Organizing Digital Libraries by Automated Text Categorization

Henri Avancini[1], **Andreas Rauber**[2], and **Fabrizio Sebastiani**[1]

(1) Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche
Via Giuseppe Moruzzi, 1
56124 Pisa, Italy

(2) Institut für Softwaretechnik und Interaktive Systeme
Technische Universität Wien
Favoritenstraße 9-11/188
1040 Wien, Austria

Abstract

Text Categorization (TC) is the discipline concerned with the construction of automatic text classifiers, i.e. programs capable of assigning to a document one or more among a set of predefined categories based on the content of the document. Building these classifiers is itself done automatically, by means of a general inductive process that learns the characteristics of the categories from a set of preclassified documents. In this paper we discuss a class of applications, *automatic indexing with controlled vocabularies*, that is of direct concern to organizing digital libraries. We exemplify this class of applications by discussing an ongoing project aimed at classifying scientific papers about computer science with respect to the ACM Classification Scheme.

> When it was proclaimed that the Library contained all books, the first impression was one of extravagant happiness. All men felt themselves to be the masters of an intact and secret treasure. There was no personal or world problem whose eloquent solution did not exist in some hexagon. (...) As was natural, this inordinate hope was followed by an excessive depression. The certitude that some shelf in some hexagon held precious books and that these precious books were inaccessible, seemed almost intolerable.
>
> [Jorge Luis Borges, *The Library of Babel*, 1941]

1 Introduction

In the short story *The Library of Babel* Jorge Luis Borges envisions the world as a library of infinite size which contains *all* possible books, i.e. all possible sequences of symbols that can be built out of a given alphabet. Some of these sequences are totally random juxtapositions of symbols, with no morphological or syntactic well-formedness according to any known language; some others are morphologically and syntactically well-formed according to some known language, but are not meaningful according to its commonly accepted semantics; still some others are meaningful but untruthful; and some are both meaningful and truthful.

In such an all-encompassing, chaotic, unorganized library, humans are engaged in the endless quest for truth, which takes the form of the quest not of generically meaningful and truthful books, but of *the* book, the one that reveals eternal truth. Of course, if there is such an eternal truth, the Library contains such a book, since it contains them all: this justifies

the "extravagant happiness" of humans, but the formidable character of this task also justifies their "depression".

Borges' death in 1986 sadly deprived him of the chance to know about the World Wide Web, definitely the most faithful enactment of his Library that has been realized to date, and probably one of the most faithful we can ever think of. Any Web user has surely experienced the happiness and depression Borges speaks of, in first realizing that the Web contains enormous amounts of useful information just a few clicks away, and in then realizing that without appropriate tools one might need to access huge quantities of irrelevant information before hitting on the relevant items.

The disciplines of *Information Retrieval* (IR) and *Text Categorization* (TC) have attacked the problem of *information overload* from two orthogonal perspectives. While IR strives to provide *better search tools* for seeking information in an unstructured collection of documents [3], the purpose of TC is that of automatically providing *better structuring* of this collection so as to make search easier; it is on this latter discipline that this paper concentrates.

TC (see [33] for an introduction and review) is the discipline concerned with the construction of *automatic text classifiers*, i.e. programs capable of assigning to a document one or more among a set of predefined categories based on the content of the document. Building these classifiers is itself done automatically, by means of a general inductive process that learns the characteristics of the categories from a set of preclassified documents.

TC has a number of applications, some of them quite esoteric, including text filtering [26], personalized information delivery [29], word sense disambiguation [11], junk mail filtering [2, 9, 31], Web page classification under hierarchical Internet directories [10], author identification for literary texts of unknown or disputed authorship [6, 14], automated identification of text genre [13, 21, 25, 35], automated survey coding [18], and automated essay grading [24]. In this paper we discuss a class of applications, *automatic indexing with controlled vocabularies*, that is of direct concern to organizing digital libraries.

In particular, we will discuss this class of applications by drawing from a project currently ongoing at the Istituto di Scienza e Tecnologia dell'Informazione. This project, codenamed CompCat, is an internally funded project concerned with building an interactive classifier of scientific articles in the computer science domain. Here, an author submits an article to the classifier, and the classifier suggests to the author a list of categories drawn from the ACM Classification Scheme[1] (ACMCS) ranked in order of estimated suitability to the article. The method by which classifiers are built takes advantage of the hierarchical structure of the ACMCS.

This paper is organized as follows. In Section 2 we briefly introduce the basic principles and techniques of TC. Section 3 discusses the problem of using TC for organizing digital libraries by automatically tagging articles with categories from a predefined set, and illustrates a novel approach to the problem that we have developed within the CompCat project. Section 4 concludes, discussing open problems and avenues for further research.

2 A short introduction to text categorization

Text categorization (also known as *text classification*) is the task of approximating the unknown *target function* $\breve{\Phi} : \mathcal{D} \times \mathcal{C} \rightarrow \{T, F\}$ (that describes how documents ought to be classified) by means of a function $\Phi : \mathcal{D} \times \mathcal{C} \rightarrow \{T, F\}$ called the *classifier*, where $\mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}\}$ is a predefined set of categories and $\mathcal{D}$ is a domain of documents. If $\breve{\Phi}(d_j, c_i) = T$, then d_j is called a *positive example* (or a *member*) of c_i, while if $\breve{\Phi}(d_j, c_i) = F$ it is called a *negative example* of c_i.

The categories are just symbolic labels, and no additional knowledge (of a procedural or declarative nature) of their meaning is usually available. It is usually the case that no metadata (such as e.g. publication date, document type, publication source) are available either; therefore, classification must be accomplished only on the basis of knowledge extracted from the documents themselves.

Text categorization is a *subjective* task: when two experts (human or artificial) decide whether to classify document d_j under category c_i, they may disagree, and this in fact happens with relatively high frequency. A news article on Clinton attending Dizzy Gillespie's funeral

[1] `http://info.acm.org/class/1998/ccs98.html`

could be filed under Politics, or under Jazz, or under both, or even under neither, depending on the subjective judgment of the expert.

Depending on the application, TC may be either *single-label* (i.e. exactly one $c_i \in \mathcal{C}$ must be assigned to each $d_j \in \mathcal{D}$), or *multi-label* (i.e. any number $0 \leq n_j \leq |\mathcal{C}|$ of categories may be assigned to a document $d_j \in \mathcal{D}$). A special case of single-label TC is *binary* TC, in which, given a category c_i, each $d_j \in \mathcal{D}$ must be assigned either to c_i or to its complement $\overline{c}_i$. Multi-label TC under $\mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}\}$ is usually tackled as $|\mathcal{C}|$ independent binary classification problems under $\{c_i, \overline{c}_i\}$, for $i = 1, \ldots, |\mathcal{C}|$. A *classifier for* c_i is then a function $\Phi_i : \mathcal{D} \rightarrow \{T, F\}$ that approximates the unknown target function $\breve{\Phi}_i : \mathcal{D} \rightarrow \{T, F\}$.

We can roughly distinguish three different phases in the life cycle of a TC system: document indexing, classifier learning, and classifier evaluation. The three following paragraphs are devoted to these three phases, respectively; for a more detailed treatment see Sections 5, 6 and 7, respectively, of [33].

2.1 Document indexing

Document indexing denotes the activity of mapping a document d_j into a compact representation of its content that can be directly interpreted (i) by a classifier-building algorithm and (ii) by a classifier, once it has been built. The document indexing methods usually employed in TC are borrowed from IR, where a text d_j is typically represented as a vector of term *weights* $\vec{d_j} = \langle w_{1j}, \ldots, w_{|\mathcal{T}|j} \rangle$. Here, $\mathcal{T}$ is the *dictionary*, i.e. the set of *terms* (also known as *features*) that occur at least once in at least k documents, and $0 \leq w_{kj} \leq 1$ quantifies the importance of t_k in characterizing the semantics of d_j. Typical values of k are between 1 and 5.

An indexing method is characterized by (i) a definition of what a term is, and (ii) a method to compute term weights. Concerning (i), the most frequent choice is to identify terms either with the *words* occurring in the document (with the exception of *stop words*, i.e. topic-neutral words such as articles and prepositions, which are eliminated in a pre-processing phase), or with their *stems* (i.e. their morphological roots, obtained by applying a stemming algorithm). A popular choice is to add to the set of words or stems a set of *phrases*, i.e. longer (and semantically more significant) language units extracted from the text by shallow parsing and/or statistical techniques. Concerning (ii), either statistical or probabilistic techniques are used to compute terms weights, the former being the most common option. One popular class of statistical term weighting functions is $tf * idf$ (see e.g. [32]), where two intuitions are at play: (a) the more frequently t_k occurs in d_j, the more important for d_j it is (the *term frequency* intuition); (b) the more documents t_k occurs in, the less discriminating it is, i.e. the smaller its contribution is in characterizing the semantics of a document in which it occurs (the *inverse document frequency* intuition). Weights computed by $tf * idf$ techniques are often normalized so as to contrast the tendency of $tf * idf$ to emphasize long documents.

In TC, unlike in IR, a *dimensionality reduction* phase is often applied so as to reduce the size of the document representations from $\mathcal{T}$ to a much smaller, predefined number. This has both the effect of reducing *overfitting* (i.e. the tendency of the classifier to better classify the data it has been trained on than new unseen data), and to make the problem more manageable for the learning method, since many such methods are known not to scale well to high problem sizes. Dimensionality reduction often takes the form of *feature selection*: each term is scored by means of a scoring function that captures its degree of (positive, and sometimes also negative) correlation with c_i, and only the highest scoring terms are used for document representation. Alternatively, dimensionality reduction may take the form of *feature extraction*: a set of "artificial" terms is generated from the original term set (by techniques such as supervised or unsupervised term clustering, or latent semantic indexing) in such a way that the newly generated terms are both fewer and stochastically more independent from each other than the original ones used to be.

2.2 Classifier learning

A text classifier for c_i is automatically generated by a general inductive process (the *learner*) which, by observing the characteristics of a set of documents preclassified under c_i or $\overline{c}_i$, gleans the characteristics that a new unseen document should have in order to belong to c_i. In order to build classifiers for $\mathcal{C}$, one thus needs a corpus Ω of documents such that the

value of $\breve{\Phi}(d_j, c_i)$ is known for every $\langle d_j, c_i \rangle \in \Omega \times \mathcal{C}$. In experimental TC it is customary to partition Ω into three disjoint sets Tr (the *training set*), Va (the *validation set*), and Te (the *test set*). The training set is the set of documents observing which the learner builds the classifier. The validation set is the set of documents on which the engineer fine-tunes the classifier, e.g. choosing for a parameter p on which the classifier depends, the value that has yielded the best effectiveness when evaluated on Va. The test set is the set on which the effectiveness of the classifier is finally evaluated. In both the validation and test phase, "evaluating the effectiveness" means running the classifier on a set of preclassified documents (Va or Te) and checking the degree of correspondence between the output of the classifier and the preassigned labels.

Different learners have been applied in the TC literature, including probabilistic methods, regression methods, decision tree and decision rule learners, neural networks, batch and incremental learners of linear classifiers, example-based methods, support vector machines, genetic algorithms, hidden Markov models, and classifier committees. Some of these methods generate binary-valued classifiers of the required form $\Phi : \mathcal{D} \times \mathcal{C} \rightarrow \{T, F\}$, but some others generate real-valued functions of the form $CSV : \mathcal{D} \times \mathcal{C} \rightarrow [0, 1]$ (CSV standing for *categorization status value*). For these latter, a set of thresholds τ_i needs to be determined (typically, by experimentation on a validation set) allowing to turn real-valued CSVs into the final binary decisions.

2.3 Classifier evaluation

Both *training efficiency* (i.e. average time required to build a classifier Φ_i from a given corpus Ω), *classification efficiency* (i.e. average time required to classify a document by means of Φ_i), and *effectiveness* (i.e. average correctness of Φ_i's classification behaviour) are measures of success for a learner. However, effectiveness is usually considered the most important criterion, since in most applications one is willing to trade training time and classification time for correct decisions. Also, it is the most reliable one when it comes to comparing different learners, since efficiency depends on too volatile parameters (e.g. different sw/hw platforms). In TC, effectiveness is often measured by a combination of *precision* (π), the percentage of positive categorization decisions that are correct, and *recall* (ρ), the percentage of positive, correct categorization decisions that are actually taken. Since a classifier can be tuned to emphasize one at the expense of the other, only combinations of the two are significant, the most popular combination nowadays being their harmonic mean $F_1 = \frac{2\pi\rho}{\pi+\rho}$. When effectiveness is computed for several categories, the results for individual categories must be averaged in some way; here, one may opt for *microaveraging* ("categories count proportionally to the number of their positive training examples") or for *macroaveraging* ("all categories count the same"), depending on the application. The former rewards classifiers that behave well on *frequent categories* (i.e. categories with many positive training examples), while classifiers that perform well also on infrequent categories are emphasized by the latter.

3 Automatic indexing with controlled vocabularies

An important problem in building and maintaining digital libraries is *metadata generation*, i.e. the creation of the description of a given document according to the metadata standard adopted in the digital library of interest. A typical metadata standard for digital libraries (such as e.g. Dublin Core[2]) includes several *fields* (or *slots*), each of a different nature and of varying difficulty when it comes to the task of filling them automatically.

In a digital library, some fields can be filled only by drawing data from an external source; a typical example is a field Publication Source, given that the indication of the source in which a document was published does not always appear in the document itself, or cannot always be inferred by looking at its content. Other examples of such fields are Author and Title for libraries of digital photographs.

Other fields can, in principle, be filled without help from external sources, i.e. by looking at the document only. Typical examples are the Author and the Title fields in the case of scientific articles or other such textual documents, since fillers for these slots are usually present in the

[2] `http://dublincore.org/`

Machine learning in automated text categorization

FABRIZIO SEBASTIANI
Consiglio Nazionale delle Ricerche, Italy

The automated categorization (or classification) of texts into predefined categories has witnessed a booming interest (...) classifier evaluation.

Categories and Subject Descriptors: H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing—*Indexing methods*; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—*Information filtering*; H.3.4 [Information Storage and Retrieval]: Systems and Software—*Performance evaluation (efficiency and effectiveness)*; I.2.6 [Artificial Intelligence]: Learning— *Induction*

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: Machine learning, text categorization, text classification

1. INTRODUCTION

In the last 10 years content-based document management tasks ...

Figure 1: An example first page of an article published in an ACM journal.

document itself. Although the task may be non trivial when documents are heterogeneous in format, the slot fillers can thus be extracted directly from the document; techniques such as *wrapper induction* [15] may be used for this.

Note that in the case of the Author and Title fields of scientific articles, their fillers can always be unambiguously identified in the document (when they are there) by a human. Instead, there are fields for which this is not the case: for instance, the Additional Key Words and Keyphrases field used in the description of articles published in ACM journals (see the example in Figure 1) constitutes a harder problem, since it should be filled with the keywords and keyphrases contained in the article that are *most representative* of the article's content. The task of their identification is obviously subjective, so that more complex *keyphrase extraction* techniques must be employed [36].

A still trickier problem (and the one we concentrate on in this section) is represented by fields such as the Categories and Subject Descriptors field used in ACM journals (see again Figure 1), which must be filled by means of one or more categories drawn from the ACMCS. The reason why the problem is trickier is that a category such as "H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing—*Indexing methods*" may be attached to an article even if none of the expressions "Information Storage and Retrieval", "Content Analysis and Indexing" and "Indexing methods" (i.e. the labels traversed in the root-to-leaf path in the category tree) occur in the text of the article: this kind of medatada cannot thus be *extracted* from the document, but must be *generated* following a process that involves some kind of understanding (i) of the content of the document and (ii) of the semantics of the categories belonging to the chosen classification scheme.

This latter metadata generation problem is known as the problem of *automatic indexing with controlled vocabularies*. This was a hugely important problem up to the late '80s, since until then information retrieval systems were mostly of a Boolean nature, and thus required documents to be represented (i.e. "indexed") by lists of categories drawn from a predefined set, a so-called *controlled vocabulary*, often consisting of a thematic hierarchical thesaurus (e.g.

the NASA thesaurus for the aerospace discipline[3], or the MESH thesaurus for medicine[4]). Since the attribution of the appropriate categories to a document was not within reach of the technology of the time, this attribution used to be performed manually by trained professionals, a slow and expensive process. The costs involved in this process encouraged the research in automatic means of indexing documents by means of controlled vocabularies. The seminal work by Maron [27] is universally considered as the birth of TC, and automatic indexing with controlled vocabularies is the application that has spawned most of the early research in TC [4, 12, 19, 20]. Various text classifiers have been explicitly conceived for document indexing even in recent times [17, 30, 37].

Recalling Section 2, note that automatic indexing with controlled vocabularies is typically a multi-label TC task.

3.1 Indexing under hierarchically structured classification schemes

CompCat is an internally funded, ongoing project at Istituto di Elaborazione dell'Informazione, concerned with building an interactive classifier of scientific articles about computer science, with categories being drawn from the ACMCS. This latter is a tree-shaped hierarchy consisting of 1474 categories among which 258 are internal nodes and 1216 are leaves; its maximum depth is 4, and documents can be classified both in (third-level) internal nodes (e.g. categories such as "H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing") and/or in leaves (e.g. categories such as "H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing—*Indexing methods*") of the tree[5].

The classifier is to be used within the ERCIM Technical Reference Digital Library (ETRDL), a networked digital library of technical reports in the computer science and mathematics domains [1]. The idea that underlies this project is to provide an interactive tool by means of which an author wishing to contribute a technical report to the digital library can receive support in deciding the ACMCS categories that she should attribute to the paper. Usually, authors do not have an in-depth knowledge of the ACMCS, and studying it in order to pick the most appropriate categories for the paper is for them a time-consuming process. As a consequence, an interactive tool that, after receiving the paper as input, returns a list of ACMCS categories ranked in order of estimated appropriateness to the paper, is going to be a useful tool, since the author needs only scan the top part of the list, picking appropriate entries and stopping when satisfied.

It would certainly be possible to build a tool that classifies scientific papers under the ACMCS by means of the basic techniques described in Section 2 (and in Section 2.2 in particular), by simply building a binary classifier for each category c_i. However, this naive approach fails to exploit the hierarchical structure of the ACMCS, treating it as a "flat" set of categories. Notwithstanding the ubiquity of hierarchically organized category sets, TC techniques that exploit this structure (a task that we will call *hierarchical text categorization*) have been rare, the first attempts dating to the mid '90s [38]. Omitting to exploit this hierarchical structure is inadequate, since it is quite intuitive that the relationships of dependence (e.g. "more general than", "more specific than", "sibling of", etc.) between the categories belonging to the catalogue could provide a valuable source of information for the classifier learning task.

The method we use in CompCat is based on the "shrinkage" method for hierarchical text categorization introduced by McCallum et al. [28], but exploits it in a novel way by combining it with a hierarchical clustering method.

Before describing the original shrinkage method and our own variant of it, we first introduce the naive Bayesian classification method, which shrinkage uses as basic learner.

[3] `http://www.sti.nasa.gov/nasa-thesaurus.html`

[4] `http://www.nlm.nih.gov/mesh/meshhome.html`

[5] In this paper we will always refer to the currently latest version of the ACMCS, which dates back to 1998. Also, note that we will not strictly adhere to the terminology of the ACMCS, where a distinction is made between "subject descriptors" (that basically correspond to third- or fourth-level leaves) and "categories" (all the other nodes); since the distinction does not have a functional character, we will use the name "categories" for all of them.

3.2 Naive Bayesian classification

Naive Bayesian classification is usually achieved by training a probabilistic classifier for each category c_i of interest[6]. Given d_j, a probabilistic classifier for c_i outputs a value $P(c_i|\vec{d_j})$ denoting the probability that a random document with representation $\vec{d_j} = \langle w_{1j}, \ldots, w_{|\mathcal{T}|j} \rangle$ belongs to c_i. In what follows, we will assume that our task consists in individuating the *leaf* categories into which document d_j should be classified[7]. The classification of d_j thus consists in choosing the leafs c_k for which $P(c_k|\vec{d_j})$ is maximized.

McCallum et al. [28] address the problem through the use of a *Bernoulli model of document generation*, in which each c_i is represented by a "coin" with $|\mathcal{T}|$ faces (one face for each $t_k \in \mathcal{T}$), each characterized by a probability of occurrence θ_{ik} such that $\sum_{k=1}^{|\mathcal{T}|} \theta_{ik} = 1$ for all $c_i \in |\mathcal{C}|$. A document d_j of length n is obtained by tossing this coin n times, and is represented by a vector $\vec{d_j} = \langle w_{1j}, \ldots, w_{|\mathcal{T}|j} \rangle$ where w_{kj} represents the number of times t_k occurs in d_j (therefore, $n = \sum_{k=1}^{|\mathcal{T}|} w_{kj}$). The probability that a document with $\vec{d_j}$ as representation is generated, given that coin c_i is used, is

$$P(\vec{d_j}|c_i) \quad = \quad P(n) \frac{n!}{w_{1j}! w_{2j}! \ldots w_{|\mathcal{T}|j}!} \prod_{k=1}^{|\mathcal{T}|} \theta_{ik}^{w_{kj}} \tag{1}$$

where $P(n)$ denotes the probability that the coin is tossed exactly n times. For estimating the θ_{ik} parameters one can use plain *maximum likelihood* (ML) estimation, i.e.

$$\hat{\theta}_{ik} = \frac{\sum_{d_j \in c_i} w_{kj}}{\sum_{t_r \in \mathcal{T}} \sum_{d_j \in c_i} w_{rj}} \tag{2}$$

It is now easy to compute $P(c_i|\vec{d_j})$ through Bayes' theorem, i.e.

$$P(c_i|\vec{d_j}) = \frac{P(\vec{d_j}|c_i)P(c_i)}{P(\vec{d_j})} = \frac{P(\vec{d_j}|c_i)P(c_i)}{\sum_{c_r \in \mathcal{C}} P(\vec{d_j}|c_r)} \tag{3}$$

where $P(c_i)$ can be estimated as $\frac{1 + |\{d_j \in c_i\}|}{|\mathcal{C}| + \sum_{c_r \in \mathcal{C}} |\{d_j \in c_r\}|}$ and $P(\vec{d_j}|c_i)$ is given by (1). Note that for the purpose of computing (3) it is not necessary to estimate the $P(n)$ factor in (1), since this factor would appear both at the numerator and at the denominator of (3), and would thus disappear from it.

3.3 Improving parameter estimation by shrinkage

McCallum et al. [28] have proposed the use of a technique called *shrinkage* for improving the effectiveness of probabilistic naive Bayesian text classifiers. Shrinkage is a technique from statistics that allows a more robust estimation of parameters than ML estimation or its variants (such as ones obtained by *Laplace smoothing*). The purpose of shrinkage is to obtain better estimates of θ_{ik} than the ones obtained by (2), since this latter may yield unreliable estimates if c_i is an infrequent category. This is typically the case with leaf categories in a hierarchy with many classes and fine-grained distinctions between leaves. If c_i is a leaf distant δ steps from the root, shrinkage builds more robust estimates of θ_{ik} (indicated as $\check{\theta}_{ik}$) by interpolating the ML estimates $\hat{\theta}_{ik}$ obtained for c_i with the ML estimates $\hat{\theta}_{ik}^r$ obtained for its ancestors $\pi^r(c_i)$, for $r \in \{1, \ldots, \delta\}$. This means computing

$$\check{\theta}_{ik} = \sum_{r=0}^{\delta+1} \lambda_i^r \hat{\theta}_{ik}^r \tag{4}$$

where $\hat{\theta}_{ik}^0 = \hat{\theta}_{ik}$, the $\hat{\theta}_{ik}^r$ estimates are obtained according to (2), and the λ_i^r are the interpolation weights, with $\sum_{r=0}^{\delta+1} \lambda_i^r = 1$. Note that (4) assumes the existence of a "virtual" parent

[6] See [33, Section 6.2] for an introduction to and review of research on probabilistic TC.

[7] Although ACMCS also allows the tagging of an article by an internal node whose children are leaves, we will assume, without loss of generality, that tagging d_j by such a node is achieved by tagging d_j by all its children leaves.

$\pi^{\delta+1}(c_i)$ of the root characterized by the uniform estimate, i.e. such that $\hat{\theta}_{ik}^{\delta+1} = \frac{1}{|\mathcal{T}|}$ for all $t_k \in \mathcal{T}$; this is done in order to smooth the parameters for those terms that are rare also in the root category (i.e. in the entire training set), and eliminates the need for Laplace smoothing. The λ_i^r weights are determined by applying a variant of the expectation maximization (EM) algorithm [5] on a validation set.

The ML estimate $\hat{\theta}_{ik}$ is the most specific to c_i, because it is based on data from c_i alone, but it is not robust, since these data are typically few. Symmetrically, the estimate $\hat{\theta}_{ik}^{\delta}$ is the least specific to c_i, since it is also based on data loosely related to c_i, but it is more robust, since it is based on a very large dataset. Shrinkage thus trades specificity for robustness by combining them all. In order to maximize the independence of the ML estimates that are combined by (4), the ML estimate $\hat{\theta}_{ik}^{r}$ for $\pi^r(c_i)$ is computed by using, instead of all the data from $\pi^r(c_i)$, the data from $\pi^r(c_i)$ minus the data from its child $\pi^{r-1}(c_i)$.

McCallum and colleagues have found that this model improves the effectiveness of naive Bayesian classification especially for those categories for which few positive training instances are available; in these cases, the estimates of the naive Bayesian parameters are unreliable, and shrinkage can help in making them more robust. This is important for our application, since the ACMCS exhibits fine-grained distinctions among categories, and because of this many leaf categories are scarcely populated.

3.4 Replacing the ACMCS with an artificially generated hierarchy

In the CompCat project we exploit the added power of shrinkage over naive Bayesian classification in a novel way. Our key idea is that of discarding the naturally occurring ACMCS hierarchy and generating an artificial hierarchy H that has better statistical properties. While a user of the ETRDL digital library will keep referring to ACMCS (and indeed will have no knowledge even of the existence of H), the learning algorithm uses H instead of ACMCS, with the aim of obtaining better classification effectiveness. The two hierarchies have different internal nodes, different maximal depth, and different structure in general, but have exactly the same leaves.

How can we obtain a hierarchy with better statistical properties than the naturally occurring one? For the purposes of this work, a measure of goodness of the statistical properties of a hierarchy will be the gain in effectiveness that shrinkage on the hierarchy achieves over pure naive Bayesian classification. The main hypothesis that underlies our work is that a naturally occurring hierarchy such as ACMCS *is optimized for human use, but not necessarily for classifier use*. In fact, the structure of the ACMCS was designed by humans for humans, with the aim of subdividing the field of computer science into disciplines and subdisciplines in a way that reflects the perception of workers in the field and that optimizes the perusal of the hierarchy by its users. We instead aim at subdividing the field *in a way that reflects the perception of statistically-motivated text management algorithms*. This will be done for the sole purpose of achieving a better classification behaviour on the ACMCS (natural) hierarchy; the fact that the two hierarchies have exactly the same leaves guarantees that the user can ignore the existence of H, since the documents are eventually classified in the leaves, which are also leaves of ACMCS.

Let us imagine that in order to achieve the subdivision of the computer science field into disciplines and subdisciplines the designers of the ACMCS have worked in a bottom-up way, grouping disciplines they recognized to be similar into superdisciplines, and so on. A similar procedure could be performed automatically by a clustering method, provided a measure of similarity between the individual items (in our case: the leaf categories) is defined. The key difference between the two hierarchies is thus the notion of similarity that is being used; a notion based on "human semantics" in the former case, and a notion based on statistical word distributions in the latter. Since it is statistical word distributions that both naive Bayesian classification and shrinkage use, intuition suggests that it is a hierarchy of the latter kind that shrinkage would best profit from.

We generate an artificial hierarchy by first learning a naive Bayesian classifier for each (leaf or nonleaf) category c_i and then clustering these classifiers (by means of the technique we describe in Section 3.5); this superimposes a hierarchical structure on the set of categories, recursively grouping "similar" categories into supercategories.

3.5 Creating a hierarchical view of the classifiers

Several clustering approaches for the detection of hierarchical structure within the input data are available. Examples of these are all flavours of the single- or complete-linkage clustering family, graph-based models relying on a minimal spanning tree, and others.

For our initial experiments we have chosen the *Growing Hierarchical Self-Organizing Map* algorithm [7], a fully adaptive, hierarchical extension of the *Self-Organizing Map* method [22]. The main reasons for this choice are the convenient ways of analyzing and interpreting the resulting structure due to the overall topological organization of the clusters provided by this neural network model, plus the flexibility and automatic adaptability of the approach. Furthermore, the GHSOM has frequently been employed successfully in the text clustering domain, scaling well to the typically very high-dimensional, sparse feature spaces [8].

3.5.1 Self-Organizing Maps

The GHSOM can be viewed as a divisive stochastic model, conventionally using hard assignment of data points onto clusters. Its principles are based on the *Self-Organizing Map* (SOM) [23], a popular unsupervised neural network model providing a topology-preserving mapping from a high-dimensional feature space onto a two-dimensional output space in such a way that similar data points are located spatially close together on the resulting map. The SOM consists of neural processing units, commonly arranged in the form of a rectangular grid. Each of the units i is assigned an n-dimensional model vector $m_i = [\mu_{i1}, \mu_{i2}, ..., \mu_{in}]^T, m_i \in \Re^n$, where n is the dimensionality of the feature space to be analyzed. The training process of SOMs may be described in terms of input pattern presentation and model vector adaptation. Each training iteration t starts with the random selection of one input pattern $x = [\xi_1, \xi_2, \ldots, \xi_n]^T$, $x \in \Re^n$. This input pattern is presented to the SOM and each unit determines its activation following some activation function (for which we use the Euclidean distance), selecting the unit with the smallest distance as the *winner* c of the current learning iteration:

$$c : m_c(t) = \min_i ||x(t) - m_i(t)|| \tag{5}$$

Subsequently, the model vector of the winner, as well as those of units in a time-decreasing neighborhood of c are adapted in order to more closely represent the presented input signal. This adaption is performed as a gradual reduction of the difference between the corresponding components of the input pattern and the weight vector as given in Expression 6, where $\alpha(t)$ represents a monotonically decreasing learning rate, and $h_{ci}(t)$ an over time decreasing neighborhood function centered on the winner and monotonically decreasing with increasing distance from the winner.

$$m_i(t+1) = m_i(t) + \alpha(t) \cdot h_{ci}(t) \cdot [x(t) - m_i(t)] \tag{6}$$

A Gaussian may be used to model the neighbourhood function, i.e.

$$h_{ci}(t) = \exp\left(-\frac{||r_c - r_i||^2}{2\sigma^2(t)}\right) \tag{7}$$

with r_i representing the two-dimensional vector pointing to the location of unit i within the grid, and $||r_c - r_i||$ denoting the distance between units c, i.e. the winner of the current training iteration, and i in terms of the output space.

It is common practice that at the beginning of training a wide area of the output space is subject to adaptation. The spatial width of units affected by adaptation is reduced gradually during the training process. Such a strategy allows the formation of large clusters at the beginning and fine-grained input discrimination towards the end of the training process. The spatial width of adaptation is guided by means of the time-varying parameter σ.

3.5.2 Growing Hierarchical Self-Organizing Maps

While the SOM has been successfully applied in numerous domains of data analysis, two short-comings have to be noted, one being the fixed size of the output grid, the definition of

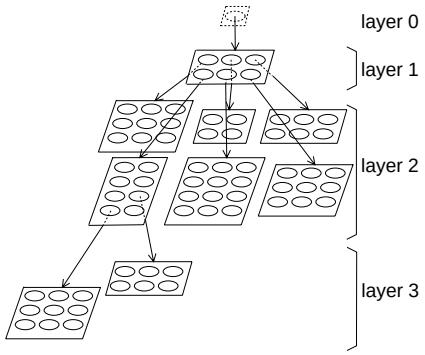

Figure 2: GHSOM reflecting the hierarchical structure of the input data.

which actually depends on a-priori unknown input data requirements, the second being the limited support for the detection of hierarchical structures.

The key idea of the GHSOM [7] is to use a hierarchical structure of multiple layers, where each layer consists of a number of independent SOMs. One SOM is used at the first layer of the hierarchy. For every unit in this map a SOM might be added to the next layer of the hierarchy which is trained only with the subset of data items mapped onto the unit it derives from, thus effectively splitting the data space. This principle is repeated with the third and any further layers of the GHSOM.

To overcome the limitations of the fixed network architecture we rather use an incrementally growing version of the SOM, similar in spirit to the *Growing Grid* model [16]. This relieves us from the burden of predefining the network's size, which is instead determined during the unsupervised training process.

Training commences with a layer 0, consisting of only one single unit. The model vector of this unit is initialized as the average of all input data. The training process basically starts with a small map of usually 2×2 units in layer 1, which is self-organized according to the standard SOM training algorithm. This training process is repeated for a fixed number λ of training iterations. After λ training iterations the unit with the largest deviation between its weight vector and the input vectors represented by this very unit is selected as the error unit. In between the error unit and its most dissimilar neighbor in terms of the input space, either a new row or a new column of units is inserted. The model vectors of these new units are initialized as the average of their neighbors.

An obvious criterion to guide the training process is the *quantization error* q_i, calculated as the sum of the distances between the model vector of a unit i and the input vectors mapped onto this unit, or its respective *mean quantization error* (mqe_i). It is used to determine the general data representation quality of a map as identified by its overall quantization error QE, defined as the sum of the units' individual q_i's, or again its respective mean over all units (MQE). A map grows until its QE is reduced to a certain fraction τ_1 of the q_i of the unit i in the preceding layer of the hierarchy. Thus, the map now represents the data mapped onto the higher layer unit i in more detail.

As outlined above the initial architecture of the GHSOM consists of one SOM. This architecture is expanded by another layer in case of dissimilar input data being mapped on a particular unit. These units are identified by a rather high quantization error q_i which is above a threshold τ_2. This threshold basically indicates the desired granularity level of data representation as a fraction of the initial quantization error at layer 0. In such a case, a new map will be added to the hierarchy and the input data mapped on the respective higher layer unit are self-organized in this new map, which again grows until its QE is reduced to a fraction τ_1 of the respective higher layer unit's quantization error q_i. Note that this usually will not lead to a balanced hierarchy. The depth of the hierarchy will rather reflect the diversity in input data distribution which we will find in our data collection. Depending on the desired fraction τ_1 of QE reduction we may end up with either a very deep hierarchy with small maps, a flat structure with large maps, or (in the most extreme case) only one large map. The growth of the hierarchy is terminated when no further units are available for expansion. A graphical representation of a GHSOM is given in Figure 2. The map in layer 1 consists of

3×2 units and provides a rough organization of the main clusters in the input data. The six independent maps in the second layer offer a more detailed view on the data. Two units from one of the second layer maps have further been expanded into third-layer maps to provide sufficiently granular input data representation.

4 Conclusion

We have described an application, automatic indexing with controlled vocabularies, that is of direct concern to the problem of creating a digital library, and we have exemplified the problem by referring to an ongoing project concerned with building an interactive classifier of scientific articles in the computer science domain.

For the construction of such an automated classifier we are using a novel combination of our own implementation of the "shrinkage" parameter estimation model for naive Bayesian classifiers with a hierarchical version of the SOM clustering method. This allows to exploit the additional information provided by the hierarchical structure of the category set, while at the same time benefitting from the good statistical properties of a hierarchy automatically generated through clustering.

Acknowledgements

We are grateful to William Arms and Donna Bergmark for providing us the corpus of documents from the ACM Digital Library with which the CompCat experiments are being carried out.

References

[1] Antonella Andreoni, Maria Bruna Baldacci, Stefania Biagioni, Carlo Carlesi, Donatella Castelli, Pasquale Pagano, and Carol Peters. Developing a European technical reference digital library. In *Proceedings of ECDL-99, 3rd European Conference on Research and Advanced Technology for Digital Libraries*, pages 343–362, Paris, FR, 1999. Lecture Notes in Computer Science, number 1696, Springer Verlag, Heidelberg, DE.

[2] Ion Androutsopoulos, John Koutsias, Konstandinos V. Chandrinos, and Constantine D. Spyropoulos. An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages. In Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval*, pages 160–167, Athens, GR, 2000. ACM Press, New York, US.

[3] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, Reading, US, 1999.

[4] Harold Borko and Myrna Bernick. Automatic document classification. *Journal of the Association for Computing Machinery*, 10(2):151–161, 1963.

[5] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[6] Joachim Diederich, Jörg Kindermann, Edda Leopold, and Gerhard Paaß. Authorship attribution with support vector machines. *Applied Intelligence*, 19(1/2):109–123, 2003.

[7] Michael Dittenbach, Dieter Merkl, and Andreas Rauber. The growing hierarchical self-organizing map. In *Proceedings of IJCNN-00, 11th International Joint Conference on Neural Networks*, volume VI, pages 15–19, Como, IT, 2000. IEEE Computer Society.

[8] Michael Dittenbach, Andreas Rauber, and Dieter Merkl. 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. In *Proceedings of DEXA-01, 12th International Conference on Database and Expert Systems Applications*, Münich, DE, 2001. Springer Verlag, Heidelberg, DE.

[9] Harris Drucker, Vladimir Vapnik, and Dongui Wu. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054, 1999.

[10] Susan T. Dumais and Hao Chen. Hierarchical classification of Web content. In Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval*, pages 256–263, Athens, GR, 2000. ACM Press, New York, US.

[11] Gerard Escudero, Lluís Màrquez, and German Rigau. Boosting applied to word sense disambiguation. In Ramon López De Mántaras and Enric Plaza, editors, *Proceedings of ECML-00, 11th European Conference on Machine Learning*, pages 129–141, Barcelona, ES, 2000. Springer Verlag, Heidelberg, DE. Published in the "Lecture Notes in Computer Science" series, number 1810.

[12] B.J. Field. Towards automatic indexing: automatic assignment of controlled-language indexing and classification from free indexing. *Journal of Documentation*, 31(4):246–265, 1975.

[13] Aidan Finn, Nicholas Kushmerick, and Barry Smyth. Genre classification and domain transfer for information filtering. In Fabio Crestani, Mark Girolami, and Cornelis J. van Rijsbergen, editors, *Proceedings of ECIR-02, 24th European Colloquium on Information Retrieval Research*, pages 353–362, Glasgow, UK, 2002. Springer Verlag, Heidelberg, DE. Published in the "Lecture Notes in Computer Science" series, number 2291.

[14] Richard S. Forsyth. New directions in text categorization. In Alex Gammerman, editor, *Causal models and intelligent data management*, pages 151–185. Springer Verlag, Heidelberg, DE, 1999.

[15] Dayne Freitag and Nicholas Kushmerick. Boosted wrapper induction. In *Proceedings of AAAI-00, 17th Conference of the American Association for Artificial Intelligence*, pages 577–583, Austin, US, 2000.

[16] Bernd Fritzke. Growing Grid – A self-organizing network with constant neighborhood range and adaption strength. *Neural Processing Letters*, 2(5):1–5, 1995.

[17] Norbert Fuhr and Gerhard Knorz. Retrieval test evaluation of a rule-based automated indexing (AIR/PHYS). In Cornelis J. van Rijsbergen, editor, *Proceedings of SIGIR-84, 7th ACM International Conference on Research and Development in Information Retrieval*, pages 391–408, Cambridge, UK, 1984. Cambridge University Press.

[18] Daniela Giorgetti, Irina Prodanof, and Fabrizio Sebastiani. Mapping an automated survey coding task into a probabilistic text categorization framework. In Elisabete Ranchhod and Nuno J. Mamede, editors, *Advances in Natural Language Processing*, pages 115–124. Springer Verlag, Heidelberg, DE, 2002. Published in the "Lecture Notes in Computer Science" series, number 2389.

[19] W. A. Gray and A. J. Harley. Computer-assisted indexing. *Information Storage and Retrieval*, 7(4):167–174, 1971.

[20] H.S. Heaps. A theory of relevance for automatic document classification. *Information and Control*, 22(3):268–278, 1973.

[21] Brett Kessler, Geoff Nunberg, and Hinrich Schütze. Automatic detection of text genre. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of ACL-97, 35th Annual Meeting of the Association for Computational Linguistics*, pages 32–38, Madrid, ES, 1997. Morgan Kaufmann Publishers, San Francisco, US.

[22] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[23] Teuvo Kohonen. *Self-organizing maps*. Springer Verlag, Heidelberg, DE, 1995.

[24] Leah S. Larkey. Automatic essay grading using text categorization techniques. In W. Bruce Croft, Alistair Moffat, Cornelis J. van Rijsbergen, Ross Wilkinson, and Justin Zobel, editors, *Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval*, pages 90–95, Melbourne, AU, 1998. ACM Press, New York, US.

[25] Yong-Bae Lee and Sung H. Myaeng. Text genre classification with genre-revealing and subject-revealing features. In Micheline Beaulieu, Ricardo Baeza-Yates, Sung Hyon Myaeng, and Kalervo Järvelin, editors, *Proceedings of SIGIR-02, 25th ACM International Conference on Research and Development in Information Retrieval*, pages 145–150, Tampere, FI, 2002. ACM Press, New York, US.

[26] David D. Lewis. The TREC-4 filtering track: description and analysis. In Donna K. Harman and Ellen M. Voorhees, editors, *Proceedings of TREC-4, 4th Text Retrieval Conference*, pages 165–180, Gaithersburg, US, 1995. National Institute of Standards and Technology, Gaithersburg, US.

[27] M.E. Maron. Automatic indexing: an experimental inquiry. *Journal of the Association for Computing Machinery*, 8(3):404–417, 1961.

[28] Andrew K. McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In Jude W. Shavlik, editor, *Proceedings of ICML-98, 15th International Conference on Machine Learning*, pages 359–367, Madison, US, 1998. Morgan Kaufmann Publishers, San Francisco, US.

[29] Michael J. Pazzani and Daniel Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997.

[30] Stephen E. Robertson and P. Harding. Probabilistic automatic indexing by learning from human indexers. *Journal of Documentation*, 40(4):264–270, 1984.

[31] Georgios Sakkis, Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Constantine D. Spyropoulos, and Panagiotis Stamatopoulos. Stacking classifiers for anti-spam filtering of e-mail. In Lillian Lee and Donna Harman, editors, *Proceedings of EMNLP-01, 6th Conference on Empirical Methods in Natural Language Processing*, pages 44–50, Pittsburgh, US, 2001. Association for Computational Linguistics, Morristown, US.

[32] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988. Also reprinted in [34], pp. 323–328.

[33] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[34] Karen Sparck Jones and Peter Willett, editors. *Readings in information retrieval*. Morgan Kaufmann, San Mateo, US, 1997.

[35] Efstathios Stamatatos, Nikos Fakotakis, and George Kokkinakis. Automatic text categorization in terms of genre and author. *Computational Linguistics*, 26(4):471–495, 2000.

[36] Peter D. Turney. Learning algorithms for keyphrase extraction. *Information Retrieval*, 2(4):303–336, 2000.

[37] Konstadinos Tzeras and Stephan Hartmann. Automatic indexing based on Bayesian inference networks. In Robert Korfhage, Edie Rasmussen, and Peter Willett, editors, *Proceedings of SIGIR-93, 16th ACM International Conference on Research and Development in Information Retrieval*, pages 22–34, Pittsburgh, US, 1993. ACM Press, New York, US.

[38] Erik D. Wiener, Jan O. Pedersen, and Andreas S. Weigend. A neural network approach to topic spotting. In *Proceedings of SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval*, pages 317–332, Las Vegas, US, 1995.