Advanced Text Lexical Analysis Software

J.P.Redonnet, Nov 26,2008

1 – Introduction

The goal is to create a complete set of tools to analyze a text, enough complete to be useful for everybody. Mainly, this is the opportunity to study the integration of different algorithms into a single program, to experiment and improve the machine learning algorithms (perceptron, winnow), to experiment different fuzzy string matching algorithms (edit distance, sound like) and word singularization algorithms.

2 – Program name

This is a set of programs, named 'TexLexAn'

3 – Objectives and possibilities

3.1 – Analyse any text to determine some commons informations such as:

- The difficulty to read/understand.
- The reading time.
- The repetition of words.
- The percentage of basic/common words.
- The distribution of words per number of syllabes.

3.2 – The most interesting function of this software is the automatic classification of texts.

- The text is automatically classified in predefined categories, such as: protocol, law, literature thriller, literature scifi, poem, (and/or) complain, agreement, procedure, proposal, technic, scientific, juridic, treat...
- Plagia can be detected by searching inside a dictionary of famous sentences sorted by authors.

3.3 – Other functions.

- The character coding (utf8:Unix and utf16:Windows) is detected.
- This program recognize automatically the main language used in the text (this version implements the English and the French languages, but can be easily extended to any language using the ascii characters.)

3.4 – Futur.

- Automatic summarization will be added.

4 – Applications:

- Evaluate the quality of a document (readibility, repetition, main ideas)
- Redirection of email to the proper service. Complain goes to the customer service, resume to H.R., commercial to marketing...
- Anti-spam.
- Better web site analyse for indexation.
- Illegal or reprobated email detection (terrorism, pedophile, drug...)
- Automatic presorting of text submitted to the editors.

5 - Principle

The text analysis goes through 10 steps:

1 - Characters coding detection and conversion in ascii.
2 - Language detection and automatic selection of the dictionary.
3 - Tokenizing (text divided in words & sentences are marked).
4 - Syllables and basic words counting.
5 - Suppression of low significant words (stopwords).
6 - Extraction of each term and its occurrence in the text.
7 - Conversion of plurals in singulars.
8 - Searching each term inside a dictionary and weighting each class.
9 - Sorting each class by weight in the text.
10 - Displaying / recording the results.

NOTE: A term and a 'keyterm' can be a single word or keyword, or a group of words where their order in the group is important. For instance: 'new jersey' , 'happy new year', 'world wide web'

6 - Learning feature

In supervised learning, 'the learning loop' allows the user to update the dictionaries with new classes, new terms, or to consolidate the weighting. Each text analysed provides a list of 'keyterms' (single words & group of words) with their occurrences in the text that will be used by the program to enrich its 'knowledge base'. In present time, an human operator is required to provide the class label.

The futur versions should include an unsupervised learning algorithm allowing the program to learn by itself after the initial supervised learning. The expert will have to validate or to correct the weak assumptions made by the classifier.

Note: The learning loop can work as a feedback when the operator provides the correct class label of a mis-categorized text.

7 - The Classifier - Brief description

7.1 Linear classifier

This is the most interesting part of this program; the classification or categorization is generally considered as part of the artificial intelligence.

Classifiers can be splitted in two several categories, we will consider here only the supervised and the unsupervised learning classifier.

a) The supervised learning consists to train the system with a group of data already labelled and called vectors (texts precategorized in our case). The classification algorithm estimates the best coefficients of a mathematical model.

b) The unsupervised learning consists to let the program to cluster the corpus of uncategorized (unlabeled)examples. A Maximization-Expectation algorithm is generally used to group texts sharing the maximum of similarities. Then the clusterized texts can be used to build a mathematical model.

c) The semi-supervised learning starts with the minimum of data already labeled (texts manually categorized) and continues its training with the main corpus of text unlabeled. This technics allows to save the time of an expert charged to categorize manually the training sets of texts, and to get a better accuracy of the classification than the unsupervised learning classifiers can provide.

The goal of training of supervised, unsupervised or semi-supervised classifier is to build a model that will be used to class a new text. The simplest mathematical model is a polynomial equation of the first order:

$$f(x) = a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_n \cdot x_n + b = \sum_{j=1}^{n} a_j \cdot x_j + b$$

The simplest classifiers attribute one polynomial of first order to each class. With the assumptions, there are no interactions between each terms x_j and the relation is purely linear between each term the score f(x). The problematic is to determine with maximum of robustness the coefficients a_i (called weights). The well-known linear regression is one of the methods used.

Bayesian independence classifier categorizes a document by estimating the probability that a set of terms extracted from the text belongs a class. The Naïve Bayes assumptions that the terms i are independents allows the simple relation:

$$Pr(d|c_i) = \prod_{j=1}^{n} Pr(w_j|c_i)$$

Where the probability that a text d belongs to the class c_i is the product of the probability of each term w_j in the text belongs to the class c_i.

Unfortunately in practice, the assumption that terms are independents is rarely verified, and the probability $Pr(d|c_i)$ is very often inaccurate. But because the Naïve Bayes classifier simply attributes the value TRUE to the class with the highest estimated probability, this method finally works pretty well regarding its simplicity.

$$C = argmax\left(Pr(c_i) \cdot \prod_{j=1}^{n} Pr(w_j|c_i) \right)$$

Where C is the class attributed to the document.

The probabilities $Pr(c_i)$ and $Pr(w_i|c_i)$ are estimated from the training texts.

See [1] for general informations on Text classification.

7.2 Implementation

The study of Gärtner and Flash [2] shows the Naïve Bayes performs pretty well in many cases; and Rennie, Shih, Teevan and Karger [3] describe several technics that correct some deficiencies in the application of the NB classifier, improving significantly its performances. Ng Andrew and Jordan [4] shows that the NB classifier often converges faster than the logistic classifier, and Zang H. [5] presents a novel explanation of the performance of NB.

In order to make this application usable on the majority of the microcomputer; a simple linear classifier derived and adapted of the Naïve Bayes (NB) has been chosen for its rapidity, its good performances if the weights are carefully calculated, and its simplicity.

Computation of the scores:

The simplest linear function is used to compute the score y_i of each class i. x_j is the number of occurrence of the term j in the text and a_{ij} its weight of in the class i.

$$y_i = a_{i1} \cdot x_{i1} + a_{i2} \cdot x_{i2} + \ldots + a_{in} \cdot x_{in} = \sum_{j=1}^{n} a_{ij} \cdot x_j$$

Note: The highest score corresponds to the most probable class. Scores computed with the number of occurrences are dependant of the document length; because they are all biased in the same proportion their ranks are corrects.

The program ranks the scores from the highest (most probable class) to the lowest (less probable class) and computes their normalized values those are independents of the document length.

$$Y_1 = \frac{y_i}{\sum_{1}^{n} y_k}$$ n: last class

Computation of the weights:

Weights are computed from a set labeled texts (training texts).
- In case the training texts are approximately of the same size:

$$a_{ij} = \frac{x_{ij}}{\sum_{k=1}^{n} x_{kj}}$$

- In case the classes do not have the same size of training texts, it is important to normalise the weight:

a_m Weight maximum found.

$$a'_{ij} = \frac{a_{ij}}{a_m} \cdot \frac{L_m}{L_i}$$

L_m Size of the training texts for the class m (maximum a)
L_i Size of the training texts for the class i.

Salton and Buckley [6] present several typical term-weighting formulas and compare their performance. It appears that the best fully weighted system and the best weighted probabilistic weight (1) provide the best results for their collection of texts. They note the importance of the length normalization too.

Best weighted probabilistic weight (1): $a_{ij} = 0.5 + \frac{0.5 \cdot x_{ij}}{x_m}$

x_m the maximum of occurrence of a term m

It is obvious the weighting formula is important in the classification accuracy and it appears the performance of a formula depends of the characteristics of the texts analyzed, such as the length, the number of different 'keyterms', their frequency and noise in the texts. It seems interesting to compute several weighting formula, then to choose one would fit the best to the characteristics of the documents analyzed.

Note: The version 1 of this program only works with positive weight values, but it is possible to use negative weight too. This case will mean that some 'keyterms' may decrease the probability to have a class considered.

Text and Lexical transformations:

The classifier cannot work on a raw text. The first step is the detection of the main language of the text. The next step is the tokenization where each words are extracted. During this step, any special single characters, punctuations marks, and articles are discarded. Uppercase characters are converted in lowercase and finally plurals are converted in singulars.

Bags of terms:

All terms in the text are regrouped and their occurrences computed. So The position of each term is the text is lost. The classifier works on the concept of a bag of terms.

String matching:

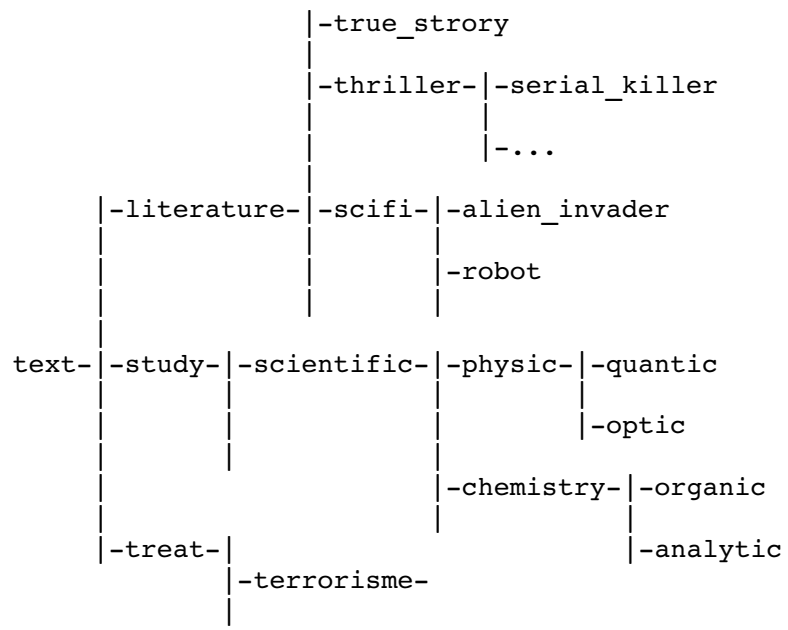
Each term of the bag of term is searched inside a 'dictionary' to retrieve its class appartenance and its weight in this class. For this search, there is two possibilities of terms matching:
- A strict matching. Its advantage is to be fast but misspelled terms cannot be found in the dictionaries. Because misspelled terms do not participate, the scores can be biased.

- A fuzzy matching. Misspelled terms can be retrieved, but the algorithms are slow and can introduce a bias in case of confusion between two terms of close spelling. In this program, the Levenshtein algorithm is used to compute strings distance and find the closest term in the dictionary. It is an o(mn) algorithm, performant but time costly when both strings to compare are long.

8 – Data structures

File mame: keyworder.'lang'.dic'n'
(lan:language abbreviated, n:number of words in keywords)
Function: dictionaries weighing each keyword
Structure: Class_number Class_name:/W_1\term$_1$/W_2\term$_2$/..../W_i\term$_i$/
The sequence /...\ delimits a number(the weight of the keyword)
The sequence \.../ delimits a group of words (a keyword).
Size: ~20KBytes/class (2 words/term) => File size ~10MBytes for 500 classes.

Tree structure of the class name



9 – Programs

The job is divided in 5 programs, allowing more flexibility, requiring less resources and facilitating the debugging. The text analyzer 'texlexan' and its graphical frontend texlexan.py are designed to run on a workstation (document analyze). The text analyzer alone can run on a server (emails, blogs screening...). These programs could be released under the GNU public licence, allowing the community to improve them.

9.1 Graphical frontend
Program name: texlexan.py
Function: Provide a confortable GUI between the program texlexan (runs in CLI) and the user. Able to produce some graphics
Interface: graphic
Language: python + (pygtk)

9.2 Text Analyzer
Program name: texlexan
Function: Analyze a text. Recognize & convert charset. Recognize main language & open the corresponding dictionaries, select the simplification & the plural to singular conversion rule. Compute syllabes distribution and text readability. Estimate reading time. Compute repetition.
Search & count same sentence occurrence. Weight keywords and finally list classes
(text theme) per probability.
Interface: CLI (command line interface)
Language: c
Input Files: filename.txt (text to analyze) , keyworder.'lang'.dic'n'

Output Files: keywords.build (records are added at the end)
(lan:language abbreviated, n:number of words in keywords)

9.3 Dictionaries Builder

Program name: buildkeybase
Function: Split the keyword.build file in several dictionaries. There is one dictionary for each class in each language for each number of words allowed in one term.
Each dictionaries are updated: occurrences increased, terms added.
Nb of files = Nb of classes x Nb of languages x Nb of words/term
Interface: CLI
Language: c
Input Files: keyword.build
Output Files: text-'classname'.'lang'.dico'n'
(lan:language abbreviated, n:number of words in keywords)

9.4 Temporary Dictionnaries Builder

Program name: globalkeybase
Function: Convert plural in singular, regroup terms in a single file.
There is one dictionary for each class in each language for each number of words allowed in one term.
There is one global dictionary for each class x each language x number of words per term.
Nb of files = Nb of classes x Nb of languages x Nb of words/term + Nb of languages x Nb of words/term
These dictionaries are temporaries and used by statistic analyzer.
Each dictionaries are recreated.
Interface: CLI
Language: c
Input Files: text-'classname'.'lang'.dico'n' , text-generic.'lang'.dico'n'
Output Files: text-'classname'.'lang'.dico'n'.work , text-full.'lang'.dico'n'.work
(lan:language abbreviated, n:number of words in keywords)

9.5 Statistic analyzer

Program name: analyzekeybase
Function: Compute the weights, keep the most significant 'keyterms' (threshold on the weights and the occurrences), normalize, record the final dictionaries.
There is one final dictionary for each language x number of words per term.
Interface: CLI
Language: c
Input Files: text-'classname'.'lang'.dico'n'.work
Output File: keyword.'lang'.dic'n'
(lan:language abbreviated, n:number of words in keywords)

Coding languages prefered:
- Python to manage the graphical interface. This language is pretty slow (bytecodes interpreted) but offers an excellent productivity, is easily portable and many powerful libraries (graphics, statistics...)
- C (C/C++) where speed is required.

10 - Improvements

Optimization:
- The strict search uses the strstr function from the standard c library. It is an O(m*n) algorithm; the Boyet-More or Knuth-Morris-Prath O(m+n) algorithms should be faster.
- The fuzzy string matching algorithm uses the Levenshtein distance O(m*n); Cole R. and Hariharan R. [7] give two fastest algorithms for finding approximate matches $O(nk^3/m+n+m)$ and $O(nk^4/m+n+m)$
- A sound like string matching algorithm [8] (double metaphone) should give some interesting results, the difficulty is to implement one for every language.

Learning algorithm classifier:
- Increase the weights of the terms found when the operator agrees the text categorization (answer=YES) and decrease the weights when he refuses it.

Keyterms list:
- Limit the list of 'keyterms' to the most significants.

Dictionnaries sharing:
Collect via internet all the 'keywords.build' files (text name, 'keyterms' list and frequency, class labelled) and merge them into a single large 'keywords.build' file. This large corpus of preprocessed texts will used to create the dictionaries ready to use and to download.

Automatic summary:
Add a summarizing algorithm to the TexLexAn to complete the tools set of TexLexAn. Pardo, Rina and Numes [9] use a combination of sentences scoring and keywords to select the sentences to include in the summary. This method should be easily implemented by reusing the sentences segmentation and keywords, both already exist in the texlexan program.

11 - Conclusion

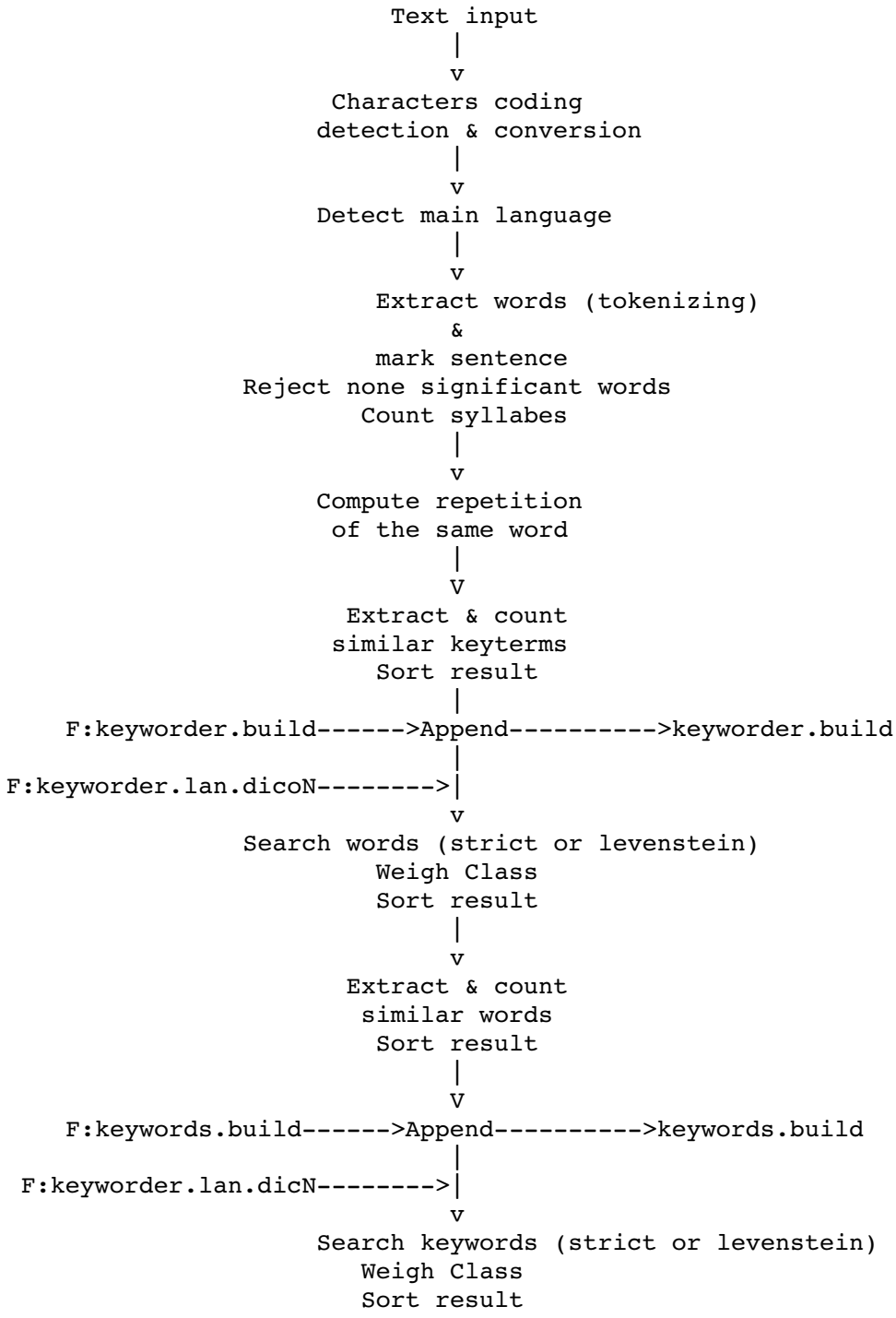
These programs are at their early developments, but theirs possibilities appear promising and should interest a lot of people working on many text documents.

There are still a lot of work to improve the texts classification, in particularly, to find a fast and good fuzzy string matching algorithm, to have a good 'keyterms' weightings system, to make the program able to learn by itself with the strict minimum of supervision. So certainly, the most difficult part of this program is the texts classifications but its the most interesting too.

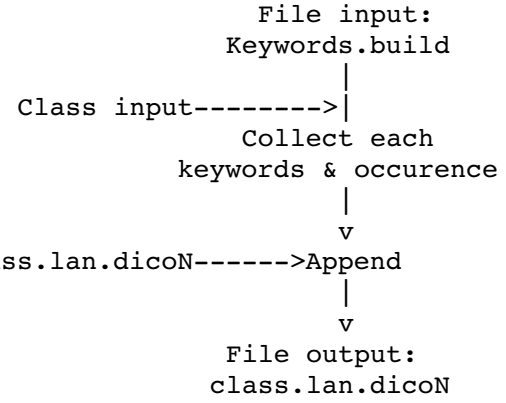
Finally, we did not find under the GNU licence a software including a plagiarism detector, text classifier, summarizer, text readability and reading time, so this one could be the first one.

10 – Annexes
10.1 – Flow charts

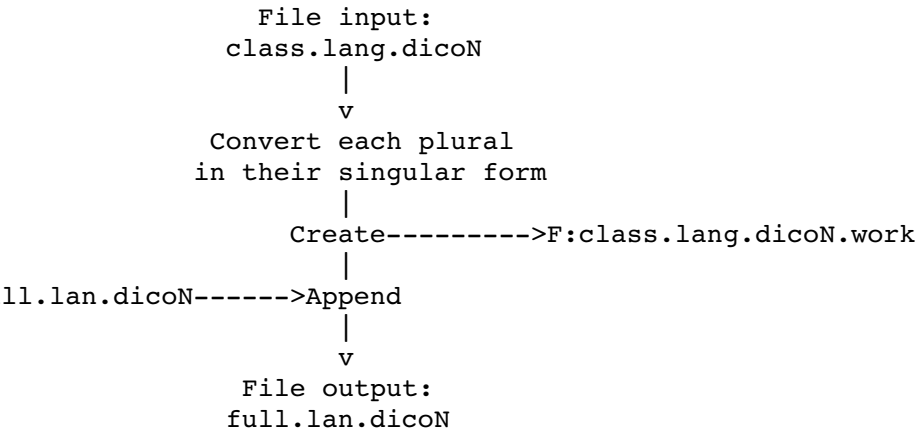
Program: TexLexAn



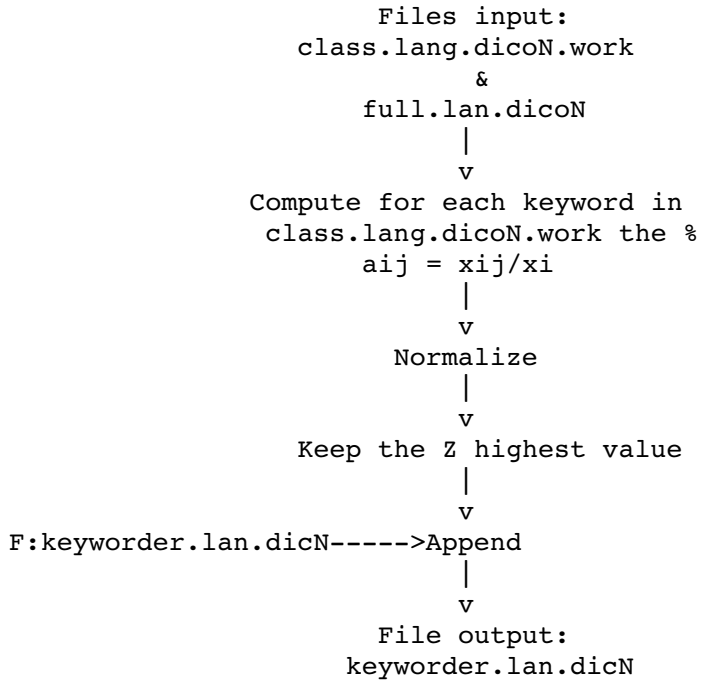
Program: buildkeybase



Program: globalkeybase



Program: analysekeybase



Learning loop:

The text to analyze is an ascii/utf8 file.

keyworder.lan.dico.N
contains classes, keyterms & weights

texlexan is the main program

Results are displayed and/or recorded. The operator enters the class / validate the results

keywords.build is updated with every text analyzed. It contains all the terms and theirs occurrences

buildkeybase places keyterms + occur. in each class.

class.lan.dicoN contains keyterms + occur. One file per class x language x nb of words / term.

lan:language
N:nb of words/term

globalkeybase converts plurals in singulars => occur. are updated. Regroups all the class for one language, one size of terms in 'full.lan.dicoN

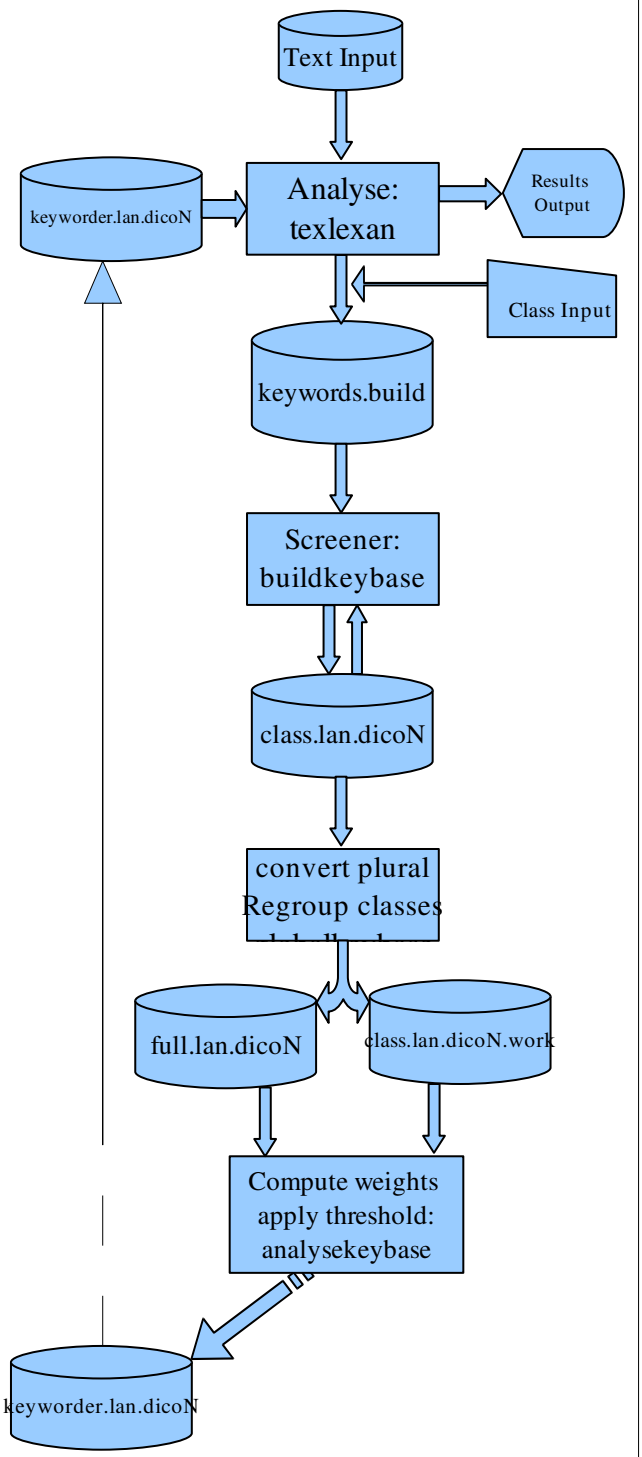
analysekeybase computes the weights, the threshold limits the number of terms per class to most significants.

keyworder.lang.dicoN structure:

class_name:/weight\keyterm/....

Ex.keyworder.en.dico3 =>

text-literature-scifi:/9\brave-new-world/.../7\light-year-away/



10.2 References:

[1] Ikonomakis M., Kotsiantis S., Tampakas V. Text classification using machine learning techniques. WSEAS TRANSACTIONS on COMPUTERS, Issue 8, Volume 4, pp 966-974 (August 2005)

[2] Gärtner T., Flash P. Weighted Bayesian Classification based on support vector machine (parag. 4 Comparison) Proceeding of the eighteenth International Conference on Machine Learning (ICML-2001) http://www.cs.bris.ac.uk/Publications/Papers/1000560.pdf

[3] Rennie J., Lawrence S., Teevan J., Karger D. Tackling the poor assumptions of naive Bayes text classifiers. Proceeding of the twentieth international conference on machine learning (ICML-2003)

[4] Ng Andrew, Jordan Michael On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.9829

[5] Zhang H. The optimality of naive Bayes (2004) American Association for Artificial Intelligence.

[6] Salton G., Buckley C Term-weighting approches in automatic text retrival. Information Proceesing & Management Vol 24, N° 5, pp 513-523 (1988)

[7] Cole R., Hariharan R. Approximate string matching: A simpler faster algorithm SIAM J. COMPUT. Vol. 31 N°6 pp 1761-1782 (2002)

[8] Zobel Z., Dart P. Phonetic String Matching: Lessons from Information retrieval. http://goanna.cs.rmit.edu.au/~jz/fulltext/sigir96.pdf

[9] Pardo T.A.S., Rino L.H.M., Nunes M.D.G.V. Extractive summarization: how to identify the gist of a text. International Information Technology Symposium (2002) www.dc.ufscar.br/~lucia/articles/I2TS2002-PardoEtAl.pdf