Cluster Labeling for the Blogosphere

Patrick Hennig
Hasso-Plattner-Institut
University of Potsdam, Germany
patrick.hennig@hpi.uni-potsdam.de

Philipp Berger
Hasso-Plattner-Institut
University of Potsdam, Germany
philipp.berger@hpi.uni-potsdam.de

Christoph Meinel
Hasso-Plattner-Institut
University of Potsdam, Germany
christoph.meinel@hpi.uni-potsdam.de

Claus Steuer
Hasso-Plattner-Institut
University of Potsdam, Germany
claus.steuer@hpi.uni-potsdam.de

Christian Wuerz
Hasso-Plattner-Institut
University of Potsdam, Germany
christian.wuerz@hpi.uni-potsdam.de

***Abstract*—Hierarchical Cluster Labeling helps users to quickly understand and analyze hierarchical clusters. This may be used to enhance search engine results or interactive browsing like it is being used in the BlogIntelligence application. The hierarchical organization of data helps to represent different levels of detail. Hierarchical clustering may be quite common, but there are few good solutions for labeling those clusters. We decided to lay the focus of this work on labeling binary hierarchical clusters. Current approaches focus either on statistical features of the clustered documents or external sources like Wikipedia. We combined those ideas to profit from both advantages and created an algorithm, that can handle clustered documents as well as terms.**

I. Introduction

Since every day millions of posts are being published the huge collection of web documents inside the blogosphere is getting bigger and bigger . Clustering this ever-changing collection is a very time consuming task. BlogIntelligence [1] is providing a smart search engine for the blogosphere, including harvesting, analysis and presenting the results in a very meaningful way.

In our daily life we often have to deal with unordered collections of documents. These can be articles from different streams like news agencies, social networks and blogs or documents returned by search queries. With growing size it becomes increasingly difficult to get an overview of the covered topics and find documents of interest. One solution to this problem is hierarchical clustering. Hierarchical clustering is a well known task in information retrieval where a considerable amount of research has been done. However when users have to interact with the generated hierarchy it is important to label the clusters, so that the user knows what topic is covered by a cluster. Comparatively less research has been done on automatic labeling of cluster hierarchies. Next to document cluster hierarchies we consider another cluster type where we have a set of terms per cluster node instead of documents.

A proper label for a single cluster node in the hierarchy should fulfill the following criteria:

- It should be general enough to describe all documents or terms in the cluster node (in a document cluster hierarchy a non leaf cluster node contains all the documents of its children).
- It should be specific enough to differ from its child and sibling cluster nodes.
- It should be a generalization of its child cluster labels and accordingly a specialization of its parent cluster label.

This paper proposes two algorithms that automatically find appropriate labels for binary hierarchical clusters, where each cluster node can have a maximum of two children. The first algorithm relies solely on the statistical features of the terms inside the documents of the cluster hierarchy. To label a cluster node it takes into account statistical features of the terms inside the cluster node itself, in its child cluster nodes and its sibling cluster nodes. The second algorithm uses the DBpedia category network[13] to find appropriate labels. While the first algorithm can only be applied to document clusters the second algorithm also works for term clusters. Indeed the second algorithm does not use the documents at all but only a collection of relevant terms for each cluster node which can be generated using the first algorithm.

Throughout the paper we focus on cluster hierarchies that resemble binary trees. Although both algorithms can operate on any tree hierarchy the restriction to binary trees has some implications for the algorithms. Most notably the resulting clusters contain fewer documents compared to non binary hierarchies. We assume that this is the reason why most algorithms proposed by other researchers, that we implemented, achieved poor results on our test cluster hierarchies, although they performed well on the significant bigger cluster hierarchies on which they were originally evaluated.

The paper is structured as follows. Section II discusses related work on cluster labeling. In Section III both algorithms are presented. A description of the test framework and the result of our evaluation is given in Section IV. Finally our findings and possible future work is summarized in Section V and VI.

A. BlogIntelligence

With a wide circulation of more than 200 million *weblogs* worldwide, *weblogs* with good reason, are one of the most

[1] http://www.blog-intelligence.com

important data streams in the World Wide Web. Therefore, weblogs offer access to latest information discussed in the real world. Since writing posts in weblogs goes along with a high editorial effort, the available information is of major interest. However, for a user it is becoming harder and harder to gain an overview of all discussions in the blogosphere. Hence, a system that collects information from the blogosphere and presents it to the user in a very meaningful way would be of great use.

Therefore, mining, analyzing, modeling and presenting this enormous amount of data is the overall aim of the project the presented work is integrated in. This enables the user to detect technical trends, political climates or news articles about a specific topic. Most approaches to mining and analyzing such a huge amount of data focus on offline algorithms which use pre-aggregated results. This is in contrast to the continuously growing nature of the World Wide Web. As a result, including the latest data is one of the key aspects of data mining on the web. This is exactly the topic covered by the *BlogIntelligence* project.

The presented work in this paper is integrated into the *BlogIntelligence* project. There are three main steps involved to visualize blogs in the BlogIntelligence project:

1) Extraction: In the extraction step the blogs are basically crawled. In order to achieve this, a purpose-built crawler needs to be used as traditional crawlers do not fully meet the particularities of blogs as opposed to conventional websites.

2) Analysis: The analysis step prepares the crawled data for visualization. Each blog is analyzed by multiple *Analyzers*, that process its details in certain ways. Among potentially others, there are *data analyzers* that store the meta information about the blogs into the database, *content analyzers* that store information about the content which allow content-related analyses and there are *network analyzers* that store information on the relationships and links between blogs or other communities.

3) Visualization: The last step within the BlogIntelligence framework is the visualization of the analyzed information. The Blog IntelliTrends solution is part of this last step as it provides the stored data via an interface and visualizes them in client applications.

II. Related Work

There are two major groups of works considering cluster labeling.

The first one uses the statistical features of the clusters to extract label candidates [23, 24, 6]. These labels are extracted directly from the text of the documents. Therefore the documents often get summarized by the most relevant words [18, 23, 15]. A naive approach is to use the most frequent word as a label for the cluster [2]. The resulting labels are usually too general and not a good representative for the cluster. Another relevant area [23, 6] also considers the *negative frequency* of the terms, the frequency outside of the cluster. But there are just a few approaches considering the labeling of hierarchical cluster hierarchies [23] instead of flat clusters [9].

The second group of works tries to find an appropriate label using external sources like Wikipedia or other Ontology Databases [21, 1, 8, 20, 14, 12, 3]. Usually these approaches are preprocessing the Wikipedia dump [22, 20] or they are using a structured version like DBpedia [13, 16]. Magatti et al. [14] even rely on the Open Directory Project[2] [4]. All are based on the idea that an appropriate label does not have to appear in the cluster itself. Carmen et al. [1] try to map the considered documents to Wikipedia articles and label the clusters with the Wikipedia Category titles of these articles. In addition of using Wikipedia, Lau et al. [12] also queries the Google web search engine to get label candidates. Coursey et al. [3] are building an *encyclopedic graph* that represents the Wikipedia articles and categories as nodes.

The first algorithm of our paper is based on the Descriptive Score (*DScore*) algorithm from Treeatpituk and Callan [23]. They calculate multiple statistical features for each label candidate. To learn the significance of each feature the algorithm is trained with manually labeled clusters. Their approach also considers bi-grams and tri-grams as label candidates. The results of the DScore algorithm for our binary clusters where not satisfactory, hence we adapted it to our use-case by, among others, using different statistical features.

The idea of our second algorithm is influenced by the work of Hulpus et al. [8]. They are using a graph-based approach to label clustered terms or documents. Therefore they create for a group of words: a so-called *sense graph*. It consists of DBpedia Categories, DBpedia Ontologies and the *YAGO* [7] vocabulary (Wikipedia and WordNet[3] [5]) that represent the sense of the considered words. The edges between those nodes represent the generalization between them. To extract a label candidate, they use different measures to find a central element in their graph. However, since their algorithms main objective it topic labeling, it does not consider hierarchical relation between the clusters.

Our approach differs from the mentioned work as it uses both techniques, the statistical features and the DBpedia Database. We have created two algorithms and combined them to improve the results. Therefore our approach can handle hierarchical clusters of documents and terms.

III. Algorithms

The first algorithm proposed in this paper for labeling hierarchical document clusters is derived from the Descriptive Score (DScore) algorithm developed by Treeratpituk and Callan [23]. To determine the label of a cluster node S DScore first selects a set of label candidates from all unigram, bi-gram and tri-gram phrases that occur in the documents of S. For each label candidate different statistical features are calculated with respect to the cluster node S and the parent cluster node P. These features are then used as variables for a multiple linear regression model. The result of the model for a phrase p is a real value that denotes how descriptive a phrase is for cluster S. A value of 1 means that p is a good description for S and hence a good label, whereas a value of 0 denotes a bad label. The weights of the regression model are learned using labeled hierarchical document clusters extracted from the Open Directory Project [4]. To train the model the descriptive score of each label candidate L, in each cluster extracted from ODP, must be estimated. Therefore the number of overlapping words

[2] http.dmoz.org

[3] http://wordnet.princeton.edu/

between each Synonym SL of L and the known correct label CL is calculated and normalized by $max(len(SL), len(CL))$. In our tests (Section IV-C) with binary document clusters the DScore algorithm found the correct label only in 12,62% of all cases (using only the top result and considering partial or synonym matches as correct). The big difference between our test result and the 53% measured by Treeratpituk and Callan can be explained by the much smaller number of documents per cluster node in our test clusters. Their test cluster contained 21,143 documents distributed over 165 cluster nodes with a maximum depth of 3. Our test clusters contained in average 373 documents distributed over 15 cluster nodes with an average maximum depth of 7. We will now describe how we changed the DScore algorithm in order to yield better results for our comparatively sparse binary cluster hierarchies.

A. Optimizing DScore for sparse cluster hierarchies

1) Label candidate selection:: The label candidates of a cluster node S are selected with the following steps:

1) Unigram, bi-gram and tri-gram phrases of all documents are extracted and stemmed using Krovetz's stemmer [11].
2) Stopwords are removed using a fixed list. Bi-gram and tri-gram phrases that consists of more than 50% stopwords or that end with a stopword are also removed.
3) Like in DScore the label candidates are selected based on their document frequency in S. However using a fixed threshold of 20% for unigram and 5% for bi-gram and tri-gram phrases leads to all unigram phrases being selected when the cluster node contains less than 6 documents. All bi-gram and tri-gram phrases are selected when the cluster node contains less than 21 documents which is almost always the case in our sparse clusters. Instead of using fixed values the thresholds depend on the number of documents in the cluster denoted by $|S|$:

$$\text{Unigram Threshold: } 0.1 \ + \frac{0.9}{ln(|S|)}$$
$$\text{Bi\Tri-gram Threshold: } 0.05 + \frac{0.95}{4 \cdot ln(|S|)}$$

The threshold decreases logarithmic with increasing number of documents in the cluster. In general bi-gram and tri-gram phrases occur less frequent than unigram phrases. Hence the decrease is boosted by a factor of 4 for these phrases.

2) Model Variables:: For each label candidate of S the following statistical features are calculated and used as input variables for the linear regression model:

a) Macro normalized document frequency ($MNDF_C$):: For leaf cluster nodes the macro normalized document frequency is equal to the normalized document frequency ($NDF_C/|C|$) used in DScore which is the fraction of documents in the cluster node that contain the phrase p. In non leaf cluster nodes $MNDF_C$ is the mean of the $MNDF_{C_i}$ for each child cluster C_i of C.

$$MNDF_C(p) = \begin{cases} \frac{|\{d \in C : p \in d\}|}{|C|}, children(C) = \emptyset \\ \frac{\sum_{C_i \in children(C)} MNDF_{C_i}(p)}{|children(C)|}, children(C) \neq \emptyset \end{cases}$$

Using the macro normalized document frequency instead of the normalized document frequency in the non leaf cluster nodes prevents a bias towards phrases from children with a lot of documents. Consider a cluster node C with the children C_1 containing 20 documents and C_2 containing 10 documents. A phrase p_1 that occurs in 50% of the documents in C_1 and C_2 is a better label for C than a phrase p_2 that occurs in 75% of the documents in C_1 and only in 10% of the documents in C_2. The normalized document frequency favors p_2 (0.53) over p_1 (0.5), macro normalized document frequency favors p_1 (0.5) over p_2 (0.425). The $MNDF_C$ for the self cluster node S and the sibling cluster node N of S are calculated and used as input variables. The assumption is that a good label for S occurs frequent in S and less frequent in the sibling cluster node N.

b) Normalized document frequency child variance (Var_{MNDF_C}):: A Phrase that occurs frequent in one child cluster but less frequent in the other is less likely to be a good label for S. Hence the variance of the children macro normalized document frequency is computed for each label candidate p:

$$Var_{MNDF_C}(p) = \frac{1}{|children(C)|} \cdot \sum_{C_i \in children(C)} \left[MNDF_{C_i}(p) - E\left(MNDF_{children(C)}(p)\right)\right]^2$$

For leaf cluster nodes Var_{MNDF_C} is always 0.

c) Term frequency - inverse cluster frequency ($TF - ICF$):: The relevance of a term for a document from a collection of documents is often determined using the $TF-IDF$ measure. Likewise $TF - ICF$ computes the relevance of a phrase p for cluster node S with respect to all other nodes in the hierarchy.

$$TF - ICF_S(p) = TF(p) \cdot ICF(p)$$
$$TF_S(p) = \frac{\sum_{d \in S} f(p, d)}{max\{\sum_{d \in S} f(w, d) : d \in S : w \in d\}}$$
$$ICF_S(p) = ln\left(\frac{|L| - |leafs(S)|}{\sum_{C \in (L - leafs(S))} NDF_C(p) + 1}\right)$$

The set containing all leaf cluster nodes is denoted by L. The function $leafs(C)$ returns the set of leaf cluster nodes that are descendants of C. If C itself is a leaf node $leafs(C)$ returns C. The raw frequency of a phrase p in a document d is denoted by $f(p, d)$.

Additional to the features described above the number of words in a phrase $LEN(p)$ is used as an input for the model. The following formula shows the complete linear regression model for the DScore algorithm optimized for binary cluster

hierarchies:

$$\begin{aligned} OptimizedDScore(S,p) = c_0 \\ + c_1 \cdot LEN(p) \\ + c_2 \cdot MNDF_S(p) \\ + c_3 \cdot MNDF_N(p) \\ + c_4 \cdot Var_{MNDF_S}(p) \\ + c_5 \cdot TF-ICF_S(p) \end{aligned}$$

After training the algorithm with different binary document cluster hierarchies sampled from ODP the following weights were learned:

$$\vec{\omega} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix} = \begin{pmatrix} -0.033097 \\ -0.000981 \\ +0.008843 \\ -0.000564 \\ -0.054826 \\ +0.044546 \end{pmatrix}$$

3) Labeling:: In order to label a cluster node S the label candidates L are selected as described above. For each phrase p in L the feature vector

$$\vec{x} = \big(1.0, LEN(p), MNDF_S(p), MNDF_N(p), Var_{MNDF_S}(p), TF-ICF_S(p)\big)^T$$

is calculated. The dot product of the weight vector and feature vector yields the score of p: $score_p = \vec{\omega} \cdot \vec{x}$. The label candidates are ordered descending by their score. The first n candidates are assigned as labels of S.

a) Fallback:: When no phrase in S satisfies the conditions for a label candidate all phrases are ranked in descending order by their document frequency. Phrases with the same document frequency are ordered by term frequency.

b) Collapse:: If n is greater than one it may happen that label candidates of different word length overlap with each other. E.g.: "Life on mars", "on mars", "mars". In such a case the last two labels do not add any relevant information for the user. Hence if an assigned label p_1 contains another assigned label p_2 they are collapsed into $p_3 = p_1$ the score of p_3 is $max(score_{p_1}, score_{p_2})$.

B. Using DBpedia for hierarchical term cluster labeling

A term or a phrase can consist of multiple words. We define a cluster node S from a hierarchical term cluster as a set of terms $S = \{t_1, ..., t_n\}$. The relevance of a term with respect to the cluster node S is given as a probability value by $rel_S(t) \in (0, 1)$. Two types of term cluster hierarchies are considered. In the first type only leaf nodes can contain terms, hence S is empty for all non leaf nodes. In the second type every cluster node can contain terms.
The second algorithm proposed in this paper can deal with both types. Since every cluster node consists only of a few terms, statistical analysis as in the first algorithm will not help finding a good label. Therefore we use the DBpedia [13] category network as an ontology to find proper labels for each cluster. The DBpedia project extracts structured information from Wikipedia and makes them available in different datasets. The core of DBpedia are "things" which we will refer to as DBpedia resources (DBPResource). Each DBPResource is denoted by an URI and represents a single Wikipedia article. DBPResources are assigned to Wikipedia Categories which are represented using the SKOS vocabulary [17]. Together the Categories form a directed graph. Child and parent categories can be found following the *skos:broader* and *skos:narrower* edges. Related categories are linked via *skos:related* edges.

1) Mapping DBpedia resources to cluster nodes :: The first step of the algorithm maps to each cluster node S the DBPResources R that are best described by the terms in S. Therefore every term $t \in S$ is compared against the most important terms of each DBPResource. A dataset with the most important terms for every DBPResource is provided by the DBpedia NLP Project [16]. The terms are selected from the Wikipedia paragraphs linking to the resource using the tf-idf measure. We add the terms of the article titles and apply stopword removal as well as Krovetz stemming before comparing them with the terms in S. The following function computes the probability that a DBPResource R is described by a cluster node S. The DBPResource is represented by its most important words $R = \{w_0, ..., w_1\}$:

$$\begin{aligned} match(S,R) = \frac{|S \cap R|}{|S|} \cdot \left(1.0 - \frac{|R-S|^2}{|R|^2}\right) \cdot \frac{\sum_{t \in S \cap R} rel_S(t)}{|S \cap R|} \\ = P(R|S) \end{aligned}$$

If the terms of a cluster node are equal to the important terms of a DBPResource and the relevance of all terms with respect to the cluster node is 1, than the probability that the resource is described by the cluster node is 1. The probability decreases if less terms of the cluster node are terms of the resource or if the matching terms are less important to the cluster node. Similarly, the probability decreases if less terms of the resource are terms of the cluster node. To prevent a bias towards resources with fewer terms this factor is softened by squaring the fraction of non matched terms. In our algorithm we map the 5 DBPResources with the highest match probability to the cluster node. Higher values may increase the precision of the algorithm, which has to be evaluated in future tests.

2) Mapping categories to cluster nodes:: After the DBPResources for a cluster node have been selected the categories to which the resources belong are assigned to the cluster node. Each category is scored. Categories with a higher score are more likely to be the real category of the cluster node. The score of a category c with respect to cluster node S and its DBPResources Φ is computed as follows:

$$\begin{aligned} score_{(S,\Phi)}(c) = \frac{f(c,\Phi)}{max\{f(c',\Phi) : c' \in cat(\Phi)\}} \\ \cdot max\{match(S,R) : R \in \Phi \wedge c \in cat(R)\} \end{aligned}$$

In the equation $f(c, \Phi)$ denotes the number of resources in Φ that belong to category c. The function $cat(x)$ yields the categories of x, where x can be a single resource or a set of resources. The more of the resources assigned to S belong to category c the higher is the score of c. Furthermore the score of c depends on the maximum of the match probabilities of all resources in Φ that belong to c.

3) Filtering categories using the category graph:: Now that we have a set of possible categories for each cluster node, we can use the category graph to remove categories that do

not fit into the hierarchy. We define the following notation for the category graph:

1) $c_1 \xrightarrow{broad,x} c_2$ denotes a path with length x between category c_1 and c_2 following only edges in *skos:broader* or *skos:related* direction, with at least one edge in *skos:broader* direction.

2) $c_1 \xrightarrow{narrow,x} c_2$ denotes a path with length x between category c_1 and c_2 following only edges in *skos:narrower* or *skos:related* direction, with at least one edge in *skos:narrower* direction.

The goal is to find the best possible mapping of the category graph onto the cluster hierarchy. This goal is achieved in two bottom-up traversals over the cluster hierarchy.

a) First traversal:: In the first traversal all categories of a cluster node are removed that do not have at least one path to one of the categories of the child nodes. For every non leave node S with child nodes C_1 and C_2 we collect the following set of categories:

$$C = \left\{ c : c \in cat(S) \wedge c \xrightarrow{narrow,x} c_{child} \in \left(\bigcup_{G \in children(S)} cat(G) \right) \right\}$$

The set C only contains categories assigned to S that are connected to at least one category of the child nodes of S. After several tests we set the maximum path length to 5. In most cases if no path has been found between two categories using a maximum search distance of 5 even a higher search distance won't find a path. Furthermore the longer the distance between the parent category and the child category the more unlikely it is that both categories are good labels for their nodes. This is also reflected by the new score of the categories in C:

$$newscore_S(c) = score_{(S,\Phi)}(c) \cdot \left(1.0 - \frac{1}{6|children(S)|} \cdot \sum_{G \in children(S)} minDist(c, G) \right)$$

$$minDist(c, G) = \begin{cases} \min\{x : c \xrightarrow{narrow,x} c_{child} \in cat(G)\}, \\ \{c \xrightarrow{narrow,x} c_{child} \in cat(G) \wedge x < 6\} \neq \emptyset \\ 6 \end{cases}$$

The function $minDist$ returns the length of the shortest path between c and any category in G or 6 if there is no path with a maximum length of five. The right term of the $newscore_S$ is greatest when c has a path of length one to at least one category of each child node. C is assigned to S, so that $cat(S) = C$. After the first traversal only these categories remain that are generalizations of at least one of their child node categories.

b) Second traversal:: The second traversal is similar to the first but this time only leaf nodes are considered. All categories of a leaf node are removed that do not have at least one path (in broader direction) to one of the categories of the parent node. If the direct parent node has no categories left, the next parent is considered until one with categories is found or the root node is reached. In the latter case no categories will be removed from the leaf node. The new score of the remaining leaf node categories is computed similar to the first traversal.

c) Third traversal:: After the first two traversals only categories remain that fit into the hierarchy of the cluster. As a result there are cluster nodes that do not have any category assigned. The goal of the third traversal is to close these gaps if possible. For each cluster node S with $cat(S) = \emptyset$ we distinguish between two types of gaps:

1) The first type is present if a direct or indirect parent of S and at least one of the children has categories assigned: $\exists P, G : G \in children(S) \wedge cat(G) \neq \emptyset \wedge P \in parents(S) \wedge cat(P) \neq \emptyset$. To close the gap we first collect all categories that are ancestors of the child node categories and have a path to at least one of the categories of the parent node.

$$C = \{c : c_{parent} \xrightarrow{narrow,x} c \wedge \exists c_{child} : c_{child} \xrightarrow{broader,y} c\}$$
$$c_{parent} \in cat(P)$$
$$c_{child} \in \bigcup_{G \in children(S)} cat(G)$$

The score of each category in C is computed using the $newscore$ function from the first traversal. Since the initial score of the category denoted by $score_{(S,\Phi)}$ is not defined it is replaced with the average score of the category of each child node with the shortest path to c (0 if no such path exists). This way categories that are the common ancestor of most of the children are favored over these that only cover some children. If two categories cover the same number of children, the one that has the smallest average distance to its child categories or that is connected to the more important child categories gets the higher score.

2) S belongs to the second type if it has no direct or indirect parent with categories assigned, but at least one child node with categories: $\forall P \in parents(S) : cat(P) = \emptyset \wedge \exists G \in children(S) : cat(G) \neq \emptyset$. In this case we collect all ancestors and use the same scoring method as before.

4) Selecting the label:: At this point each cluster node S has a set of categories $cat(S)$. The label of the category with the highest score given by $newscore_S$ is used as the label of S, together with the label of the highest scored DBPResource in R. If S has no categories assigned, only the highest scored DBPResource from R is used. In the case that R is empty too, no label is assigned.

C. Combining both algorithms

A weakness of the first algorithm is that it can find the correct label of a cluster node S only if the label is contained in the documents of S. However even if that is not the case the algorithm will likely find terms that describe the topic of the correct label. Another weakness is that the hierarchical relation between the terms of different cluster nodes cannot be guaranteed. Sometimes the algorithm finds the same label for the parent cluster and a child cluster. Unfortunately it can't decide for which node the term is the better label. Both

Cluster	Documents	Leaf Nodes	MaxDepth
Business	799	13	6
Computers	205	7	6
Health	510	13	6
Recreation	307	16	7
Regional	113	19	7
Science 1	101	9	5
Science 2	760	21	8
Society	30	19	7
Sports 1	182	13	8
Sports 2	730	24	8

TABLE I: Characteristics of the ODP clusters used as test data. It shows their root category, number of documents, number of leaf nodes and their maximal depth.

issues are addressed by the second algorithm. Since it uses an external data source, namely DBpedia, it can find labels that were not in the terms assigned to the cluster node. Due to the category graph the hierarchical relationship between the labels of different clusters can be guaranteed to a certain extend. We propose a third algorithm for hierarchical document clusters that is a combination of our first two algorithms. First we use our optimized DScore algorithm to find the best label candidates for each cluster node S. We than apply the second algorithm that uses the label candidates as the terms of the cluster node.

IV. Experimental Results

In this section, we describe our experiments and the results we gained from them. Our test data collection is proposed in the Section IV-A. In the Section IV-B we describe our evaluation methods and the Section IV-C presents our results.

A. Data

To test our algorithms we have randomly sampled 10 binary hierarchical document clusters from the Open Directory Project. This clusters have "ground truth" labels assigned by human editors. We assured that our test clusters and our training clusters were completely disjoint. Our selected clusters contained in average 373 documents distributed over 15 leaf nodes with an average maximum depth of 7 from under 9 root categories. Table I shows the exact characterization of our test clusters.

B. Evaluation Method

We consider a label L generated by the algorithm as correct if L or a synonym of L exactly or partly matches the correct ODP label CL. Therefore a synonym list was obtained from WordNet [5].

L is an *exact match* of the correct label CL if L or any synonym of L is equal to CL. For example, "tennis tournament" and "tennis championship" are exact matches for the ODP label "tennis tournament".

L is a *partial match* of CL if L or any synonym of L share at least one word with CL that is not a stopword. For example, "tournament" and "championship" are partial matches for the ODP label "tennis tournament".

Cluster	Exact matches	Partial matches	Misses	Hit Rate (%)
Business	3	0	25	10.71
Computers	2	2	9	30.77
Health	0	1	24	4.00
Recreation	2	0	31	6.06
Regional	4	3	34	17.07
Science 1	2	0	15	11.76
Science 2	2	3	37	11.90
Society	1	4	33	13.16
Sports 1	2	3	24	17.24
Sports 2	6	0	45	11.76
Total	24	16	277	12.62

TABLE II: Results of the original DScore. Exact Matches and partial matches both count as correct label.

Cluster	Exact matches	Partial matches	Misses	Hit Rate (%)
Business	6	2	20	28.57
Computers	8	0	5	61.54
Health	2	1	22	12.00
Recreation	3	2	28	15.15
Regional	8	3	30	26.83
Science 1	2	0	15	11.76
Science 2	10	1	31	26.19
Society	9	3	26	31.58
Sports 1	4	3	22	24.14
Sports 2	10	0	41	19.61
Total	62	15	240	24.29

TABLE III: Results of the optimized DScore. Exact Matches and partial matches both count as correct label.

C. Experimental Results

We evaluated our model on the ground truth ODP data. Therefore we extracted the raw text of the linked websites using boilerpipe [10]. The goal of our experiment was to evaluate our optimized DScore algorithm (Section III-A) and our combined algorithm (Section III-C) in comparison to the original DScore algorithm [23].

Table II shows the results of the original DScore algorithm as described in the paper. We have an average hit ratio, considering exact and partial matches of 12.62%. When comparing this results with the results of our optimized version of the DScore algorithm (Table III), we see major improvements. Our hit ratio nearly doubled. The original DScore algorithm was not constructed for binary clusters. All adaptions we made, helped the algorithm to work with smaller clusters.

The best results had our combined algorithm (Table IV) with an average hit ratio of 30.57%.

V. Future Work

To improve the algorithm we plan to enlarge and enhance the category graph with other ontology databases like YAGO [7] and SUMO [19]. Further work could go into the research of a more sophisticated model, for mapping the category graph onto the hierarchical cluster, that allows different disconnected hierarchies to exist concurrently.

Cluster	Exact matches	Partial matches	Misses	Hit Rate (%)
Business	3	1	22	15.38
Computers	4	0	9	30.77
Health	6	2	17	32.00
Recreation	6	2	25	24.24
Regional	10	2	28	30.00
Science 1	7	1	9	47.06
Science 2	20	4	18	57.14
Society	8	0	30	21.05
Sports 1	7	3	19	34.48
Sports 2	9	1	41	19.61
Total	80	16	218	30.57

TABLE IV: Results of the combined Algorithm DScore. Exact Matches and partial matches both count as correct label.

VI. Conclution

In this paper we presented one algorithm for labeling hierarchical documents clusters using statistical methods and a second algorithm for labeling hierarchical term clusters using by DBpedia as an Ontology. We made the second algorithm applicable for hierarchical document clusters by combining it with the first algorithm. Both were tested on manually labeled binary document clusters obtained from the Open Directory Project. We showed that our first algorithm performs significantly better on binary cluster hierarchies than the original DScore algorithm on which it is based. The results have been further improved by our combined algorithm. The combined algorithm has three major advantages. First it is able to find labels that did not occur as terms in the underlying documents. Second, since we make use of an ontology to find labels that best match the hierarchy given by the cluster, we can guarantee to a certain extend that the resulting labels have a correct hierarchical relationship to each other. The usage of the ontology also enables us to discard labels that do not fit into the hierarchy.

References

[1] D. Carmel, H. Roitman, and N. Zwerdling. Enhancing cluster labeling using wikipedia. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 139–146. ACM, 2009.

[2] S.-L. Chuang and L.-F. Chien. A practical web-based approach to generating topic hierarchy for text segments. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 127–136. ACM, 2004.

[3] K. Coursey, R. Mihalcea, and W. Moen. Using encyclopedic knowledge for automatic topic identification. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 210–218. Association for Computational Linguistics, 2009.

[4] Open Directory Project. http://www.dmoz.org/, 1998. [Online; accessed 14-Februar-2014].

[5] C. Fellbaum. *WordNet*. Wiley Online Library, 1999.

[6] E. Glover, D. M. Pennock, S. Lawrence, and R. Krovetz. Inferring hierarchical descriptions. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 507–514. ACM, 2002.

[7] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

[8] I. Hulpus, C. Hayes, M. Karnstedt, and D. Greene. Unsupervised graph-based topic labelling using dbpedia. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 465–474. ACM, 2013.

[9] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[10] C. Kohlschtter. boilerpipe. https://code.google.com/p/boilerpipe/, 2009. [Online; accessed 07-Februar-2014].

[11] R. Krovetz. Viewing morphology as an inference process. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 191–202. ACM, 1993.

[12] J. H. Lau, K. Grieser, D. Newman, and T. Baldwin. Automatic labelling of topic models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1536–1545. Association for Computational Linguistics, 2011.

[13] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, et al. Dbpedia-a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2013.

[14] D. Magatti, S. Calegari, D. Ciucci, and F. Stella. Automatic labeling of topics. In *Intelligent Systems Design and Applications, 2009. ISDA'09. Ninth International Conference on*, pages 1227–1232. IEEE, 2009.

[15] Q. Mei, X. Shen, and C. Zhai. Automatic labeling of multinomial topic models. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 490–499. ACM, 2007.

[16] P. N. Mendes, M. Jakob, and C. Bizer. Dbpedia for nlp: A multilingual cross-domain knowledge base. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, May 2012.

[17] A. Miles, B. Matthews, M. Wilson, and D. Brickley. Skos core: simple knowledge organisation for the web. In *International Conference on Dublin Core and Metadata Applications*, pages pp–3, 2005.

[18] M. Muhr, R. Kern, and M. Granitzer. Analysis of structural relationships for hierarchical cluster labeling. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 178–185. ACM, 2010.

[19] I. Niles and A. Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9. ACM, 2001.

[20] T. Nomoto. Wikilabel: an encyclopedic approach to labeling documents en masse. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2341–2344. ACM, 2011.

[21] P. Schönhofen. Identifying document topics using the wikipedia category network. *Web Intelligence and Agent Systems*, 7(2):195–207, 2009.

[22] Z. S. Syed, T. Finin, and A. Joshi. Wikipedia as an

ontology for describing documents. In *ICWSM*, 2008.

[23] P. Treeratpituk and J. Callan. Automatically labeling hierarchical clusters. In *Proceedings of the 2006 international conference on Digital government research*, pages 167–176. Digital Government Society of North America, 2006.

[24] P. Treeratpituk and J. Callan. An experimental study on automatically labeling hierarchical clusters using statistical features. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 707–708. ACM, 2006.