004	0.014	1	0.028
12876	activ[NN(POS)]	"parafibromin …	"parafibromin tumo…	0.744	0.014	1	0.01
12867	level[NNS(POS)]	"parafibromin …	"parafibromin tumo…	0.648	0.014	1	0.009
12817	syndrome[NN(POS)]	"parafibromin …	"parafibromin tumo…	1.13	0.014	1	0.016
12772	Unite[NNP(POS)]	"parafibromin …	"parafibromin tumo…	1.322	0.014	1	0.018
12795	PAF1[NNP(POS) PR…	"parafibromin …	"parafibromin tumo…	2.004	0.014	1	0.028
12843	c-myc[JJ(POS) PR…	"parafibromin …	"parafibromin tumo…	2.004	0.056	4	0.111
12829	repress[NN(POS)]	"parafibromin …	"parafibromin tumo…	2.004	0.014	1	0.028
12815	experi[NNS(POS)]	"parafibromin …	"parafibromin tumo…	1.322	0.014	1	0.018
12824	growth[NN(POS)]	"parafibromin …	"parafibromin tumo…	1.415	0.014	1	0.02
12791	c-myc promoter[PR…	"parafibromin …	"parafibromin tumo…	2.004	0.014	1	0.028
12884	allevi[NN(POS)]	"parafibromin …	"parafibromin tumo…	2.004	0.014	1	0.028
12788	Nate[NNP(POS)]	"parafibromin …	"parafibromin tumo…	1.708	0.014	1	0.024
12775	PAF1 complex subu…	"parafibromin …	"parafibromin tumo…	2.004	0.014	1	0.028
12868	transform[NN(POS)]	"parafibromin …	"parafibromin tumo…	2.004	0.014	1	0.028
12854	c-myc proto-oncog…	"parafibromin …	"parafibromin tumo…	2.004	0.028	2	0.056
12838	tumor[NN(POS)]	"parafibromin …	"parafibromin tumo…	1.536	0.028	2	0.043
12856	action[NN(POS)]	"parafibromin …	"parafibromin tumo…	1.184	0.014	1	0.016
12825	prolifer[NN(POS)]	"parafibromin …	"parafibromin tumo…	1.708	0.028	2	0.047
12786	physiolog[NN(POS)]	"parafibromin …	"parafibromin tumo…	1.708	0.014	1	0.024
12877	cell[NNS(POS)]	"parafibromin …	"parafibromin tumo…	0.713	0.014	1	0.01
12810	Academi[NNP(POS)]	"parafibromin …	"parafibromin tumo…	1.708	0.014	1	0.024
12870	Chromatin[NNP(POS)]	"parafibromin …	"parafibromin tumo…	2.004	0.014	1	0.028

Fig. 7. A bag of words data table with an additional column containing tf, idf and $tfidf$ frequencies.



Document column: Orig Document

[✓] Ignore tags

[ ] Bitvector

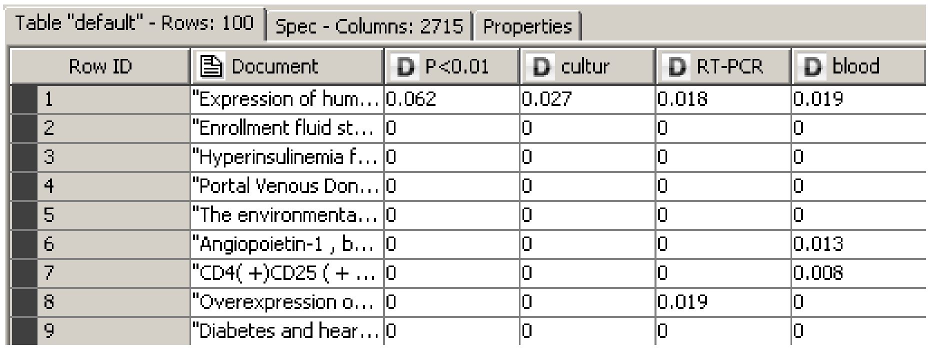
Vector value: IDF

- TF abs
- TF rel
- IDF
- TFIDF

Fig. 8. The dialog of the document vector generator node.

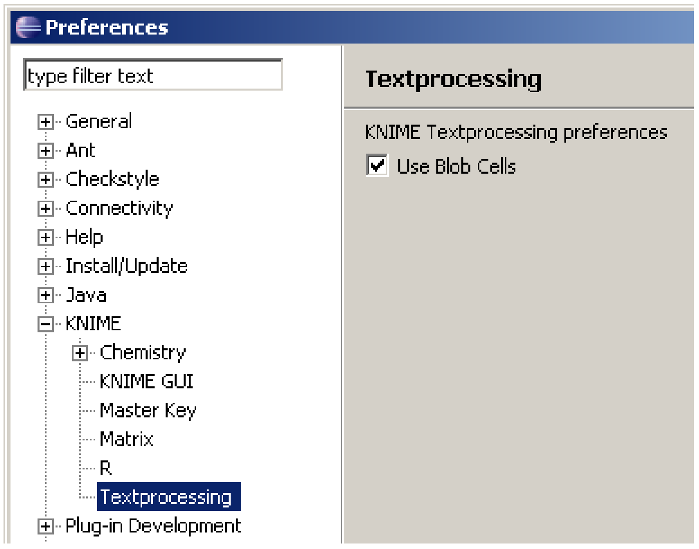
deep preprocessing is activated. In figure 6 such kind of a data table is depicted. Furthermore it can be seen that the document titles of the middle column and those of the right column differ. The original document on the right still retains its original title, whereas the document in the middle column, affected by the impacts of deep preprocessing, lost several terms during the filtering process. Not only the documents body is changed by the filtering, the title can change as well.

In addition to the original document column, frequency columns are added by node of the category "Frequencies". Figure 7 shows a data table with additional columns containing tf, idf and $tf * idf$ frequencies. According to these frequencies certain filters can be applied, as described in 2.5. The final data table structure change is usually the transformation into a vector space, such as the document or term vector space. It is possible to create bit vectors, or to specify a frequency column, which entries are considered as vector values. The dialog



Row ID	Document	P<0.01	cultur	RT-PCR	blood
1	"Expression of hum...	0.062	0.027	0.018	0.019
2	"Enrollment fluid st...	0	0	0	0
3	"Hyperinsulinemia f...	0	0	0	0
4	"Portal Venous Don...	0	0	0	0
5	"The environmenta...	0	0	0	0
6	"Angiopoietin-1 , b...	0	0	0	0.013
7	"CD4(+)CD25 (+ ...	0	0	0	0.008
8	"Overexpression o...	0	0	0.019	0
9	"Diabetes and hear...	0	0	0	0

Fig. 9. A data table consisting of document vectors.



Textprocessing

KNIME Textprocessing preferences

[✓] Use Blob Cells

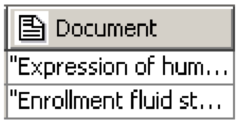
Fig. 10. Preference page of the text processing plugin.

of the document vector generator is shown in figure 8. The before computed frequencies can be selected as vector values. Figure 9 illustrates the output data table of the document vector generator. The first column contains the document, the following the specified frequency accordant to the document and a certain term, displayed in the column headings. Based on these vectors conventional data mining methods, like clustering or categorization, can be applied.

4.2 Document(Blob)Cell

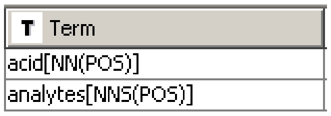
It is obvious that almost all of the data tables processed by the nodes of the text processing plugin contain documents and/or terms. Therefore three new data types have been implemented, the *DocumentCell*, the *DocumentBlobCell* and the *TermCell*. Basically the DocumentCell and the DocumentBlobCell are the same except that the super class of the DocumentBlobCell is *BlobDataCell*. This means that, in contrast to a usual *DataCell*, the multiple occurrence of a certain document in a data table is referenced. For instance the bag of word data table contains tuples of terms and documents, which leads to multiple occurrences of the same document in that table. To save memory all occurrences of a certain document, after its first listing, are referenced, meaning that the data table holds only an address of the corresponding data cell.

This behavior can be switched off in the preference page of the text processing plugin, which can be found as a sub page in the KNIME preferences. Figure 10 shows the text processing preference page and the check box to specify the usage of blob cells. Be aware that if no blob cells are used, this will consume a lot of memory since the same documents are stored multiple times in the data tables.



Document
"Expression of hum…
"Enrollment fluid st…

Fig. 11. The item of the document cell displayed in the column heading.



Term
acid[NN(POS)]
analytes[NNS(POS)]

Fig. 12. The item of the term cell displayed in the column heading.

The benefit of not using blob cells is an increase of processing speed, since the references don't have to be resolved. However, the benefit can not compensate the drawback if one have to deal with limited memory. By default this check box is checked and blob cells are used. Uncheck this box only after carefully balancing the reasons!

In figure 11 a column containing document cells is illustrated. The heading of this column shows the icon of the document cell. Table views, like the output table view or the interactive table only depict the title of the document.

4.3 TermCell

For the adequate representations of terms the data type *TermCell* is provided by the plugin. A term is not only a string, but a list of words bundled together, constituting a multi word, or term. Furthermore several tags can be attached to a term, accordant to the terms meaning and the specific tagger. For instance the words "New York City" can be recognized as a named entity, more precisely a location, by a named entity tagger and bundled together to a term. Additionally the tag "Location" can be attached to that term. According to the used taggers and the meaning of terms it is possible that terms may have more then one tag attached.

Figure 12 illustrates a column containing term cells. In the column heading the term cell icon is displayed. The string representation of a term, shown in table views is assembled by its words first followed by the attached tags in brackets. First the name of the tag is shown, i.e. "NN" for noun and second in parenthesis the type of the tag, i.e. "POS" for part of speech.

4.4 Internal data structures

The two kinds of document cells are both based on the essential *Document* class. This class encapsulates the document itself, its data and meta data. Basically a document consist of sections, which can be chapter, abstract or title. To each section an annotation can be attached, which specifies the kind of section, i.e.

"Title". A section again is composed several paragraphs, and a paragraph consists of sentences. A sentence contains terms, and terms contain words and tags, as mentioned before. Figure 13 illustrates the composition of a document.

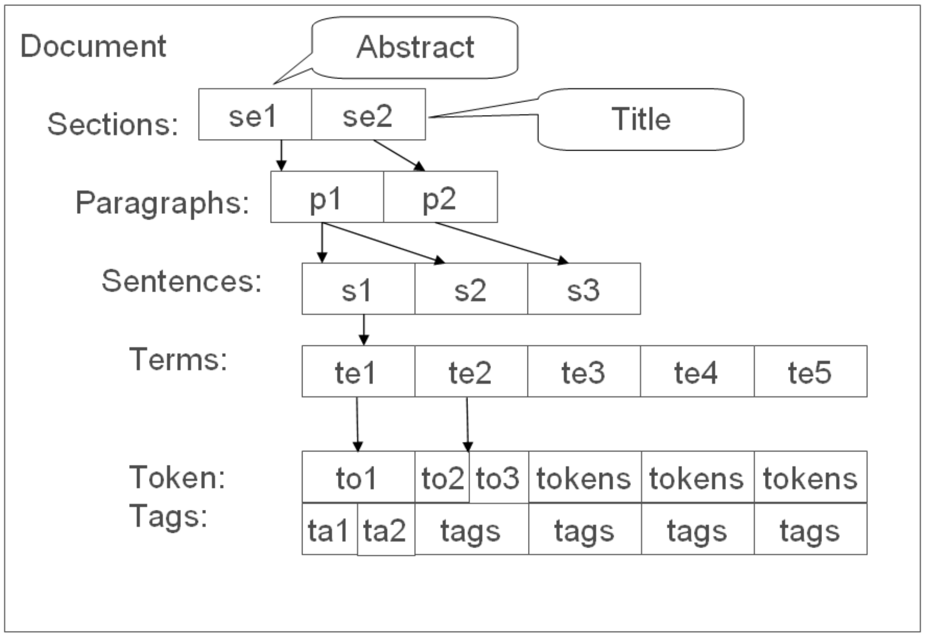


Fig. 13. The internal structure of a document.

This level of granularity allows to access all parts of the document easily and enables to capture the principal part of the structure of several document formats.

4.5 Serialization

As all data cells, the document cells can be serialized. The serialization is done via XML, more detail via DML the internally used XML format to save documents. Since the plugin provides two parser nodes, one to parse DML and one for SDML it is useful to convert the documents to parse into DML or SDML. If you already have implemented a parser node which is able to read your specific document format, of course a conversion is unnecessary. But implementing a parser node however is more time consuming than converting your files. In the following the DML and SDML specification is shown.

DML specification

Listing 1.1. DTD of the DML format.

```
<!ELEMENT Document
 (Category*,Source*,DocumentType?,Authors?,
 PublicationDate?,Section+,Filename?)>

<!ELEMENT Category (#PCDATA)>
<!ELEMENT Source (#PCDATA)>
<!ELEMENT Filename (#PCDATA)>
<!ELEMENT DocumentType (#PCDATA)>

<!ELEMENT Authors (Author*)>
<!ELEMENT Author (Firstname?, Lastname)>
<!ELEMENT Firstname (#PCDATA)>
<!ELEMENT Lastname (#PCDATA)>

<!ELEMENT PublicationDate (Day?, Month?, Year)>
<!ELEMENT Day (#PCDATA)>
<!ELEMENT Month (#PCDATA)>
<!ELEMENT Year (#PCDATA)>

<!ELEMENT Section (Paragraph+)>
<!ATTLIST Section Annotation CDATA #REQUIRED>
<!ELEMENT Paragraph (Sentence+)>
<!ELEMENT Sentence (Term+)>
<!ELEMENT Term (Word+,Tag*)>
<!ELEMENT Word (#PCDATA)>
<!ELEMENT Tag (TagValue,TagType)>
<!ELEMENT TagType (#PCDATA)>
<!ELEMENT TagValue (#PCDATA)>
```

SDML specification

Listing 1.2. SDML-DTD

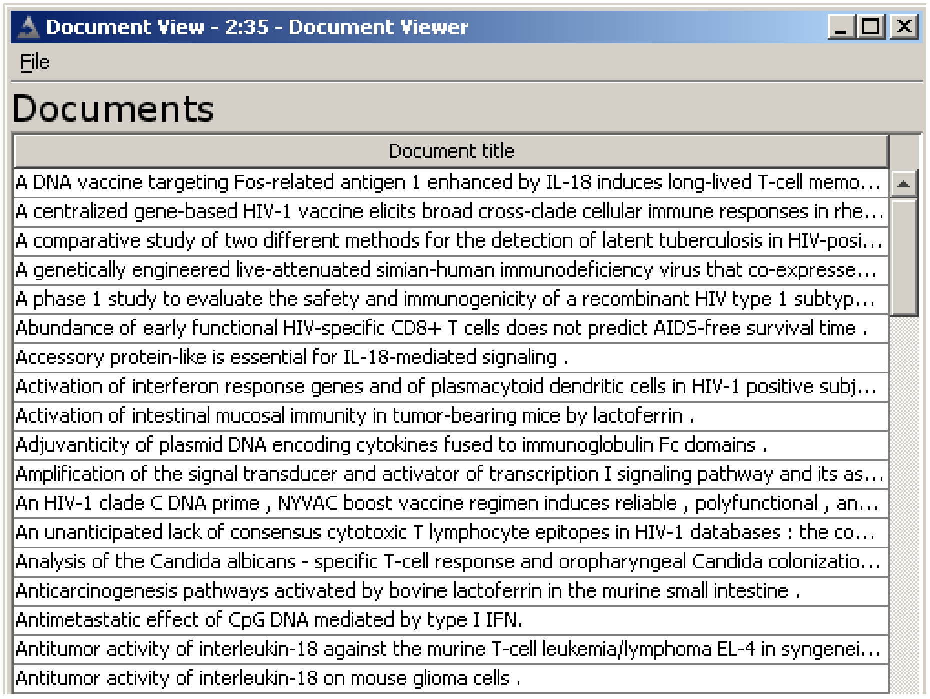
```
<!ELEMENT Document
 (Authors?,PublicationDate?,Title,Section+)>

<!ELEMENT Authors (Author*)>
<!ELEMENT Author (Firstname?, Lastname)>
<!ELEMENT Firstname (#PCDATA)>
<!ELEMENT Lastname (#PCDATA)>

<!ELEMENT PublicationDate (Day?, Month?, Year)>
<!ELEMENT Day (#PCDATA)>
<!ELEMENT Month (#PCDATA)>
<!ELEMENT Year (#PCDATA)>

<!ELEMENT Titlel (#PCDATA)>
<!ELEMENT Section (#PCDATA)>
<!ATTLIST Section annotation CDATA #REQUIRED>
```

5 Visualization

So far there are two nodes that visualize textual data, the "Document Viewer" and the "Tagcloud". The "Document Viewer" node visualizes documents in a very simple way. A data table containing a list of documents can be used as input data. The document viewer shows first a list of all available documents to visualize, as shown in Figure14.

A double click on a particular document row will open a new frame showing the details of the corresponding document. All the detail data of documents are shown. In the bottom most panel, filename, publication date, document type, categories and sources are displayed. In the panel above all authors names are shown and in the main panel the text of the document and its title are displayed. Furthermore assigned tags of a specified type can be highlighted, which can be seen in Figure 15. Here terms that have ABNER tags assigned are highlighted red.

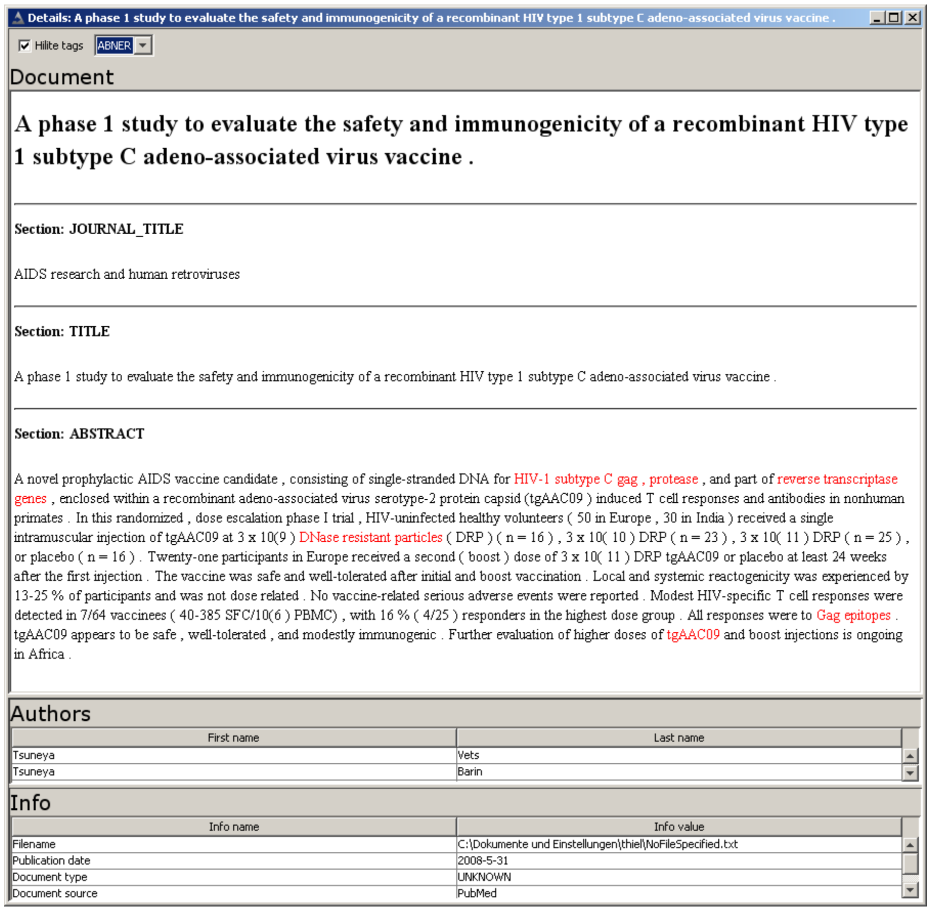
The "Tagcloud" node provides a typical tagcloud with some additional options, such as different arrangements of terms, i.e. size sorted, alphabetic, inside out, adjustable minimal and maximal font size, or transparency of terms etc. The node requires a input data table consisting of a bag of word with a additional column containing a weight or score of each term. This weight is used to



Documents

Document title
A DNA vaccine targeting Fos-related antigen 1 enhanced by IL-18 induces long-lived T-cell memo...
A centralized gene-based HIV-1 vaccine elicits broad cross-clade cellular immune responses in rhe...
A comparative study of two different methods for the detection of latent tuberculosis in HIV-posi...
A genetically engineered live-attenuated simian-human immunodeficiency virus that co-expresse...
A phase 1 study to evaluate the safety and immunogenicity of a recombinant HIV type 1 subtyp...
Abundance of early functional HIV-specific CD8+ T cells does not predict AIDS-free survival time .
Accessory protein-like is essential for IL-18-mediated signaling .
Activation of interferon response genes and of plasmacytoid dendritic cells in HIV-1 positive subj...
Activation of intestinal mucosal immunity in tumor-bearing mice by lactoferrin .
Adjuvanticity of plasmid DNA encoding cytokines fused to immunoglobulin Fc domains .
Amplification of the signal transducer and activator of transcription I signaling pathway and its as...
An HIV-1 clade C DNA prime , NYVAC boost vaccine regimen induces reliable , polyfunctional , an...
An unanticipated lack of consensus cytotoxic T lymphocyte epitopes in HIV-1 databases : the co...
Analysis of the Candida albicans - specific T-cell response and oropharyngeal Candida colonizatio...
Anticarcinogenesis pathways activated by bovine lactoferrin in the murine small intestine .
Antimetastatic effect of CpG DNA mediated by type I IFN.
Antitumor activity of interleukin-18 against the murine T-cell leukemia/lymphoma EL-4 in syngenei...
Antitumor activity of interleukin-18 on mouse glioma cells .

Fig. 14. The list of available documents.


Details: A phase 1 study to evaluate the safety and immunogenicity of a recombinant HIV type 1 subtype C adeno-associated virus vaccine .

☑ Hilite tags ABNER

Document

A phase 1 study to evaluate the safety and immunogenicity of a recombinant HIV type 1 subtype C adeno-associated virus vaccine .

Section: JOURNAL_TITLE

AIDS research and human retroviruses

Section: TITLE

A phase 1 study to evaluate the safety and immunogenicity of a recombinant HIV type 1 subtype C adeno-associated virus vaccine .

Section: ABSTRACT

A novel prophylactic AIDS vaccine candidate , consisting of single-stranded DNA for HIV-1 subtype C gag , protease , and part of reverse transcriptase genes , enclosed within a recombinant adeno-associated virus serotype-2 protein capsid (tgAAC09) induced T cell responses and antibodies in nonhuman primates . In this randomized , dose escalation phase I trial , HIV-uninfected healthy volunteers (50 in Europe , 30 in India) received a single intramuscular injection of tgAAC09 at 3 x 10(9) DNase resistant particles (DRP) (n = 16) , 3 x 10(10) DRP (n = 23) , 3 x 10(11) DRP (n = 25) , or placebo (n = 16) . Twenty-one participants in Europe received a second (boost) dose of 3 x 10(11) DRP tgAAC09 or placebo at least 24 weeks after the first injection . The vaccine was safe and well-tolerated after initial and boost vaccination . Local and systemic reactogenicity was experienced by 13-25 % of participants and was not dose related . No vaccine-related serious adverse events were reported . Modest HIV-specific T cell responses were detected in 7/64 vaccinees (40-385 SFC/10(6) PBMC) , with 16 % (4/25) responders in the highest dose group . All responses were to Gag epitopes . tgAAC09 appears to be safe , well-tolerated , and modestly immunogenic . Further evaluation of higher doses of tgAAC09 and boost injections is ongoing in Africa .

Authors

First name	Last name
Tsuneya	Vets
Tsuneya	Barin

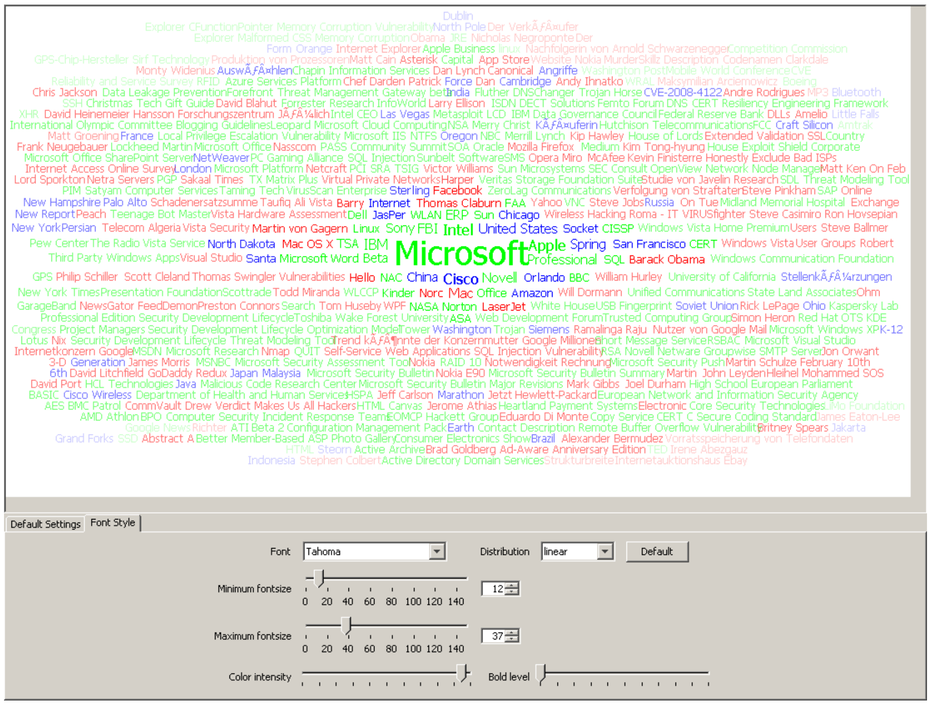
Info

Info name	Info value
Filename	C:\Dokumente und Einstellungen\thiel\NoFileSpecified.txt
Publication date	2008-5-31
Document type	UNKNOWN
Document source	PubMed

Fig. 15. The view of a particular document, showing all its data, such as authors, categories, publication date, text etc.

determine the terms font size, its transparency and boldness. Figure 16 shows a tag cloud containing named entities, such as organizations, locations and persons. It can be seen that the term “Microsoft” is the largest term due to its high frequency which is used as weight value.

Additionally the terms can be colored differently. Therefore the on the one hand the “Color Manager” node can be used which assigns colors to each row according to the values of a specified column. In Figure 16 named entity tags have been extracted as strings and colors have been assigned based on these tag values. Organization are colored green, persons red and locations blue. The tags have been assigned by the “OpenNLP NE Tagger” node.



Font: Tahoma
Distribution: linear
Minimum fontsize: 12
Maximum fontsize: 37

Fig. 16. A tag cloud view.

6 Acknowledgments

Thanks to Pierre-Francois Laquerre and Iris Adä for their work on chi-square keyword extractor and keygraph keyword extractor resp. the tag cloud display.

References

1. KNIME: Konstanz Information Miner. http://www.knime.org
2. Settles, B.: Abner: an open source tool for automatically tagging genes, proteins and other entity names in text. Bioinformatics **21**(14) (Jul 2005) 3191–3192
3. Porter, M.F.: An algorithm for suffix stripping. (1997) 313–316
4. Matsuo, Y., Ishizuka, M.: Keyword extraction from a single document using word co-occurrence statistical information (2004)
5. KeyGraph: automatic indexing by co-occurrence graph based onbuilding construction metaphor. In: IEEE International Forum on Research and Technology Advances in Digital Libraries, 1998. ADL 98. Proceedings. (1998)