HUTCH MODEL IN INFORMATION STRUCTURING

Heikki Hyötyniemi
Helsinki University of Technology
Control Engineering Laboratory
P.O. Box 5400, FIN-02015 HUT, Finland

This paper presents a generic methodology for combining quantitative *and* qualitative *data analysis approaches. The idea is based on the view of natural dependency structures existing in the high-dimensional data space: There are clusters, and within them there are fine-tuned local linear subspaces. For example, given a set of document "fingerprints", the developed algorithm is capable of constructing such a model based on the statistical relationships between the terms (words) in the documents. A visual view of the "generalized keywords" among the documents is found, extending the idea of WEBSOM. Wider horizons are also open here; the model can be seen as an approximation of the cognitive chunking process. This is demonstrated in the framework of handwritten digit analysis.*

1 INTRODUCTION

As the amount of available information has exploded, tools for Data Mining, and specially for Knowledge Mining, are becoming invaluable [2], [6], [18]. True knowledge can only exist in a brain, but in a limited sense, when "understanding" is restricted to mastering the dependency structures among the data entities, this kind of *contextual semantics* offers new possibilities. Assuming that one is able to construct data processing mechanisms so that there is a correspondence between data structures and mental representations (see [13]), the resulting data structures can (hopefully) be interpreted by humans — and (hopefully) some feeling of "intelligence" emerges.

There are a plenty of ambitious terms here — like *understanding* and *emergence* — and, indeed, in the field of knowledge mining one has to face different kinds of challenges. The main objective in this paper is to propose how the following very different but equally relevant points of view could be integrated:

1. *Theory of complex systems* offers intuitions and tools for defining and analysis of processes that result in *emergent phenomena.*

2. *Cognitive science* gives us understanding of human information processing, introducing the ideas of *long-term meory (LTM)* and *short-term memory (STM),* and the idea of *chunks.*

3. *Pragmatism* prevents us from becoming too whimsical — when dealing with real world data, fast and robust processing is of utmost importance.

Indeed, in this paper an approach to manipulating *real world data* following the ideas of *cognitive science* and using the tools from *complex systems research* is discussed. In concrete terms, an *algorithm* for data (knowledge) mining is implemented.

2 HUTCH MODEL

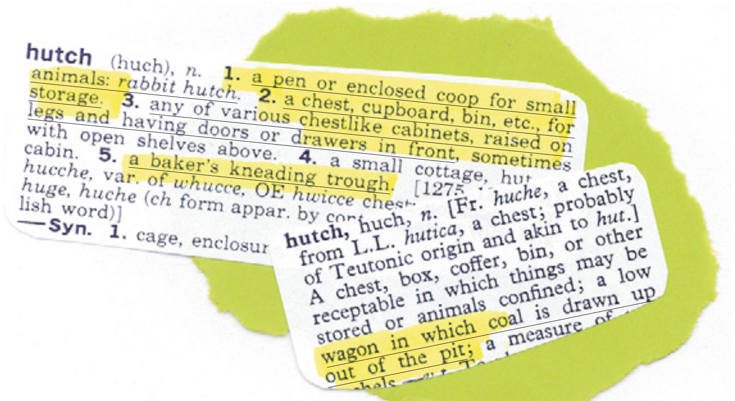
There are many hypotheses concerning mental processes[1]. Among the most concrete ones are the *cognitivistic* ideas, including the concepts like *mental representations,* and *capacity limitations.* In this context, only some key features are concentrated on, forgetting about details, and it must be recognized that the claims here only concern some kind of "artificial cognition". The *chunks* are assumed to be the "mental atoms", the basic units of which the higher-level perceptions are constructed; it is assumed that only a fixed number (known as *short-term memory capacity STM*) of all available chunks (known as *long-term memory capacity LTM*) are simultaneously used for representing the observation. This kind of principle — only having a subset of all available data structures active at a time — introduces new kinds of mathematical problems; such problem setting is known as *sparse coding.*

There is a continuum between *dense* and *sparse* representations. Density is a measure for how many of the available constructs are used for representing the data item: Dense coding means that all of them are used, typically resulting in structureless but mathematically efficient implementations; sparse coding means that only a few of them are active at any given time, resulting in a structured, physically perhaps better motivated models and better representative sets of features. For example, *Latent Semantic Indexing* (LSI), being based on *principal component analysis* PCA (see [1]), only constructs one global model for the whole data, always using all model constructs [4]; this would mean that STM = LTM in the cognitivist terminology. In the other end of the continuum, the self-organizing map (SOM) method [16], for example, being a clustering method, constructs local models for different regions of the data space, thus employing only one of the model prototypes at a time, so that STM = 1.

The GGHA approach (see [11]) implements a combination of these two extremes, always using a fixed number of available *numeric chunks* (see [12], [14]). GGHA is an extension of the generalized Hebbian algorithm GHA originally developed by Prof. Erkki Oja (see [7]): In GHA, the principal components are iteratively extracted, whereas in GGHA, various different trains of components can be extracted, making it possible to model also non-unimodal mixture models, and, in some cases, returning *independent components* (assuming that the underlying components are shared by various patterns; in this case one should perhaps speak of *sparse components*). GGHA explicitly searches for sparsity, assuming that the underlying data is multimodal but the different subdistributions share common elements in their underlying linear structure. Explicitly fixing the number of employed candidate features means that the matching process inevitably becomes a sequential step-by-step process in GGHA.

The HUTCH Model (brief for "HUT Chunking Model", see Fig. 1) is an *extension of GGHA* into the direction of *complex systems* (see [15]). An additional nonlinearity is introduced in the system structure: This modification should facilitate searches for "active" features more explicitly and efficiently, hopefully resulting in faster convergence.

[1]... And there also exists a wealth of approaches towards "cognitively motivated" processing of complex data. Let us briefly study one prominent representative of such approaches — the Kohonen SOM that is based on self-organization [16]. The intuition that the human brain would do its wonders by applying the same principles is perhaps too high-spirited, but there are other cognitively clever innovations: SOM is capable of cleverly making the complex data conceptually understandable by utilizing the human's marvellous visual pattern recognition capability. This idea has been applied in "semantic maps" (see [22], [24]), and for Web mining in WEBSOM (see [9], [8]).



hutch (huch), *n.* **1.** a pen or enclosed coop for small animals: *rabbit hutch.* **2.** a chest, cupboard, bin, etc., for storage. **3.** any of various chestlike cabinets, raised on legs and having doors or drawers in front, sometimes with open shelves above. **4.** a small cottage, hut cabin. **5.** a baker's kneading trough. [1275 hucche, var. of *whucce,* OE *hwicce* chest huge, *huche* (*ch* form appar. by con lish word)]
—**Syn. 1.** cage, enclosur

hutch, huch, *n.* [Fr. *huche,* a chest, from L.L. *hutica,* a chest; probably of Teutonic origin and akin to *hut.*] A chest, box, coffer, bin, or other receptable in which things may be stored or animals confined; a low wagon in which coal is drawn up out of the pit; a measure of

Figure 1: Can the data model capture something *life-like*? Can it be used for storage, can it be used for *kneading,* and can it be utilized in mining?

This nonlinearity also automatically causes sparsity to emerge without any outside intervention. More sophisticated pattern matching becomes also possible, being based on parallel rather than sequential processing; this is based on the following iteration:

$$x(k+1) = f_{\text{cut}}(Ax(k) + a(f)). \tag{1}$$

The generalized *cut function* is the key to sparsity and emerging structure; it is a vector-valued function, being defined elementwise as

$$f_{\text{cut},i}(x) = \begin{cases} x_i, & \text{if } x_i > 0, \text{ and} \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

for all $i = 1, \ldots, \dim(x)$. Assuming that the process (1) converges to some $\bar{x}$, this vector can be interpreted as the "internal image" corresponding to the input data f. Above, A is a compatible real-valued matrix, and $a(f)$ is a vector; the construction of these data structures, given the *chunk matrix* C and the input data vector f, will be explained later.

In the algorithm in Fig. 2, the chunk vectors C_i, where $1 \leq i \leq \text{LTM}$, have the same dimension as the input data vectors f. Before the algorithm is started, these vectors are first initialized so that their elements have random positive values, and the vector lengths are normalized to unity; the algorithm is iterated until convergence of the vectors C_i is reached. The algorithm is constructed so that C always remains positive-valued. Note that the Steps 4 (adaptation) and 5 (normalization) could be combined in the same way as in GHA. Normalization of decomposed inputs F_c can be included in the algorithm to make it possible to emphasize the additive features as compared to the potentially very dominating center point vector, thus compensating for the possibly pathological eigenvalue distribution in the data covariance matrix. This means that before the Step 4 of the algorithm, apply the following for all $c \in \Gamma$ (the role of ϵ is to help avoiding problems if there exist pathological input vectors of zero length):

$$F_c \leftarrow F_c / \sqrt{F_c^T F_c + \epsilon}. \tag{3}$$

In the algorithm, the function d returns the grid distance between two nodes. Parameter σ is the neighborhood radius (see [16]), and γ is the gradient algorithm step size; these parameters gradually decay towards zero. Typically, if there is scarcity of memory and the features in the data are distinct, no self-organization can be seen in the final results; the role of the underlying SOM is to keep all vectors C_i involved in the adaptation.

HUTCH model adaptation:

1. Take the next input vector sample and make it positive-valued:

   $$f \leftarrow f_{\text{cut}}(f).$$

2. Match the data against the model (see below), determining the data structures:

   $$\Gamma, \quad \text{and} \quad F_c, \text{ for all } c \in \Gamma.$$

3. Calculate the neighborhoods:

   $$h_{c,i} = \exp\left(-\frac{d^2(c,i)}{2\sigma^2}\right), \quad \text{for all } c \in \Gamma, \quad 1 \le i \le \text{LTM}. \tag{4}$$

4. Apply the self-organizing algorithm using all vectors F_c as input:

   $$C_i \leftarrow C_i + \gamma h_{c,i} \cdot (F_c - C_i), \quad \text{for all } c \in \Gamma, \quad 1 \le i \le \text{LTM}. \tag{5}$$

5. Normalize the chunk vectors:

   $$C_i \leftarrow C_i / \sqrt{C_i^T C_i}, \quad \text{for all } 1 \le i \le \text{LTM}. \tag{6}$$

6. Update the parameters γ and σ:

   $$\gamma \leftarrow \lambda \cdot \gamma, \quad \sigma \leftarrow \lambda \cdot \sigma. \tag{7}$$

7. If the model has not yet converged, go back to Step 1.

Figure 2: The not-so-short description of the HUTCH algorithm

The features characterizing a domain field, or *chunks,* are (in this case) high-dimensional vectors; they are stored as columns in the matrix C. The number of chunks is typically much lower than the dimension of data vectors f, meaning that compression of data takes place; hopefully only the irrelevant or noisy information is disregarded. Assuming that a set of domain-oriented chunks has been found, the input can be approximately reconstructed as a weighted sum of them. After the "internal image" $\bar{x}$ of an observation has been found, determining the *latent coordinate values* representing the data vector f, the corresponding estimate for the data vector can be reconstructed as

$$\hat{f} = f_{\text{cut}}\left(C \cdot \bar{x}\right). \tag{8}$$

No matter whether the model is being learned or the ready-to-use model is being applied, the essential task is that of finding the internal image $\bar{x}$ representing the input data vector f. There exist two essentially different ways to match the data against the model represented by the vectors in C:

- The "PCA-oriented", *inflating* sequential deconstruction approach is the generalization of GHA, resulting in a robust feature extraction, explicitly ripping off the most probable features from f.

- The "ICA-oriented", *deflating* parallel deconstruction approach is more sophisticated, carrying out explicit matrix inversion, resulting in mathematically best possible representation for f.

`HUTCH` model matching (inflating version):

1. Set the set of candidates empty:
$$\Gamma = \{\ \}. \tag{9}$$
2. Calculate the correlations between f and the remaining chunk vectors:
$$\phi = (C - C_\Gamma)^T \cdot f. \tag{10}$$
3. Select the chunk c having the best correlation with the input vector:
$$c = \arg\max_{1 \le i \le \mathrm{LTM}} \ \{\ \phi_i\ \}. \tag{11}$$
4. Update the set of candidates:
$$\Gamma = \Gamma + \{\ c\ \}, \quad F_c = f, \quad \bar{x}_c = \phi_c. \tag{12}$$
5. Eliminate the contribution of the feature c by setting
$$f \leftarrow F_c - \bar{x}_c \cdot C_c, \qquad f \leftarrow f_{\mathrm{cut}}(f). \tag{13}$$
6. If the iteration limit STM has not yet been reached, go back to Step 2.

Figure 3: Matching the `HUTCH` model against data — first alternative

`HUTCH` model matching (deflating version):

1. Select all of the available chunks in the set of candidates:
$$\Gamma = \{\ 1, \ldots, \mathrm{LTM}\ \}. \tag{14}$$
2. Starting from $x(0) = \bar{x}$, iterate the following until it converges to new $\bar{x}$:
$$x(k+1) = f_{\mathrm{cut}}\left(\left((1-\mu)\cdot I - \mu C_\Gamma^T C_\Gamma\right)\cdot x(k) + \mu C_\Gamma^T \cdot f\right). \tag{15}$$
3. If the intended number of chunks (STM) has been reached, terminate; calculate
$$F_c = \frac{\bar{x}_c}{\sqrt{\bar{x}^T \bar{x}}} \cdot \left(f - \sum_{\substack{i \in \Gamma \\ i \ne c}} \bar{x}_i C_i \right), \quad \text{for all } c \in \Gamma. \tag{16}$$
4. Otherwise, drop out the least significant chunk c from Γ:
$$\Gamma = \Gamma - \left\{\ \arg\min_{c \in \Gamma} \{\bar{x}_c\}\ \right\}. \tag{17}$$
5. Go back to Step 2.

Figure 4: Matching the `HUTCH` model against data — second alternative

There exist a couple of notations in the algorithms that deserve a closer look. First, Γ stands for the set of integers that contains those chunk indices that are assumedly involved when explaining f. After model matching, there should be STM elements in Γ. The matrix F contains the "chunk-wise" input data vectors; that is, F_c contains the contribution of f in the direction of C_c. It is assumed that the matrix C_Γ is a copy of C where all other columns are zeroed except those whose indices are given in the set Γ; this means that in $C - C_\Gamma$ only those columns are zeroed whose index is found in Γ.

Indeed, these two approaches, as presented in Figs. 3 and 4, can be combined. For example, to reach fast operation, still having mathematically well-founded extraction of feature contributions, the appropriate set of chunk indices in Γ can be selected using inflation (version 1), even though the final determination of the chunk-wise input vectors F_i were carried out applying the more sophisticated matching scheme (version 2). No matter which version is applied, the internal image vector has to be initialized, so that $\bar{x} = \mathbf{0}$ before the model matching iterations are started.

In the model matching version 2, the iteration in Step 2 in principle carries out matrix inversion, matching the vectors against the data; this inversion is implemented as an iteration that is based on the steepest descent approach, where μ is the step size. Because of the nonlinearity f_{cut} in the model, inversion is not mathematically exact, always trying to find a *positive* solution. Due to the bad convergence properties of gradient algorithms, it is clever to start new iterations from the previous solution when dropping out candidates during deflation. The Step 3 in the model matching version 2 also deserves some explanation: Here the input is decomposed into chunk-wise components. Note that $f \approx \sum_{i \in \Gamma} \bar{x}_i C_i$; from this one can solve the estimated contribution of the vector f in different chunk directions. Further, the relevance of the input is approximated by scaling with $\bar{x}_c$. However, there are a few special cases that need to be taken care of: First, if the input cannot at all be represented by the chunks, all $\bar{x}_i$'s are very low, making adaptation slow; second, if the chunk vectors are not orthogonal, the sizes can be rather high making the adaptation perhaps too fast[2]. It turns out that when one takes the overall vector length of $\bar{x}$ into account, the pahological cases can be avoided; that is why the result is additionally multiplied by $1/\sqrt{\bar{x}^T \bar{x}}$.

3 EXAMPLE: CHUNKS IN IMAGE DATA

To study the properties of the proposed approach, the chunking algorithm was first tested in an environment where the contents of the chunks can easily be inspected: Visual features were extracted from image data. It can be assumed that the same information-theoretic principles apply no matter what is the level in the conceptual hierarchy; now the *lowest level* is studied, so that the results can be compared to existing evidence. It has been recognized that in the case of lowest-level visual perception, the nervous system decomposes visual scenes into line segments, etc., and the representation of images is sparsely coded (for example, see [5], [20], [10]).

As testing material, a database of hand-written digits was used [17]. There were over 8000 training samples and over 8000 validation samples, containing equal number of examples of all digits from 0 to 9, coded as monochrome images in a 32×32 grid — meaning that the data dimension was 1024 (see Figs. 5 and 6).

[2]At least if the "cut function" f_{cut} is omitted, almost parallel chunks may start fighting against each other, meaning that their weights may explode, the matrix to be inverted becoming close to singular

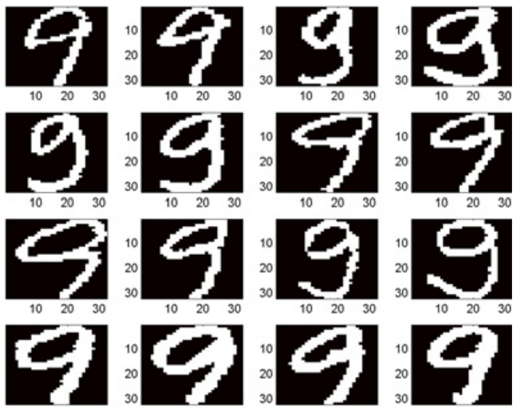


Figure 5: Typical examples of hand-written "9" in a 32×32 grid

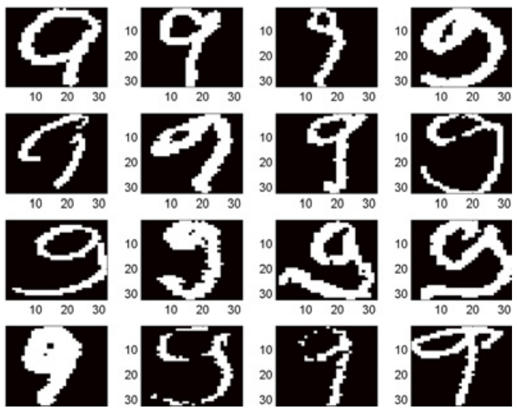


Figure 6: Examples of deformed "9" (about 20% of all the available "9"s)

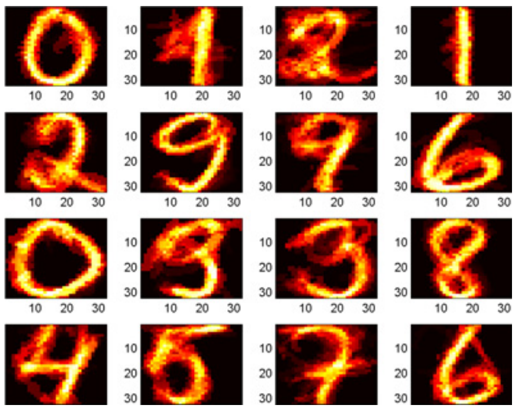


Figure 7: One chunk extracted

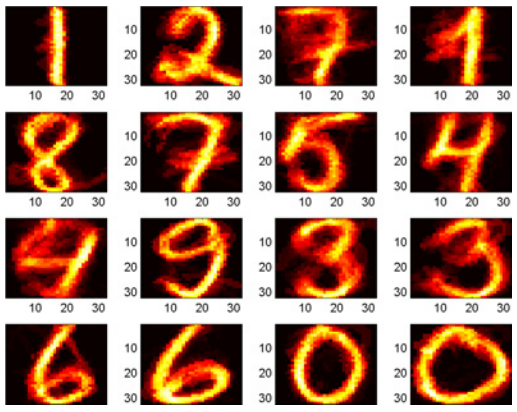


Figure 8: Two simultaneous chunks

The model matching version 2 (Fig. 4) was applied assuming that the long-term memory capacity LTM was fixed to 16, whereas the short-term memory capacity STM was varied from 1 to 6 (the results are shown in Figs. 7, 8, 9, 10, 11, and 12). Note that no matter how many features are extracted, one of the prototypes is always allocated specially for 1!

Before adaptation, the feature prototypes C_i were first initialized to random positive values. The original adaptation factor was $\gamma(0) = 0.002$, and the neighborhood radius was $\sigma(0) = 2$; these were kept constant during one epoch, whereas between epochs the forgetting factor was $\lambda = 0.5$. About a dozen epochs through the training material were executed.

Note that when STM = 1, the results are essentially the same as with Kohonen SOM. However, now there were too few nodes available to reach any self-organization among the nodes. Indeed, the balance between the data complexity, overall memory capacity (LTM), and the number of simultaneously active chunks (STM) is a delicate matter. Having different parameter combinations, the results typically become qualitatively very different; even if the parameters are kept the same, the results may differ from each other after adaptation. Interestingly enough, whatever is the faith of adaptation, the results typically "look good", or at least "interesting". It seems that also the human

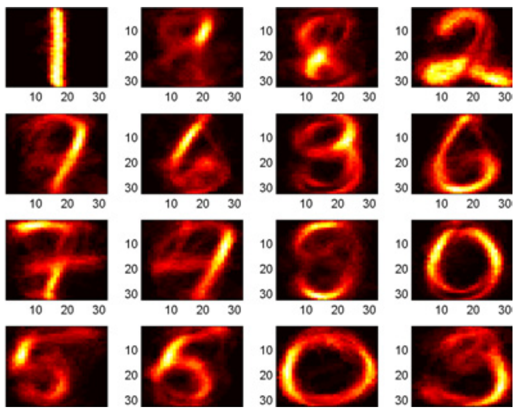


Figure 9: Three simultaneous chunks

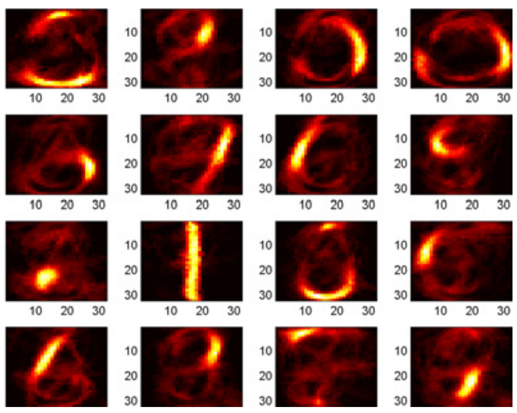


Figure 10: Four simultaneous chunks

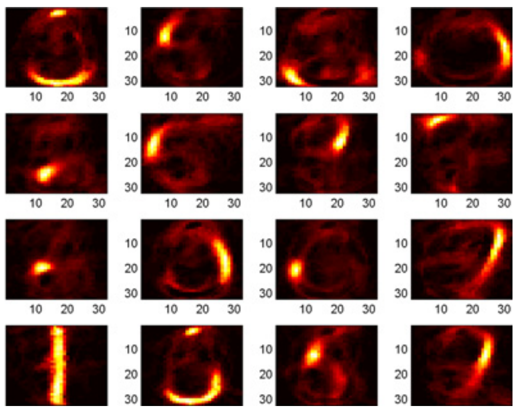


Figure 11: Five simultaneous chunks

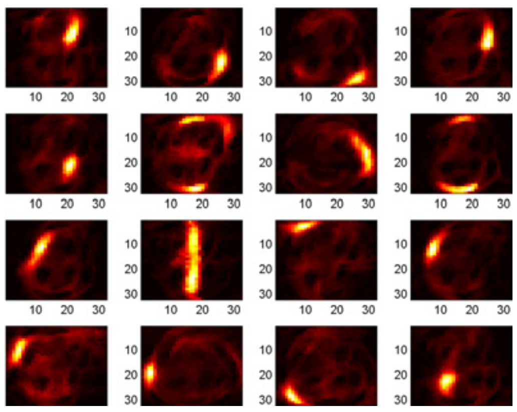


Figure 12: Six simultaneous chunks

categorizations are typically not unique (see [23]). Nevertheless, it can be claimed that the overall number of chunks (16) was too low in this case.

It seems that the results look what they should, line and curve segments emerging, perhaps the intuitively most appropriate features being found when STM is 3 or 4. The approach seems to give a nice view of the structure within the data.

How this model could be utilized in practice, if gaining insight is not enough? Below, the features were utilized in classification. It needs to be noted that each class from 0 to 9 employs the same chunks that were extracted when all the digit material was simultaneously used for training; the distributions between the chunk activations can be used to reveal the appropriate class. The data f itself, and the corresponding chunk scores in $\bar{x}$ deliver us evidence; to proceed, let us define the "extended data" vector, containing the feature weights $\bar{x}(t)$ corresponding to the input data $f(t)$:

$$\xi(t) = \left(\frac{\bar{x}(t)}{f(t)} \right). \tag{18}$$

Using the Bayesian approach one can derive a formula for conditional probability separately for each class $\mathcal{C}$:

$$p\left(\mathcal{C}|\xi\right) = \frac{p(\mathcal{C}) \cdot p(\xi|\mathcal{C})}{p(\xi)}. \tag{19}$$

Assume that $T_\mathcal{C}$ denotes the number of training samples in class $\mathcal{C}$. If T is the total number of all training samples, the a priori probability for class $\mathcal{C}$ is

$$p(\mathcal{C}) = \frac{T_\mathcal{C}}{T}. \tag{20}$$

The global and class-wise data distributions can be assumed to be Gaussian, so that for densities there holds

$$\left\{ \begin{array}{rcl} p(\xi) & = & \frac{1}{\sqrt{(2\pi)^{\dim \xi} \cdot |\Sigma|}} \cdot \exp\left(-\frac{1}{2}(\xi - \bar{\xi})^T \Sigma^{-1} (\xi - \bar{\xi})\right) \\ p(\xi|\mathcal{C}) & = & \frac{1}{\sqrt{(2\pi)^{\dim \xi} \cdot |\Sigma_\mathcal{C}|}} \cdot \exp\left(-\frac{1}{2}(\xi - \bar{\xi}_\mathcal{C})^T \Sigma_\mathcal{C}^{-1} (\xi - \bar{\xi}_\mathcal{C})\right) \end{array} \right. \tag{21}$$

Here, $\bar{\xi}$ and $\bar{\xi}_\mathcal{C}$ are the global and class-wise average vectors, respectively, and Σ and $\Sigma_\mathcal{C}$ are the corresponding covariances; these can be calculated for the global model as

$$\bar{\xi} = \frac{1}{T} \cdot \sum_{t=1}^{T} \xi(t) \qquad \text{and} \qquad \Sigma = \frac{1}{T} \cdot \sum_{t=1}^{T} \xi(t)\xi^T(t), \tag{22}$$

and, correspondingly, for the local (class-wise) model as

$$\bar{\xi}_\mathcal{C} = \frac{1}{T_\mathcal{C}} \cdot \sum_{f(t) \in \mathcal{C}} \xi(t) \qquad \text{and} \qquad \Sigma_\mathcal{C} = \frac{1}{T_\mathcal{C}} \cdot \sum_{f(t) \in \mathcal{C}} \xi(t)\xi^T(t). \tag{23}$$

Because the covariance matrices depend on the distribution of ξ, and because the optimal $\bar{x}$ therein depends on these covariance matrices, adaptation of the model parameters $\bar{\xi}_\mathcal{C}$ and $\Sigma_\mathcal{C}$ for all classes $\mathcal{C}$ becomes a complicated iterative process; the details of this process are skipped here. The resulting "conflict matrix" when using the 6-chunk model for classification of the validation data was as follows:

Correct class → Classification ↓	"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"
"0"	96.0	0	1.0	0.5	0	1.1	1.3	0	0.1	2.4
"1"	0	94.5	0	0.1	0.5	0	0.1	1.4	0.4	1.1
"2"	0.1	0.2	90.7	1.1	0.5	1.1	1.0	1.1	3.0	0.1
"3"	0	0.1	0.8	88.7	0	1.9	0	0.2	0.6	5.2
"4"	0.1	2.3	0.1	0.1	89.3	0.5	0.5	1.2	0.1	7.2
"5"	0.2	0.1	1.0	0.3	0.1	91.1	0.2	0	1.5	0.8
"6"	2.4	0.2	1.2	0	0.3	0.7	95.9	0.1	0.8	0
"7"	0	1.1	0.5	1.2	2.4	0	0	93.4	0.4	4.2
"8"	0.6	0.3	2.5	1.1	3.3	1.2	0.6	0.4	90.3	1.0
"9"	0.3	1.0	1.9	6.7	3.2	2.1	0.1	2.0	2.3	77.6

The overall hit rate was 90.8%; this is far from the best possible results (the level of almost 98% can be reached, see [17]). It can be commented that this kind of features are not very well suited for classification — note that *similarities* among the digits were searched for rather than differences, and, in this sense, it is understandable that such common features are not optimally suited for distinguishing between patterns. Class-wise feature extraction would be needed to find the common phenomena optimally characterizing the different classes. Also, extraction of higher-level features (whole strokes, and structure among the strokes) would also help in classification. However, such polishing is not concentrated on in this paper; rather, it is shown how the extracted chunks themselvaes can be valuable for gaining intuition about a complex domain field.

4 MODELING OF TEXTUAL DOCUMENTS

In the digit analysis example above, low-level statistical dependencies were utilized for determining the features. On the higher levels, when searching for some kind of *concepts,* the above approach that is based solely on statistical properties, feels like too weak. However, it is worth trying.

Thinking pragmatically, an automatic tool for finding a structure among a collection of textual documents, finding contextual relationships between texts could be invaluable for preliminary analyses of large bodies of knowledge. For example, assume that you are asked to get acquainted to some special field — first you go to a database and extract all documents with your keywords, but after that you let the algorithm look at the material and construct an overview. Instead of having hundreds of fragmentary documents, you would have some kind of table of contents giving "handles" into the documents! Other related applications could involve automatic modeling of news archives, and (collaborative) filtering. This kind of a service could be accessible through WWW, so that this tool could be seen as another route towards "Semantic Web".

Data mining in textual knowledge bases can roughly be divided in two mainstream approaches[3]: Either one tries to do "quantitative" analysis (for example, Latent Semantic Indexing LSI [4]), or one does "qualitative" analysis (different kinds of *cluster analysis* approaches). However, when studying the many-sided nature of complex data, it is evident that both of these two extremes alone are insufficient if trying to capture the essence of the data in an efficient way: What is needed is a framework for constructing integrated models with the capability of simultaneously representing the coarse and the fine structure buried in the information. Whereas the cluster centers are discrete and qualitative, the features modifying the cluster prototypes are continuous and quantitative. It turns out that one should have a tool for finding *sparse coded linear latent structures* in the data (see [13]). This seems to motivate the use of `HUTCH`-like approaches — the chunks can stand for cluster prototypes and fine-tuning subspace axes.

To test the above modeling approach in a realistic environment, raw textual material from the `Inspec` database was used. This database contains information about scientific publications. First, all abstracts with the keyword "knowledge mining" were downloaded. To make the problem more challenging, and to test the method against natural textual data with no semantic preanalysis, only the abstracts of the papers were utilized, not the titles or the explicit keywords that would also have been available. The abstracts varied in length from 37 to 280 words, the overall number of words was about 2000, and there were almost 300 documents ($K = 260$).

The documents were presented using so called "fingerprints": The fingerprints are high-dimensional integer vectors containing the term counts. The data dimension becomes very high because each term has an entry of its own in the fingerprint vector, no matter if that term is used in that document or not. What comes to the semantic contents, this kind of representation of the documents is, of course, extremely crude, but assuming that the terms in the document are more or less characteristic to the domain

[3]Of course, there exist a wealth of different kinds of approaches. For example, *Bayesian networks* (see [19], etc.) have been applied to extracting high-level structures from complex data; however, it can be argued that this approach goes *too far,* constructing causal models between variables. Since Hume it has been evident that this is not possible, given the data alone. One can reliably only deduce correlation structures among data

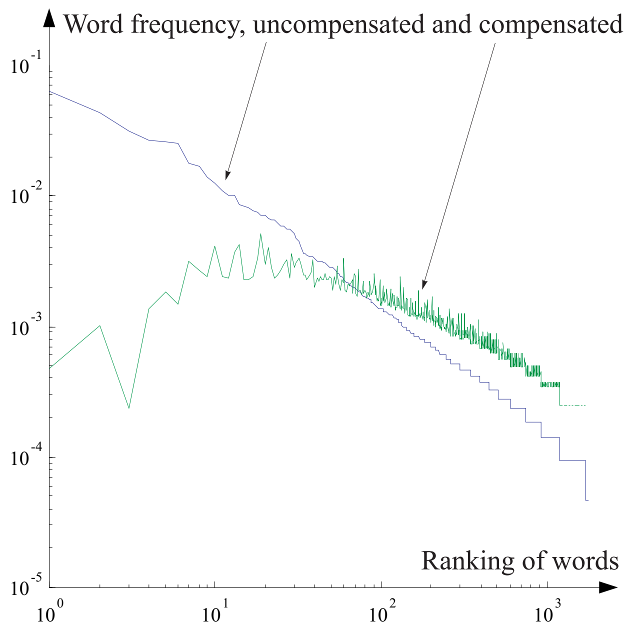


Figure 13: The "visibility" of different words in the documents. The words are distributed on the horizontal axis, ordered so that the originally most frequent words have been collected on the left, others following in the descending frequency order. On the log–log axis, the original word frequencies (the upper curve) decay almost linearly ("Zipf's law")

area, the interdependencies among the terms determine some kind of *contextual semantics* in the set of fingerprints. Linguistic analysis is now cut to minimum; the words found in the documents are used directly as entries when constructing the input term vector f, only the plural s's are eliminated in a very simplified morphological analysis phase.

Preprocessing of the material is a delicate matter; modeling of the high-dimensional space becomes difficult if the data is not appropriately conditioned. There are no absolute truths that would apply to preprocessing, but some heuristics exist. It has been claimed that the so called TFIDF weighting ("Term Frequency — Inverse Document Frequency") often gives good results [21]; this modification of the fingerprints can be defined termwise as

$$f_i \quad \leftarrow \quad f_i \cdot \log \frac{K}{K_i}.$$

Here, K is the overall number of available documents, and K_i is the number of such documents where the term number i is found. This weighting means that if a term is found in all documents, its weight will be zero; this can be motivated so that such unselective "stop-words" (like the words "the", "and", etc.) have no value when distinguishing documents from each other. The overall number of documents should be large enough to give some statistically plausible estimate of the term frequencies. After the TFIDF weighting, the "virtual" frequencies, or the significances of the words in the modeling process have been modified so that more specific words hopefully carrying some categorization information are emphasized, whereas the too common words are automatically rejected (see Fig. 13). Correspondingly, terms that are found only once in the corpus material are now neglected — they have no value when searching for similarities between documents.

In the algorithm, all fingerprint vectors are normalized to unit length. This means that all training documents are assumed to have the same weight when used in model construction. This may not always be justified; notice that in longer documents the distribution of terms is likely to be more accurate than in shorter ones.

When the algorithm was run having four chunks available, and always searching for two features from each document (that is, LTM = 4 and STM = 2), the process converged

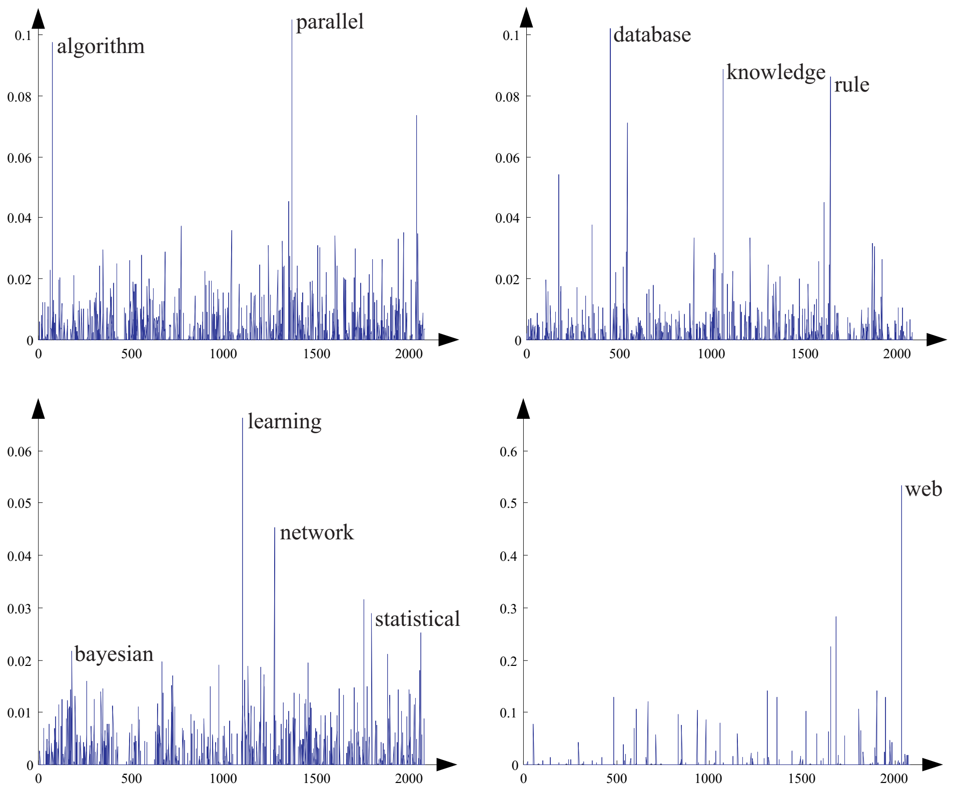


Figure 14: "Generalized keywords" — the chunks extracted by the algorithm

to a state where the chunk contents were as shown in Fig. 14. Each document can thereafter (more or less accurately) be represented as a weighted sum of these features that spanned the degrees of freedom among the documents. Below, the extracted chunks are characterized by showing the titles of the three best-matching document abstracts, being the "most prototypical" for those chunks; the chunks have also been labeled (by human inspection):

Chunk #1: *Technical implementations*

0.44	Parallel algorithms for discovery of association rules
0.38	WaveCluster: a wavelet-based clustering approach for spatial data in very large databases
0.37	High performance OLAP and data mining on parallel computers

Chunk #2 *Knowledge discovery*

0.38	Knowledge mining in databases: an integration of machine learning methodologies with database technologies
0.36	Knowledge Explorer: a prototype for mining knowledge from databases
0.35	Mining first-order knowledge bases for association rules

Chunk #3 *Models and methods*

0.33	Scoring models in Clementine for business advantage
0.32	Mathematical programming in data mining
0.32	Scalable techniques for mining causal structures

Chunk #4 *Web techniques?*

0.90	Semi-structured data extraction and schema knowledge mining
0.89	Semi-structured data extraction and schema knowledge mining
0.89	Semi-structured data extraction and schema knowledge mining

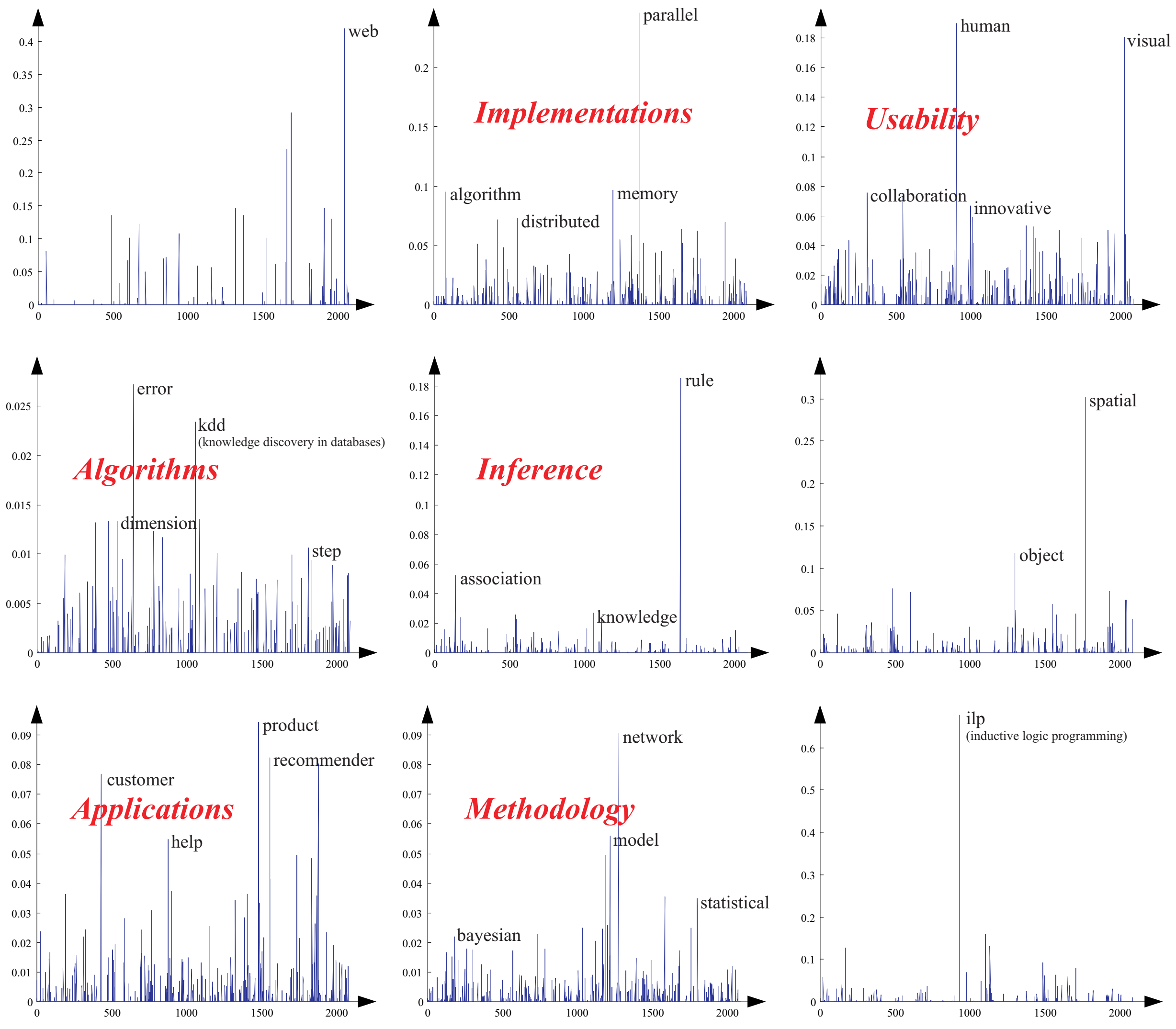


Figure 15: The chunks extracted when more prototypes are available (chunk labels being determined afterwards manually)

Note that the fourth chunk has seemingly been allocated for representing a single "outlier" document: It turns out that this document has been stored in the database various times, and its effect has evidently become very dominant. Similarly, if the most frequent words had not been compensated by TFIDF, most probably a separate chunk would have been needed to capture the bias caused by the wealth of stop-words, characterizing the "standard English language".

A larger model was also constructed: Nine chunk prototypes (located in a 3×3 grid) were employed, and three chunks were extracted from each sample. It seems that there really emerges some more sophisticated contents-related structure in the model (see Fig. 15). Note that, again, one of the chunks (chunk #1 this time) has been allocated for the same outlier as in the previous experiment. It seems that the "general relevance" (as opposed to the "outlier-likeness") of a chunk can be estimated from the sparseness of the chunk vectors: If there are only a few words with non-zero weight, the number of documents matching that chunk has been low.

The results reveal that there are problems in the model robustness: Some words are heavily weighted in the final chunks that do not have special value in distinguishing between different document contents. This means that spurious errors can be made in document modeling if the frequency of such non-informative words is changed. This problem is due to the statistical, unavoidable principles: If the data space is too high-dimensional as compared to the number of training samples, as it now is, occasional coincidences not obeying the true word frequency can have significant effect on the constructed model. To circumvent this problem, the dimension of the data space should be reduced. Some kind of *semantic terms* carrying some semantic information should be used instead of the bare words as input data. When attacking such linguistic and semantic problems (for example, see [3]) the nice simple starting point utilized here — everything being represented as real-valued vectors, and all operations being based on linear algebra and matrix calculus — does no more apply.

ACKNOWLEDGEMENT

I am grateful to Jorma Laaksonen, Dr.Tech., from the HUT Laboratory of Computer and Information Science for the hand-written digit material.

REFERENCES

[1] Basilevsky, A.: *Statistical Factor Analysis and Related Methods.* John Wiley & Sons, New York, 1994.

[2] Chen, M.-S., Han, J., and Yu, P.S.: Data Mining: An Overview from Database Perspective. *IEEE Transactions on Knowledge and Data Engineering,* 1997.

[3] Chomsky, N.: *Syntactic Structures.* Mouton, The Hague, The Netherlands, 1957.

[4] Foltz, P.W.: Using Latent Semantic Indexing for Information Filtering. *Proceedings of the Conference on Office Information Systemsm,* Cambridge, Massachusetts, 1990, pp. 40–47.

[5] Földiák, P.: Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics,* Vol. 64, 1990, pp. 165–170.

[6] Glymour, C., Madigan, D., Pregibon, D., and Smyth, P.: Statistical Themes and Lessons for Data Mining. *Data Mining and Knowledge Discovery,* Vol. 1, pp. 11–28, 1997.

[7] Haykin, S.: *Neural Networks. A Comprehensive Foundation.* Macmillan College Publishing, New York, 1994.

[8] Honkela, T., Kaski, S., Kohonen, T., and Lagus, K.: Self-Organizing Maps of Very Large Document Collections: Justification for the WEBSOM method. In *Classification, Data Analysis, and Data Highways* (eds. Balderjahn, I., Mathar, R., and Schader, M.), Springer-Verlag, Berlin, 1998, pp. 245–252.

[9] Honkela, T., Kaski, S., Lagus, K., and Kohonen, T.: *Newsgroup Exploration with WEBSOM Method and Browsing Interface.* Helsinki University of Technology, Laboratory of Computer and Information Science, Report A32, 1996.

[10] Hoyer, P.O. and Hyvärinen, A.: A Multi-Layer Sparse Coding Network Learns Contour Coding from Natural Images. *Vision Research,* Vol. 42, No. 12, pp. 1593–1605, 2002.

[11] Hyötyniemi, H.: *Constructing Non-Orthogonal Feature bases.* Proceedings of the International Conference on Neural Networks (ICNN'96), June 3–6, 1996, Washington DC, pp. 1759–1764.

[12] Hyötyniemi, H. and Saariluoma, P.: Chess — Beyond the Rules. In Timo Honkela (ed.): *Games, Computers and People (Pelit, tietokone ja ihminen),* Finnish Artificial Intelligence Society, Helsinki, Finland, 1999, pp. 100-112.

[13] Hyötyniemi, H.: On Mental Images and 'Computational Semantics'. In *Proceedings of the 8th Finnish Artificial Intelligence Conference STeP'98* (eds. Koikkalainen, P. and Puuronen, S.), Finnish Artificial Intelligence Society, Helsinki, Finland, 1998, pp. 199–208.

[14] Hyötyniemi, H.: Semantic Feature Extraction: Reader-Specific Text Document Classification. In the *Proceedings of the 8th Finnish Artificial Intelligence Conference STeP'98,* Jyväskylä, Finland, September 7–9, 1998, pp. 55–0.

[15] Hyötyniemi, H.: *Studies on Emergence and Cognition — Parts 1 & 2.* Finnish Artificial Intelligence Conference (STeP'02), December 16–17, 2002, Oulu, Finland.

[16] Kohonen, T.: *Self-Organizing Maps.* Springer-Verlag, Berlin, 1995.

[17] Laaksonen, J.: *Subspace Classifiers in Recognition of Handwritten Digits.* Acta Polytechnica Mathematica; Mathematics, Computing and Management in Engineering series No. 84, Espoo, 1997.

[18] Mannila, H.: Methods and Problems in Data Mining. *Proceedings of the International Conference on Database Theory,* Delphi, Greece, January 1997.

[19] Myllymäki, P., Silander, T., Tirri, H., and Uronen, P.: Bayesian Data Mining on the Web with B-Course. *Proceedings of the 2001 IEEE International Conference on Data Mining,* San Jose, USA, November 2001, pp. 626–629.

[20] Olshausen, B.A. and Field, D.J.: Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research,* Vol. 37, 1997, pp. 3311–3325.

[21] Salton, G. and Buckley, C.: Term Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management,* Vol. 24, No. 5, 1988, pp. 513–523.

[22] Ritter, H. and Kohonen, T.: Self-Organizing Semantic Maps. In *Biological Cybernetics* **61**, 1989, pp. 241–254.

[23] Rosch, E.: Principles of Categorization. In *Cognition and Categorization* (eds. Rosch, E. and Lloyd, B.B.), Erlbaum, Hillsdale, New Jersey, 1978, pp. 27–48.

[24] Scholtes, J.C.: Kohonen Feature Maps in Full-Text Data Bases: A Case Study of the 1987 Pravda. In *Proceedings of "Informatiewetenschap '91",* the Netherlands, December 12, 1991, pp. 203–220.