Categorization and Visualization of News Items

Peppe Bergqvist, Henrik Eneroth, Joel Hinz, Robert Krevers, Gustav Ladén, Anders Mellbratt
Cognitive Science Program, University of Linköping, June 2006

Finding and sorting out relevant articles and news items from the vast amounts of information available on the internet is becoming an increasingly time-consuming task. In this paper, we present a program that uses a Growing Self Organizing Map adjusted to allow for incremental changes, and a simple linguistic method to sort out news articles in a way that is relevant to the end user while still giving him or her power to choose what to read and what to ignore, with focus on information abstraction and categorization. We wanted to see whether it was possible to create such a program combined with a user interface that is easy to understand. Our results show that such a project is indeed feasible, although further research is needed to make the system better.

1. Introduction and Objectives

The modern society has undergone significant changes in how information is distributed. In particular, the introduction and breakthrough of the internet has suddenly given us an opportunity to receive vast amounts of information quickly. However, whereas previously it might have been hard to find information at all, we are now faced with the problem of sorting news articles, blog entries, e-mails and other forms of written communication – a task which is time-consuming and potentially costly. We believe that to not only be annoying but also unnecessary, and therefore we propose a program here that attempts to automate the process of finding relevant new information quickly and then presents the results to the end user in a way that is user friendly but not time consuming.

Objectives

The goal of the project was therefore to build a program that could meet the user requirements of saving time and reading only relevant items, established through the use of a persona and scenarios. Given our multiple areas of interest, there were three significantly different questions that we wanted answers to:

Is it possible to create such a user interface?

Is it possible to sort news items into categories automatically?

Is it technically possible to draw and to modify a map that suits the user interface needed?

2. Pre-Design Process and Method

Study of similar systems

There are many who have worked with information retrieval, categorization

visualization and user interface design – but separately. Few projects have tried to combine the three of them.

Dittenbach, Rauber and Merkl suggested a process using similar linguistic methods and Kohonen maps [1]. Another classification method is the WebDCC algorithm [2]. The two of them have in common that they generate maps that are hierarchic, while we decided to implement a flat, 2-D map because our focus lies on the end user and that would create an easier-to-understand interface.

Identifying users

To best make decisions on the interface design, we devised a persona of a young man called Lars, who is interested in technology news but seldom able to find his way through the jungle of articles he doesn't want to read when browsing for specific topics.

Using Lars, we also created two scenarios [3] in which he sits in front of his computer for four minutes and for half an hour, respectively, with his girlfriend waiting for him in the living room.

The persona then enabled us to establish plausible requirements Lars would have on the program. Mainly, he has two demands:

Categorization: If the articles were pre-sorted already, Lars would not have to manually skim through every one of them in order to decide which to read later on.

Chronology: Lars is interested in what's new in technology, and he has no wish to read about technology from last year or even from two weeks ago. Hence we have to distinguish recent articles from older ones. A binary or discrete representation (with few steps) of time is really all that's needed to satisfy him – for instance, an article may be ”very new”, ”new”, ”old” or ”ancient”. If he could easily see the age of an item, he would be able to instantly decide whether or not he had the time and interest to read it.

Assuming Lars has a decent memory, he will not be interested in reading an article that he's seen already (at least not by clicking inadvertently), and thus we will want to display the information in a way that helps him avoid this kind of redundancy.

Prototype design and testing

Five prototype alternatives were constructed to evaluate through heuristic evaluation.

- Alternative 1 used colors to indicate chronology, giving the time aspect too much focus.
- Alternative 2 was heavily biased towards displaying a continuous time axis. Although it might not hurt to use such a system, there is really no good reason to do so either.
- Alternative 3 was too hard to understand, rendering it useless.
- Alternative 4 was also hard-to-grasp and hard to get an overview of. Besides, there are no ready-to-use technical solutions available for the implementation of it.

Since these four are all in some way faulty, we decided on our fifth alternative, which is shown in figure 1.

After having chosen a design, we conducted a test with four volunteers, a so-called lo-fidelity prototyping [4]. The volunteers were all males 20-25 studying cognitive science at Linköping University. Having few test users enabled us to better evaluate the earlier constructed persona.

A number of news articles of different types and lengths about current events in different subjects were chosen for use in the test.

They were grouped together by subject and represented by dots on a whiteboard drawing, which was subsequently redrawn to match changes caused by the test users ”using” the program prototype. The articles had been printed on regular paper and were held up against the whiteboard when supposed to be shown.

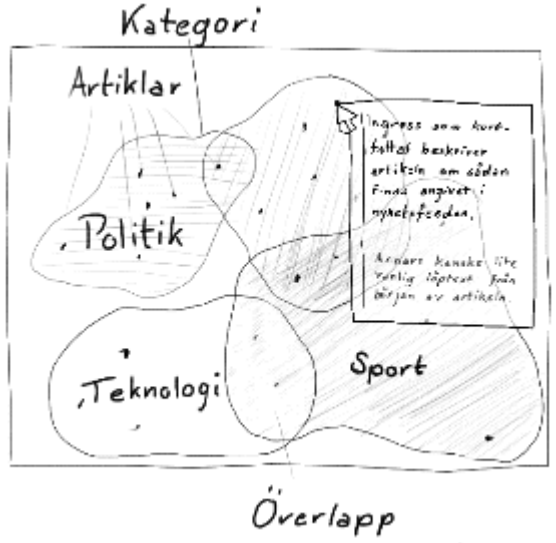

figure 1

The test users were presented with functions for a pointer, zooming, and an on/off button for displaying categories. There uses were not explained to the users beforehand, so the users had to figure out how to use them on their own. Upon activation of the category display, the categories were shown alongside the articles.

Users 1 and 2 got to use the interface without us initially showing the categories; however after their commenting on the contrary being better the third and fourth test users had the category display turned on from the start.

System procedure

Knowing what Lars wanted, we could decide on a flow scheme for the program. First, an RSS feed (which in our case basically is a dynamically updated list of references to articles) is fetched. The articles that it points to are then parsed and turned into weight vectors, after which the articles are stored in the local database and the map is drawn by the interface.

3. Algorithms

Linguistics

For the analysis of incoming texts, we decided to implement the Rocchio algorithm as explained by Salton [5] (though he does not call it by name), and then normalizing our results and comparing different vectors in the following way.

1. First, run the text through a stop list of very common words and function words to prevent words such as “and”, “she”, or “in” from occurring (and thus getting massive points, given their high frequencies).

2. Reduce each word to its stem form. For instance, “categories” and “categorize” would both turn into “categor”. The stop list and the stemming algorithm were both taken from Martin Porter’s Snowball project [6].

3. Calculate the weight W_{ij} of every word stem $_j$ in every document $_i$, using the formula

$$W_{ij} = tf_{ij} * \log \frac{N}{df_j}$$

   where N is the total number of documents in the collection, tf_{ij} the term frequency of $_j$ in $_i$, and $\log(N/df_j)$ the inverse document frequency; that is, the total number of documents containing $_j$.

4. Normalize each weight and create a vector of all words and weights for every document.

5. Compare the vector of a new text to that of a category (created by adding the vectors of each article in that category) to find out how much they resemble each other. The similarity between two weight vectors Dr and Ds is calculated using

$$sim(Dr, Ds) = \sum_{i,j=1}^{t} Wr_i * Ws_j$$

6. In order to compare vectors the results must first be normalized. For this, we used the Dice coefficient [6].

$$norm\ sim(Dr, Ds) = \frac{2 * sim(Dr, Ds)}{sim(Dr, Dr) + sim(Ds, Ds)}$$

7. If the match is good enough, the article is sorted into the category. Should there be no decent similarity to any category, the highest weight or weights in the article will be used to create a new category, which, theoretically, the user would be able to rename at will.

Self Organizing Maps

A Self Organizing Map is a 2-dimensional map that takes vectors as input and stores them at some point in a map, depending on how similar that point is to the vectors already in the map. We use this to relate news articles to other news articles and to display the result spatially along the x- and y-axes. Our reason for choosing the Self Organizing Maps (SOM) [7] approach was to get an easy overview of the categories in maintenance. The following parts will explain how each of the three SOM algorithms works.

The SOM algorithm

The map is initialized in four steps, and then trained in five repeating steps.

1. A weight vector with weights between 0 and 1 is assigned to each document. In our case, we get the vector from the analysis of the text.

2. The size of the map is determined. Note that the original SOM cannot change size once created.

3. A weight vector is initiated for each cell in the map, and assigned numbers for each dimension. We use all zeroes at initialization.

4. The training value T and the alpha value, which determines how quickly T drops, are initiated. T usually starts out at 1 and decreases to 0, at which point the training stops. It influences the level of adaptation in each training cycle. The alpha value is usually low – 0.005 for instance.

5. The training starts with the selection of a random document.

6. The best matching cell – the one with the lowest Euclidean distance between the cell vector and the document vector – is selected, using the formula

$$\sqrt{\sum_{i=d}^{i=d} (w1_i - w2_i)^2}$$

   If multiple cells share the lowest value, one of them is selected at random.

7. The document is placed in the best suitable cell, which adapts to the document. The amount of the adaptation is decided by T. The cells adjacent to the winning cell also adapt to the document, but only by an amount decided by T/2.

8. T is set to T - alpha.

9. If T > 0, go to step 5.

Growing SOM

The arguably most obvious downside of the SOM is its inability to change size, a problem which the GSOM solves [8]. It's initialized with fewer cells, often as few as 2x2, and the training phase is increased to utilize two more steps.

The first training phase is roughly the same as in the SOM. The difference is that the accumulated error value, the biggest distance between the cell and its most deviant document, is then calculated for the winning cell during the growing phase, and if it is greater than a pre-determined threshold value, new cells are added around it.

After this, a smoothing place takes place in which the network is trained with a low T value, thus restructuring the map slightly or smoothing it out. No new documents can be added during this phase.

Incremental Growing SOM

In our program, there is a constant flow of new articles, and redrawing the map every time an unseen article is added to the collection would require unnecessarily large amounts of computer power and not be very user-friendly, since the map would probably be drawn with completely different shapes every time.

By creating the IGSOM, we could keep track of the impact of each document on the map. When a new keyword (dimension) arrives, its weight for every document is calculated and the algorithm backtracks through the history of the map, with all cells updating to the value they would have gotten if the keyword had existed at the time.

The training cycle is changed too. At assimilation of a new document the map is trained with a high T value (T > 0.5), and afterwards it is smoothed out with a low T value (T < 0.2). Thus, a new document makes a high impact on the map on arrival, but all items are given a chance to change places if needed. Also, if several items are added at once, the map will be smoothed out after assimilation of the full batch.

XMLDB

We needed a tool for intra-program communication that was powerful enough to handle networks of data, granular enough to allow for fast updating of small pieces of it, and easy enough to let different parts of the process communicate directly. Therefore, we constructed the XMLDB – to let program parts interact with a database using XML.

The data is stored "flatly" in an ordinary database, but structured in a way that makes it represent networks of data rather than the otherwise commonly used tables. XMLDB reaches its full potential when handling large amounts of data containing many internal relations.

Though all communicated data is in XML, XMLDB is not just an advanced XML file. All XML that it responds with is generated rather than fetched, meaning that it can be very dynamic and selective.

4. Conclusions and Discussion

Given our own evaluations as well as the theoretical framework, we conclude the following:

- Using an Incremental Growing Self Organizing Map is a sound way of presenting the information to the end user since it makes it technically possible to modify a map without having to start from scratch every time a new article is assimilated, and since it is compatible with a user interface that satisfies our persona.
- The linguistic analysis works. Actually, this is not really surprising

since there is good scientific support for the theory, but our sample texts were usually much shorter than the longer articles and books often used to evaluate it. A smaller sample of only 90 mostly short texts in three different subjects – politics, sports and technology – showed a tendency of giving articles matched against their correct categories a similarity of, on average, 2-3 times the average of the similarities between articles matched against incorrect categories.

- Our conclusion is that the algorithm mostly needs either longer texts or a large number of documents to compare to. This is not really a problem, as our program is meant to gather large amounts of texts quickly. Also, weblog writers seldom employ proofreaders like newspapers do (mostly), and therefore those texts are likely to contain more errors in spelling and grammar.

- The prototype testing showed that it's possible to create a user friendly and intuitive interface in a way that we wanted to. Most functions were used instantly and without problems by the test users, except for the one that toggled the visibility of categories. Using colors for a better display of categories might be an idea that we did not test, but it might also make the whole interface even harder to understand [9].

  One user suggested a function for displaying the number of persons to have read a certain article, a kind of "social news-reading". This is indeed interesting, and we toyed with the idea of sharing categories and articles through an internet community, but it's also something that this article does not cover.

  Results from the tests also showed that finding an article that had been positioned in the intersection of two categories and realizing what it was about was easier than grasping that distance between different articles in the same category mattered as well.

5. References

(1) Dittenbach, Michael, Rauber, Andreas, Merkl, Dieter (2001), *Business, Culture, Politics, and Sports - How to Find Your Way Through a Bulk of News? On Content-Based Hierarchical Structuring and Organization of Large Document Archives*, Proceedings of the 12th International Conference on Database and Expert Systems Applications, Springer Lecture Notes on Computer Science, Sept. 3-7 2001

(2) Amandi, Analía, Godoy, Daniela (2006, June-July), *Modeling User Interests by Conceptual Clustering*, Information Systems, 31(4-5), pp. 247-265

(3) Cooper, Alan (2004), *Inmates Are Running the Asylum, The: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity*, Sams

(4) Preece, Jennifer, Rogers, Yvonne, Sharp, Helen (2002), *Interaction Design: Beyond Human-Computer Interaction*, John Wiley and Sons, Inc.

(5) Salton, Gerard (1989), *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Weasly

(6) Porter, Martin (2001-2005), *Snowball: A Language for Stemming Algorithms* [www], http://snowball.tartarus.org/, last visited 28 May 2006

(7) Kohonen, Teuvo (1989), *Self-Organization and Associative Memory*, Springer-Verlag, 3rd edition

(8) Alahakoon Damminda., Halgamuge, Saman K. and Srinivasan, Bala (2000, May), *Dynamic Self Organising Maps with Controlled Growth for Knowledge Discovery*, IEEE Transactions on Neural Networks, 11(3), pp. 601-614

(9) Tufte, Edward (1997), *Visual Explanations*, Cheshire, CT: Graphics Press