Codeless Topic Modelling with KNIME

Annika Hamachers
German Police University, Münster
annika.hamachers@dhpol.de

Abstract

This tutorial walkes the reader through a workflow that employes a topic modelling algorithm - nameliy Latent Dirichlet Allociation (LDA) - in KNIME.

NLP (Natural Language Processing) tasks are one of the big strengths of the KNIME Analytics platform. And as such, also topic modelling and the necessary preprocessing of your data can be easily accomplished with it – particularly since there exists a plethora of already pre-built workflows ready for download on the KNIME Hub. For the purpose of this tutorial, we for instance heavily rely on a workflow created by Julian Thiel and kindly provided for reuse and modification under the creative commons license.

The complete workflow, we adjusted for our purposes and that we will walk though step by step in this tutorial looks like this:

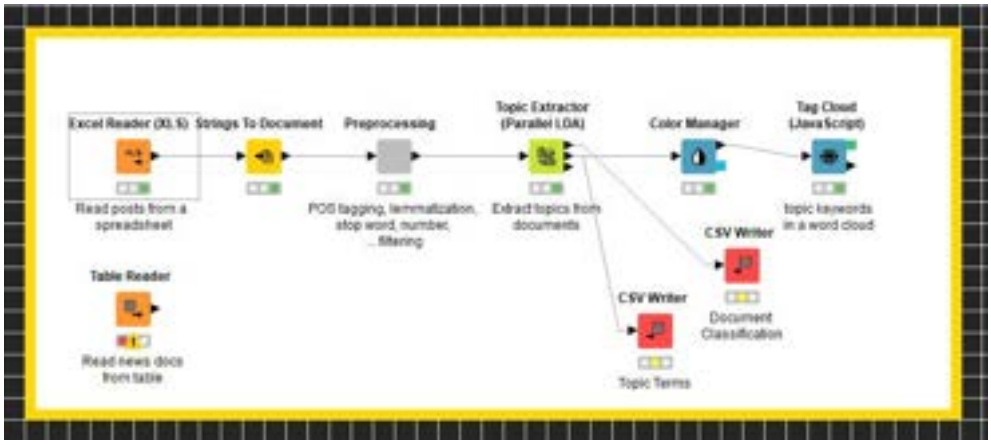


Figure 1: The workflow we will build

The core building block of this workflow is KNIME's **Topic Extractor** node that performs **Latent Dirichlet allocation** (LDA) on a set of distinct texts (called 'documents' in the realm of NLP).

EXCURSE ON LDA

LDA – which was first proposed by Pritchard, Stephens, and Donnelly in the context of population genetics in 2000 and applied to machine learning in 2003 by Blei, Ng, and Jordan – is an algorithm that builds on a generative statistical model for an *unsupervised classification* of texts in order to detect some natural groups of items (aka words) even when we're not sure what we're looking for. Those natural groups can thus be interpreted as underlying semantic structures (aka topics) of the texts. In an extremely simplified way, we can say, LDA treats each document as a mixture of topics, and each topic in turn as a mixture of words. What LDA does is that it calculates a) how prevalent each of the identified topics is in each document and b) how much each word in the entire corpus of texts contributes to each topic. We then filter the most representative words for each topic and can try to make logical sense of them (aka interpret topics in a meaningful way). For more technical details on the algorithms, please refer to the original articles (here and here) or this very insightful Medium post.
Please, be also aware that LDA is not synonymous with topic modelling per se but is just one of multiple possible approaches to choose from – though probably the most common one. LDA will work well in many scenarios. Depending on the type of texts that you want to find topics in, other approaches might, however, be more useful (e.g., fuzzy c means for particularly short texts such as Twitter posts). As already discussed in my general intro to KNIME, only because something can be done easily in KNIME without much background knowledge, this does not necessarily mean that it is particularly useful!

Yet, if you decide, LDA is the way to go, you cannot dive into topic extraction

right away, but have to present thoroughly prepared data to the Topic Extractor node. Thus, the starting node for this workflow needs, of course, to be a reader node – in our case an **Excel Reader** because the data we want to work with are stored in this format. Configuring this reader is easy: Just provide the path to where the data can be found and make sure to check 'Table contains column names in row number' if your spreadsheet has headers like mine. The other settings can be left at their default.

Figure 2: The Excel Reader Node

The node itself nicely shows you the data table it creates from your configurations. In our case we read in posts we gathered from the website of an organization called 'Institut für Staatspolitik'. Those posts promote the publications of this organization which is known for its far-right and German-nationalist world view. We wanted to investigate if these views are also mirrored in topics detected across the supposedly neutral and 'scientific' publications they sell.

However, you can also spot a little 'S' next to the 'Post' column header. This means that KNIME read in 'Post' as a string variable because it detected that it consists of text input. The Topic extractor unfortunately only wants to work with a very specific type of string: a document. And accordingly, the next node is a **String to Document** converter. By telling this node which column to interpret as documents (in our case we only have the column 'Post' available anyway), we get

an updated data table that has been appended by a 'Document' column. The other options can remain at their defaults (even tokenization because we will perform and explain this step later at another node).

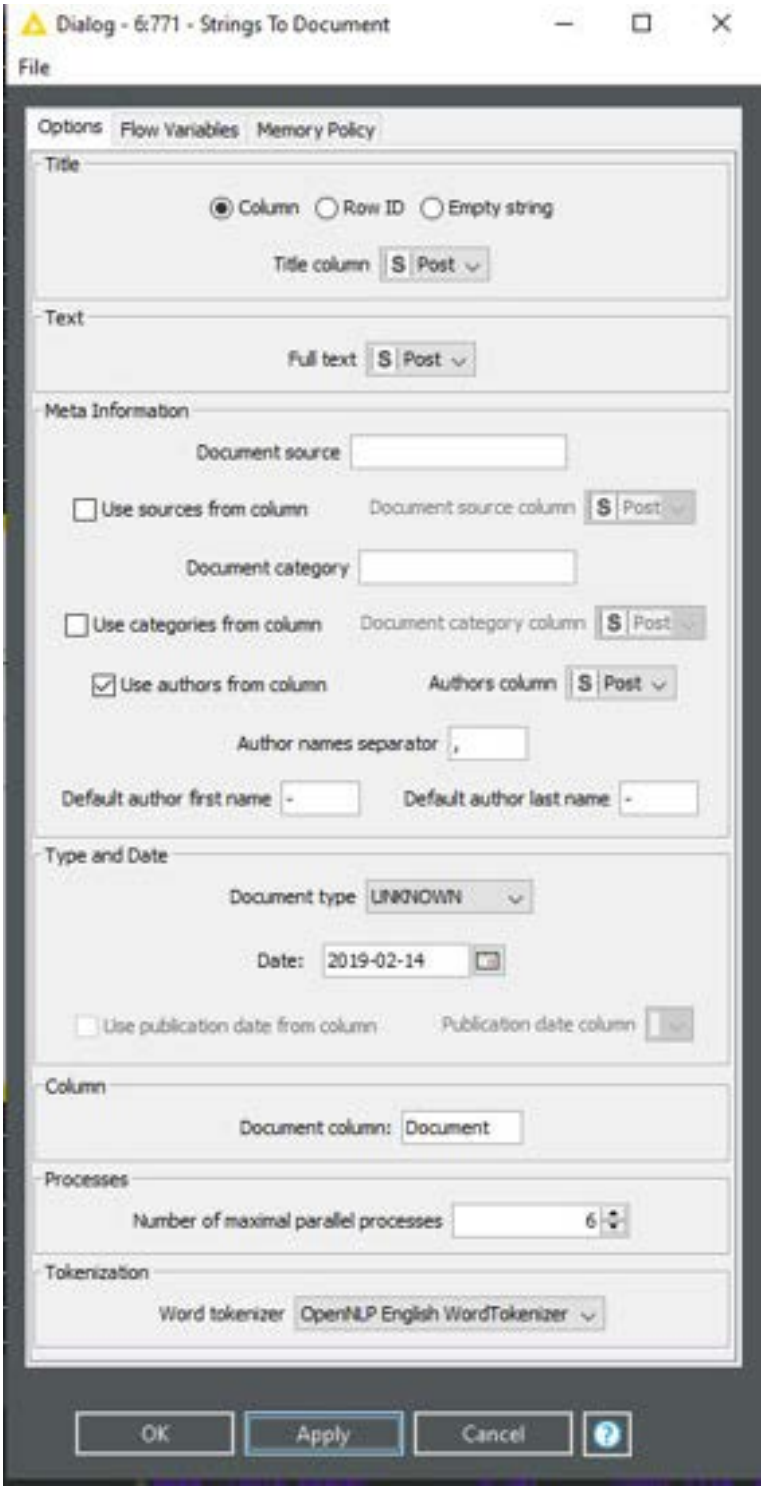


Figure 3: The Strings to Document Node

You can check the result by right-clicking on the node and selecting 'String and

document output table'.

Figure 4: Output of the Strings to Document Node

After this, we encounter a gray node symbol with quite a lengthy description. This is a so-called **meta node** – a wrapper node containing a bunch of other nodes. Such a wrapping makes sense when you have a step in your workflow which contains a lot of different sub steps – such as data preprocessing prior to the actual analysis as in our case. If we were to include them in the main workflow right away, this workflow would become quite overcrowded and confusing. Thus, a meta node is a nice way of ordering things. To access and configure such a meta node, we need to select 'Wrapped Meta Node' and then 'Open' from its context menu.

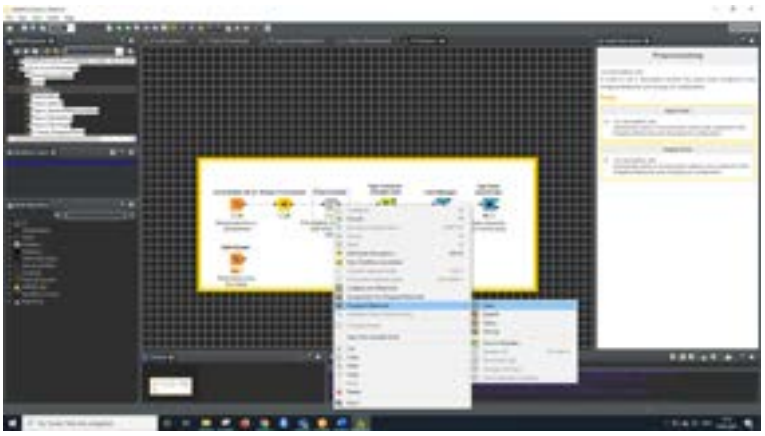


Figure 5: Opening the Metanode

This opens a workflow within the workflow consisting of eight preprocessing steps which we will explain in the upcoming paragraphs.

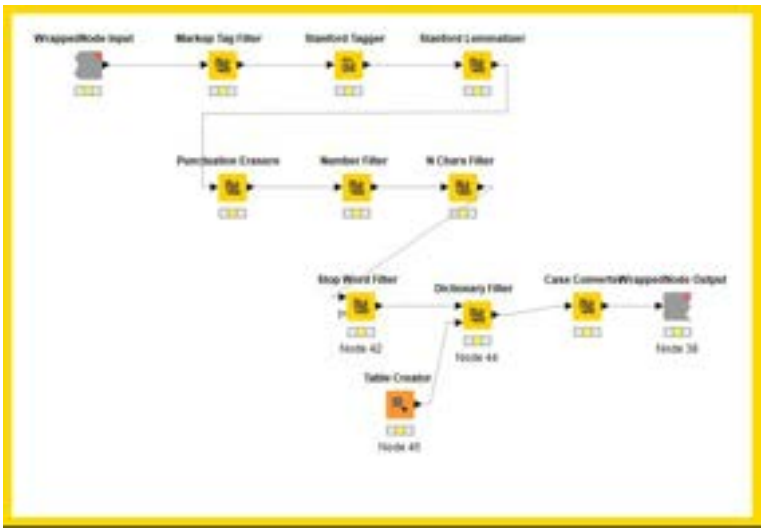


Figure 6: The unwrapped Metanode

The first node you encounter in this chain is the **'Markup Tag Filter'**. The name and description of the node are a bit confusing: It says that it removes all markup language tags which basically just means that it removes all styling from the text and, thus, makes headings, abstracts, body text etc. all the same. Since our input data are not styled in a particular way, at all, choosing this node might appear quite pointless at first glance. However, it is still useful for us because it serves as our tokenizer: By configuring this node with a word tokenizer that recognizes the language of the input correctly, it splits the texts in the document column into its single words which is an important precondition for topic modelling because now reoccurring words (aka tokens) can be counted.

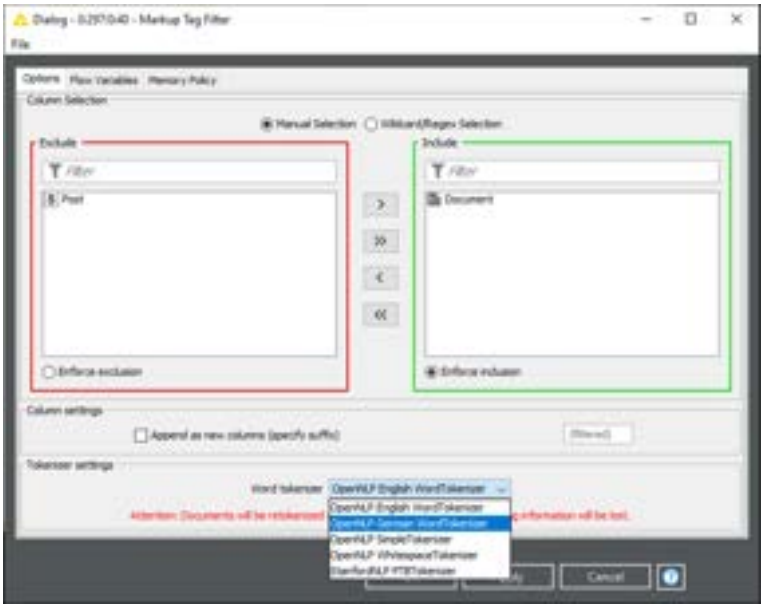


Figure 7: The Markup Tag Filter

If you do not have the right selection options for your language available here, you'll have to install the matching *KNIME Textprocessing Language Pack*. Just go to 'Help' and 'Install New Software' and search for it.

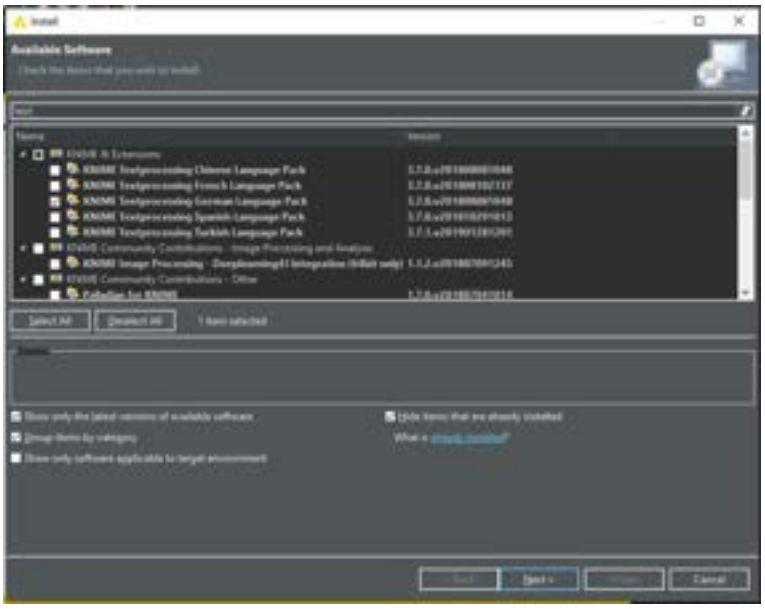


Figure 8: Installing KNIME Textprocessing

And do not worry if your machine seems to lag a bit on installing additional extensions. Some of them are quite big and may take some time to be copied onto your hard drive and unfortunately the progess bar that informs you on the installation status is hidden quite well at the buttom right of the KNIME GUI.

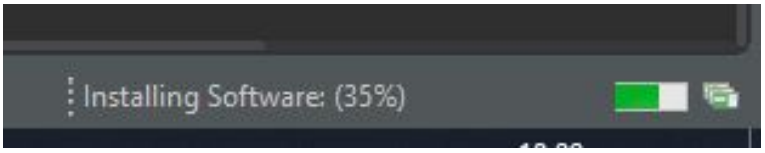


Figure 9: Installation Progress

The next node in line is the **'Stanford Tagger'**. This node assigns a part of speech (POS) tag to each token in the documents. This step can be treated as optional, however, it helps when you want to be able to differentiate words that are spelled the same but fulfil a different function in a sentence (e.g., to tell apart the token 'fly' as in 'I am going to fly to Italy next summer.' and 'fly' as in 'I was annoyed by the fly buzzing around my head'.)

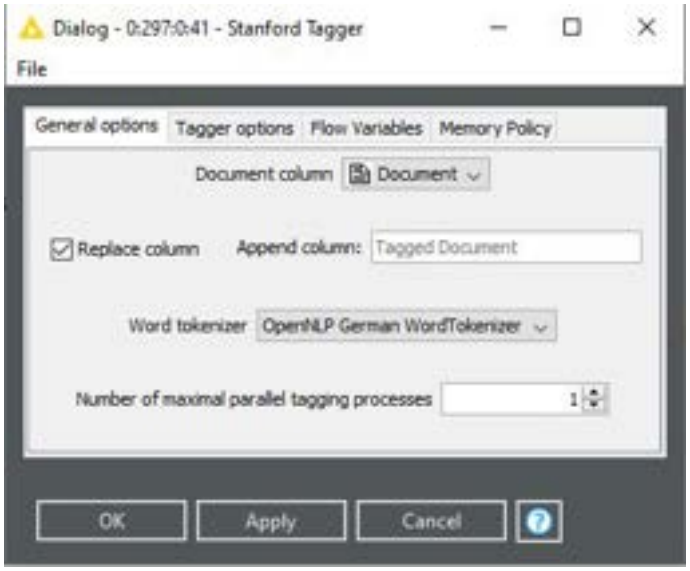


Figure 10: Configuring the Tagger

The **'Stanford Lemmatizer'** is also an optional node and usually not very helpful in cases where you want to process non-English text. This node is useful when you want different variants of a word to be treated as the same (e.g., 'dish' and 'dishes' or 'communicate' and 'communicating') because it truncates the tokens in the documents in order to create the root 'lemma' of each term. Therefore, it removes inflections, e.g., in case of plurals, pronoun case, and verb endings. Since lemma recognition is based heavily on the Part-Of-Speech (POS) tags, either the Stanford tagger node or the POS tagger node has to be applied before using it.

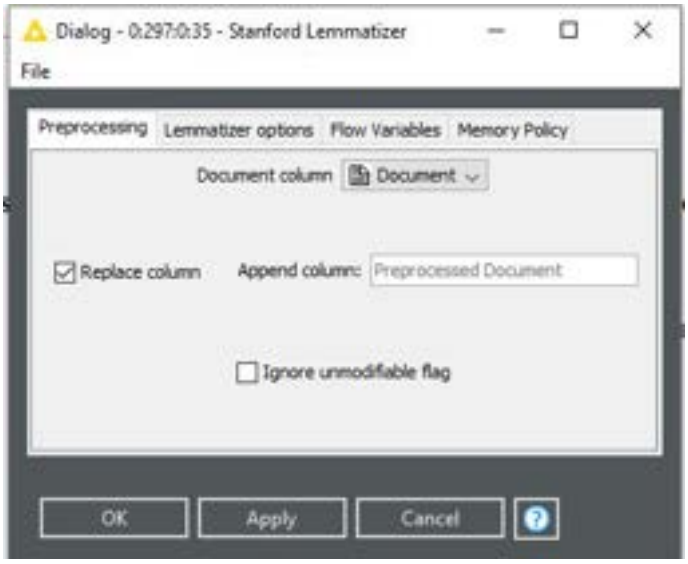


Figure 11: Configuring the Lemmatizer

Next, remove all punctuation (such as ,.:;()/...) from the corpus.

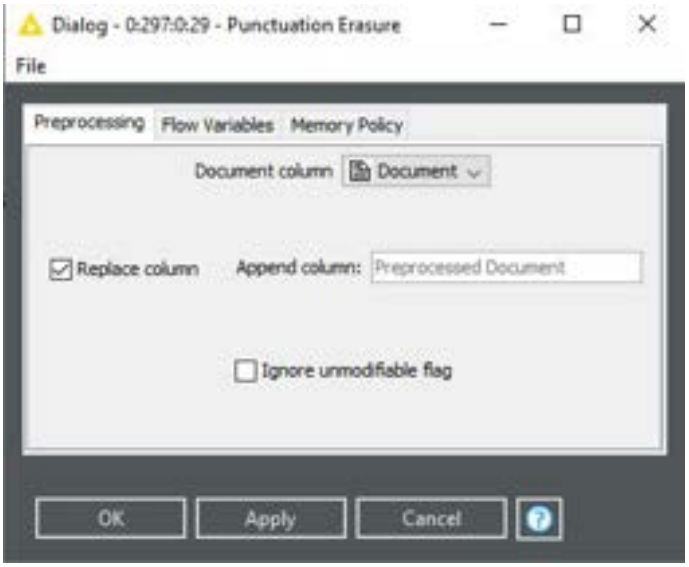


Figure 12: Erasing Punctuation

The 'N Chars Filter' node lets you drop short words from your corpus. For this workflow, I set the filter so that all words will be ignored that do not at least consist of three characters.

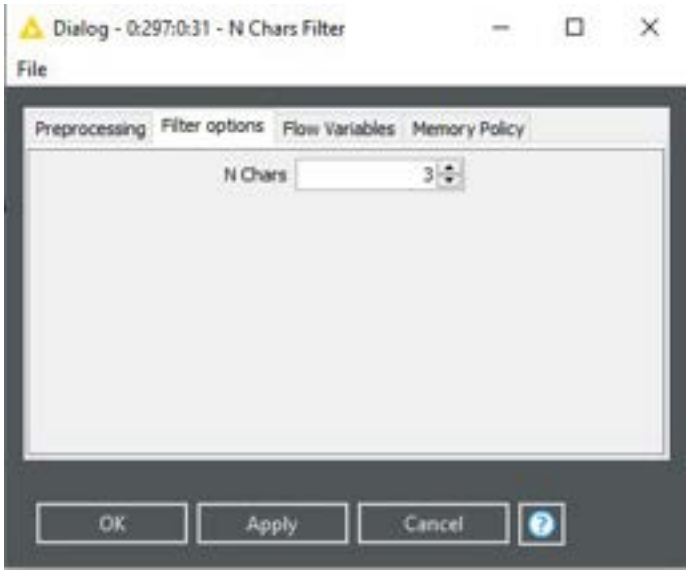


Figure 13: Configuring special characters to be removed

If you are interested in only analysis plain text, you usually also will want to drop numbers and, thus, include the **'Number Filter'**. This node lets you eliminate either tokens that are all numbers ('Filter terms representing numbers') or even tokens that might be a mixture of letters and numbers ('Filter terms containing numbers').

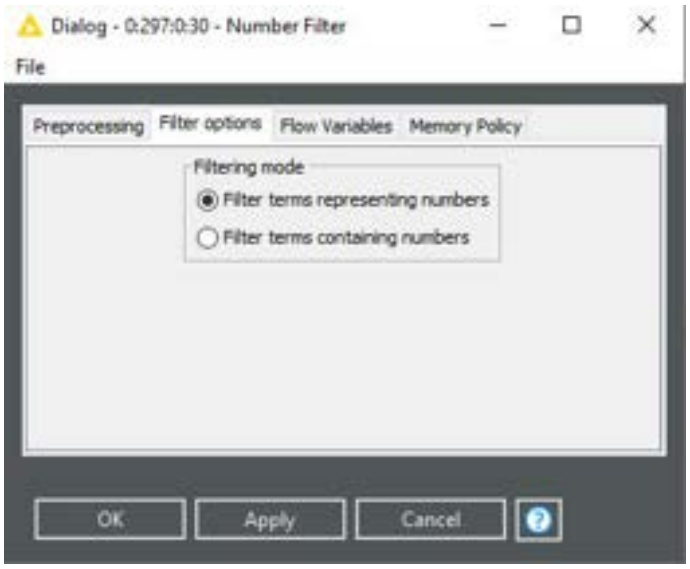


Figure 14: Configuring numbers to be removed

Natural language usually contains a lot of words which do not carry a lot of meaning or are particularly uninteresting for you, so that you would not want them to be considered in your topic analysis because they might dilute the results. Those terms are called 'stop words' in NLP and KNIME has its own node to exclude them. The **'Stop Word Filter'** can be configured basically in two ways: You can either

use an already built-in list with stopwords identified for a specific language and/or provide a costum list with stopwords you gathered yourself.

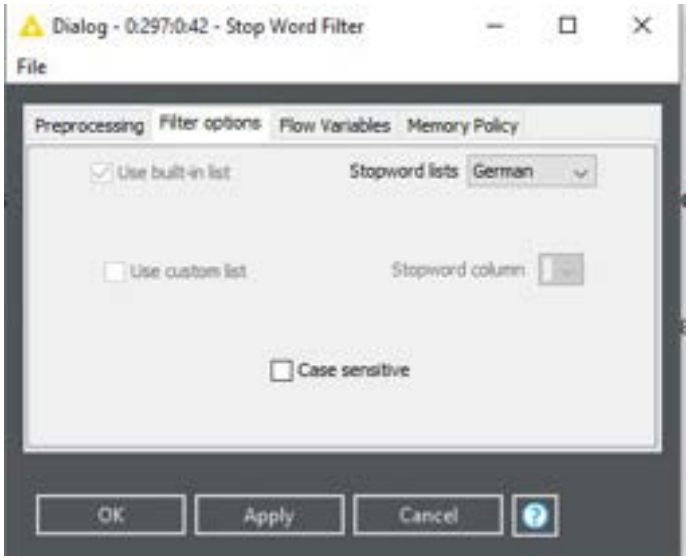


Figure 15: Configuring stop words to be removed

If you choose to use a custom list, you will have to connect a 'Table Creator' node to the second input port of the 'Stop Word Filter' and configure it by adding the terms you want excluded.

Figure 16: Adding customized stop words to be removed

The **case converter node** finally lets you transform the entire text to lower case letters. This is quite useful because otherwise words which are only capitalized because they appear at the beginning of a sentence would be treated as additional tokens.

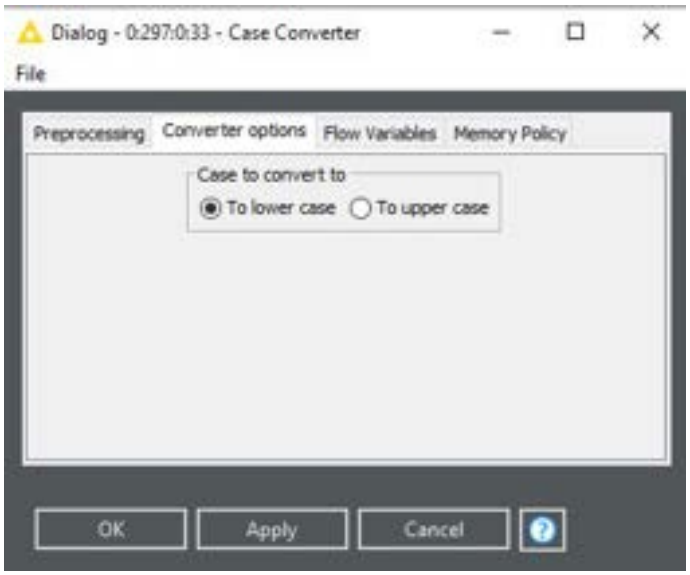


Figure 17: Converting everything to lower case

Now, we are finally done preprocessing our data and can close the meta node again and tackle the main task of this workflow: perform the actual topic extraction with the LDA node. The Flow Variables and Memory Policy panels can be left untouched, again. You will, however, want to specify the **number of distinct topics** the algorithm shall look for and the **number of most relevant words per topic** you want to be able to access later. As for the number of topics, finding an appropriate value here usually involves a bit trial and error: You start with a good initial guess on the number of topics that might be 'hidden' in your text corpus and try some alternative solutions. Then you just pick the solution that provides you with topics that can be interpreted in the most meaningful way.

The additional options can usually be neglected: The **seed** can be left at the default or be chosen randomly. This is just a necessary starting point to draw random numbers from. The **alpha** parameter defines the prior weight of a topic k in a document. Normally a number less than 1, e.g. 0.1, is a good choice to model sparse topic distributions (i.e. few expected topics per document). The **beta** parameter defines the prior weight of a word w in a topic. Normally a number much less than 1, e.g. 0.001, is a good choice to model sparse word distributions (i.e. few expected words per topic). A higher **number of iterations** makes the result of your calculations more robust/reliable but also slows down the performance. 1000 is usually a good value here. If your computer is very slow, you might want to lower it. The **number of threads** is only interesting for parallel computation if you want to speed things up. It lets you declare in how many threads the input document collection shall be split for the calculation Since my machine has eight cores, I choose that number.

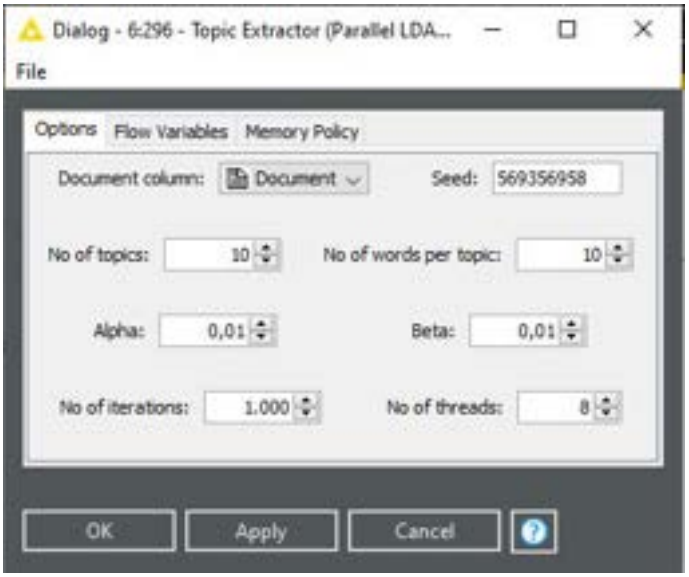
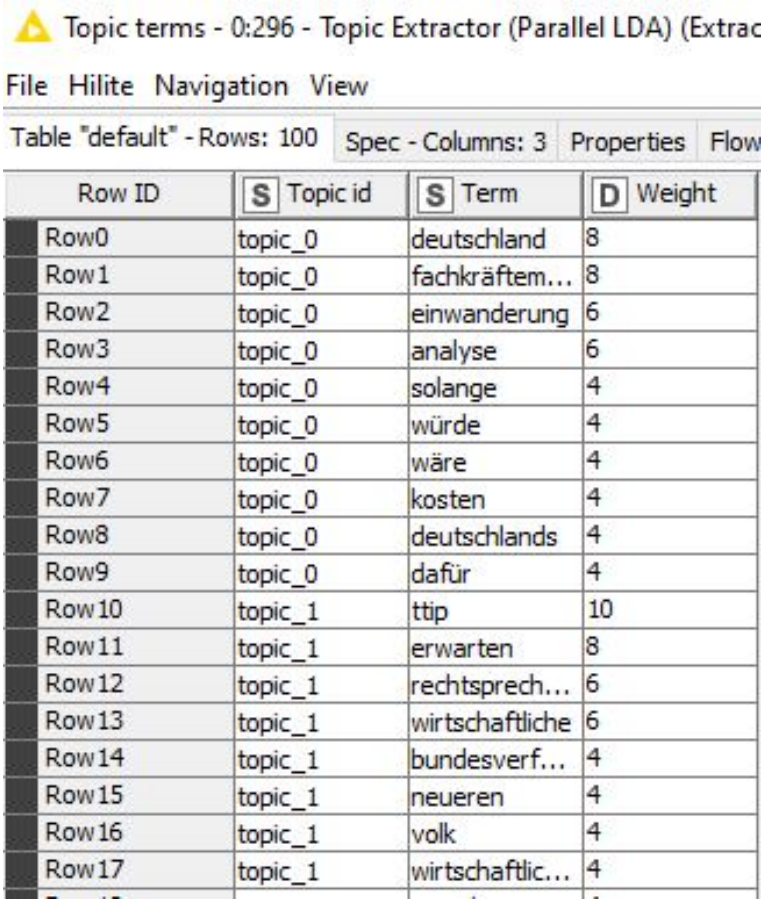


Figure 18: Setting up the LDA

The first output port produces the table for the document classification. Here, KNIME calculates the probability to belong to a certain topic for each document and finally assigns one of the ten topics to each of the posts based on those values.

Figure 19: LDA document classification

The second output port provides us with the topic terms table: Here we see which terms from the corpus contributed the most to each of the topics the algorithm differentiates and thus assigns different weights to.



Row ID	S Topic id	S Term	D Weight
Row0	topic_0	deutschland	8
Row1	topic_0	fachkräftem...	8
Row2	topic_0	einwanderung	6
Row3	topic_0	analyse	6
Row4	topic_0	solange	4
Row5	topic_0	würde	4
Row6	topic_0	wäre	4
Row7	topic_0	kosten	4
Row8	topic_0	deutschlands	4
Row9	topic_0	dafür	4
Row10	topic_1	ttip	10
Row11	topic_1	erwarten	8
Row12	topic_1	rechtsprech...	6
Row13	topic_1	wirtschaftliche	6
Row14	topic_1	bundesverf...	4
Row15	topic_1	neueren	4
Row16	topic_1	volk	4
Row17	topic_1	wirtschaftlic...	4

Figure 20: LDA topic terms

If you want to access those results also outside of KNIME (e.g., when you want to perform additional statistical analysis on them or share them with someone who does not have KNIME), it is a good idea to also incorporate writer nodes into your workflow. In this particular case, I included two **CSV Writers** that save the tables in comma separated format – which can easily be read into R or opened with MS Excel. For writer nodes the default settings are usually quite feasible, all you have to do is specify an output location on your computer (the last part of which is the name you want the file to have) which tells the node where to save your results.

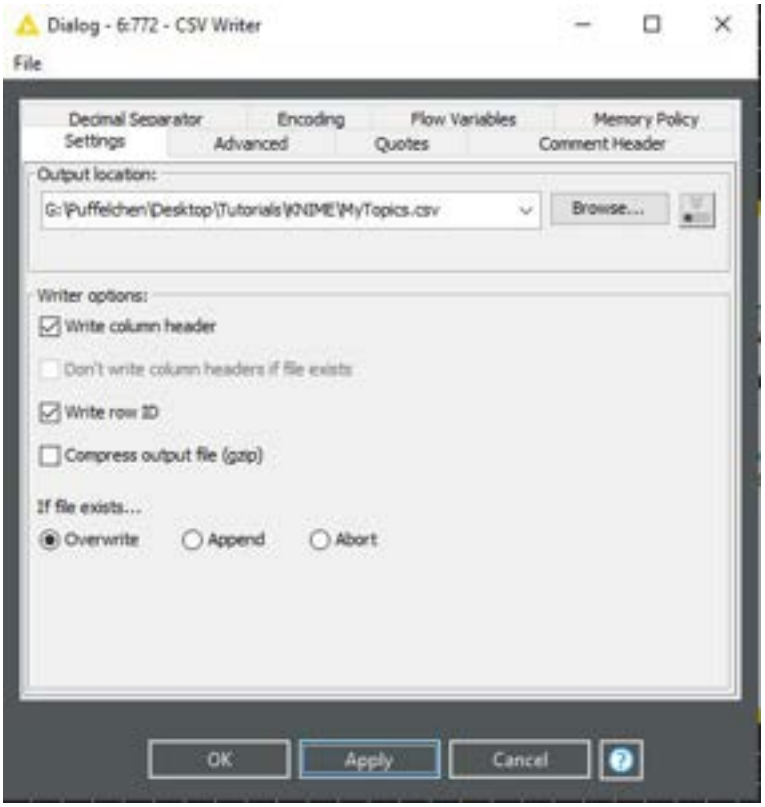


Figure 21: Saving the topics to disc

Now, we have our topics, can interpret them, and elaborate on them. But just as for most analyses, we might also want to visualize the result in a way that is more appealing than a plain data table. A particularly nice means to visualize the topic modelling results is to create a word cloud that displays the terms the algorithm identified as most imported in a circular shape (a cloud), renders their size proportional to their weight and colors them according to the topic they account for.

KNIME has an extra node for such a word cloud (the JavaScript powered **Tag Cloud** node). Yet, in order to incorporate also the last-mentioned feature (the coloring of different topics), we need to push the data through another node first: In the **Color Manager** we define which color is associated with which topic (we can choose from predefined sets or set colors manually with HEX values, RGB values etc.)

Figure 22: Configuring colors for the tag cloud

We then can configure the very last node – the Tag Cloud itself. Here we can play around with several settings such as the maximum number of words to appear in the cloud, the size of the plot or features like the font family or font size range (on the second panel of the configuration dialogue). What is important here, is, however, to select the correct column for the words to be displayed and to display them according to their 'Weight' variable.

Figure 23: Configuring the tag cloud

Now, we have everything set-up and can run the entire analysis. This time, however, we therefore select 'Execute and Open View' from the context menu of the last node:

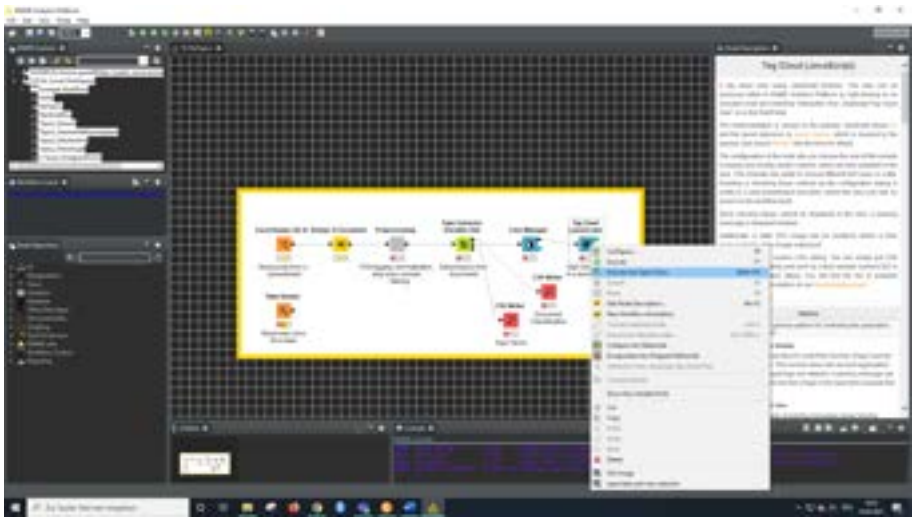


Figure 24: Executing the entire workflow

This way, we do not only get all the calculations done but are provided with an interactive graph that lets us explore the tag cloud we created from the weighted topic terms and lets us make some additional adjustments (e.g., such as switching to another font).

Figure 25: The final tag cloud

If you allready installed KNIME on you machine and it looks different: Do not worry! I changed the appearance of my version to 'dark' via the 'Preferences' section you can access by clicking on 'File'. Moreover, the different panels which are shown by default when launching KNIME can vary a bit depending on the version of your installation (mine is KNIME 3.7.2). If you are missing some of them, please click 'View' in the top left and select additional panels to be displayed.

Beware however that (as we'll see later) some nodes have multiple output or input ports because they can read or produce different types of data. In those cases, you'll have to read the node description and check, which port processes which data and, thus, which one you'll have to choose in order to work properly with other nodes.

Of course, the workflow, we built here, was fairly simple and most of the time you will need to put together much more complex node chains for your purposes. Even with the help of the node description panel this might become quite a tedious task because you would likely have to browse through a lot of nodes and try out several different configurations before the workflow finally does what you want it to do. Luckily, however, you normally do not have to build entire workflows from scratch. There are good chances that somebody might have already done exactly what you are up to (or at least something similar) and that you can download an entire, nicely documented and pre-configured workflow from KNIME Hub. So, ALWALYS search there first before starting your own work!