I-SG : INTERACTIVE SEARCH GROUPING

SEARCH RESULT GROUPING USING INDEPENDENT COMPONENT ANALYSIS CLASSIFICATION

Thomas Lauritsen and Thomas Kolenda
Department of Mathematical Modeling, Building 321
Technical University of Denmark, DK-2800 Lyngby, Denmark
Phone: +45 4525 3920
E-mail: tl@ingolf.dk,thko@imm.dtu.dk

Abstract. We present a computational simple and efficient approach to unsupervised grouping the search result from any search engine. Along with each group a set of keywords are found to annotate the contents. This approach leads to an interactive search trough a hierarchial structure that is build online. It is the users task to improve the search, trough expanding the search query using the topic keywords representing the desired groups. In doing so the search engine limits the space of possible search results, virtually moving down in the search hierarchy, and so refines the search.

INTRODUCTION

In application of assisting the Internet WWW user while using a search engine, we purpose an interactive search that groups the search result of a given search engine into underlying topics and suggests keywords for further refinement of the search query.

The classification into topics is done using *independent component analysis* (ICA), that has shown to be a natural basis for representing text [4]. The number of classes are found with a prior on not having two many classes, since this might confuse the user. The ICA classification supports a hierarchial structure depending on the number of classes, thus separating its contents at various context levels. Separating into only a few classes does therefore not limit the search, thus lets the user have a better understandable overview of the search, only on the expense of more user iterations through the search.

In this application we demonstrate the ability of the ICA algorithm to separate text into the underlying topics. We annotate the topic group with ranked keywords that describe each topic best, and further give an alternative webpage ranking that describes how close a webpage is to the class it has been grouped with.

Getting started:

Read the text file `README.txt` for installation and execution instructions.

This project has been made in our spare time and we thank the Department of Mathematical Modeling at the Technical University of Denmark for letting us use there facilities.

MODEL FRAMEWORK

The framework is based in the *vector space model* (VSM) presented by Salton [7], that transforms text into a word–frequency subspace. In here documents/webpages are easily compared, simply by measuring the angle between them, equivalent to their dot product. As such, the word–frequency from each webpage is then counted from a given search result and placed in a common term/document[1] matrix as shown in figure 1. The term/document matrix is beforehand filtered for trivial words by a stop–word list, discarding of e.g. and, of and the, and also words that occur only in few documents are removed. To avoid problems of different document lengths and too dominant term frequencies a normalization is subsequently multiplied to each element in the pre-normalized term/document matrix $\widetilde{\mathbf{X}}$. An element $(\widetilde{t}, \widetilde{d})$in the $t \times d$ term/document matrix is herby formulated as,

$$\mathbf{X}_{\widetilde{t},\widetilde{d}} = \widetilde{\mathbf{X}}_{\widetilde{t},\widetilde{d}} \cdot \frac{1}{\sqrt{\widetilde{\mathbf{X}}_{\widetilde{d}} * \widetilde{\mathbf{X}}_{\widetilde{d}}}} \cdot \log\left(\frac{\sum_{t,d} \widetilde{\mathbf{X}}}{\sum_{d} \widetilde{\mathbf{X}}_{\widetilde{t}}}\right), \tag{1}$$

where $\widetilde{\mathbf{X}}_{\widetilde{d}}$ is the $\widetilde{d}$ column vector amd $\widetilde{\mathbf{X}}_{\widetilde{t}}$ is the $\widetilde{t}$ row vector in the pre-normalized term/document matrix.

Employing *principal component analysis* to find a better subspace representation from the term/document matrix $\mathbf{X}$, Deerwester [3] set up the framework named *latent semantic indexing* (LSI) using *singular vector decomposition* (SVD). In here he observed that document were grouping in ray like structures that held the same semantic meaning thus the same topic. In figure 2 (top row) the projection of the first three LSI dimensions are shown, clearly reflecting the groping structure of the topics. A subspace of lower dimension can hereby be found using the first k LSI components, and words of *polysemy* [2] or *synonymy*[3] meanings are closely aligned together.

[1] We shall regard words as terms and webpages as documents in the following text.

[2] Polysemy: Words that have more than one meaning/topic, e.g. Jaguar can be a cat or a car.

[3] Synonymy: One meaning/topic can be describer by different words.

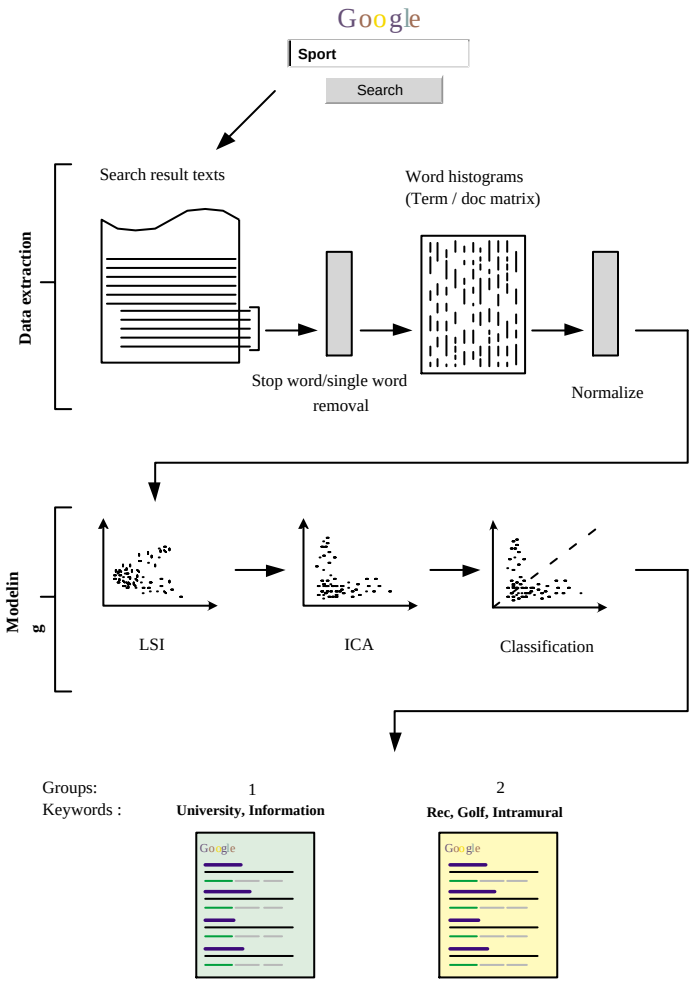

Figure 1: At the beginning of the search the user submits a search query to the search engine, e.g. sport. The highest ranked webpages returned from the search engine are filtered for trivial words and a term/document matrix is generated. The normalized term/document matrix is then mapped trough LSI and ICA to find the independent underlying topics and each document/webpage is classified in that regard. The keywords that dominate a topic class are presented to the user to re-submit the search query using one or more of the keywords.

In the LSI model the components are bound to be orthogonal, thus not being able to align well with the directions of the grouping structures. Further extending this approach to *independent component analysis* (ICA) solves this problem [4]. We are hereby able to classify each document as being assigned to the component by which it is closest in angle, see figure 2.

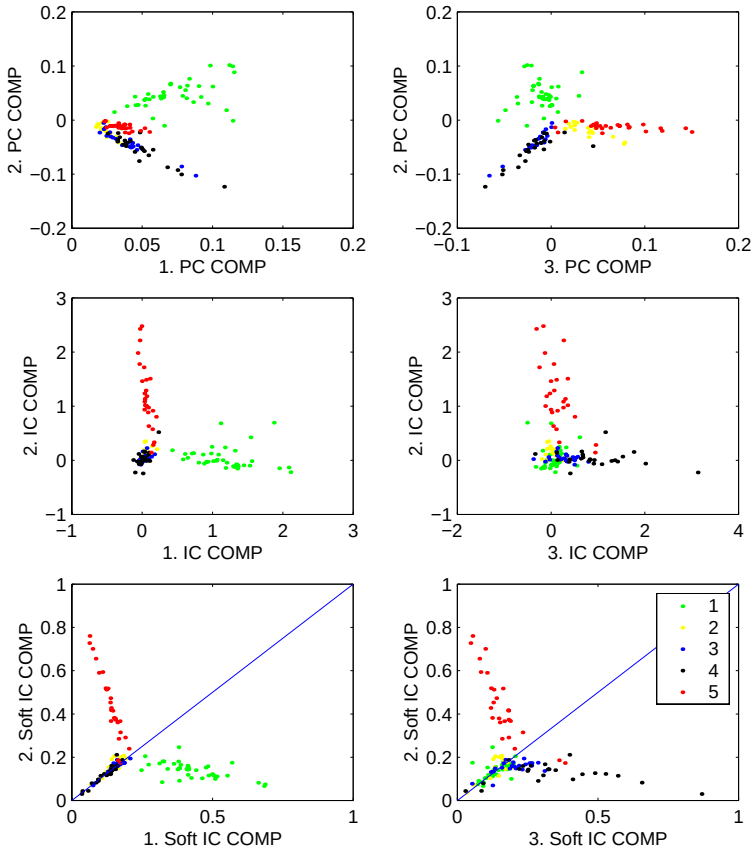

Figure 2: Medical abstracts from the Medline database are projected onto the LSI and ICA basis where each dot represents a document. The coloring shows the manually labled five classes that are used for verification. The LSI dimensions (top) are shown by the first PCA components and clearly show the ray like grouping structure. Further employing ICA (middle) with the LSI basis makes the grouping structures align nicely along the ICA basis. Finally classification (bottom) of each document onto a given class amounts to finding the IC component for which each document is closest in angle, or simply the biggest IC component for each document. In the bottom figures softmax normalized IC components are shown, so to express the probability for each class given a document.

The model hereby consists of two basis transformation, one from LSI and

ICA, decomposing the term/document matrix into,

$$\underset{t\times d}{\mathbf{X}} = \underset{t\times k}{\mathbf{T}} \cdot \underset{k\times k}{\mathbf{A}} \cdot \underset{k\times d}{\mathbf{S}} \quad , \tag{2}$$

where $\mathbf{T}$ holds the term eigenvectors from the SVD[4], $\mathbf{A}$ is the ICA mixing matrix and $\mathbf{S}$ holds the separated documents. By matrix inversion of $\mathbf{A}$ the IC components $\mathbf{S} = \mathbf{A}^{-1}\mathbf{L}\mathbf{D}^{\top}$ is found.

Classifying a document $\widetilde{d}$ to a class label l is done by finding the IC component $\mathbf{S}_{\widetilde{d}}$ with the largest value,

$$l_{\widetilde{d}} = \arg\max_{\widetilde{k}} \mathbf{S}_{\widetilde{k},\widetilde{d}} \tag{3}$$

The collected basis $\mathbf{TA}$ holds the mixing proportions coming from the term space to the lower dimensional topic space. Back projecting the found topic basis vectors lets us find the most dominant terms/keywords for each topic. In figure 3 a unit normalized column in the combined $\mathbf{TA}$ matrix is shown, using the data shortly described in figure 2. Normalizing the columns to unit length formes a natural ranking of the keywords equivalent to their dominance/importance in a topic.

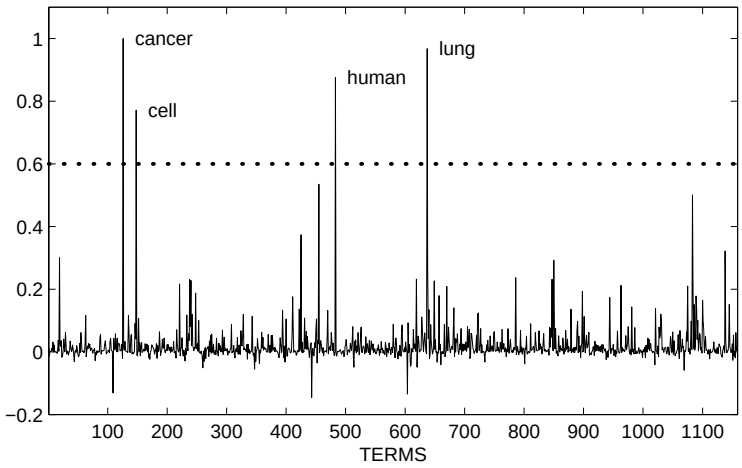

Figure 3: Keywords describing each topic can be found as the most dominant terms centered on that topic. Using a threshold lets us determine a limited number of keywords that are ranked by size.

[4]The singular vector decomposition (SVD), decomposes the term/document matrix $X = TLD^{\top}$, where T holds the term eigenvectors, L the eigenvalues and D the document eigenvectors

Optimization

From eq. (3) we are left to find the SVD decomposition $\mathbf{X} = \mathbf{TLD}^\top$, the ICA mixing matrix $\mathbf{A}$ and its inverse. The SVD is done using the LAPACK[5] [6] as is the matrix inversion. The inverse mixing matrix $\mathbf{W}$ is found using a iterative gradient optimization with the Bell and Sejnowski *infomax* ICA algorithm [2], together with the *natural gradient* by Amari [1].

The log likelihood of the ICA model can be written as,

$$\log p(\mathbf{X}|\mathbf{A}) = N \log \det \mathbf{W} + \sum_{k,d} \log \cosh(\mathbf{S}), \tag{4}$$

The gradient update with the natural gradient correction is given as,

$$\Delta \mathbf{W} = \mu \, (N\mathbf{I} - \tanh(\mathbf{S})\mathbf{X}^\top \mathbf{W}^\top)\mathbf{W}, \tag{5}$$

where $\mathbf{I}$ is the $k \times k$ identity matrix and μ is the step size of the gradient that is fixed. Initially the parameters in $\mathbf{W}$ are set to the identity matrix.

Number of classes

Finally we are left with the problem of finding the number of topics thus the optimal number of components. Using the maximum likelihood formulation of the LSI space from Minka [5] we can write the collected likelihood for the model $p(\mathbf{X}|\mathbf{A}, \mathbf{L})$.

Wanting a computational simple model and still reasonable estimation we use the *Bayes information criterion* (BIC), and find the model with the highest probability

$$p(\mathbf{X}|\mathbf{A}, \mathbf{L}) \; d^{-\frac{D}{2}}, \tag{6}$$

for different values of k. Where $D = k(2t - k + 1)/2 + 1 + k^2$ is the number of free parameters in the model. Typically k is tested in the range of 1..4, as we prefer not to have two many classes that might confuse the end result that the user has to see.

Topic ranking of webpages

The angle between a given document and the IC component that correspond to its class label describes how well it fits the keywords for that class. At the same time, documents that are close to zero in all components are very effected to the noise background, thus less likely to be member of that given class to which it has been assigned. We therefore formulate a topic rank r having two competing terms, one for the angle and one for the distance to zero. A given document $\widetilde{d}$ is ranked,

$$r_{\widetilde{d}} = \left[1 - \cos^{-1}\left(\frac{\mathbf{S}_{k_{\max}\widetilde{d}}}{|\mathbf{S}_{\widetilde{d}}|}\right) \frac{4}{\pi}\right] \left[\frac{|\mathbf{S}_{\widetilde{d}}|}{\max(\mathbf{S}_{k_{\max}})}\right], \tag{7}$$

[5] This can be done more computational efficient and online for realtime implementations.

where $|\mathbf{S}_{\widetilde{d}}|$ is the vector norm of the $\widetilde{d}$ document, $\mathbf{S}_{k_{\max}\widetilde{d}}$ is the largest value in $\mathbf{S}$ for document $\widetilde{d}$ and $\max(\mathbf{S}_{k_{\max}})$ is the largest value in $\mathbf{S}$ for the class to which document $\widetilde{d}$ belongs.

CODE IMPLEMENTATION

This section will briefly describe the implementation of our contest submission. The code has been written entirely in C++ and compiles and runs on RedHat Linux without warnings. The code makes heavy use of the STL for containers etc. Wherever possible, the code has been written for clarity, not for execution speed or minimal memory consumption. Datastructures and algoritms have generally been chosen to keep the overall runtime complexity low.

The source code is reasonably commented and it's encouraged to take a look through the sources for the best overview of the implementation details.

The search

The search part of our implementation is extremely naive and simple. It searches through the documents in a linear fashion and tests every words for a match. Every time a match is found in a document, the URL and all terms of the documents are written to a plain text file called `searchresult.txt`.The search ends when one hundred matching documents has been found or when there's no more documents to search through.

The searcher is implemented in the *ripper-framework* supplied by Google for this contest, and THIS SHOULD NOT BE CONSIDERED OUR CONTEST CONTRIBUTION but a naive supplement for Googles search engine. The searcher is implemented as a derivation of the ParseHandler-class in the file parsehandler-search.cc.

The searcher is invoked through the ripper-framework as follows and will prompt for the search term upon invocation.

```
ripper --search [pre-parsed repository files]
```

The classification

The classifier is an independant piece of software. It reads the contents of the file `../searchresult.txt` (probably generated by the searcher, see above) and does classification on this data. The program doesn't accept any parameters, so invocation is done by simply running it as follows.

```
searchclassifier
```

When classification is done, the result is written as HTML to the file

```
result.html
```

Global parameters

The classifier has a number of parameters which controls the classification. These are placed in the `Constants` class which is implemented in the files `constants.h` and `constants.cc`. See these files for the actual parameters and default values.

Structure

The implementation is divided in a handfull of classes. Each class is implemented in a pair of `.h` and `.cc` files with the same name as the class. The main entry point and core functions of the code is located in the file `searchclassifier.cc`.

The most important classes are the `TermDoc`, the `Classifier` and the `Outputter`. These classes does the main part of the work, but are relying also on a few aggregated classes. The `Matrix` class also plays an important role, as it is the center of all calculations. See figure 4 for a structural layout of the implementation.

The `TermDoc` class is responsible for building the `term-doc` matrix. This can be done in a few steps, starting by configuring the class with stopwords and word endings. Then documents and words should be added in sequential order and finally `Process` should be executed. `Process` filters the data for stopwords, empty documents etc. When `Process` has ben executed the final term-doc matrix can be generated by a call to `CreateMatrix`. The `TermDoc` uses the classes `Endings`, `Stopwords` and `Wordmap` (which in turns uses `Wordcounter`) to accomplish these actions.

The class `Classifier` does the actual classification of the term-doc matrix. By calling the method `Classify` and supplying a `TermDoc` instance, the `Classifier` runs SVD, BIC and ICA on the term-doc matrix. Finally the documents will be classified and keywords for each class will be extracted. The `Classifier` can be qeuried about number of classes and documents and keywords for each class.

The `Outputter` class generates an HTML file with the result of the classification. The method `Output` takes an instance of the `Classifier` and formats and writes the data from this instance in an HTML file called `result.html`.

The `Matrix` class implements a few basic matrix operations like addition and multiplication and some specialized operations for this application. It

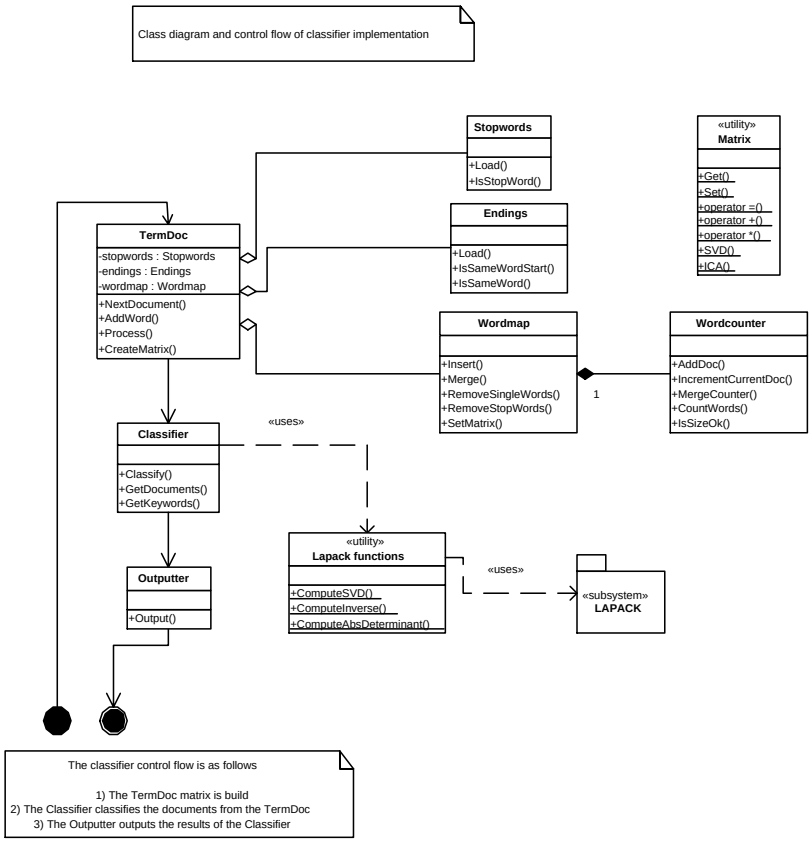

Figure 4: Structural layout of the implementation.

also uses LAPACK [6] for doing non-trivial calculations like SVD and matrix inversion..

REFERENCES

[1] S. Amari, A. Cichocki and H. H. Yang, "A New Learning Algorithm for Blind Signal Separation," in D. S. Touretzky, M. C. Mozer and M. E. Hasselmo (eds.), **Advances in Neural Information Processing Systems**, The MIT Press, 1996, vol. 8, pp. 757–763.

[2] A. Bell and T. Sejnowski, "An Information-Maximization Approach to Blind Separation and Blind Deconvolution," **Neural Computation**, vol. 7, pp. 1129–1159, 1995.

[3] S. Deerwester, S. Dumais, G. Furnas, T. Landauer and R. Harshman, "Indexing by Latent Semantic Analysis," **J. Amer. Soc. for Inf. Science**, vol. 41, pp. 391–407, 1990.

[4] T. Kolenda, L. Hansen and S. Sigurdsson, "Independent Components in Text," **M. Girolami, editor, Advances in Independent Component Analysis, Springer-Verlag**, pp. 229–250, 2000.

[5] T. Minka, "Automatic Choice of Dimensionality for PCA," **In proc. of NIPS'2000**, vol. 13, 2000.

[6] Netlib, "CLAPACK (f2c'ed version of LAPACK)," 2002, http://www.netlib.org/clapack/.

[7] G. Salton, **Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer**, Addison-Wesle, 1989.