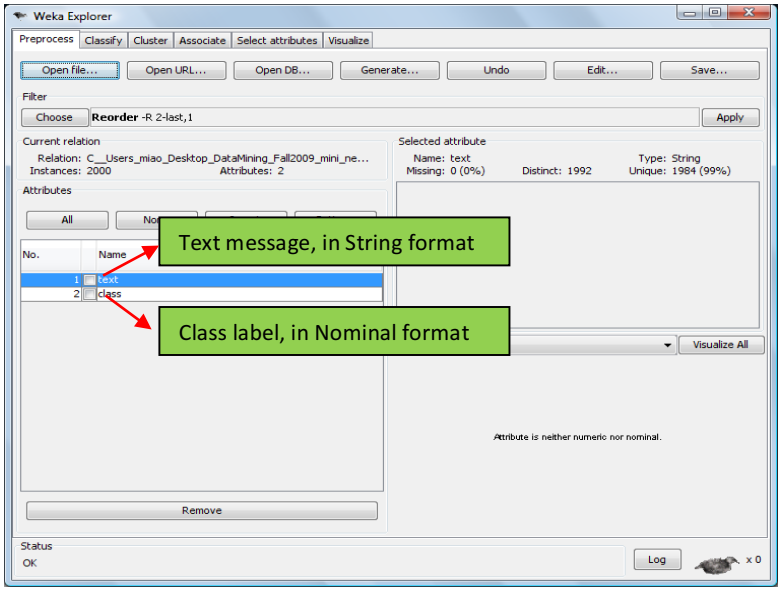
Text Categorization with Weka: Topic Classification

In this demo part, we will use the news group data, “mini_20newsgroup_arff.arff”.

It contains 2,000 files and is a mini version of the whole 20newsgroup data. 20newsgroup data is a collection of posts gathered from 20 news groups on Usenet (an Internet discussion system for knowledge sharing). The posts are labeled with class information i.e. “sci.med” and “sci.space”. There are 20 classes in the data set, representing 20 different topics. The files are mostly messages between users on the news group. See http://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups for more information.

1. Preprocess Data

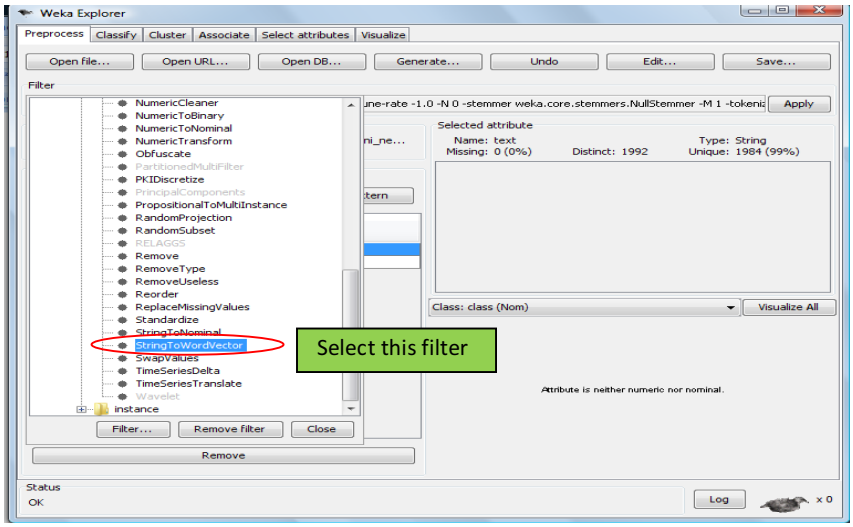
In the Weka Interface, click “Explorer” and import “20newsgroup_arff.arff” on the “Preprocess” panel. After importing, we can see there are two attributes: text and class. “text” is the textual data (messages), and its type is String compared against other attribute formats like Numeric and Nominal; class attributes shows the class label of this text message, and here we have 20 possible values for the class label.



1) Convert “text” from String to Word Vector

The text mining mechanism requires us represent textual data in the vector format. The vector is composed of all words in the collection. We will do it using the “StringToWordVector” filter.

On the “Preprocessing” panel, click on the Filter “Choose” button, and select “weka->filters->unsupervised->attribute->StringToWordVector”.



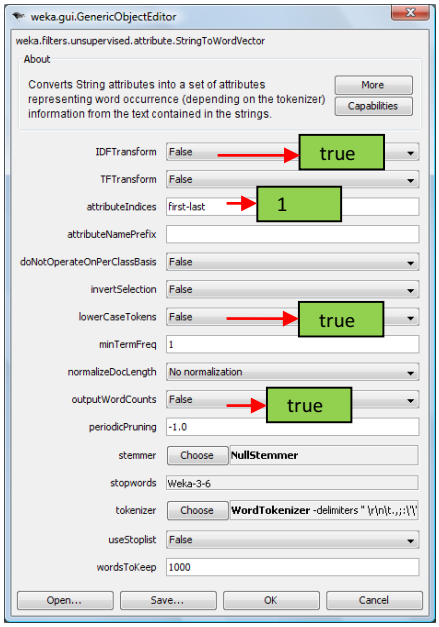
Then set parameters for the StringToWordVector filter. Click on the textbox besides the "Choose" button for Filter to bring up the "weka.gui.GenericObjectEditor". Set the following parameters:

IDFTransform=true (use the IDF weighting for vector values)

attributeIndices=1 (the position of the String attribute)

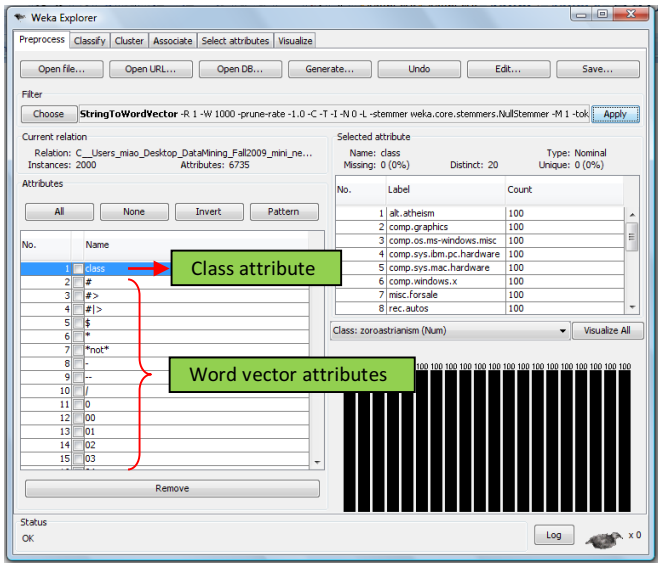
lowerCaseTokens=true (process words to lower cases)

outputWordCounts=true (output count of words, instead of 0s and 1s which indicate existence of a specific word in a document)



After setting the parameters, click the "Apply" button for Filter to run the filter. You will see the resulted attributes like this after converting the "text" attribute to word vectors. Now there are 6735 attributes. The 1st one is the class label, and the remaining 6734 ones are all word vectors

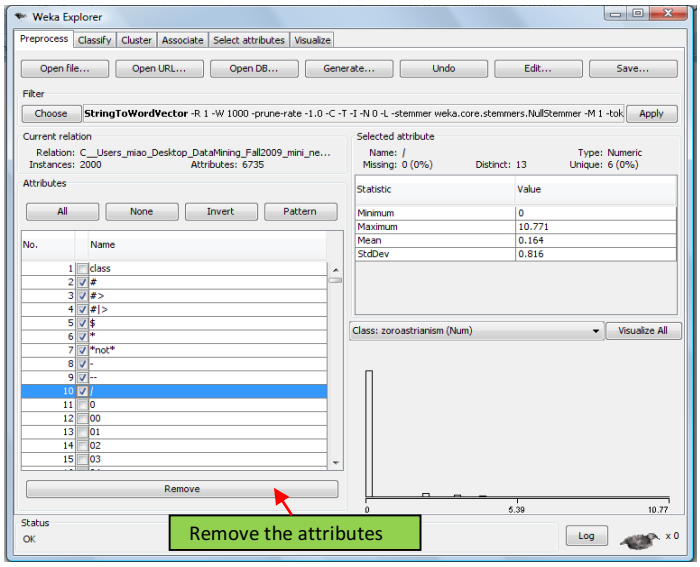
generated from the original “text” attribute. Note after the “StringToWordVectors” preprocessing, the class label becomes the first attribute in the data set.



2) Remove meaningless attributes

By looking closely at the word vector attributes, we may find the 2nd to the 10th attributes seem meaningless. For example, the 2nd attribute is about weighting of string “#” in a document, and the 3rd attribute is about “#>”, which doesn’t make much sense. We can remove these attributes by select them and click the “Remove” button. You can also remove other attributes that you think are useless and see how the classifier performs without these attributes.

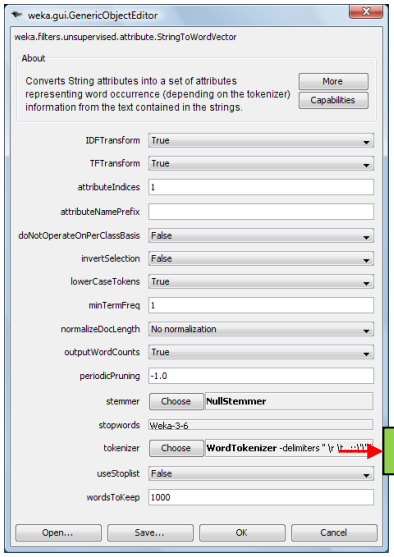
Can you think of any text categorization tasks in which the non-word characters might be useful? One example is the emoticons in social media data. This tells us that whether a token is meaningful or not actually depends on the text categorization task. It is better to use feature selection methods to choose the most relevant tokens rather than arbitrarily remove attributes.



3) Add delimiters

After removing attributes 2-10, when we scroll down to current attribute 55, it shows the attribute is about “<1993apr14”. There is a “<” symbol in the attribute string, and same things happens on attributes 55-69. This is probably because the data is message, and the “<” symbol can be removed by setting delimiters. If we add “<” to delimiters which split up words, then the “<1993apr14” string won’t appear in the word dictionary, and instead “1993apr14” will appear as a string.

To add symbols to delimiters, first we will import the “mini_20newsgroup_arff.arff” again (or use the undo button) since we will do it over again on the raw data. Then bring up the parameter setting for StringToWordVector again. Set other parameters as described in part 1) of this section. In the “tokenizer” parameter, click on the textbox besides the “Choose” button and bring up the “weka.gui.GenericObjectEditor”.

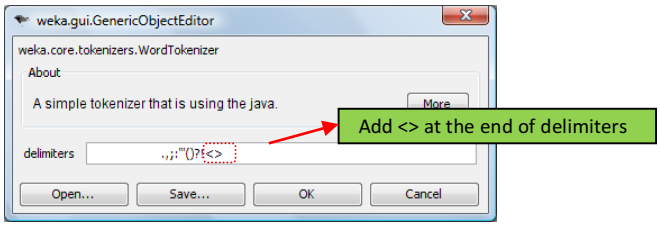


1. Import mini_20newsgroup_arff.arff again

2. Set other parameters as in part 1)

3. Click to edit delimiters

The default delimiters are listed. Add the delimiters at the end of the existing delimiters. For example here we add “<” and “>” symbols to the delimiters (no double quote when writing on the delimiter field). You can add more delimiters as you examine the attributes.



Then run the StringToWordVector filter again by clicking on the “Apply” button. You can see the 70th attribute is now “1993apr14” without symbol “<”.

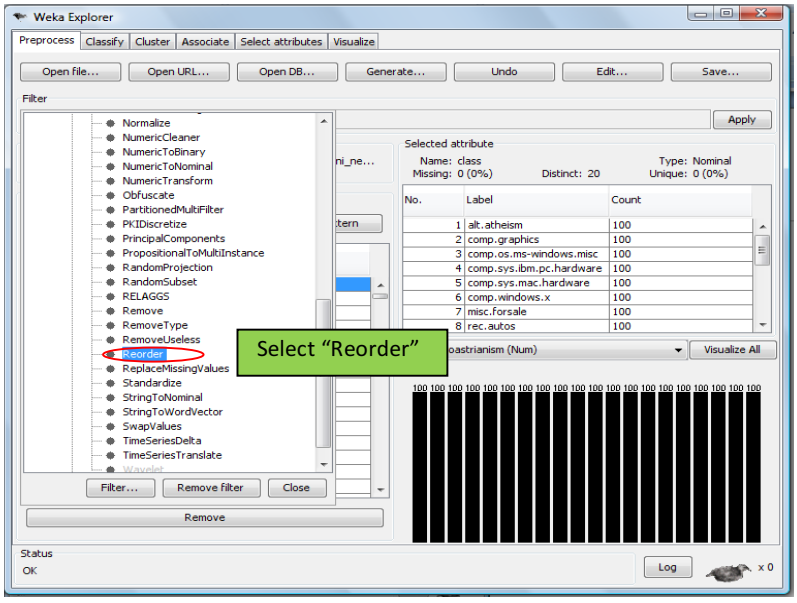
Remove meaningless attributes as described in part 2) of this section. Here let’s remove 2nd-4th attributes for demo purpose.

4) Reorder attributes

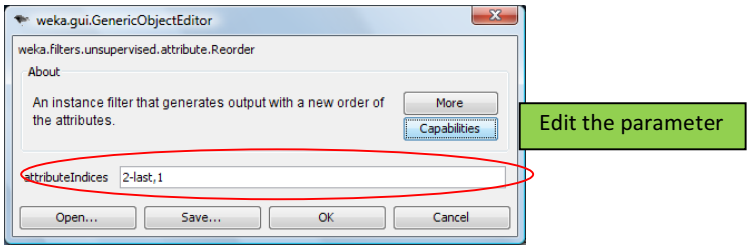
Note that for now the “class” attribute is the first attribute while Weka takes the last attribute for class attribute by default. So we need to reorder the attributes to move “class” attribute to the last position. That is, move the 2nd to last attributes to the front of the “class” attribute.

Click on the Filter “Choose” button, and select “weka->filters->unsupervised->attribute->Reorder”.

You can also change the label attribute under the “classify” tab.



In the parameter setting menu for this filter, set “attributeIndices” to “2-last,1”, which will move the “class” attribute to the last position. Then click the “Apply” button to run the filter.



3. Text Categorization

1) Naive Bayes

By now we have the data ready for running classifiers on. First let's try naïve Bayes. Make sure you choose the NaiveBayesMultiNomial algorithm using the default parameter settings. This algorithm is particularly tailored toward processing text data.

Test this algorithm based on the following data pre-processing options. Note that we change one parameter at one time.

(1) Default setting (word presence and absence)
(2) Change stemmer to LovingStemmer
(3) Change back to default setting, then change outputWordCount to True (word frequency)
(4) then change stemmer to LovingStemmer

Delete the Results to make room for the next run, saving results if necessary.

Document your result into the following table:

parameter setting	overall accuracy
(1) Boolean representation	
(2) Boolean + stemming	
(3) Raw Frequency representation	
(4) Raw frequency + stemming	

Table 1. using Multi-nominal Naïve Bayes model to categorize news groups

Note that although Weka provides tfidf option for this algorithm, actually according to both (McCallum and Nigam 1998) and (Mitchell 1997). Multi-nomial Naïve Bayes text classifiers should not take tfidf or normalized tf values as input.