Categorizing Web Documents in Hierarchical Catalogues

Ingo Frommholz
University of Dortmund, Germany
frommhol@ls6.cs.uni-dortmund.de

Abstract

Automatic categorization of web documents (e.g. HTML documents) denotes the task of automatically finding relevant categories for a (new) document which is to be inserted into a web catalogue like *Yahoo!*. There exist many approaches for performing this difficult task. Here, special kinds of web catalogues, those whose category scheme is hierarchically ordered, are regarded. A method for using the knowledge about the hierarchy to gain better categorization results is discussed. This method can be applied in a post-processing step and therefore be combined with other known (non-hierarchical) categorization approaches.

1 Introduction

The Internet, mainly the World Wide Web and the Usenet, offers a lot of information to the interested user. The number of documents accessible via the net is growing rapidly. To manage this chaotic state, engines like AltaVista[1] or *Yahoo!*[2] offer mechanisms to search for the documents that the user needs. Some of them, like AltaVista, let the user type in keywords describing the desired document. Others, like *Yahoo!*, put the documents into a hierarchically ordered category scheme so that the user can browse through these categories to satisfy his information needs.

Categorization of web documents (e.g. HTML documents) denotes the task of finding relevant categories for a (new) document which is to be inserted into such a web catalogue. This is mostly done manually. But the large number of new documents which appear on the World Wide Web and need to be categorized raises the question of whether and how this task can be performed automatically. Since information filtering and categorization are closely related, and information filtering and information retrieval are two sides of the same coin ([1]), automatic categorization can be done using well-known means of classical IR. If there is a category scheme and documents that are sorted into this given scheme, a document which is to be categorized can be regarded as a query (the *query document*) to this collection.

The categorization of web documents is performed in two steps. In the indexing step, the documents in the collection as well as the query documents are transformed into their document description, which can be e.g. a vector of term weights for each document. In the classification step, one possibility is to create a ranking of the categories according to the query document. From this ranking, a classification decision is made (e.g. to assign the first ranked categories to the query document). There are many approaches for performing the tasks in these two steps. For example, the documents can be indexed using the well-known $tf \times idf$ approach ([12]), or by using a probabilistic, description-oriented approach as described in [6], where relevance descriptions and linear regression are used to achieve an indexing function which calculates the desired term weights. The classification is performed by a *classifier*. A classifier inputs a document and outputs a category. In [15], a k-nearest-neighbour (kNN) classifier is presented. A probabilistic interpretation of this is introduced in [6]. Here, a ranking of categories C with respect to a the query document d is created by estimating the probability $P(d \rightarrow C)$, using the non-classical logic described in [13] and [14]. [16] contains a survey of other known classifiers.

All the methods described above lack the consideration of an underlying hierarchical structure. But Internet collections like *Yahoo!* provide a hierarchical category scheme. The question is whether and how the knowledge about

[1] http://www.altavista.digital.com/
[2] http://www.yahoo.com/

the hierarchy can be exploited for improving effectiveness in document classification. There exist several approaches on this task. Ruiz and Srinivasan ([11]) use a hierarchical neural network, consisting of gate and expert nodes. Each internal node in the hierarchy is a gate node, and each leaf node in the hierarchy an expert node. Beginning from the root node, each gate node receives an input x (the document vector) and is set to true or false, depending whether the document contains the concept represented by the node or not. On a path from the root node to a leaf node, the expert is activated if every gate node on this path is set to true. Then, the activated expert node decides if it assigns the document to the category (represented by the expert node) or not. Chakrabarti et. al. ([2]) exploit the hierarchy for the estimation of the probability $P(C|d)$ for every leaf category C. Here, for a path $C_1, \ldots, C_k = C$ from a root node C_1 to a leaf node C, $P(C_1|d) = 1$ since the root category subsumes all other categories. $P(C|d)$ is calculated with $P(C_i|d) = P(C_{i-1}|d)\,P(C_i|C_{i-1}, d)$ (see [2] for further details). Furthermore, their approach filters out shared jargon at each level of the hierarchy. For example, the term "sport" might be useful to distinguish between the "Sports" and the "Computers" category on a high level in the hierarchy. For the subcategories of the "Sports" category, this term is nearly useless for discrimination. Thus, on this lower level, other terms must be found to distinguish between the "Sports" subcategories. A similar idea is used by Koller and Sahami ([8]). The classification task is divided into a set of smaller classification tasks by producing some split in the classification hierarchy. For the "Sports" example above, the classifier would first distinguish documents about sports from documents about computers. Then, if the classifier has decided that a document is about sports, a decision in the subcategories of "Sports" is made (using another, adapted set of terms for discrimination, filtering shared jargon). This approach has the advantage that subcategories of rejected top-level-categories will not be considered. The drawback is that once a wrong decision on the top level is made, this can not be corrected. McCallum et. al. ([9]) use a technique called "shrinkage" for achieving better classification results. Hierarchy information is used for the estimation of the probability $P(w_t|C_j)$ of word w_t given the class C_j. On a path from a parent node C to C_j in the hierarchy, all nodes on this path are involved in the estimation of $P(w_t|C_j)$. $P(w_t|C_j)$ itself is used to calculate the probability $P(C_j|d)$; the class with the highest probability is selected by the classifier. The advantage of this approach is that it deals well with collections having only a small number of training examples. Dumais and Chen ([3]) use Support Vector Machines with a decision threshold p. The hierarchy is exploited by using a multiplicative scoring function and a sequential Boolean approach. For example, if there are two hierarchy levels, $P(C_1) \cdot P(C_2)$ (with C_1 on the first (top) level and C_2 direct subcategory) is calculated; this is the multiplicative approach. The category on the second level with the highest multiplicative probability is chosen by the classifier. In the sequential Boolean approach, $P(C_1)\&\&P(C_2)$ is computed. Similar to the approach in [8], large numbers of second level categories do not need to be tested, since both of the constraints $P(C_1) > p$ and $P(C_2) > p$ must be satisfied.

The hierarchical approach described in this paper is different from those presented above. All of these approaches deal with hierarchies that have a tree structure. Our approach can theoretically deal with all hierarchies forming an acyclic graph[3]. Furthermore, in the hierarchical approaches described above, documents are only classified into categories that are leaf nodes in the underlying hierarchy tree. But collections like *Yahoo!* also have documents inserted into their inner categories. The approach presented here considers this fact, being able to categorize a document into an inner category. Another difference between our approach and the other approaches is that the hierarchy is considered in a post-processing step. Given the results achieved by a non-hierarchical classifier, our hierarchical classifier can calculate new results based on which a (hopefully) better classification decision can be made. This has advantages and disadvantages. The advantage is that the results of other, non-hierarchical classifiers might be reused and improved. The disadvantage is that a pre-classification by a non-hierarchical classifier is needed, which has a negative effect on overall performance.

The classifier estimates and uses the probability $P(C \rightarrow C')$ that a category C implies another category C' (also using the non-classical logic from [13, 14]). The classifier has been evaluated using the results gained with the megadocument approach introduced in [7]. So this approach will be described first. Then, the notion of global and local implication probability is defined, which is used for estimating $P(C \rightarrow C')$. It is shown how these approximations of $P(C \rightarrow C')$ can be calculated. Finally, experiments and their results are described and conclusions are presented.

[3]Theoretically because, for efficiency reasons, we had to use a tree-structured collection for our experiments.

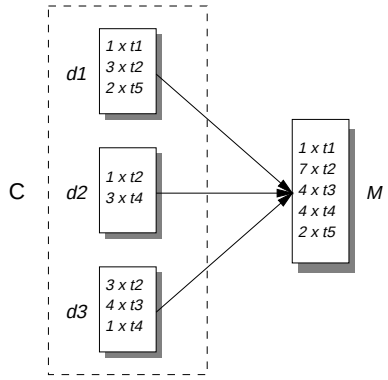

Figure 1: Documents merged to a megadocument

2 The Megadocument Approach

The megadocument approach is introduced in [7]. The basic idea is to merge all positive examples of a category into a so-called *megadocument* which represents the category. Figure 1 illustrates the approach. Web documents can be merged so that the features of a term in the document do not get lost. For example, a feature of a term t in a web document might be that this term appears in the title of this document. All terms in the titles of the web documents building the megadocument form the title of the megadocument. If t belongs to such a web document, it would appear in the title of the megadocument and would not lose its feature of appearing in the title. By this means, the probabilistic, description-oriented approach can be used for indexing megadocuments ([4]).

In [7], $tf \times idf$ indexing is used for the experiments. If we have a collection of indexed megadocuments, the new document can be regarded as a query to this collection (as mentioned above). Thus, methods used in information retrieval like the vector space model can be used for the classification task. We thus achieve a ranking of megadocuments with respect to the certain query document. For instance, we could choose the categories represented by the top ranked megadocuments to be the categories the query document belongs to.

3 Using the Hierarchy

We will now present how an underlying hierarchy in a category scheme can be used for the categorization task. The hierarchy can be seen as a directed, acyclic graph as shown in figure 2.

The prerequisite for the approach introduced below is a non-hierarchical classifier which is able to assign weights to every category in the category scheme. The megadocument approach is an example of such a classifier. There, the calculated weights of every category are used for the classification decision. Another example is the probabilistic, description oriented approach in [6] where the classification is performed on the probability $P(d \rightarrow C)$ that a document

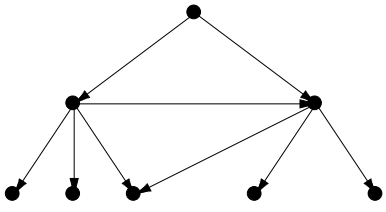

Figure 2: A hierarchy graph

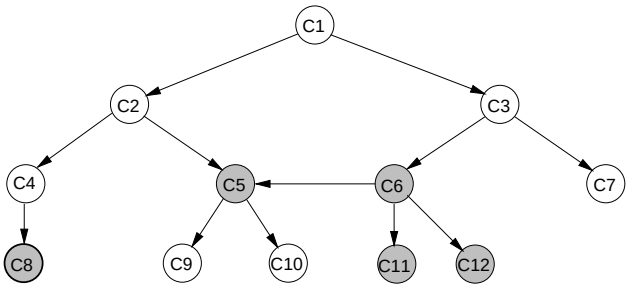

Figure 3: Ranking example

d implies a category C. So if we are talking about "non-hierarchical classifiers", we mean classifiers like these.

3.1 Motivation and Basic Idea

The motivation for using the hierarchy is reflected in figure 3. Let us assume that the non-hierarchical classifier has chosen C_8 to be the category that should be assigned to the query document d (C_8 is the top ranked category). And let us further assume that the categories shaded grey are weighted only slightly less than C_8. One can argue that a category in the area around C_6 would be more appropriate for d, because the non-hierarchical classifier has assigned relatively high weights to the categories near C_6. In contrast, the categories near C_8 in the hierarchy graph have lesser weights. We now say that if for a category C there are categories with high weights around it, this is a positive proposition for d belonging to C. Vice versa if there are categories around C with low weights. The basic idea is that for estimating the final (i.e. considering the hierarchy) weight of a category C, all other categories C' in the category set $\mathcal{C}$ contribute their weights to the one for C, according to their proximity to C. For instance, if we calculate the final weight for C_6, the non-hierarchical weight (calculated by the non-hierarchical classifier) of C_{11} should be considered more than the non-hierarchical weight of C_4.

3.2 Exploiting the Hierarchy

To use the hierarchy, we estimate the probability $P(C \to C')$ that a category C implies another category C'. In other words, if there is a document in category C, to which extent could this document be assigned to C' as well? This value might be provided externally, when the category scheme is defined. Another way, based on documents already in the collection, is to present the documents of a category C to a user who can then determine the probability in the following way:

For every document d, decide whether this document might belong to the category C' as well. So we see a category as a set of documents and the user now creates a subset containing the documents in C that might also be assigned to C'. Using the non-classical logic described in [13], we now get

$$P(C \to C') = P(C'|C) = \frac{|C \cap C'|}{|C|}.$$

Unfortunately, in most collections it won't be realistic to perform this estimation for every category pair. In *Yahoo! Computers & Internet* for example, we would have more than 7.8 million category pairs. So another way to approximate $P(C \to C')$ has to be found. To this purpose, we will later define the notions of local and global implication probabilities. The local implication probabilities are used for computing the global implication probabilities which can be used as an approximation for $P(C \to C')$.

If we have an estimation of $P(C \to C')$, and a non-hierarchical classifier yields $P(d \to C)$ as a weight for a category C in the category set $\mathcal{C}$ with respect to a query document d, then $P_H(d \to C)$, the weight considering the

hierarchy, is defined as

$$P_H(d \to C) = \sum_{C' \in \mathcal{C}} P(d \to C') \cdot P(C') \cdot P(C' \to C) \tag{1}$$

where $P(C')$ is a normalization factor and can be set to $1/|\mathcal{C}|$.

3.3 Local Implication Probability

The *local implication probability* $P(C \dot{\to} C')$ is the probability that a category C implies another category C' whereat these categories must be neighbours. Two categories are neighbours if there is an edge between them in the hierarchy graph (e.g. the categories C_6 and C_{11} in figure 3 are neighbours, whereas C_{11} and C_{12} are not). So this probability is not defined for every category pair, but only for categories together with their direct super- and subcategories in the hierarchy graph. Depending on the number of categories in the category scheme and their relations, it might be possible that users estimate every $P(C \dot{\to} C')$ manually. Another way would be to use global values α and β for all $P(C_U \dot{\to} C_O)$ and $P(C_O \dot{\to} C_U)$ with C_O as a direct supercategory of C_U. To match the hierarchical character of a category scheme so that C_O is a generalization of C_U, $P(C_O \dot{\to} C_U) < P(C_U \dot{\to} C_O)$ should be proposed. Because of the generalization, it is reasonable to assume that the probability that a document belonging to C_U might also belong to C_O is higher than the probability the other way round. For example, in *Yahoo! Computers & Internet* exists the category `Linux` with the direct subcategory `Linux Kernel Source`. A document about Linux kernel source is a document about Linux (but not necessary a document about Linux in general), whereas a document about Linux in general does not have to be a document about Linux kernel source. Thus, it would be less dramatic if a document about Linux kernel source is found in the category about Linux, whereas a document about Linux in general shouldn't be categorized in a category about Linux kernel source (this is our point of view). Now, one might efficiently estimate two global constants α and β with $\alpha < \beta$ and $P(C_O \dot{\to} C_U) \approx \alpha$ and $P(C_U \dot{\to} C_O) \approx \beta$. Furthermore, the fan-out and fan-in of a category node in the hierarchy graph could be considered. $P(C_O \to C_U)$ might decrease with the number of direct subcategories of C_O, as well as $P(C_U \to C_O)$ with the number of direct supercategories of C_U.

Motivated by the megadocument approach, we can regard a category as a set of terms. Thus, the local implication probabilities can be calculated on a term basis as ([13])

$$P(C_O \dot{\to} C_U) = \frac{|C_O \cap C_U|}{|C_O|} \tag{2}$$

and

$$P(C_U \dot{\to} C_O) = \frac{|C_U \cap C_O|}{|C_U|}. \tag{3}$$

The problem here is that $P(C_O \dot{\to} C_U) < P(C_U \dot{\to} C_O)$ is violated if $|C_U| > |C_O|$.

Using the hierarchy graph and the local implication probabilities, we can now define the *probabilistic hierarchy graph*. Let $\mathcal{C}$ be the set of categories. $G = (\mathcal{C}, E)$ is the hierarchy graph with $E \subset \mathcal{C} \times \mathcal{C}$. Now the probabilistic hierarchy graph $H = (\mathcal{C}, E')$ with $E' \subset \mathcal{C} \times \mathcal{C}$ can be developed like this:

- If $(C, C') \in E$, then $(C, C') \in E'$ and $(C', C) \in E'$.
- If $w : E' \longrightarrow [0, 1]$ is a function assigning the edge value to every edge, then $w(C, C') = P(C \dot{\to} C')$ and $w(C', C) = P(C' \dot{\to} C)$.

Figure 4 shows an example of a probabilistic hierarchy graph.

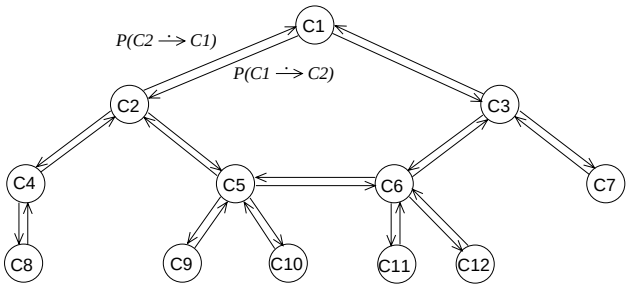

Figure 4: Exemplary probabilistic hierarchy graph

3.4 Global Implication Probability

The *global implication probability* $P(C \overset{*}{\rightarrow} C')$ is calculated from the local implication probabilities and the probabilistic hierarchy graph. We use this as an approximation for $P(C \rightarrow C')$, so

$$P(C \rightarrow C') \approx P(C \overset{*}{\rightarrow} C').$$

For the computation of $P(C \overset{*}{\rightarrow} C')$, we regard each local implication as a single event and assume their independence. This way, we can use probabilistic Datalog (*pDatalog*$_I$) for the calculation. Datalog is a function-free Horn clause predicate logic. *pDatalog*$_I$ enhances Datalog with probability theory, assuming the independence of events (see ([5] for further details on *pDatalog*$_I$). With *HySpirit*[4], there exists an implementation of *pDatalog*$_I$. A program that calculates $P(C \overset{*}{\rightarrow} C')$ written in *pDatalog*$_I$ is shown in figure 5. Here, the probabilistic hierarchy graph from figure 4 is implemented with $P(C_O \rightarrow C_U) = 0.3$ and $P(C_U \rightarrow C_O) = 0.5$. The ground fact `0.3 localimply_o(c1,c2).` says that C_1 is supercategory of C_2 and $P(C_1 \rightarrow C_2)$ is 0.3. `0.5 localimply_u(c2,c1).` says that C_2 is subcategory of C_1 and $P(C_2 \rightarrow C_1)$ is 0.5. These two kinds of ground facts code the local implication probabilities. The `globalimply` rule describes how $P(C \overset{*}{\rightarrow} C')$ is calculated. C implies C' globally if C implies C' locally or C implies a category C^* globally and C^* implies C' locally. The input `?- globalimply(c1,c6)` would yield $P(C_1 \overset{*}{\rightarrow} C_6)$ = 0.13095. This value is computed by *pDatalog*$_I$ in the following way ([5]):

[4] `http://www.hyspirit.com/`

```
0.3 localimply_o(c1,c2).          0.5 localimply_u(c2,c1).
0.3 localimply_o(c2,c5).          0.5 localimply_u(c5,c2).
0.3 localimply_o(c6,c5).          0.5 localimply_u(c5,c6).
0.3 localimply_o(c1,c3).          0.5 localimply_u(c3,c1).
0.3 localimply_o(c3,c6).          0.5 localimply_u(c6,c3).
...

globalimply(C1,C2) :- localimply_o(C1,C2).
globalimply(C1,C2) :- localimply_u(C1,C2).
globalimply(C1,C2) :- globalimply(C1,C) & localimply_o(C,C2).
globalimply(C1,C2) :- globalimply(C1,C) & localimply_u(C,C2).
```

Figure 5: *pDatalog*$_I$ program for calculating $P(C \overset{*}{\rightarrow} C')$

All loopless paths between C_1 and C_6 in the probabilistic hierarchy graph are considered. These are: $C_1 - C_3 - C_6$ and $C_1 - C_2 - C_5 - C_6$. As said before, every local implication is seen as a single event with a specific probability. Let the expression lio(c,c′) describe the event "c implies c′ locally and is the direct subcategory of c′" (`localimply_o(c,c').` would be the according ground fact). Analogical liu(c,c′) (and `localimply_u(c,c').` as the according ground fact). With K_1 = lio(c1,c2) ∧ lio(c2,c5) ∧ liu(c5,c6) and K_2 = lio(c1,c3) ∧ lio(c3,c6), $K_1 \vee K_2$ is a so-called *event expression* in disjunctive normal form. Using the sieve formula, *pDatalog*$_I$ now calculates $P(C_1 \stackrel{*}{\rightarrow} C_6)$ as

$$\begin{aligned} P(C_1 \stackrel{*}{\rightarrow} C_6) &= P(K_1 \vee K_2) \\ &= P(K_1) + P(K_2) - P(K_1 \wedge K_2) \\ &= P(\text{lio(c1,c2)}) \cdot P(\text{lio(c2,c5)}) \cdot P(\text{lio(c5,c6)}) + P(\text{lio(c1,c3)}) \cdot P(\text{lio(c3,c6)}) - \\ &\quad P(\text{lio(c1,c2)}) \cdot P(\text{lio(c2,c5)}) \cdot P(\text{lio(c5,c6)}) \cdot P(\text{lio(c1,c3)}) \cdot P(\text{lio(c3,c6)}) \\ &= 0.3 \cdot 0.3 \cdot 0.5 + 0.3 \cdot 0.3 - 0.3 \cdot 0.3 \cdot 0.5 \cdot 0.3 \cdot 0.3 = 0.13095 \end{aligned}$$

More generally, given an event expression $K_1 \vee \ldots \vee K_n$, the sieve formula computes

$$P(K_1 \vee \ldots \vee K_n) = \sum_{i=1}^{n} (-1)^{i-1} \left(\sum_{\substack{1 \le j_1 < \ldots \\ < j_i \le n}} P(K_{j_1} \wedge \ldots \wedge K_{j_i}) \right). \tag{4}$$

Note that *pDatalog*$_I$ eliminates all non-loopless paths between C_1 and C_6 by applying the absorbation law. For example, the path $C_1 - C_3 - C_7 - C_3 - C_6$ would result in K_3 = lio(c1,c3) ∧ lio(c3,c7) ∧ lio(c7,c3) ∧ lio(c3,c6). $K_2 \vee K_3$ is the same as K_2, so K_3 will be removed by *pDatalog*$_I$.

3.5 Alternative Computation of $P(C \stackrel{*}{\rightarrow} C')$

Since the computation of $P(C \stackrel{*}{\rightarrow} C')$ may be very expendable and one might want to be independent of HySpirit or other *pDatalog*$_I$ implementations, other ways of calculating $P(C \stackrel{*}{\rightarrow} C')$ will be shown. If we have a probabilistic hierarchy graph H, we can define the *path implication probability* $P(C \stackrel{PF}{\longrightarrow} C')$ as the probability that C implies C' regarding only the acyclic path PF between C and C'. Let us take the example shown in figure 4. Here, we have the path $PF = C_1 - C_2 - C_5 - C_6$ as one possible path between C_1 and C_6. If we have $P(C_1 \rightarrow C_2)$ = 0.3, $P(C_2 \rightarrow C_5)$ = 0.3 and $P(C_5 \rightarrow C_6)$ = 0.5, $P(C_1 \stackrel{PF}{\longrightarrow} C_6)$ would be $0.3 \cdot 0.3 \cdot 0.5 = 0.045$. Let $E_{(i,j)}$ denote the event that C_i implies C_j locally. Λ is a function that has an edge sequence as input and calculates the probability that all events represented by the edges in the sequence occur. For instance, if $S = ((C_i, C_j), (C_k, C_l), (C_m, C_n))$ then $\Lambda(S) = P(E_{(i,j)} \wedge E_{(k,l)} \wedge E_{(m,n)})$.

With these definitions, we can now present an algorithm that calculates $P(C \stackrel{*}{\rightarrow} C')$ according to the sieve formula:

Input:

- Category nodes C and C'.
- Matrix H as $n \times n$ array representing the probabilistic hierarchy graph. n is the number of nodes in H. Elements are the local implication probabilities (used by Λ).
- List $P = (PF_1, \ldots, PF_k)$ of cycle-free paths between C and C'.

Output: $P(C \stackrel{*}{\rightarrow} C')$

1: $p = 0$
2: **for** $i = 1$ to k **do**

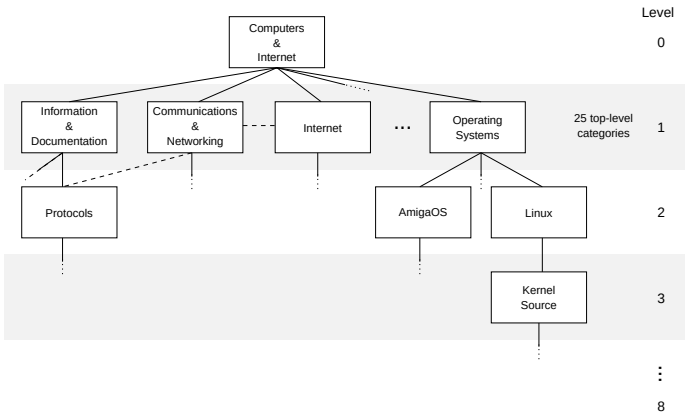

Figure 6: ***Yahoo! Computers & Internet***

3: $s = 0$
4: **for all** i-tuple $(PF_{j_1}, \ldots, PF_{j_i})$ of loopless paths from P with $1 \leq j_1 < \ldots < j_i \leq k$ **do**
5: calculate edge sequence $S_{PF_{j_1}}, \ldots, S_{PF_{j_i}}$ so that there is no duplicate edge.
6: $s = s + \left(\Lambda(S_{PF_{j_1}}) \cdot \ldots \cdot \Lambda(S_{PF_{j_i}})\right)$
7: **end for**
8: $p = p + (-1)^{i-1} \cdot s$
9: **end for**

The sequence $S_{PF_{j_1}}, \ldots S_{PF_{j_i}}$ in line 5 is built to eliminate duplicate edges in the according path list $(PF_{j_1}, \ldots, PF_{j_i})$ which would be duplicate events. For example, if we have two paths with the edges (C_1, C_2) and (C_2, C_3) in path 1, and (C_4, C_2) and (C_2, C_3) in path 2, one possible edge sequence would be $(C_1, C_2), (C_2, C_3), (C_4, C_2)$, eliminating one occurrence of (C_2, C_3).

For efficiency reasons (the sieve formula has exponential complexity), the number of paths might have to be limited. One way to do this in a reasonable manner is to sort the loopless paths between two categories by decreasing path implication probabilities and choose the k first paths from this sorted list, neglecting the others. A k-shortest-path algorithm as described in [10] can be adapted to this problem and used for the calculation ([4]).

4 Experiments

4.1 Test Collection

Some experiments have been performed to evaluate our hierarchical classifier. To evaluate the effectiveness of a categorization method, a test collection is needed which is split into two parts, the training documents (in our case the documents building the megadocuments), and the test documents. Our test collection was *Yahoo! Computers & Internet*, a subset of *Yahoo!*. This is a hierarchical collection; an excerpt can be seen in figure 6. Our frozen mirror of *Yahoo! Computers & Internet* consists of 2806 categories on 9 levels with 18639 documents. Besides having direct sub- and supercategories, a category might be linked to other categories as well. In the figure, the dashed lines between categories represent such links. There are 893 such links in the test collection, but for efficiency reasons we don't consider them. Thus, the resulting hierarchy graph is a tree, so there is only one acyclic path between two category nodes in the probabilistic hierarchy graph.

On level 1 of the *Yahoo! Computers & Internet* category scheme, there are 25 top-level categories (the *top25 categories*, see figure 6.). These are regarded in particular. C is top25 category of C' if C is in level 1 of *Yahoo! Computers & Internet* (according to the levels seen in figure 6) and C' is a subcategory of C, or if C' itself is the top25 category, i.e. $C' = C$. For instance, `Operating Systems` is the top25 category of `Linux`, `AmigaOS`, `Kernel`

`Source`, and of all others of its subcategories.

4.2 Evaluation Techniques

In this section, the techniques used for evaluating our approach are described. These are the same methods as used in [6].

4.2.1 Top Ranked and Average Precision

For the evaluation, the test collection is split into two parts, the set of training documents (the documents building the megadocuments), and the set T of test documents. For each test document d, we get a ranking of the categories with respect to d. The *top ranked categories* are the categories in the first rank of the ranking; our classifier decides to assign them to d. We calculate the *top ranked precision* TR which is defined as the fraction of the total number of correct decisions to the total number of decisions taken:

$$TR = \frac{\text{\#correct decisions}}{\text{\#decisions}} \tag{5}$$

We want to see how the approach performs with respect to lower ranks. To do this, we merge all rankings to one global ranking and mark all categories as relevant that are the correct categories with respect to the appropriate test document. So there are multiple entries of categories in the global ranking, whereat one category might be marked relevant in one case (because it was the relevant category to a specific test document) and not marked relevant in another case. For example, if we have three test documents d_1, d_2 and d_3, and four categories $C_1, \ldots, C_4$, we might get the rankings $R_1 = (C_1, C_4, \underline{C_3}, C_2)$, $R_2 = (C_3, \underline{C_2}, C_4, C_1)$ and $R_3 = (\underline{C_3}, C_2, C_4, C_1)$. A category is relevant to a test document if this document belongs to the category. Relevant categories are underlined. The classifier has assigned weights to every category in a ranking. So by merging the three rankings, we would get one global ranking determined by the weights of the categories. In this new global ranking, a category is seen as a relevant category if it was relevant in the ranking it comes from. In our example, one possible ranking derived from R_1, R_2 and R_3 could be $R = (C_1, C_3, \underline{C_2}, C_4, C_4, \underline{C_3}, C_2, C_4, C_1, \underline{C_3}, C_2, C_1)$. As said before, there are multiple occurrences of a category in the ranking. In R, the first C_1 might be taken from R_1, the second C_1 from R_2 etc. All (occurrences of) C_1 would be considered as different categories in R. Once we have this global ranking, we can use well-known evaluation techniques from information retrieval. Thus, if one chooses the first n categories[5] from a ranking, *precision* is the number of relevant categories chosen divided by n. *Recall* is the fraction of the relevant categories found to all relevant categories in the ranking. If we assume a strong order in our example ranking R (i.e. no category has the same weight as another one) and choose the first four categories, we would get a precision of 0.25 (one relevant category within four categories chosen) and a recall of $1/3$ (one relevant category found in the four categories, three relevant categories total in the ranking). One can determine a fixed number of recall points and calculate the precision for every recall point. In our experiment, the precision values for the 100 recall points $1/100, 2/100, \ldots, 1$ were computed. Using these precision values, an *average precision* can be calculated.

4.2.2 Top25 Matches

In [6] and [7], not only the exact matching of a category (i.e. if exactly the same category is assigned to the document as in the test collection) is regarded, but it is also recognized if a test document has matched the appropriate top25 category. So besides the selection of the exact category for a test document, another strategy is to select a top25 category only. This has been done in two ways:

1. Choose the top25 categories of the top ranked categories. This means that every category is relevant to the test document that belongs to the same top25 category as the test document. This way, our ranking is the same, but we have more categories marked relevant with respect to the test document.

[5] Usually, information retrieval deals with rankings of documents. We take the unusual path of dealing with rankings of categories and applying IR evaluation methods to them.

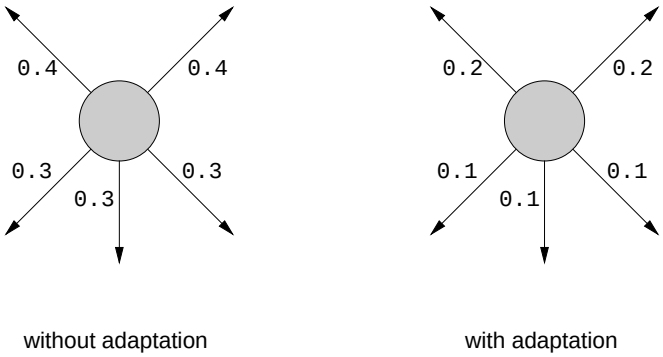

Figure 7: Considering the different kinds of fan-out with adaptation

2. Determine a new ranking of top25 categories using a kNN-like method. Given a ranking R of categories w.r.t. a test document d, we consider the k top ranked categories in R, which build the set NN. With this set, we compute weights for each top25 category:

$$w_{d,C^{25}}^{25} = \sum_{C \in NN} \frac{w_{d,C} \cdot P(C \to C^{25})}{|NN|} \tag{6}$$

with $w_{d,C}$ as the weight of a category C w.r.t. a document d (calculated by the hierarchical classifier), and $P(C \to C^{25})$ = 1 if C^{25} is top25 category of C, and 0 else. Using the calculated weights w^{25} (which are defined for top25 categories only), the top25 categories are ranked according to decreasing weight w^{25}.

The top ranked and average precision are retrieved for the exact match and for the two possible top25 matches.

4.3 Description of the Experiments

Eight experiments have been performed. The ninth experiment, the baseline, was performed by C.P.-Klas in [7] using the megadocument approach with $tf \times idf$ indexing. We chose the experiment in which the top 10 terms according to their idf values were regarded for each test document, since this experiment was the most effective one of those described in [7]. The results of this experiment are the term weights calculated by the non-hierarchical classifier and are used as an input for the hierarchical classifier according to (1)[6].

Our experiments can be sorted into two classes: those with an intellectual (i.e. manual) estimation of $P(C \to C')$, and those where the local implication probabilities are computed. Each experiment was performed using and not using *adaptation*. Adaptation means that the fan-out of each node in the probabilistic hierarchy graph is considered. There are two different kinds of fan-out: the number of direct subcategories and the number of direct supercategories (which is 1 in our case). Figure 7 shows the effect of the adaptation. The edge values of the probabilistic hierarchy graph are divided by the number of subcategories or supercategories, respectively. In this example, the category has two supercategories and three subcategories. So the values of the edges from the category to its supercategories are divided by 2 (from 0.4 to 0.2), and the values of the edges to its subcategories are divided by 3 (from 0.3 to 0.1). Thus, the local implication probabilities decrease if the number of direct supercategories (or subcategories, respectively) increases.

Since there wasn't any hint in the category scheme of *Yahoo! Computers & Internet* for the estimation of $P(C \to C')$, we used intellectual estimation to see how our approach performs with different edge weights in the probabilistic hierarchy graph. The following estimations were made (with the experiment name in parentheses and C_O as direct supercategory of C_U):

[6]The weights calculated by the megadocument approach can not be interpreted as $P(d \to C)$. Nevertheless, we take the calculated weights instead of this probability.

- $P(C_O \dot{\rightarrow} C_U)$ = 0.1 and $P(C_U \dot{\rightarrow} C_O)$ = 0.2 (`0102`). Here we test what happens if there is a low influence of the hierarchy.
- $P(C_O \dot{\rightarrow} C_U)$ = 0.2 and $P(C_U \dot{\rightarrow} C_O)$ = 0.8 (`0208`). The local implication from a supercategory to a subcategory is low, but high in the other direction.
- $P(C_O \dot{\rightarrow} C_U)$ = 0.7 and $P(C_U \dot{\rightarrow} C_O)$ = 0.8 (`0708`). High influence of the hierarchy.

To see what happens if we try to calculate the local implication probabilities using the data material in the collection, we performed another experiment, using the megadocuments formed for the experiments in [7] before. $P(C_O \dot{\rightarrow} C_U)$ was calculated using equation (2) and for the calculation of $P(C_U \dot{\rightarrow} C_O)$ equation (3) was used. But we did not consider all terms in a megadocument, only the top 50 terms by their idf value. Therefore the name of the experiment is `IDF50`. In most cases, the local implication probabilities have been estimated as 0, because the intersection of the according megadocuments was empty. The advantage of this method is that these estimations are less rigid than the ones achieved by the intellectual estimations described above.

Every experiment was repeated using adaptation; this is indicated by the postfix `adapt` in the name of the experiment. The computations of the global implication probabilities were done in a pre-processing step. For the calculation of a global implication probability to a category C, we only considered those categories $C' \in \mathcal{C}$ with $P(C' \overset{*}{\rightarrow} C) > 0.01$ (this was done for efficiency reasons).

The name of the baseline experiment is `OH`.

4.4 Results

Finally, we present the results of the experiments. TR indicates the top ranked precision, AV the average precision. In the first column is the name of the experiment. In columns 2 and 3 are the results with respect to the exact match, in the next two columns are the results of the experiments regarding top25 categories and the last two columns show the results of the kNN selection of top25 categories. Results that were better than the appropriate baseline results are highlighted.

Experiment	TR	AV	T25 TR	T25 AV	T25k TR	T25k AV
`OH`	14,45%	18,13%	52,74%	18,93%	54,12%	62,73%
`IDF50`	**14,47%**	**18,2%**	52,12%	**18,97%**	**54,36%**	**62,92%**
`IDF50 adapt`	**14,54%**	**18,24%**	52,14%	**18,95%**	**54,34%**	**62,9%**
`0102`	13,58%	17,41%	51,83%	**19,04%**	**55,03%**	**63,19%**
`0102 adapt`	14,45%	18,13%	52,74%	18,93%	54,12%	62,72%
`0208`	8,23%	11,31%	39,3%	18,54%	53,15%	60,34%
`0208 adapt`	9,74%	13,58%	49,51%	18,68%	53,21%	61,4%
`0708`	6,75%	9,45%	34,22%	18,46%	49,39%	57,22%
`0708 adapt`	9,71%	13,57%	48,54%	18,79%	53,23%	61,22%

Table 1: Results of the experiments

We can see that the usage of hierarchy information can improve precision values. For the exact match, the IDF50 experiments achieve better results than the baseline. Here, adaptation had a positive effect in every experiment. All the other experiments had a more or less detrimental effect with respect to the baseline. For the top25 experiments, `0102` performed best, followed by the IDF50 experiments. It is interesting that for both of these experiments, adaptation often had a negative effect on precision (in contrast to the exact match). We achieve better results according to the top25 categories if we choose the top25 categories using the kNN selection.

5 Conclusions

The results of the experiments show that using hierarchy information for the categorization task can improve classification quality, although the improvement observed in the experiments is very small. The results of the IDF50 experiment

show that the idea of calculating the local implication probabilities is a way to consider the local conditions between two neighbouring categories. This way of estimating $P(C \dot{\rightarrow} C')$ is more flexible than the presented intellectual estimation, although the `0102` experiment yielded the best results for top25 categories. Since in the IDF50 experiments $P(C \dot{\rightarrow} C')$ was 0 in most cases, other ways (if there) of using the knowledge provided by the category scheme and the previously assigned documents should be considered in order to achieve better estimations of $P(C \dot{\rightarrow} C')$.

The results also show that the kNN selection is a good way to choose a top25 category for a new document to be inserted into the collection.

6 Acknowledgments

Barbara Lutes of GMD was so kind to proofread this paper and make useful annotations. So did the three anonymous reviewers who also helped improving the quality of the paper.

References

[1] Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of the acm.*, 35(12):29, 1992.

[2] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *International Journal on Very Large Data Bases, 7(3) pp163-178*, 7(3):163–178, 1998.

[3] 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.

[4] Ingo Frommholz. Automatische Kategorisierung von Web-Dokumenten. Diploma thesis, Universität Dortmund, Fachbereich Informatik, 2001. In German.

[5] Norbert Fuhr. Probabilistic datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.

[6] Norbert Gövert, Mounia Lalmas, and Norbert Fuhr. A probabilistic description-oriented approach for categorising web documents. In Susan Gauch and Il-Yeol Soong, editors, *Proceedings of the Eighth International Conference on Information and Knowledge Management*, pages 475–482, New York, 1999.

[7] Claus-Peter Klas and Norbert Fuhr. A new effective approach for categorizing web documents. In *Proceedings of the 22th BCS-IRSG Colloquium on IR Research*, 2000.

[8] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97)*, pages 170–178, 1997.

[9] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)*, pages 359–367, 1998.

[10] Marta Pascoal, José Luís Santos, and Ernesto Quierós Vieira Martins. An algorithm for ranking loopless paths. Report, 1999. `http://www.mat.uc.pt/~eqvm/cientificos/investigacao/Artigos/MPS_novo.ps.gz`.

[11] M. Ruiz and P. Srinivasan. Hierarchical neural networks for text categorization. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '99)*, pages 229–237, New York, 1999. ACM.

[12] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[13] C. J. van Rijsbergen. A non-classical logic for information retrieval. *The Computer Journal*, 29(6):481–485, 1986.

[14] S.K.M. Wong and Y.Y. Yao. On modeling information retrieval with probabilistic inference. *ACM Transactions on Information Systems*, 13(1):38–68, 1995.

[15] Yiming Yang. Expert network: Effective and efficent learning from human decisions in text categorisation and retrieval. In W. Bruce Croft and C. J. van Rijsbergen, editors, *Proceedings of the Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 13–22, London, et al., 1994. Springer-Verlag.

[16] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In SIGIR, editor, *SIGIR '99, Proceedings of the 22nd International Conference on Research and Development in Information Retrieval*, pages 42–49, New York, 1999. ACM.